Università degli Studi di Ferrara

DOTTORATO DI RICERCA IN
SCIENZE DELL'INGEGNERIA

CICLO XXIV

COORDINATORE Prof. Stefano Trillo

LOGIC AND CONSTRAINT PROGRAMMING FOR COMPUTATIONAL SUSTAINABILITY

Settore Scientifico Disciplinare ING-INF/05

Dottorando
Dott. Cattafi Massimiliano

Tutore
Prof. Gavanelli Marco

Cotutore
Prof.ssa Lamma Evelina

Anni 2009/2011

Abstract

Computational Sustainability is an interdisciplinary field that aims to develop computational and mathematical models and methods for decision making concerning the management and allocation of resources in order to help solve environmental problems.

This thesis deals with a broad spectrum of such problems (energy efficiency, water management, limiting greenhouse gas emissions and fuel consumption) giving a contribution towards their solution by means of Logic Programming (LP) and Constraint Programming (CP), declarative paradigms from Artificial Intelligence of proven solidity.

The problems described in this thesis were proposed by experts of the respective domains and tested on the real data instances they provided. The results are encouraging and show the aptness of the chosen methodologies and approaches.

The overall aim of this work is twofold: both to address real world problems in order to achieve practical results and to get, from the application of LP and CP technologies to complex scenarios, feedback and directions useful for their improvement.

Acknowledgements

I would like to thank my advisers Marco Gavanelli and Evelina Lamma for their constant support and precious teachings, not only during my Ph.D., but throughout my University studies.

I am grateful to all my co-authors for the fruitful shared work and exchange of ideas. I would like to mention Marco Alberti and Fabrizio Riguzzi for their guidance in my first projects and publications, Paola Mello, Michela Milano and Maddalena Nonato for their help in completing and extending my range of interests, Paolo Cagnoli, Marco Franchini, Stefano Alvisi and Federico Malucelli for their challenging problem proposals.

I am glad I had the opportunity to spend one semester in close, very enjoyable, cooperation with Rosa Herrero, from the Universitat Autònoma de Barcelona, during her research visit at our department.

I also had the pleasure to spend one semester in the stimulating environment of the Cork Constraint Computation Centre and I would like to express my gratitude to Helmut Simonis for mentoring me and to all the other people of the lab who made me feel very welcome and contributed to make my experience worthwhile.

I would like to dedicate this thesis to my mom, whose effort to inspire me towards curiosity, creative thought and autonomy proved fundamental also during the development of this work.

Contents

List of Figures

List of Tables

LIST OF TABLES

1

Introduction

Computational Sustainability is an interdisciplinary field whose aim is to apply techniques from computer science, information science, operations research, applied mathematics, and statistics for balancing environmental, economic, and societal needs for sustainable development (i.e., using the words of the 1983 United Nations Brundtland Commission, development that meets the needs of the present generation without compromising the ability of future generations to meet their own needs) (2).

The focus is developing computational and mathematical models and methods for decision making concerning the management and allocation of resources in order to help solve some of the most challenging problems related to sustainability.

In this thesis we present various problems related to the domain of Computational Sustainability, on a broad spectrum which encompasses several of its critical aspects: energy efficiency, water management, limiting greenhouse gas emissions and fuel consumption.

We contribute towards their solution addressing them by means of Logic Programming (LP) and Constraint Programming (CP), which are paradigms from Artificial Intelligence of proven theoretical and practical solidity.

LP and CP offer many advantages under the point of view of declarativeness and flexible expressivity, which are crucial in an area such as Computational Sustainability, where the scenario is often dynamic and faceted and where problem specifics have to be gathered from different domains of expertise often difficult to formalize in a straightforward way. The rapid prototyping features of LP and CP allow one to deliver working models in short time, test them on scaled-down instances and get quick feedback from the experts of the domain in order to

(incrementally) refine the model and improve efficiency in a second stage on top of the existent implementation.

This pattern strongly inspired the approach to the problems described in this thesis, which were proposed by experts of the respective domains (such as environmental agencies, hydraulic engineers, human resource departments) and tested on the real data instances they provided. The results are encouraging and show the aptness of the chosen methodologies and approaches.

The overall aim of this work is twofold: both to address real world problems in order to achieve practical results and to get from the application of LP and CP technologies to complex scenarios feedback and directions useful for their improvement.

Publications

Papers containing part of the work described in this thesis were presented in various venues:

Conferences and book chapters:

M. Cattafi, E. Lamma, F. Riguzzi, and S. Storari. Incremental declarative process mining. In N. T. Nguyen and E. Szczerbicki, editors, *Smart Information and Knowledge Management: Advances, Challenges, and Critical Issues, volume 260 of Studies in Computational Intelligence*, pages 103–127. Springer, Heidelberg, Germany, 2009.(3)

M. Alberti, M. Cattafi, F. Chesani, M. Gavanelli, E. Lamma, M. Montali, P. Mello, and P. Torroni. Integrating abductive logic programming and description logics in a dynamic contracting architecture. *Web Services, IEEE International Conference on* 254–261, 2009.(4)

Journals:

M. Alberti, M. Cattafi, F. Chesani, M. Gavanelli, E. Lamma, P. Mello, M. Montali, and P. Torroni. A Computational Logic Application Framework for Service Discovery and Contracting. *International Journal of Web Services Research*, 8(3):1-25, July-September 2011 (5).

M. Cattafi, M. Gavanelli, M. Milano and P. Cagnoli. Sustainable biomass power plant location in the Italian Emilia-Romagna region. *ACM Transactions on Intelligent Systems and Technology*, 2(4), July 2011 (6).

M. Cattafi, M. Gavanelli, M. Nonato, S. Alvisi, and M. Franchini. Optimal placement of valves in a water distribution network with CLP(FD). *Theory and Practice of Logic Programming*, 11(4-5):731-747, 2011. *Best paper award at the 27th International Conference on Logic Programming (ICLP 2011)* (7).

Synopsis

This thesis is organized as follows. We first provide some preliminaries about Logic Programming and Constraint Programming in Chapter 2. In Chapter 3 we present a Computational Logic framework whose aim is to ease the complexity of Service Oriented Architectures, a paradigm which can give a considerable contribution in terms of reducing energy consumption and greenhouse gas emissions of data centres. Energy and greenhouse emissions are also the subject of Chapter 4, this time on the side of energy production from biomass. Some kinds of biomass are a promising source of renewable and low carbon footprint energy but careful assessment of their supply context is necessary in order not to deplete the potential advantages. Chapter 5 is dedicated to another fundamental aspect of sustainability: water and its careful distribution by design of (topologically) robust networks. In Chapter 6 focus is on societal needs and their balance with environmental and economic aspects, the context being home health care and the challenges it poses. Finally, in Chapter 7 we conclude and suggest some future work.

1. INTRODUCTION

2

Preliminaries

2.1 Logic Programming

A *first order alphabet* Σ is a set of predicate symbols and function symbols (or functors) together with their arity. A *term* is either a variable or a functor applied to a tuple of terms of length equal to the arity of the functor. If the functor has arity 0 it is called a *constant*. An *atom* is a predicate symbol applied to a tuple of terms of length equal to the arity of the predicate. A *literal* is either an atom a or its negation $\neg a$. In the latter case it is called a *negative literal*. In logic programming, predicate and function symbols are indicated with alphanumeric strings starting with a lowercase character while variables are indicated with alphanumeric strings starting with an uppercase character.

A *clause* is a formula C of the form

$$h_1 \vee \ldots \vee h_n \leftarrow b_1, \ldots, b_m$$

where $h_1, \ldots, h_n$ are atoms and $b_1, \ldots, b_m$ are literals, whose separation by means of commas represents a conjunction. A clause can be seen as a set of literals, e.g., C can be seen as

$$\{h_1, \ldots, h_n, \neg b_1, \ldots, \neg b_m\}.$$

In this representation, the disjunctions among the elements of the set are left implicit.

Which form of a clause is used in the following will be clear from the context. $h_1 \vee \ldots \vee h_n$ is called the *head* of the clause and $b_1, \ldots, b_m$ is called the *body*. We will use $head(C)$ to indicate either $h_1 \vee \ldots \vee h_n$ or $\{h_1, \ldots, h_n\}$, and $body(C)$ to indicate either $b_1, \ldots, b_m$ or $\{b_1, \ldots, b_m\}$, the exact meaning will be clear from the context. When $m = 0$ and $n = 1$, C is

called a *fact*. When $n = 1$, C is called a *program clause*. When $n = 0$, C is called a *goal*. The conjunction of a set of literals is called a *query*. A clause is *range restricted* if all the variables that appear in the head appear as well in the body.

A *theory* P is a set of clauses. A *normal logic program* P is a set of program clauses.

A term, atom, literal, goal, query or clause is *ground* if it does not contain variables. A *substitution* θ is an assignment of variables to terms: $\theta = \{V_1/t_1, \ldots, V_n/t_n\}$. The *application of a substitution to a term, atom, literal, goal, query or clause* C, indicated with $C\theta$, is the replacement of the variables appearing in C and in θ with the terms specified in θ.

The *Herbrand universe* $H_U(P)$ is the set of all the terms that can be built with function symbols appearing in P. The *Herbrand base* $H_B(P)$ of a theory P is the set of all the ground atoms that can be built with predicate and function symbols appearing in P. A *grounding* of a clause C is obtained by replacing the variables of C with terms from $H_U(P)$. The grounding $g(P)$ of a theory P is the program obtained by replacing each clause with the set of all of its groundings. A *Herbrand interpretation* is a set of ground atoms, i.e. a subset of $H_B(P)$. In the following, we will omit the word 'Herbrand'.

Let us now define the truth of a formula in an interpretation. Let I be an interpretation and ϕ a formula, ϕ is true in I, written $I \models \phi$ if

- $a \in I$, if ϕ is a ground atom a;
- $a \notin I$, if ϕ is a ground negative literal $\neg a$;
- $I \models a$ and $I \models b$, if ϕ is a conjunction $a \wedge b$;
- $I \models a$ or $I \models b$, if ϕ is a disjunction $a \vee b$;
- $I \models \psi\theta$ if $\phi = \forall \mathbf{X}\psi$ for all θ that assign a value to all the variables of $\mathbf{X}$;
- $I \models \psi\theta$ if $\phi = \exists \mathbf{X}\psi$ for a θ that assigns a value to all the variables of $\mathbf{X}$.

A clause C of the form

$$h_1 \vee \ldots \vee h_n \leftarrow b_1, \ldots, b_m$$

is a shorthand for the formula

$$\forall \mathbf{X} h_1 \vee \ldots \vee h_n \leftarrow b_1, \ldots, b_m$$

where $\mathbf{X}$ is a vector of all the variables appearing in C. Therefore, C is true in an interpretation I iff, for all the substitutions θ grounding C, if $I \models body(C)\theta$ then $I \models head(C)\theta$, i.e., if

$(I \models body(C)\theta) \rightarrow (head(C)\theta \cap I \neq \emptyset)$. Otherwise, it is false. In particular, a program rule is true in an interpretation I iff, for all the substitutions θ grounding C, $(I \models body(C)\theta) \rightarrow h \in I$.

A theory P is true in an interpretation I iff all of its clauses are true in I and we write

$$I \models P.$$

If P is true in an interpretation I we say that I is a *model* of P. It is sufficient for a single clause of a theory P to be false in an interpretation I for P to be false in I.

For normal logic programs, we are interested in deciding whether a query Q is a logical consequence of a theory P, expressed as

$$P \models Q.$$

This means that Q must be true in a model $M(P)$ of P that is assigned to P as its meaning by one of the semantics that have been proposed for normal logic programs (e.g. (8, 9, 10)).

For theories, we are interested in deciding whether a given theory or a given clause is true in an interpretation I. This can be achieved with the following procedure (11). The truth of a range restricted clause C on a finite interpretation I can be tested by asking the goal ?-$body(C), \neg head(C)$ against a database containing the atoms of I as facts. By $\neg head(C)$ we mean $\neg h_1, \ldots, \neg h_m$. If the query fails, C is true in I, otherwise C is false in I.

In some cases, we are not given an interpretation I completely but we are given a set of atoms J and a normal program B as a compact way of indicating the interpretation $M(B \cup J)$. In this case, if B is composed only of range restricted rules, we can test the truth of a clause C on $M(B \cup J)$ by running the query ?-$body(C), \neg head(C)$ against a Prolog database containing the atoms of J as facts together with the rules of B. If the query fails, C is true in $M(B \cup J)$, otherwise C is false in $M(B \cup J)$.

2.2 Inductive Logic Programming

Inductive Logic Programming (ILP) (12) is a research field at the intersection of Machine Learning and Logic Programming. It is concerned with the development of learning algorithms that adopt logic for representing input data and induced models. Recently, many techniques have been proposed in the field and they were successfully applied to a variety of domains. Logic proved to be a powerful tool for representing the complexity that is typical of the real

world. In particular, logic can represent in a compact way domains in which the entities of interest are composed of subparts connected by a network of relationships. Traditional Machine Learning is often not effective in these cases because it requires input data in the flat representation of a single table.

The problem that is faced by ILP can be expressed as follows:

Given:

- a space of possible theories $\mathcal{H}$;
- a set E^+ of positive example;
- a set E^- of negative examples;
- a background theory B.

Find a theory $H \in \mathcal{H}$ such that;

- all the positive examples are covered by H
- no negative example is covered by H

If a theory does not cover an example we say that it rules the example out so the last condition can be expressed by saying the "all the negative examples are ruled out by H".

The general form of the problem can be instantiated in different ways by choosing appropriate forms for the theories in input and output, for the examples and for the covering relation.

In the *learning from entailment* setting, the theories are normal logic programs, the examples are (most often) ground facts and the coverage relation is entailment, i.e., a theory H covers an example e iff

$$H \models e.$$

In the learning from interpretations setting, the theories are composed of clauses, the examples are interpretations and the coverage relation is truth in an interpretation, i.e., a theory H covers an example interpretation I iff

$$I \models H.$$

Similarly, we say that a clause C covers an example I iff $I \models C$.

We concentrate on learning from interpretation so we report here the detailed definition:

Given:

- a space of possible theories $\mathcal{H}$;
- a set E^+ of positive interpretations;
- a set E^- of negative interpretations;
- a background normal logic program B.

Find a theory $H \in \mathcal{H}$ such that;

- for all $P \in E^+$, H is true in the interpretation $M(B \cup P)$;
- for all $N \in E^-$, H is false in the interpretation $M(B \cup N)$.

The background knowledge B is used to encode each interpretation parsimoniously, by storing separately the rules that are not specific to a single interpretation but are true for every interpretation.

The algorithm ICL (13) solves the above problem. It performs a covering loop (function ICL in Figure 2.1) in which negative interpretations are progressively ruled out and removed from the set E^-. At each iteration of the loop a new clause is added to the theory. Each clause rules out some negative interpretations. The loop ends when E^- is empty or when no clause is found.

The clause to be added in every iteration of the covering loop is returned by the procedure FindBestClause (Figure 2.1). We look for clauses that cover as many positive interpretations as possible and rule out as many negative interpretations as possible. Starting from the clause $false \leftarrow true$, that rules out all the negative interpretations but also all the positive ones, the search gradually refines it. The algorithm is based on a beam search: the beam is a (finite) pool of candidate partial clauses to refine. At each iteration the best one, according to a certain heuristic, is selected and the worst one, according to the same criterion, is discarded if the beam is full. We use notation $p(\ominus|\overline{C})$ for the probability that an example interpretation is classified as negative given that it is ruled out by the clause C. This probability can be a suitable heuristic and it is computed as the number of ruled out negative interpretations over the total number of ruled out interpretations (positive and negative). The refinements of a clause are obtained by generalization. A clause C is *more general* than a clause D if the set of interpretations covered by C is a superset of those covered by D. This is true if $D \models C$. However, using logical implication as a generality relation is impractical because of its high computational cost. Therefore, the syntactic relation of *θ-subsumption* is used in place of implication: D

```
function ICL(E+, E−, B)
initialize H := ∅
do
    C := FindBestClause(E+, E−, B)
    if best clause C ≠ ∅ then
        add C to H
        remove from E− all interpretations that are false for C
while C ≠ ∅ and E− is not empty
return H

function FindBestClause(E+, E−, B)
initialize Beam := {false ← true}
initialize BestClause := ∅
while Beam is not empty do
    initialize NewBeam := ∅
    for each clause C in Beam do
        for each refinement Ref ∈ δ(C) do
            if Ref is better than BestClause then BestClause := Ref
            if Ref is not to be pruned then
                add Ref to NewBeam
                if size of NewBeam > MaxBeamSize then
                    remove worst clause from NewBeam
    Beam := NewBeam
return BestClause
```

Figure 2.1: ICL learning algorithm

θ-subsumes C (written $D \geq C$) if there exist a substitution θ such that $D\theta \subseteq C$. If $D \geq C$ then $D \models C$ and thus C is more general than D. The opposite, however, is not true, so θ--subsumption is only an approximation of the generality relation. For example, let us consider the following clauses:

$C_1 = accept(X) \leftarrow true$

$C_2 = accept(X) \vee refusal(X) \leftarrow true$

$C_3 = accept(X) \leftarrow invitation(X)$

$C_4 = accept(alice) \leftarrow invitation(alice)$

Then $C_1 \geq C_2$, $C_1 \geq C_3$ but $C_2 \not\geq C_3$, $C_3 \not\geq C_2$ so C_2 and C_3 are more general than C_1, while C_2 and C_3 are not comparable. Moreover $C_1 \geq C_4$, $C_3 \geq C_4$ but $C_2 \not\geq C_4$ and $C_4 \not\geq C_2$ so C_4 is more general than C_1 and C_3 while C_2 and C_4 are not comparable.

From the definition of θ-subsumption, it is clear that a clause can be refined (i.e. generalized) by applying one of the following two operations on a clause

- adding a literal to the (head or body of the) clause
- applying a substitution to the clause.

FindBestClause computes the refinements of a clause by applying one of the above two operations. Let us call $\delta(C)$ the set of refinements so computed for a clause C. The clauses are thus gradually generalized until a clause is found that covers all (or most of) the positive interpretations while still ruling out some negative interpretations.

The literals that can possibly be added to a clause are specified in the *language bias*, a collection of statements in an ad hoc language that prescribe which refinements have to be considered. Two languages are possible for ICL: $\mathcal{D}$LAB and rmode (see (14) for details). Given a language bias which prescribes that the body literals must be chosen among $\{invitation(X)\}$ and that the head disjuncts must be chosen among $\{accept(X), refusal(X)\}$, an example of refinements sequence performed by FindBestClause is the following:

$false \leftarrow true$

$accept(X) \leftarrow true$

$accept(X) \leftarrow invitation(X)$

$accept(X) \vee refusal(X) \leftarrow invitation(X)$

The refinements of clauses in the beam can also be pruned: a refinement is pruned if it is not statistical significant and if it cannot produce a value of the heuristic function larger than that of the best clause. As regards the first type of pruning, a statistical test is used, while as regards

the second type of pruning, the best refinement that can be obtained is a clause that covers all the positive examples and rules out the same negative examples as the original clause.

When a new clause is returned by FindBestClause, it is added to the current theory. The negative interpretations that are ruled out by the clause are ruled out as well by the updated theory, so they can be removed from E^-.

2.2.1 Incremental Inductive Logic Programming

The learning framework presented in Section 2.2 assumes that all the examples are provided to the learner at the same time and that no previous model exists for the concepts to be learned. In some cases, however, the examples are not all known at the same time and an initial theory may be available. When a new example is obtained, one approach consists of adding the example to the previous training set and learning a new theory from scratch. This approach may turn out to be too inefficient, especially if the amount of previous examples is very high. An alternative approach consists in revising the existing theory to take into account the new example, in order to exploit as much as possible the computation already done. The latter approach is called Theory Revision and can be described by the following definition:

Given:

- a space of possible theories $\mathcal{H}$;
- a background theory B
- a set E^+ of previous positive example;
- a set E^- of previous negative examples;
- a theory H that is consistent with E^+ and E^-
- a new example e.

Find a theory $H' \in \mathcal{H}$ such that;

- H' is obtained by applying a number of transformations to H
- H' covers e if e is a positive example, or
- H' does not cover e if e is a negative example.

Theory Revision has been extensively studied in the learning from entailment setting of Inductive Logic Programming. In this case, examples are ground facts, the background theory is a normal logic program and the coverage relation is logical entailment. Among the systems that have been proposed for solving such a problem are: RUTH (15), FORTE (16) and Inthelex (17)

All these systems perform the following operations:

- given an uncovered positive example e, they generalize the theory T so that it covers it
- given a covered negative example e, they specialize the theory T so that it does not cover it

As an example of an ILP Theory Revision system, let us consider the algorithm of Inthelex that is shown in Figure 2.2. Function Generalize is used to revise the theory when the new example is positive. Each clause of the theory is considered in turn and is generalized. The resulting theory is tested to see whether it covers the positive example. Moreover, it is tested on all the previous negative examples to ensure that the clause is not generalized too much. As soon as a good refinement is found it is returned by the function.

Function Specialize is used to revise the theory when the new example is negative. Each clause of the theory involved in the derivation of the negative example is considered in turn and is specialized. The resulting theory is tested to see whether it rules out the negative example. Moreover, it is tested on all the previous positive examples to ensure that the clause is not specialized too much. As soon as a good refinement is found it is returned by the function.

While various systems exist for Theory Revision in the learning from entailment setting, to the best of our knowledge no algorithm has been proposed for Theory Revision in the learning from interpretation setting.

2.3 Abductive Logic Programming and the $\mathcal{S}$CIFF system

Abductive Logic Programming (18) is useful in a number of situations involving reasoning with incomplete knowledge and dynamic occurrence of events. $\mathcal{S}$CIFF (19) is an extension of the IFF abductive language and proof-procedure (20). Its main motivations originate from the domain of agent interaction where the definition of agent communication languages and protocols, their semantic characterization and their verification are key issues. Its aim is to specify in a declarative but intensional way what are "admissible interactions" among agents, and to

function Generalize(E^-, e, B, H)
repeat
    pick a clause C from H
    obtain a set of generalizations $\delta(C)$
    for each clause $C' \in \delta(C)$
        let $H' := H \setminus \{C\} \cup \{C'\}$
        test H' over e and over all the examples in E^-
        if H' cover e and does not cover any negative example then
            return H'
until all the clauses of H have been considered
// no generalization found
add a new clause to H that covers e and is consistent with E^-
let H' be the new theory
return H'

function Specialize(E^+, e, B, H)
repeat
    pick a clause C used in the derivation of e in H
    obtain a set of specializations $\rho(C)$
    for each clause $C' \in \rho(C)$
        let $H' := H \setminus \{C\} \cup \{C'\}$
        test H' over e and over all the examples in E^+
        if H' does not cover e and covers all positive examples then
            return H'
until all the clauses of H used in the derivation of e have been considered
// no specialization found
add e to H as an exception
let H' be the new theory
return H'

Figure 2.2: Inthelex Theory Revision algorithm

provide a computational tool to decide whether a given interaction is admissible (*verification of compliance*).

An Abductive Logic Program (ALP) is defined as the triplet $\langle KB_S, \mathcal{A}, IC \rangle$, where KB_S is a logic program in which the clauses can contain special atoms, that belong to the set $\mathcal{A}$ and are called *abducibles*. Such atoms are not defined by means of clauses in the KB_S, and, as such, they cannot be proved: their truth value can be only hypothesized. In order to avoid unconstrained hypotheses, a set of *integrity constraints* (IC) must always be satisfied. Integrity constraints, in SCIFF are in the form of implications, and can relate abducible literals, defined literals, as well as constraints with Constraint Logic Programming semantics (see Section 2.4.6).

Given an abductive logic program $T = \langle P, Ab, IC \rangle$ and a formula G, the goal of abduction is to find a (possibly minimal) set of ground atoms Δ (*abductive explanation*) in predicates in Ab which, together with P, "entails" G, i.e., $P \cup \Delta \models G$, and such that $P \cup \Delta$ "satisfies" IC, e.g. $P \cup \Delta \models IC$ (see (18) for other possible notions of integrity constraint "satisfaction"). Here, the notion of "entailment" $\models$ depends on the semantics associated with the logic program P (there are many different choices for such semantics (21)).

2.3.1 Syntax of the SCIFF language

The SCIFF language is composed of entities for expressing *events and expectations about events* and *relationships between events and expectations*.

Events are the abstractions used to represent the actual behaviour.

Definition 1 *An* event *is an atom:*

- *with predicate symbol* $\mathbf{H}$*;*
- *whose first argument is a ground term; and*
- *whose second argument is an integer.*

Intuitively, the first argument is meant to represent the description of the happened event, according to application-specific conventions, and the second argument is meant to represent the time at which the event has happened.

Example 1

$$\mathbf{H}(tell(alice, bob, query\_ref(phone\_number), dialog\_id), 10) \tag{2.1}$$

says that alice asked bob his phone_number with a query_ref message, in the context identified by dialog_id, at time 10.

A *negated event* is a negative literal $\mathbf{not}\ \mathbf{H}(\ldots,\ldots)$. We will call *history* a set of happened events, and denote it with the symbol $\mathbf{HAP}$.

Expectations are the abstractions used to represent the desired events from an external viewpoint (cannot be enforced, only expected to be as one would like them to be).

Expectations are of two types:

- *positive*: representing some event that is expected to happen;
- *negative*: representing some event that is expected *not* to happen.

Definition 2 *A* positive expectation *is an atom:*

- *with predicate symbol* $\mathbf{E}$*;*
- *whose first argument is a term; and*
- *whose second argument is a variable or an integer.*

Intuitively, the first argument is meant to represent an event description, and the second argument is meant to tell for what time the event is expected (which should not be confused with the time at which the expectation is generated, which is not modeled by the SCIFF's declarative semantics). Expectations may contain variables, which leaves the expected event not completely specified. Variables in positive expectations are always existentially quantified: if the time argument is a variable, for example, this means that the event is expected to happen at *any* time.

Example 2 *The atom*

$$\mathbf{E}(\mathit{tell}(\mathit{bob}, \mathit{alice}, \mathit{inform}(\mathit{phone\_number}, X), \mathit{dialog\_id}), T_i) \tag{2.2}$$

says that bob is expected to inform alice that the value for the piece of information identified by phone_number is X, in the context identified by dialog_id. Such (inform) event should happen at some time T_i to fulfil the expectation.

A *negated positive expectation* is a positive expectation with the explicit negation operator $\neg$ applied to it. Variables in negated positive expectations are quantified as those in positive expectations.

Definition 3 *A* negative expectation *is an atom:*

- *with predicate symbol* $\mathbf{EN}$*;*
- *whose first argument is a term; and*
- *whose second argument is a variable or an integer.*

Intuitively, the first argument is meant to represent an event description, and the second argument is meant to tell in which time points the event is expected not to happen. As well as positive expectations, negative expectations may contain variables which are typically universally quantified: for example, if the time argument is a variable, then the event is expected not to happen at *all* times.

Example 3 *The atom*

$$\mathbf{EN}(\mathit{tell}(\mathit{bob}, \mathit{alice}, \mathit{refuse}(\mathit{phone\_number}), \mathit{dialog\_id}), T_r) \tag{2.3}$$

means that bob is expected not *to refuse to alice his phone_number, in the context identified by dialog_id, at* any *time.*

A *negated negative expectation* is a negative expectation with the explicit negation operator $\neg$ applied to it. Variables in negated negative expectations are quantified as those in negative expectations.

Is is worth noticing that

$$\neg\mathbf{E}(\mathit{tell}(\mathit{bob}, \mathit{alice}, \mathit{refuse}(\mathit{phone\_number}), \mathit{dialog\_id}), T_r)$$

is different from

$$\mathbf{EN}(\mathit{tell}(\mathit{bob}, \mathit{alice}, \mathit{refuse}(\mathit{phone\_number}), \mathit{dialog\_id}), T_r)$$

. The intuitive meaning of the former is: no *refuse* is expected by Bob (if he does, we simply did not expect him to), whereas the latter has a different, stronger meaning: it is expected that Bob does not utter *refuse* (by doing so, he would frustrate our expectations).

2.3.1.1 Syntax with explicit quantifiers

Although SCIFF offers a very rich, flexible and transparent (i.e., implicit) *quantification* of variables, which keeps the language neat and simple, and makes it suitable to express interaction protocols in an intuitive way, in some contexts it can be useful to explicit the quantifiers.

ICs can thus be expressed as follows:

$$Body \rightarrow \exists(ConjP_1) \vee \ldots \vee \exists(ConjP_n) \vee \forall\neg(ConjN_1) \vee \ldots \vee \forall\neg(ConjN_m) \qquad (2.4)$$

where $Body$, $ConjP_i$ $i = 1, \ldots, n$ and $ConjN_j$ $j = 1, \ldots, m$ are, as in the conjunctions of literals built over event atoms, over predicates defined in the background or over built-in predicates such as $\leq, \geq, \ldots$. The variables appearing in the body are implicitly universally quantified with scope the entire formula. The quantifiers in the head apply to all the variables appearing in the conjunctions and not appearing in the body.

We will use $Body(C)$ to indicate $Body$ and $Head(C)$ to indicate the formula $\exists(ConjP_1) \vee \ldots \vee \exists(ConjP_n) \vee \forall\neg(ConjN_1) \vee \ldots \vee \forall\neg(ConjN_m)$ and call them respectively the *body* and the *head* of C. We will use $HeadSet(C)$ to indicate the set $\{ConjP_1, \ldots, ConjP_n, ConjN_1, \ldots, ConjN_m\}$.

$Body(C)$, $ConjP_i$ $i = 1, \ldots, n$ and $ConjN_j$ $j = 1, \ldots, m$ will be sometimes interpreted as sets of literals, the intended meaning will be clear from the context. We will call P *conjunction* each $ConjP_i$ for $i = 1, \ldots, n$ and N *conjunction* each $ConjN_j$ for $j = 1, \ldots, m$. We will call P *disjunct* each $\exists(ConjP_i)$ for $i = 1, \ldots, n$ and N *disjunct* each $\forall\neg(ConjN_j)$ for $j = 1, \ldots, m$.

An example of an IC written with explicit quantifiers is:

$$\begin{array}{l} a(bob, T), T < 10 \\ \rightarrow \exists(b(alice, T1), T < T1) \\ \vee \\ \forall\neg(c(mary, T1), T < T1, T1 < T + 10) \end{array} \qquad (2.5)$$

IC (2.5) means: if *bob* has executed action a at a time $T < 10$, then $alice$ must execute action b at a time $T1$ later than T or $mary$ must not execute action c for 9 time units after T. The disjunct $\exists(b(alice, T1), T < T1)$ stands for $\exists T1(b(alice, T1), T < T1)$ and the disjunct $\forall\neg(c(mary, T1), T < T1, T1 < T + 10)$ stands for $\forall T1\neg(c(mary, T1), T < T1, T1 < T + 10)$.

2.4 Constraint Satisfaction Problems

2.4.1 Definition

Many problems in Artificial Intelligence can be considered as instances of Constraint Satisfaction Problems.

Definition 4 *A* Constraint Satisfaction Problem (CSP) *is a triple* $P = \langle X, D, C \rangle$ *where* $X = \{X_1, X_2, \ldots, X_n\}$ *is a set of unknown variables,* $D = \{D_1, D_2, \ldots, D_n\}$ *is a set of domains and* $C = \{c_1, \ldots, c_m\}$ *is a set of constraints. Each* $c_l(X_{i_1}, \ldots, X_{i_k})$ *is a relation, i.e., a subset of the Cartesian product of the involved variables* $D_{i_1} \times \cdots \times D_{i_k}$*. An assignment* $A = \{X_1 \mapsto d_1, \ldots, X_n \mapsto d_n\}$ *(where* $d_1 \in D_1, \ldots, d_n \in D_n$*) is a solution iff it satisfies all the constraints.*

If all the constraints in a CSP are unary or binary (involve at most two variables), we have a *binary CSP*. Binary CSPs are often represented as graphs: each variable is a node and each constraint is an arc that links the involved variables.

Applications include various domains: scheduling, planning, logistics, timetabling, artificial vision, molecular biology, etc (22, 23, 24, 25, 26, 27, 28).

2.4.2 Algorithms

In general, the tasks posed in the constraint satisfaction problem paradigm are computationally intractable (NP-hard). Various types of algorithms have been proposed to solve Constraint Satisfaction Problems; an exhaustive survey is beyond the scope of this thesis. A good introduction is given in (29).

2.4.3 Consistency Techniques

In order to reduce the complexity of looking for a solution of a CSP, domain values that are provably not part of any solutions can be deleted (*domain filtering*) testing *consistency*.

Definition 5 *A unary constraint* $c(X_i)$ *is* Node-Consistent *iff* $\forall v \in D_i,\ v \in c(X_i)$*; i.e., each value* v *belonging to the domain of variable* X_i *is consistent.*

Definition 6 *A binary constraint* $c(X_i, X_j)$ *is* Arc-Consistent *iff* $\forall v \in D_i\ \exists u \in D_j$ *such that* $(v, u) \in c(X_i, X_j)$ *and vice-versa. In other words, for each value* v *belonging to the domain of variable* X_i *there exists a value* u *in the domain of* X_j *that is consistent with* v *and for each value in the domain of* X_j *there exists a supporting value in the domain of* X_i*.*

```
procedure REVISE(X_i, X_j)
  DELETE ← false;
  for each v in D_i do
    if there is no such u in D_j such that (u, v) is consistent,
      then
        delete v from D_i;
        DELETE ← true;
    endif;
  endfor;
  return DELETE;
end REVISE

procedure AC-3
  Q ← C  % Set of all the constraints
  while not Q empty
    select and delete any constraint c(X_k, X_m) from Q;
    if REVISE(X_k, X_m) then
      Q ← Q ∪ {c(X_i, X_k) such that c(X_i, X_k) ∈ C, i ≠ k, i ≠ m}
    endif
  endwhile
end AC-3
```

Figure 2.3: Algorithm AC-3

A Constraint network is Arc-Consistent if all the constraints are Arc-Consistent. Every binary CSP can be converted into an equivalent (i.e., with the same solution set) CSP which is Arc-Consistent. Many algorithms achieving Arc-Consistency have been proposed. All these algorithms are based on the following idea: if a binary constraint $c(X_i, X_j)$ is not arc-consistent, there will be a value v that is not supported. Supposing that v is in the domain of X_i, there is no value in the domain of X_j that is consistent with v. In this case, v can be safely removed from the domain of X_i. When we delete a value, we have a list of things that must be re-considered. In AC-3 (30), that is often used because of its simple implementation, the list contains constraints (or, equivalently, arcs of the constraint graph). AC-3 is reported in Figure 2.3.

Other levels of consistency have been defined: Path-Consistency (31) and k-consistency (32, 33) achieve higher level of consistency, but they are more expensive. In most cases, Arc-Consistency is a good tradeoff, hence it is widely used.

If we deal with non-binary constraints (also called *global* constraints), Arc-Consistency can be extended:

Definition 7 *An n-ary constraint $c(X_1, \ldots, X_n)$ is* Generalized Arc-Consistent (GAC) *iff $\forall v \in D_i$, $\forall j \neq i$, $\exists v_{k_j} \in D_j$ such that $(v_{k_1}, \ldots, v_{k_{i-1}}, v_i, v_{k_{i+1}}, \ldots, v_{k_n}) \in c(X_1, \ldots, X_n)$.*

Even when a non-binary CSP can be converted into a binary one (34), the use of global constraints is often beneficial. The model can be expressed in a more declarative, clear and concise way, and propagation is usually stronger too.

2.4.4 Systematic Search Algorithms

Systematic search algorithms build a search tree and explore it to find a feasible assignment.

2.4.4.1 Backtracking

Backtracking (or Standard Backtracking, or Chronological Backtracking) (35) incrementally attempts to extend a partial solution that specifies consistent values for some of the variables, toward a complete solution, by repeatedly choosing a value for another variable consistent with the values in the current partial solution. We have a labelling phase and a test for consistency phase.

In the labelling phase an unassigned variable is chosen and assigned a value in its domain. In the test phase, the last assignment is checked for consistency with all the other variables that were assigned before. If an inconsistency is detected, the last assignment is undone, and another value is chosen.

2.4.4.2 Forward Checking

Forward Checking (FC) (36) is a commonly used algorithm for solving CSPs, because it is simple and effective. This algorithm interleaves a labelling phase and a propagation phase. In the labelling phase a tentative value v is assigned to a given variable X as in Standard Backtracking. In the propagation phase, unassigned variables connected to X are considered,

and all values inconsistent with v removed from the corresponding domains, so only consistent values remain.

When a domain becomes empty, we have a failure, and the algorithm (chronologically) backtracks.

2.4.5 Constraint Optimization Problems

In some cases, finding a feasible solution is not enough, and a *preference* between solutions must be expressed. This preference is usually given with a function, that maps solutions into a cost (for minimization problems) or to a profit (for maximization problems).

2.4.5.1 Definition

For our needs, we define the Constraint Optimization problem as follows:

Definition 8 *A Constraint Optimization Problem (COP) is a pair $Q = \langle P, g \rangle$ such that P is a CSP and $g : D_1 \times \cdots \times D_n \mapsto S_t$ (where $\langle S_t, \leq \rangle$ is a totally-ordered set) is a merit function that maps each solution tuple into a value. A solution A of P is also a solution of the COP iff $\nexists A'$ solution of P such that $g(A) < g(A')$.*

We have thus a total order induced by the function f on the set of possible CSP solutions. We will consider a maximization problem, being straightforward the extension to minimization problems.

2.4.5.2 Algorithms

The usual algorithm for solving COPs is Branch-and-Bound (37).

Branch and Bound is a general search method; a possible description of Branch-and-Bound is as follows. The whole problem is divided into two or more subproblems, as in a tree-search. When we find a solution, we impose that the solution must not be worse than the current optimal solution. A sketch of the algorithm is presented in Figure 2.4.

The comparison between the current optimum and the value in each node of the search tree can be performed in different ways. In the original formulation, an Upper Bound must be computed in every node of the search tree; if the Upper Bound is worse than the current best, the branch can be pruned off, because it cannot lead to an improvement of the solution.

```
let CSP = ⟨X, D, C⟩, COP = (CSP, g)
let AP = {CSP}                          % Set of Active Problems
CurSol = −∞                             % Current best solution
While AP ≠ ∅
        choose P ∈ AP, let P = ⟨X_p, D_p, C_p⟩
        if P is inconsistent
             AP ← AP \ {P}
        Else if P has only one solution S_p
             % Add the constraint g(X) > f(S_p) to all the active problems
             ∀Q ∈ AP, Q = ⟨X_q, D_q, C_q⟩, C_q ← C_q ∪ {g(X_q) > g(S_p)}
             let CurSol ← S_p
        Else % Branch
             Generate the set CP of children of P
             AP ← AP \ P ∪ CP
End While
Return CurSol
```

Figure 2.4: B&B Algorithm

2.4.6 Constraint Logic Programming

Constraint Logic Programming is a class of programming languages at the intersection of Constraint Solving and Logic Programming (38). The paradigm of Logic Programming, as shown in Section 2.1 is *declarative* and thus allows a separation between *logic* and *control* (39). The *logic* part is responsible for correctness and describes information, given as facts and relations, which must be manipulated and combined to compute the desired result. The *control* part is responsible for efficiency, and embeds the strategies and control of the manipulations and combinations. An ideal programming methodology would be first of all concerned with *what* is the desired result; then, if necessary, with *efficiency*, or *how* to obtain the result. In Prolog, the most common logic programming language, the *logic* is given by *first order predicate logic* and the *control* is based on *SLD-resolution*.

However, one drawback of Logic Programming is that the manipulated objects are uninterpreted structures and equality only holds between objects that are syntactically identical. So, for instance, $1 + 2$ is a syntactic object that cannot unify with 3.

The Constrain Logic Programming (CLP) scheme was introduced by Jaffar and Lassez

(40). It represents a class of declarative languages, CLP($\mathcal{C}$) which are parametric in the constraint domain $\mathcal{C}$. For example, $\mathcal{C}$ can be the set of real numbers, and we have the CLP($\mathcal{R}$) (41), or the Finite Domain sort, thus we have CLP(FD). Constraint Logic Programming gives an interpretation to some of the syntactic structures and replaces unification with constraint solving in the domain of the computation.

More formally, the constraint domain $\mathcal{C}$ contains the following components (42):

- The *constraint domain signature* $\Sigma_{\mathcal{C}}$. It is a signature (i.e., a set of functions and predicate symbols) that represents the subset of the syntactic structures that are interpreted in the domain $\mathcal{C}$.
- The *class of constraints* $\mathcal{L}_{\mathcal{C}}$, which is a set of fist-order formulas.
- The *domain of computation* $\mathcal{D}_{\mathcal{C}}$, which is a Σ-structure that is the intended interpretation of the constraints. It consists of a set D and a mapping from the symbols in $\Sigma_{\mathcal{C}}$ to the set D.
- The *constraint theory*, $\mathcal{T}_{\mathcal{C}}$, which is a Σ-theory that describes the logical semantics of the constraints.
- The *solver*, *solv*$_{\mathcal{C}}$, which is a solver for $\mathcal{L}_{\mathcal{C}}$, i.e., a function that maps each formula to $true$, $false$ or $unknown$, indicating that the set of constraints is satisfiable, unsatisfiable or that it cannot tell.

In this thesis, we will mainly refer to the FD and $\mathcal{R}$ sorts. In FD the domain of the computation can be any finite set of elements. One of the most important instances is the set of integers comprised between two values $[-Max.. + Max]$: in this case the signature contains the integers, plus operations like $+, -, *, /$ and constraints like $=, <, >, \leq, \geq$ that are interpreted as in mathematics. In most CLP languages (43, 44, 45), the solver $solv_{FD}$ is based on (Generalized) Arc-Consistency.

There are various implementations of the $\mathcal{R}$ sort with different characteristics, we will mostly refer to the one presented in (46).

3

Computational Logic tools for Green IT

3.1 Introduction

Service Oriented Architectures (SOA), exploiting the Internet as a means of communication, are emerging as a simple and effective paradigm for distributed application development. Heterogeneous entities, in terms of hardware and software settings, can interoperate effectively following well established communication standards.

This 'decentralization' of software has proved not only to be beneficial under the point of view of software engineering but also with respect to sustainability. Computing requirements have risen sharply over the last years, and data centres have acquired a considerable and growing impact in terms of energy consumption and carbon footprint. According to the American Environmental Protection Agency, data centers consumed 0.7% of total US electricity in 2000 and 1.5% in 2006, while the Department of Energy estimate for 2010 is around 3% (1).

SOA enables businesses to delegate tasks to specialized structures which can benefit of economies of scale and thus the whole system can use resources in a more flexible and efficient way. Some independent analyst firms have conducted quantitative studies in order to show that

> "the growth of cloud computing will have a very significant positive effect on data center energy consumption. Few, if any, clean technologies have the capability to reduce energy expenditures and greenhouse gas production with so little business disruption. Software as a service, infrastructure as a service, and platform as a service are all inherently more efficient models than conventional alternatives, and

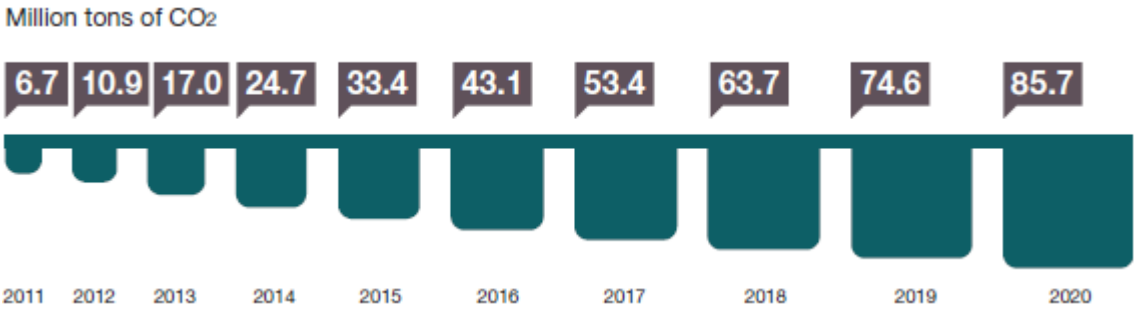

Figure 3.1: Model derived carbon emission savings enabled by cloud computing according to (1)

> their adoption will be one of the largest contributing factors to the greening of enterprise IT." (47).

According to one of the available reports, "US businesses with annual revenues of more than $1 billion can cut carbon emissions by 85.7 million metric tons annually by 2020 as a result of spending 69% of infrastructure, platform and software budgets on cloud services" (1), see also Figure 3.1.

However organizing software as a composition of services introduces elements of considerable complexity related to the need to coordinate their interaction. Interoperability of services has a 'syntactic' aspect, which can be achieved by using standards and expressing interfaces of what types of data are taken in input and produced in output by services (48). But it needs to be accompanied by a 'semantic' one, with a formal specification of what the service 'does' and the technological intelligent tools to understand and elaborate them.

3.1.1 Semantic Web Services

Service providers can expose their policies, in order for potential customers to evaluate the feasibility of a fruitful interaction. Such an evaluation can be performed manually, if the service's interface is bound not to change over time.

However, a more promising approach is for customers to choose the appropriate service provider at run-time, based on the service provider's exposed policies.

This approach requires a reasoning engine that reasons on the provider and customer's goals and policies, in order to devise a sequence of actions that lets both achieve their goals, while respecting their policies.

In (49), Alberti et al. proposed a contracting architecture based on the SCIFF abductive logic framework presented in Section 2.3. In that work, the policies were expressed by means of integrity constraints, and the domain knowledge was expressed in SCIFF's logic clauses. The sound and complete SCIFF proof procedure was employed to find, if possible, a course of action that was satisfactory for both.

However, in a practical perspective, we envisage a possible improvement in representing the domain specific knowledge exploiting the vast body of results from the Knowledge Representation field and, in particular, Description Logics. In this way, we can address the Ontology layer of the Semantic Web stack.

In this chapter, we propose an approach to let SCIFF access existing knowledge, expressed by means of an ontology: by interfacing SCIFF with ontological reasoners, distinguishing syntactically the ontology-related predicates and delegating their computation to the external reasoners.

This approach bears two main practical advantages: first, it makes the knowledge encoded in ontologies available to SCIFF, and second it takes advantage of the formal properties enjoyed by the existing ontological reasoners for the associated computational tasks, in particular concerning decidability and efficiency.

Moreover we deal with a dual aspect of contracting: learning and updating policies from existent interactions. This is a very important issue because policies can be rather complex and even experts of the domain can find it very difficult to express them in a formal language. The overall process can be very time-consuming and error-prone, while classification of an interaction as compliant (or not) with respect to the intended behaviour is usually easier.

Applying techniques inspired by Inductive Logic Programming (see Section 2.2) authors of (50) propose a framework for policy learning. We extend it in order to include the cases, often relevant in the dynamic domain of Web Services, in which policies are already partially specified and one wants to update them taking advantage of new available knowledge.

After recalling in Section 3.2 how Semantic Web Services contracting can be represented in SCIFF we show, in Section 3.3, how it could benefit of added ontological expressivity. We describe a possible implementation and its experimental evaluation in Sections 3.4.1 and 3.4.2 respectively.

In Section 3.5 we discuss in detail the problem of learning policies from histories of previous interactions. After explaining how this knowledge can be represented in logic in Section 3.6, we present a framework designed to learn policies from such information (Section 3.7) and

its extension which is able to update policies when new knowledge is available (Section 3.8). The two approaches are compared in Section 3.9. In Section 3.10 we discuss related work in the field.

3.2 Contracting with SCIFF

In this section, we briefly recall the SCIFF-based contracting framework (49).

A web service's policy is defined, in the SCIFF language, as an abductive logic program (ALP). In particular, the integrity constraints can relate the web services' information exchanges with the expected input from peer web services. The possible relations can be of various types, and include temporal relations, such as deadlines, linear constraints, inequalities and disequalities, all defined by means of constraints. The definitions stated in this way are then used to make assumptions on the possible evolutions of the interaction.

3.2.1 A contracting scenario

In (49) the SCIFF framework is applied to contracting in an e-commerce scenario, which we extend here in order to demonstrate our approach to integration.

The two actors are *eShop* (which wants to sell a device) and *alice* (a potential customer for that device), each with policies expressed as SCIFF ICs. SCIFF is used as a reasoner in a component that acts as a mediator, considering the actors' policies and trying to devise a course of action that will let both reach their goals. The two actors' policies, expressed in SCIFF integrity constraints, are as follows.

***alice*'s policy.** "If the shop asks me to pay cash, I will, but if the shop asks me to pay by credit card, I will require evidence of the shop's affiliation to the Better Business Bureau."

$$\begin{array}{l}
\mathbf{H}(tell(Shop, alice, ask(pay(Item, cc))), Ta) \rightarrow \\
\mathbf{H}(tell(alice, Shop, request_guar(BBB)), Trg) \wedge \\
\mathbf{E}(tell(Shop, alice, give_guar(BBB)), Tg) \wedge Tg > Trg \wedge Trg > Ta. \\
\\
\mathbf{H}(tell(Shop, alice, ask(pay(Item, cc))), Ta) \wedge \\
\mathbf{H}(tell(Shop, alice, give_guar(BBB)), Tg) \rightarrow \\
\mathbf{H}(tell(alice, Shop, pay(Item, cc)), Tp) \wedge Tp > Ta \wedge Tp > Tg. \\
\\
\mathbf{H}(tell(Shop, alice, ask(pay(Item, cash))), Ta) \rightarrow \\
\mathbf{H}(tell(alice, eShop, pay(Item, cash)), Tr) \wedge Ta < Tr.
\end{array} \tag{3.1}$$

***eShop*'s policy.** "If an acceptable customer requests an item from me, then I expect the customer to pay for the item with an acceptable means of payment. If the customer is not acceptable, I will inform him/her of the failure. If an acceptable customer pays with an acceptable means of payment, I will deliver the item. If a customer requests evidence of my affiliation to the Better Business Bureau (BBB), I will provide it."

$$\begin{array}{l}
\mathbf{H}(tell(Customer, eShop, request(Item)), Tr) \wedge accepted_payment(How) \rightarrow \\
accepted_customer(Customer) \wedge \mathbf{H}(tell(eShop, Customer, ask(pay(Item, How))), Ta) \wedge \\
\mathbf{E}(tell(Customer, eShop, pay(Item, How)), Tp) \wedge Tp > Ta \wedge Ta > Tr \\
\vee rejected_customer(Customer) \wedge \mathbf{H}(tell(eShop, Customer, inform(fail)), Ti) \wedge Ti > Tr. \\
\\
\mathbf{H}(tell(Customer, eShop, pay(Item, How)), Tp) \\
\wedge accepted_customer(Customer) \wedge accepted_payment(How) \rightarrow \\
\mathbf{H}(tell(eShop, Customer, deliver(Item)), Td) \wedge Td > Tp. \\
\\
\mathbf{H}(tell(Customer, eShop, request_guar(BBB)), Trg) \rightarrow \\
\mathbf{H}(tell(eShop, Customer, give_guar(BBB)), Tg) \wedge Tg > Trg.
\end{array} \tag{3.2}$$

The notion of acceptability for customers and payment methods from *eShop*'s viewpoint, defined by the *accepted_customer/1* and *accepted_payment/1* predicates, is defined in *eShop*'s knowledge base. In (49), only EU residents are accepted customers, as defined by the following clauses:

```
accepted_customer(Customer):-
    resident_in(Customer, Location),
    accepted_destination(Location).
rejected_customer(Customer):-
    resident_in(Customer, Location),
    not accepted_destination(Location).
accepted_destination(european_union).
accepted_payment(cc).
accepted_payment(cash).
```

This knowledge is merged with that provided by the customer, for example

```
resident_in(alice,european_union).
```

so that in the resulting knowledge base, on which SCIFF operates, `accepted_customer(alice)` is true.

However, this method would not work in case the customer declared `resident_in(alice,italy)`, as `italy` does not unify with `european_union`. One could, of course, add to the knowledge base `accepted_destination/1` facts for all the current EU members, but such knowledge should be updated when new countries join the EU. In other cases, acceptability could be defined by a transitive, symmetric relation, which could introduce cycles in the knowledge base, possibly leading to loops.

3.3 Representing domain knowledge with ontologies

An alternative way to represent (part of) the domain specific knowledge is to use technology and concepts developed, with focus on this very purpose, in the Knowledge Representation field, and to rely, in particular, on ontologies. The W3C recommendation for ontology representation on the Web is the Web Ontology Language (OWL) (51) based on the well established semantics of Description Logics (52) and on XML and RDF syntax. Using OWL for domain

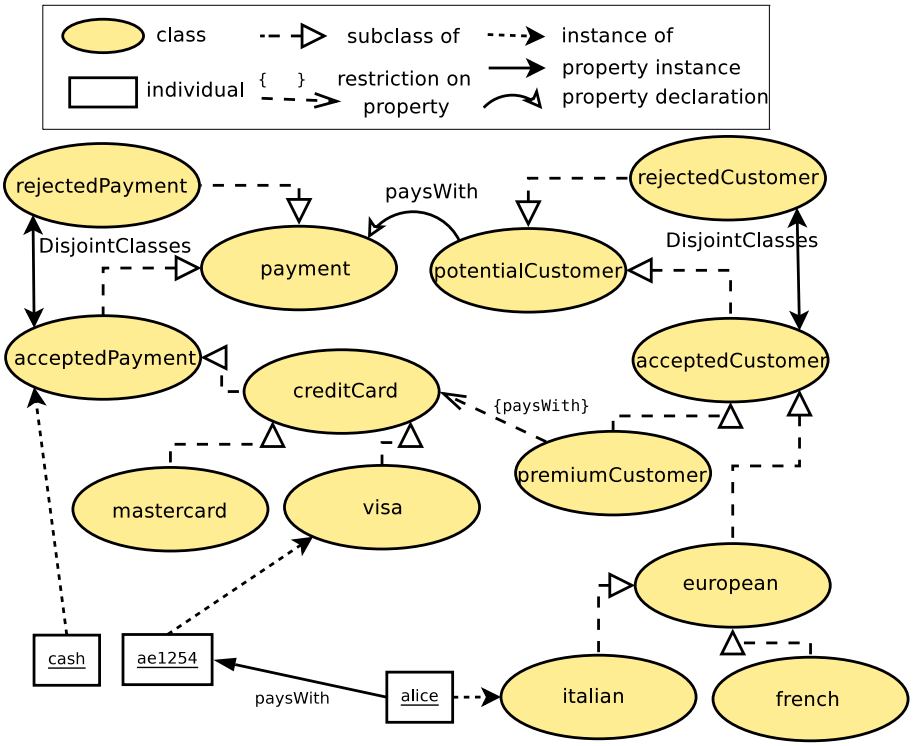

Figure 3.2: A graphical representation of the ontology

knowledge representation improves expressiveness (with such features as stating sub-class relations, constructing classes on property restrictions or by set operators, defining transitive properties and so on) yet keeping decidability (if using OWL Lite or OWL DL) in a straightforward and domain modeling-oriented notation. Moreover, since OWL is tailored for the Web, it provides support for expressing knowledge in distributed contexts (identified by URIs) and its recognized standard status is a warranty on interoperability and reuse issues. As a plus, it can be mentioned that community driven development of Semantic Web tools already provides good support for OWL ontology management tasks such as editing (53) also for not KR-skilled users.

For instance, in Fig. 3.2 we show a possible ontological representation of *eShop*'s policies concerning acceptable customers and means of payments, merged with *alice*'s own knowledge.

The following OWL axiom says that the `acceptedCustomer` class is a subclass of the `potentialCustomer` class, and that it is disjoint from the

rejectedCustomer class:

```
<owl:Class rdf:about="#acceptedCustomer">
     <rdfs:subClassOf rdf:resource="#potentialCustomer" />
     <owl:disjointWith rdf:resource="#rejectedCustomer" />
</owl:Class>
```

The following fact states that cash is an instance of the acceptedPayment class:

```
<owl:Thing rdf:about="#cash">
      <rdf:type rdf:resource="#acceptedPayment" />
</owl:Thing>
```

The following axiom declares the paysWith property:

```
<owl:ObjectProperty rdf:ID="paysWith">
    <rdfs:domain rdf:resource="#potentialCustomer" />
    <rdfs:range rdf:resource="#payment" />
</owl:ObjectProperty>
```

The following fact states that alice is an instance of italian, with value ae1254 for the paysWith property:

```
<owl:Thing rdf:about="#alice">
     <rdf:type rdf:resource="#italian" />
     <paysWith rdf:resource="#ae1254" />
</owl:Thing>
```

It can be noticed that the ontological notation for the KB makes it possible to infer that alice is European (and therefore an accepted customer) even if she just states being Italian, while if we had put resident_in(alice, italy) instead of resident_in(alice, europe) in the KB at the end of Sect. 3.2.1 she would not have been recognized as such.

Moreover, we defined a class premiumCustomer representing the accepted customers who pay with a credit card, and which we could use to add refinement to policies (for instance providing a faster delivery). Since alice is an accepted customer and pays with her credit card, the ontological reasoning allows to recognize her as a premiumCustomer.

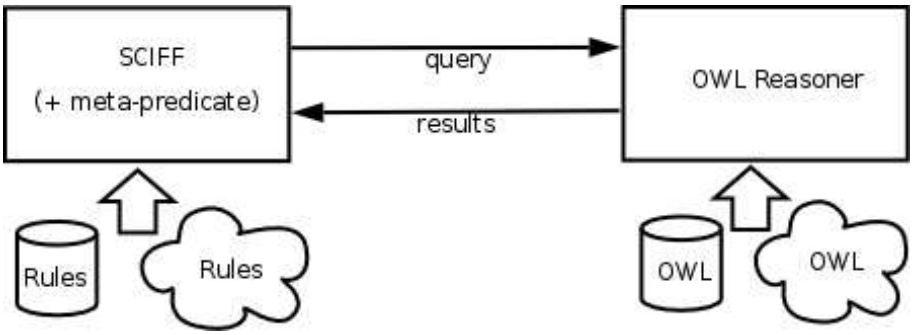

Figure 3.3: Integration architecture

3.4 Handling semantic knowledge with SCIFF

In this section we describe our implementation of the SCIFF access to OWL ontologies. It should be noticed that even if OWL relies on the Open World Assumption, when SCIFF comes to reasoning involving it, the logic programming peculiar Closed World Assumption is reintroduced, thus assuming to have all (relevant) information on the domain available at that time and providing usual features such as negation as failure.

3.4.1 Interfacing SCIFF and ontological reasoners

An external specific ontology-focused component interfaced with SCIFF can be, when necessary, queried in order to perform the actual ontological reasoning and give back results. As represented in Fig. 3.3, the architecture for this solution involves a Prolog meta-predicate which invokes the ontological reasoning on desired goals, an intercommunication interface from SCIFF to the external component (which incorporates a query and results translation schema) and the actual reasoning module. Both modules can access both local and networked knowledge.

Goals given to the meta-predicate are handled like suggested in (54) and (55) considering single arity predicates as “belongs to class (with same name of predicate)" queries and double arity ones as “are related by property (with same name of predicate)" queries. To reduce the overhead caused by external communication, our implementation of the meta-predicate provides a caching mechanism: it is first checked if a similar query (i.e., involving the same predicate) has been performed before and, only if not, the external reasoner is invoked and answers are cached by storing them as Prolog facts (by means of `asserta/1}`) . The OWL reasoning module uses the Pellet (56) API, while the communication interface uses the Jasper

N	Load time	Query time	Total
100	~ 0	~ 0	~ 0
500	1.0	~ 0	1.0
1000	1.0	~ 0	1.0
5000	2.0	1.2	3.2
10000	4.0	2.8	6.8

Table 3.1: Performance results (all times are in seconds, average over 50 runs)

Prolog-Java library (44). This solution enables access to the full OWL(-DL) expressivity, including features such as equivalence of classes and properties, transitive properties, declaration of classes on property restriction and property-based individual classification.

3.4.2 Experimental evaluation

The approach was tested successfully in simple reasoning scenarios, such as the one depicted in Fig. 3.2.

To assess its scalability, we experimented with randomly generated ontologies. Each ontology, composed of N classes, was built starting from its root node, and recursively attempting, for each node, five attempts of child generation, each with probability $1/3$.

The reasoner was queried about the belonging of an entity to the hierarchy root class. We report in Tab. 3.1 the time spent for loading the ontology into the reasoner (i.e. the time spent for parsing ICs and loading the ontology into a persistent OWLOntology object) and for the actual query. The approach appears to scale reasonably (on PC equipped with an Intel Celeron 2.4 GHz CPU), the loading time is usually higher than the query time.

3.5 Learning and Updating Policies

We have shown how the possibility to express Web Service policies in formal languages and the use of Computational Logic tools to perform reasoning about their composition and coordination has great potential to ease the complexity of these operations, thus bringing the SOA scenario closer to a feasible extended adoption.

However, writing policies in a formal language is often a very complicated issue itself. Sometimes not enough knowledge about the domain is available, or the domain expert finds it difficult to express it formally.

For this reason it is often desirable to be able to let computers do the learning of how policies should be formalized abstracting from the observation of actual interactions which can be easily labeled as compliant or not compliant by a (human) classifier.

Moreover the dynamic environment in which such services usually operate calls for a suitable approach to policy updating when something changes, new evidence is available or a draft of policy compiled by a human expert needs to be refined.

In order to study this aspect we can get inspiration from the research in the field of Business Process Management (BPM) (see e.g. (57)).

In the current knowledge society, the set of business processes an organization performs in order to achieve its mission often represents one of the most important assets of the organization. It is thus necessary to be able to describe them in details, so that they can be stored and analyzed. In this way we can preserve and/or improve them.

Often organizations do not have a formal or precise description of the processes that they perform. The knowledge necessary to execute the processes is owned by the individual workers but not by the organization as a whole, thus exposing it to possible malfunctions if a worker leaves.

However, modern information systems store all the actions performed by individual workers during the execution of a process. These action sequences are called *traces* and the set of all the traces recorded in a period of time is called a *log*. The Process Mining research area (58) proposes techniques for inferring a model of the process from a log.

Very often processes change over time to reflect mutated external or internal conditions. In this case, it is necessary to update their models. In particular, given a process model and a new log, we want to modify the model so that it conforms also with the new log. We call this activity Incremental Process Mining. In this chapter we show that revising an existing model may be more effective than learning a new model from scratch from the previous and new log. Moreover, in some cases the previous log may not be available, thus making model updating necessary.

We present a technique for solving this problem: we modify the process mining system DPML (Declarative Process Model Learner) (50) to be able to update a preexisting model given new traces.

We choose Logic Programming for the representation of traces and process models in order to exploit its expressiveness for the description of traces together with the wide variety of learning techniques available for it. An activity can be represented as a logical atom in which

the predicate name indicates the type of action and the arguments indicate the attributes of the action. One of the attributes is the time at which the action has been performed. Thus a trace can be represented as a set of instantiated atoms, i.e., a logical interpretation.

In order to represent process models, we use a subset of the SCIFF language (19, 59). A model in this language is a set of logical integrity constraints in the form of implications. Given a SCIFF model and a trace, there exists an interpreter that checks whether the trace satisfies or not the model. Such a representation of traces and models is declarative in the sense that we do not explicitly state the allowed execution flows but we only impose high level constraints on them.

DPML is able to infer a SCIFF theory from a set of positive and negative traces. Positive traces represent correct executions of the business process, while negative traces represent process executions that have been judged incorrect or undesirable.

Given that these traces are represented as logical interpretations and that SCIFF integrity constraints are similar to logical clauses, DPML employs Inductive Logic Programming techniques (12) for learning from interpretations. In particular, it modifies the ICL system (13) that learns sets of logical clauses from positive and negative interpretations.

We present the system IDPML (Incremental Declarative Process Model Learner) that faces the problem of revising an existing theory in the light of new evidence. This system is an adaptation of DPML and adopts techniques developed in Inductive Logic Programming (such as (17)) to perform theory revision.

IDPML generalizes theories that do not satisfy new positive traces, as well as specializes theories that do not exclude new negative examples. To this purpose, we exploit the generalization operator presented in (50) for SCIFF theories. Moreover, we define a specialization operator.

IDPML is experimentally evaluated revising the model of a process regarding a hotel management and the model of an electronic auction protocol. In the "hotel management" case the available traces are divided into two sets: one containing “old” traces and one containing “new” traces. Then two experiments are performed: in the first we learn a theory with DPML using the “old” traces and we revise the theory with IDPML using the “new” traces, while in the latter we learn a theory with DPML from “old” and “new” traces. Then we compare the accuracy of the final theories obtained and the running time. A similar comparison is performed on the auction protocol, except that in this case the initial theory is not learned using some “old” traces but it is a modified (worse) version of the actual model to simulate the revision of an

(imperfect) theory written down by a user. Results associated to these experiments show that revising a theory is more efficient that inducing it from scratch. Moreover, the models obtained are more accurate on unseen data.

3.5.1 Business Process Management

Every organization performs a number of *business processes* in order to achieve its mission. Complex organizations are characterized by complex processes, involving many people, activities and resources. The performances of an organization depend on how accurately and efficiently it enacts its business processes. Formal ways of representing business processes have been studied in the area of Business Processes Management (see e.g. (60)), so that the actual enactment of a process can be checked for compliance with a model.

Recently, the problem of automatically inferring such a model from data has been studied by many authors (see e.g. (58, 61, 62)). This problem has been called Process Mining or Workflow Mining. The data in this case consists of execution traces (or histories) of the business process. The collection of such data is made possible by the facility offered by many information systems of logging the activities performed by users.

Let us now describe in detail the problem that is solved by Process Mining. A *process trace* T is a sequence of events. Each event is described by a number of attributes. The only requirement is that one of the attributes describes the event type. Other attributes may be the executor of the event or event specific information.

An example of a trace is

$\langle a, b, d \rangle$

that means that activity a was performed first, then b and finally d.

A *process model* PM is a description of the process in a language that expresses the conditions a trace must satisfy to be compliant with the process, i.e., to be a correct enactment of the process. An interpreter of the language must exists that, when applied to a model PM and a trace T, returns *yes* if the trace is compliant with the description and *no* otherwise. In the first case we write $T \models PM$, in the second case $T \not\models PM$. A bag of process traces L is called a *log*. Often, in Process Mining, only compliant traces are used as input of the learning algorithm, see e.g. (58, 61, 62). We consider instead the case where we are given both compliant and non compliant traces. This is the case when we want to distinguish successful process executions from unsuccessful ones.

The approaches presented in (58, 61, 62) aim at discovering complex and procedural process models, and differ by the structural patterns they are able to mine. While recognizing the extreme importance of such approaches, recently (63) pointed out the necessity of discovering declarative logic-based knowledge, in the form of process fragments or business rules/policies, from execution traces. Declarative languages seem to fit better complex, unpredictable processes, where a good balance between support and flexibility is of key importance.

(63) presents a graphical language for specifying process flows in a declarative manner. The language, called ConDec, does not completely fix the control flow among activities, but rather envisages a set of constraints expressing policies/business rules for specifying what is forbidden or mandatory in the process. Therefore, the approach is inherently open and flexible, because workers can perform actions if those are not explicitly forbidden. ConDec adopts an underlying semantics by means of Linear Temporal Logics (LTL), and can also be mapped onto the logic programming-based framework SCIFF.

An important topic related to declarative process specifications concerns their discovery starting from execution traces, i.e., *Declarative Process Mining*. Most works in this field deal with the discovery of procedural process models (such as Petri Nets or Event-driven Process Chains (64, 65)) from data. Recently, some works have started to appear on the discovery of logic-based declarative models: (50, 66, 67) study the possibility of inferring essential process constraints, easily understandable by business analysts and not affected by procedural details.

3.6 Representing Process Traces and Models with Logic

A process trace can be represented as a logical interpretation: each event is modeled with an atom whose predicate is the event type and whose arguments store the attributes of the action. Moreover, an extra argument is added to the atom indicating the position in the sequence. For example, the trace:

$\langle a, b, d \rangle$

can be represented with the interpretation

$\{a(1), b(2), d(3)\}$.

If the execution time is an attribute of the event, then the position in the sequence can be omitted.

Besides the trace, we may have some general knowledge that is valid for all traces. We assume that this background information can be represented as a normal logic program B. The

rules of B allow one to complete the information present in a trace I: rather than simply I, we now consider $M(B \cup I)$, the model of the program $B \cup I$ according to one of the semantics for normal logic programs.

For example, consider the trace

$I = \{ask\_price(bike, 1), tell\_price(500, 2), buy(bike, 3)\}$

of a bike retail store and the background theory

$B = \{high\_price(T) \leftarrow tell\_price(P, T), P \geq 400\}$.

that expresses information regarding price perceptions by clients. Then $M(B \cup I)$ is

$\{ask\_price(bike, 1), tell\_price(500, 2), high\_price(2), buy(bike, 3)\}$

in which the information available in the trace has been enlarged by using the background information.

The process language we consider was proposed in (50) and is a subset of the $\mathcal{S}$CIFF language (see Section 2.3.1.1).

A process model in our language is a set of *integrity constraints*.

An *IC C is true in an interpretation* $M(B \cup I)$, written $M(B \cup I) \models C$, if, for every substitution θ for which $Body$ is true in $M(B \cup I)$, there exists a disjunct $\exists(ConjP_i)$ or $\forall\neg(ConjN_j)$ that is true in $M(B \cup I)$. If $M(B \cup I) \models C$ we say that the trace I is *compliant* with C.

In (50) it is shown that the truth of an IC in an interpretation $M(B \cup I)$ can be tested by running the query:

?-$Body, \neg(ConjP_1), \ldots \neg(ConjP_n), ConjN_1, \ldots, ConjN_m$

against a Prolog database containing the clauses of B and atoms of I as facts.

If the N conjunctions in the head share some variables, then the following query must be issued

?-$Body, \neg(ConjP_1), \ldots \neg(ConjP_n), \neg(\neg(ConjN_1)), \ldots, \neg(\neg(ConjN_m))$

that ensures that the N conjunctions are tested separately without instantiating the variables.

Let us recall IC 2.5 from Section 2.3.1.1:

$$\begin{aligned} & a(bob, T), T < 10 \\ \rightarrow & \exists(b(alice, T1), T < T1) \\ & \vee \\ & \forall\neg(c(mary, T1), T < T1, T1 < T + 10) \end{aligned}$$

For IC 2.5, the query is

?-$a(bob, T), T < 10, \neg(b(alice, T1), T < T1), \neg(\neg(c(mary, T1), T < T1, T1 < T + 10))$

If the query finitely fails, the IC is true in the interpretation. If the query succeeds, the IC is false in the interpretation. Otherwise nothing can be said. It is the user's responsibility to write the background B in such a way that no query generates an infinite loop. For example, if B is acyclic then a large class of queries will be terminating (68).

A *process model H is true in an interpretation $M(B \cup I)$* if every IC of H is true in it and we write $M(B \cup I) \models H$. We also say that trace I is compliant with H.

3.7 Learning ICs Theories

In order to learn a theory that describes a process, we must search the space of ICs. To this purpose, we need to define a generality order in such a space.

IC C is *more general than* IC D if C is true in a superset of the traces where D is true. If $D \models C$, then C is more general than D.

Similarly to the case of clauses, (50) defined the notion of θ-subsumption also for ICs.

Definition 9 (Subsumption) *An IC D θ-*subsumes *an IC C, written $D \geq C$, iff a substitution θ exists for the variables in the body of D or in the N conjunctions of D such that*

- $Body(D)\theta \subseteq Body(C)$ *and*
- $\forall ConjP(D) \in HeadSet(D)$, $\exists ConjP(C) \in HeadSet(C) : ConjP(C) \subseteq ConjP(D)\theta$ *and*
- $\forall ConjN(D) \in HeadSet(D)$, $\exists ConjN(C) \in HeadSet(C) : ConjN(D)\theta \subseteq ConjN(C)$

For example, IC 2.5 is subsumed by the IC

$$\begin{aligned} & a(bob, 4) \\ \rightarrow & \exists(b(alice, T1), 4 < T1, T1 < 4 + 10) \\ & \vee \\ & \forall\neg(c(mary, 5), 4 < 5) \end{aligned} \tag{3.3}$$

with the substitution $\{T/4, T1/5\}$.

It was proved in (50) that implication and θ-subsumption for ICs share the same relation as for clauses.

Theorem 1 $D \geq C \Rightarrow D \models C$.

Thus, θ-subsumption can be used for defining a notion of generality among ICs, which can be used in learning algorithms.

In order to define a refinement operator, we must first define the language bias. We use a language bias that consists of a set of IC templates. Each template specifies

- a set of literals BS allowed in the body,
- a set of disjuncts HS allowed in the head. Each disjunct is represented as a pair $(Sign, DS)$ where
  - $Sign$ is either $+$ or $-$ and specifies where it is a P or an N disjunct,
  - DS is the set of literals allowed in the disjunct.

(50) defined a refinement operator from specific to general (upward operator) in the following way: given an IC D, the set of *upward refinements* $\delta(D)$ of D is obtained by performing one of the following operations

- adding a literal from BS to the body;
- removing a literal from a P disjunct in the head;
- adding a literal to an N disjunct in the head;
- adding a disjunct from HS to the head: the disjunct can be
  - a formula $\exists(d_1 \wedge \ldots \wedge d_k)$ where $DS = \{d_1, \ldots, d_k\}$ is the set of literals allowed by the IC template for D for a P disjunct,
  - a formula $\forall\neg(d)$ where d is allowed by the IC template for D for a N disjunct.

Here we also define a refinement operator from general to specific (downward operator) because we will need it to perform theory revision. The operator inverts the operations performed in the upward operator, i.e., given an IC D, the set of *downward refinements* $\rho(D)$ of D is obtained by performing one of the following operations

- removing a literal from the body of D;
- adding a literal to a P disjunct in the head;

- removing a literal from an N disjunct in the head;
- removing a disjunct from the head when
  - it is a P disjunct $\exists(d_1 \wedge \ldots \wedge d_k)$ where $\{d_1, \ldots, d_k\}$ is the set of literals allowed by the IC template for D for the P disjunct,
  - it is an N disjunct containing a single literal $\forall\neg(d)$.

The learning problem we consider is an adaptation to ICs of the learning from interpretation setting of ILP:

Given

- a space of possible process models $\mathcal{H}$
- a set E^+ of positive interpretations;
- a set E^- of negative interpretations;
- a background normal logic program B.

Find: a process model $H \in \mathcal{H}$ such that

- for all $T^+ \in E^+$, $M(B \cup T^+) \models H$;
- for all $T^- \in E^-$, $M(B \cup T^-) \not\models H$;

If $M(B \cup T) \models C$ we say that IC C *covers* the trace T and if $M(B \cup T) \not\models C$ we say that C *rules out* the trace T.

In order to solve the problem, (50) proposes the algorithm DPML (Declarative Process Model Learner) that is an adaptation of ICL (13).

DPML is obtained from ICL by using the testing procedure and the refinement operator defined for SCIFF ICs in place of those for logical clauses.

3.8 Incremental Learning of ICs Theories

We propose a variation of DPML able to deal with theory revision that we call Incremental DPML (IDPML). As in Section 2.2.1, we call $\mathcal{H}$ the space of possible theories, B the background theory, E^+ the set of previous positive examples, E^- the set of previous negative ones

and T the theory (obtained by DPML or expressed by a human expert) we would like to refine to make it consistent with the new examples: $Enew^-$ and $Enew^+$. Figure 3.4 shows the IDPML algorithm.

The initial theory, together with old and new positive examples and old negative ones, is given as input to RevisePositive whose aim is to revise the theory in order to cover as many positive examples as possible. The output of RevisePositive is then given as input, together with all sets of examples, to ReviseNegative, whose revision tries to rule out the negative examples, and whose output is the overall revised theory.

RevisePositive cycles on new positive examples and finds out which constraints (if any) of the previous theory are violated for each example. An inner cycle generalizes all such constraints in order to make the theory cover the ruled out positive example. The generalization function performs a beam search with $p(\ominus|\overline{C})$ as the heuristic (see Section 2.2) and δ as the refinements operator (see Section 3.7). For theory revision, however, the beam is not initialized with the most specific constraint (i.e. $\{false \leftarrow true\}$) but with the violated constraint.

Since some of the previously ruled out negative examples may be again covered after the generalization process, ReviseNegative checks at first which negative examples, either old or new, are not ruled out. Then it selects randomly an IC from the theory and it performs a specialization cycle until no negative examples are covered. The Specialize function is similar to the Generalize one with δ replaced with ρ as the refinement operator (see Section 3.7).

It is also possible that some negative examples cannot be ruled out just by specializing existing constraints, so after ReviseNegative a covering loop (as the one of DPML) has to be performed on all positive examples and on the negative ones which are still to be ruled out.

3.9 Experiments

In this section we present some experiments that have been performed for investigating the effectiveness of IDPML. In particular, we want to demonstrate that, given an initial theory H and a new set of examples $Enew$, it can be more beneficial to revise H in the light of $Enew$ than to learn a theory from $E \cup Enew$. Another use case consists in the revision of an (imperfect) theory written down by a user and its comparison with the theory learned from scratch using the same set of examples.

function IDPML($T, E^+, E^-, Enew^+, Enew^-, B$)
  H:= RevisePositive($T, E^+, E^-, Enew^+, B$)
  H:= ReviseNegative($H, E^+, E^-, Enew^+, Enew^-, B$)
  H:=$H \cup$ DPML($E^+ \cup Enew^+, RuledOut(Enew^-, H), B$)
  return H

function RevisePositive($T, E^+, E^-, Enew^+, B$)
  foreach $e^+ \in Enew^+$
    VC:=FindViolatedConstraints(T, e^+)
    T:= $T - VC$
    E^+:=$E^+ \cup \{e^+\}$
    foreach $vc \in VC$
      c:= Generalize(vc, E^+, E^-, B)
      T:= $T \cup \{Best(c, vc)\}$
  return T

function Generalize(vc, E^+, E^-, B)
  $Beam$:=$\{vc\}$
  $BestClause$:= $\emptyset$
  while $Beam \neq \emptyset$
    foreach $c \in Beam$
      foreach ref of c
        $BestClause$:= $Best(ref, c)$
        $Beam$:= $Beam \cup \{ref\}$
        if $size(Beam) > MaxBeamSize$
          $Beam$:= $Beam - \{Worst(Beam)\}$
  return $Beam$

function ReviseNegative($T, E^+, E^-, Enew^+, Enew^-, B$)
  $Enew^-$:=TestNegative($T, E^-, Enew^-$)
  E^+:=$E^+ \cup Enew^+$
  H:=$\emptyset$
  while $T \neq \emptyset \wedge Enew^- \neq \emptyset$
    pick randomly an IC c from T
    T:=$T - \{c\}$
    nc:= Specialize($c, E^+, Enew^-, B$)
    H:=$H \cup \{Best(c, nc)\}$
    $Enew^-$:= $Enew^- - RuledOut(Enew^-, Best(c, nc))$
  return H

Figure 3.4: IDPML algorithm

3.9.1 Hotel Management

Let us first consider a process model regarding the management of a hotel and inspired by (69). We generated randomly a number of traces for this process, we classified them with the model and then we applied both DPML and IDPML.

The model describes a simple process of renting rooms and services in a hotel. Every process instance starts with the registration of the client name and her preferred way of payment (e.g., credit card). Data can also be altered at a later time (e.g., the client may decide to use another credit card). During her stay, the client can require one or more room and laundry services. Each service, identified by a code, is followed by the respective registration of the service costs into the client bill. The cost of each service must be registered only if the service has been effectively provided to the client and it must be registered only once. The cost related to the nights spent in the hotel must be billed. It is possible for the total bill to be charged at several stages during the stay.

This process was modeled by using eight activities and eight constraints. Activities *register_client_data*, *check_out* and *charge* are about the check-in/check-out of the client and expense charging. Activities *room_service* and *laundry_service* log which services have been accessed to by the client, while billings for each service are represented by the corresponding activities. For each activity, a unique identifier is introduced to correctly charge the clients with the price for the services they effectively made use of.

Business related aspects of our example are represented as follows:

- (C.1) every process instance starts with activity *register_client_data*. No limits on the repetition of this activity are expressed, hence allowing alteration of data;
- (C.2) *bill_room_service* must be executed after each *room_service* activity, and *bill_room_service* can be executed only if the *room_service* activity has been executed before;
- (C.3) *bill_laundry_service* must be executed after each *laundry_service* activity, and *bill_laundry_service* can be executed only if the*laundry_service* activity has been executed before;
- (C.4) *check_out* must be performed in every process instance;
- (C.5) *charge* must be performed in every process instance;
- (C.6) *bill_nights* must be performed in every process instance.

- (C.7) *bill_room_service* must be executed only one time for each service identifier;
- (C.8) *bill_laundry_service* must be executed only one time for each service identifier;

The process model is composed of the following ICs:

$$
\begin{aligned}
(IC.1)\quad & true \\
& \rightarrow \exists(register\_client\_data(Trcd) \wedge Trcd = 1).
\end{aligned}
$$

$$
\begin{aligned}
(IC.2)\quad & room\_service(rs\_id(IDrs), Trs) \\
& \rightarrow \exists(bill\_room\_service(rs\_id(IDbrs), Tbrs) \wedge \\
& IDrs = IDbrs \wedge Tbrs > Trs). \\
& bill\_room\_service(rs\_id(IDbrs), Tbrs) \\
& \rightarrow \exists(room\_service(rs\_id(IDrs), Trs) \wedge \\
& IDbrs = IDrs \wedge Trs < Tbrs).
\end{aligned}
$$

$$
\begin{aligned}
(IC.3)\quad & laundry\_service(la\_id(IDls), Tls) \\
& \rightarrow \exists(bill\_laundry\_service(la\_id(IDbls), Tbls) \wedge \\
& IDls = IDbls \wedge Tbls > Tls). \\
& bill\_laundry\_service(la\_id(IDbls), Tbls) \\
& \rightarrow \exists(laundry\_service(la\_id(IDls), Tls) \wedge \\
& IDbls = IDls \wedge Tls < Tbls).
\end{aligned}
$$

$$
\begin{aligned}
(IC.4)\quad & true \\
& \rightarrow \exists(check\_out(Tco)).
\end{aligned}
$$

$$
\begin{aligned}
(IC.5)\quad & true \\
& \rightarrow \exists(charge(Tch)).
\end{aligned}
$$

$$
\begin{aligned}
(IC.6)\quad & true \\
& \rightarrow \exists(bill\_nights(Tbn)).
\end{aligned}
$$

$$
\begin{aligned}
(IC.7)\quad & bill\_room\_service(rs\_id(IDbrs1), Tbrs1) \\
& \rightarrow \forall\neg(bill\_room\_service(rs\_id(IDbrs2), Tbrs2) \wedge \\
& IDbrs1 = IDbrs2 \wedge Tbrs2 > Tbrs1).
\end{aligned}
$$

$$
\begin{aligned}
(IC.8)\quad & bill\_laundry\_service(la\_id(IDbls1)), Tbls1) \\
& \rightarrow \forall\neg(bill\_laundry\_service(la\_id(IDbls2), Tbls2) \wedge \\
& IDbls1 = IDbls2 \wedge Tbls2 > Tbls1).
\end{aligned}
$$

Constraint $(C.1)$ has been mapped to a IC by imposing that the *"register_client_data"* activity is expected to happen at time 1 (the first activity in an execution trace).

	revision			full dataset		
dataset	time	accuracy		time	accuracy	
		μ	σ		μ	σ
1	4123	0.732539	0.0058	18963	0.702367	0.0068
2	4405	0.757939	0.0223	17348	0.686754	0.0269
3	6918	0.825067	0.0087	13480	0.662302	0.0180
4	3507	0.724764	0.0257	17786	0.679003	0.0248

Table 3.2: Revision compared to learning from full dataset for the hotel scenario

For this process, we randomly generated execution traces and we classified them with the above model. This was repeated until we obtained four training sets each composed of 2000 positive examples and 2000 negative examples. Each training set was randomly split into two subsets, one containing 1500 positive and 1500 negative examples, and the other containing 500 positive and 500 negatives examples. The first subset is used for getting an initial theory, while the second is used for the revision process.

DPML was applied to each training sets with 3000 examples. The theories that were obtained were given as input to IDPML together with one of the 1000 examples training set. Finally, DPML was applied to each of the complete 4000 examples training sets.

The models obtained by IDPML and by DPML on a complete training set were then applied to each example of the other three training set in order to measure the accuracy. In this case it is defined as the number of compliant traces that are correctly classified as compliant plus the number of non-compliant traces that are correctly classified as non-compliant divided by the total number of traces.

In table 3.2 we show a comparison of time spent (in seconds) and resulting accuracies from the theory revision process and from learning based on the full dataset. The μ sub-columns for accuracy present means of results from tests on the three datasets not used for training, while in the σ one standard deviations can be found.

As it can be noticed, in this case revising the theory to make it compliant with the new logs is faster than learning it again from scratch, and the accuracy of the results is higher.

3.9.2 Auction Protocol

Let us now consider an interaction protocol among agents participating in an electronic auction (70).

The auction is sealed bid: the auctioneer communicates the bidders the opening of the auction, the bidders answer with bids over the good and then the auctioneer communicates the

bidders whether they have won or lost the auction.

The protocol is described by the following ICs (71).

$$\begin{aligned} & bid(B, A, Quote, TBid) \\ \rightarrow & \exists(openauction(A, B, TEnd, TDL, TOpen), \\ & TOpen < TBid, TBid < TEnd) \end{aligned} \tag{3.4}$$

This IC states that if a bidder sends the auctioneer a *bid*, then there must have been an *openauction* message sent before by the auctioneer and such that the bid has arrived in time (before $TEnd$).

$$\begin{aligned} & openauction(A, B, TEnd, TDL, TOpen), \\ & bid(B, A, Quote, TBid), \\ & TOpen < TBid \\ \rightarrow & \exists(answer(A, B, lose, Quote, TLose), \\ & TLose < TDL, TEnd < TLose) \\ \vee & \exists(answer(A, B, win, Quote, TWin), \\ & TWin < TDL, TEnd < TWin) \end{aligned} \tag{3.5}$$

This IC states that if there is an *openauction* and a valid *bid*, then the auctioneer must answer with either *win* or *lose* after the end of the bidding time ($TEnd$) and before the deadline (TDL).

$$\begin{aligned} & answer(A, B, win, Quote, TWin) \\ \rightarrow & \forall\neg(answer(A, B, lose, Quote, TLose), TWin < TLose) \end{aligned} \tag{3.6}$$

$$\begin{aligned} & answer(A, B, lose, Quote, TLose) \\ \rightarrow & \forall\neg(answer(A, B, win, Quote, TWin), TLose < TWin) \end{aligned} \tag{3.7}$$

These two ICs state that the auctioneer can not answer both *win* and *lose* to the same bidder.

A graphical representation of the protocol is shown in Figure 3.5.

The traces have been generated in the following way: the first message is always *openauction*, the following messages are generated randomly between *bid* and *answer*. For *answer* messages, values *win* and *lose* are selected randomly with equal probability. The times are selected randomly from 2 to 10. Once a trace is generated, it is tested with the above ICs. If the trace satisfies all the ICs it is added to the set of positive traces, otherwise it is added to the set of negative traces. This process is repeated until 500 positive and 500 negative traces are generated for length 3, 4, 5 and 6.

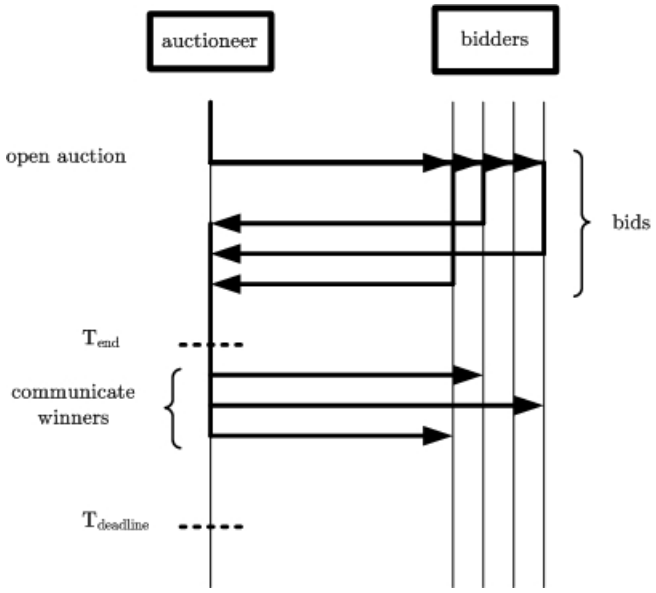

Figure 3.5: Sealed bid auction protocol.

We then considered 500 randomly selected traces (half positive and half negative). We applied both DPML and IDPML to this dataset, the latter starting with a version of the model that was manually modified to simulate an imperfect theory written down by a user.

The results in Table 3.3 confirm those of Table 3.2: revision offers benefits both in time and accuracy.

3.10 Related work

3.10.1 Semantic Web Services

One of the first works about the ontological representation of knowledge in a logic programming context is F-Logic (72) (implemented in Flora-2 (73)). F-Logic ontologies are similar to the formalism of frames and provide an attribute-value and taxonomic (with inheritance) structure for objects and classes. However they lack the feature of class definition based on restriction on attributes (or properties), which is a peculiar characteristic of DL (and OWL). Even if the Semantic Web community adopted the DL paradigm for the ontology layer of the Semantic Web cake, the urge for introducing rules for improving expressivity, for instance to model Semantic Web Services, partially made F-Logic come back as a suitable candidate, like in the case of the WSMO (Web Services Modeling Ontology (74)) framework. An extensive study of how rules and ontologies can be integrated, with a specific focus on Semantic Web, can be found in (75) where a language, WSML, is proposed to be used as a Web Service Mod-

	revision			full dataset		
dataset	time	accuracy		time	accuracy	
		μ	σ		μ	σ
1	820	0.962812	0.0043	1373	0.921687	0.0043
2	1222	0.962937	0.0043	1403	0.939625	0.0041
3	806	0.96375	0.0039	1368	0.923312	0.0044
4	698	0.961125	0.0018	1618	0.937375	0.0020
5	743	0.963875	0.0038	1369	0.92350	0.0042

Table 3.3: Revision compared to learning from dataset for the auction scenario

eling Language in WSMO. Kifer et al. in (76) propose a comprehensive solution based on Flora-2, in which F-logic is used to express ontologies and Transaction Logic is used to model declaratively how services behave.

On the other hand, work is being done for the foundations of a tight integration of Description Logic and Logic Programming and one of the most important issues is how to deal with the different assumptions (Closed World for LP, Open World for DL) made by these two families of languages. In (77) decidability and complexity results are presented for an integration of DL and Disjunctive Datalog, while in (78) an integration of DL and LP is based on the Minimal Knowledge and Negation as Failure (MKNF) logic (79). In (80) this proposal is extended providing a three-valued semantics. In our work, we apply the closed world assumption to ontological predicates, when computing their truth value in a SCIFF derivation; this simplification has proved acceptable for the applications that we have considered.

3.10.2 Learning and Updating Policies

Process mining is an active research field. Notable works in such a field are (61, 62, 64, 65, 81, 82).

In particular in (81, 82) (partially) declarative specifications (thus closer to our work) are adopted.

In (82) activities in business process are seen as planning operators with pre-conditions and post-conditions. In order to explicitly express them, *fluents* besides activities (i.e., properties of the world that may change their truth value during the execution of the process) have to be specified. A plan for achieving the business goal is generated and presented to the user which has to specify whether each activity of the plan can be executed. In this way the system collects positive and negative examples of activities executions that are then used in a learning phase. Our work remains in the traditional domain of BPM in which the pre-conditions and

post-conditions of activities are left implicit.

In (81) sets of process traces, represented by Petri nets, are described by high level process *runs*. Mining then performs a merge of runs regarding the same process and the outcome is a model which may contain sets of activities that must be executed, but for which no specific order is required. However, runs are are already very informative of the process model; in our work, instead, mining starts from traces, which are simply a sequence of events representing activity executions.

In (83) events used as negative examples are automatically generated in order to partially take away from the user the burden of having to classify activities. We are interested in the future to investigate automatic generation of negative traces.

A useful survey about theory revision methods, including a brief description of common refinement operators and search strategies, can be found in (84). With regard to the taxonomy proposed there, we deal with proper revision (not restructuring) both under the specializing and generalizing points of view.

In (85), the authors address structural process changes at run-time, once a process is implemented, in the context of adaptive process-aware information systems. Basically, adaptive systems are based on loosely specified models able to deal with uncertainty, i.e. able to be revised to cover unseen positive examples. The implemented process must be able to react to exceptions, i.e. it must be revised to rule-out unseen negative examples. Both kinds of revision must guarantee that compliant traces with the previous model are still compliant with the revised one. In (85), the authors consider process models expressed as Petri nets, where structural adaptation is based on high-level change patterns, previously defined in (86). They review structural and behavioral correctness criteria needed to ensure the compliance of process instances to the changed schema. Then, they show how and under which criteria it is possible to support dynamic changes in the ADEPT2 system, also guaranteeing compliance. Similarly to them, we address the problem of updating a (declarative and rule-based, in our case) process model while preserving the compliance of process instances to the changed model. This is guaranteed, however, not by identifying correctness criteria, but rather by the theory revision algorithm itself. We think that their approach is promising, and subject for future work, in order to identify both change patterns to be considered (up to now we consider one generalization and one refinement operator only) and correctness criteria under which the revision algorithm might be notably improved.

4

Biomass Plant Placement with Energy-Effective Supply

4.1 Introduction

Although the end of fossil fuels seems postponed to a midterm future, the search for new energy sources is recently having a new boost: after decades of warnings from the environmental experts, citizens and governments start to realize that we cannot continue polluting our planet indefinitely. Carbon oxides are known to increase the greenhouse effect, widely recognized as the main responsible for climate change. As former OPEC minister Ahmed Zaki Yamani said, *"The Stone Age did not come to an end because we had a lack of stones, and the oil age will not come to an end because we have a lack of oil"*.

In the search for reducing carbon oxides emissions, biomass-powered power plants are very promising, because they provide energy with an almost carbon neutral process since biomass mostly comes from trees and vegetables that during their life converted carbon dioxide to oxygen.

Burning them returns part of the energy they got from sunlight: it can be considered a kind of renewable type of energy too.

Also, for countries that mainly rely on imported energy, biomass power may mean an economy less dependant on the price of oil.

For these reasons, energy from biomass is currently receiving substantial governmental funding. On the other hand, unfortunately, building a biomass plant does not necessarily imply any improved sustainability, as the plant is inserted into an environment, with complex interrelations, including production of fumes, the need for refrigeration, transport of the biomass from the production sites to the power plant, and so on. Without taking into consideration such

aspects, the project is quickly doomed to failure.

Even more importantly, there are economic aspects that rule the whole life of the biomass plant. For example, in Italy energy producers that use renewable sources have the right to sell energy to the energy provider at a higher price than the market price (87). Since energy selling price is higher than the market price, the plant manager can afford to buy biomasses at higher prices than the market price. This helps the development of new renewable power plants, but on the other hand may lure the manager to produce energy with low-cost fuel, like palm-tree oil imported from overseas. Unfortunately, the energy needed to transport the fuel overseas is much higher than the energy actually produced by burning it in the biomass plant. Nevertheless, it gives good revenue to the plant manager. However, even excluding these extreme situations, it may actually happen that the transport of the biomass to the plant consumes more energy than that produced in the plant. Again, the reason is economic: the producers of biomass may be lured by the high revenues to carry biomass from far-away production sites, or with highly polluting and/or energy demanding trucks.

One of the issues is the combinatorial nature of the problem. A good location of the biomass plant is of vital importance for the economic survival of the project, and it includes solving complex location and transportation problems. Many works in the literature deal with this aspect (88, 89, 90).

The second issue is the sustainability of the project, and this is affected by more subtle aspects. In this chapter, we explicitly address the biomass plant location problem with the aim of maximizing the sustainability, starting from real world data gathered in the Emilia-Romagna region of Italy.

The rest of the chapter is organized as follows. We first define the problem, and show the collected data from the Emilia-Romagna region. Then, in Section 4.3, we define a CLP($\mathcal{R}$) (see Section 2.4.6) model that optimally locates one or more biomass power plants in a devised area of the region, optimizing in particular the ecological sustainability of the project. In Section 4.4 we study the complexity of the problem. We show the results obtained from the CLP($\mathcal{R}$) model in Section 4.5. We discuss how this contribution relates to already existent work in the area in Section 4.6.

4.2 Problem description

Biomass power plants may be aimed at obtaining energy from a variety of different fuels: from garbage, to forest/sawmill residues, to manure (91). In this work, we address the problem of the optimal location of a biomass plant that uses wood obtained from forests. This does not mean deforestation, as wood can come from dead trees, branches, tree stumps, etc., as well as

Sensitivity theme	Wind powered generators	Biomass plants	Hydroelectric plants
Vulnerable waters	yes	maybe	maybe
Airports	no	maybe	maybe
Archaeology sites	maybe	maybe	maybe
Military regions	no	no	no
Unstable terrain	no	no	no
Rivers	no	no	yes
Natural parks	no	no	yes
Water wells	yes	no	maybe
...			

Table 4.1: Excerpt from the interference matrix *"sensitivity themes vs. power plants"* in the Emilia-Romagna region

from *selection cutting*, that is the practice of removing mature timber to improve the timber stand. In sustainable forest management, a selection cutting criterion requires to cut less than the yearly produced biomass.

Before building a power plant, one should get the required authorisations from the regional bureaus. As stated earlier, this includes an evaluation of the environmental impact of the power plant. Various locations are not acceptable for a power plant, due to various reasons that come from geological information, regulations, statistics on pollution, types of crops in the surrounding fields, etc. Other locations could be acceptable but raise some issues that need further investigation before the project can be accepted.

To define disallowed areas, and areas requiring further investigations, the environmental experts of the Emilia-Romagna region reported into a table the possible interferences between various types of power plants with *sensitivity themes*, i.e., types of areas that may be sensitive to such a plant. Table 4.1 is an excerpt of such a table.

For each theme, the required information are gathered into a map, given as a shapefile of a Geographical Information System (GIS). Shapefiles can include points, line segments, or polygons representing areas with common characteristics. Through a GIS software, one can overlap shapefiles as layers of a map. In this way, by overlapping all the maps of themes incompatible with some power plant type, the map of incompatibility for a given plant type can be generated: it is the set of areas in which there is at least one layer (theme) that forbids such plant (see Figure 4.1). From this process a set of feasible locations is identified.

We consider the problem of locating a plant that uses biomass obtained from forests. Biomass is then transported to the power plant; we suppose that the transport is through roads.

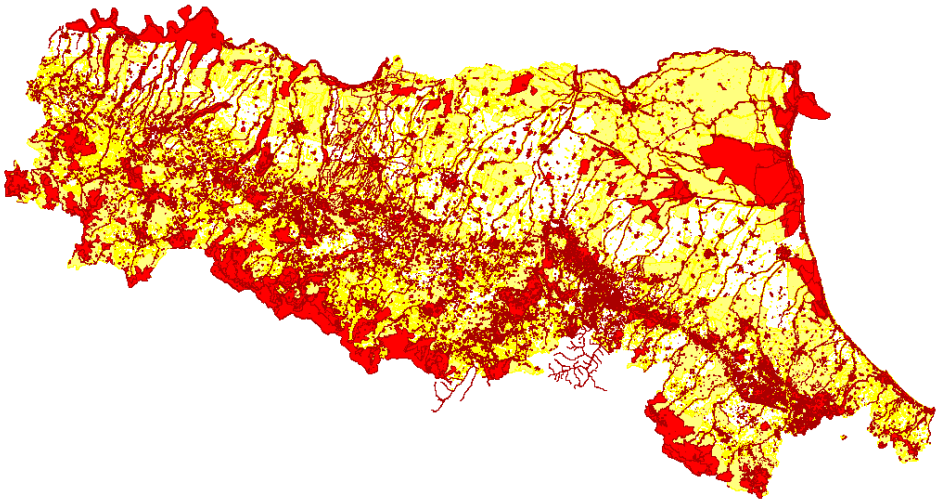

Figure 4.1: Map of incompatibility for biomass power plants in the Emilia-Romagna region.

The regional agency kindly provided us with the GIS map of roads as well as the shapefile of forests. The shapefile of roads is a set of line segments: each road segment is a straight line segment connecting two points, given with longitude and latitude. A road is then a sequence of segments; the length of each segment is between 6 and 20 metres.

We built a graph containing as nodes all points from the roads' shapefile. Two nodes are connected if there is a straight road segment connecting the two.

We assume that the biomass plants will be built in close proximity to some road, and that biomass is collected near a road, to allow for a convenient transportation of biomass material from the collection point to the plant. In our model, only the nodes of the obtained graph are possible biomass plant locations. On the other hand, there is no point in selecting the actual positioning with a precision of 20 metres, so we use a sampling of the nodes based on proximity.

In the graph, each node is labelled as *red* if the node is internal in a polygon incompatible with biomass plants, it is *green* if it is a biomass collection point, it is *white* otherwise. Each arc is labelled with its length. Thus, candidate positions for the power plants are the white nodes, biomass is collected at green nodes, and transported to the power plants by following paths in the graph.

In the following, we propose a CLP($\mathcal{R}$)model to address the plant location problem.

4.3 A CLP($\mathcal{R}$) Model

The CLP($\mathcal{R}$) model used to address the given problem relies on a problem representation given as a graph. Starting from the general graph described earlier, we produce a smaller bipartite graph, that connects possible locations of the biomass power plants to the location of the forests. Each arc is labelled with the shortest path length between the two nodes; the (shortest path) distance between two nodes i and j is called d_{ij}, and it is a parameter (it is a known constant).

We have an array of unknowns P that represents the possible locations of the plants: the element p_i can be 0 or 1; it is 1 iff we build a power plant in node i. We suppose we want to build in the devised region a number of plants which belongs to the fixed range Min^p to Max^p ; we ensure this fact through the constraint

$$Min^p \leq \sum_{i=1}^{N^P} p_i \leq Max^p. \tag{4.1}$$

We also have N^F forest nodes, i.e., nodes from which the owners can collect wood that can be used to produce energy in the biomass plant. As said earlier, wood can come from forest residues (such as dead trees, branches, tree stumps, etc.) or from *selection cutting*. Each forest node i has a production of wood $prod_i$ that cannot be exceeded, because the quantity of wood removed from a region should not exceed the yearly produced biomass, in order to have a sustainable process. We have a matrix of unknowns C_{ij} that links power plant nodes with forest nodes. Element C_{ij} gives the quantity of wood that is provided from forest node i to plant node j. The constraints include that the total quantity of wood provided by a forest node i does not exceed its carry capacity:

$$\forall i \in \{1..N^F\} \qquad \sum_{j=1}^{N^P} C_{ij} \leq prod_i.$$

Each power plant has a minimal quantity of wood $mindemand$ it must receive in order to be operative; otherwise there is no point in installing such a plant

$$\forall j \in \{1..N^P\} \qquad \sum_{i=1}^{N^F} C_{ij} \geq p_j \cdot mindemand.$$

Symmetrically, there is a maximum quantity of wood a power plant can accept:

$$\forall j \in \{1..N^P\}, \qquad \sum_{i=1}^{N^F} C_{ij} \leq p_j \cdot maxdemand.$$

Note that this constraint also imposes that one cannot provide wood to a plant that is not installed.

Finally, we have economic parameters and sustainability parameters.

We assume that owners can collect residual wood from the nearby forests, and will transport them to one plant (usually the closest one), in exchange for money. Now, if the revenue is lower than the supply cost, then there will be no deal, so we need to take into account two important parameters: the unit price paid for the wood, $Price^{wood}$, and the unit supply cost. The supply cost for the forest owner includes a collection cost and a transport cost. We assume the collection cost is proportional to the quantity of collected wood (although it could depend on many factors, like slope and extension of the wood). For the transport cost, we consider a parameter $Cost^{trans}$, that represents the cost of transporting one unit of wood for a distance of one unit. So, we have that plant j can receive wood from forest node i only if it is advantageous under the point of view of profit to collect and transport wood from i to j. The transport cost is proportional to the quantity and to the distance through constant $Cost^{trans}$, i.e. the cost to transport a quantity C_{ij} from node i to node j is

$$d_{ij} \cdot Cost^{trans} \cdot C_{ij}, \tag{4.2}$$

so the supply cost becomes

$$Cost^{collect} \cdot C_{ij} + d_{ij} \cdot Cost^{trans} \cdot C_{ij}. \tag{4.3}$$

The revenue for the seller of wood is proportional to the quantity, so if she provides C_{ij} units of wood, she will get a revenue of

$$Price^{wood} \cdot C_{ij} \tag{4.4}$$

from which she expects some minimum profit

$$Profit^{wood} \cdot C_{ij}, \tag{4.5}$$

for example, we can have that $Profit^{wood} = 0.2 \cdot Price^{wood}$: it is a constant parameter explaining that the seller will not sell the biomass if her profit is below 20% the total price of the biomass. Combining equations (4.3), (4.4) and (4.5) we get that the net revenue is acceptable iff

$$Cost^{collect} + d_{ij} \cdot Cost^{trans} + Profit^{wood} \leq Price^{wood}$$

is satisfied. If it is not, then we impose that $C_{ij} = 0$.

From a sustainability viewpoint, the aim is that the whole system generated by the new plant produces renewable energy (at the net of consumed energy). The system contains the plant, as well as the forest owners collecting wood and transporting it to the plant, so we must ensure that the energy spent for the transport and for building the plant does not overpass that produced by the wood in the plant.

The wood transport is usually performed by means of some vehicle of the wood provider, which can be efficient or not. We fix an average efficiency parameter, and call E^{fuel} the energy provided by the fuel necessary to move one unit of wood for one distance unit. On the other hand, we have that one unit of wood provides energy E^{wood}. Also, we have to consider that building a plant requires itself an energy investment, and such aspect should be taken into account in the energy balance: we will consider this contribute in a term γ, that will be detailed in the following. Thus the objective function considering transportation and sustainability issues is

$$max(f) = \sum_{j=1}^{N^P} \sum_{i=1}^{N^F} C_{ij} \left(E^{wood} - d_{ij} E^{fuel} \right) - \gamma. \tag{4.6}$$

The energy required to build a new plant depends mainly on its location and its size. The contribution for the location can be considered as a coefficient α_i associated to each node $i \in \{1..N^P\}$ in the graph that is candidate to host a plant. The size of the plant can be given, e.g., in terms of its input power. In our case, the power the plant gets as input is the energy of biomass it receives divided by the time the plant is operative:

$$\mathcal{P}_j = \left(\sum_{i=1}^{N^F} C_{ij} \cdot E^{wood} \right) / T^{op}.$$

The relation between the size of the plant and the energy required to build it, as reported in the documents of the European Commission (92), is given by

$$\mathcal{E} = \mathcal{E}^0 \cdot \left(\frac{\mathcal{P}}{\mathcal{P}^0} \right)^s$$

where $\mathcal{E}^0$ is the energy required to build a plant of reference size $\mathcal{P}^0$ (in some location with $\alpha = 1$). The scale factor s goes usually from 0.5 to 1. Clearly, in the extreme case of $s = 1$ we have a linear relation: building two plants of 1MW or one plant of 2MW has exactly the same energy requirement[1]; instead, when $s < 1$ we can exploit economy of scale for the plant. So, knowing the energy $\mathcal{E}^0$ required to build a plant of size $\mathcal{P}^0$, the energy required to build a plant in node j is given by the non-linear relation

$$\mathcal{E}_j = \alpha_j \mathcal{E}^0 \left(\frac{\mathcal{P}_j}{\mathcal{P}^0} \right)^s = \alpha_j \mathcal{E}^0 \left(\frac{\sum_{i=1}^{N^F} C_{ij} \cdot E^{wood}}{\mathcal{P}^0 T^{op}} \right)^s. \tag{4.7}$$

Equation (4.7) provides the total energy required to build a new plant in node j. Such energy should be considered in the objective function, as we want the whole system to produce energy; however, the other terms in Equation (4.6) represent the produced energy and the cost

[1] Provided that the locations are comparable, i.e., they have the same coefficient α.

of transport *per year*, so to add up the two terms we need to amortize the total energy $\mathcal{E}_j$ during the lifespan of the plant. We simply divide $\mathcal{E}_j$ for the average number of years a plant is productive, T_j^{life}, and obtain as new objective function:

$$max(f) = \sum_{j=1}^{N^P} \left(\sum_{i=1}^{N^F} C_{ij}(E^{wood} - d_{ij}E^{fuel}) - p_j \frac{\alpha_j \mathcal{E}^0}{T_j^{life}} \left(\frac{\sum_{i=1}^{N^F} C_{ij} \cdot E^{wood}}{\mathcal{P}^0 T^{op}} \right)^s \right) \quad (4.8)$$

Such objective function is nonlinear and, since the exponent s is at most 1, the term in Equation (4.7) is concave (whenever $s < 1$).

In order to fit the nonlinear function in the CLP($\mathcal{R}$) model, we approximate it with a piecewise linear function.

Given a (possibly nonlinear) function $y = g(x)$, we sample the curve g in k points $x_1, \ldots, x_k$; the piecewise linear approximation $y' = g'(x) \simeq g(x)$ is defined as

$$\begin{aligned} x &= \textstyle\sum_{i=1}^{k} \lambda_i \cdot x_i \\ y' &= \textstyle\sum_{i=1}^{k} \lambda_i \cdot y_i \end{aligned}$$

where $\lambda_i \in [0..1]$ are new continuous variables subject to the constraints

$$\sum_{i=1}^{k} \lambda_i = 1$$

At most two of the λ_i can be nonzero

The two nonzero λ_i must be adjacent

Clearly the last two conditions are not linear; they can be stated in a CLP($\mathcal{R}$) model introducing new integer 0-1 variables, but there exists a more efficient option. In some CLP($\mathcal{R}$) solvers, one can declare $(\lambda_1, \ldots, \lambda_k)$ as a Special Order Set of type 2 (SOS2) (93), and the solver will exploit this information for efficient branching heuristics.

It should be noticed that, for the sake of sustainability, the problem cannot be just formulated under the point of view of economics, as the market prices of biomass and fuel needed for transportation do not reflect the respective energy potential. This is caused by a number of reasons, the most relevant of which is that institutions such as the European Union or the regional government issue incentives for the use of the former (e.g. trying to push entrepreneurs towards renewable energy). For this reason it is possible that the economic net revenue for the plant is positive even if the whole system, including the transport chain and all the rest which is needed in order to make the plant work, actually *consumes* energy at the net.

4.4 Complexity

We show that the problem addressed in this chapter is NP-hard, even in the case in which there are no restrictions on the plants' capacity[1].

Theorem 2 *The uncapacitated biomass plant location problem is NP-hard.*

We prove NP-hardness by reduction from the Facility Location Problem.

A Facility Location Problem is defined as the 4-tuple $\langle D, F, f, d \rangle$, where

- D is a set of clients
- F is a set of potential facility locations
- d is a function $d : D \times F \rightarrow R^+$ representing the distance between clients and possible facility locations
- f is a function $f : F \rightarrow R^+$ representing the cost of opening a new facility

The objective is to find the set $S \subseteq F$ that minimizes

$$\sum_{i \in S} f_i + \sum_{j \in D} (min_{i \in S} d_{ij}) \tag{4.9}$$

Given an instance of the facility location, we build the following biomass plant location problem:

- the set of forest nodes is equal to the set of clients D
- the possible positions of the biomass plant are the potential facility locations
- the energy required to build a plant is independent of the size of the plant ($s = 0$)
- the cost of opening a new biomass plant coincides with function f $\left(f_i = \frac{\alpha_j \mathcal{E}^0}{T_j^{life}} \right)$
- the distance between forest and plant nodes coincides with the distance function d
- the number of plants is not constrained (Min^p=0 and $Max^p = |F|$)
- all forest nodes produce a unit of biomass ($\forall i \in \{1..N^F\}, prod_i = 1$)
- $Cost^{collect} = 0$, $Cost^{trans} = 0$, $Price^{wood}$ is any number greater than zero

[1]Thanks to Alberto Caprara for his suggestions on the proof of NP-hardness.

- E^{wood} is a sufficiently high number ($E^{wood} > E^{fuel} \cdot max_{i=1..N^F, j=1..N^P}(d_{ij})$)
- the biomass plant does not have (constraining) minimum and maximum capacity ($mindemand = 0$, $maxdemand = \sum_{i=1}^{N^F} prod_i$)

Now, we decompose the objective function (4.8) into the terms:

$$max(f) = \sum_{j=1}^{N^P} f_j \cdot p_j + \sum_{i=1}^{N^F}\sum_{j=1}^{N^P} C_{ij}E^{wood} - \sum_{i=1}^{N^F}\sum_{j=1}^{N^P} C_{ij}d_{ij}E^{fuel}$$

The first term is the same as in Equation (4.9). From the definition of E^{wood}, we have that the coefficient $(E^{wood} - d_{ij}E^{fuel})$ in the objective function (4.8) is always positive. Since there are no limits on the capacity of a plant, all the available biomass can be collected, so in an optimal solution of the biomass plant placement problem the second term is the constant $\sum_{i=1}^{N^F}\sum_{j=1}^{N^P} C_{ij}E^{wood} = \sum_{i=1}^{N^F} prod_i$. From this observation descends immediately that the optimal solution of the biomass plant placement problem provides an optimal solution of the Facility Location Problem, if every forest node provides biomass only to one plant.

Let us suppose that, in the optimal solution of the biomass plant placement problem, one forest node i provides biomass to two plants, let us call them j_1 and j_2. In this case, the two plants must be at the same distance from node i ($d_{ij_1} = d_{ij_2}$), because otherwise all the biomass could be carried to the closest facility (as we do not have capacity constraints), providing a better value of the objective function (which contradicts the fact that such solution is optimal). Since the two plants are at the same distance, we can move all the available biomass to one of them, say j_1, without changing the value of the objective function. We can apply this method to all forest nodes that serve more than one plant and obtain an optimal solution of the Facility Location Problem.

4.5 Experimental results

We experimented our CLP($\mathcal{R}$) model on two subregions of the Emilia-Romagna region, delimited by rectangles in Figure 4.2.

In Figures 4.3 and 4.4 the detailed views are shown of, respectively, the western and eastern areas, with the available roads (the main roads are thicker), the forest areas (solid filled surfaces) and areas where biomass plant location is impossible (hatched).

The two areas were selected with different characteristics, in order to study our approach in different scenarios. The western area has large availability of forests, and it is wider than the eastern area; this area could support the presence of many power plants, which can make the problem difficult to solve computationally. The eastern area is more restricted, and it contains

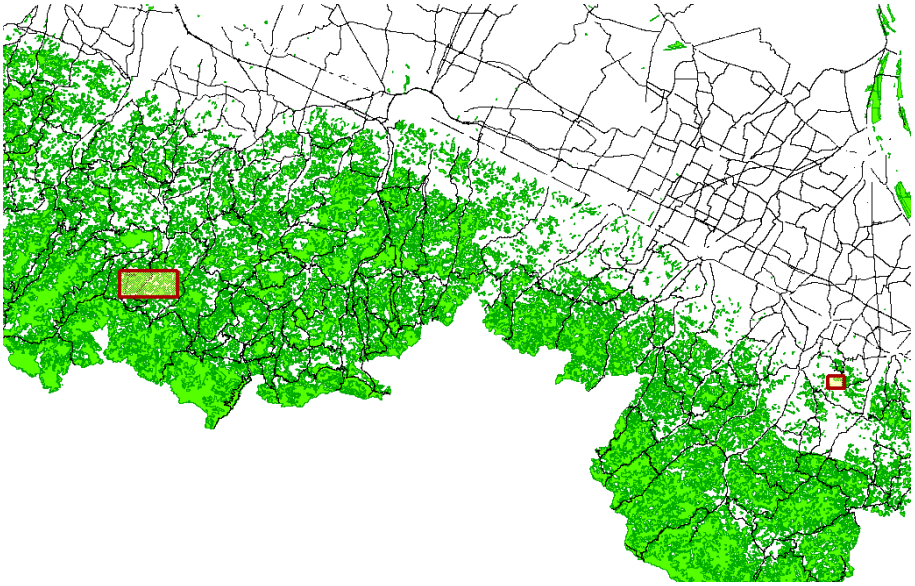

Figure 4.2: The areas used in our tests

less forest nodes, so in this case transport issues are particularly relevant, as biomass may have to be collected from many dispersed forests.

We ran a series of experiments on those areas.

In Figures 4.3 and 4.4 results are shown of optimal placement without considering energy cost of plant construction. The squares are the optimal placements of plants found by the model and the different sizes show the associated biomass demand and energy production. In Figure 4.4 the spot on one of the forest areas shows, for example, one of the biomass supply points associated to the linked plant in the optimal provisioning plan, while the cross shows the placement of one plant as prescribed by the solution obtained choosing as the objective function the maximization of profit rather than the energy balance.

In both areas the goal was to place a number of plants not greater than 5 and whose production had to be more than 0.2 MW but less than 1 MW (thus adopting the point of view of distributed power generation).

The considered values for energy parameters are $E^{wood} = 15000MJ/tonne$ for biomass and $E^{fuel} = 7MJ/km$ for fuel (we are considering fuel consumption being 0.1 lt/km for a load of one tonne). The market price for biomass has beed set to $Price^{wood} = 30euros/tonne$, and collection cost to $Cost^{collect} = 25euros/tonne$. We assume the biomass seller accepts as a minimum profit 15% of $Price^{wood}$ ($Profit^{wood} = 4.5euros$) for selling biomass and $Cost^{trans} = 0.2euros/km$. All these parameters are of course affected by some variability:

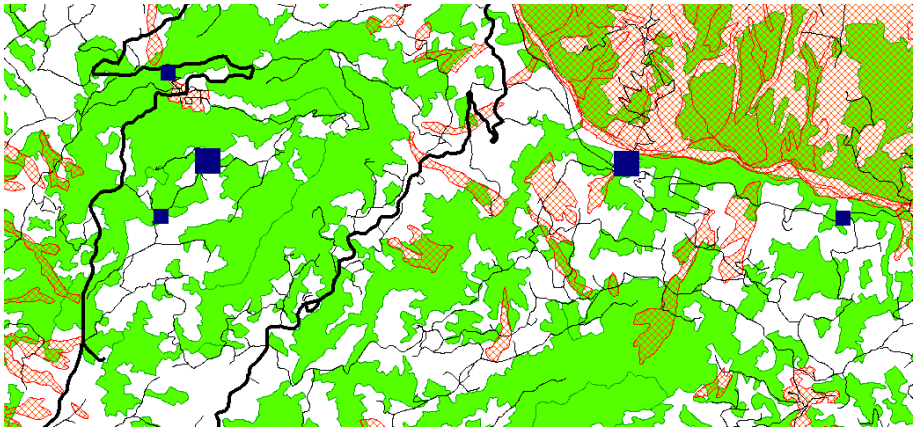

Figure 4.3: Detail of the western area. Squares mark optimal placements of plants without considering plant construction energy investment. Different sizes show the associated biomass demand and energy production.

different kinds of wood have different E^{wood}, fuel consumption and E^{fuel} depend on the means of transport (small vans, big trucks, tractors, etc.), the collection cost depends on the characteristics of the collection area, and the price of biomass and the expected profit follow market dynamics. All the values of the parameters were estimated by the environmental experts of the regional agency, based on their experience in the environmental impact evaluation and, whenever possible, on officially available data. For parameters with high variability they suggested to take the average value.

The western area is mapped on approximately 300 nodes, while the eastern counts approximately 100. Results were obtained in 10 minutes in the former case and 5 seconds in the latter one, using a 2 GHz Core 2 Duo processor.

The total net energy produced by the plants in the two optimal configurations is respectively 88500 GJ and and 16800 GJ for the western and eastern areas, while the total energy necessary for the transportation of the biomass to the power plant is, respectively 96133 MJ and 4707 MJ. In both cases, the net produced energy is positive, which shows that the optimal placement reached its objective. On the other hand, when choosing to maximize the profit in the eastern area, while the produced energy is the same, the energy necessary for the transportation of the biomass rises to 16793 MJ so while the energy balance keeps being positive, it is less favourable.

In Figures 4.5 and 4.6 results of placement are shown considering also the energy investment for plant construction. The environmental experts of the regional agency suggested that, in order to build a reference plant of $\mathcal{P}^0 = 0.5MW$ the required energy is $\mathcal{E}^0 = 45TJ$; the

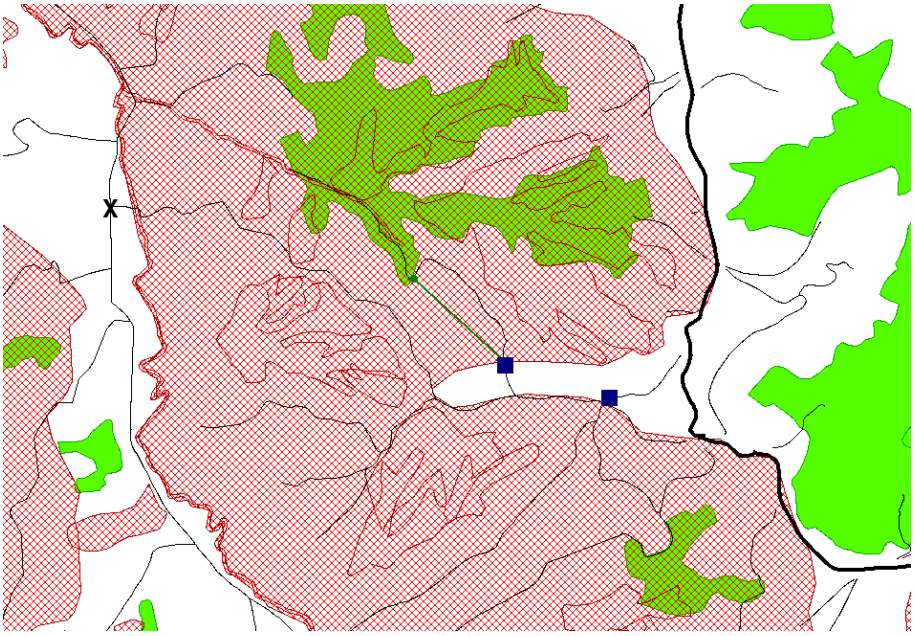

Figure 4.4: Detail of the eastern area. Squares mark optimal placements of plants without considering plant construction energy investment. Different sizes show the associated biomass demand and energy production. Forest area spot linked to plant shows one of the biomass supply points in the optimal provisioning plan. The cross shows the placement of one plant when the goal is maximizing profit.

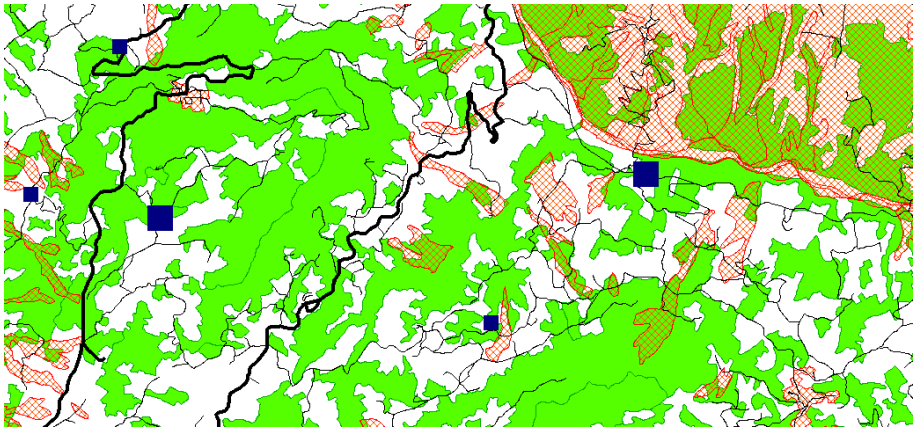

Figure 4.5: Detail of the western area. Squares mark optimal placements of plants without considering plant construction energy investment. Different sizes show the associated biomass demand and energy production.

exponent in Equation 4.7 has value $s = 0.8$, and the average lifetime of a plant is $T^{life} = 25$ years. The total net energy produced by the plants in the two optimal configurations is respectively 78995 GJ and and 14858 GJ for the western and eastern areas.

It is interesting to study the total net energy produced by the whole system varying the number of biomass power plants. In Figure 4.7, we show the results for the eastern area. After a sharp rise of the curve, due to the fact that only one plant cannot use all the available biomass, the curve decreases slowly, with an optimum number of plants equal to 2. This effect shows that there are opportunities given by economies of scale: the configuration with two plants (including a bigger one) is more rewarding than building 5 small ones. It is worth noting that with a coarser model, that does not consider the non-linearities in Equation (4.7), such effect would not appear.

However, it is worth noting that the production of energy, and, as a consequence, the sustainability of the defined choice, depends on the value of the parameters. As said earlier, we took average values provided by the experts for the transport cost $Cost^{trans}$, the energy generated by the fuel E^{fuel}, as well as the price and energy for biomass (respectively, $Price^{wood}$ and E^{wood}); however, we know that the price of fuel has high variability, that depends on the market, the rising demand of developing countries, the taxation level, etc. The energy of fuel is more stable, but still it depends on the type of fuel available in the devised area, and in the Emilia-Romagna region there are at least four types of widely distributed fuels (petrol, diesel fuel, methane, and propane). We wanted to assess the sustainability of the power plant and its provisioning system, and find out if, for some combinations of the parameters, the sustainabil-

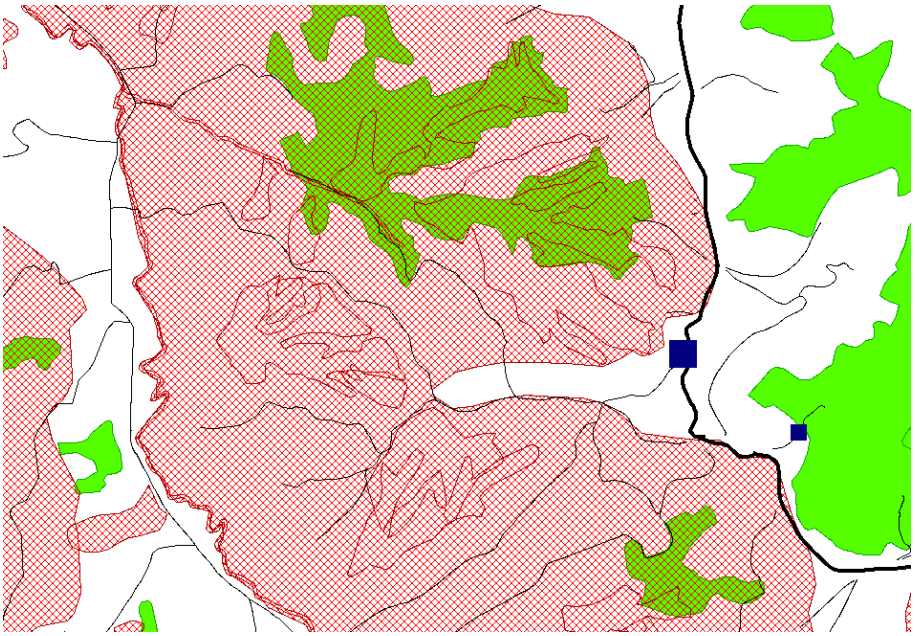

Figure 4.6: Detail of the eastern area. Squares mark optimal placements of plants considering plant construction energy investment. Different sizes show the associated biomass demand and energy production.

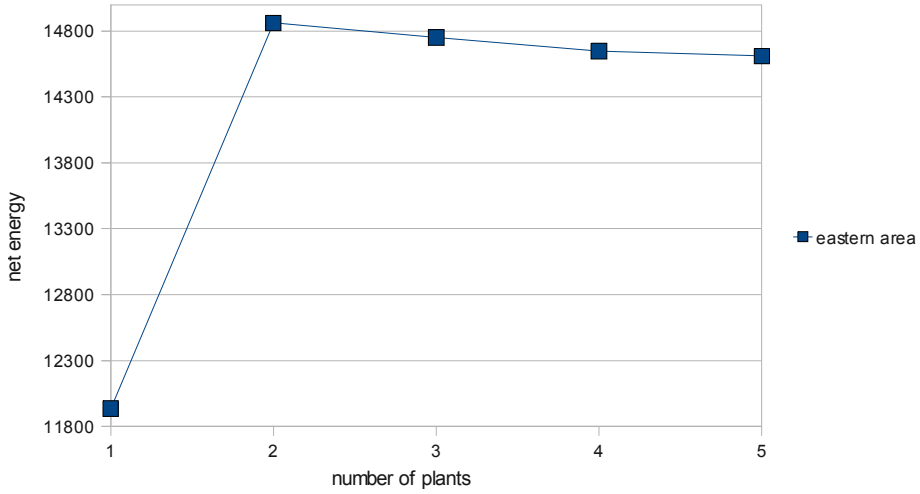

Figure 4.7: Net energy (GJ) produced in the eastern area varying the number of installed power plants.

ity of the system drops.

In Figure 4.8, we show the variation of the net energy production varying the other parameters, computed in the eastern area. It emerges that the important parameters are actually the ratios between the parameters. The biomass / fuel cost ratio expresses the relation between the biomass price the plant manager pays for one load of biomass ($Price^{wood}$), and the transport cost for one unit of length ($Cost^{trans}$). The biomass / fuel energy ratio expresses the relation between the energy potential contained in one load of biomass (E^{wood}) and the energy used to transport the biomass for one unit of length (E^{fuel}).

In the abscissa of the graph of Figure 4.8, we have the ratio $Price^{wood}/Cost^{trans}$. Higher prices for biomass allow forest owners to collect biomass not only from the very close woods but also from some more distant ones, thus influencing in a positive way the objective function at the beginning. In fact, Figure 4.8 shows that the net produced energy initially increases as the ratio $Price^{wood}/Cost^{trans}$ grows. However, at some point a saturation occurs, that can be due to various factors: for instance the limit on the total power production may be reached (maximum number of plants times the maximum energy produced by each plant).

On the z axis the ratio between energies E^{wood}/E^{fuel} is plotted; we have fixed the energy produced by biomass E^{wood} and varied the E^{fuel} energy necessary to transport a unit of biomass for a unit of distance. In this way, we can take into account both different types of fuel, and different types of transport media (e.g., small or big trucks) with varying efficiency (e.g., with old or new engine types). As expected, reducing the energy required for transportation increases the net produced energy up to a saturation and, vice-versa, the increase in the amount of energy used to transport one load of biomass causes a decrease in the maximum net energy that can be produced. It is worth noting that there is a threshold in which the system, as a whole, starts *consuming* energy, instead of producing it. Note, however, that sometimes even if the net produced energy is negative, the net revenue for the various actors (the plant management and the forest owners) is not zero: the system consumes energy but still produces *money*. This is the worst possible situation from a sustainability viewpoint: the biomass plant pollutes the local region with fumes and, at the same time, it consumes energy.

But there is also an even worse risk, much more subtle: if the environmental assessment of the plant (and, most importantly, of the whole system) is done after building the plant, workers will become frustrated, and stakeholders will think that investing in the green economy is futile.

It is of utmost importance to identify these possible situations as early as possible, before the plants are built, and local authorities should intervene before the required authorisations are given.

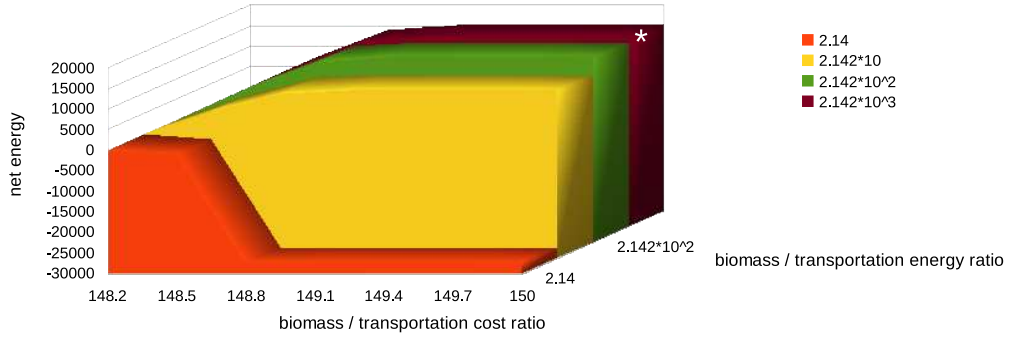

Figure 4.8: Net energy (GJ) as a function of (dimensionless) ratios of biomass and transportation related parameters. On x axis, ratio of cost of a biomass load and cost to move the load for one length unit. On z axis, ratio of energy of a biomass load and energy to move the load for one length unit. (*) indicates the parameter context of previous experiments.

4.6 Related work

The problem of biomass power plant location is widely investigated in the literature with the aim of defining optimal positioning with respect to economic aspects.

Bruglieri and Liberti (89) study the problem of running and planning a biomass-based production process. The aim is to optimize a process in which various plants process material, and exchange the products. For example, there can be a fermentation/distillation plant taking as input cane or beetroot and providing as output some type of alcohol, that can be input for another plant. In the optimization of running of the biomass-based process, the authors suppose a fixed amount of various output commodities is required, and minimize the operation costs (i.e., the cost of supplying plants with input commodities, transportation costs and processing costs). The authors also consider the planning of the production process, that includes deciding where and what type of processing plants one should install in various available locations. Our aim is more on the sustainability aspects rather than on the economic ones. Our model includes the energy required for building the power plant, that is approximated with a piecewise linear function of the plant capacity.

Reche-López et al. (90) consider the problem of placing a biomass power plant. They compare various metaheuristics (Simulated Annealing, Tabu Search, Genetic Algorithms and Particle Swarm Optimization) to find a near-optimal positioning of the biomass plant. They consider only economic factors to decide the biomass plant placement, i.e., their objective is to find the placement that gives maximum revenue. Another incomplete iterative GIS-based approach for the identification of candidate power plants is described in (94). It identifies

possible restrictions to the biomass power plant location and iteratively identifies potential locations along with the cultivated areas needed for the biomass collection. In this chapter, we adopted a complete approach, that provides a provably optimal solution, while Reche-López et al. compare metaheuristics, that may provide sub-optimal solutions.

Other approaches (95, 96) make use of a spread-sheet model to support decisions in the coppice harvesting, storage and transport and to model the costs related to the growing short rotation coppices in UK (coppicing is a method of woodland management which takes advantage of the fact that some trees reshoot from the stump or roots if cut down). These approaches converged in (97) in a comprehensive decision support system based on spread-sheet model for the bio-energy assessment.

An approach making use of a sophisticated MILP model is described in (88). Similarly to our approach a Geographic Information System integrated with an Integer Linear Programming model is used. Differently from our approach, the one presented in (88) relies on an objective function that estimates the economic costs-benefit balance related to the power plant; the benefit derives from the sale of the produced energy, while the costs include the cost of plant installation/maintenance, the transportation cost, the biomass harvesting cost, and the energy distribution cost. Clearly, energy efficiency is not considered while it is an essential aspect in sustainable biomass power plants. Related to the latter, also (98) makes use of a Mixed Integer Linear Programming model for the definition of an economic energy supply structure based on biomass. The model has been tested in the state of Brandeburg, Germany in (99). None of these works consider the (non-linear) function that relates the size of the plant with the energy required to build it, while we approximate it with a piecewise linear function.

Simulation and optimization are compared by De Mol et al. (100) on the biomass logistic-related problems. The aim is to define the location of the biomass plant along with biomass pre-treatment sites. The optimization model is divided into three components each associated to a type of biomass that are then combined in a knapsack model. The objective function is again the minimization of costs of biomass flow (variable costs) and those related to pre-treatment (fixed costs). Again, energy efficiency is not considered, nor the non-linear relation defining the energy required for building the plant is taken into account.

As explained earlier in this chapter, it may be the case that the revenue is positive even if the system is not sustainable, and it actually consumes energy instead of producing it; for this reason, we adopted a sustainability viewpoint, that optimizes the net produced energy. We agree that economic factors should not be overlooked, and we explicitly considered the budget of the forest owners.

Another important contribution with respect to related literature concerns the wider aim of the overall decision support system. It is the first time, to our knowledge, that an effort is done

to collect in a unique decision support system all aspects of the strategic environmental assessment along with the corresponding optimization problems. Among these problems biomass location is an important cornerstone.

The formulation of our problem is related to the Facility Location Problem, for which there exists a wide literature on the basic models and algorithms (101, 102, 103, 104, 105), as well as surveys (106, 107, 108, 109, 110) and books (111, 112, 113, 114).

One of the differences in our model with respect to the classical facility location problem stands in the fact that not all clients must be served. Charikar et al. (115) study various extensions of the facility location in which a small number of clients can be denied service, in order to obtain a reduction of the total cost. In particular, when there is a small number of *outliers*, the cost reduction can be significant. The authors propose various extensions, that can be grouped into two main categories. The *robust facility location* has a fixed parameter p that represents the number of clients that must be serviced; the classical facility location problem can be obtained by setting p equal to the number n of clients. The *facility location with penalties* associates each client j with a penalty p_j: for each client that does not receive the service, the objective function gets the corresponding penalty. Again, the classical facility location problem can be obtained by setting all penalties to ∞. The authors propose polynomial approximation algorithms for these cases.

Brimberg and ReVelle (116) address the problem as a biobjective problem: the first objective is minimizing the costs, while the second is minimizing the unsatisfied demand. They find the efficient frontier with the weighting method, i.e., they optimize a weighted sum of the two objectives, and then vary the weights to find all the frontier.

In our formulation, if a forest node j does not provide biomass to any plant, the objective function implicitly incurs in a penalty corresponding to its production capabilities $prod_j \cdot E^{wood}$, so our problem could be cast as a facility location with penalties (if the opening cost of each plant was independent of the plant size and there were no upper/lower limits on the plant size).

Holmberg and Lin (117) consider a variant of the facility location problem in which the cost of a facility depends on the size of the plant through a staircase function; they also propose heuristics based on Lagrangian relaxation. Wollenweber (118) proposes a number of further extensions to such model and proposes a heuristic approach. Instead of the staircase, we used a continuous piecewise linear approximation of the energetic cost of opening a biomass plant; moreover, our model does not require all clients to be served.

5

Aqueduct Valve Placement for Minimal Service Disruption

5.1 Introduction

Water is one of the fundamental human needs and, even if it is too often taken for granted in Western societies, it arguably is one of mankind's most precious and endangered resources. Using the words of the United Nations Water For Life commission, "Beyond meeting basic human needs, water supply and sanitation services, as well as water as a resource, are *critical to sustainable development*"(119).The same source highlights the critical position of urban settlements in this scenario, in particular among fast-growing, low-income populations: "the exploding urban population growth creates unprecedented challenges, among which provision for water and sanitation have been the most pressing and painfully felt when lacking. Two main challenges related to water are affecting the sustainability of human urban settlements: the lack of access to safe water and sanitation, and increasing water-related disasters such as floods and droughts. These problems have enormous consequences on human health and well-being, safety, the environment, economic growth and development".

The dedicated report prepared by the Center for Strategic and International Studies of Sandia National Laboratories (120) suggests that "*robust capacity building is essential*. Results achieved around the world by existing technical aid and infrastructure development programs can be vastly improved with greater efforts to support regional capacity building. These efforts should be aimed at regional education, political and economic innovation and technical expansion sufficient for long-term operation and maintenance by local, indigenous institutions. They must also include both technical and institutional capacity-building". Robustness is not just related to the quality of building materials, but is a general measure of the system aptness to

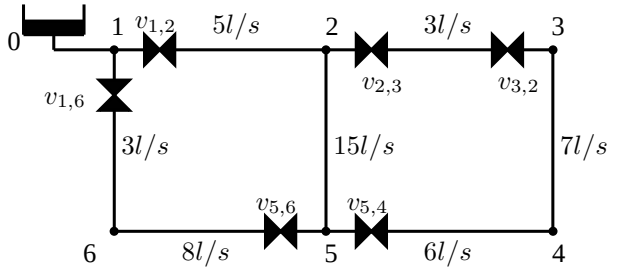

Figure 5.1: A schematic water distribution system with valves

limit undesired behaviour when something goes wrong.

An aqueduct is a complex system that includes a main component to transport water and a water distribution component, that brings the water to the users. The water distribution network can be thought of as a labelled graph, in which pipes are represented as undirected edges. In the network, there is at least one special node that is the source of the water (node 0 in Figure 5.1); users are then connected to the edges of the water distribution network. Each user has a demand (in litres per seconds) that is quantified by the hydraulic engineer through the available data. In particular, such a demand is frequently expressed as a daily average value. Each edge of the graph is labelled with the total demand of the users linked to it. For example, in Figure 5.1, the edge connecting nodes 2 and 5 (let us name it $e_{2,5}$) has a demand of $15l/s$ (that may be due, e.g., to five clients each requesting $3l/s$ on average).

When designing a water distribution network, one of the steps is designing the isolation system: in case a pipe has to be repaired (e.g., because of a break), a part of the network has to be disconnected from the rest of the network, in order to allow workers to fix the broken pipe. The isolation system consists of a set of *isolation valves*, that are placed in the pipes of the network. Once closed, the isolation valve blocks the flow of water through the valve itself. In common practice, a valve is usually placed in a pipe near one of the two endpoints; this means that in each pipe at most two valves can be placed. If in some pipe there are two valves, this means that this single pipe can be isolated by closing both the valves. In the example of Figure 5.1, the edge $e_{2,3}$ connecting nodes 2 and 3 has two valves, so in case this pipe is damaged, valves $v_{2,3}$ and $v_{3,2}$ will be closed, isolating only $e_{2,3}$.

However, placing two valves in each pipe is often not a viable option, because each valve has a cost; the cost is not only due to the manufacturing and physical placing of the valve, but also to the fact that the pipe is more fragile and deteriorates more quickly near valves. In case there are not two valves in each pipe (as it is usually the case in real distribution networks), the isolation of a pipe implies the closure of more than two valves and thus the isolation of more than one pipe. In this case, more users other than those connected to the broken pipe will

remain without service during pipe substitution. Suppose that the pipe $e_{3,4}$ connecting nodes 3 and 4 is damaged. In order to de-water it, workers have to close valves $v_{3,2}$ and $v_{5,4}$; as a result edge $e_{5,4}$ will be de-watered as well, and the clients that take water from it will have no service as well. Valves partition the network in the so-called *sectors*, that are, intuitively, those parts of the distribution network enclosed by some set of valves: edges $e_{3,4}$ and $e_{4,5}$ are in the same sector, so they cannot be de-watered independently one from the other.

The usual measure of the disruption in the service is the *undelivered demand*, i.e., the demand (in litres per second) that is not fulfilled during the repair operations; in the case there is need to de-water edge $e_{3,4}$, the disruption is the demand of the edges $e_{3,4}$ and $e_{4,5}$, i.e., $7 + 6 = 13l/s$. However, notice that the undelivered demand does not always coincide with the sector the damaged pipe belongs to. For example, pipe $e_{2,5}$ belongs to the sector consisting of the edges $e_{1,2}$ and $e_{2,5}$, that is surrounded by valves $v_{1,2}$, $v_{2,3}$, $v_{5,4}$, and $v_{5,6}$; however by closing these four valves, we will de-water a larger part of the network: edges $e_{2,3}, e_{3,4}$, and $e_{4,5}$ will be de-watered even though they are not in the same sector of the broken pipe. This effect is called *unintended isolation* (121).

The design of the isolation system consists of placing in the distribution network a given number of valves such that, in case of damage, the disruption is "minimal". Of course, the level of disruption depends on which pipe has to be fixed. In Figure 5.1 we have four sectors: if $e_{2,3}$ is damaged, the undelivered demand during repair is $3l/s$, if one of $\{e_{3,4}, e_{4,5}\}$ is broken, the undelivered demand is $13l/s$, if the broken pipe is $e_{1,6}$ or $e_{5,6}$ the undelivered demand is $3 + 8 = 11l/s$, while for sector $\{e_{1,2}, e_{2,5}\}$ the undelivered demand is $36l/s$, corresponding to the demand of $\{e_{1,2}, e_{2,5}, e_{2,3}, e_{3,4}, e_{4,5}\}$. A usual measure (122) is to take the worst case, and assign to the placement shown in Figure 5.1 (characterised by 6 valves) the effect of the maximal possible disruption: $36l/s$.

In (122) authors address the design of an isolation valve system as a two-objective problem: one objective is minimizing the number of valves in the isolation system, and the other is the minimization of the (maximum) undelivered demand. They adopt a genetic algorithm that is able to provide near-Pareto-optimal solutions, and apply it to the Apulian distribution network. The genetic algorithm provides good solutions in a very short time, but it is incomplete, so it does not provide, in general, Pareto-optimal solutions, but only solutions that are hopefully near to the Pareto front. The real optimal Pareto front remains unknown.

We believe that a complete search algorithm could provide better solutions, although at the cost of a higher computation time. Since the problem should be solved during the design of the valve system, there is no need to have a solution in real-time, and an algorithm providing a provably Pareto-optimal solution may be preferable with respect to incomplete algorithms, even with higher computation times.

In this chapter, we address the same two-objective problem studied by (122) as a sequence of single-objective ones; this is always possible when one of the objectives is integer (123, 124, 125). Given the number of valves, we model the design of the isolation valve system as a two-player game, and solve it with a minimax approach (126). As the game has an exponential number of moves, we reduce the search space by pruning redundant branches of the search tree, implementing the minimax algorithm in Constraint Logic Programming on Finite Domains (CLP(FD)) (see Section 2.4.6), in particular we used ECL^{i}PS^e (127). Our algorithm is complete, so it is able to find the optimal solutions and prove optimality; we show improvements on the best solutions known in the literature, up to 10% of the objective function value.

The rest of the chapter is organized as follows. In the next section, we give a formal description of the problem, then we propose the minimax interpretation in Section 5.3. We give a CLP(FD) model in Section 5.4, then we detail some implementation issues in Section 5.5 and we report experimental results in Section 5.6. We discuss related work in Section 5.7.

5.2 Problem description

A water distribution network is modelled as a weighted undirected graph $G \equiv (N, E)$, where $N = \{1, \ldots, n\}$ is a set of nodes and $E = \{e_{ij}\}$ is a set of edges. Each edge e_{ij} has an associated weight $w(e_{ij})$ called *demand*.

In the network, there are some nodes identified by the set Σ that are called *sources*.

Valves can be positioned near one of the ends of a pipe; we will refer to valve on edge e_{ij} near to node i as v_{ij}, while v_{ji} is a valve on the same edge, but close to node j.

Given a number N_v of valves to be positioned in the network, the objective is to position the valves in the network such that:

1. it is possible to isolate any pipe in the network. Formally, given an edge e_{ij}, it is possible to identify a set of valves C to be closed such that there is no path from any source node $s \in \Sigma$ to the edge e_{ij} that does not contain a valve $v \in C$. Since the set C of valves to be closed depends on the damaged pipe e_{ij}, we will also write $C(e_{ij})$. Note that there is only one reasonable set $C(e_{ij})$ of valves to be closed given a broken edge e_{ij}: intuitively only the valves directly reachable from e_{ij} will be closed. For example, in Figure 5.1 if the broken edge is $e_{3,4}$ then $C(e_{3,4}) = \{v_{3,2}, v_{5,4}\}$ and it does not make sense to close farther valves, such as $v_{2,3}$, because in order to reach $v_{2,3}$ from $e_{3,4}$ we have to overpass other valves ($v_{3,2}$).

2. the objective is to minimize the maximum undelivered demand (UD). Formally, let $D(C)$

be the set of edges that do not receive water when the valves in C are closed, i.e., those edges for which there is no path from any source node to the edge:

$$D(C) = \{e_{ij} \in E | \forall s \in \Sigma, \nexists Path(s, e_{ij})\}$$

. The objective function to be minimized is

$$UD = max_{e_{ij} \in E} \sum_{e_{kl} \in D(C(e_{ij}))} w(e_{kl}).$$

5.3 Game model

The problem can be considered as a two-player game, consisting of the following three moves:

- the first player decides a placement of N_v valves in the network;
- the second player selects one pipe to be damaged;
- the first player closes a set of valves that de-waters the damaged pipe.

The cost for the first player (and reward for the second) is the undelivered demand: the total demand (in litres per seconds) of all users that remain without service when the broken pipe is de-watered.

Given this formalization, the well-known minimax algorithm is applicable (126).

As we said in Section 5.2, choosing the last move is very easy, as there is only one reasonable solution: close all valves that are reachable from the broken pipe, without overpassing other valves. An implementation of this algorithm is given by (121).

Clearly the first step of the first player is the most sensitive, because it can generate a wide number of alternatives. In a network with N_e edges and with N_v valves, the search space is $\binom{2N_e}{N_v}$, since each edge can host up to two valves. However, some of the moves are not very interesting, for three main reasons that will be explained in detail in the next section. First, some solutions are clearly non-optimal. Second, some are symmetric, and provide valve placements that, although different, represent equivalent solutions. Third, after some solution is known, there is no point in looking for worse solutions: as soon as the current search branch cannot lead to solutions better than the incumbent, we can stop the search, backtrack, and continue from a more promising branch.

Each of these three cases provides a possible pruning of the search space, that can exponentially speed-up the computation with respect to a naive approach. The first two cases can be thought of as *constraints*, while the third can be though of as a *bound*: all of them can be simply cast in Constraint Logic Programming on Finite Domains (CLP(FD)).

5.4 Constraint Logic Programming model

We can now show how to model in CLP(FD) the valve placement problem. We first provide a simple minimax algorithm, then improve it with the three types of pruning hinted at earlier.

5.4.1 A minimax implementation in CLP(FD)

We associate a Boolean variable to each possible position of a valve (so we have two Boolean variables for each edge in the graph); if the variable takes value 1, then the given end of the edge hosts a valve, otherwise, if the variable takes value 0, there is no valve in such location. In the following, the list of these variables is called $Valves$.

The two-player game can be implemented as follows:

```
solve(Valves,N_v):-
        impose_constraints(Valves,N_v),
        minimize(
            (
                assign_valves(Valves),
                maximize(
                    (
                        break_pipe(Broken),
                        close_valves(Valves,Broken,ClosedValves),
                        undelivered_demand(Valves,ClosedValves,UD)
                    ), UD, MaxUD
                )
            ),
            MaxUD,MinMaxUD
        ).
```

The `minimize/3` and `maximize/3` meta-predicates are predefined in most CLP(FD) languages; declaratively, `minimize(G,F,V)` provides, amongst the solutions of goal G (bindings to the variables in G that make true the goal G), the solution that provides the minimum value for variable F (128, 129); such minimal value is bound to variable V. In other words, the result of $minimize(G, F, V)$ is equivalent to the Prolog goal

$$findall(F, G, List), minlist(List, V)$$

where $minlist$ finds the minimum value V in the $List$.

Operationally, it has a better performance, since it does not need to find all the solutions of G, collect them in a $List$, and find the minimum, but it implements a form of branch-and-bound. Operationally, `minimize` `call`s goal G and, if it succeeds providing some binding

F/F^*, it imposes a new unbacktrackable constraint $F < F^*$; then it continues the search. The unbacktrackable constraint is considered in the constraint store of all the nodes of the search tree, and prunes every node that cannot possibly provide a lower value than F^*. When the goal G fails, the optimal value is the last value obtained as F^* (130), if it exists. `maximize` is treated symmetrically.

Predicate `impose_constraints` posts all the constraints of the model to the constraint solver. It contains the constraint stating that there are N_v valves in the distribution network; other constraints will be described in Section 5.4.2.

`assign_valves` starts the search on the $Valves$ variables.

After finding a possible positioning of the valves that satisfies all constraints, a maximisation phase tries the moves of the opponent player: it searches (predicate `break_pipe`) the pipe that, if damaged, can be fixed only giving a maximum disruption of the service. When the opponent has chosen a pipe to break, we can compute the valves that should be closed to allow for substitution of the $Broken$ pipe; finally, we compute the undelivered demand.

Thus, the internal `maximize` finds, amongst the moves of the opponent player (`break_pipe`), the move that gives the maximum undelivered demand; such value is bound to variable $MaxUD$. The first player, instead, chooses the placement of the valves (`assign_valves`) with the aim of minimizing the value $MaxUD$.

5.4.2 Reducing the number of moves

The number of moves of the first player is huge even for small networks and number of valves. However, as hinted at earlier, some configurations can be avoided, as shown in the next paragraphs.

5.4.2.1 Redundant valves and symmetries

Consider the network in Figure 5.2. Even without knowing the demand on the various pipes, we can tell that some of the valves are redundant, just by looking at the topology of the network.

Valve $v_{1,2}$ cannot be used to identify a sector: the pipe immediately on the left of the valve belongs to the same sector as the pipe immediately on the right. In fact, there is a closed path going from one side of the valve to opposite side: starting from node 1, we can go to node 3, then 4, then 2, and we reach the opposite side of the same valve without having met any other valve. The same holds for valve $v_{3,5}$: there exists the path $(3, 4, 6, 5)$ that connects one end of the valve to the other end.

In general, we can say that in any closed path of the network, there cannot be exactly one valve. No valves means that the whole path will be contained in a sector, which is sensible.Two

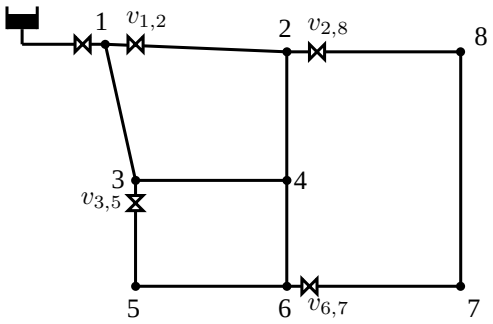

Figure 5.2: A network with redundant valves

valves or more can mean that the path is divided into two or more sectors.So, for each closed path, one could impose a constraint saying that the number of valves in such path cannot be equal to 1.

Indeed, the number of paths is exponential in the size of the network, however we can choose to impose such constraint only for a limited number of closed paths. We decided to impose one such constraint for each (boundary of a) *face*, that is a concept of planar graphs. When drawing the graph on a plane, each of the regions surrounded by edges of the graph is called a *face*. The number of faces of a planar graph is always polynomial, as proven by Euler.

Notice that, when a node is connected to exactly two edges, we have a symmetry. For example, consider node 8 in Figure 5.2: in one assignment, we could have a valve $v_{8,2}$, while another assignment could be identical but with a valve in $v_{8,7}$. These two solutions are symmetric, because the fact that node 8 is in the same sector as edge $e_{2,8}$ or as $e_{7,8}$ is irrelevant, since nodes do not have a contribution to the objective function. So, we can impose the symmetry breaking constraint $v_{8,7} = 0$. This simple observation can provide a notable speedup in the search, because real networks often have this situation.

5.4.2.2 Bounding

Consider a node in the search tree that selects the move for the first player (predicate `assign_valves/1`): in a generic node, some of the v_{ij} variables will be assigned value 1 (meaning that some valves have already been placed), some variables will have value 0 (meaning that in such position there is no valve), and some will still be unassigned.

Consider the example in Figure 5.3: circles represent positions in which there is no valve, while strokes are variables still unassigned. Even though we do not have a complete placement, we can already say that there is a sector containing at least edges $e_{7,8}$, and $e_{6,7}$. The opponent player will have the option of damaging, e.g., pipe $e_{7,8}$, causing an undelivered demand that

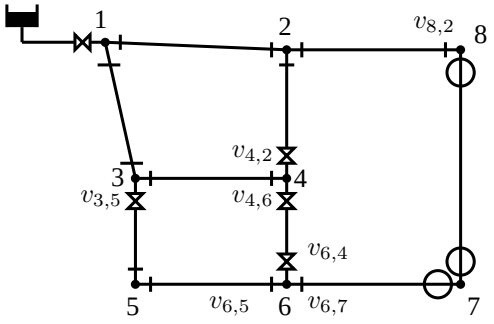

Figure 5.3: A partial assignment: circles mean absence of valve, strokes are variables not assigned yet

is no less than $w(e_{7,8}) + w(e_{6,7})$. So if the cost of such sector is worse than the current best solution found by the first player (i.e., $w(e_{7,8}) + w(e_{6,7}) > UD^{best}$), there is no point in continuing the search on the current branch. Note that this bound considers only the cost of the sector, without including unintended isolation.

We can also perform a reasoning similar to reduced costs pruning (131, 132). Suppose that $w(e_{7,8}) + w(e_{6,7}) < UD^{best}$ but adding $w(e_{2,8})$ is enough to overpass the current best UD^{best} (i.e., $w(e_{2,8}) + w(e_{7,8}) + w(e_{6,7}) > UD^{best}$): this means that we cannot afford to include edge $e_{2,8}$ in the same sector, and the only possibility to get a solution better than UD^{best} is to separate the two sectors, placing a valve in $v_{8,2}$. Thus, we can impose $v_{8,2} = 1$.

5.5 Implementation details

5.5.1 Incremental bound computation

The bound described in Section 5.4.2.2 is very powerful, and reduces significantly the number of explored nodes. However, it can be rather expensive in terms of computing time, if implemented naively. In fact, it implies computing the cost of the sector one edge belongs to, which means identifying the sector, possibly visiting a significant part of the graph. So computing it again and again during search can make it very time consuming.

Instead of restarting from scratch the identification of the sectors and computing their cost at each node of the search tree, we compute them incrementally.

We associate to each node i of the graph a variable S_i that represents the sector the node belongs to, and the lower bound LB_i on the cost of the sector S_i.

A constraint is associated with each edge of the graph $e_{i,j}$, and relates the two variables $v_{i,j}$

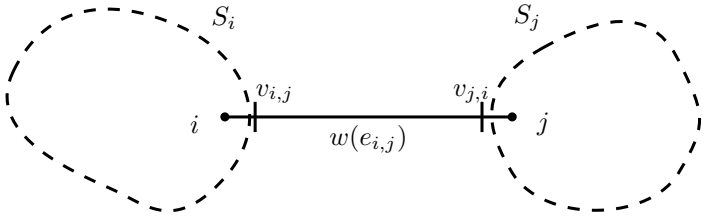

Figure 5.4: Example of propagation of the lower bound when joining sectors

and $v_{j,i}$ with the two sectors S_i and S_j, and with their lower bounds LB_i and LB_j:

$$lower\_bound(v_{ij}, v_{ji}, S_i, S_j, LB_i, LB_j). \tag{5.1}$$

Declaratively, constraint (5.1) states that, given the value of variables $v_{i,j}$ and $v_{j,i}$, the undelivered demand cannot be lower than (the maximum of) the two bounds LB_i and LB_j.

Operationally, the constraint (5.1) is awakened when one of the variables $v_{i,j}$ or $v_{j,i}$ becomes ground.

Initially no valve is placed, and each node is (tentatively) a sector by itself with associated lower bound zero (since no demand is associated to nodes).

If variable $v_{i,j}$ takes value 0, this means that there will be no valve in such position, so the sector S_i will have to include edge $e_{i,j}$, and we increment the value of the lower bound LB_i by $w(e_{i,j})$ (see Figure 5.4).

If both variables $v_{i,j}$ and $v_{j,i}$ have value 0, this means that the two sectors S_i and S_j should be joined: we unify the corresponding variables $S_i = S_j$, and increment the value of the lower bounds: we compute the cost of the joined sector as $LB_i + LB_j + w(e_{i,j})$.

Moreover, as explained in Section 5.4.2.2, if $v_{i,j} = 0$ and $v_{j,i}$ is not ground yet, but $LB_i + LB_j + w(e_{i,j})$ is greater than the current best solution, then joining the two sectors would give a solution worse than the current best, so we can impose a valve near node j, i.e., $v_{j,i} = 1$.

5.5.2 Dealing with unintended isolation

As mentioned in Section 5.4.2.2, the bound computed by constraint (5.1) does not take into account unintended isolation. However, when evaluating the total damage associated to the breaking of a certain pipe (which results in the isolation of a certain sector) it is necessary to consider also this aspect. Predicate `undelivered_demand/3` finds the correspondent actual value of the objective function used in `maximize/3`.

Its algorithm is based on the following principle. Isolating a sector is equivalent to removing, from the graph describing the network, the part of the graph which belongs to said

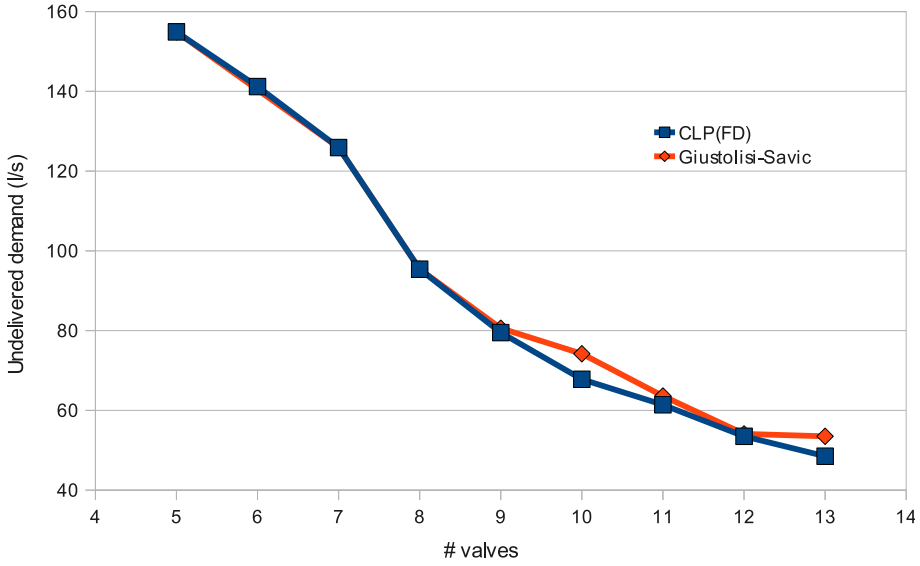

Figure 5.5: Comparison between the approximate Pareto front computed by Giustolisi-Savić and the optimal Pareto front obtained in CLP(FD)

sector. It is then subsequently possible to determine the connected components of the obtained subgraph (computational complexity is linear in the size of the subgraph (133)). The `graph_algorithms` library (134) of ECL^{i}PS^e Prolog provides efficient implementations of predicates for such operations on graphs.

Selecting the connected component which includes the source node and summing up the demands on the pipes contained in it gives the *deliverable* demand. The total undeliverable demand can thus be obtained subtracting the deliverable demand from the total network demand.

5.6 Experimental results

We compare our results with those reported by (135), and we apply our CLP(FD) algorithm on the Apulian water distribution network reported in that paper. Both the software and the instance are available on the web (136). The network has 23 nodes and 33 edges. In (135) authors adopt a multi-objective genetic algorithm, that minimizes both the number of valves and the undelivered demand. The aim is to find the so-called Pareto frontier (125); in this problem, a solution belongs to the frontier if there is no way to reduce the undelivered demand without increasing the number of valves (and, vice-versa, it is impossible to reduce the number of valves without increasing the undelivered demand). The genetic algorithm, however, is not able to prove that a solution is indeed Pareto-optimal, and provides an approximation of the

Pareto frontier, i.e., a set of points that are hopefully near to the real Pareto frontier. Moreover, (135) use a simplifying assumption: *"in order to reduce greatly the search space of the optimizer, the constraint of a maximum of one valve for each pipe was tested"*. In the paper, they report the best found solutions obtained with a number of valves ranging from 5 to 13.

We computed the true Pareto-optimal frontier by varying the number of valves from 5 to 13 valves, and computing for each value the best placement. The comparison of the near-Pareto-optimal frontier and the true Pareto-optimal frontier obtained with our CLP(FD) program is shown in Figure 5.5. It is worth noting that authors of (135) do not provide a solution with 6 valves, possibly because their algorithm was not able to find a solution with undelivered demand lower than that obtained with 5 valves. We proved, instead, that such a solution exists and adding a valve reduces the damage. Excluding this case, when the number of valves is low (up to 8 valves) their algorithm found the real optimum, probably due to the fact that the search space is still not very wide, so the genetic algorithm is able to explore a wider percentage of the search space. When the number of valves increases, their algorithm gets farther from the real optimum, with a gap of about 10% with 10 and 13 valves. Note also that we were able to find a solution with 12 valves that gives the same undelivered demand that (135) compute with 13 valves: in this sense, we were able to save one valve (out of 13) maintaining the same cost for undelivered demand.

The computation time is reported in Figures 5.6 (linear scale) and 5.7 (log scale). All experiments were done on a computer featuring an Intel Core 2 Duo T7250 2GHz processor with 4GB of RAM (note, however, that the current implementation does not use parallelism, and uses only one core). We show the performance of the basic algorithm, and of the improved versions that include the reduction of redundant valves (Section 5.4.2.1) and the bound (Section 5.4.2.2) varying the number of valves. From the graph in linear scale (Figure 5.6) we can see that each of the improvements has a significant impact in terms of reduction of the computation time. When the number of valves is low, the elimination of redundant valves (i.e., imposing that in a face there cannot be exactly one valve, Section 5.4.2.1) has a very strong effect, while the bound has almost no effect. On the other hand, when the number of valves increases, the bound seems to have a higher impact. Combining the two, we get a further improvement, with a reduction of the computation time of more than two orders of magnitude.

Figure 5.7 shows that the computing time grows less than exponentially with respect to the number of valves. This can be explained by the fact that the search space does not grow exponentially, but it varies as the binomial coefficient.

ECL^{i}PS^e has two implementations of the branch-and-bound predicate for minimization (130). One, called `min_max`, restarts the search after a new solution is found; this means that the first part of the search tree is explored every time a new solution is found; on the other

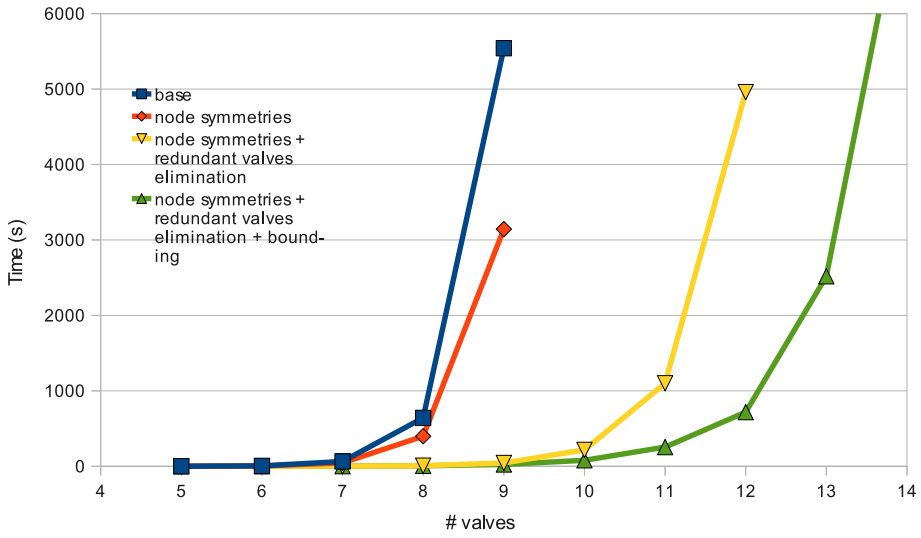

Figure 5.6: Computation time of the algorithms including different optimizations

hand, restarting the search allows ECL^{i}PS^e to add the unbacktrackable constraint from the root node of the search tree, and propagate effectively on all the nodes of the tree. The second, called `minimize`, avoids the restarts and continues the search, taking the risk that the newly added unbacktrackable constraint will not be able to propagate immediately, but only after some changes to the domains of the cost variable has happened. In our application, we found that `min_max` was about one order of magnitude slower than `minimize`.

Indeed the computation time is much higher than the one reported in (135) where the whole (near) Pareto frontier is obtained in just 10 minutes on an older computer. However, our algorithm is able to find the true optimum and prove its optimality, which is well-known to be often more difficult than finding the optimum itself, so in a fair comparison the time required for proving optimality should not be taken into account. In Figure 5.8 we show the anytime behaviour of our algorithm, i.e., we plot the solution quality with respect to the computing time in a typical instance. Indeed, our algorithm takes about 50 minutes to compute just one point of the Pareto frontier. However, looking closer at the graph one notices that our algorithm gets to a reasonable quality in a few seconds, it takes 27 minutes to get to the same quality obtained by the genetic algorithm, then it is able to improve on it and takes 37 minutes (total) to find the real optimal solution.

(135) also show the graph of the (near) Pareto-optimal frontier with a higher number of valves, but they do not report the solutions, so we cannot make a comparison. We tested our algorithm with the same number of valves (up to 24); we could not prove optimality, but we were able to find reasonable solutions within a few minutes.

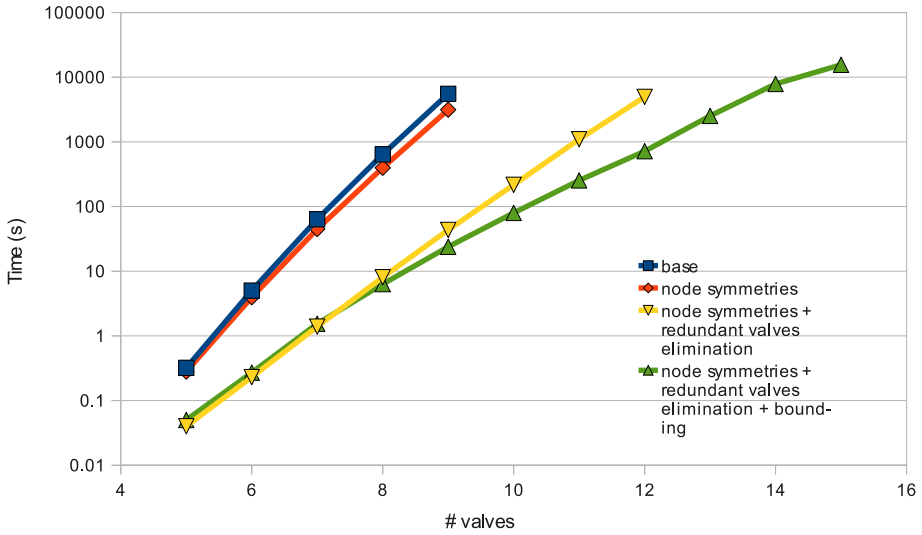

Figure 5.7: Computation time of the algorithms including different optimizations, log scale

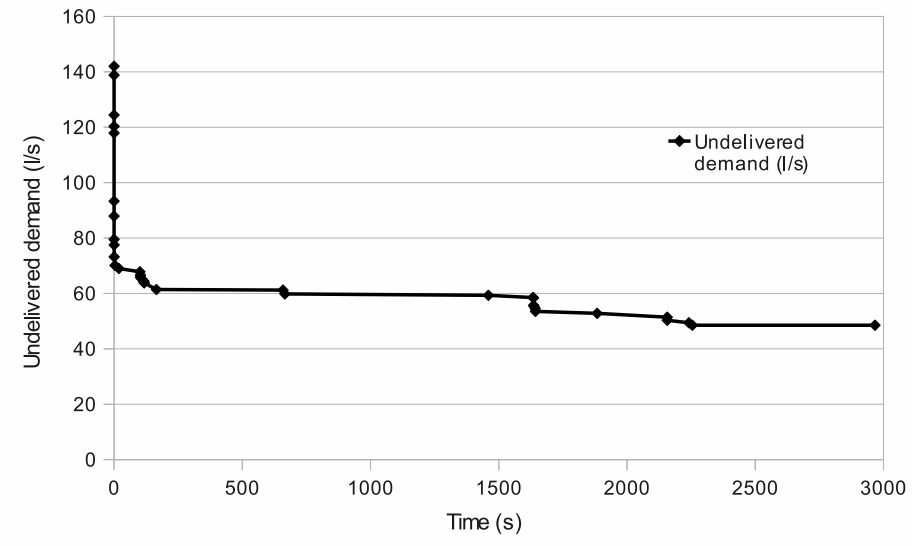

Figure 5.8: Anytime behaviour of the CLP(FD) algorithm: solution quality with respect to the computation time. Number of valves $N_v = 13$

5.7 Related work

In the literature of hydraulic engineering, two main problems related to the isolation valves in a pipe network have been faced, that is a) the identification of the segments and undesired disconnections that occur after a set of isolation valves has been closed and b) the (near) optimal location of the set of isolation valves.

As far as the first topic concerns, in the literature there are a number of studies regarding segment identification and the undesired disconnections that occur following the closure of a set of isolation valves. In particular, the methods proposed in (121) and (137) are based on a dual representation of the network, with segments treated as nodes and valves as links. The methods proposed in (138) and (122) use topological incidence matrices to identify the segments.

As far as the second topic concerns, in (122), a method is presented for the near-optimal placement of isolation valves based on a multi-objective genetic algorithm. Given that the placement of isolation valves is the result of a compromise between the need to reduce the costs tied to purchasing and installing the valves and the simultaneous need to assure high system reliability in the event of routine or non-routine maintenance, authors of (122) use the number of valves to be installed–as a surrogate for cost– and the maximum demand shortfall (the demand shortfall represents the unsupplied water demand after isolating a segment) in the different (disconnected) segments of the network as the objective functions to be minimised. Authors of (138) instead propose a different couple of objective functions, that is total cost of the set of valves, the cost of each valve being calculated as a function of the pipe diameter, and the weighted average unsupplied demand associated with the segments. Also in (138) the optimization is solved through a multi-objective genetic algorithm. All of these works use incomplete algorithms, that cannot ensure that the found solution is the real optimum; to the best of our knowledge, our proposal is the first complete algorithm to address the valve placement problem.

The valve placement problem has some similarities with the graph partitioning problem, in which the goal is to partition a graph into (almost) equal-size parts by removing the minimal number of edges or (in the weighted edges case) such that the total weight of the edges which connect different parts is minimized. In general, graph partitioning is NP-hard (139). Most works in the literature deal with heuristics or approximation algorithms.

One of the first works in the area is (140): authors propose a greedy algorithm which outputs a graph bisection. Starting from an initial solution (which can be suggested by some criterion or also be found randomly), each step of the algorithm evaluates the improvement in the objective function that would be obtained moving a vertex from one partition to the other

and takes the best choice. Iteration goes on until convergence to a local optimum is reached. The bisection can be applied recursively to partition further. In (141) the algorithm is improved so that the asymptotic behaviour of the algorithm is linear rather than quadratic.

A different approach is based on the *spectral* analysis of the graph. A graph can be represented by its incidence matrix: a square matrix $N \times N$ (if N is the number of vertices) whose (i, j) element is 1 if there is an edge from vertex i to vertex j and 0 otherwise. Its representation as a Laplacian matrix is obtained as the (matricial) difference between the diagonal matrix which has in position (i, i) the degree of the node i, and the incidence matrix. The set of the eigenvalues of the Laplacian is the graph *spectrum* ((142) gives an extended account on the subject). Since it was shown (143) that the second smallest eigenvalue of the Laplacian associated to a graph contains information about its connectivity, various partitioning heuristics were proposed relying on the eigenvectors (144, 145). In comparison with other heuristics, spectral methods provide good quality partitions at an increased computational cost (necessary to compute the matrix eigenvalues).

Various kinds of heuristics can be used in multilevel schemes, which reduce the size of the graph by collapsing vertices and edges, partition the smaller graph, and then uncoarsen it to construct a partition for the original graph. In (146) spectral methods are employed to partition the smaller graph, and use a variant of the Kernighan-Lin algorithm to periodically refine the partitions. In (147) a coarsening heuristic is adopted, for which the size of the partition of the coarse graph is within a small factor of the size of the final partition.

The special case of *planar* graphs (i.e. graphs which can be drawn without intersecting edges) is of particular interest for our application since it is often the case for water supply networks. Finding the optimal solution is NP-hard also for the planar case, however the *planar separator theorem* (148) states that a bisection in which the biggest set contains at most two thirds of the vertices and whose separator contains $O(\sqrt{n})$ vertices can be found in linear time.

Other related problems are the multicut problems (149), in which the aim is to find the minimal set of edges (or nodes) such that given pairs of nodes are no longer connected. In our case, instead, the aim is to disconnect a possibly small part of the network while keeping connected all the rest.

The algorithms for graph partitioning or solving multicut problems are clearly not directly applicable to the valve placement problem, also because of the issue of unintended isolation mentioned in Section 5.1. However, it would be interesting to hybridise our algorithm with some of the techniques available for such problems; we plan to study the feasibility of such approaches in future work.

6

Workload-Balanced and Loyalty-Enhanced Home Health Care

6.1 Introduction

The aim of Computational Sustainability is balancing environmental, economic, and societal needs for sustainable development. One of the most important societal needs is health care, which is at the same time one of the most (under an economic but also environmental point of view) expensive to achieve.

The relevant cuts in government spending, affecting health and welfare budgets, which have occurred in most countries all over Europe and North America in the last years, have spurred research on how to improve efficiency and minimize the costs of the services provided in those fields. One of current trends to achieve costs reduction and maintain service quality of health services is to close peripheral hospitals, reduce patients hospitalization, and concentrate the service at few, big structures, able to provide specialized treatments and high quality consultancy.

At the same time, though, patients who do not need to be treated in a hospital must be provided health care directly at their homes. The aim is to gain all the advantages coming from keeping the patient in a friendly environment while saving on hospitalization expenses. Indeed, high quality home health care following hospital dismissal has proved essential in reducing hospital readmissions, if able to cope with preventable complications.

The challenge is to deliver the service in a cost and resource effective manner, while achieving high medical treatments quality standards (i.e. comparable to those achievable in a hospital). Decentralization introduces elements of complexity in the rostering of personnel assigned to home health care. The area of intervention is no more local and common but scattered on a

Service	Time
First visit	1 hour
Enema	30 min
General check	15 min
Tracheotomy check and tube change	15 min
Central venous catheter	30 min
Electrocardiogram	30 min
Emogasanalysis	15 min
Mouth cleaning	5 min

Table 6.1: Excerpt of the services with average service time

territory which can also be rather extended. There is a need for accurate logistics taking into account the well being both of patients (who need constancy in the assigned nurses) and nurses (who need balanced and reasonable workloads) while keeping under control expenses in terms of money (to pay for fuel and occasional long hours of nurses) and resources (fuel consumption, pollution) in order to avoid disadvantages to overcome the (potential) advantages.

In this chapter we describe an application concerning home health care in the city of Ferrara, Italy, and how it can be addressed with Constraint Programming.

In the rest of this Section we describe in detail the problem and the specific requirements we need to face. In Section 6.2 we present a CP model which captures them. The search strategies devised to complement it are described in Section 6.3 and extensively tested in Section 6.4. Section 6.5 contains a discussion about related work.

6.1.1 The home health care service in Ferrara

At present, the home health care (HHC) service in the city of Ferrara, Italy, is managed by the local agency of the National Health Service (NHS), namely AUSL 109. All patients who are not self sufficient and in need of medical treatment are eligible. Requests are characterized by a patient identifier (name and address), a medical treatment, and the specific day of the week when the treatment must be delivered (each patient can have more than one request per week).

Service is provided by a set of qualified nurses. Every day, each nurse who is on duty starts her job from the hospital, visits the patients in her list delivering the required treatments and traveling by car from one patient's home to the next eventually returning to the hospital.

A treatment lasts from 5 to 60 minutes, depending on its specific characteristics; Table 6.1 shows an excerpt of the set of treatments with average service times. Moreover, a single patient

may need several different treatments in the same day. Such requests are usually carried out as a whole by a single nurse and thus handled as a single request. Therefore, requests duration is quite heterogeneous over the whole set of requests.

While Ferrara is a medium-size town (about 150,000), the area administered by AUSL 109 is rather large and its population aging. Although most of the population is concentrated in town, a number of elderly people live in the countryside and they are those more likely to be enrolled in the service. Therefore, the service is characterized at the same time by a high variance of duration and a significant geographical dispersion of the requests.

Scheduling such a service poses several challenges; a good solution should achieve:

- from the NHS point of view, the minimization of the travel time over the service time; i.e. the travel time during which a nurse is on duty but is not delivering any service.
- from the nurses point of view, the equidistribution of the workload, which can not be guaranteed by simply equally subdividing patients, due to heterogeneous requests.
- from the patient point of view, a good degree of *loyalty*, i.e., the number of different nurses who are in charge of a single patient should be minimal.

6.1.2 The problem data

Resources At present, a total of 15 nurses is involved. Every working day (Monday to Friday), 12 nurses are on duty, out of which 9 operate in the morning and 3 in the afternoon.

Working rules A duty should last up to 7.12 hours on a working day. This limit is not strictly enforced, provided that the threshold of 36 hours per week is not exceeded over a planning horizon of 4 weeks.

Demand As a representative sample of the set of requests, we consider the month of February 2010, with a total of 3323 requests, subdivided among 458 patients. Several patients are located either in town or in its suburbs, though quite a few live in some neighboring towns in a 20km radius.

6.1.3 Aim of the project

Optimizing the health home care scheduling is essential to make the service cost-effective and to avoid the so called *burn out* phenomenon, i.e. nurses who get tired and act in an unfriendly manner to patients or even leave the job. We start by examining the current solution approach, as it is manually carried out in AUSL 109.

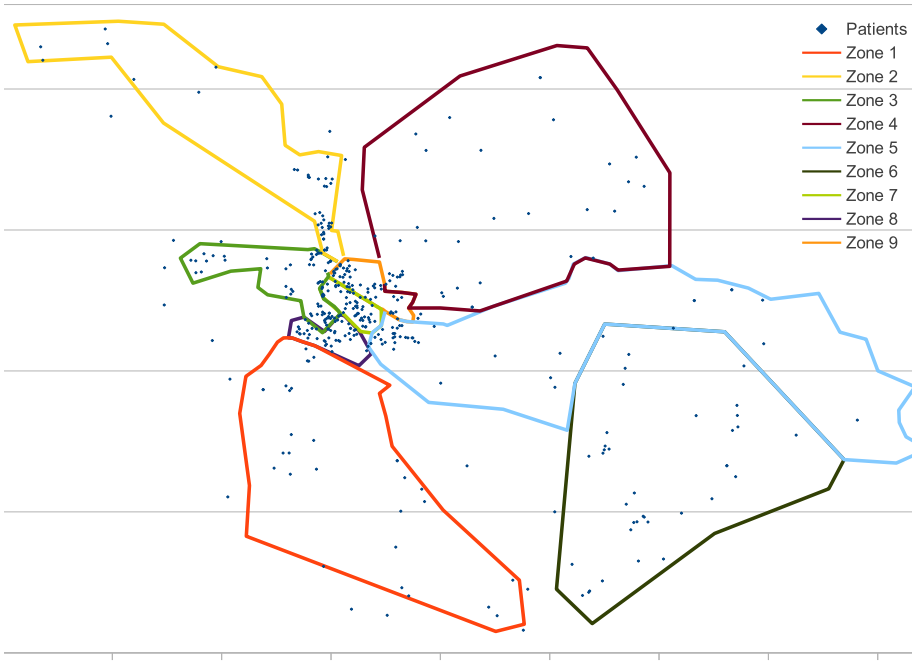

Figure 6.1: The 9 zones in which the area is divided, dots show where patients are located

At present, nurses organize their duties themselves. In order to simplify the subdivision of the patients to the nurses, the territory pertaining AUSL 109 has been partitioned statically into 9 zones (shown in Figure 6.1) considering several factors, such as the distribution of the population, its age and historical data. Each nurse on duty on Monday morning receives in charge most of the patients belonging to one such area. Then the nurse tries to fit her patients requests for the whole week into her working shifts of the week, while complying with the maximum weekly workload allowed. The assignment of request to daily shifts follows a sort of greedy criterion, trying to stuff as many requests as possible within each shift. Such decisions are not driven by any optimization criteria, and the routing is not necessarily optimal within the day, leaving apart what could be gained in terms of traveled distance if requests would be exchanged between nurses. If a nurse can not fulfill all the requests of *her* patients, those that are not accomplished are forwarded to the chief nurse, who is in charge of the service coordination, and get reassigned to the other colleagues. Due to the greedy procedure followed, the nurse weekly schedules have very different workloads and balancing this load over the months leads to a detriment in loyalty. Nurses complain about such disparities, and have difficulties adapting their schedule to new incoming patients, new treatments, or to any other change. Moreover, if the workload balance could be improved by optimizing the routing component, nurses could be available at the hospital for others tasks, thus reducing the overall costs. In addition, an

improved routing plan would impact on the direct expenses related to gas and car usage, which contribute to the overall cost.

Decisions that have to be taken to build a solution are the following: partitioning the requests to a set of duties, and for each duty determining the sequence of services to be carried out optimizing the traveling times.

6.2 Modeling the problem in CP

Constraint (Logic) Programming on Finite Domains, with its declarative features and flexible expressivity, has the right characteristics to represent a good approach to the HHC problem. It offers the possibility, much needed in this domain, to smoothly add and relax constraints in order to follow change in the requirements. Moreover it allows one to easily model the required optimization criteria which can be difficult to formalize under other paradigms. Although the strength of CP lies more in the feasibility aspects of our problem, while it might be penalized by the optimal routing side of it, CP is a very effective framework for the integration of components based on different approaches, as it will be described more in detail in the following.

The input of our model consists of:

- a set $\mathcal{S}_{serv}$ of services, of size N_s ; for each service s we know the patient pat_s, the day day_s and the duration dur_s
- a matrix of distances D; the element $d_{i,j}$ is the travel time from patient i to patient j (if i and j are both greater than 0), or from/to the hospital (if $i = 0$ or $j = 0$)
- $\mathcal{S}_{nurse} = \{1, \ldots, N_n\}$ is the set of available nurses
- N_d is the number of days considered in the scheduling
- $MinutesPerDay$ is the amount of minutes available per day for each nurse (including service time and travel time)
- $MinutesPerWeek$ is the amount of minutes available in one week. We assume

$$MinutesPerWeek = MinutesPerDay \cdot N_d$$

  even if, as explained in Section 6.1, it can also be in a different, more relaxed, way.

A solution is an assignment of a nurse to each service, respecting the constraints both on the day and on the week workloads. The quality of the solution depends on how balanced

the week workloads of the nurses are and on how many different nurses take care of the same patient during the week.

So, to each service s we associate a decision variable $Nurse_s$ that can take a value between 1 and the number of available nurses N_n.

It can be useful to represent the nurses variables also in their Boolean channeling version, using constraint reification. We have a matrix SN of size $N_s \times N_n$ such that

$$\forall s \in \mathcal{S}_{serv}, \forall n \in \mathcal{S}_{nurse},\ SN_{s,n} = 1 \iff Nurse_s = n.$$

We want to compute the workload of each nurse n in each day d: $DayWL_{n,d}$. Each day workload is the sum of the total service time and the travel time

$$DayWL_{n,d} \quad = \quad T^{svc}_{n,d} + T^{trv}_{n,d} \qquad (\forall n \in \mathcal{S}_{nurse}, \forall d \in 1 \dots N_d)$$

and the week workload $WeekWL$ is simply the sum of the respective day workloads

$$WeekWL_n \quad = \quad \textstyle\sum_{d=1}^{N_d} DayWL_{n,d} \qquad (\forall n \in \mathcal{S}_{nurse}).$$

The service time is the total time of the durations of the services given by nurse n in day d;

$$T^{svc}_{n,d} = \sum_{s \in \mathcal{S}_{serv}, day_s = d} dur_s \cdot SN_{s,n}.$$

The travel time $T^{trv}_{n,d}$ of nurse n in day d is computed by a constraint $traveltime$, whose meaning will be explained in Section 6.2.2.

There are various ways to achieve balanced week workloads for the nurses (150). We chose to minimize the maximum week workload, obtained as

$$MaxWeekWL = max_{n \in \mathcal{S}_{nurse}}(WeekWL_n)$$

One way to obtain maximum loyalty is to minimize the number of nurses that visit a same patient. Let $ServicePat_p$ be the set of services of patient p. The information if a patient p is visited during the week by nurse n is given by:

$$PN_{p,n} = \bigvee_{s \in ServicePat_p} SN_{s,n} \qquad \forall p \in \mathcal{S}_{patient}, \forall n \in \mathcal{S}_{nurse}$$

(where we identify truth values *true* and *false* with 1 and 0, respectively); then

$$LoyaltyPenalty = \sum_{p \in \mathcal{S}_{patient}, n \in \mathcal{S}_{nurse}} PN_{p,n}$$

The objective is a weighted sum of the two components

$$min(\alpha_1 \cdot MaxWeekWL + \alpha_2 \cdot LoyaltyPenalty), \tag{6.1}$$

where α_1 and α_2 are positive real numbers that can be chosen by the user in order to reflect the current priorities adopted in the AUSL. Of course, such values can be tuned later on in order to achieve the best results.

6.2.1 Using more Global Constraints

The model can be modified in order to exploit some more global constraints (see also Section 2.4.3) if they are available in the chosen constraint programming system.

With respect to service time, one can notice similarities with classic problems studied in AI, such as bin packing and the (multi)knapsack problems (151, 152), in which a set of N_{items} items with size $size_i$ have to be placed in a set N_k of containers (the bin/knapsack) without exceeding their capacities.

The global constraint $multiknapsack$ (also known as $binpacking$) (153) is suited to solve such problems; a multiknapsack problem can be defined with the constraint

$$multiknapsack(a, size, load)$$

where

- $a \equiv \{a_i\}$ is a set of decision variables a_i with domain $a_i :: 1..N_{knapsacks} (\forall i \in 1..N_{items})$, with the intended meaning that $a_i = j$ iff item i is assigned to knapsack j;
- $size \equiv \{size_i\}$ is the array containing the size of each item
- $load \equiv \{load_i\}$ is a set of constrained variables that represent the actual load of each knapsack, i.e. the sum of the sizes of the items assigned to that knapsack. In order to solve a multi-knapsack problem, the $load_i$ domains are constrained to take values up to the capacity of the knapsacks: $load_i \leq capacity_i$.

In the HHC problem, we can consider each service s as an item of size dur_s which has to be placed in one of the bins/knapsacks whose load is $T^{svc}_{n,d}$ with $d = day_s$.

We can thus bind the $Nurse$ decision variables with the whole T^{svc}vector: writing

$$multiknapsack(Nurse, dur, T^{svc}),$$

we enforce[1] $(\forall n \in \mathcal{S}_{nurse}, \forall d \in \{1..N_d\})$:

$$T^{svc}_{n,d} = \sum_{s \in \mathcal{S}_{serv}, d = day_s} [Nurse_s = n] \times dur_s.$$

With respect to loyalty, one way to optimize it is minimizing the number of nurses that visit each patient. For each patient $p \in \mathcal{S}_{patient}$ we would like to link the array of decision variables $Nurse_p$ with the number of different values in it. One of the most important global constraints,

[1] Square brackets in the formula are Iverson brackets (154): let P be a statement, $[P]$ evaluates to 1 if P is $true$ and to 0 otherwise.

$alldifferent$ (155, 156), enforces that all the variables of the array passed as argument to it are given a different value. The global constraint $NValue$ (157, 158, 159) can be thought of as a soft version of *alldifferent*, and it counts the number of different values in a vector of variables. Counting the different values in the set of decision variables associated to a certain patient p we obtain the number of different nurses who take care of that patient:

$$NValue(Nurse_p, PN_p) \quad (\forall p \in \mathcal{S}_{patient})$$

and we can compute the overall loyalty violations summing up the single contributions:

$$LoyaltyPenalty = \sum_{p \in \mathcal{S}_{patient}} PN_p.$$

6.2.2 Addressing the Routing

In order to satisfy the constraint on the maximum day workload, one should compute both the service time and the travel time of a nurse's daily shift. On the other hand, computing the travel time of a nurse implies computing the optimal route that connects all the patients assigned to her on that day, which is a (hard) optimization problem in its own, known as the Traveling Salesman Problem (TSP) (160).

We decided to embed a TSP solver inside a (user-defined) constraint $traveltime$, so that it can perform constraint propagation during search of the main problem (i.e., the HHC). Notice that in a given node of the search tree, some of the patients will be assigned to a nurse, while others will still be unassigned; the constraint $traveltime$ should compute the bounds of the travel time, namely the minimal travel time of the nurse (assuming that she will be assigned a minimal number of patients, amongst those in her domain), and the maximum one (assuming that she will be assigned all the possible patients in her domain).

In order to compute the travel time $T^{trv}_{n,d}$ of a nurse n in day d, we need to provide to such constraint: (1) the patients associated to the services (with their locations), (2) the matrix of distances D, (3) and the services to be provided by nurse n in day d. More precisely, it will be the sub-set

$$\mathcal{S}_{serv(n,d)} \triangleq \{s \in \mathcal{S}_{serv} | Nurse_s = n, day_s = d\}.$$

In other words, the actual parameters are:

$$T^{trv}_{n,d} = traveltime(pat_s, D, \mathcal{S}_{serv(n,d)}).$$

Operationally, the $traveltime$ constraint awakes every time one of the $Nurse$ variables is instantiated. We know which service s was instantiated, therefore the corresponding nurse n and day d. The constraint computes the TSP corresponding to the patients that must be visited

by nurse n in day d. This provides a valid lower bound to the actual travel time, and can be used in a branch-and-bound search. When all the $Nurse$ variables are ground, the cost of the TSP becomes the real travel time, and we are able to fix the value of T^{trv} to the TSP cost.

As far as TSP solvers are concerned, it is possible to choose between implementing one, e.g. in Constraint Programming (161, 162), and using an off-the-shelf solution.

We found that implementing a solver in pure CP slowed down significantly the search (as also witnessed in the literature (163)).

The state of the art in the area is probably Concorde (164), which relies on linear programming with the addition of specific TSP-targeted features. However it is not easily customizable and it is oriented to large instances, while the number of patients visited by a nurse in a single day in our case is rather limited.

We adopted a solver based on Lagrangian Metaheuristics (165) which performs very well on small TSPs and proved to be suitable for integration in our system.

6.3 Search Strategies

In CP search strategy design is responsible for the shape of the search tree and the way to explore it. We devised and applied several search strategies to our problem with the aim of incorporating knowledge of the problem and its structure to rapidly drive the search towards (good) solutions.

A first, general purpose one (**GS** – Generic Search) performs a depth first search selecting the variable to assign, at each node, according to the First Fail heuristic (i.e. the variable with the smallest domain is chosen) (166). The value to assign is selected randomly among the ones in the domain.

Since our problem is characterized by a huge search tree which is difficult to fully explore in a reasonable time, we adopted restarts (167): we want to avoid that (wrong) decisions taken near the root drive the search into a bad area without being reconsidered (thus inducing the well-known heavy-tail behavior (168)). This Generic Search with Restarts (**GSR**) is based on the optimal timeout sequence (169), as in (170).

We also modified the variable selection heuristics; instead of selecting the variable with the smallest domain within the services of the whole week, we try to compute the assignments of the single days, i.e., we first assign all the services of the first day, then the second day, etc. Within each day, we select first the variable with the smallest domain. This strategy was applied without restarts, Generic Search by Day (**GSD**), and with restarts: Generic Search by Day with Restarts (**GSDR**).

We devised a search heuristic which is more tailored to the problem structure (**LGS**).

The variable selection criterion selects first variables related to services with bigger durations, which is a sensible choice when dealing with $multiknapsack$ problems. Let the selected variable be associated to service s of patient pat_s, the value selection heuristic tries to assign a nurse who is already visiting pat_s in the attempt to keep the $LoyaltyPenalty$ low. If more than one nurse was already assigned to that patient previously, the one with the lowest $WeekWL$ is selected, in the attempt to keep workloads balanced.

This heuristic can also be adopted in the context of a Large Neighborhood Search (**LGS+LNS**), which is a technique that hybridizes Constraint Programming and Local Search (171) by restoring the domains of some variables after finding a solution and reoptimizing the restricted problem with a limit on the number of failures. With very few changes, it allows to adapt a CP model into a hybridized procedure that scales very well with the problem size, explores a neighborhood large enough to avoid the need for a metaheuristic, and requires minimal parameter tuning: a basic relaxation procedure, such as relaxing a randomly chosen subset of the variables, as we chose to do, usually works well in practice.

6.4 Experiments and Results

As expressed by the model described in Section 6.2, in our problem we have two optimization criteria: balancing the week workloads and obtaining the best possible loyalty. It is possible to optimize according to only one of them by imposing that either α_1 or α_2 in Equation 6.1 is 0.

Optimization according to one criterion, e.g. the maximum week workload, will yield a certain value for the other one (i.e., in this case, the loyalty penalization). It is subsequently possible to repeat the optimization on the week workload, but this time imposing that the value for the loyalty penalization has to be inferior than the one found previously. Iterating this process until no feasible solution is found one finds the Pareto-optimal front of solutions. In Figure 6.2 it is shown such a front obtained by iteratively running search LGS+LNS, explained in Section 6.3, with a 10 minutes timeout on one weekly instance.

Information from Pareto fronts can be useful in order to tune the values of parameters α_1 and α_2. Our evaluations lead us to adopt $\alpha_1 = \alpha_2$: in Figure 6.2, for instance, this choice leads to the point of coordinates $(225, 1724)$ which offers a good balancing of the two criteria.

6.4.1 ECL^{i}PS^e implementation

One version of the program was implemented in the open-source CLP software system ECL^{i}PS^e 6.0 (127).Tests reported in Table 6.2 have been performed on a computer with an Intel i5 processor 2.40 GHz and 4GB of memory. Four weekly instances have been used to test the proposed model using five of the search strategies in Section 6.3.

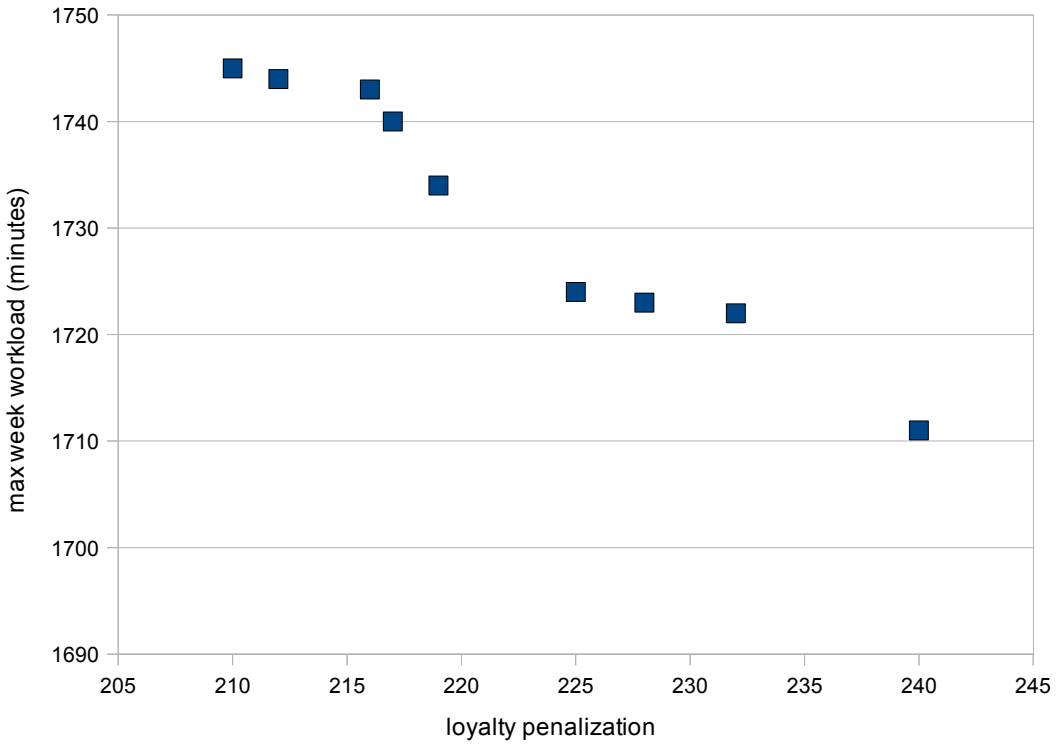

Figure 6.2: Pareto front of solutions of one weekly instance obtained using the LGS+LNS search

Table 6.2 shows the best results obtained for the five search strategies running them for a maximum of 10 minutes. The randomized algorithms (the GSXX strategies) were run 20 times each. For each week, we show the Objective and the corresponding $MaxWeekWL$ and $LoyaltyPenalty$. The results are compared to the solution hand-made (HMS) by the nurses based on the geographical subdivision of the area into 9 zones as explained in Section 6.1.1. We can see that the model was very effective, as all the search strategies were able to improve the hand-made solution and that the LGS strategy outperforms all the general purpose search strategies, Moreover, LGS was able to improve on the hand-made solution both in terms of equidistribution of the workload and in terms of loyalty, thus improving both the working conditions of the nurses and the service quality for the patients.

	First Week	Second Week	Third Week	Fourth Week
	Objective=WL+LP	Objective=WL+LP	Objective=WL+LP	Objective=WL+LP
GS	2203 1918 + 285	2371 2064 + 307	2331 2033 + 298	2387 2063 + 324
GSR	2125 1841 + 284	2347 2040 + 307	2270 1963 + 307	2345 2022 + 323
GSD	2185 1905 + 280	2351 2052 + 299	2255 1963 + 292	2389 2071 + 318
GSDR	2097 1811 + 286	2345 2033 + 312	2263 1958 + 305	2342 2011 + 331
LGS	1982 1782 + 200	2277 2042 + 235	2181 1954 + 227	2290 2034 + 256
HMS	2356 2124 + 232	2405 2153 + 252	2395 2141 + 254	2433 2146 + 287

Table 6.2: Comparison of search strategies and hand-made solution (ECL^iPS^e).

Since some of the search strategies use randomization, we also show box plots representing the data obtained on the various repetitions (Figures 6.3, 6.4, 6.5, 6.6). The plots reveal that restarts are the most important factor in the general purpose strategies. The strategies not using restarts in some cases were not able to find any solution in the allotted time, and often were unable to improve the hand-made solution. However, a search strategy tailored for the problem is able to provide a great improvement on the general purpose ones.

6.4.2 Comet implementation

Another implementation is based on the Comet (172) Constraint Programming system. Comet offers the implementation of the $multiknapsack$ and $NValue$ Global Constraints thus allowing the alternative model implementation explained in Section 6.2.1. Moreover since it is a system strongly oriented towards Constraint Based Local Search and affine hybrid methods, it is straightforward to implement a search strategy encompassing a Large Neighborhood Search as mentioned in Section 6.3.

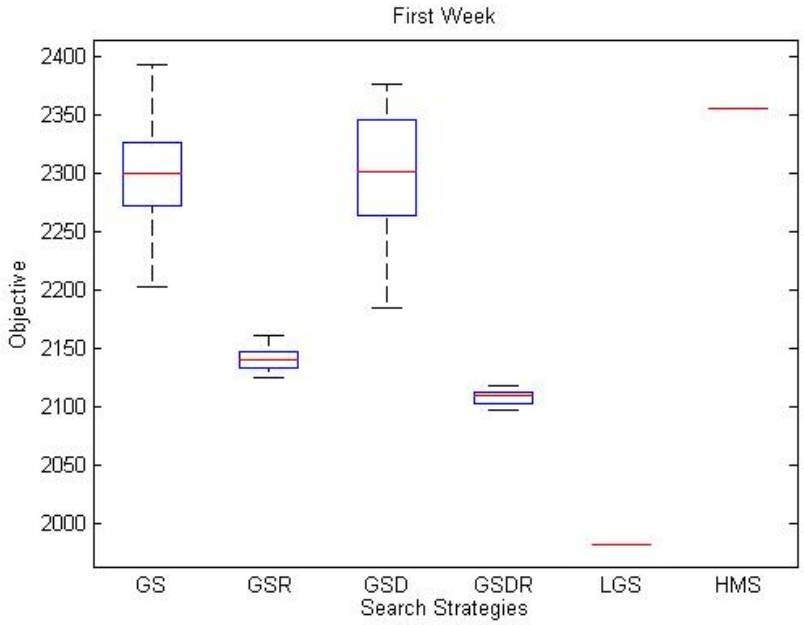

Figure 6.3: Results of the runs on the first week (ECL^{i}PS^e).

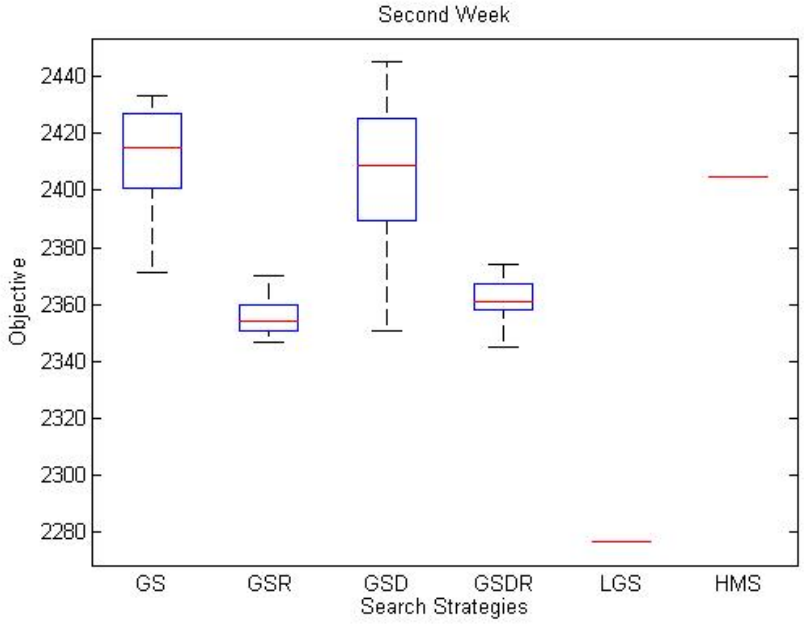

Figure 6.4: Results of the runs on the second week (ECL^{i}PS^e).

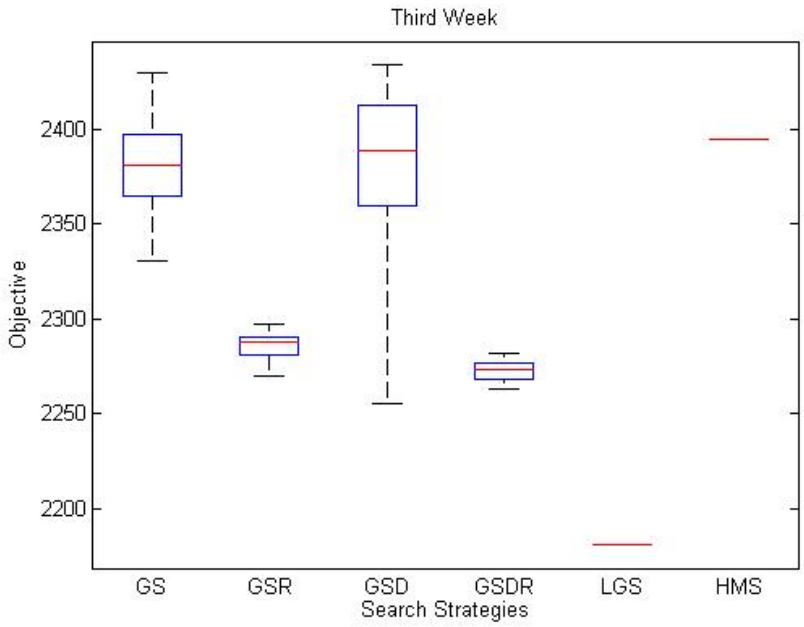

Figure 6.5: Results of the runs on the third week (ECL^{i}PS^e).

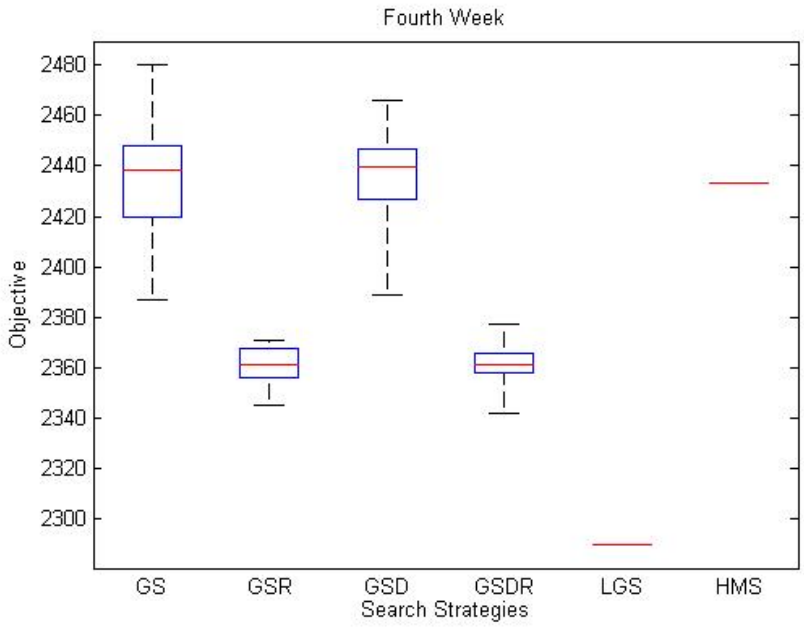

Figure 6.6: Results of the runs on the fourth week (ECL^{i}PS^e).

	First Week	Second Week	Third Week	Fourth Week
	OF=WL+LP (after)	OF=WL+LP (after)	OF=WL+LP (after)	OF=WL+LP (after)
GSR	2176 1896+280 (531)	2303 2000 + 303 (466)	2260 1960 + 300 (511)	2317 1992 + 325 (339)
LGS	2001 1789 + 212 (101)	2262 2030 + 232 (5)	2181 1954 + 227 (4)	2336 2080 + 256 (5)
LGS+LNS	1949 1724 + 225 (588)	2189 1959 + 230 (590)	2095 1872 + 223 (573)	2213 1946 + 267 (558)
HMS	2356 2124 + 232	2405 2153 + 252	2395 2141 + 254	2433 2146 + 287

Table 6.3: Comparison of search strategies and hand-made solution (Comet)

Tests were performed on a netbook with an Intel Atom Processor N450 1.66 GHz and 1 GB of RAM on four weekly real instances with a 10 minutes timeout.

In Table 6.3, we report results on one of the generic search strategies (GSR), LGS, LGS+LNS and HMS.

For each column we report the value of the objective function (OF) with its components of max weekly workload (WL) in minutes and loyalty (LP), moreover in brackets is included the time (in seconds) after which the solution was actually found. It can be noticed that our application always finds a better solution than the hand made one and that our search heuristic is able to find very quickly a good solution, which, with LNS, is improved further.

6.5 Related work

Bertels and Fahle (173) address the HHC with a hybrid approach, combining CP, local search and LP. The instances addressed by Bertels and Fahle (173) consider rather tight time windows, in fact *"Due to time window constraints, in the HHC only few permutations correspond to feasible orderings. In our approach we enumerate those orderings by a CP approach, and we use an LP to find optimal start times with respect to the objective."* Also Steeg and Schröder (174) solve a problem in Home Health Care with a strong focus on satisfiability, due to the impact of constraints such as time windows and nurse specific skills. The instance in Ferrara does not have such constraints, so the only criterion for assigning the order of patients in a nurse's schedule is minimizing the route. However, this does not mean that the problem is underconstrained or easier to solve. On the contrary, the search space is wider (as it is not pruned by simple constraints like time window or nurse skills), and enumerating the feasible orders is not a viable approach. The number of feasible solutions, on the other hand, does not necessarily increase: in a real-world instance new nurses are hired only when necessary, so the actual workload of the nurses is very close to their maximum working hours. This means that if

the routes are not very close to the optimum, the daily-workload constraint becomes infeasible: one has to find the optimum of the TSP just to find a feasible solution to the HHC problem or to assess infeasibility.

Home Care is a problem related to Home Health Care, although the latter is more focused on medical services while the former provides also social services to patients. For this reason, Home Care instances are almost always characterized by tight constraints on time windows. Laps Care (175) is a system for Home Care using an iterative method: an initial solution uses a single route for each service, then routes are joined until no further improvement is possible.

The HHC has some similarities with the Capacitated Vehicle Routing Problem (CVRP) which has been tackled by many solution methods, including branch and cut (176), meta-heuristics (177), and hyperheuristics (178). However, there are some important differences with CVRP that make all the efficient method developed for the classic CVRP not applicable in our case. In fact, our problem is more similar to a time constrained VRP for which the classical CVRP methods are not so efficiently adapted. Moreover, while we try to minimize the total traveled distance, as in the VRP, on the other hand we have to balance the workload and this component of the objective function is difficult to address by OR methods. Finally the loyalty is the component of the objective function that makes our problem very peculiar. It is worth noticing that the two criteria are often in contrast: any solution assigning to a single nurse all the services of the same patient would minimize the loyalty, but this would go to the detriment of workload balancing.

The popularity of hybridization of CP and Local Search (179, 180) is witnessed by the existence of solvers tailored for this hybridization (172). Rousseau et al. (181) propose a large neighborhood search in which CP explores a neighborhood with three operators. Kilby et al. (182) consider vehicle routing problems with side constraints, and compare classical OR approaches with CP models. Various works consider how to solve the TSP, or its variant with Time Windows in CP or with hybrid algorithms (163, 183, 184).

To address load balancing problems, constraints such as `deviation` (185) and `spread` (186) can be used in some cases. These constraints aim at minimizing the variance or the mean absolute deviation, and obtain very good balancing. On the other hand, they are not applicable in our case, because they require the total load to be fixed, while in our case it depends on a routing part, so it is not constant. Also, one way to reduce the variance of a set is to increase its minimum value, which in our case could amount to worsen the assignment of the least busy nurse (e.g., by giving her distant patients), without reducing the workload for the other nurses.

7

Conclusions

In this thesis we studied various problems related to Computational Sustainability addressed by means of Logic and Constraint Programming.

We dealt with some of the main aspects of sustainability. Energy is a key issue under several points of view: its efficient use, its renewability, its (environmentally friendly) production. Tightly related to this aspect is the matter of greenhouse gas emissions, whose volume in the atmosphere needs to be contained in order to avoid (disruptive) climate changes. Water, which is a vital need, must be managed carefully in order to meet increasing consumption boosted by industrial demand and an ever growing world population: it is necessary in the first place to drastically cut its waste. In general, decision makers, and governmental agencies in particular, face the responsibility to make the right technological and social choices and allocate resources sensibly in order to keep current standards of living reasonably high without burdening future generations. This also means encouraging various relevant service providers to efforts in order to make the most of the resources they are assigned.

The European Commission 'Horizon 2020' framework programme for research and innovation (187) reflects this awareness. In the 'Better Society' track, among the proposed challenges there are:

- 'Health, demographic change and wellbeing;' with relation to this subject we described in Chapter 6 an application for scheduling in home health care.
- 'Secure, clean and efficient energy;' Chapter 4 relates to this issue addressing the biomass plant placement problem.
- 'Climate action, resource efficiency and raw materials'; Chapter 4 is relevant also with respect to climate action. In Chapter 3 we mentioned how 'cloud computing' can enable savings in terms of greenhouse gas emissions and electricity consumption and we de-

scribed our contributions aimed at supporting this scenario. In Chapter 5 we dealt with the efficient distribution of water by optimal design of aqueduct isolation systems.

The problems we addressed were proposed by experts of the respective domains. Indeed, one of the challenges of applying computational techniques to such diverse areas is bridging the respective perspectives. However it is necessary in order to gain a correct detailed understanding of the matters at hand and of their subtleties. Moreover, having the opportunity to test our approaches on real data allowed us to direct our design efforts toward a practical working perspective.

We showed how Computational Logic and Constraint Programming can effectively contribute to the solution of such problems. Their declarative, powerful expressivity is apt to model complex domain knowledge and problem requirements in an elegant and concise way, at the same time allowing one to efficiently tune the implementation.

In the following, we summarize the main contributions contained in each chapter.

7.1 Computational Logic tools for Green IT

The rapidly increasing need for electricity of IT infrastructures calls for innovation in hardware and software architectures enabling deployment in the most efficient way. The 'cloud computing' paradigm offers, among its various advantages, benefits in terms of reduction of energy consumption and greenhouse gas emission of data centers (1, 47).

Such a scenario is promising but also implies increased complexity whose burden needs to be taken over by intelligent tools in order to fully unleash the vision's potential. We proposed contributions supporting this perspective based on Computational Logics.

We tackled the issue of automated service composition (5), proposing an integration of the SCIFF abductive framework for service contracting with Description Logics reasoning and knowledge representation. In this way we brought together the best features of both areas and the result is a framework which can effectively express complex policies and efficiently query rich knowledge bases.

We also dealt with the dual issue of learning and updating service policies from records of actual interactions. We extended the DPML system (50), which is able to infer a process model expressed by a set of logical integrity constraints starting from logs containing positive and negative traces. The resulting IDPML system (3) is able to revise theories when new evidence is available. This allows one to deal with the case in which the process changes over time and new traces are periodically collected. Experimental evaluation showed that, when new evidence becomes available, revising the current theory is faster than learning a new one from scratch, and the accuracy is higher too.

7.2 Biomass Plant Placement with Energy-Effective Supply

Under the point of view of sustainable energy production, we focused on wooden biomass. It is a very promising source of energy under the point of view of renewability and carbon neutrality and for this reason governments and other public institutions tend to incentivate its use, e.g. by means of tax benefits.

However, plants need to be built, biomass has to be collected and transported. All these tasks imply energy consumption, usually coming from traditional fossil fuel. If economic incentives are not aimed correctly, they could propel a paradoxical situation in which the biomass plant and its context, taken as a whole, is lucrative despite its energetic balance being negative. When deciding about incentives, therefore, one of the most important aspects to assess, is the plant placement and its intended supply chain.

We devised, implemented and tested on real instances, in cooperation with the environmental agency of Emilia-Romagna region, an optimization tool which addresses this problem (6). In order to make accurate decisions it considers a wide range of data, including information about impact of power plant construction in different zones, location of green areas and transportation networks.

In the literature this problem is usually addressed under the point of view of profit maximization (88, 90). Our CLP($\mathcal{R}$) model still takes into account economic factors (fundamental in order to assess the feasibility of the business plan), but its emphasis is on the maximization of the net energy considering the whole system (i.e. including, besides the functioning plant, its construction and the collection and transportation of biomass).

7.3 Aqueduct Valve Placement for Minimal Service Disruption

The need to preserve water resources requires improvements in the robustness of aqueducts. This property is not just related to the construction materials, but it depends on the overall design. A robust acqueduct has as less fragile components as possible, is reasonably maintainable and is apt to contain service disruption and water losses when damages occur.

One of the fundamental choices under this point of view is related to the placement of valves. When a pipe needs to be repaired, it has to be disconnected from the rest of the network. This effect is achieved by closing a suitable subset of the installed valves. Valves are expensive and require high maintenance efforts, so it makes sense to keep their number low. It therefore becomes crucial to place them in a way which maximizes their contribution.

We addressed this problem by means of CLP(FD) (7) in cooperation with hydraulic engineers who suggested, in accordance with common practices and the literature, to aim for the

minimization of the maximum undeliverable demand.

We tested our algorithm on real instances and compared it with results in the literature. Our approach is complete (finds the global optimum) and was able to improve some of the existing solutions. It is worth noticing that even if its computation time can become high when the number of valves approaches the number of pipes in the network, it can still be considered acceptable since it is meant to run during the slow-paced design phase of the aqueduct.

7.4 Workload-Balanced and Loyalty-Enhanced Home Health Care

In the overall balance of environmental, societal and economic needs in which sustainability is rooted, a special place is occupied by health care, in particular in its modern, flexible (and thus more complex) forms. One of these variations is home health care: patients who need regular treatments, but do not require the specific facilities of a hospital, can stay at their homes where they are visited by nurses when needed. This solution is beneficial both for the National Health Service, because keeping patients in hospitals is expensive, and for the patients, who can stay in a familiar environment.

However organizing shifts for the nurses in such a scenario becomes way more complicated. A solution based on a rough partitioning of the served area in zones assigned to different nurses, as it is often done in practice, can easily lead to unbalanced workloads, which are a cause of stress and also strain on interpersonal relationships on the workplace. Moreover, the need to ease the most burdened workloads usually results in patients who are visited, during the same week, by two or more different nurses, which is usually perceived as unpleasant.

We addressed the home health care problem in Ferrara in cooperation with the local National Health Service agency. We devised CP models and search heuristics aimed at finding schedules and routing plans such that nurses' workloads are balanced and stay within certain bounds and patients are not visited by too many different nurses. The results achieved in tests on some past instances, proved to be significantly better than the ones obtained by hand by the nurses.

As a by-product, we also reduced workloads in terms of travel time: a better assignment of the patients to the nurses means that they avoid to travel for overly long distances and that their vehicles consume less fuel and reduce their polluting impact.

In the obtained solutions, nurses save approximately 3 hours per week of travel time, which means a significant saving of person-hours and of money, that can be devoted to provide an even better service or to serve a higher number of patients. In particular in a situation of economic crisis and cuts to the social welfare, this can be a relevant improvement.

References

[1] PAUL DICKINSON ET AL. **Cloud Computing – The IT Solution for the 21st Century**. Technical report, Carbon Disclosure Project, 2011. vii, 25, 26, 106

[2] CARLA P. GOMES. **Challenges for Constraint Reasoning and Optimization in Computational Sustainability**. In Gent (188), pages 2–4. 1

[3] MASSIMILIANO CATTAFI, EVELINA LAMMA, FABRIZIO RIGUZZI, AND SERGIO STORARI. **Incremental Declarative Process Mining**. In EDWARD SZCZERBICKI AND NGOC THANH NGUYEN, editors, *Smart Information and Knowledge Management*, **260** of *Studies in Computational Intelligence*, pages 103–127. Springer, 2010. 2, 106

[4] MARCO ALBERTI, MASSIMILIANO CATTAFI, FEDERICO CHESANI, MARCO GAVANELLI, EVELINA LAMMA, MARCO MONTALI, PAOLA MELLO, AND PAOLO TORRONI. **Integrating Abductive Logic Programming and Description Logics in a Dynamic Contracting Architecture**. In *ICWS*, pages 254–261. IEEE, 2009. 2

[5] MARCO ALBERTI, MASSIMILIANO CATTAFI, FEDERICO CHESANI, MARCO GAVANELLI, EVELINA LAMMA, PAOLA MELLO, MARCO MONTALI, AND PAOLO TORRONI. **A Computational Logic Application Framework for Service Discovery and Contracting**. *Int. J. Web Service Res.*, **8**(3):1–25, 2011. 2, 106

[6] MASSIMILIANO CATTAFI, MARCO GAVANELLI, MICHELA MILANO, AND PAOLO CAGNOLI. **Sustainable biomass power plant location in the Italian Emilia-Romagna region**. *ACM TIST*, **2**(4):33, 2011. 2, 107

[7] MASSIMILIANO CATTAFI, MARCO GAVANELLI, MADDALENA NONATO, STEFANO ALVISI, AND MARCO FRANCHINI. **Optimal placement of valves in a water distribution network with CLP(FD)**. *TPLP*, **11**(4-5):731–747, 2011. 3, 107

[8] K. L. CLARK. **Negation as Failure**. In *Logic and Databases*. Plenum Press, 1978. 7

[9] MICHAEL GELFOND AND VLADIMIR LIFSCHITZ. **The Stable Model Semantics for Logic Programming**. In *Proceedings of the Fifth International Conference and Symposium on Logic Programming*, pages 1070–1080, 1988. 7

[10] A. VAN GELDER, K. A. ROSS, AND J. S. SCHLIPF. **The Well-founded Semantics for General Logic Programs**. *Journal of the ACM*, **38**(3):620–650, 1991. 7

[11] L. DE RAEDT AND L. DEHASPE. **Clausal Discovery.** *Machine Learning*, **26**(2-3):99–146, 1997. 7

[12] S. MUGGLETON AND L. DE RAEDT. **Inductive Logic Programming: Theory and Methods**. *Journal of Logic Programming*, **19/20**:629–679, 1994. 7, 36

[13] L. DE RAEDT AND W. VAN LAER. **Inductive constraint logic**. In *Proceedings of the 6th Conference on Algorithmic Learning Theory*, **997** of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 1995. 9, 36, 42

[14] **ICL Manual**. Available at: http://www.cs.kuleuven.be/∼ml/ACE/Doc/ACEuser.pdf. 11

[15] HILDE ADÉ, BART MALFAIT, AND LUC DE RAEDT. **RUTH: an ILP Theory Revision System**. In ZBIGNIEW W. RAS AND MARIA ZEMANKOVA, editors, *Methodologies for Intelligent Systems, 8th International Symposium, ISMIS '94, Charlotte, North Carolina, USA, October 16-19, 1994, Proceedings*, **869** of *Lecture Notes in Computer Science*, pages 336–345. Springer, 1994. 13

[16] BRADLEY L. RICHARDS AND RAYMOND J. MOONEY. **Automated Refinement of First-Order Horn-Clause Domain Theories**. *Machine Learning*, **19**(2):95–131, 1995. 13

[17] FLORIANA ESPOSITO, GIOVANNI SEMERARO, NICOLA FANIZZI, AND STEFANO FERILLI. **Multistrategy Theory Revision: Induction and Abduction in INTHELEX**. *Machine Learning*, **38**(1-2):133–156, 2000. 13, 36

[18] A. C. KAKAS, R. A. KOWALSKI, AND F. TONI. **Abductive Logic Programming**. *Journal of Logic and Computation*, **2**(6):719–770, 1993. 13, 15

[19] M. ALBERTI, F. CHESANI, M. GAVANELLI, E. LAMMA, P. MELLO, AND P.TORRONI. **Verifiable agent interaction in abductive logic programming: The SCIFF framework**. *ACM Trans. Comput. Log.*, **9**(4), 2008. 13, 36

[20] T. H. FUNG AND R. A. KOWALSKI. **The IFF proof procedure for abductive logic programming**. *Journal of Logic Programming*, **33**(2):151–165, November 1997. 13

[21] KRZYSZTOF R. APT AND ROLAND N. BOL. **Logic Programming and Negation: A Survey**. *Journal of Logic Programming*, **19/20**:9–71, 1994. 15

[22] MICHELE LOMBARDI AND MICHELA MILANO. **Constraint Based Scheduling to Deal with Uncertain Durations and Self-Timed Execution**. In DAVID COHEN, editor, *CP*, **6308** of *Lecture Notes in Computer Science*, pages 383–397. Springer, 2010. 19

[23] JEAN-NOËL MONETTE, YVES DEVILLE, AND PASCAL VAN HENTENRYCK. **Just-In-Time Scheduling with Constraint Programming**. In ALFONSO GEREVINI, ADELE E. HOWE, AMEDEO CESTA, AND IOANNIS REFANIDIS, editors, *ICAPS*. AAAI, 2009. 19

[24] AGOSTINO DOVIER, ANDREA FORMISANO, AND ENRICO PONTELLI. **An Investigation of Multi-Agent Planning in CLP**. *Fundam. Inform.*, **105**(1-2):79–103, 2010. 19

[25] HELMUT SIMONIS. **A Hybrid Constraint Model for the Routing and Wavelength Assignment Problem**. In Gent (188), pages 104–118. 19

[26] HADRIEN CAMBAZARD, EMMANUEL HEBRARD, BARRY O'SULLIVAN, AND ALEXANDRE PAPADOPOULOS. **Local search and constraint programming for the post enrolment-based course timetabling problem**. *Annals of Operations Research*, **194**:111–135, 2012. 10.1007/s10479-010-0737-7. 19

[27] EVELINA LAMMA, PAOLA MELLO, MICHELA MILANO, RITA CUCCHIARA, MARCO GAVANELLI, AND MASSIMO PICCARDI. **Constraint Propagation and Value Acquisition: Why we should do it Interactively**. In THOMAS DEAN, editor, *IJCAI*, pages 468–477. Morgan Kaufmann, 1999. 19

[28] ALESSANDRO DAL PALÙ, AGOSTINO DOVIER, FEDERICO FOGOLARI, AND ENRICO PONTELLI. **Exploring Protein Fragment Assembly Using CLP**. In TOBY WALSH, editor, *IJCAI*, pages 2590–2595. IJCAI/AAAI, 2011. 19

[29] EDWARD P.K. TSANG. *Foundation of Constraint Satisfaction*. Academic Press, 1993. 19

[30] ALAN K. MACKWORTH. **Consistency in Networks of Relations**. *Artificial Intelligence*, **8**:99–118, 1977. 20

[31] UGO MONTANARI. **Networks of Constraints: Fundamental Properties and Applications to Picture Processing**. *Information Sciences*, **7**:95–132, 1974. 21

[32] EUGENE C. FREUDER. **Synthesizing Constraint Expressions**. *Communication of the ACM*, **21**(11):958–966, 1978. 21

[33] EUGENE C. FREUDER. **A Sufficient Condition for Backtrack Free Search**. *Communication of the ACM*, **29**(1):24–32, 1982. 21

[34] FAHIEM BACCHUS AND PETER VAN BEEK. **On the Conversion between Non-Binary and Binary Constraint Satisfaction Problems**. In *National Conference on Artifical Intelligence (AAAI-98)*, pages 310–318, Madison, Wisconsin, USA, July 26-30 1998. AAAI Press / The MIT Press. 21

[35] SOLOMON W. GOLOMB AND LEONARD D. BAUMERT. **Backtrack Programming**. *Journal of the ACM*, **12**(4):516–524, 1965. 21

[36] ROBERT M. HARALICK AND GORDON L. ELLIOTT. **Increasing tree search efficiency for constraint satisfaction problems**. *Artificial Intelligence*, **14**(3):263–313, October 1980. 21

[37] E.L. LAWLER AND D.E. WOOD. **Branch-and-Bound Methods: a Survey**. *Operations Research*, **14(4)**:699–719, 1966. 22

[38] J. JAFFAR AND M.J. MAHER. **Constraint Logic Programming: a Survey**. *Journal of Logic Programming*, **19-20**:503–582, 1994. 23

[39] ROBERT KOWALSKI. *Logic for Problem Solving*. North-Holland, New York, Amsterdam, Oxford, 1979. 23

[40] JOXAN JAFFAR AND JEAN-LOUIS LASSEZ. **Constraint Logic Programming**. In *Conference Record 14th Annual ACM Symposium on Principles of Programming Languages*, pages 111–119, Munich, Germany, January 21–23 1987. ACM SIGACT/SIGPLAN. 24

[41] JOXAN JAFFAR, SPIRO MICHAYLOV, PETER J. STUCKEY, AND ROLAND H.C. YAP. **The CLP(R) Language and System**. *ACM Transaction on Programming Language and Systems*, **14**(3):339–395, 1992. 24

[42] J. JAFFAR, M.J. MAHER, K. MARRIOTT, AND P.J. STUCKEY. **The semantics of constraint logic programs**. *Journal of Logic Programming*, **37(1-3)**:1–46, 1998. 24

[43] ABDERRAHAMANE AGGOUN, DAVID CHAN, PIERRE DUFRESNE, EAMON FALVEY, HUGH GRANT, WARWICK HARVEY, ALEXANDER HEROLD, GEOFFREY MACARTNEY, MICHA MEIER, DAVID MILLER, SHYAM MUDAMBI, STEFANO NOVELLO,

BRUNO PEREZ, EMMANUEL VAN ROSSUM, JOACHIM SCHIMPF, KISH SHEN, PERIKLIS ANDREAS TSAHAGEAS, AND DOMINIQUE HENRY DE VILLENEUVE. *ECL^{i}PS^e User Manual, Release 5.2*. IC-Parc, Imperial College, London, UK, 2001. 24

[44] **SICStus Prolog user manual, Release 3.11.0**, October 2003. `http://www.sics.se/isl/sicstus/`. 24, 34

[45] MEHMET DINCBAS, PASCAL VAN HENTENRYCK, HELMUT SIMONIS, ABDERRAHMANE AGGOUN, THOMAS GRAF, AND FRANÇOISE BERTHIER. **The Constraint Logic Programming Language CHIP**. In *Proceedings of the International Conference on Fifth Generation Computer System*, pages 693–702, Tokyo, Japan, November 28–December 2 1988. OHMSHA Ltd. Tokyo and Springer-Verlag. 24

[46] KISH SHEN AND JOACHIM SCHIMPF. **Eplex: Harnessing Mathematical Programming Solvers for Constraint Logic Programming**. In PETER VAN BEEK, editor, *Principles and Practice of Constraint Programming - CP 2005*, **3709** of *Lecture Notes in Computer Science*, pages 622–636. Springer Berlin / Heidelberg, 2005. 10.1007/1156475146. 24

[47] PIKE RESEARCH. **Cloud Computing Energy Efficiency**. Technical report, 2010. 26, 106

[48] W3C. **Web Services Description Language (WSDL) 1.1**. `http://www.w3.org/TR/wsdl`, March 2001. 26

[49] MARCO ALBERTI, FEDERICO CHESANI, MARCO GAVANELLI, EVELINA LAMMA, PAOLA MELLO, MARCO MONTALI, AND PAOLO TORRONI. **Web service contracting: specification and reasoning with SCIFF**. In ENRICO FRANCONI, MICHAEL KIFER, AND WOLFGANG MAY, editors, *ESWC*, **4519** of *LNAI*, 2007. 27, 28, 30

[50] E. LAMMA, P. MELLO, F. RIGUZZI, AND S. STORARI. **Applying Inductive Logic Programming to Process Mining**. In *Inductive Logic Programming, 17th International Conference*, number 4894 in Lecture Notes in Artificial Intelligence, pages 132–146, Heidelberg, Germany, 2008. Springer. 27, 35, 36, 38, 39, 40, 41, 42, 106

[51] **W3C Recommendation: OWL, Web Ontology Language**. `http://www.w3.org/TR/owl-guide/`, 2004. 30

[52] CARSTEN LUTZ. **Description Logic Resources**. `dl.kr.org`, 2008. 30

[53] NATALYA FRIDMAN NOY, MICHAEL SINTEK, STEFAN DECKER, MONICA CRUBÉZY, RAY W. FERGERSON, AND MARK A. MUSEN. **Creating Semantic Web Contents with Protégé-2000**. *IEEE Int. Systems*, **16**(2):60–71, 2001. 31

[54] U. HUSTADT, B. MOTIK, AND U. SATTLER. **Reducing $\mathcal{SHIQ}^-$ Description Logic to Disjunctive Datalog Programs**. In D. DUBOIS, C. WELTY, AND M.-A. WILLIAMS, editors, *KR2004*. 33

[55] DENNY VRANDEČIĆ, PETER HAASE, PASCAL HITZLER, YORK SURE, AND RUDI STUDER. **DLP-An introduction**. Tech.Rep., Univ. Karlsruhe, 2006. 33

[56] BIJAN PARSIA AND EVREN SIRIN. **Pellet: An OWL DL Reasoner**. In FRANK VAN HARMELEN, editor, *ISWC 2004*, 2004. 33

[57] M. DUMAS, M. REICHERT, AND M. SHAN, editors. *Business Process Management, 6th International Conference, BPM 2008*, **5240** of *LNCS*. Springer, 2008. 35

[58] W. M. P. VAN DER AALST, B. F. VAN DONGEN, J. HERBST, L. MARUSTER, G. SCHIMM, AND A. J. M. M. WEIJTERS. **Workflow mining: A survey of issues and approaches.** *Data Knowl. Eng.*, **47**(2):237–267, 2003. 35, 37, 38

[59] M. ALBERTI, M. GAVANELLI, E. LAMMA, P. MELLO, AND P. TORRONI. **An Abductive Interpretation for Open Societies**. In A. CAPPELLI AND F. TURINI, editors, *Proceedings of the 8th Congress of the Italian Association for Artificial Intelligence (AI*IA 2003)*, **2829** of *LNAI*. Springer Verlag, 2003. 36

[60] D. GEORGAKOPOULOS, M. F. HORNICK, AND A. P. SHETH. **An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure.** *Distributed and Parallel Databases*, **3**(2):119–153, 1995. 37

[61] R. AGRAWAL, D. GUNOPULOS, AND F. LEYMANN. **Mining Process Models from Workflow Logs.** In *Proceedings of the 6th International Conference on Extending Database Technology, EDBT'98*, **1377** of *LNCS*, pages 469–483. Springer, 1998. 37, 38, 50

[62] G. GRECO, A. GUZZO, L. PONTIERI, AND D. SACCÀ. **Discovering Expressive Process Models by Clustering Log Traces.** *IEEE Trans. Knowl. Data Eng.*, **18**(8):1010–1027, 2006. 37, 38, 50

[63] M. PESIC AND W. M. P. VAN DER AALST. **A Declarative Approach for Flexible Business Processes Management**. In *2006 International Business Process Management Workshops*, **4103** of *LNCS*, pages 169–180. Springer, 2006. 38

[64] WIL M. P. VAN DER AALST, TON WEIJTERS, AND LAURA MARUSTER. **Workflow Mining: Discovering Process Models from Event Logs.** *IEEE Trans. Knowl. Data Eng.*, **16**(9):1128–1142, 2004. 38, 50

[65] B. F. VAN DONGEN AND W. M. P. VAN DER AALST. **Multi-phase Process Mining: Building Instance Graphs.** In *23rd International Conference on Conceptual Modeling*, **3288** of *LNCS*, pages 362–376. Springer, 2004. 38, 50

[66] E. LAMMA, P. MELLO, M. MONTALI, F. RIGUZZI, AND S. STORARI. **Inducing Declarative Logic-Based Models from Labeled Traces**. In *Proceedings of the 5th International Conference on Business Process Management*, number 4714 in Lecture Notes in Computer Science, pages 344–359, Heidelberg, Germany, 2007. Springer. 38

[67] FEDERICO CHESANI, EVELINA LAMMA, PAOLA MELLO, MARCO MONTALI, FABRIZIO RIGUZZI, AND SERGIO STORARI. **Exploiting Inductive Logic Programming Techniques for Declarative Process Mining**. *LNCS Transactions on Petri Nets and Other Models of Concurrency, ToPNoC II*, **5460**:278–295, 2009. 38

[68] KRZYSZTOF R. APT AND MARC BEZEM. **Acyclic Programs.** *New Generation Comput.*, **9**(3/4):335–364, 1991. 40

[69] M. PESIC, H. SCHONENBERG, AND W. M. P. VAN DER AALST. **DECLARE: Full Support for Loosely-Structured Processes**. In *11th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2007)*, pages 287–300. IEEE Computer Society, 2007. 45

[70] A. CHAVEZ AND P. MAES. **Kasbah: An agent marketplace for buying and selling goods**. In *Proceedings of the First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM-96)*, pages 75–90, London, April 1996. 47

[71] **SOCS Protocol Repository**. Available at: http://edu59.deis.unibo.it:8079/SOCSProtocolsRepository/jsp/index.jsp. 48

[72] MICHAEL KIFER, GEORG LAUSEN, AND JAMES WU. **Logical Foundations of Object Oriented and Frame Based Languages**. *Journal of the Association for Computing Machinery*, **42**(4):741–843, May 1995. 49

[73] **FLORA-2: An Object-Oriented Knowledge Base Language**. `http://flora.sourceforge.net/`. 49

[74] D. ROMAN, U. KELLER, H. LAUSEN, J. DE BRUIJN, R. LARA, M.STOLLBERG, A.POLLERES, C.FEIER, C.BUSSLER, AND D. FENSEL. **Web Service Modeling Ontology**. *Appl. Ontology*, **1(1)**, 2005. 49

[75] J. DE BRUIJN. *Semantic Web Language Layering with Ontologies, Rules, and Meta-Modeling*. PhD thesis, Faculty of Mathematics, Computer Science and Physics of the University of Innsbruck, 2008. 49

[76] MICHAEL KIFER, RUBEN LARA, AXEL POLLERES, CHANG ZHAO, UWE KELLER, HOLGER LAUSEN, AND DIETER FENSEL. **A Logical Framework for Web Service Discovery**. In D. MARTIN, R. LARA, AND T. YAMAGUCHI, editors, *SWS*, **119** of *CEUR Workshop Proc.*, 2004. 50

[77] RICCARDO ROSATI. **DL+log: Tight Integration of Description Logics and Disjunctive Datalog**. In PATRICK DOHERTY, JOHN MYLOPOULOS, AND CHRISTOPHER A. WELTY, editors, *Proceedings, Tenth International Conference on Principles of Knowledge Representation and Reasoning, Lake District of the United Kingdom, June 2-5, 2006*, pages 68–78. AAAI Press, 2006. 50

[78] BORIS MOTIK AND RICCARDO ROSATI. **A Faithful Integration of Description Logics with Logic Programming**. In MANUELA M. VELOSO, editor, *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, pages 477–482, 2007. 50

[79] VLADIMIR LIFSCHITZ. **Nonmonotonic Databases and Epistemic Queries**. In JOHN MYLOPOULOS AND RAYMOND REITER, editors, *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 381–386, Sidney, Australia, august 1991. Morgan Kaufmann. 50

[80] MATTHIAS KNORR, JOSÉ JÚLIO ALFERES, AND PASCAL HITZLER. **Towards Tractable Local Closed World Reasoning for the Semantic Web**. In JOSÉ NEVES, MANUEL FILIPE SANTOS, AND JOSÉ MACHADO, editors, *Progress in Artificial Intelligence, 13th Portuguese Conference on Aritficial Intelligence, EPIA 2007, Workshops: GAIW, AIASTS, ALEA, AMITA, BAOSW, BI, CMBSB, IROBOT, MASTA, STCS, and TEMA, Guimarães, Portugal, December 3-7, 2007, Proceedings*, **4874** of *Lecture Notes in Computer Science*, pages 3–14. Springer, 2007. 50

[81] J. DESEL AND T. ERWIN. **Hybrid specifications: looking at workflows from a runtime perspective**. *Int. J. Computer System Science & Engineering*, **15**(5):291 – 302, 2000. 50, 51

[82] H. M. FERREIRA AND D. R. FERREIRA. **An Integrated Life Cycle for Workflow Management Based on Learning and Planning**. *Int. J. Cooperative Inf. Syst.*, **15**(4):485–505, 2006. 50

[83] S. GOEDERTIER. *Declarative techniques for modeling and mining business processes*. PhD thesis, Katholieke Universiteit Leuven, Faculteit Economie en Bedrijfswetenschappen, 2008. 51

[84] STEFAN WROBEL. **First Order Theory Refinement**. In LUC DE RAEDT, editor, *Advances in Inductive Logic Programming*, pages 14 – 33. IOS Press, Amsterdam, 1996. 51

[85] MANFRED REICHERT, STEFANIE RINDERLE-MA, AND PETER DADAM. **Flexibility in Process-Aware Information Systems**. *T. Petri Nets and Other Models of Concurrency*, **2**:115–135, 2009. 51

[86] BELA MUTSCHLER, MANFRED REICHERT, AND STEFANIE RINDERLE. **Analyzing the Dynamic Cost Factors of Process-Aware Information Systems: A Model-Based Approach**. In *CAiSE*, pages 589–603, 2007. 51

[87] **Delibera 6 del Comitato Interministeriale Prezzi, 29 aprile 1992 (Deliberation 6 of the inter-ministry committee for prices,** 29^{th} **of april 1992)**, 1992. 54

[88] D. FREPPAZ, R. MINCIARDI, M. ROBBA, M. ROVATTI, R. SACILE, AND A TARAMASSO. **Optimizing Forest biomass exploitation for energy supply at regional level**. *Biomass and Bioenergy*, **26**:15–24, 2003. 54, 70, 107

[89] MAURIZIO BRUGLIERI AND LEO LIBERTI. **Optimal running and planning of a biomass-based energy production process**. *Energy Policy*, **36**(7):2430–2438, July 2008. 54, 69

[90] P. RECHE-LÓPEZ, N. RUIZ-REYES, S. GARCÍA GALÁN, AND F. JURADO. **Comparison of metaheuristic techniques to determine optimal placement of biomass power plants**. *Energy Conversion and Management*, **50**(8):2020 – 2028, 2009. 54, 69, 107

[91] ISABEL THOMAS. *The Pros and Cons of Biomass Power*. Rosen publishing, New York, 2008. 54

[92] EUROPEAN COMMISSION. **Integrated Pollution Prevention and Control Reference Document on Economics and Cross-Media Effects**, July 2006. Available at http://eippcb.jrc.es/reference/ecm.html. 59

[93] E.M.L. BEALE AND J.A. TOMLIN. **Special Facilities in a General Mathematical Programming System for Nonconvex Problems Using Ordered Sets of Variables**. In J. LAWRENCE, editor, *Proceedings of the 5th International Conference on Operations Research*, pages 447–454. Tavistock Publications, 1970. 60

[94] D. VOIVONTAS, D. ASSIMACOPOULOS, AND E.G. KOUKIOS. **Assessment of biomass potential for power production: a GIS based method**. *Biomass and Bioenergy*, **43**:101–112, 2001. 69

[95] C.P. MITCHELL. **New cultural treatments and yield optimisation**. *Biomass and Bioenergy*, **9**:11–34, 1995. 70

[96] C.P. MITCHELL, A.V. BRIDGWATER, D.J. STEVENS, A.J. TOFT, AND M.P. WATTERS. **Technoeconomic assessment of biomass to energy**. *Biomass and Bioenergy*, **9**:205–226, 1995. 70

[97] C.P. MITCHELL. **Development of decision support system for bioenergy applications**. *Biomass and Bioenergy*, **18**:265–278, 2000. 70

[98] J. NAGEL. **Determination of an economic energy supply structure based on biomass using a mixed-integer linear optimisation model**. *Ecological Engineering*, **16**:91–102, 2000. 70

[99] J. NAGEL. **Biomass in energy supply, especially in the state of Brandeburg, Germany**. *Ecological Engineering*, **16**:103–110, 2000. 70

[100] R.M. DE MOL, M.A.H JOGEMS, P. VAN BEEK, AND J.K. GIGLER. **Simulation and OPtimization of the logistic of biomass fuel collection**. *Journal of Agricoltural Science*, **45**:219–228, 1997. 70

[101] ALFRED A. KUEHN AND MICHAEL J. HAMBURGER. **A Heuristic Program for Locating Warehouses**. *Management Science*, **9**(4):643–666, 1963. 71

[102] M. L. BALINSKI. **Integer Programming: Methods, Uses, Computations**. *Management Science*, **12**(3):253–313, 1965. 71

[103] BASHEER M. KHUMAWALA. % bf An Efficient Branch and Bound Algorithm for the Warehouse Location Problem. *Management Science*, **18**(12):B–718–731, 1972. 71

[104] DONALD ERLENKOTTER. **A Dual-Based Procedure for Uncapacitated Facility Location**. *Operations Research*, **26**(6):992–1009, November-December 1978. 71

[105] A.M. GEOFFRION AND R. MCBRIDE. **Lagrangean Relaxation Applied to Capacitated Facility Location Problems**. *AIIE Transactions*, **10**(1):40–47, March 1978. 71

[106] C. H. AIKENS. **Facility location models for distribution planning**. *European Journal of Operational Research*, **22**(3):263 – 279, 1985. 71

[107] MARGARET L. BRANDEAU AND SAMUEL S. CHIU. **An overview of representative problems in location research**. *Management Science*, **35**(6):645–674, 1989. 71

[108] CHARLES S. REVELLE AND GILBERT LAPORTE. **The Plant Location Problem: New Models and Research Prospects**. *Operations Research*, **44**(6):864–874, 1996. 71

[109] S.H. OWEN AND M.S. DASKIN. **Strategic facility location: a review**. *European Journal of Operational Research*, **111**(3):423–447, 1998. 71

[110] ANDREAS KLOSE AND ANDREAS DREXL. **Facility location models for distribution system design**. *European Journal of Operational Research*, **162**(1):4–29, 2005. 71

[111] M.S. DASKIN. *Network and discrete location - models, algorithms and applications*. Wiley, New York, 1995. 71

[112] DILEEP R. SULE. *Logistics of facility location and allocation*. CRC Press, January 2001. 71

[113] ZVI DREZNER AND HORST W. HAMACHER, editors. *Facility location: applications and theory*. Springer, Berlin, 2002. 71

[114] REZA ZANJIRANI FARAHANI AND MASOUD HEKMATFAR, editors. *Facility Location: Concepts, Models, Algorithms and Case Studies*. Springer, Berlin, 2009. 71

[115] MOSES CHARIKAR, SAMIR KHULLER, DAVID M. MOUNT, AND GIRI NARASIMHAN. **Algorithms for facility location problems with outliers**. In S. RAO KOSARAJU, editor, *SODA '01: Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, pages 642–651, Philadelphia, PA, USA, 2001. Society for Industrial and Applied Mathematics. 71

[116] JACK BRIMBERG AND CHARLES REVELLE. **A bi-objective plant location problem: cost vs. demand served**. *Location Science*, **6**:121–135, 1998. 71

[117] KAJ HOLMBERG AND JONAS LING. **A Lagrangean heuristic for the facility location problem with staircase costs**. *European Journal of Operational Research*, **97**(1):63–74, February 1997. 71

[118] JENS WOLLENWEBER. **A multi-stage facility location problem with staircase costs and splitting of commodities: model, heuristic approach and application**. *OR Spectrum*, **30**(4):655–673, October 2008. 71

[119] WORLD WATER ASSESSMENT PROGRAMME (UNITED NATIONS). *Water for people, water for life: a joint report by the twenty-three UN agencies concerned with freshwater*. The United Nations world water development report. UNESCO Pub., 2003. 73

[120] CENTER FOR STRATEGIC AND INTERNATIONAL STUDIES. **Addressing Our Global Water Future**. Technical report, Sandia National Laboratories, 2005. 73

[121] HWANDON JUN AND G. V. LOGANATHAN. **Valve-Controlled Segments in Water Distribution Systems**. *Journal of Water Resources Planning and Management*, **133**(2):145–155, March/April 2007. 75, 77, 87

[122] ORAZIO GIUSTOLISI AND DRAGAN A. SAVIĆ. **Identification of Segments and Optimal Isolation Valve System Design in Water Distribution Networks**. *Urban Water Journal*, **7**(1):1–15, 2010. 75, 76, 87

[123] LUC N. VAN WASSENHOVE AND LUDO F. GELDERS. **Solving a bicriterion scheduling problem**. *European Journal of Operational Research*, **4**(1):42–48, 1980. 76

[124] CARMEN GERVET, YVES CASEAU, AND DENIS MONTAUT. **On Refining Ill-Defined Constraint Problems: A Case Study in Iterative Prototyping**. In *PACLP-99*, pages 255–275, London, 1999. 76

[125] MARCO GAVANELLI. **An Algorithm for Multi-Criteria Optimization in CSPs**. In FRANK VAN HARMELEN, editor, *ECAI 2002. Proceedings of the 15th European Conference on Artificial Intelligence*, pages 136–140, Lyon, France, July 21-26 2002. IOS Press. 76, 83

[126] STUART J. RUSSELL AND PETER NORVIG. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2 edition, 2003. 76, 77

[127] KRZYSZTOF R. APT AND MARK WALLACE. *Constraint Logic Programming using ECL^{i}PS^e*. Cambridge University Press, 2006. 76, 98

[128] K. MARRIOTT AND P.J. STUCKEY. **Semantics of Constraint Logic Programs with Optimization**. In H. AÏT-KACI, M. HANUS, AND J.J. MORENO-NAVARRO, editors, *ICLP Workshop: Integration of Declarative Paradigms*, pages 23–35, 1994. 78

[129] F. FAGES. **From Constraint Minimization to Goal Optimization in CLP Languages**. In E. FREUDER, editor, *CP'96*, **1118** of *LNCS*, 1996. 78

[130] S. PRESTWICH. **Three Implementations of Branch-and-Bound in CLP**. In *Proceedings of Fourth Compulog-Net Workshop on Parallelism and Implementation Technologies*, Bonn, September 1996. 79, 84

[131] FILIPPO FOCACCI, ANDREA LODI, AND MICHELA MILANO. **Cost-Based Domain Filtering**. In JOXAN JAFFAR, editor, *CP*, **1713** of *Lecture Notes in Computer Science*, pages 189–203. Springer, 1999. 81

[132] FILIPPO FOCACCI, ANDREA LODI, AND MICHELA MILANO. **A Hybrid Exact Algorithm for the TSPTW**. *INFORMS Journal on Computing*, **14**(4):403–417, fall 2002. 81

[133] JOHN HOPCROFT AND ROBERT TARJAN. **Algorithm 447: efficient algorithms for graph manipulation**. *Communications of the ACM*, **16**:372–378, June 1973. 83

[134] ECLIPSE DOCUMENTATION. *graph_algorithms library*. 83

[135] ORAZIO GIUSTOLISI AND DRAGAN A. SAVIĆ. **Optimal design of isolation valve system for water distribution networks**. In J.E. VAN ZYL, A.A. ILEMOBADE, AND H.E. JACOBS, editors, *Proceedings of the 10th Annual Water Distribution Systems Analysis Conference WDSA2008*, 2008. 83, 84, 85

[136] MASSIMILIANO CATTAFI AND MARCO GAVANELLI. **A CLP(FD) program for the Optimal Placement of Valves in a Water Distribution Network**. Source code available at `http://www.ing.unife.it/docenti/MarcoGavanelli/software/vp/`, April 2011. 83

[137] J.-J. KAO AND P.-H. LI. **A segment-based optimization model for water pipeline replacement**. *J. Am. Water Works Assoc.*, **99**(7):83–95, 2007. 87

[138] E. CREACO, M. FRANCHINI, AND S. ALVISI. **Optimal placement of isolation valves in water distribution systems based on valve cost and weighted average demand shortfall**. *Journal of Water Resources Planning and Management*, **24**(15):4317–4338, 2010. 87

[139] MICHAEL R. GAREY AND DAVID S. JOHNSON. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990. 87

[140] BRIAN KERNIGAN AND SHEN LIN. **An efficient heuristic procedure for partitioning graphs**. *Bell Systems Technical Journal*, **49**:291–307, 1970. 87

[141] C. M. FIDUCCIA AND R. M. MATTHEYSES. **A linear-time heuristic for improving network partitions**. In *Proceedings of the 19th Design Automation Conference*, DAC '82, pages 175–181, Piscataway, NJ, USA, 1982. IEEE Press. 88

[142] FAN R. K. CHUNG. *Spectral Graph Theory*. 1994. 88

[143] M. FIEDLER. **Algebraic connectivity of graphs**. *Czechoslovak Mathematical Journal*, **23**(98):298–305, 1973. 88

[144] BRUCE HENDRICKSON AND ROBERT LELAND. **An improved spectral graph partitioning algorithm for mapping parallel computations**. *SIAM Journal on Scientific Computing*, **16**:452–469, March 1995. 88

[145] A. SPIELMAN AND S.-H. TENG. **Spectral Partitioning Works: Planar graphs and finite element meshes. Technical Report. University of Berkeley**, 1996. 88

[146] BRUCE HENDRICKSON AND ROBERT LELAND. **A multilevel algorithm for partitioning graphs**. In *Proceedings of the 1995 ACM/IEEE conference on Supercomputing (CDROM)*, Supercomputing '95, New York, NY, USA, 1995. ACM. 88

[147] GEORGE KARYPIS AND VIPIN KUMAR. **A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs**. *SIAM J. Sci. Comput.*, **20**:359–392, December 1998. 88

[148] RICHARD J. LIPTON AND ROBERT ENDRE TARJAN. **A Separator Theorem for Planar Graphs**. *SIAM Journal on Applied Mathematics*, **36**(2):177–189, 1979. 88

[149] REINHARD PICHLER, STEFAN RÜMMELE, AND STEFAN WOLTRAN. **Multicut Algorithms via Tree Decompositions**. In TIZIANA CALAMONERI AND JOSEP DÍAZ, editors, *Algorithms and Complexity, 7th International Conference, CIAC 2010*, **6078** of *Lecture Notes in Computer Science*, pages 167–179. Springer, 2010. 88

[150] H. SIMONIS. **Models for Global Constraint Applications**. *Constraints*, **12**:63–92, 2007. 94

[151] RICHARD E. KORF. **An Improved Algorithm for Optimal Bin Packing**. In GEORG GOTTLOB AND TOBY WALSH, editors, *IJCAI*, pages 1252–1258. Morgan Kaufmann, 2003. 95

[152] ALEX S. FUKUNAGA AND RICHARD E. KORF. **Bin completion algorithms for multi-container packing, knapsack, and covering problems**. *J. Artif. Int. Res.*, **28**:393–429, March 2007. 95

[153] PAUL SHAW. **A Constraint for Bin Packing**. In MARK WALLACE, editor, *CP*, **3258** of *Lecture Notes in Computer Science*, pages 648–662. Springer, 2004. 95

[154] DONALD E. KNUTH. **Two notes on notation**. *Am. Math. Monthly*, **99**:403–422, May 1992. 95

[155] JEAN-CHARLES RÉGIN. **A Filtering Algorithm for Constraints of Difference in CSPs**. In BARBARA HAYES-ROTH AND RICHARD E. KORF, editors, *AAAI*, pages 362–367. AAAI Press / The MIT Press, 1994. 96

[156] WILLEM JAN VAN HOEVE. **The alldifferent Constraint: A Survey**. *CoRR*, **cs.PL/0105015**, 2001. 96

[157] CHRISTIAN BESSIÈRE, EMMANUEL HEBRARD, BRAHIM HNICH, ZEYNEP KIZILTAN, AND TOBY WALSH. **Filtering Algorithms for the NValueConstraint**. *Constraints*, **11**(4):271–293, 2006. 96

[158] EMMANUEL HEBRARD, DÁNIEL MARX, BARRY O'SULLIVAN, AND IGOR RAZGON. **Soft Constraints of Difference and Equality**. *J. Artif. Intell. Res. (JAIR)*, **41**:97–130, 2011. 96

[159] WILLEM JAN VAN HOEVE. **A Hyper-Arc Consistency Algorithm for the Soft Alldifferent Constraint**. *CoRR*, **cs.PL/0407043**, 2004. 96

[160] D.L. APPLEGATE. *The traveling salesman problem: a computational study*. Princeton series in applied mathematics. Princeton University Press, 2006. 96

[161] N. BELDICEANU AND E. CONTEJEAN. **Introducing global constraints in CHIP**. *Mathematical and Computer Modelling*, **20**(12):97 – 123, 1994. 97

[162] LATIFE GENÇ KAYA AND J. N. HOOKER. **A filter for the circuit constraint**. In *Proceedings of the 12th international conference on Principles and Practice of Constraint Programming*, CP'06, pages 706–710, Berlin, Heidelberg, 2006. Springer-Verlag. 97

[163] Y. CASEAU AND F. LABURTHE. **Solving small TSPs with constraints**. In *ICLP*, 1997. 97, 104

[164] DAVID APPLEGATE, ROBERT E. BIXBY, VASEK CHVÁTAL, AND WILLIAM J. COOK. **Cutting planes and the traveling salesman problem (abstract only)**. In DAVID B. SHMOYS, editor, *SODA*, page 429. ACM/SIAM, 2000. 97

[165] DANIEL GUIMARANS, ROSA HERRERO, DANIEL RIERA, ANGEL A. JUAN, AND JUAN JOSÉ RAMOS. **Combining probabilistic algorithms, Constraint Programming and Lagrangian Relaxation to solve the Vehicle Routing Problem**. *Ann. Math. Artif. Intell.*, **62**(3-4):299–315, 2011. 97

[166] ROBERT M. HARALICK AND GORDON L. ELLIOTT. **Increasing Tree Search Efficiency for Constraint Satisfaction Problems**. *Artif. Intell.*, **14**(3):263–313, 1980. 97

[167] H. KAUTZ, E. HORVITZ, Y. RUAN, C. GOMES, AND B. SELMAN. **Dynamic Restart Policies**. In *AAAI/IAAI*, 2002. 97

[168] TUDOR HULUBEI AND BARRY O'SULLIVAN. **The Impact of Search Heuristics on Heavy-Tailed Behaviour**. *Constraints*, **11**(2-3):159–178, 2006. 97

[169] M. LUBY, A. SINCLAIR, AND D. ZUCKERMAN. **Optimal Speedup of Las Vegas Algorithms**. *Inf. Process. Lett.*, 1993. 97

[170] C. SINZ AND M. ISER. **Problem-Sensitive Restart Heuristics for the DPLL Procedure**. In O. KULLMANN, editor, *SAT*, **5584** of *LNCS*. Springer, 2009. 97

[171] P. SHAW. **Using Constraint Programming and Local Search Methods to solve Vehicle Routing Problems**. In *CP98*, LNCS. Springer-Verlag, 1998. 98

[172] P. VAN HENTENRYCK AND L. MICHEL. *Constraint-based local search*. MIT Press, 2005. 100, 104

[173] S. BERTELS AND T. FAHLE. **A hybrid setup for a hybrid scenario: combining heuristics for the home health care problem**. *Computers & OR*, **33**(10), 2006. 103

[174] JÖRG STEEG AND MICHAEL SCHRÖDER. **A Hybrid Approach to Solve the Periodic Home Health Care Problem**. In JÖRG KALCSICS AND STEFAN NICKEL, editors, *Operations Research Proceedings 2007*, **2007** of *Operations Research Proceedings*, pages 297–302. Springer Berlin Heidelberg, 2008. 103

[175] P. EVEBORN, M. RÖNNQVIST, H. EINARSDÓTTIR, M. EKLUND, K. LIDÉN, AND M. ALMROTH. **Operations Research Improves quality and efficiency in Home Care**. *Interfaces*, 2009. 104

[176] ROBERTO BALDACCI, PAOLO TOTH, AND DANIELE VIGO. **Exact algorithms for routing problems under vehicle capacity constraints**. *Annals of OR*, **175**, 2010. 104

[177] DAVID PISINGER AND STEFAN ROPKEA. **A general heuristic for vehicle routing problems**. *Computers & Operations Research*, **34**(8), 2007. 104

[178] PABLO GARRIDO, CARLOS CASTRO, AND ERIC MONFROY. **Towards a Flexible and Adaptable Hyperheuristic Approach for VRPs**. In HAMID R. ARABNIA, DAVID DE LA FUENTE, AND JOSÉ ANGEL OLIVAS, editors, *IC-AI*. CSREA Press, 2009. 104

[179] ERIC MONFROY, FRÉDÉRIC SAUBION, AND TONY LAMBERT. **On Hybridization of Local Search and Constraint Propagation**. In BART DEMOEN AND VLADIMIR LIFSCHITZ, editors, *ICLP*, 2004. 104

[180] M. WALLACE. **Hybrid Algorithms in Constraint Programming**. In F. AZEVEDO, P. BARAHONA, F. FAGES, AND F. ROSSI, editors, *CSCLP*, **4651** of *LNCS*, 2006. 104

[181] L.-M. ROUSSEAU, M. GENDREAU, AND G. PESANT. **Using Constraint-Based operators to solve the Vehicle Routing Problem with Time Windows**. *J. of Heuristics*, **8**, 2002. 104

[182] P. KILBY, P. PROSSER, AND P. SHAW. **A Comparison of Traditional and Constraint-based Heuristic Methods on VRPs with Side Constraints**. *Constraints*, 2000. 104

[183] G. PESANT, M. GENDREAU, J-Y. POTVIN, AND J-M. ROUSSEAU. **An exact Constraint Logic Programming algorithm for the TSP with Time Windows**. *Transp. Sci.*, 1998. 104

[184] F. FOCACCI, A. LODI, AND M. MILANO. **A hybrid exact algorithm for the TSPTW**. *INFORMS Journal on Computing*, **14**(4):403–417, 2002. 104

[185] PIERRE SCHAUS, YVES DEVILLE, PIERRE DUPONT, AND JEAN-CHARLES RÉGIN. **The Deviation Constraint**. In PASCAL VAN HENTENRYCK AND LAURENCE A. WOLSEY, editors, *CPAIOR*, **4510** of *Lecture Notes in Computer Science*, pages 260–274. Springer, 2007. 104

[186] GILLES PESANT AND JEAN-CHARLES RÉGIN. **SPREAD: A Balancing Constraint Based on Statistics**. In PETER VAN BEEK, editor, *CP*, **3709** of *Lecture Notes in Computer Science*, pages 460–474. Springer, 2005. 104

[187] EUROPEAN COMMISSION. **Horizon 2020, the framework programme for research and innovation**. http://ec.europa.eu/research/horizon2020/. 105

[188] IAN P. GENT, editor. *Principles and Practice of Constraint Programming - CP 2009, 15th International Conference, CP 2009, Lisbon, Portugal, September 20-24, 2009, Proceedings*, **5732** of *Lecture Notes in Computer Science*. Springer, 2009. 109, 111