Constraint Satisfaction: Algorithms, Complexity Results, and Applications

by
Francesco Lupia

University of Calabria
DIMES - Dipartimento di Ingegneria Informatica, Modellistica, Elettronica e Sistemistica
PhD Program in Systems and Computer Engineering

Professor Francesco Scarcello, Advisor
Associate Professor Gianluigi Greco, Co-advisor

To my family

ACKNOWLEDGEMENTS

First, I want to thank my advisors Francesco Scarcello and Gianluigi Greco. I believe they are the best advisors I could ever desire. They complete me in many aspects. Francesco has been an endless source of inspiration for me. His advice goes well beyond the research and academic context to cover many aspects of my life. Perhaps, the greatest thing that I can think of is that he constantly push my abilities to think and reason about things in a meticulous way. He is always willing to take time out of his busy days to help with important problems and all the unexpected issues that come up. Thanks to Francesco I met Gianluigi. He is truly another source of countless motivations. He constantly impresses me at how good he is in everything he does and for his ability to be in involved in many different things at the same time. Francesco and Gianlugi are constantly teaching me to take a simple idea, to find generalizations and implications and to push it to the limits. But this is only the beginning.

I thank Walter Lucia for sparking my initial interest for research and for all the interactions that we had in the last 10 years. I am also indebted with Antonella Guzzo and Luigi Pontieri for their priceless help in the last three years. Now let me thank my office colleagues Valeria Fionda and Carmine Dodaro with whom during my "PhD days" I had a huge number of valuable discussions. A special thank you goes to Luca Ghionna for his valuable feedback on some aspects of this dissertation. I especially thank Domenico Saccà who worked behind the scenes to support me

with the fundings and I must say thanks to Nicola Leone for allowing me to stay and feel part of the department of Mathematics and Computer Science while I was working with Gianluigi. Moreover I want to thank all the friends from the DIMES and Mathematics and Computer Science departments. In the last three years, it was not all about research and work. I especially thank Massimo and Alessandro for being always present. And for the same reason Francesco deserves a big thank you and our friend Antonio. I also special thank my other two cousins Pasquale and Salvatore. Finally, I am who I am thanks to my mother, my father, my sister Valentina and my grandmother Maria and grandfather Pasquale. I wish I will have the opportunity to return their love and support.

TABLE OF CONTENTS

LIST OF FIGURES

Figure

LIST OF TABLES

Table

ABSTRACT

Constraint Satisfaction: Algorithms, Complexity Results, and Applications

by
Francesco Lupia

A fundamental problem in the field of Artificial Intelligence and related disciplines, in particular Database theory, is the *constraint satisfaction problem (or CSP)* which comes as a unifying framework to express a wide spectrum of computational problems. Examples include graph colorability, planning, and database queries. The goal is either to find one solution, to enumerate all solutions, or counting them. As a very general problem, it comes with no surprise that in most settings CSPs are hard to solve. Indeed considerable effort has been invested by the scientific community to shed light on the computational issues of this problem, with the objective of identifying easy instances (also called islands of tractability) and exploiting the knowledge derived from their solution to help solving the harder ones.

My thesis investigates the role that structural properties play in the computational aspects of CSPs, describes algorithms to exploit such properties, and provides a number of specific tools to solve efficiently problems arising in database theory, game theory, and process mining.

CHAPTER 1

Introduction

Constraint satisfaction is a very general framework to express a wide range of computational problems. Examples include graph colorability, planning, satisfiability, scheduling, and database queries. Being such a general framework, we well expect computational issues to come up. Indeed, a great effort has been invested by the scientific community to study the complexity of this general problem with the objective to identify classes of easy instances known as "islands of tractability". It is well known that the structure of the constraint satisfaction problem (*short: CSP*), which is represented by a constraint (hyper)graph, plays a significant role in the study of its computational complexity. Indeed, while such problems are intractable in general, whenever the input has some kind of structures (for example, the constraint graph is a tree), we may compute a solution efficiently, e.g., via dynamic programming. Moreover, by precisely understanding the structure of easy problems, we may also be able to approximate hard problems by manipulating their structure in a suitable way. Unlike most approaches in the CSP research community that focus on the problem of computing just one solution to CSPs, we are mainly interested here in finding all solutions or in counting the number of solutions efficiently (i.e., without actually computing them). Database queries are an important domain of

application of our techniques. Indeed, even structurally simple queries may have an exponential number of solutions w.r.t. the input size, so that even state-of-the-art solvers are not helpful. We believe that structural decomposition methods, besides their theoretical interest, may find useful applications in solving real-world instances of these problems.

The thesis also presents a further line of research where CSP techniques are employed to model and solve Process Mining problems. Such problems arise in the study of workflow management systems that aim at supporting the execution of business processes in various contexts. In particular, to support the (re)design and the optimization of existing processes, it is desirable to identify the best strategies to schedule the execution of any given set of activities, by looking at the choices made in the past and registered in execution logs.

In summary, the thesis exploits the general CSP framework as a weapon to analyze and to attack problems arising in different areas of research. The main results, of both theoretical and practical interest, are listed in the next section.

1.1 Contributions

- **Database theory.** In this area of research, we use CSP structural techniques for the problem of counting solutions of SQL queries involving "COUNT" aggregates, which occur very often in practice. First, we propose a novel structural decomposition notion that extends the framework of greedy tree projections by adding weights. Then we design, analyze, and give a polynomial time implementation of an algorithm for computing a restricted variant of such decompositions, called monotone greedy tree projections. Then, we identify a restricted class of weighting functions for which the computation of an optimal weighted

decomposition is tractable. Furthermore, we propose optimization algorithms, which exploit the novel structural framework, that can be used to boost the performances of relational DBMS.

- **Game theory.** In this field we focused on coalitional games, which model situations where players can obtain higher worths by collaborating with each other, rather than by acting in isolation. We consider resource allocation problems and in particular the notion of Shapley value and Banzhaf value, which are two well-known solution concepts to provide fair worths distribution. For this class of games, is known that computing either solution concept is an intractable problem, formally #P-hard. Motivated by these bad news, we identified large islands of tractability for both solution concepts, by exploring the structure of the interaction graphs among the players. We showed that tractability holds in scenarios where each good is owned by at most two agents (independently of their interactions). Moreover, we study allocation games where interaction graphs have bounded treewidth: Our main result is that both the Shapley value and the Banzhaf value can be computed in polynomial time for this class of games. A key ingredient for the polynomial-time tractability of the problem is a CSP encoding allowing us to exploit the CSP machinery for counting solutions. Note that our restrictions capture scenarios of practical interest, such as the allocation problem arising in the Italian Research Assessment program promoted by ANVUR.

- **Process mining.** The third direction of research is devoted to the study of workflow management systems that aim at supporting the execution of business processes in various contexts. We design and implement a "hybrid" approach to process discovery. A mining method is conceived which can take into account

a wide variety of constraints over the causal dependencies that are available to the analyst. This is particularly useful for circumventing problems emerging when log completeness does not hold. We formulate algorithms in terms of reasoning problems over precedence constraints, and we analyze the computational complexity of the proposed setting, by taking into account various qualitative properties regarding the kinds of constraints being allowed, and by tracing the tractability frontier w.r.t. them. In particular, we show that for the classes of constraints that are not covered by our algorithms, an efficient solution algorithm is unlikely to exist at all, because process discovery turns out to be NP-hard over them.

1.2 Outline of the dissertation

The rest of the dissertation is organized as follows. In Chapter 2, we discuss the setting of constraint satisfaction problems, including structural decomposition methods, counting problems and preliminaries on computational complexity. In Chapter 3, we briefly review some basic concepts from database theory. In Chapter 4, we present the novel notion of weighted monotone greedy tree projection. We analyze the complexity of computing greedy tree projections in different settings, and we give an algorithm that can be exploited for all problems that can be solved efficiently on acyclic and quasi-acyclic instances. In Chapter 5 we take a first step towards designing an Hybrid optimizer for counting problems. In Chapter 6 we review basic concepts from game theory, and we discuss a real-world application modeled as a coalitional game. In Chapter 7 we identify structural restrictions for the class of allocation games that guarantee the tractability of computing the Shapley value. We also strengthen some hardness result about this problem. In Chapter 8 we intro-

duce the framework of process mining and we discuss process-discovery methods. In Chapter 9, we analyze the complexity of computing business process models under precedence constraints imposed over the topology of the mined model. We chart the frontier of tractability for this problem. We present a CSP encoding for this problem and heuristic algorithms that however solve exactly the problem over some restricted classes. In Chapter 10 we describe a real-world scenario that can be modeled as a mining problem, and hence can benefit of our approaches. Finally, in Chapter 11 we conclude by discussing further research directions.

CHAPTER 2

Setup and Preliminaries

In this chapter we review the framework of Constraint satisfaction problems (CSPs) and we explore the structure of these problems from a complexity-theoretic perspective exploiting the concept of decomposition. In particular, we will first present two relevant decomposition methods for (hyper)graph based structures: the treewidth, which is the most powerful decomposition method on graphs, and the hypertree width, which is its natural counter-part over hypergraphs. Then we will show that both methods are specializations of a more general decomposition scheme called tree projection, which we will describe in Section 2.4.3. Finally we will introduce counting problems in Section 2.5. Meanwhile, we need to define the basic concepts that we will use throughout the thesis.

2.1 Constraint satisfaction problems

A fundamental problem in the field of Artificial Intelligence and related disciplines, in particular Database theory, is the *constraint satisfaction problem (or CSP)* which provides a unifying framework to express a wide spectrum of computational problems, such as graph colorability, model checking, planning, satisfiability, scheduling, theorem proving and database queries. Crudely, the goal of a CSP is to assign one value to each variable so that the constraints are all satisfied. While this problem

appears very general and indeed suitable to describe many problems, it is not surprising that the price to be paid for this generality is its high computational complexity. Formally, a CSP instance is a triple $\mathcal{I} = \langle Var, U, \mathbf{C} \rangle$, where Var is a finite set of variables, U is a finite domain of values, and $\mathbf{C} = \{C_1, C_2, \ldots, C_q\}$ is a finite set of constraints. Each constraint C_v, for $1 \leq v \leq q$, is a pair (S_v, r_v), where $S_v \subseteq Var$ is a set of variables called the *constraint scope*, and r_v is a *constraint relation*, i.e., a set of substitutions $\theta : S_v \to U$ indicating the allowed combinations of simultaneous values for the variables in S_v. For each variable $X \in Var$, we denote by $\texttt{dom}(X)$ its *domain*, i.e., the set of all elements $u \in U$ for which some constraint relation contains a substitution θ with $\theta(X) = u$. In the following, a substitution from a set of variables $V \subseteq Var$ to U is transparently viewed as the set of pairs of the form X/u, where $\theta(X) = u$ is the value to which $X \in V$ is mapped. A substitution θ *satisfies* a constraint C_v if its restriction to S_v, i.e., the set of all pairs $X/u \in \theta$ such that $X \in S_v$, occurs in r_v. A *solution* to $\mathcal{I}$ is a substitution $\theta : Var \mapsto U$ that satisfies all constraints. The set of all solutions is denoted by $\Theta(\mathcal{I})$. If $\mathcal{W}$ is a set of variables, then $\Theta(\mathcal{I}, \mathcal{W})$ denotes the set of all solutions in $\Theta(\mathcal{I})$ restricted to the variables in $\mathcal{W}$. Variables outside $\mathcal{W}$ can be viewed as *auxiliary* ones. That is, they are used for internal encoding activities, and they are not required in the output. The problem is either to find all solutions, one solution, or to verify that a certain substitution θ is a solution [Dechter and Pearl, 1987]. While the last problem is easy, the first two are NP-hard and in particular the first can be much harder than the second, as we will discuss later in this chapter.

Example 2.1. As an example, consider the map coloring problem (i.e., graph colorability) reported in Figure 2.1. The goal is to assign k-colors to the provinces of Calabria such that no two adjacent provinces have the same color. We can formulate

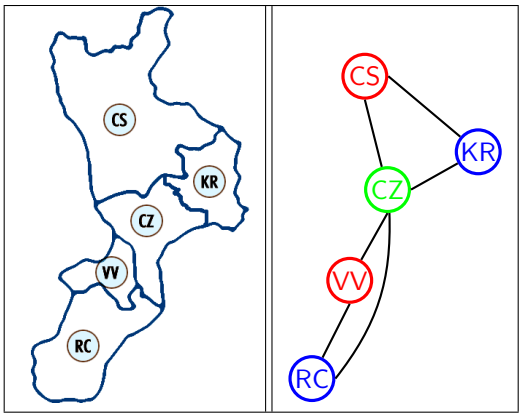


Table 2.1: The provinces of Calabria and its colored graph

this problem as CSP where we define the variables to be the provinces: $\{CS,\ CZ,\ KR,\ VV\ and\ RC\}$, the domain of each variable is the set $\{red,\ green,\ blue\}$ and the constraints impose that two adjacent provinces must have different colors. A solution to this problem is a combination of simultaneous values for the variables that satisfies all constraints. For the example above, there are many possible solutions, such as $\{CS = red,\ CZ = green,\ KR = blue,\ VV = red,\ RC = blue\}$. $\triangleleft$

Observe that, the problem described above is a binary constraint satisfaction problem over a finite domain, since all constraints are imposed between pairs of variables (i.e., the arity of these constraints is just 2). In addition, it is worth noting that any n-ary CSP can be converted to a binary CSP by means of a suitable transformation. Also note that we described the problem in terms of graph and this is not by chance. In fact, with each CSP instance $\mathcal{I}$, it is naturally to associate a constraint (hyper)graph $G(\mathcal{I})$ whose nodes are the variables and where there is a (hyper)edge between any pair of variables appearing within the same constraint scope. Indeed, the constraint graph describes the structure induced by the constraints on the variables and it plays a crucial role in our understanding of the complexity of CSPs. We will come back to this in Section 2.3.

For the sake of completeness we show also an example of n-ary CSP.

Example 2.2. Consider the Boolean Satisfiability Problem (SAT, for short), that is, given a Boolean formula ϕ in conjunctive normal form (CNF), is there an assignment of 0s and 1s to the variables in input such that the formula returns the value 1. SAT is a constraint satisfaction problem where the CNF formula ϕ corresponds to a CSP instance $\mathcal{I}$ whose variables are the same variables of ϕ, whose domain is $U = \{0, 1\}$ and where each constraint $C_i \in \mathbf{C}$ corresponds to a clause $c_i \in \phi$. For example, the clause $c_{xyz} = (x \vee \neg y \vee \neg z)$ corresponds to a constraint (S_{xyz}, r_{xyz}) where r_{xyz} is the ternary relation $\{(0,0,0), (0,0,1), (0,1,0), (1,0,0), (1,0,1), (1,1,0), (1,1,1)\}$ on $\{0, 1\}$.

We conclude this section by observing that CSPs can be expressed via *Constraint programming*, a declarative programming paradigm where users are just in charge of specifying the problem at hand in terms of a constraint satisfaction problem, instead of formalizing the steps needed for its solution. Constraint programming systems exploit, indeed, general search mechanisms (such as backtracking) enriched with powerful speed-up methods (such as constraint propagation techniques) in order to find a solution such that meets declarative specification. We will detail how such mechanisms work in the next section. Finally, it is worth noting that a constraint language is tractable if we can solve the associated CSP in polynomial time.

2.2 Solving CSPs

Now that we introduced the basic terminology, we go back to the main issue of CSP and we point out some interesting properties of this problem. When dealing with general CSPs, the major drawback is represented by their high complexity that often calls for heuristic methods to achieve a solution in reasonable time. Therefore,

much effort has been spent by the scientific community in finding an efficient solution for the constraint satisfaction problem. Perhaps, the most important direction is the identification of *tractable classes* of CSPs and taking advantage of such classes. In fact, we can use the knowledge about solving tractable CSP classes to help the solution of harder problems. The approaches to identify tractable CSP classes can be divided into two main categories [Gottlob *et al.*, 2000]:

- Tractable structural restrictions. This includes all tractable classes of CSPs that are identified only by analyzing the structure of the constraint scopes.
- Tractability due to restricted constraint relations. This includes all classes that are tractable as a consequence of particular properties of the constraint relations.

In this dissertation, we will focus on tractable structural restrictions. But before we start describing the tools that help us address this issue, let us discuss what happens in the general case. It turns out that the most commonly used algorithm for solving CSPs is the backtracking search.

2.2.1 Backtracking search

Backtracking search is as simple as systematic. Basically it consists of the following steps. Given a CSP instance $\mathcal{I} = \langle Var, U, \mathbf{C} \rangle$, it starts in a state where no assignment has been done for the variables in Var. Then it traverses the variables in a predetermined order, it picks a variable, it assigns a value and if the assigment is consistent (i.e., it satisfies all the constraints), it repeats. Eventually, if no consistent assignment can be found for the next variable then the algoritm backtracks the search tree until it finds a variable that can have its value set consistently to something else. Backing all the way up the tree to the root, and finding no more values means that there is no solution. A backtracking algorithm that computes one

solution is given below.

Algorithm 1 backtrack_search

Input: $\mathcal{I} = \langle Var, U, \mathbf{C} \rangle$ a CSP instance
Output: $TRUE$ if there exists a solution $FALSE$ otherwise.

```
1:  (F, A) := (∅, ∅);
2:  while there are unassigned var do
3:      X_i := pick(Var);
4:      for all v ∈ U(X_i) do
5:          A := A ∪ {X_i = v};
6:          if {X_i = v} ∈ Forbid then
7:              continue;
8:          end if
9:          F := F ∪ {X_i = v};
10:         A := A ∪ {X_i = v};
11:         if ∃ c ∈ C s.t. c is not satisfied then
12:             A := A \ {X_i = v};
13:         else
14:             goto 2;
15:         end if
16:     end for
17:     unroll previous choice;                          ▷ also clear F
18:     if A = ∅ then
19:         return FALSE
20:     end if
21: end while
22: return TRUE
```

When we use this algorithm for finding one solution, it is natural to ask the following questions.

- How do we chose the next variable and its value?
- Can we prune the search space, and thereby search less?
- Can we learn from our mistakes, i.e., bad variable selections?

For instance, the problem of selecting a variable and assigning a value plays a significant role in the execution time. In fact, the easy approach that consists in picking the next variable and next value from a static list is not always the most efficient approach. Indeed is not easy to make the right choice and thus several heuristic proposal have been offered by the AI community. For example we can choose the variable with the smallest number of remaining values in its domain.

Another approach could be the one that chooses the variable that is part of the most remaining unsatisfied constraints. Moreover, once we choose a variable, we select from its domain the value that rules out the fewest choices for neighboring variables. It turns out that Algorithm 1 is not very efficient or intelligent since it does not use all the information available from the constraint set and the assigned variables. In fact, we can use the value assigned to available and the contraints in order to probe the search space, that is we restrict the available future values for the other variables. If we empty the domain of another variable, that is we are unable to assign a consistent value to another variable, then we know that we can backtrack without exploring the remaining subtree below the current variable.

We conclude this section, by pointing out again that there are many situations (such as database query answering), in which we are interested not only in finding one solution but we require all solutions. In this setting, it easy to see that things become computationally much harder since even the simplest constraint satisfaction problem instance may have an exponential number of satisfying solutions. Therefore, it is sensible to propose algorithms that generate solutions with polynomial delay. For the remainder of this thesis, we will consider both these settings.

2.3 Structural properties of CSPs

In this section, we change our perspective by focusing on the structure of CSPs with the objective to identify structural properties that lead to *tractable classes* of CSPs. Indeed, taking advantage of the understading of such classes often increases our abilities in solving harder problems. As we already noted, many problems arising in different areas of computer science can be described via CSPs. Often, such problems are intractable in general, but whenever the input has a desired structure

(for example, the constraint graph is a tree), we can compute solution efficiently via dynamic programming. In graph-based problems, the intractability is frequently due to cyclicity. A natural question to ask is therefore whether the degree of cyclicity affects the complexity of the problem. The answer to this question is "yes": the more the input is *tree-like* the easier it becomes. In particular, it was recognized in both AI and database theory community that the most important property in the context of CSPs (and conjunctive database queries) is *acyclicity* [Gottlob *et al.*, 2000].Therefore, multiple effort has been made in finding a way to measure the degree of cyclicity of graphs (and hypergraphs) with the objective to identify nearly-acyclic instances resulting in the definition of a broad set of *structural decomposition methods*. Decomposition techniques work as follows: First we associate to each instance problem $\mathcal{I}$ the graph G (or a hypergraph $\mathcal{H}$). Then, clusters of vertices are arranged in some tree-shape in order to obtain an acyclic data structure. Finally we can formalize the concept of degree of cyclicity using the *width* of the decomposition (of the instance problem $\mathcal{I}$) that corresponds to the size of the largest cluster of vertices occurring in the decomposition. The *treewidth*, which we will review more technically in the next section, is currently the best way to measure the degree of cyclicity of a graph since it subsumes all the other measures.

2.4 Structural decomposition methods

2.4.1 Tree decompositions

A *tree decomposition* [Robertson and Seymour, 1984] of a graph G is a pair $\langle T, \chi \rangle$, where $T = (N, E)$ is a tree, and χ is a labeling function assigning to each vertex $v \in N$ a set of vertices $\chi(v) \subseteq \mathit{nodes}(G)$, such that the following conditions are satisfied:

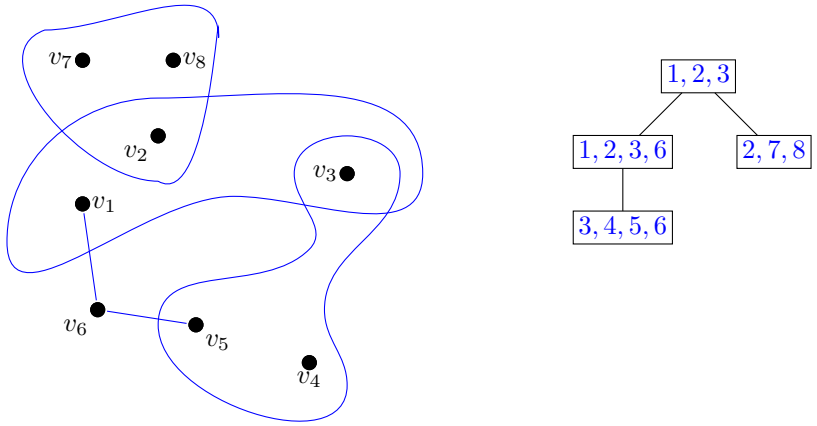


Figure 2.1: A hypergraph $\mathcal{H}$ and a tree decomposition of $\mathcal{H}$ of width 3

1. for each node $Y \in \mathit{nodes}(G)$, there exists $p \in N$ such that $Y \in \chi(p)$;
2. for each edge $\{X, Y\} \in \mathit{edges}(G)$, there exists $p \in N$ such that $\{X, Y\} \subseteq \chi(p)$;
3. for each node $Y \in \mathit{nodes}(G)$, the set $\{p \in N \mid Y \in \chi(p)\}$ induces a (connected) subtree of T.

The *width* of $\langle T, \chi \rangle$ is the number $\max_{p \in N}(|\chi(p)| - 1)$.

In particular, the *treewidth* of $\mathcal{G}$ is the minimum width over all the tree decompositions that can be constructed. Deciding whether a given graph has treewidth bounded by a fixed natural number k is known to be feasible in linear time [Bodlaender and Fomin, 1996]. Moreover, the treewidth of a hypergraph $\mathcal{H}$ is 1 if $\mathcal{H}$ is acyclic, and it is equal to the treewidth of its primal graph otherwise. As an illustrative example, consider the Figure 2.1

2.4.2 (Generalized) Hypertree decompositions

Unfortunately, there exist a number of real-world problems where very complex relationships between objects may arise, leading to loss of information whenever these relationships are represented as graphs. In particular, while binary relationships can naturally be illustrated with simple graphs, general relationships are best described

by hypergraphs. Since hypergraphs generalize graphs, we can define the degree of cyclicity of a hypergraph and we can carry out investigations on its properties.

A *hypergraph* $\mathcal{H}$ is a pair (V, H), where V is a finite set of nodes and H is a set of hyperedges such that, for each $h \in H$, $h \subseteq V$. In the following, we denote V and H by $nodes(\mathcal{H})$ and $edges(\mathcal{H})$, respectively.

A hypergraph $\mathcal{H}$ is *acyclic* (more precisely, α-acyclic [Fagin, 1983]) if, and only if, it has a join tree [Bernstein and Goodman, 1981], i.e., a tree whose vertices are the hyperedges of $\mathcal{H}$ such that, whenever a node $X \in V$ occurs in two hyperedges h_1 and h_2 of $\mathcal{H}$, then h_1 and h_2 are connected in JT, and X occurs in each vertex on the unique path linking h_1 and h_2. In words, the set of vertices in which X occurs induces a connected subtree of JT. Similar to the notion of tree decomposition, there exists the (generalized) hypertree decomposition over hypergraphs which, in turn, leads to the associated notion of hypertree-width.

A hypertree for a hypergraph $\mathcal{H}$ is a triple $\langle T, \chi, \lambda \rangle$, where $T = (N, E)$ is a rooted tree, and χ and λ are labeling functions that associate each vertex $v \in N$ with two sets $\chi(v) \subseteq nodes(\mathcal{H})$ and $\lambda(v) \subseteq edges(\mathcal{H})$.

The width of a hypertree is the size of its largest λ, i.e., $\max_{p \in T}(|\lambda(p)|)$. We denote the set of vertices N of T by $vertices(T)$, and the root of T by $root(T)$. Moreover, for any $p \in N$, T_p denotes the subtree of T rooted at p. If T' is a subtree of T, we define $\chi(T') = \bigcup_{v \in vertices(T')} \chi(v)$.

A generalized hyertree decomposition of a hypergraph $\mathcal{H}$ is a hypertree $HD = \langle T, \chi, \lambda \rangle$ such that: (1) for each hyperedge $h \in edges(\mathcal{H})$, there exists $p \in vertices(T)$ such that $vars(h) \in \chi(p)$; (2) for each node $Y \in nodes(\mathcal{H})$, the set $\{p \in vertices(T) \mid Y \in \chi(p)\}$ induces a (connected) subtree of T; (3) and finally, for each node $p \in vertices(T), \chi(p) \subseteq nodes(\lambda(p))$

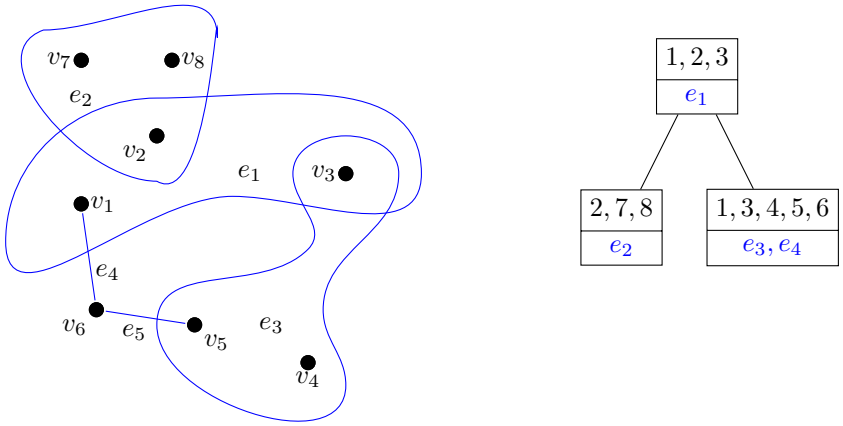


Figure 2.2: A hypergraph $\mathcal{H}$ and a hypertree decomposition of $\mathcal{H}$ of width 2

In particular, the *hypertree width* of $\mathcal{H}$ is the minimum width over all the hypertree decompositions that can be constructed.

The hypertree decomposition HD is a *complete* decomposition if for each hyperedge $h \in edges(\mathcal{H})$, there exists $p \in vertices(T)$ such that $vars(h) \in \chi(p)$ and $h \in \lambda(p)$. Moreover, a generalized hypertree decomposition is a hypertree decomposition if it satisfies the following special condition: for each node $p \in vertices(T), nodes(\lambda(p)) \cap \chi(T_p) \subseteq \chi(p)$.

Observe that, deciding whether a given hypergraph has hypertree width bounded by a fixed natural number k is feasible in polynomial time (and, actually it is highly-parallelizable) [Gottlob *et al.*, 2002]. Moreover, it is known that for any hypergraph $\mathcal{H}$ the generalized hypertree width $ghw(\mathcal{H})$ is always smaller than or equal to the hypertree width $hw(\mathcal{H})$, and more precisely, $ghw(\mathcal{H}) \leq hw(\mathcal{H}) \leq ghw(\mathcal{H}) + 1$ [Adler *et al.*, 2007].

Deciding whether a given hypergraph has generalized hypertree width bounded by a fixed natural number k is NP-complete [Gottlob *et al.*, 2009].

Figure 5.1 shows an example of hypertree decomposition.

2.4.3 Tree projections

As noted before, these methods have in common the idea of transforming a given cyclic hypergraph (graph) into an acyclic one, by organizing its vertices (or its edges) into a polynomial number of clusters and by arranging these clusters into a tree-shape structure – a so-called *decomposition tree*. Given the decomposition tree, we can then evaluate the original problem with a cost that is exponential in the width of the decomposition (i.e., the cardinality of the largest cluster) but polynomial in the combined size of the input and the output. Consequently, great effort has been spent by the scientific community in finding the best method to identify nearly acyclic graphs (hypergraphs) resulting in the definition of various methods that regardless of the technical differences, are just a special case of a general framework called *tree projections*. The result is the following formal definition for this general framework. For a given pair of hypergraphs $(\mathcal{H}_1, \mathcal{H}_2)$, a tree projection of $\mathcal{H}_1$ w.r.t $\mathcal{H}_2$ is an acyclic hypergraph $\mathcal{H}_a$ such that each hyperedge of $\mathcal{H}_1$ is contained in some hyperedge of $\mathcal{H}_a$, that is in turn contained in some hyperedge of $\mathcal{H}_2$. $\mathcal{H}_2$ is called the *resource hypergraph* and it is arbitrary. Hence, whenever we compute $\mathcal{H}_2$ with some specific method from the hypergraph $\mathcal{H}_1$, we obtain, as special cases, some known *purely* structural decomposition methods. In fact, observe that the decomposition method based on the submodular width [Dániel Marx, 2010], which is not purely structural, is the only one among all the structural decomposition methods that does not fit this framework.

Let $\mathcal{H}_1, \mathcal{H}_2$ be two hypergraphs. We assume w.l.o.g. that $nodes(\mathcal{H}_1) = nodes(\mathcal{H}_2)$. Given $(\mathcal{H}_1, \mathcal{H}_2)$, we say that $\mathcal{H}_1 \leq \mathcal{H}_2$, iff each hyperedge of $\mathcal{H}_1$ is contained in at least one hyperedge of $\mathcal{H}_2$. Let $\mathcal{H}_1 \leq \mathcal{H}_2$; then, a *tree projection* of $\mathcal{H}_1$ w.r.t $\mathcal{H}_2$ is an acyclic hypergraph $\mathcal{H}_a$ such that $\mathcal{H}_1 \leq \mathcal{H}_a \leq \mathcal{H}_2$. We say that $\mathcal{H}_a$ is a tree projection

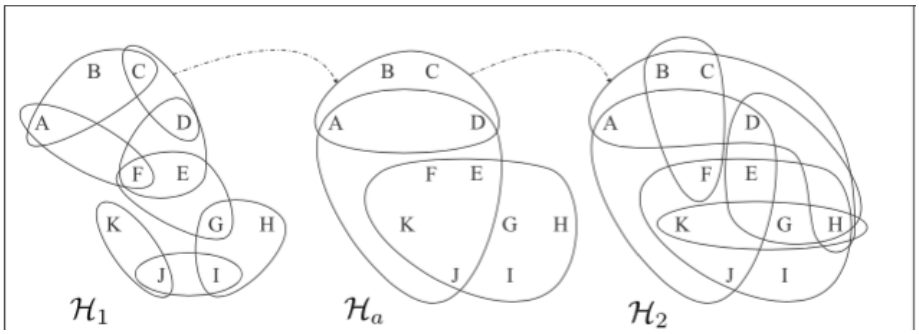


Figure 2.3: A tree projection $\mathcal{H}_a$ for the pair of hypergraphs $(\mathcal{H}_1, \mathcal{H}_2)$.

for the pair $(\mathcal{H}_1, \mathcal{H}_2)$ if it exists.

In fact, by slightly abusing notation, we use the term tree projection to refer to a join tree of such a sandwich acyclic hypergraph or, equivalently, a width-1 hypertree decomposition of $\mathcal{H}_a$. More precisely, we say that a tree projection of $\mathcal{H}_1$ w.r.t $\mathcal{H}_2$ is a triple $HD = \langle T, \chi, \lambda \rangle$ where $T = (N, E)$ is a rooted tree; χ and λ are labeling functions that associate each vertex $v \in N$ with two sets of vertices $\chi(v) \subseteq \mathit{nodes}(\mathcal{H}_1)$ and $\lambda(v) \in \mathit{edges}(\mathcal{H}_2)$; and such that the following conditions hold: (1) for each hyperedge $h \in \mathit{edges}(\mathcal{H}_1)$, there exists $v_h \in N$ with $h \subseteq \chi(v_h)$; (2) for each node $v \in \mathit{nodes}(\mathcal{H}_1)$, the set $\{v_h \in N \mid v \in \chi(v_h)\}$ induces a (connected) subtree of T;

Throughout the dissertation we assume that $\mathcal{I}$ is a given CSP instance, and we shall seek to compute its solutions (possibly restricted to a desired set of output variables) by combining the solutions of suitable sets of subproblems, available as additional distinguished constraints called resource views. Intuitively, each original constraint of $\mathcal{I}$ is associated with a new distinguished hyperedge in the resource hypergraph. Importantly, note that a resource view in order to be a legal subproblem must be not more restrictive than the full problem. Moreover, any resource view has to be at least as restrictive as the original constraint associated with it.

Example 2.3. Consider the hypergraph $\mathcal{H}_1$ on the left of Figure 2.3 together with other two hypergraphs that are discussed next. $\mathcal{H}_1$ represents the CSP instance $\mathcal{I}$ whose variables are given by $nodes(\mathcal{H})$ and where each constraint $C_i \in \mathbf{C}$ corresponds to a hyperedge appearing in $edges(\mathcal{H})$. For example, the constraint set $\mathbf{C}$ is given by

$$\mathbf{C} := \{\langle A, B, C\rangle, \langle A, F\rangle, \langle C, D\rangle, \langle D, E, F\rangle,$$
$$\langle E, F, G\rangle, \langle G, H, I\rangle, \langle I, J\rangle, \langle J, K\rangle\}.$$

To solve $\mathcal{I}$, assume that a set of resource views is available, playing the role of additional distinguished constraints. The set of variables of each resource view is a hyperedge in the hypergraph $\mathcal{H}_2$. In the middle between $\mathcal{H}_1$ and $\mathcal{H}_2$, Figure 2.3 it is reported the hypergraph $\mathcal{H}_a$ which covers $\mathcal{H}_1$, and which is in its turn covered by $\mathcal{H}_2$ e.g., $\{C, D\} \subseteq \{A, B, C, D\} \subseteq \{A, B, C, D, H\}$. Since $\mathcal{H}_a$ is in addition acyclic, $\mathcal{H}_a$ is a tree projection of $\mathcal{H}_1$ w.r.t. $\mathcal{H}_2$.

Note that all the (known) structural decomposition methods can be recast as special cases of tree projections, since they just differ in how they define the set of views to be built for evaluating the CSP instance. For example, consider the generalized hypertree decomposition approach. Given a hypergraph $\mathcal{H}$ and a natural number $k > 0$, let $\mathcal{H}^k$ denote the hypergraph over the same set of nodes as $\mathcal{H}$, and whose set of hyperedges is given by all possible unions of k edges in $\mathcal{H}$, i.e., $edges(\mathcal{H}^k) = \{h_1 \cup h_2 \cup \cdots \cup h_k \mid \{h_1, h_2, \ldots, h_k\} \subseteq edges(\mathcal{H})\}$. Then, it is well known and easy to see that $\mathcal{H}$ has generalized hypertree width at most k if, and only if, there is a tree projection for $(\mathcal{H}, \mathcal{H}^k)$.

Similarly, for tree decompositions, let $\mathcal{H}^{tk}$ be the hypergraph over the same set of nodes as $\mathcal{H}$, and whose set of hyperedges is given by all possible clusters $B \subseteq nodes(\mathcal{H})$ of nodes such that $|B| \leq k + 1$. Then, $\mathcal{H}$ has treewidth at most k if, and

only if, there is a tree projection for $(\mathcal{H}, \mathcal{H}^{tk})$.

2.5 Counting problems

So far, we discussed how to solve CSPs exploiting its structural properties. However, in real-world applications like database query answering, often we do not concentrate on finding one solution or just counting the number of solutions, possibly focusing on a restricted set of output variables (as we will discuss in Chapter 5). For example, the "select" part of an SQL query allows users to specify a set of output variables, so that query answering amounts at *enumerating* all solutions projected over those variables, rather than just deciding whether there is any. In [Atserias *et al.*, 2013], it has been shown that the enumeration problem where all variables are output variables (i.e., "SELECT *" queries where no variable is projected out) can be solved in *polynomial time* on a class **C** of queries if, and only if, the number of solutions is always polynomial, and that this is the case, if and only if, the queries in **C** have bounded *fractional edge cover number*. Similar tight worst-case bounds for conjunctive queries with arbitrary sets of output variables have been derived in [Gottlob *et al.*, 2012]. Note however that even easy instances may have an exponential number of solutions. Therefore, for enumeration problems it is sensible to propose algorithms computing solutions with *polynomial delay*, or even with *linear delay* [Bagan *et al.*, 2007]. Again, acyclicity has been shown to be a key for tractability in this setting [Yannakakis, 1981; Koch, 2006; Kimelfeld and Sagiv, 2006; Bagan *et al.*, 2007], and generalizing these results to decomposition methods is an active area of research (see, e.g., [Bulatov *et al.*, 2012; Greco and Scarcello, 2013a]). Closely related to the enumeration problem is the problem of *counting* the solutions of constraint satisfaction problems (see, e.g, [Bulatov *et al.*, 2013; Bulatov, 2013; Gomes

et al., 2007; Pesant, 2005]). This problem is equivalent to counting the number of query answers in the related field of database theory, which occurs in the presence of SQL queries specifying "COUNT" aggregates. Note that these queries are very often at the basis of decision support systems examining large volumes of data in order to get insights on critical business questions. In this thesis we analyze the well known TPC-DS, which is a decision support benchmark,[1] consisting of business-oriented complex queries (many of them involve counting) over which structural decomposition methods can exhibit their effectiveness. Indeed, recently *structural tractability results* have been generalized to counting problems, as summarized below.

Theorem 2.4 (cf. [Pichler and Skritek, 2013; Greco and Scarcello, 2014b])**.** *Counting the number of substitutions in $\Theta(\mathcal{I}, \mathcal{W})$ is feasible in polynomial time, on classes of CSP instances $\mathcal{I}$ such that the treewidth of $G(\mathcal{I})$ is bounded by a constant, and the size of the domain of each variable not in $\mathcal{W}$ is bounded by some constant, too.*

Note that, differently from the case of the standard decision and computation problems, the result is established under the additional condition that auxiliary variables have a bounded domain. If the condition is not met, then #P-complete instances can be exhibited Pichler and Skritek [2013].

Example 2.5. As a tiny example, consider the following counting problem. We want to count the number of PhD students that attended some Summer School about Constraint Satisfaction. We can express this problem as a CSP: Consider the CSP instance $\mathcal{I}$ whose variables are the set $\{Name, Affiliation, School, Year, Subject, Location\}$, whose domain is $U = \{...\}$ and where the constraint set $\mathbf{C}$ is as follows.

$$\{student\langle Name, Affiliation\rangle, attendend\langle Name, School\rangle,$$
$$school\langle School, Year, Subject, Location\rangle, equal\langle Subject, "CS"\rangle\}.$$

[1] http://www.tpc.org/tpcds/default.asp

The intended meaning is that we want to count over all solutions of this CSP the number of *distinct* occurrences of any student for which there exists some school in some year that she attended and that was about Constraint Satisfaction. Note that, we have just one output variable that is the student's name. All other variables are just auxiliary (i.e, existential variables in the logic-based syntax). The objective is to compute just the number of (desired) solutions. In fact, in counting problems we may get wrong numbers if such variables are not properly considered. In this example, we could get as a result the number 8 that may come from the one student attending 3 summer school on constraint satisfaction, 3 on Game Theory and 2 on Mining, or by 8 distinct students on constraint satisfaction. The counting problem is quite different from the enumeration problem where even if the complexity may change, we still have all the solutions and thus we can get the right number by just projecting out the useless variables.

2.5.1 The Langford problem

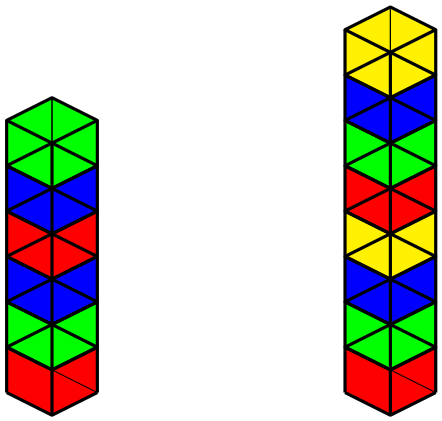


Figure 2.4: Langford problem

Langford's problem goes back to the Scottish mathematician *C. Dudley Langford* who once observed his young boy stacking colored blocks. While he was watching, he observed that his son piled six colored blocks in a such a way that there was one block

between the blue pair, two blocks between the red pair and three blocks between the green pair. This is exemplified on the left of Figure 2.5.1. Then he added a pair of yellow blocks and he came up with a solution like the one depicted in the right of Figure 2.5.1. It turns out that, these solutions are unique for 3 and 4 pairs of colored blocks. One could always reverse the order and cycling colors but this is irrelevant. Indeed, the number of distinct solutions is just one. Then, Langford tried to find a solution for 5 and 6 pairs of blocks but he did not succeed. Later he was able to find a solution for 7 and 8 pairs but again he was unable to find a solution for 9 and 10 pairs. Therefore, it was natural for him to investigate more and he came up with the following question. What values of n admit a solution? [Davies, 1959] gave an answer to that question proving that for $5, 6, 9$ and 10 there not exists a solution. In particular he proved that it is possible to find a solution only if n is a multiple of four or one less than a multiple of four. The study of this problem is fascinating in that for small numbers of blocks, we are able to find a solution even without using a computer. Indeed, we have done that for $n = 3$ and $n = 4$ in Figure 2.5.1. Therefore, we wonder how many solutions exist for higher values of n. While finding one solution for any number of blocks is quite easy, counting all solutions becomes pretty hard. Just to give an idea, the exact number of distinct solutions for $n \geq 28$ is not known as of today. Moreover, $n = 27$ was determined only in 2015. Perhaps, we want to look at the numbers in Table 2.5.1 in order to realize how difficult are these kinds of problems.

n	# of solutions
1	-
2	-
3	1
4	1
5	-
6	-
7	26
8	150
9	-
10	-
11	17,792
12	108,144
13	-
14	-
15	39,809,640
16	326,721,800
17	-
18	-
19	256,814,891,280
20	2,636,337,861,200
21	-
21	-
23	3,799,455,942,515,488
24	46,845,158,056,515,936
25	-
26	-
27	111,683,606,778,027,803,456
28	?
29	-

Table 2.2: Known solutions for the Langford problem. A dash (-) means that there is no solution for the given value of n while (?) stands for unknown.

2.5.2 Configuration problems: Renault example

As an interesting real-world example of counting problem, we describe the Renault Megane configuration problem [Amilhastre et al., 2002] used in CSP competitions as a benchmark for CSP solvers. The problem consists of 108 variables that encodes informations about the Renault megane such as the type of engine, country, options like air cooling, etc. with domains ranging from 2 to 43 and with a total of 858 constraints which can be compressed to 149 constraints. Furthermore, many of them are non binary indeed arity of constraint scopes is up to 10. Finally, the constraint

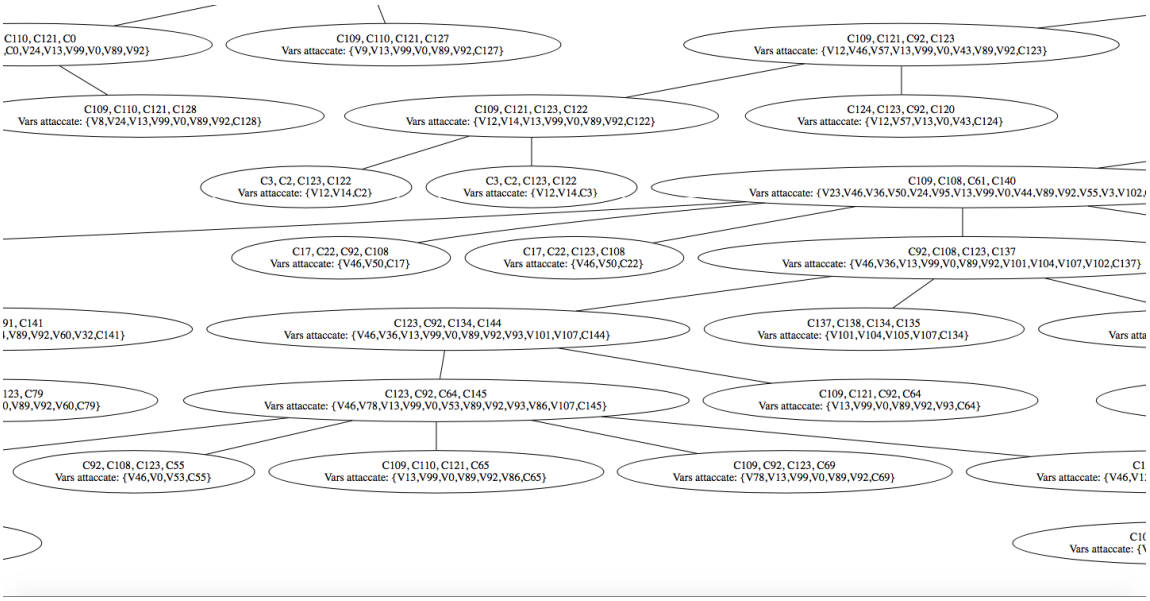


Figure 2.5: A portion of hypertree decomposition of width 4 for the Renault megane configuration problem

relations comprise about 200000 tuples. We analyzed this problem by applying our decomposition tools and despite the big numbers in play, we discovered that the *hypertree-width* associated with it is just 4 as we shall see in more detail in Chapter 4. Moreover, we observed that it is possible to find one solution for this problem very quickly. However, if our objective is to find all the solutions, the problem becomes much harder to solve. In Figure 2.5 we show a portion of the decomposition tree for this problem. The total number of solutions known for this problem is $2.84 * 10^{12}$.

We conclude this chapter by giving some preliminaries on complexity theory.

2.6 Preliminaries on Complexity Theory

We recall some notions of complexity theory which we will use later in this thesis, by referring the reader to the book by [Garey and Johnson, 1979] for more on this subject.

Decision problems are maps from strings (encoding the input instance over a fixed alphabet, e.g., the binary alphabet $\{0,1\}$) to the set $\{"yes","no"\}$. We are often interested in computations carried out by *non-deterministic* Turing machines. We recall that these are Turing machines that, at some points of the computation, may not have one single next action to perform, but a *choice* between several possible next actions. A non-deterministic Turing machine answers a decision problem if, on any input x, (i) there is at least one sequence of choices leading to halt in an accepting state if x is a "yes" instance (such a sequence is called accepting computation path); and (ii) all possible sequences of choices lead to a rejecting state if x is a "no" instance. The class of decision problems that can be solved by non-deterministic Turing machines in polynomial time is denoted by NP.

A decision problem A_1 is *polynomially reducible* to a decision problem A_2, if there is a polynomial-time computable function h (called reduction) such that, for every x, $h(x)$ is defined and x is a "yes" instance of A_1 if and only if $h(x)$ is a "yes" instance of A_2. A decision problem A is NP-*hard* if every problem in NP is polynomially reducible to A; if A is NP-hard and belongs to NP, then A is said to be NP-*complete*. Thus, problems that are complete for NP are the most difficult problems in NP. In particular, it is unlikely that an NP-complete problem can be solved in polynomial time because it would entail P=NP.

Part I

CSP and Databases

CHAPTER 3

Basic Definitions from Database Theory

In this chapter we will discuss the connection between constraint satisfaction problems and database theory. Various important problems in the field of database theory, such as the problem of evaluating *Boolean Conjunctive Queries* over a relational database, or the problem of conjunctive query containment are equivalent to solving constraint satisfaction problems as pointed out by several authors, e.g., [Gyssens *et al.*, 1994; Kolaitis and Vardi, 2000].

We will start with the required background of database theory by examining relational structures, relational databases, the concept of conjunctive query and SQL queries. The notions defined in this chapter will be intensely used in the next chapters.

3.1 Relational structures

It is well known that constraint satisfaction problems can be recast as homomorphism problems for relational structures. Let $\mathcal{U}$ and $\mathcal{X}$ be disjoint infinite sets that we call the *universe of constants* and the *universe of variables*, respectively. A (relational) vocabulary τ (also called *schema* in the field of databases) consists of a name r and a finite set of relation symbols of specified finite arities. A *relational structure* $\mathcal{A}$ over τ (short: τ-structure) is an instance of that (relational) schema with

a specific domain. More formally, it consists of a universe $A \subseteq \mathcal{U} \cup \mathcal{X}$ and, for each relation symbol r in τ, of a relation $r^{\mathcal{A}} \subseteq A^{\rho}$, where ρ is the arity of r. For instance, directed and undirected graphs are relational structures where the vocabulary has a single binary relation E, which for digraphs corresponds to the edge relation, while for graphs includes the pairs (x, y) and (y, x) for each edge $\{x, y\}$. In the same way as for graphs, we can define *homomorphisms* of structures. Let $\mathcal{A}$ and $\mathcal{B}$ be two τ-structures with universes A and B, respectively. A *homomorphism* from $\mathcal{A}$ to $\mathcal{B}$ is a mapping $h : A \mapsto B$ such that $h(c) = c$ for each constant c in $A \cap \mathcal{U}$, and such that, for each relation symbol r in τ and for each tuple $\langle a_1, \ldots, a_\rho \rangle \in r^{\mathcal{A}}$, it holds that $\langle h(a_1), \ldots, h(a_\rho) \rangle \in r^{\mathcal{B}}$. For any mapping h (not necessarily a homomorphism), $h(\langle a_1, \ldots, a_\rho \rangle)$ is used, as usual, as a shorthand for $\langle h(a_1), \ldots, h(a_\rho) \rangle$.

A τ-structure $\mathcal{A}$ is a substructure of a τ-structure $\mathcal{B}$ if $A \subseteq B$ and $r^{\mathcal{A}} \subseteq r^{\mathcal{B}}$, for each relation symbol r in τ.

3.2 Relational databases

Let τ be a given vocabulary. A *database instance* (or, simply, a database) D over $D \subseteq \mathcal{U}$ is a finite τ-structure D whose universe is the set D of constants. For each relation symbol r in τ, r^{D} is a *relation instance* (or, simply, relation) of D.

A database schema DS is a finite set $\langle R_1, ..., R_m \rangle$) of relation schemas.

The elements of relations are called *tuples*. In this thesis, we will adopt the logical representation of a database [Ullman, 1989; Abiteboul *et al.*, 1995], where a tuple $\langle a_1, ..., a_\rho \rangle$ of values from D belonging to the ρ-ary relation (over symbol) r is identified with the *ground atom* $r(a_1, ..., a_\rho)$. Accordingly, a database D can be viewed as a set of ground atoms.

3.3 Conjunctive queries

Among all the classes of relational database queries, the most thoroughly analyzed and probably the most important one is the class of Conjunctive queries.

A *Conjunctive query* Q is a first order formula $\exists \bar{X} \Phi$ where $\Phi = r_1(\mathbf{u_1}) \wedge ... \wedge r_m(\mathbf{u_m})$ is a conjunction of atoms (relation names), $r_1, ..., r_m$ (with $m > 0$) are relation symbols, $\mathbf{u_1}, ..., \mathbf{u_m}$ are lists of terms (i.e., variables or constants), and $\bar{X} = X_1, ..., X_n$ is the list of quantified variables of Φ. We say that the query Q is *simple* if every atom is defined over a distinct relation symbol. The conjunction Φ is denoted by $form(Q)$, while the sets of all atoms in Q is denoted by $atoms(Q)$. For any set A of atoms, $vars(A)$ is the set of all variables in A, and $vars(Q)$ is used for short in place of $vars(atoms(Q))$. Moreover, we define $free(Q) = vars(Q) \setminus \{X_1, ..., X_n\}$ as the set of the free variables in Q.

For a database D over D, Q^{D} denotes the set of all substitutions $\theta : vars(Q) \mapsto D$ such that for each $i \in \{1, ..., m\}$, $\theta'(r_{\alpha_i}(\mathbf{u_i})) \in \mathrm{D}$, where $\theta'(t) = \theta(t)$ if $t \in vars(Q)$ and $\theta'(t) = t$ otherwise (i.e., if the term t is a constant).

Note that the conjunction $form(Q)$ can be viewed as a relational structure $\mathcal{Q}$, whose vocabulary τ_Q and universe U_Q are the set of relation symbols and the set of terms occurring in its atoms, respectively. For each symbol $r_i \in \tau_Q$, the relation $r_i^{\mathcal{Q}}$ contains a tuple of terms $\mathbf{u}$, for any atom of the form $r_i(\mathbf{u}) \in atoms(Q)$ defined over r_i. In the special case of simple queries, every relation $r_i^{\mathcal{Q}}$ of $\mathcal{Q}$ contains just one tuple of terms. According to this view, elements in Q^{D} are in a one-to-one correspondence with homomorphisms from $\mathcal{Q}$ to D_Q, where the latter is the (maximal) substructure of D over the (sub)vocabulary τ_Q. Hereinafter, for the sake of presentation, we freely use interchangeably queries and databases with their relational structures, e.g., we

may use Q and D in place of $\mathcal{Q}$ and D_Q. Moreover, $||Q||$ and $||\mathrm{D}||$ denote the sizes of the underlying relation structures, according to their standard encoding (see, e.g., [Grohe, 2007]).

According to the above notation, $\pi_{\mathit{free}(Q)}(Q^{\mathrm{D}})$ is the set of answers of the conjunctive query Q on the database D (over the *output* variables $\mathit{free}(Q)$). We denote by $\mathrm{count}(Q, \mathrm{D})$ the problem of computing the cardinality of this set, i.e., the number of (distinct) answers of Q on D.

3.4 Relational algebra.

For any set $W \subseteq \mathit{vars}(Q)$ of variables and set S of substitutions, we denote by $\pi_W(S)$ the set of the restrictions of the substitutions in S over the variables in W. If θ is a substitution with domain W, then we denote by $\sigma_\theta(S)$ the set $\{\theta' \in S \mid \pi_W(\{\theta'\}) = \{\theta\})$. If S_1 and S_2 are sets of substitutions with domains W_1 and W_2, respectively, then we denote by $S_1 \bowtie S_2$ the set of all substitutions θ over $W_1 \cup W_2$ such that $\pi_{W_1}(\{\theta\}) \subseteq S_1$ and $\pi_{W_2}(\{\theta\}) \subseteq S_2$. We use $S_1 \ltimes S_2$ as a shorthand for $\pi_{W_1}(S_1 \bowtie S_2)$.

3.5 Hypergraphs and structural restrictions

There is a very natural way to associate a hypergraph $\mathcal{H}_\mathcal{V} = (N, H)$ with any set $\mathcal{V}$ of atoms: the set N of nodes consists of all variables occurring in $\mathcal{V}$; for each atom in $\mathcal{V}$, the set H of hyperedges contains a hyperedge including all its variables; and no other hyperedge is in H. For a query Q, the hypergraph associated with $\mathit{atoms}(Q)$ is denoted by $\mathcal{H}_Q$.

Let Q be a conjunctive query, and let $\mathcal{V}$ be a set of atoms each one defined over a specific relation symbol not occurring in Q. These atoms play the role of additional resources that can be used to answer (counting problems on) Q, so that

we are abstracting here all purely *structural decomposition methods* proposed in the literature, which just differ in how they define and build the set of such available resources. Following the framework in [Greco and Scarcello, 2010], we say that $\mathcal{V}$ is a *view set* for Q if, for each atom $q \in atoms(Q)$, $\mathcal{V}$ contains an atom w_q with the same list of variables as q (but with a different relation symbol). Each atom in $\mathcal{V}$ is called a *view*; in particular, atoms of the form w_q are called *query views*. The set of all query views will be denoted by $views(Q)$.

Let D be a database whose vocabulary includes all relation symbols in Q and $\mathcal{V}$. In a structural decomposition method, query views are initialized with the same tuples as their associated query atoms while, for any other view $w \in \mathcal{V}$, w^{D} is initialized by including solutions of some subquery over the variables in w. Using such views for evaluating Q is possible whenever D is a *legal* database (on $\mathcal{V}$ w.r.t. Q), i.e., (i) $w_q^{\mathrm{D}} \subseteq q^{\mathrm{D}}$ holds, for each query view $w_q \in views(Q)$; and (ii) $w^{\mathrm{D}} \supseteq \pi_{vars(w)}(Q^{\mathrm{D}})$, for each view $w \in \mathcal{V}$. Intuitively, all original "constraints" are there, and views are not more restrictive than the original query.

In this thesis, we look for "structural" restrictions to Q guaranteeing that the problem $\mathrm{count}(Q, \mathrm{D})$ can be efficiently solved by exploiting the views in $\mathcal{V}$.

3.6 SQL queries

Structured Query Language *(SQL)* is a declarative language used to define, query and update data in a relational database. In SQL, which is without doubts the most widely accepted database programming language, the user is in charge to specify what he wants rather than how to do it. Indeed, the complex problem of generating an efficient query plan is left to the query optimizer of the DBMS, which will decide how to process and retrieve the data. Not only SQL is interesting due to its wide use,

but also because it is well known that SQL queries without nested statements and inequality comparisons (i.e., Select/Project/Join) are equivalent in expressive power to conjunctive queries. Therefore it is of great interest to propose efficient solutions for SQL query execution plan.

Select/Project/Join queries are generated using the following SQL statement.

$$\texttt{SELECT} \quad R_1(\mathbf{u_1}), ..., R_m(\mathbf{u_m})$$

$$\texttt{FROM} \quad R_1, ..., R_m$$

$$\texttt{WHERE} \quad R_i(\mathbf{u_k}) = R_j(\mathbf{u_z})$$

We next focus on this kind of SQL queries and in additional functionality that includes aggregating data (e.g., counting). We will discuss this in more detail in Chapter 5.

CHAPTER 4

A Weighted Structural Decomposition Technique

4.1 Introduction

In this chapter we move a further step towards the understanding of the structural tractability of constraint satisfaction problems by focusing on a general framework based on the notion of tree projection, which generalizes all known decomposition methods. Indeed we have seen in Chapter 2 that the tree projections unify all purely structural decomposition methods proposed in the literature. However, it is well known that, computing general tree projections is intractable, formally NP-hard [Gottlob *et al.*, 2009]. This could in principle result in a limited applicability of the framework. Motivated by the bad news, in [Greco and Scarcello, 2010] a relevant tractable class of tree projections is identified, known as the class of *greedy tree projections*. In this chapter we design and implement an algorithm for a restricted monotone variant, called monotone greedy tree projection. Up to now, our algorithm, known as GreedyTreeProjection$_{\text{DM}}$ *(in short GTP_{DM})* is the only polynomial-time algorithm for constructing such decompositions. We note however, that in many applications, finding a purely structural decomposition is not the best we can do. For instance, when dealing with CSP evaluation we can think of minimizing the number of vertices of the decomposition tree having the largest width w. Equivalently, in the

related query answering problem we would like to exploit data information such as the size of relations, the selectivity of attributes, indexes, etc. In our framework, we address this issue by extending GTP_{DM} with the novel notion of weighted (monotone) greedy tree projections, in order to combine the purely structural decomposition approach with quantitative aspects. Precisely we equip the decomposition method with an evaluation function that can be used to model many practical contexts where we can exploit additional knowledge on the problem at hand. Therefore we aim at computing greedy tree projections having the smallest weights. To this end, it is worth nothing that, computing a minimal weighted monotone greedy tree projection may be in general harder than computing a (purely structural) one. In this chapter, we still show a polynomial time algorithm for the weighted setting based on a restricted class of evaluation functions. Our results, find application in all those problems that can be solved efficiently on acyclic and quasi-acyclic instances, and in particular, it can be exploited immediately for solving CSPs where constraints are represented as finite relations encoding allowed tuples of values. Finally, we discuss implementation issues and a preliminary testing activity is performed, giving evidence of competitive results compared to the state-of-the-art decomposition methods known in the literature. This is the first step that needs to be done in order to emphasize the practical usefulness of structural approaches to CSP evaluation and query answering. We will return to this in more detail in the next chapter.

4.2 Monotone Greedy Tree projections

Most structural decomposition methods can be characterized through hypergraph games that are variations of the *Robber and Cops* graph game that characterizes the notion of treewidth. In particular, decomposition trees somehow correspond to mono-

tone winning strategies, where the escape space of the robber on the hypergraph is shrunk monotonically by the cops. In fact, unlike the treewidth case, there are hypergraphs where monotone strategies do not exist, while the robber can be captured by means of more complex non-monotone strategies. However, these powerful strategies do not correspond in general to valid decompositions.

In [Greco and Scarcello, 2010] it is given a general way to exploit the power of non-monotone strategies, by allowing a disciplined form of non-monotonicity, that induces valid decomposition trees. Moreover, it is shown that deciding the existence of a (non-monotone) greedy winning strategy (and compute one, if any) is tractable. In fact, greedy strategies can be computed in polynomial time. To establish the result, a useful technical property is that greedy strategies can only involve a polynomial number of configurations (a configuration is a pair (h, C), where h is the hyperedge controlled by the cop, and C is a $[h]$-component where the Robber stands).

Thus, based on them (even on non-monotone ones) it is possible to construct, again in polynomial time, tree projections, which are called greedy. We refer the reader to the work done by [Greco and Scarcello, 2010] for more on this subject. In this thesis, we give an implementation of greedy tree projections based on monotone strategies, a restricted variant that is easier to deal with for our purposes, which include the extension to the optimal weighted framework. Hereinafter, we write simply greedy tree projections for monotone greedy tree projections.

4.3 Normal form

To formalize our results, let us introduce some additional definitions and notations, which we will use later. Given two hypergraphs $\mathcal{H}_1$ and $\mathcal{H}_2$, in general many greedy tree projections with useless redundancies may exist. Hence, we would like

to have a suitable notion of minimality that allow us to identify the most desirable greedy tree projection. In fact, for several structural decomposition methods, normal forms have been defined to restrict the search space of decomposition trees, without loosing any useful decomposition. We make this more precise in the following.

Let $HD = \langle T, \chi, \lambda \rangle$ and $HD = \langle T', \chi', \lambda' \rangle$ be two greedy tree projections. We write $HD \preceq HD'$ if for each $h \in H \setminus H'$, there is a set $h' \in H' \setminus H$ with $h \subseteq h'$, where $H = \{\chi(p) \mid p \in \mathit{vertices}(T)\}$ and $H' = \{\chi'(p') \mid p' \in \mathit{vertices}(T')\}$

Definition 4.1. Let $HD = \langle T, \chi, \lambda \rangle$ be a greedy tree projection of $\mathcal{H}_1$ w.r.t. $\mathcal{H}_2$. We say that *HD* is in *normal form* if the following conditions hold:

1. for each vertex $p \in \mathit{vertices}(T)$, $\chi(T_p) \cap \lambda(p) = \chi(p)$.
2. for each tree projection HD' of $\mathcal{H}_1$ w.r.t. $\mathcal{H}_2$ that satisfies (4), $HD \preceq HD'$ holds.

Hereinafter, we denote by $\mathsf{nfTP}(\mathcal{H}_1, \mathcal{H}_2)$ the set of all greedy tree projections in normal form of $\mathcal{H}_1$ w.r.t. $\mathcal{H}_2$.

Definition 4.2. Assume that a hypergraph $\mathcal{H}$ is given. Let V, W, and $\{X, Y\}$ be sets of nodes. Then, X is said $[V]$-adjacent (in $\mathcal{H}$) to Y if there exists a hyperedge $h \in \mathit{edges}(\mathcal{H})$ such that $\{X, Y\} \subseteq (h - V)$. A $[V]$-path from X to Y is a sequence $X = X_0, \ldots, X_\ell = Y$ of nodes such that X_i is $[V]$-adjacent to X_{i+1}, for each $i \in [0...\ell\text{-}1]$. We say that W is $[V]$-connected if $\forall X, Y \in W$ there is a $[V]$-path from X to Y. A $[V]$-component (of $\mathcal{H}$) is a maximal $[V]$-connected non-empty set of nodes $W \subseteq (\mathit{nodes}(\mathcal{H}) - V)$.

For any $[V]$-component C, let $\mathit{edges}(C) = \{h \in \mathit{edges}(\mathcal{H}) \mid h \cap C \neq \emptyset\}$, and for a set of hyperedges $H \subseteq \mathit{edges}(\mathcal{H})$, let $\mathit{nodes}(H)$ denote the set of nodes occurring in H, that is $\mathit{nodes}(H) = \bigcup_{h \in H} h$. For any component C of $\mathcal{H}$, we denote by $\mathrm{Fr}(C, \mathcal{H})$ the *frontier* of C (in $\mathcal{H}$), i.e., the set $\mathit{nodes}(\mathit{edges}(C))$. Moreover, $\partial(C, \mathcal{H})$

denote the *border* of C (in $\mathcal{H}$), i.e., the set $\mathrm{Fr}(C, \mathcal{H}) \setminus C$. Note that $C_1 \subseteq C_2$ entails $\mathrm{Fr}(C_1, \mathcal{H}) \subseteq \mathrm{Fr}(C_2, \mathcal{H})$. We write simply $\mathrm{Fr}(C)$ or $\partial(C)$, whenever $\mathcal{H}$ is clear from the context.

With all the technical notions in place, we are now ready to present GTP_{DM}.

4.4 An algorithm for computing greedy tree projections in normal form

As we already said in the previous sections, GTP_{DM} is a polynomial time algorithm for computing greedy tree projections in normal form. It is worth noting that our algorithm finds application in all those problems that can be solved efficiently on acyclic and quasi-acyclic instances, and it can be exploited immediately for solving constraint satisfaction problems where constraints are represented as finite relations encoding allowed tuples of values. GTP_{DM} receives as input a constraint hypergraph $\mathcal{H}_1$ and resource hypergraph $\mathcal{H}_2$ and output a greedy tree projection of $\mathcal{H}_1$ w.r.t. $\mathcal{H}_2$ if it exists. It starts by building a directed bipartite graph CG, called *Candidates Graph*, that stores all the informations that we need to compute any greedy tree projection in normal form. The nodes are partitioned in two sets which we call *Component* and *Resource* nodes. *Component* nodes have the form (S, C) where S is some hyperedge contained in $\mathcal{H}_2$ and C is the component which we have to decompose using available *Resource* nodes. In particular, the root node is a special *Component* node $\{(\emptyset, \mathit{nodes}(\mathcal{H}_1))\}$ which represents the entire problem. *Resource* nodes have the form (R, C') where R is some hyperedge contained in $\mathcal{H}_2$ and C' is the set of $[R]$-*component* that have to be decomposed. The node (R, C') has a number of outgoing arcs pointing to all the nodes of the form (S, C) for which it is a candidate solution, that is, for which $\mathrm{Fr}(C) \subset \mathit{vars}(R)$ and $\mathit{vars}(R) \cap \partial(C) \neq \emptyset$ holds. Moreover, (R, C') has a number of incoming arcs for each $[R]$-*component* that is included in C',

as each of these nodes represents a subproblem of (R, C'). If a node $C \in Component$ nodes has no candidate solutions, i.e., if $outgoing(C) = \emptyset$, then it is not solvable. We immediately exploit this information by removing all of its outgoing arcs, for which it was a subproblem. On the other hand, if it has some candidates, whenever all of them have been completely evaluated, it can propagate this information through its outgoing arcs. Once all nodes have been processed, the information encoded in the candidates graph CG is enough to compute every greedy tree projection of $\mathcal{H}_1$ w.r.t $\mathcal{H}_2$ in normal form, if any. One of these greedy tree projections is eventually selected through the recursive procedure SelectGreedyTreeProj.

Algorithm 2 GreedyTreeProjection$_{\text{DM}}$

Input: $\langle \mathcal{H}_1, \mathcal{H}_2 \rangle$ two hypergraphs
Output: $HD = \langle T, \chi, \lambda \rangle$ a *hypertree of* $\mathcal{H}_1$ *w.r.t.* $\mathcal{H}_2$

1: $N_{Root} \leftarrow \{(\emptyset, nodes(\mathcal{H}_1))\}$ ▷ Root component
2: RESOLVE$((N_{Root}))$
3: **if** $incoming(N_{Root}) \geq 1$ **then**
4: $E \leftarrow \{\emptyset\}$ ▷ The tree of decomposition is empty, initially
5: Select $p \in incoming(N_{Root})$
6: **add** the edge $\{p, N_{Root}\}$ to E
7: SELECTGREEDYTREEPROJ(p)
8: **Output** *HD*
9: **else**
10: **Output***failure*
11: **end if**

12: **procedure** SELECTGREEDYTREEPROJ(p)
13: **for all** $q \in incoming(p)$ **do**
14: Select $p' \in incoming(q)$
15: **add** the edge $\{p', q\}$ to E
16: SELECTGREEDYTREEPROJ(p')
17: **end for**
18: **end procedure**

Algorithm 3 Resolve(N_C)

1: $CandidateEdges \leftarrow \{h_R \in edges(H_2) \mid h_R \cap \mathrm{Fr}(N_C) \neq \emptyset\}$;
2: **for all** $cover \in covers(\mathrm{Fr}(N_C), CandidateEdges)$ **do**
3: **if** $vars(cover) \cap \partial(N_C)) \neq \emptyset$ **then**
4: $N_R \leftarrow (cover, vars(cover) \cap (\partial(N_C) \cup \mathrm{Fr}(N_C)))$
5: $N_{Rcomps} \leftarrow findComponents(\partial(N_c), vars(cover))$
6: **for all** $N_{C'} \in N_{Rcomps}$ **do**
7: RESOLVE($N_{C'}$)
8: **if** $incoming(N_{C'}) > 0$ **then**
9: **add** an edge from $N_{C'}$ to N_R
10: **else**
11: break
12: **end if**
13: **end for**
14: **if** $incoming(N_R) = |N_{Rcomps}|$ **then**
15: **add** an edge from N_R to N_C
16: **return**
17: **else continue**
18: **end if**
19: **else continue**
20: **end if**
21: **end for**

4.4.1 Speeding-up computation through greedy coverings

In line 1 of Algorithm 3, rather than trying to decompose the component node using all the available resources, we can restrict the search space by selecting a subset of edges $\mathcal{H}_2$ such that for each $R \in \mathcal{H}_2$ we have that $\mathrm{Fr}(C) \subset vars(R)$. In addition to this, maintaining appropriate data structures (such as a list of "good" resources sorted by the size of $vars(R) \cap \partial(C)$) allows us to quickly decompose the considered component node.

We now examine the complexity of computing pure greedy tree projections in more detail.

Theorem 4.3. *Given a constraint hypergraphs $\mathcal{H}_1$ and a resource hypergraph $\mathcal{H}_2$, Algoritm 2 runs in time $O(nm^2 \log m)$ using dynamic programming, where n is the number of nodes of $\mathcal{H}_1$ and m is the number of edges of $\mathcal{H}_2$.*

Proof. Let $\mathcal{H}_1, \mathcal{H}_2$ be the constraint and the resource hypergraphs respectively. Let

$n = nodes(\mathcal{H}_1)$ and $m = edges(\mathcal{H}_2)$. Observe that, for each edge $R \in \mathcal{H}_2$ there are at most $O(n)$ $[R]$-*components*. Thus, since we store the component nodes, the number of recursive calls to Algorithm 3 is bounded by $O(nm)$. What it remains to analyze is the number of loops in line 2 of Algorithm 3. Observe that, for any component node C the auxiliary procedure **covers** computes a subset of edges $\mathcal{H}_2$ such that for each $R \in covers(\mathcal{H}_2)$ we have that $\mathrm{Fr}(C) \subset vars(R)$. This number is clearly bounded by m. Eventually, to speed up the computation these edges are sorted by the size of $vars(R) \cap \partial(C)$. Therefore, an upper bound for the overall complexity is given by $O(nm^2 \log m)$. Importantly, note that the polynomial runtime is guaranteed under the assumption that already visited components are stored and reused, otherwise, the same subtree could be constructed multiple times which would result in an exponential runtime. □

4.5 Evaluation functions

In this section, we define the notion of weighted greedy tree projection, in order to combine the power of pure greedy tree projections with quantitative approaches. More precisely, we equip our restricted tree projections with evaluation functions that can be used to model situations where we have further information at hand on the given problem, besides its hypergraph representation. For instance, consider concrete contexts like database query answering where we can use statistics such as the size of relations, the selectivity of attributes, indexes, etc., to improve the runtimes of join algorithms. In fact, all commercial database systems exploit such statistics (and do not care about structural properties). In our implementation, users can specify an evaluation function f and the algorithm will compute, if it exists, a f-minimal greedy tree projection.

Now we describe *evaluation functions* in detail. Let $\mathbb{D}$ be a given domain of values, and assume that $\preceq$ is a total order defined on it. An *evaluation function* f over $\mathbb{D}$ associates an element $f(V, h, \mathcal{S}) \in \mathbb{D}$ with each $V, h \subseteq \mathcal{V}$ and multiset $\mathcal{S}$ of elements in $\mathbb{D}$. The value of a greedy tree projection $HD = \langle T, \chi, \lambda \rangle$, with $p = root(T)$, under f is inductively defined as:

$$f(HD) = f\left(\chi(p), \lambda(p), \{f(\langle T_s, \chi, \lambda \rangle) \mid s \text{ is a child of } p\}\right).$$

A greedy tree projection $HD \in \mathsf{nfTP}(\mathcal{H}_1, \mathcal{H}_2)$ is *f-minimal* if $f(HD) \preceq f(HD')$, for each $HD' \in \mathsf{nfTP}(\mathcal{H}_1, \mathcal{H}_2)$. Moreover, we say that an evaluation function is *valuation-monotonic* if for each $V, h, \mathcal{S}, \mathcal{S}'$, $\mathcal{S}' = \mathcal{S} \setminus \{v\} \cup \{v'\}$ with $v \preceq v'$ implies $f(V, h, \mathcal{S}) \preceq f(V, h, \mathcal{S}')$.

Theorem 4.4. *For any valuation-monotonic function f, a f-minimal greedy tree projection can be computed in polynomial time.*

Proof. Consider the algorithm WeightedGreedyTreeProjection$_{\text{DM}}$ shown below, which returns a f-minimal greedy tree projection of a hypergraph $\mathcal{H}_1$ w.r.t. a hypergraph $\mathcal{H}_2$. We note that, this algorithm is derived from Algorithm 2 with a simple change in line 2 where we call Algorithm 5 in place of Algorithm 3. Algorithm 5 is in its turn a variant of Algorithm 3 with the addition of lines 15, 16 and 17. This variant does the same operations performed in Algorithm 2 and in addition for every successful resource node r, it set $weight(r) \leftarrow f(r)$. In particular, for each component node c, we keep only the child having the minimum weight between all children. The decomposition tree is then computed very efficiently using a branch-and-bound technique to avoid exploring solutions that are certainly not optimal. Once the weighted candidates graph is build, all the informations that we need are in place to compute the f-minimal greedy tree projection.

Algorithm 4 WeightedGreedyTreeProjection$_{\mathrm{DM}}$

Input: $\langle \mathcal{H}_1, \mathcal{H}_2 \rangle$ two hypergraphs
Output: $HD = \langle T, \chi, \lambda \rangle$ a *hypertree of* $\mathcal{H}_1$ *w.r.t.* $\mathcal{H}_2$

$N_{Root} \leftarrow \{(\emptyset, nodes(\mathcal{H}_1))\}$ ▷ Root component
RESOLVE$_{\mathrm{W}}((N_{Root}))$
if $incoming(N_{Root}) \geq 1$ **then**
  $E \leftarrow \{\emptyset\}$ ▷ The tree of decomposition is empty, initially
  Select a minimum weighted $p \in incoming(N_{Root})$
  add the edge $\{p, N_{Root}\}$ to E
  SELECTWGREEDYTREEPROJ(p)
  Output *HD*
else
  Output *failure*
end if

procedure SELECTWGREEDYTREEPROJ(p)
  for all $q \in incoming(p)$ **do**
    Select a minimum weighted $p' \in incoming(q)$
    add the edge $\{p', q\}$ to E
    SELECTWGREEDYTREEPROJ(p')
  end for
end procedure

Algorithm 5 RESOLVE$_{\mathrm{W}}(N_C)$

$CandidateEdges \leftarrow \{h_R \in edges(H_2) \mid h_R \cap \mathrm{Fr}(N_C) \neq \emptyset\}$;
for all $cover \in covers(\mathrm{Fr}(N_C), CandidateEdges)$ **do**
  if $vars(cover) \cap \partial(N_C)) \neq \emptyset$ **then**
    $N_R \leftarrow (cover, vars(cover) \cap (\partial(N_C) \cup \mathrm{Fr}(N_C)))$
    $N_{Rcomps} \leftarrow findComponents(\partial(N_c), vars(cover))$
    for all $N_{C'} \in N_{Rcomps}$ **do**
      RESOLVE$_{\mathrm{W}}(N_{C'})$
      if $incoming(N_{C'}) > 0$ **then**
        add an edge from $N_{C'}$ to N_R
      else
        break
      end if
    end for
    if $incoming(N_R) = |N_{Rcomps}|$ **then**
      $weight(N_R) \leftarrow f(N_R)$
      if $weight(N_C) < weight(N_R)$ **then**
        $weight(N_C) \leftarrow weight(N_R)$
        add an edge from N_R to N_C
      end if
    else continue
    end if
  else continue
  end if
end for

The running time of WeightedGreedyTreeProjection$_{\mathrm{DM}}$ is $O(nm^2 \log m)$ from The-

orem 4.3, plus the time needed to compute $f(r)$ for each resource node r. Because we require the function f to be polynomial-time computable, it follows that Algorithm 4 can find a f-minimal greedy tree projection, if it exists, in polynomial time.

□

4.6 Non-monotic valuation functions

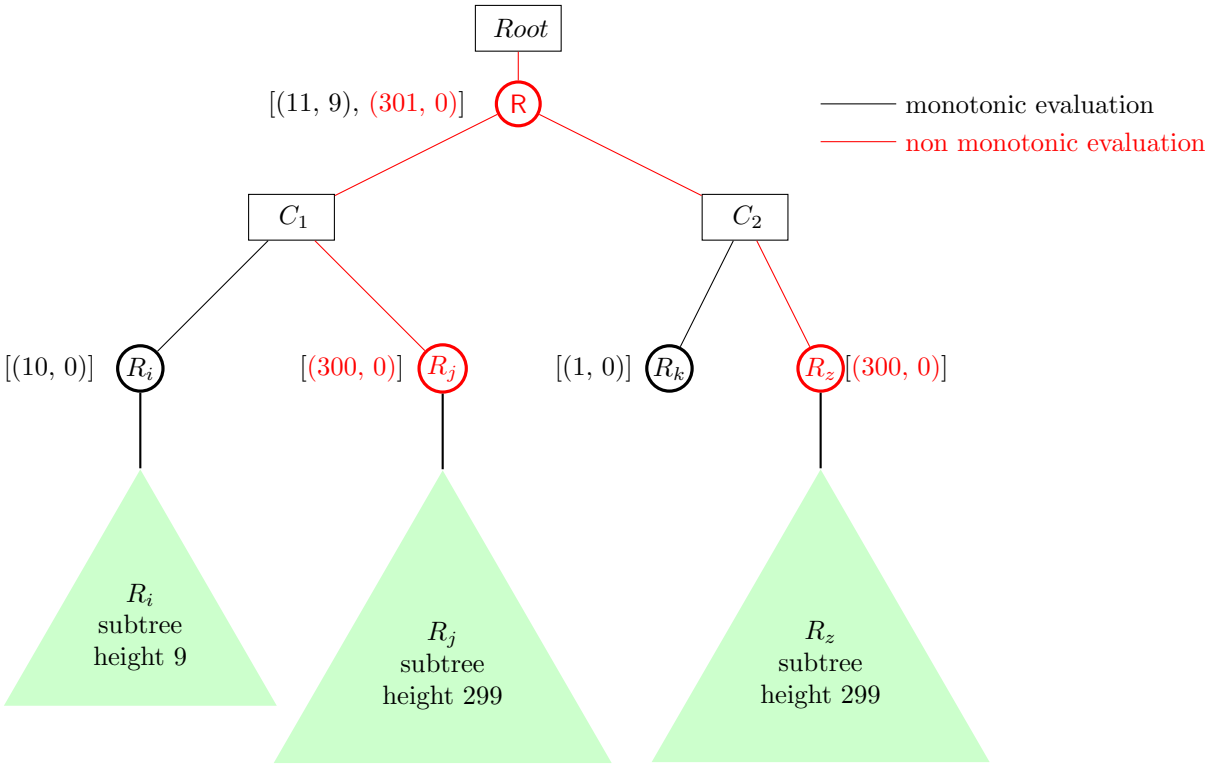


Figure 4.1: Weighted candidates graph in example of Section 4.6

There are many practical situations where we required to process large datasets. Example includes social networks, linked data and also biological motifs. To deal with the complexity of processing such large data, it is usually exploited parallel computing. However, it is known that parallelization performs bad on trees that are unbalanced. Therefore in this setting, we aim at computing the most balanced decomposition. We formalize this idea by defining the evaluation function as follows.

$$\psi(p) = (h(p), i(p)) \quad \text{where}$$

$$h(p) = 1 + \max_{s \in children(p)} h(s),$$
$$i(p) = \max\Big(\max_{s \in children(p)} i(s), \Big(\max_{s \in children(p)} h(s) - \min_{s \in children(p)} h(s)\Big)\Big)$$

Let $h(p)$ be the height and $i(p)$ be the imbalance of the tree T rooted at p and consider the illustration reported in Figure 4.1 to help the intuition. Observe the candidates graph and note that we represent resource nodes with a circle while component nodes are represented with a rectangle. Our objective is to single out the decomposition having the smallest height $h(R)$. From the illustration above we see that the children of R are C_1 and C_2. The weight of R therefore depends on the costs (and hence on the domain values) of C_1 and C_2. Consider C_1 and assume that $\psi(R)$ is valuation-monotonic. Therefore since R_i and R_j have the same imbalance value, i.e., 0, at this level they are interchangeable and thus we choose R_i to solve C_1. After selecting R_i as a solution node for the subproblem node C_1 we move to C_2 and we select R_k for the same reason. Thus we can compute $\psi(R)$ with R_i and R_k being the children of R and we get as result the domain value $(11, 9)$ and hence the $weight(R) = 9$. Now assume to select R_j in place of R_i and R_z in place of R_k. Again at this level, from a local perspective it does not change anything since the imbalance is always 0. However, if we compute again $\psi(R)$ we get a domain value of $(301, 0)$ and hence the $weight(R) = 0$, which is smaller than before. It turns out that if we assume $\psi(R)$ to be valuation-monotonic, we fail at computing most balanced decomposition. We can easily fix this by assuming $\psi(R)$ to be non-monotonic. In this case it easy to see that it can explore all the possible domain values without getting stuck in local minima.

4.7 Experimental results

In this section we present the performances of our implementation compared with those of the approaches listed in Table 4.7 which represent the state-of-the-art algorithms for computing a minimal (purely structural) hypertree decomposition. In particular, *opt-k-decomp* is the first polynomial time algorithm for constructing hypertree decomposition of minimal width less than or equal to a given constant k. *det-k-decomp* is a backtracking-based algorithm for computing hypertree decompositions of width smaller or equal than k. *det-k-decomp* is the best algorithm known for this task up to date. We observe that these methods compute a hypertree decomposition of the given CSP instance, therefore we set our general tool in hypertree decomposition mode. Our experiments evaluate the methods on four sets of instances involving many problems from industry and academic research. This is detailed below.

Problem instance selection. To compare these algorithms, we tried to identify a set of instances that could cover the largest number of application domains. Therefore we identified 4 categories: real-world problems, academic, random and pattern. We performed a random selection of instances in each category. Finally 26 instances were considered in our benchmark.

Hardware and software configuration. Experiments have been performed on a dedicated machine, equipped with an Intel Core i7-3770k 3.5 GHz processor with 12 GB (DDR3 1600 MHz) of RAM, and running Linux Debian Jessie. Our algorithm is implemented in Java and the code was executed on the JDK 1.8.0_05-b13. Moreover, we used a C++ implementation of opt-k-decomp and the source code was compiled with GCC 4.9.3. Regarding det-k-decomp, we downloaded the binary executable

available from here [1]. Default settings were used for the various methods. We set a timeout of 300 seconds to the total execution time of the methods. For each instance we report the average computation time (based on three runs), the number of instances correctly decomposed and the number of wins in finding the minimal width. Of course if a method times out we exclude the related running time from the means. Results are reported in Table 4.7 and4.7.

	Instance	constraints	variables	*Here* width	*Here* time	opt-k-decomp width	opt-k-decomp time	det-k-decomp width	det-k-decomp time
Real world	instExtCW-m1c-ogd-vg16-20	36	320	16	0	–	–	16	0
	instExtCW-m1c-ogd-vg16-29	35	304	16	0	–	–	16	0
	instExtCW-m1c-words-vg16-18	34	288	16	0	–	–	16	0
	instExtCW-m1c-words-vg16-20	36	320	16	0	–	–	16	0
	instExtCW-m1c-lex-vg15-17	32	255	15	0	–	–	15	0
	instExtCW-m1c-lex-vg16-18	34	288	16	0	–	–	16	0
	renault-mod-16-ext	149	111	3	0	+	+	3	0
	renault-mod-32-ext	154	111	5	0	+	+	5	0
	renault-mod-48-ext	149	108	4	0	+	+	4	0
	instExtlangford-2-8	128	16	8	20	+	+	8	0
Academic	magicSquare-3-glb	9	9	1	0	1	1	1	0
	magicSquare-20-glb	400	43	1	0	1	0	1	0
	perfect-square-packing-20-20	953	800	3	0	+	+	3	0
	perfect-square-packing-38-38	3518	2888	3	6	+	+	3	10
	schurr-lemma-100-9-mod	2450	100	–	–	+	+	–	–
	schurr-lemma-66-4	1056	66	–	–	+	+	–	–
	schurr-lemma-50-9-mod	600	50	31	4	+	+	30	0
	schurr-lemma-23-3	210	30	9	5	+	+	8	0
Random	tightness-rand-2-40-8-753-100-80-ext	753	40	37	212	+	+	31	3
	tightness-rand-2-40-8-753-100-99-ext	753	40	41	69	+	+	31	3
	ModelRB-frb59-26-5-mgd-ext	539	59	–	–	+	+	38	0
	ModelRB-frb59-26-3-mgd-ext	547	59	–	–	+	+	37	6
	ModelB-rand-27-27-351-163-63021-ext	351	27	22	30	–	–	19	4
Pattern	graph-coloring-geom-40-6-ext	40	78	3	0	–	–	3	0
	graph-coloring-queens-5-5-5-ext	160	25	13	2	–	–	10	40
	graph-valiente-val49-50	1128	128	3	2	+	+	3	0

Table 4.1: The time (in seconds) required to solve a number of instances generated from real-world, academic, random and pattern problems. A dash (-) means timeout while (+) denotes overflow

Algorithm	# of decomposed instances	# number of wins
Here	22	15
opt-k-decomp	2	2
det-k-decomp	27	23

Table 4.2: Number of correctly decomposed instances and number of wins in finding the minimal width.

From the tables above we note that det-k-decomp, as expected, shows the best overall performances. However GreedyTreeProjection$_{\mathrm{DM}}$ closely follows on almost all

[1] http://www.dbai.tuwien.ac.at/proj/hypertree/downloads.html

instances. opt-k-decomp is the worst one because it frequently times out on difficult instances. We observe that our implementation is not optimized for speed. In fact, we note that our tool is more general than those tools that implement opt-k-decomp and det-k-decomp and in order to reduce development times we were forced to reuse as much code as possible to implement all variants and that inevitably results in performance limitations.

CHAPTER 5

An Hybrid Approach for Counting Solutions

5.1 Introduction to Hybrid Tractability

We have seen in Chapter 2 that the problem of *counting* the solutions of constraint satisfaction problems is equivalent to counting the number of query answers in the related field of database theory. This problem occurs in the presence of SQL queries specifying "COUNT" aggregates. Note that these queries are very often at the basis of decision support systems examining large volumes of data in order to get insights on critical business questions. Dealing with queries processing large amounts of data has been a crucial algorithmic challenge in relational databases. Indeed, the definition and implementation of advanced optimization strategies for SQL queries are topics widely discussed in the literature, but at the same time extremely demanding in order to design efficient RDBMS [Codd, 1970]. The optimizers are asked to calculate the best query plan, using a strategy for evaluating a query over a relational database that provides the best performance compared to all possible strategies. This process, however, forces traditional optimizers to find a good compromise between two conflicting factors [Chaudhuri, 1998]. The first factor is related to the cost of evaluating a query that can be exponential in the length of the query itself [Libkin, 2004]. The second factor is related to the complexity of the calculation

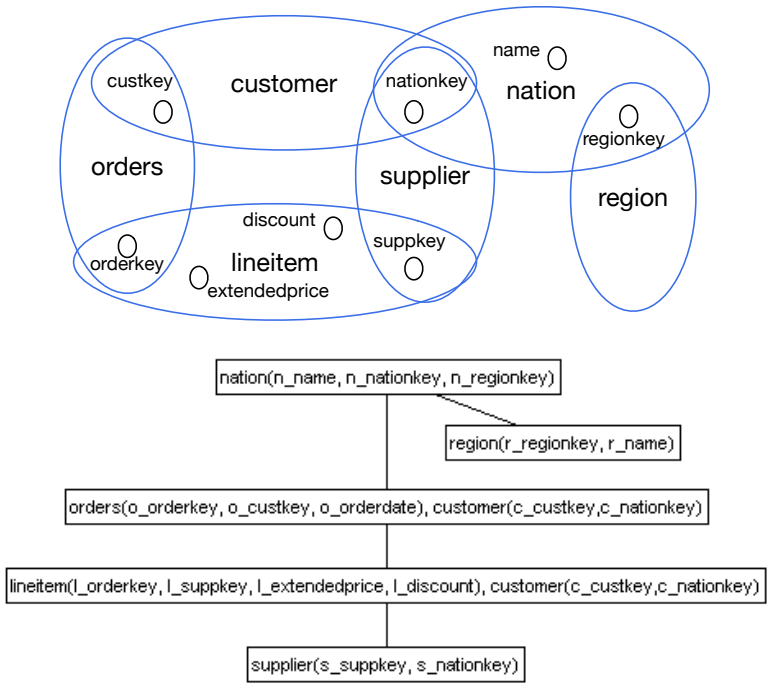


Figure 5.1: The hypergraph $\mathcal{H}(Q_2)$ and a hypertree decomposition of $\mathcal{H}(Q_2)$ of width 2

of the optimal query execution plan; optimizers are forced to limit the search space to query plans with simple structures, thus avoiding a costly search. The choice of the most promising plan, rather than optimal, is determined by a cost model based on quantitative information on the collected data such as the size of database relations and attributes selectivity. For these reasons, traditional optimizers are based on *quantitative methods*. While generally very efficient, these optimizers can not ensure a "bound" on the response time of a query; Furthermore, the execution plan produced by them is often approximate and far away from the ideal plan. To cope with the limitations of traditional optimizers and calculate an optimal query plan that guarantees a polynomial bound on response time, it has been proposed in the community of database theory, a completely different approach. The latter is based on the use of the structural properties of the query, which, as we already noted, are usually represented by the hypergraph associated with the query.

By exploiting these properties, one can respond to a wide class of queries efficiently: this class is known as the class of acyclic queries (or more generally, of bounded hypertree width). As an example, consider the following query Q_2.

```
SELECT n_name,
       sum(l_extendedprice*(1-l_discount)) AS revenue
FROM customer, orders, lineitem,
     supplier, nation, region
WHERE c_custkey = o_custkey
      and l_orderkey = o_orderkey
      and l_suppkey = s_suppkey
      and c_nationkey = s_nationkey
      and s_nationkey = n_nationkey
      and n_regionkey = r_regionkey
      and r_name = '[REGION]'
      and o_orderdate >= date '[DATE]'
      and o_orderdate < date '[DATE]' + interval '1' year
GROUP BY n_name ORDER BY revenue desc;
```

Observe its associated hypergraph $\mathcal{H}(Q_2)$ that is depicted on the top of Figure 5.1 and the corresponding hypertree decomposition of $\mathcal{H}(Q_2)$ on the bottom.

Q_2 can be now answered using a two-step plan. First, one computes for each cluster of the decomposition tree, the join of its nodes. Then, starting from the leaves, one moves bottom-up towards the root, by performing upward semi-joins of intermediate results.

Despite their very nice computational properties, structural decomposition methods have not had any serious impact on the design of commercial database optimizers, and the interest for these techniques mainly remained at a theoretical level. This is mainly because decomposition methods flatten in the query hypergraph all the quantitative aspects of data, do not take care of the output of the variables, and do not support aggregate operators and nested queries. Therefore, since we are dealing with a setting where the query and the database are both given as input (neither is constant), it is meaningful to consider decomposition techniques that are able to exploit structural properties of the query in combination with properties of the given data, such as functional dependencies or any other feature that may simplify the evaluation. Intuitively, such features may induce different structural properties that

are not identified by the "worst-possible database" perspective of purely structural methods. The motivating idea is to end up with algorithms that can be practically applied over a wide range of real world settings where, in particular, such methods do not suffice alone.

[Pichler and Skritek, 2013] have recently proposed an algorithm for counting answers to acyclic conjunctive queries, whose scaling is in the worst case exponential w.r.t. the maximum number of tuples over the database relations, denoted by m hereinafter. In fact, they also shown that counting query answers remains #P-hard over this structurally simple class. Moreover, they pointed out that their algorithm can be extended to queries that are not necessarily acyclic, but have bounded *hypertree width*. [Greco and Scarcello 2014], moved a first step towards proposing a hybrid decomposition method for counting problems. In the next section we analyze such an extension. Finally we give an implementation and we discuss the issues related to this algorithm, we also present preliminary experimental results.

Before moving to the technical details of the proposed algorithm, note that for the sake of presentation, we mostly focus on the database setting and the query answering problem. However, we remark that all results can immediately be applied to all problems that can be recast as constraint satisfaction problems (as we shall see with a tiny example).

5.2 An algorithm for counting answers

We recall that a *hypertree decomposition* [Gottlob *et al.*, 2002] of a conjunctive query Q is a generalized hypertree decomposition enjoying an additional property, often called *descendant condition*. In particular, a *hypertree* for a query Q is a triple $\langle T, \chi, \lambda \rangle$, where T is a rooted tree, and χ and λ are labeling functions associating

each vertex p of T with two sets $\chi(p) \subseteq \mathit{vars}(Q)$ and $\lambda(p) \subseteq \mathit{atoms}(Q)$. The set of the vertices of T is denoted by $\mathit{vertices}(T)$, whereas its root is denoted by $\mathit{root}(T)$. Moreover, for any $p \in N$, we denote by T_p the subtree of T rooted at p (hence, $T = T_{\mathit{root}(T)}$), and by $\chi(T_p)$ the set of all variables occurring in the χ labeling of T_p.

A *hypertree decomposition* of Q is a hypertree $HD = \langle T, \chi, \lambda \rangle$ for Q such that: (1) for each $q \in \mathit{atoms}(Q)$, there exists $p \in \mathit{vertices}(T)$ such that $\mathit{vars}(q) \subseteq \chi(p)$; (2) for each $X \in \mathit{vars}(Q)$, the set $\{p \in \mathit{vertices}(T) \mid X \in \chi(p)\}$ induces a (connected) subtree of T; (3) for each $p \in \mathit{vertices}(T)$, $\chi(p) \subseteq \mathit{vars}(\lambda(p))$; and (4) for each $p \in \mathit{vertices}(T)$, $\mathit{vars}(\lambda(p)) \cap \chi(T_p) \subseteq \chi(p)$ (descendant condition). The decomposition HD is said *complete* if for each atom $q \in \mathit{atoms}(Q)$, there exists $p \in \mathit{vertices}(T)$ such that $q \in \lambda(p)$.

This notion is a true generalization of acyclicity, as acyclic queries are precisely those having hypertree width 1. Importantly, for any fixed natural number $k \geq 1$, deciding whether a query has hypertree width at most k is in LOGCFL, and thus it is a tractable and highly parallelizable problem [Gottlob *et al.*, 2002].

Finally for each vertex p of T we define $r_p = \pi_{\chi(p)}(\bowtie_{q \in \lambda(p)} q^{\mathrm{D}})$, and $\delta(p)$ the number of its children, which we assume to be ordered in some way; If R and R' are sets of sets of substitutions, then we define an ad hoc semi join as $R \ltimes R' = \{S \ltimes S' \mid S \in R, S' \in R', S \ltimes S' \neq \emptyset\}$;

Algorithm 6 counting-answers

Input: $\langle \mathcal{H}_1, \mathcal{H}_2 \rangle$ two hypergraphs
Output: $HD = \langle T, \chi, \lambda \rangle$ a *hypertree of* $\mathcal{H}_1$ *w.r.t.* $\mathcal{H}_2$

1: **for all** $p \in vertices(T)$ **do** ▷ **(1)** —— **Initialization**
2: $R_p^0 \leftarrow \{\sigma_\theta(r_p) \mid \theta \in \pi_{free(Q)}(r_p)\}$
3: **end for**
4: $BU := \{\}$; ▷ **(2)** —— **Bottom-Up**
5: **while** $BU \neq vertices(T)$ **do**
6: **Let** $p \notin BU$ whose children $q_1, ..., q_{\delta(p)}$ are all in BU
7: $BU := BU \cup \{p\}$
8: **for** $\alpha := 1$ **to** $\delta(p)$ **do**
9: **Let** q be the α^{th} child of p
10: with its computed $R_q := R_q^{\delta(q)}$ and $c_q = c_q^{\delta(q)}$
11: $R_p^\alpha := R_p^{\alpha-1} \ltimes R_q$;
12: $c^\alpha(S) := \sum_{(*)} c_p^{\alpha-1}(S_p) \times c_q(S_q)$, for each $S \in R_p^\alpha$
13: $(*) : S_p \in R_p^{\alpha-1}, S_q \in R_q \mid S = S_p \ltimes S_q$;
14: **end for**
15: **end while**
16: **return** $\sum_{S \in R_r^{\delta(r)}} c_r^{\delta(r)}(S)$, where $r = root(T)$; ▷ **(3)** —— **Finalization**

With the above notation in place, consider Algorithm 6, which takes as input a complete hypertree decomposition $HD = \langle T, \chi, \lambda \rangle$ of a query Q, and a database D. The algorithm works as follows: In the initialization step, for each vertex p, the set r_p (of the partial solutions over p) is partitioned according to the profiles given by the configurations of values allowed for the free variables in $\chi(p)$. Let R_p^0 be the resulting partition. In the subsequent steps, p will be associated with a set R_p^α where $\alpha \in \{0, 1, ..., \delta(p)\}$, with $\delta(p)$ being the number of children of p. In particular, each of these sets consists of a set of substitutions that are manipulated via an ad-hoc semi join "$\ltimes$" operator such that: if R and R' are sets of sets of substitutions (hereinafter called *#-relations*), then $R \ltimes R' = \{S \ltimes S' \mid S \in R, S' \in R', S \ltimes S' \neq \emptyset\}$. In the bottom-up evaluation, upward semi joins are performed, and R_p^α is the #-relation associated with p after that the first α children (according to some fixed ordering) have been processed. Each set $S \in R_p^\alpha$ is associated with a value $c_p^\alpha(S)$ initialized to 1 and updated bottom-up. At the end, the value computed in the finalization phase is returned as the desired number of solutions. The correctness of this algorithm follows

from the arguments in [Pichler and Skritek, 2013]. In fact, because the running time of the algorithm in [Pichler and Skritek, 2013] is in general exponential w.r.t. m, [Greco and Scarcello, 2014] introduced the following notion to keep under control such a blow-up.

Definition 5.1. Let $b \geq 0$ be a natural number and let $HD = \langle T, \chi, \lambda \rangle$ be a hypertree (not necessarily decomposition) of a query Q. We say that a database D is *b-unbound for Q w.r.t. HD* if there is a vertex v of T and a substitution $\theta \in \pi_{free(Q)}(r_v)$ such that

$$|\sigma_\theta(r_v)| > b, \qquad \text{where } r_v = \pi_{\chi(v)}(\bowtie_{q \in \lambda(v)} q^{\mathrm{D}}).$$

The minimum b for which D is not b-unbound is the *boundness* of D for Q w.r.t. HD, and it is shortly denoted by $bound(\mathrm{D}, HD)$. □

Intuitively, the boundness value $bound(\mathrm{D}, HD)$ for Q provides an estimate on the size of the information that is required to flow along the given hypertree decomposition HD in order to answer a counting problem over D. Finally they proved the following theorem.

Theorem 5.2. *Let $HD = \langle T, \chi, \lambda \rangle$ be a width-k hypertree decomposition of Q, and let* D *be a database such that $bound(\mathrm{D}, HD) \leq h$. Then, Algorithm 6 runs in time $O(|vertices(T)| \times m^{2\times k} \times 4^h)$.*

5.3 Implementation issues and System Architecture

In the last section, we discussed Algorithm 6 which can especially be exploited to find all solutions to a counting problem by considering the whole set of variables as output ones. However, it should be noted this high level description is of great interest mainly from a theoretic perspective. Indeed a verbatim implementation will

not perform very well in practical settings where the domain of each output variable even if bounded can still contain a very high number of values. This is especially true when counting CSP solutions for classes of instances having not only a large domain but also a huge number of constraints. In fact, the algorithm requires to create a "view" for each possible tuple thus generating a terrible overhead which soon vanish all the nice theoretical computationally properties. Therefore, in complex real-world settings, this observation once more strengthen the choice of backtracking approaches based on arc-consistency and propagations over the structural ones. Indeed it is not by chance that all the CSP and SAT solvers to date completely discard decomposition methods. In this dissertation we show how to efficiently exploit our tools both standlone or to help reduce the search space of backtracking approaches.

We observe that when counting all solutions, at low level we do not really need to create a "subview" for each tuple of the relation . Indeed, it suffices to virtually augment the relation and hence each tuple with a new attribute encoding the number of extensions of that tuple. This value is initially set to 1 for each tuple and is then updated through an ad-hoc semi join operator in the bottom-up process. We give details of this operation in the next section. Intuitively our idea is to exploit the acyclicity and our semi join operator to efficiently compute the counting task. Indeed, we focused in the database setting by designing an HybridOptmizer that works on top of PostgreSQL (and possibly of any other DBMS) and is in charge of computing on optimal query plan for queries specifying the "count" aggregate. The architecture is depicted in Figure **??**. Given an SQL query, the **Parser** preliminary verifies its correctness. To this aim, we exploit the syntactical analyzer J-SQLParser[1] that has been further enhanced to check whether the query complies with the un-

[1]Available at http://jsqlparser.sourceforge.net/

derlying database schema. The resulting parse tree is then processed by the **Query Rewriter**, which prepares all data structures needed for structural analysis and optimization. In particular, the module modifies the parse tree by associating each nested SQL statement with a distinguished node storing all information to evaluate it independently of the others, in particular by propagating join constraints on the basis of the scope of attributes and relations. The **HybridOptimizer** module is the main responsible for the structural optimization. Its implementation relies on the decomposition technique described in Chapter 4 exploiting an evaluation function based on the cardinality of relations and the selectivity of attributes as presented in [Scarcello *et al.*, 2007].

In a nutshell, the optimizer works as follow. Starting from the leaves, it navigates the rewritten parse tree and calls on each node (i.e., subquery) the **Subplan Handler**, which is responsible for computing a suitable optimized plan for it. The handler preliminary invokes the **Hypergraph Manager**, which builds the hypergraph associated with the current input node. The **Hypergraph Manager** collects also information about statistics of the relations and attributes appearing in the node, by invoking the underlying system catalog. The computed hypergraph and its associated statistics are then passed to the decomposition module that computes a *hypertree decomposition*, whose width is fixed by the user at configuration time. Note that, according to the q-decomposition strategy, described in [Scarcello *et al.*, 2007] we force all output variables (i.e., selected items, attributes appearing in group by or in order by clauses) to appear in the root of the decomposition. Furthermore, we also request that the attributes appearing in some non-join condition are covered together by at least a cluster of the computed decomposition.

Finally, the decomposition is translated in an actual query plan by the **View**

Builder, which rephrases it in terms of a tree of SQL views, possibly materialized. During this phase, all non-join constraints are assigned to the node of the decomposition, for being later evaluated. The computed optimized plan is then returned to the **SubPlan Hadler** and finally to the **HybridOpitmizer** module, which arranges the computed plan node as an element of the current query plan tree. Once all nodes have been process, the final plan tree gives us the query rewrite. The key to implementing this query rewrite idea is to develop a new ad-hoc physical semi-join operator and to integrate it within the PostgreSQL database management system.

5.3.1 Hacking PostgreSQL

In this section we detail the implementation of the ad-hoc *Semi Join* operator. We start by sorting the two tables and scan them looking for a match. When a match is found, we stop moving on the outer table, we mark a pointer to the last tuple visited, set a counter, and continue scanning inner tuples until they have a match with the marked tuple. During this scan, we properly update the counter at each match. When no more matches occur on the inner, we return the marked tuple and the counter, and start again moving on the outer table. Each outer tuple matching the same set of inner tuples of the last marked one is immediately returned (without a rescan of the inner table) with the last computed counter. Note that we implemented the *Semi Join* operator inside Postgresql 9.0.2. That is because our work is based on the preliminary work published in **??**. Thus to reduce development times we decided to extend the implementation that was already available with the new ideas. However, to stress the effectiveness of our approach, in the next section we compare our hybrid 9.0.2 with the more recent 9.3.5 regular version. We are pretty sure that if we can get good results with an older version of PostgreSQL, than we can expect to be even more fast with the latest releases.

5.4 Some experiments

This section reports the results of the testing activity that we conducted in order to evaluate the effectiveness of hybrid 9.02 compared to the regular 9.3.5. Before we discuss the results, let us describe the methodology used.

Dataset. We evaluate the performances of on the TPC-DS benchmarks. TPC-DS is the de-facto industry standard benchmark for measuring the performance of modern decision support systems including, but not limited to, Big Data systems. It models analytical workloads (reporting, data mining, iterative OLAP and so on), including SQL queries of various complexity, i.e., performing aggregations (GROUP BY), various joins on large amounts of data. This benchmark effectively extends and deprecates TPC-H benchmark, improving it in multiple ways to make it broadly representative of modern decision support systems. Firstly, it makes the schema more complex (e.g. more tables), and uses less uniform distributions of the data (which makes cardinality estimations way more difficult). It also increases the number of query templates from 22 to 99, and uses modern features like CTEs, window functions and grouping sets. As recommended by TPC-DS specification we decided to use 16GB of data, specified as a parameter to the *DSGEN* tool used to create the CSV file. After loading the CSV into the database, the resulting size was 80GB (this is probably due to overhead, indexes, etc.). System configuration. We ran the experiments on a Core i7 Machine equipped with 12 GB of RAM and running Linux Debian Jessie. We installed from source the three variants of the PostgreSQL DBMS: version 9.3.5, version 9.0.2 and hybrid 9.0.2 (i.e., with structural optimization). We used GCC 4.9.3 to compile the code. Moreover, we use the default configuration for all the versions. Before we give the results, it is convenient to show one of the query

templates involved in our testing.

```
select
        cc_call_center_id as Call_Center,
        cc_name as Call_Center_Name,
        cc_manager as Manager,
        sum(cr_net_loss) as Returns_Loss
from
        call_center,
        catalog_returns,
        date_dim,
        customer,
        customer_address,
        customer_demographics,
        household_demographics
where
           cr_call_center_sk       = cc_call_center_sk
and     cr_returned_date_sk     = d_date_sk
and     cr_returning_customer_sk= c_customer_sk
and     cd_demo_sk              = c_current_cdemo_sk
and     hd_demo_sk              = c_current_hdemo_sk
and     ca_address_sk           = c_current_addr_sk
and     d_year                  = [YEAR]
and     d_moy                   = [MONTH]
and     ( (cd_marital_status       = 'M' and cd_education_status     = 'Unknown')
and     hd_buy_potential like '[BUY_POTENTIAL]%'
and     ca_gmt_offset           = [GMT]
group by cc_call_center_id,cc_name,cc_manager,cd_marital_status,cd_education_status
order by sum(cr_net_loss) desc;
```

In particular, this query template consists of a join between 6 tables and aggregations. This template is of average complexity compared to others. Moreover not all the templates are compatible with PostgreSQL. Indeed we are able to execute 41 queries out of 99. The first set of experiments compares the performance of the regular 9.0.2 version with that of 9.3.5. This preliminary test will help us to better judge the effectiveness of our approach. Figure 5.4 shows the average computation times of the two versions over 41 queries. The speedup of PostgreSQL 9.3.5 compared to the regular version 9.0.2 is about 3x. This is good news for us, because things get interesting when we turn on our structural optimization. We recall that our hybrid approach works only for counting queries, although it can easily be extended also to other aggregates. Therefore, we restrict the attention to queries involving only the count aggregate.

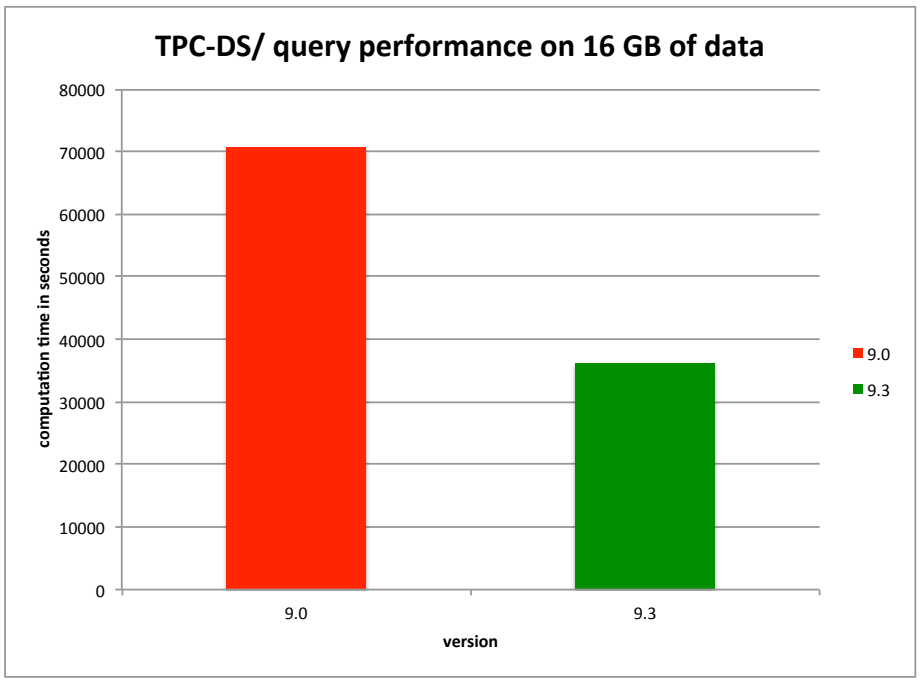


Figure 5.2: The speedup of PostgreSQL 9.3.5 compared to 9.0.2

5.4.1 Query 3 (Q3) on TPC-DS

```
select count(i_item_id)
from item, inventory, store_sales
where inv_item_sk = i_item_sk and inv_quantity_on_hand >= 100
and inv_quantity_on_hand <= 200  and ss_item_sk = i_item_sk
```

Query *Q3* is a simple acyclic query consisting of three joins which asks to count all sold item_id from the store channel with an inventory quantity on-hand between 100 and 200. Note that we are not counting distinct types of item but we are counting all the occurrences (i.e., answers). At this point, the hybrid optimizer performs the set of operations that we discussed in the previous sections and yields to the following query rewrite.

```
SELECT sum(VT1P.OUTER_COUNT)
FROM (
    SELECT  VT1.i_item_id,VT1.i_item_sk,VT1.OUTER_COUNT AS OUTER_COUNT,1 AS INNER_COUNT
    FROM
        (SELECT item.i_item_sk AS i_item_sk, item.i_item_id AS i_item_id, 1 AS OUTER_COUNT
         FROM store_sales, item
         WHERE item.i_item_sk=store_sales.ss_item_sk OFFSET 0
        ) VT1 SEMI JOIN inventory ON VT1.i_item_sk=inventory.inv_item_sk
        and inventory.inv_quantity_on_hand>=100
```

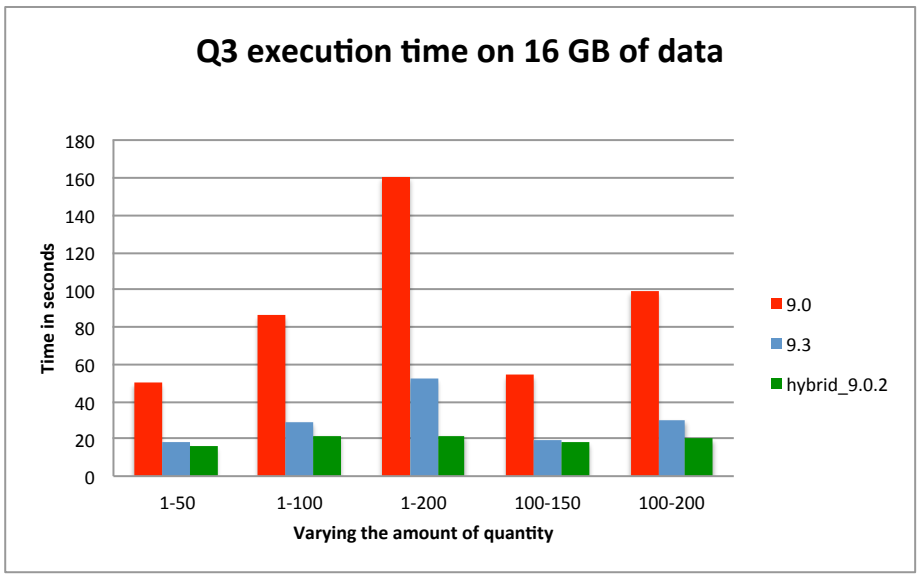


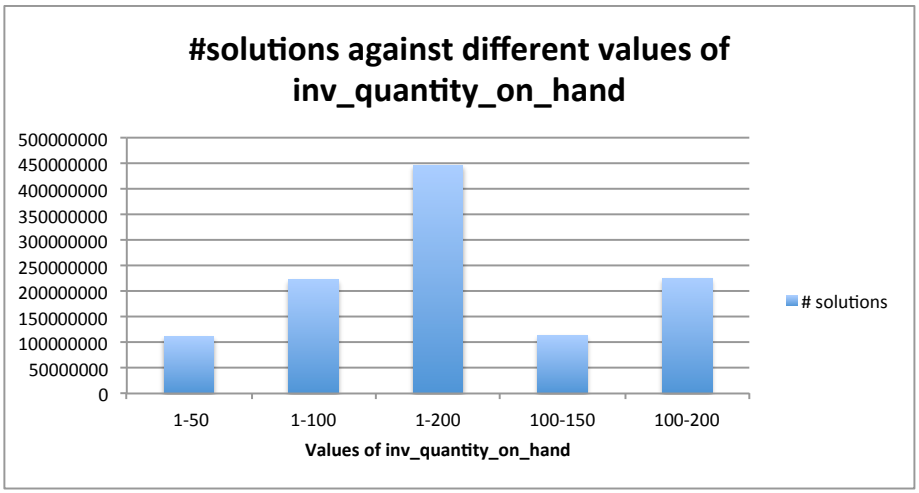
Figure 5.3: Execution time for query Q3

```
        and inventory.inv_quantity_on_hand<=200 OFFSET 0
    ) VT1P
```

We observe that in the query rewrite, we exploit the ad-hoc $SEMI\ JOIN$ operator in addition to the nested views in which we store the counters which we need to compute the solutions. The comparison of the execution times of the three variants is shown in Figure 5.4.1.

In order to assess, in a deeper and more systematic manner, the capability of our approach to exploit both structural and statistics information, the tests were carried out on the same scenario while using different values for the attribute inv_quantity_on_hand. We observe that hybrid 9.0.2 has the lowest runtime followed closely by regular 9.3.5. In particular a noticeably boost seems to be achieved when the number of computed solutions increases Figure 5.4.1. The reason is that the regular optimizer produces large intermediate results, as shown by the "explain analyze".

```
QUERY PLAN
Aggregate  (cost=2732592.31..2732592.32 rows=1 width=17)
(actual time=302591.508..302591.509 rows=1 loops=1)
  ->  Hash Join  (cost=214476.16..618195.75 rows=845758624 width=17)
        (actual time=11291.097..204287.431 rows=449493233 loops=1)
        Hash Cond: (inventory.inv_item_sk = item.i_item_sk)
```



[h]

Figure 5.4: Number of solutions against different values for the attribute inv_quantity_on_hand

```
        ->  Seq Scan on inventory  (cost=0.00..239309.00 rows=58725 width=4)
              (actual time=2.414..7619.955 rows=2241403 loops=1)
              Filter: ((inv_quantity_on_hand >= 0) AND (inv_quantity_on_hand <= 200))
        ->  Hash  (cost=160187.11..160187.11 rows=2880404 width=25)
              (actual time=11282.395..11282.395 rows=2880404 loops=1)
              Buckets: 2048  Batches: 256  Memory Usage: 999kB
              ->  Hash Join  (cost=1742.00..160187.11 rows=2880404 width=25)
                       (actual time=112.532..8689.848 rows=2880404 loops=1)
                    Hash Cond: (store_sales.ss_item_sk = item.i_item_sk)
                    ->  Seq Scan on store_sales  (cost=0.00..88240.04 rows=2880404 width=4)
                             (actual time=1.782..4777.135 rows=2880404 loops=1)
                    ->  Hash  (cost=1411.00..1411.00 rows=18000 width=21)
                                    (actual time=110.520..110.520 rows=18000 loops=1)
                          Buckets: 2048  Batches: 2  Memory Usage: 529kB
                          ->  Seq Scan on item  (cost=0.00..1411.00 rows=18000 width=21)
                                   (actual time=1.232..89.643 rows=18000 loops=1)
Total runtime: 302591.836 ms
```

5.4.2 Query 4 (Q4) on TPC-DS

Query (Q4) is an example of knowledge extraction in the TPC-DS decision support workload. It asks for all the distinct goods from sales comprising both a good of category "Women" and a good of category "Jewelry".

```
select count(distinct i3.i_item_id)
from item i1, store_sales s1, item i2, store_sales s2, item i3, store_sales s3
where s1.ss_item_sk = i1.i_item_sk and s2.ss_item_sk = i2.i_item_sk and
      s1.ss_ticket_number = s2.ss_ticket_number and i3.i_item_sk = s3.ss_item_sk
      and s1.ss_ticket_number = s3.ss_ticket_number and i1.i_category='Jewelry'
      and i2.i_category='Women'
```

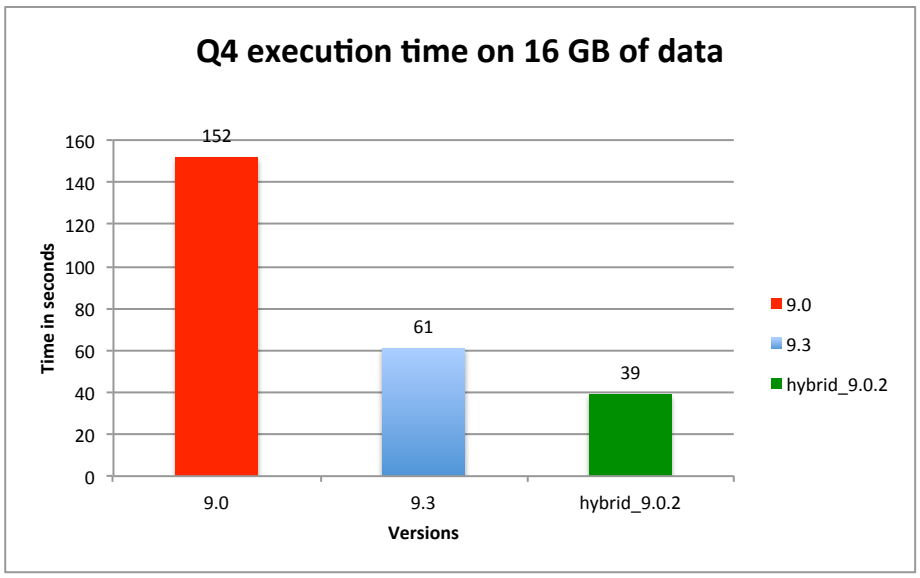


Figure 5.5: Execution time for query Q4

This query is an acyclic query involving 6 joins. The total number of distinct answers for this query is only 8952. Figure 5.4.2 shows the performance the three optimizers for this query. Interestingly, when counting distinct solutions, even if the number of solutions is small, the advantage of hybrid 9.0.2 over regular 9.3.5 is emphasized. To know why this happens, we take again a closer look at the cost of the query plan generated by the 9.3.5 optimizer reported in Figure **??** and we see that the bottleneck for the 9.3.5 is given by the cost to generate the large intermediate results (i.e., *item join store_sales*). Thus acyclicity plays a crucial role in this setting which goes well beyond all the gains obtained from the sophisticated quantitative machinery that traditional DBMS often perform.

Part II

CSP and Game Theory

CHAPTER 6

An Introduction to Game Theory

Game theory is one of the most important field in economy and mathematics. It is concerned with analyzing the strategic interactions that occur in systems of intelligent agents, both in *noncooperative* (where agents are pursuing their individual goals) and in *cooperative* contexts (where they jointly pursue some common goal). Noncooperative game theory, model those situations where players make individual choices since they are unable to cooperate with other players (or without additional benefits), and thus can apply even in situations where the interests of different agents are not in conflict. In this context, a prominent role is assumed by the concept of Nash equilibrium. This concept, that is certainly the most influential solution concept to date in the field of game theory, model in fact the rational behavior of players whom, having available a set of possible strategies with associated utility functions (that establish the payoffs that each player receives), reach a stable condition where each of them has selected an optimal strategy (with respect to the action of any other player) and thus has no incentive to deviate from it. Again, even if she knew what strategies the other agents were following, she would not want to change her strategy. Deciding whether a game admits a Nash equilibrium and eventually computing it efficiently, are problems of great interest to the scientific community. In

the case of games with mixed strategies, i.e., regulated by probability distributions, the existence of an equilibrium is guaranteed by the famous Nash's theorem, and recently the computational complexity of computing the Nash equilibrium has been fully characterized. However, in this thesis our studies are devoted to cooperative games, which have been introduced by the AI community to provide a solid mathematical framework to study scenarios where agents can obtain higher worths by collaborating with each other rather than by acting in isolation (see, e.g., [Nisan *et al.*, 2007; Osborne and Rubinstein, 1994]). However, in many real-world application, not all the outcomes are equally fair (or rational), and as a consequence we need suitable *mechanisms* for coalition formation such that no group of agents would earn more by leaving the coalition for another potential coalition. In other words, we require coalition formation mechanisms to enforce the agreements between agents. In this chapter we proceed as follows. First, we formally define the most widely used model of coalitional games and we discuss a special class of coalitional games known as *allocation games*. Then we study coalition formation and we analyze how the payoff obtained by the coalition can be divided fairly among the agents of that coalition. This is addressed by discussing the notion of solution concepts; we describe the most important solution concepts used in cooperative game theory by focusing on the concept of *Shapley value*. We conclude the chapter with a real-world example of allocation game.

6.1 Coalitional games

Coalitional (i.e., cooperative) games provide a rich mathematical framework to analyze interactions between intelligent agents. Before we start studying these games, it is helpful to formalize a summary of relevant concepts as follows. In abstract

terms, a *coalitional game* $\mathcal{G}$ is a tuple $\langle N, v\rangle$, where N is a set of agents and v is a function associating each coalition $C \subseteq N$ with the worth that agents in C can guarantee to themselves. In this thesis we restrict our attention to coalition games with transferable utilities. In the Transferable Utility (TU) setting, coalition worths can be freely distributed within a coalition, while in the non-Transferable Utility setting (NTU) coalitions are allowed to distribute worths only in some specified configurations, called consequences [Osborne and Rubinstein, 1994]. In fact, the crucial problem is to single out the most desirable distributions (of the worth associated with the grand-coalition N), usually called *solution concepts*, which can be perceived as fair and stable. Therefore, *Coalitional games* can be formalized as tuples $\mathcal{G} = \langle N, v\rangle$ where each *coalition* $C \subseteq N$ is associated with a real value $v(C)$ meant to encode the worth that agents in C obtain by collaborating with each other. The function v is *supermodular* (resp., submodular) if $v(R \cup T) + v(R \cap T) \geq v(R) + v(T)$, (resp., $v(R \cup T) + v(R \cap T) \leq v(R) + v(T)$) holds for each pair of coalitions $R, T \subseteq N$. After this brief formal description, we recall that in a coalition game, we are not interested in finding how agents within a coalition act, that is we are not concerned with the detail of their agreements, rather we look only at the payoff taken by the coalition. Thus we would like to answer these two basic questions:

1. Among all possible coalitions, which coalition will be formed?
2. How that coalition should distribute its worth among its members ?

Often, the answer to (1) is the "grand coalition", i.e, the coalition that includes all the agents in N. However, in many situations this answer depends also on the choice that has been made by taking into consideration (2).

Example 6.1. As an example of a coalitional game, consider the *voting game*. We have a set of agents N and for each coalition $C \subseteq N$, C is a *winning* coalition (i.e.,

group of agents who can come together to "win"), if it is sufficient to pass a bill into a law. Coalitions that are not willing to pass the law are called *losing* coalitions. For a *winning* coalition C we have $v(C) = 1$, and otherwise $v(C) = 0$. Therefore, every "single" agent coalition is *losing* while the *grand coalition* is *winning*.

Example 6.2. As another example, consider that there is a research project and you can choose with which colleague you will work. Then a number of questions may come to mind: for instance, do I want someone who is easy to work with or do I prefer a talented colleague, etc.? Of course, every colleague will have the same reasoning. Therefore, a condition that must be satisfied in order to form a coalition is that all the agents inside that coalition are willing to stay. Since no one has an interest to leave, we can say that the coalition that has formed is *stable*. It follows from this discussion that stability is a necessary condition. A natural question is whether stability is also a sufficient condition. We will answer this question in Section 6.4.

In coalitional game theory, an interesting property often assumed for the function v is the notion of *superadditivity*. Formally v is superadditive if $v(S \cup T) \geq v(S) + v(T), \forall S, T \subset N, S \cap T = \emptyset$. In words, this property says that cooperation leads to higher utilities for each player than the individual action. This is true if coalitions can work without interfering with one another. Note that, every game can be transformed to its superadditive form with the intuition that in the worst case agents can agree to not cooperate in order to obtain greater payoffs. Note that superadditivity has an important implication for how agents will form coalitions. In fact, the grand coalition will always form since its utility is higher than the utility of any other coalition. Moreover, if coalitions can never affect one another, either positively or negatively then we have another form of game called *additive* games. Formally, a game is additive if $v(S \cup T) = v(S) + v(T), \forall S, T \subset N, S \cap T = \emptyset$. In the rest of this

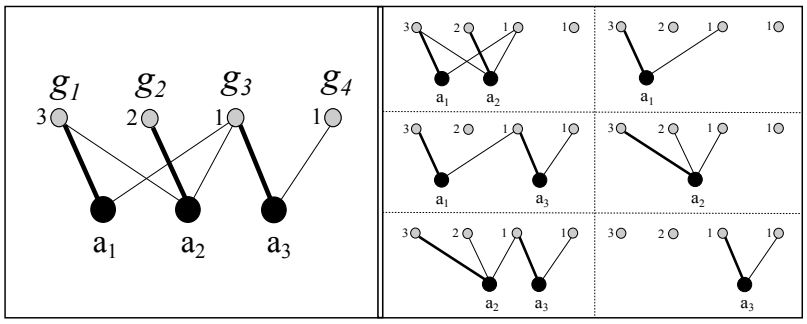


Figure 6.1: Allocation scenario $\mathcal{A}_0$ in Example 6.3.

chapter and in later chapters, when we refer to the function v, we will be making the assumption that it is superadditive.

In the next section we restrict the attention to a class of coalitional games called *allocation games*.

6.2 Allocation games

Among the various classes of coalitional games, we consider the class of *allocation games*, which is a setting to analyze fair division problems where monetary compensations are allowed and utilities are quasi-linear [Moulin, 1992]. Allocation games naturally arise in various application domains, ranging from house allocation to room assignment-rent division, to (cooperative) scheduling and task allocation, to protocols for wireless communication networks, and to queuing problems (see, e.g., [Moulin, 1992; Maniquet, 2003; Mishra and Rangarajan, 2007; **?**] and the references therein). In this thesis, we analyze the setting where, we are given an allocation scenario $\mathcal{A}$ comprising a set of goods and a set of agents, and each agent is to be assigned at most one good she is interested in. Each good g has a *value* $\texttt{val}(g) \in \mathbb{R}$ and the worth $v_{\mathcal{A}}(C)$ associated with any coalition $C \subseteq N$ is the maximum overall value that can be obtained over the assignments to agents in C only, also called allocations, hereinafter.

Example 6.3. *As an illustrative example, consider the allocation scenario* $\mathcal{A}_0$ *that is reported in Figure 6.1, by using an intuitive graphical notation. We have a set* $\{g_1, g_2, g_3, g_4\}$ *of goods that have to be allocated to three agents. Each edge connects an agent to a good she is interested in. Edges in bold identify an optimal allocation, i.e., a feasible allocation whose sum of values of the allocated goods is the maximum possible one. The value of this allocation is* $\texttt{val}(g_1)+\texttt{val}(g_2)+\texttt{val}(g_3) = 3+2+1 = 6$.

For each $C \subset \{1, 2, 3\}$ *with* $C \neq \emptyset$*, an optimal allocation restricted to the agents in* C *is also reported. Then, the associated coalitional game is* $\mathcal{G}_{\mathcal{A}_0} = \langle \{1, 2, 3\}, v_{\mathcal{A}_0} \rangle$*, where* $v_{\mathcal{A}_0}(\{1, 2, 3\}) = 6$*,* $v_{\mathcal{A}_0}(\{1, 2\}) = 5$*,* $v_{\mathcal{A}_0}(\{1, 3\}) = v_{\mathcal{A}_0}(\{2, 3\}) = 4$*,* $v_{\mathcal{A}_0}(\{1\}) = v_{\mathcal{A}_0}(\{2\}) = 3$*, and* $v_{\mathcal{A}_0}(\{3\}) = 1$. ◁

Allocation Scenario. Assume that a set $\mathbb{G}$ of goods have to be allocated to a set $N = \{1, ..., n\}$ of agents. Each good $g \in \mathbb{G}$ is associated with a real value $\texttt{val}(g) \in \mathbb{R}$, and each agent $i \in N$ can receive at most one good taken from her set of interest $\Omega(i) \subseteq \mathbb{G}$. The tuple $\mathcal{A} = \langle N, \mathbb{G}, \Omega, \texttt{val} \rangle$, with $\Omega : N \to 2^{\mathbb{G}}$ and $\texttt{val} : \mathbb{G} \to \mathbb{R}$, is an *allocation scenario*. Moreover, goods are indivisible and unshareable. Hence, an *allocation* for $\mathcal{A}$ is a function $\pi : N \to \mathbb{G} \cup \{\emptyset\}$ such that: (1) for each agent $i \in N$, $\pi(i) \neq \emptyset$ implies $\pi(i) \in \Omega(i)$; and (2) for each pair $i, i' \in N$ with $i \neq i'$, $\pi(i) \cap \pi(i') = \emptyset$ holds. We denote by $\texttt{img}(\pi)$ the set of all goods in the image of π, i.e., $\texttt{img}(\pi) = \{\pi(i) \mid i \in N \wedge \pi(i) \neq \emptyset\}$.

By slightly abusing of notation, if $S \subseteq \mathbb{G}$ is a set of goods, then $\texttt{val}(S)$ denotes the sum of their values. Moreover, if π is an allocation, then $\texttt{val}(\pi)$ denotes the value of $\texttt{img}(\pi)$. An allocation π is *optimal* (w.r.t. $\mathcal{A}$) if $\texttt{val}(\pi) \geq \texttt{val}(\pi')$ holds, for each allocation π'. The value associated with any optimal allocation w.r.t. $\mathcal{A}$ is hereinafter denoted as $\texttt{opt}(\mathcal{A})$.

Let $\mathcal{A} = \langle N, \mathbb{G}, \Omega, \texttt{val} \rangle$ be an allocation scenario and let $C \subseteq N$ be a set of agents.

The *restriction of* $\mathcal{A}$ *to* C is the *sub*-scenario $\mathcal{A}[C] = \langle C, \mathbb{G}, \Omega_C, \texttt{val}\rangle$ where Ω_C is the restriction of Ω over C. The *allocation game* induced by $\mathcal{A}$ is the tuple $\mathcal{G}_\mathcal{A} = \langle N, v_\mathcal{A}\rangle$, where $v_\mathcal{A} : 2^N \to \mathbb{R}$ is such that $v_\mathcal{A}(C) = \texttt{opt}(\mathcal{A}[C])$, for each $C \subseteq N$. The value of an empty set of goods is 0. Then, the definition trivializes for $C = \emptyset$, with $v_\mathcal{A}(\emptyset) = 0$. Moreover, note that $v_\mathcal{A}(C) \geq 0$ holds, for each $C \subseteq N$, since the allocation where no agent receives some good is a feasible one.

We are now ready to analyze the most important solution concepts used in coalitional game theory. Among all the solution concepts we will focus on the Shapley value, concluding the chapter with an application of this solution concept to a real allocation problem arising in a research assessment program in Italy described by [Greco and Scarcello, 2013b].

6.3 Solution concepts

As we said, a fundamental problem for a coalitional game $\mathcal{G} = \langle N, v\rangle$ is to single out the most desirable outcomes, usually called *solution concepts*, in terms of appropriate notions of worth distributions, i.e., of payoff vectors of the form $(x_1, ..., x_{|N|}) \in \mathbb{R}^{|N|}$ where $x_i + \cdots + x_{|N|}$ equals the worth associated with the whole set N of agents. A payoff vector x is called an *imputation* if it is *individual rational* i.e., for all players $i \in N$ we have $x_i \geq v(i)$, and *efficient* i.e., $\sum_{i\in N} x_i = v(N)$. Efficiency in this context means that the whole available worth is distributed while individual rationality indicates that if $x_i \geq v(i)$ the player i opts for participating since he would never do better going on his own. In the previous sections, we have seen that in superadditive games, we can always expect the grand coalition to form and thus the real issue is just that of the division of the whole available worth among the players. Many solution concepts have been considered in the coalitional game

theory and thoroughly studied through the years, and among all, the most known and widely-accepted are the *core*, *kernel*, *bargaining set*, *nucleolus* and the *Shapley value*. Below, we recall the notions of *core*, *kernel*, *bargaining set*, and *nucleolus* and we briefly discuss each of them.

6.3.1 The Core

Throughout the chapter we have emphasized the concept of stability several times. Stability leads to the question of whether the agents would be always willing to form the grand coalition given any payment profile. The answer to this question is given by the concept of *core* [Edgeworth, 1881; Gillies, 1959]. The *core* of a coalitional game is the set of all imputations x to which there not exists an objection. Formally, it is defined as follows.

Definition 6.4. Let $\mathcal{G} = \langle N, v \rangle$ be a coalitional game, and let x be an imputation. An imputation x is in the core of $\mathcal{G}$, if and only if

$$\forall S \subseteq N, \sum_{i \in S} x_i \geq v(S)$$

That is, the sum of the payoffs given to any group of agents must be always greater or equal to the sum of the payoffs that these agents are able to obtain independently. Consequently, we say that an imputation x in the core is "stable" because there is no coalition whose members will receive a higher payoff than in x by leaving the grand-coalition.

Example 6.5. As an example, consider let $\mathcal{G} = \langle N, v \rangle$ be a coalitional game, with the caracheristic function v given by

$$v(\{a\}) = 0 \quad v(\{a,b\}) = 20$$
$$v(\{b\}) = 0 \quad v(\{a,c\}) = 30 \quad v(\{a,b,c\}) = 45$$
$$v(\{c\}) = 0 \quad v(\{b,b\}) = 40$$

The set of imputations for this game are the triples (x_a, x_b, x_c) such that $x_a + x_b + x_c = 45$ and $x_a \geq 0, x_b \geq 0, x_c \geq 0$. For instance, if we consider the imputation x such that: $x_a = 5, x_b = 15, x_c = 25$ it is easily seen that x is in the core of $\mathcal{G}$.

Given that the core provides a concept of stability, it is a very attractive solution concept. However, there are games for which the core is empty.

Example 6.6. Let $\mathcal{G} = \langle N, v\rangle$ be a coalitional game, with $N = \{a, b, c\}, v(\{a\}) = v(\{b\}) = v(\{c\}) = 0, v(\{a, b\}) = 20, v(\{a, c\}) = 30, v(\{b, c\}) = 40, and v(\{a, b, c\}) = 42$. Consider the imputation x such that: $x_a = 4, x_b = 14$, and $x_c = 24$. Since $v(\{b, c\}) = 40 > 38 = x(\{b, c\})$ such imputation x is not in the core of $\mathcal{G}$. In particular, we can show that for this game the core is empty. In fact, observe that $v(\{a, b, c\}) = 42$ and consider the coalitions $S_1 = \{a, b\}, S_2 = \{a, c\}$ and $S_3 = \{b, c\}$. Recall from the definition that an imputation x must satisfy the following inequalities:

$$x_a + x_b \geq 20$$
$$x_a + x_c \geq 30$$
$$x_b + x_c \geq 40$$

It follows that $(x_a + x_b) + (x_a + x_c) + (x_b + x_c) \geq 20 + 30 + 40$, that is $x_a + x_b + x_c \geq 45$. Thus, the grand coalition should get a payoff of 45 instead of 42 in order to satisfy the agents. Consequently the grand coalition will never form for this game.

ϵ-core. While the core can be empty, sometimes we may also be happy with a set of imputations that are *approximately stable*. That is, we relax the notion of core and this leads to the definition of ϵ-core.

Definition 6.7. Let $\mathcal{G} = \langle N, v \rangle$ be a coalitional game, and let x be an imputation. An imputation x is in the ϵ-core of $\mathcal{G}$, if and only if

$$\forall S \subseteq N, \sum_{i \in S} x_i \geq v(S) - \epsilon$$

.

Example 6.8. Let $\mathcal{G} = \langle N, v \rangle$ and $N = \{a, b, c\}$. Let $v(S) = 1$ if $|S| > 1, v(S) = 0$ otherwise. Then it is easy to see that $(1/3, 1/3, 1/3)$ is an imputation and that is in $1/3$-core. Therefore $1/3-$core is non-empty while ϵ-core is empty for any $\epsilon < 1/3$.

The interpretation of the ϵ-core is that for any coalition there is at most an ϵ cost to deviate from the grand coalition. As a result, if the deficit $v(S) - x(S)$ is less than ϵ, the imputation x is still stable. Of course, we are interested in outcomes that minimize the ϵ.

Least core. From the last example we note that even the ϵ-core may be empty for some ϵ. However, there always exists some ϵ that is large enough to ensure that the ϵ-core is non-empty. Then, the natural problem is to find the smallest ϵ for which the ϵ-core is non-empty. This leads to the definition of the concept of the *least core*.

Definition 6.9. Let $\mathcal{G} = \langle N, v \rangle$ be a coalitional game, and let x be an imputation. An imputation x is in the ϵ-core of $\mathcal{G}$, if and only if x is the solution the following linear program.

$$\begin{aligned} &\text{minimize} && \epsilon \\ &\text{subject to} && \textstyle\sum_{i \in S} x_i \geq v(S) - \epsilon \quad \forall S \subseteq N \end{aligned}$$

Intuitively, the *least core* consists of all payoffs that minimize the incentive of all coalitions to deviate. Thus, it can be considered a refinement of the core. Observe also that if ϵ is large enough, the *least core* is always non-empty but it may contain

more than one imputation. Therefore, we would like to identify the most stable point in the least core. We pursue this objective, with the definition of the *nucleolus*.

6.3.2 The Nucleoulus

Given an imputation x, the least core minimize the maximum inequity $\epsilon(S, x) = v(S) - x(S)$ (also called deficit or excess). This quantity is a measure of the dissatisfaction of S with the proposed imputation x. The nucleolus of a game $\mathcal{G}$ not only minimize the worst inequity but given this, it also minimizes the second largest inequity. Intuitively, we look first for the coalition with the maximum degree of dissatisfaction then we try to make it smaller if possible by adjusting x. Once the largest dissatisfaction has been made as small as possible, then we address the second largest dissatisfaction and we try to adjust x to lower it as much as possible. This is formalized as follows.

Definition 6.10. Let $\mathcal{G} = \langle N, v \rangle$ be a coalitional game, and let x be an imputation. An imputation x is in the nucleoulus of $\mathcal{G}$, if and only if x is the solution to series of optimization programs $O_1, O_2, ..., O_{|N|}$ defined as follows.

$$(O_1) = \begin{cases} \text{minimize} & \epsilon \\ \text{subject to} & \sum_{i \in S} x_i \geq v(S) - \epsilon \quad \forall S \subseteq N \end{cases}$$

$$(O_i) = \begin{cases} \text{minimize} & \epsilon \\ \text{subject to} & \sum_{i \in S} x_i \geq v(S) - \epsilon \qquad\qquad \forall S \subseteq S_1 \\ & \vdots \\ & \sum_{i \in S} x_i \geq v(S) - \epsilon_{i-1} \quad \forall S \subseteq S_{i-1}\ S_{i-2} \\ & \sum_{i \in S} x_i \geq v(S) - \epsilon \qquad \forall S \subseteq 2^N\ S_{i-1} \end{cases}$$

Theorem 6.11. *For any coalitional game $\mathcal{G}(N, v)$, the nucleolus of $\mathcal{G}$ always exists and is unique.*

6.3.3 The Kernel

The kernel is a solution concept introduced by [Davis and Maschler]. Let us start by recalling its formal definition. For any pair of players i and j of a coalitional game $\mathcal{G} = \langle N, v\rangle$, we denote by $I_{i,j}$ the set of all coalitions containing player i but not player j. Recall that $\epsilon(S, x) = v(S) - x(S)$. Define the surplus $s_{i,j}(x)$ of player i against player j at an imputation x as the value $si, j(x) = \max_{S \in I_{i,j}} e(S, x) = \max_{S \in I_{i,j}}(v(S) - x(S))$.

Definition 6.12. Let $\mathcal{G} = \langle N, v\rangle$ be a coalitional game. The *kernel* of $\mathcal{G}$ is the set of all imputations x such that $s_{i,j}(x) > s_{j,i}(x) \Rightarrow x_j = v(j), \forall i, j \in N, i \neq j$.

Intuitively, the surplus of player i against j at x is the highest payoff that player i can gain (or the minimal amount i can lose, if it is a negative value) without the cooperation of j, by assuming to form coalitions with other players that are satisfied at x; thus, $s_{i,j}(x)$ is the weight of a possible threat of i against j. In particular, player i has more bargaining power than j at x if $s_{i,j}(x) > sj, i(x)$; however, player j is immune to such threat whenever $x_j = v(j)$, since in this case j can obtain $v(j)$ even by operating on her own. We say that player i outweighs player j at x if $s_{i,j}(x) > s_{j,i}(x)$ and $x_j > v(j)$. Therefore, the kernel is the set of all imputations where no player outweighs another one.

Example 6.13. Let $\mathcal{G} = \langle N, v\rangle$ be a coalitional game and let $N = \{a, b, c\}, v(\{a\}) = v(\{b\}) = v(\{c\}) = 0, v(\{a, b\}) = 20, v(\{a, c\}) = 30, v(\{b, c\}) = 40$, and $v(\{a, b, c\}) = 42$. It is easily verified that the imputation x such that $x_a = 4, x_b = 14$, and $x_c = 24$ is in the kernel of $\mathcal{G}$. Indeed, we note first that every player in N receives in x a payoff

strictly greater than what she is able to obtain by acting on her own. For this reason, in order for x to belong to the kernel of $\mathcal{G}$ it must be the case that $s_{i,j}(x) < s_{j,i}(x)$, for all distinct players i and j. By the definition of the worth function, the maximum excess that a coalition S including i and excluding j can achieve is obtained by the coalition $S \in I_{i,j}$ such that $|S| = 2$. By this, $s_{i,j}(x) = s_{j,i}(x) = 2$ for all pairs of different players i, j. Thus, the imputation x is in the kernel of $\mathcal{G}$.

6.3.4 The Bargaining set

The concept of bargaining set was defined by [Aumann and Maschler, 1964]. As usual, we recall its formal definition and then we illustrate the concept with an example.

Definition 6.14. Let $\mathcal{G} = \langle N, v\rangle$ be a coaliitonal game , and x be an imputation. Let $S \subseteq N$ be a coalition, and $y$be an S-feasible payoff vector (i.e., $y(S) = v(S)$). The pair (y, S) is an objection of player i against player j to x if $i \in S, j \in S$, and $y_k > x_k \forall k \in S$. A *counterobjection* to the objection (y, S) of i against j to x is a pair (z, T) where $j \in T, i \in T$, and z is a T-feasible payoff vector such that $z_k \geq x_k$ $\forall k \in T \setminus S$ and $z_k \geq y_k$ $\forall k \in T \cap S$. If there does not exist any *counterobjection* to (y, S), we say that (y, S) is a justified objection.

The bargaining set of a calitional game $\mathcal{G}$ is the set of all imputations x to which there is no justified objection.

Example 6.15. Let $\mathcal{G} = \langle N, v\rangle$ be a coalitional game with $N = \{a, b, c\}, v(\{a\}) = v(\{b\}) = v(\{c\}) = 0, v(\{a, b\}) = 20, v(\{a, c\}) = 30, v(\{b, c\}) = 40$, and $v(\{a, b, c\}) = 42$. Consider the imputation x such that $x_a = 8, x_b = 10$, and $x_c = 24$. An objection of player c against player a to x is $((12, 28), \{b, c\})$. Player a can counterobject to this objection using $((8, 12), \{a, b\})$. Another objection of player c against player a

to x is $((14, 26), b, c)$. In this case, player a cannot counterobject. The reason is that coalition $\{a, b\}$ receives a payoff 20 and this is not sufficient for player a to counterobject since she needs at least 8 for herself and at least 14 for player b, in order to respond to the proposal of player c. Therefore the imputation x does not belong to $B(G)$. The intuitive reason is that player a receives too much, according to this profile. Consider now the imputation x' such that $x'_a = 4, x_b = 14$, and $x_c = 24$. We focus on the objections of player a against player c. We note that, in order to object, player a has to form the coalition $S = \{a, b\}$. The excess $e(S, x)$ of S at x is 2, hence players a and b have the possibility to distribute among themselves a payoff of 2 to make the objection. But player c can always counterobject to player a because she can form the coalition $T = \{b, c\}$ whose excess at x is 2 and hence she can always match the proposal made to player b by player a in order to object. A similar argumentation holds for every objection of every player against any other. Thus x belongs to the bargaining set of $\mathcal{G}$. $\lhd$

Observe that the core is the set of all imputations that do not admit an objection. Hence *core* $\subseteq$ *bargaining set*. To conclude our analsys on the most important solution concepts in game theory, we have to consider the *Shapley value* (see, e.g., [Shapley, 1953]) to which we will dedicate the next section.

6.4 The Shapley value

The core defined a way to form coalition that are stable. However it does not make fairness considerations. In fact, outcomes in the core may be really unfair. For instance consider the following example.

Example 6.16. Let $\mathcal{G} = \langle N, v \rangle$ be a coalitional game, with $N = a, b, v(a) = v(b) = 5, v(a, b) = 20$. Consider the imputation x such that: $x_a = 20, x_b = 5$. We can easily

check that x is in in the core of $\mathcal{G}$, in fact both players cannot benefit by deviating from the grand coalition. However, the outcome is really unfair for b since she is perfectly symmetric with a. $\lhd$

Consequently, we answer the question posed in Section 6.1 negatively. Therefore, we would like to find a mechanism that allow us to divide the total payoff among the members of the grand coalition in a **fair** way. It turns out that the answer to this problem is given by *The Shapley value* [Shapley, 1953]. Crudely, the Shapley value of each player i is given by the average marginal contribution of i over all possible coalitions (s)he may participate in. Thus, while the *core* formalize the concept that the coalition has to be stable, the Shapley value provides a unique way to distribute coalitional payoff among agents in such a way that satisfy various fairness criteria. Let us start by axiomatizing the concept of fairness in our context.

The Shapley Axioms for the characteristic function v.

1. $\sum_{i \in N} v_i(G) = v(N)$ (efficiency);
2. For any v, if i and j are such that $v(S \cup i) = v(S \cup j)$ for every coalition S not containing i and j, then $\phi_i(\mathcal{G}) = \phi_j(\mathcal{G})$ (simmetry);
3. For any v, if i is a dummy player then $\phi_i(\mathcal{G}) = v(i)$ (dummy player);
4. If u and v are characteristic functions, then $\phi(u+v) = \phi(u) + \phi(v)$ (additivity);
5. If v is *supermodular*, the Shapley value belongs to the core of the game (stability);
6. If $G' = (N, v')$ is a game such that $v'(C) \geq v(C), \forall C \subseteq N$, then $\phi_i(G') \geq \phi_i(G), \forall i \in N$ (monotonicity).

Theorem 6.17. *Given a coalitional game $\mathcal{G} = \langle N, v \rangle$, there is a unique imputation that satisfies the Shapley axioms and it is called the Shapley value.*

As we already stated, the Shapley value is based on the value that each player is able to add to the possible coalitions , i.e., on its marginal contribution. Suppose we form the grand coalition by entering the players into this coalition one at a time. As each player enters the coalition, he receives the amount by which his entry increases the value of the coalition he enters. The amount a player receives by this scheme depends on the order in which the players are entered. The Shapley value is just the average payoff to the players if the players are entered in completely random order.

Definition 6.18. It turns out that the payoff associated with each agent $i \in N$ is given by

$$\phi_i(\mathcal{G}) = \sum_{C \subseteq N \setminus \{i\}} \frac{|C|!(n-|C|-1)!}{n!} \Big(v(C \cup \{i\}) - v(C) \Big), \text{ where } v(C \cup \{i\}) - v(C)$$

is the *marginal contribution* of i to the coalition $C \cup \{i\}$.

Example 6.19. *Consider again the allocation game $\mathcal{G}_{\mathcal{A}_0}$ introduced in Example 6.3. For agent 1, we have:*

$$\begin{aligned} \phi_1(\mathcal{G}_{\mathcal{A}_0}) = \quad & \tfrac{2!0!}{3!} \Big(v_{\mathcal{A}_0}(\{2,3\} \cup \{1\}) - v_{\mathcal{A}_0}(\{2,3\}) \Big) + \\ & \tfrac{1!1!}{3!} \Big(v_{\mathcal{A}_0}(\{2\} \cup \{1\}) - v_{\mathcal{A}_0}(\{2\}) \Big) + \\ & \tfrac{1!1!}{3!} \Big(v_{\mathcal{A}_0}(\{3\} \cup \{1\}) - v_{\mathcal{A}_0}(\{3\}) \Big) + \\ & \tfrac{0!2!}{3!} \Big(v_{\mathcal{A}_0}(\emptyset \cup \{1\}) - v_{\mathcal{A}_0}(\emptyset) \Big) = \tfrac{15}{6}. \end{aligned}$$

Similarly, we can derive $\phi_2(\mathcal{G}_{\mathcal{A}_0}) = \frac{15}{6}$ and $\phi_3(\mathcal{G}_{\mathcal{A}_0}) = \frac{6}{6}$. ◁

6.4.1 The Banzhaf value

Another solution concept which is similar in spirit to the Shapley value is the *Banzhaf value* (or index) (see, e.g., [Banzhaf, 1965]) for which the payoff of i is

$$\beta_i(\mathcal{G}) = \tfrac{1}{2^{n-1}} \sum_{C \subseteq N \setminus \{i\}} \Big(v(C \cup \{i\}) - v(C) \Big).$$

In the next chapter we will analyze in detail the relationship between these two solution concepts.

6.4.2 Back to the allocation games

In the context of allocations games (and when monetary transfers are possible), the prototypical solution concepts considered in the literature are the *Shapley value* and the *Banzhaf value*, since they lead to outcomes enjoying desirable properties such as equal treatment of equals, envy-freeness, and a stronger property called individual-optimality guaranteeing that, for every agent, her/his utility is the maximum possible one over any alternative optimal allocation [**?**]. However, it is well known that, in general, computing such values is intractable, formally #P-complete. This is a serious obstruction to their applicability in allocation scenarios involving many agents, and it motivates the design of approximation algorithms and the identification of subclasses of practical interest where exact computation can be carried out efficiently. In Chapter 7 we focus precisely on the latter approach. But before we do so, let us recall some basic properties about allocations games.

Basic Properties. We discuss below some properties pointed out by [**?**] and [Moulin, 1992]. Let $\mathcal{A} = \langle N, \mathbb{G}, \Omega, \mathtt{val} \rangle$ be an allocation scenario, and let C and C' be two coalitions with $C' \subseteq C \subseteq N$. Then,

(allocation) monotonicity: $v_{\mathcal{A}}(C) \geq v_{\mathcal{A}}(C')$. Moreover, if π is an optimal allocation for $\mathcal{A}[C]$, then there is an optimal allocation π' for $\mathcal{A}[C']$ such that $\mathtt{img}(\pi') \subseteq \mathtt{img}(\pi)$;

submodularity: $v_{\mathcal{A}}(C \cup \{i\})$-$v_{\mathcal{A}}(C) \leq v_{\mathcal{A}}(C' \cup \{i\})$-$v_{\mathcal{A}}(C')$, for each $i \in N \setminus C$.

Furthermore, it has be shown that, for these games, the Shapley value grants to every coalition at least its marginal contribution to the worth of the grand-coalition. Indeed, the Shapley value of $\mathcal{G}_{\mathcal{A}}$ is in the *core* of the game $\bar{\mathcal{G}}_{\mathcal{A}} = \langle N, \bar{v}_{\mathcal{A}} \rangle$, where $\bar{v}_{\mathcal{A}}(C) = v_{\mathcal{A}}(N) - v_{\mathcal{A}}(N \setminus C)$, for each coalition $C \subseteq N$.[1] More formally, the following

[1] It even holds that $\phi_i(\mathcal{G}_{\mathcal{A}}) = \phi_i(\bar{\mathcal{G}}_{\mathcal{A}})$, for each $i \in N$.

holds.

Theorem 6.20. *For each scenario* $\mathcal{A} = \langle N, \mathbb{G}, \Omega, \mathtt{val} \rangle$*, we have that:* $\sum_{i \in N} \phi_i(\mathcal{G}_\mathcal{A}) = v_\mathcal{A}(N)$ *and* $\sum_{i \in C} \phi_i(\mathcal{G}_\mathcal{A}) \geq v_\mathcal{A}(N) - v_\mathcal{A}(N \setminus C)$*, for each* $C \subseteq N$*.*

Example 6.21. *Note that* $\phi_1(\mathcal{G}_{\mathcal{A}_0}) + \phi_2(\mathcal{G}_{\mathcal{A}_0}) + \phi_3(\mathcal{G}_{\mathcal{A}_0}) = \frac{36}{6} = v_{\mathcal{A}_0}(\{1,2,3\})$*. Moreover, for each* $C \subseteq N$*, observe that* $\sum_{i \in C} \phi_i(\mathcal{G}_{\mathcal{A}_0}) \geq 6 - v_{\mathcal{A}_0}(N \setminus C)$*. For instance,* $\phi_1(\mathcal{G}_{\mathcal{A}_0}) + \phi_3(\mathcal{G}_{\mathcal{A}_0}) = \frac{21}{6} \geq 6 - v_{\mathcal{A}_0}(\{2\}) = 6 - \frac{15}{6}$. ◁

6.5 A motivating example: The Italian Research Assessment Program (VQR)

The Italian Research Assessment Program (VQR in short) is aimed at evaluating the results of the Italian scientific research in the period 2011-2014 carried out by public universities, private universities, and other research institutes. The outcome of the evaluation will be used by the National Agency for the Evaluation of Universities and Research Institutes (ANVUR) to proportionally distribute the funds allocated by the Ministry to universities and research institutes in order to support their research activities in the next years (until the subsequent evaluation process). In addition to the institutions, the VQR will also evaluate departments of universities and similar internal structures of other research institutes. At this point, it is easy to see that every research structure R is in competition with all other research structures, as every structure R is then interested in selecting and submitting its *best* research products (e.g., publications). The program is articulated as follows:

1. Authors affiliated to R are asked to self-evaluate their products, according to some evaluation criteria defined by groups of experts chosen by ANVUR. We suppose that every researcher is able to associate to each product a quality score.

2. Based on the self-evaluations being collected, R is in charge of selecting the best products to submit to ANVUR (at most) products for each one of its authors. Note that a product cannot be allocated to two researchers.

3. ANVUR carries out its assessment on the quality of products submitted and the quality profile (VQR score) of R is determined by the sum of the "true" scores associated with the products (i.e., those resulting by ANVUR actual evaluation). Eventually, R will receive funds in the next years proportionally to this score. Moreover, by using such products scores, ANVUR evaluates all substructures, too (e.g., all departments, if R is a University).

It is then natural for the structure R to exploit the VQR scores of its substructures to divide funds among them. In particular, the outcomes could be in principle used to evaluate single researchers, too. In this chapter we will show that finding a good (fair) *division* rule for scores/funds distribution is not an easy task. Moreover, we pinpoint that, without and agreement on a fair division rule, the self-evaluation performed at point (1) is unfeasible, and the research structure may miss the optimal possible submission to ANVUR, hence losing funds. Before we point out how we can apply techniques from game theory to this case study, we present the simple division rules that are believed to be applied by the ANVUR and we show that they are unfair for substructures.

6.5.1 Division rules

Let R be a research structure, and let $\mathcal{R}$ be the set of researchers affiliated with R. For each researcher $r \in R$, let $products(r, R)$ (or just $products(r)$, if R is understood from the context) be the set of the research products of r in the given period 20112014. An allocation for a set of researchers $S \subseteq R$ is a function ψ mapping each

researcher $r \in S$ to a set of publications $\psi(r) \subseteq products(r)$ with $|\psi(r)| \leq 3$ and with $\psi(r) \cap \psi(r') = \emptyset$, for each $r' \in \mathcal{S} \setminus \{r\}$. An allocation for R is an allocation for all researchers $\mathcal{R}$, while an allocation for a substructure S of R is an allocation for the researchers affiliated to S. In the VQR program, every research structure R has to submit to ANVUR for its evaluation a set of products $\mathcal{P}_\psi$ such that $\mathcal{P}_\psi = \bigcup_{r \in \mathcal{R}} \psi(r)$, for some allocation ψ for all researchers $\mathcal{R}$ affiliated to R. Then, for each $p \in \mathcal{P}_\psi$, ANVUR calculates a quality score $score_{\text{VQR}}(p)$, so that R will receive funds proportionally to its overall score $score_{\text{VQR}}(\psi) = \sum_{p \in \mathcal{P}_\psi} score_{\text{VQR}}(p)$.

While the first aim of the VQR program is to evaluate the various Italian research structures, it is known that the obtained information will be used to evaluate substructures, too (e.g., University departments). Thus, following the same principle of binding funds to VQR scores used for the main structure, it is natural to exploit such scores for money distribution inside every research structure. It is therefore of utmost importance the way VQR assigns scores to substructures. Nevertheless, as already mentioned, up to date there is no official information about such an algorithm. We argue that the score of any structure R should fairly be distributed over its substructures (and possibly over individuals), in such a way to reflect their actual contribution to the result achieved by the structure R. Formally, we need a suitable division rule.

Definition 6.22. A division rule γ for R is a real-valued function that, given a researcher $r \in \mathcal{R}$ and an allocation ψ for R, returns its score $\gamma_\psi(r, R) \geq 0$ with respect to the allocation ψ. By slightly abusing notation, for any substructure $\mathcal{S} \subseteq \mathcal{R}$, we denote by $\gamma_\psi(\mathcal{S}, R)$ the value $\sum_{r \in \mathcal{S}} \gamma_\psi(r, R)$. Whenever R is understood from the context, we just write $\gamma_\psi(r)$ and $\gamma_\psi(\mathcal{S})$, in place of $\gamma_\psi(r, R)$ and $\gamma_\psi(S, R)$, respectively.

Hereafter, we assume that $\mathcal{S}_1, \ldots, \mathcal{S}_n$ are the substructures of R. They exhaus-

tively cover the researchers of R, i.e.,

$$\begin{cases} \bigcup_{i=1}^{n} \mathcal{S}_i = \mathcal{R} \\ \mathcal{S}_i \cap \mathcal{S}_j = \emptyset, \forall i \neq j \end{cases}$$

Note that such a division rule may naturally be used for evaluating individuals, besides substructures. In fact, we shall discuss all fairness properties with reference to individual researchers, because all issues about individuals immediately extend to the substructures they belong to.

6.5.2 Desiderata for division rules

In the absence of an official division rule, most researchers believe that the score of any substructure S will be based on the naive *proj* rule where, for any researcher r, $proj_\psi(r)$ is the sum of the VQR scores of the products allocated to r in ψ, i.e.,

$$proj_\psi(r) = \sum_{p \in \psi(r)} score_{\text{VQR}}(p).$$

This rule satisfies a very basic requirement for every division rule, which we state below.

Definition 6.23 (Budget-balance)**.** A division rule γ must precisely distribute the VQR score of R over all its members, i.e., $\sum_{r \in \mathcal{R}} \gamma_\psi(r) = score_{\text{VQR}}(R)$. Clearly, because the substructures of R define a partition of its researchers, this implies that γ completely distribute the VQR score of R over all its substructures and does not distribute more than that, i.e., $\sum_{i=1}^{n} \gamma_\psi(\mathcal{S}_i) = score_{\text{VQR}}(R)$.

However, this rule is hardly perceivable as a fair one. Indeed, *proj* might lead to scenarios where some researcher r (and in turn her/his substructure) has reasonable arguments against her/his structure because of possible alternative allocations where

r would get higher scores. Thus, *proj* fails to satisfy a basic and intuitive requirement of fairness, which we formalize as follows.

Definition 6.24 (Fairness)**.** Assume that ψ and $\bar{\psi}$ are two allocations of the same research products, i.e., $\mathcal{P}_\psi = \mathcal{P}_{\bar{\psi}}$. A division rule γ must be indifferent w.r.t. the optimal allocation being selected, i.e., for each researcher $r \in \mathcal{R}, \gamma_\psi(r) = \gamma_{\bar{\psi}}(r)$. In fact, this implies that, for each substructure $\mathcal{S}_i, \gamma_\psi(\mathcal{S}_i) = \gamma_{\bar{\psi}}(\mathcal{S}_i)$ also holds.

The above criterion is definitely desirable. However, a closer look reveals that it is not enough, as there is a trivial way to circumvent this kind of fairness: just consider the rule *trivial* assigning the overall VQR score to one fixed researcher, say r_1 (independently of its actual contribution), i.e.,

$$trivial_\psi(r) = \begin{cases} \sum_{p \in \mathcal{P}_\psi} score_{\mathrm{VQR}}(p), & \text{if } r = r_1 \\ 0 & \text{if } r \in \mathcal{R} \setminus \{r_1\} \end{cases}$$

A problem with this rule is that it is not symmetric, in that a researcher r_2 that has co-authored precisely the same set of products as r_1 would be treated differently, just because of her name. Avoiding these cases and guaranteing an equal treatment of the equals is another very basic requirement, formalized as follows.

Let $\pi \colon \mathcal{R} \mapsto \mathcal{R}$ be a permutation of the researchers in $\mathcal{R}$. Let R_π be the research structure over the researchers in $\mathcal{R}$ where, for each $r \in \mathcal{R}$, $products(r, R_\pi) = products(\pi(r), R)$. Moreover, if ψ is an allocation for R, then let ψ_π be the allocation such that, for each $r \in \mathcal{R}$, $\psi_\pi(r, R_\psi) = \psi(\pi(r), R)$. Thus, R_π and ψ_π are derived by applying the permutation π, whose role is just to rename the researchers in $\mathcal{R}$. With these notions in place, we can now state the following property, which is in fact not enjoyed by *trivial*.

Definition 6.25 (Impartiality)**.** Let π be an arbitrary permutation over $\mathcal{R}$. A

division rule γ must be such that, for each $r \in \mathcal{R}$ and each allocation ψ, $\gamma_{\psi_\pi}(r, R_\pi) = \gamma_\psi(\pi(r), R)$ holds.

Yet again this is not enough. To see that, consider the very impartial rule $uniform$, where the overall ANVUR score is distributed uniformly over the various researchers, i.e.,

$$uniform_\psi(r) = \frac{score_{\text{VQR}}(R)}{|\mathcal{R}|}.$$

Clearly, uniform is unsatisfying because it does not capture our intuition that a division rule should reflect the actual contribution of each researcher to the overall evaluation of the structure. In the rest of this section, we will elaborate on this issue by using the notion of marginal contribution, which fits well our intuition of actual contributions of individual or groups to the performance of a given structure.

6.5.3 Marginal contribution

Let R be a research structure and assume that ψ is the allocation selected by the structure R, so that the set of products $\mathcal{P}_\psi$ have been submitted and evaluated by ANVUR. Let $\mathcal{S} \subseteq \mathcal{R}$ be any set of researchers. An allocation $\psi_\mathcal{S}$ for $\mathcal{S}$ is $\psi - legal$ if $\psi_\mathcal{S}(r) \subseteq \mathcal{P}_\psi$, for each $r \in \mathcal{S}$. That is, any legal allocation only considers for the assignment the products already evaluated by ANVUR. The allocation $\psi_\mathcal{S}$ is $\psi - optimal$ if there is no $\psi - legal$ $\psi'_\mathcal{S}$ such that $\sum_{r\in\mathcal{S}} \sum_{p\in\psi'_\mathcal{S}(r)} score_{\text{VQR}}(p) > \sum_{r\in\mathcal{S}} \sum_{p\in\psi_\mathcal{S}(r)} score_{\text{VQR}}(p)$.

Given the above notions, we can equip S with the following score, which is meant to assess the overall VQR score that researchers in S would have achieved if the research structure had been constituted by them only (i.e., without caring about their co-authors outside S)

$$best_\psi(\mathcal{S}) = \sum_{r\in\mathcal{S}} \sum_{p\in\psi_\mathcal{S}(r)} score_{\text{VQR}}(p),$$

where $\psi_{\mathcal{S}}$ is any $\psi-optimal$ allocation.

Note that in the extreme case where $\mathcal{S} = \mathcal{R}$ we just obtain $best_\psi(\mathcal{R}) = score_{\text{VQR}}(p)$.. That is, when all researchers of R are considered, $best_\psi$ precisely coincides with the overall VQR score.

We can now formalize the notion of marginal contribution.

Definition 6.26. Let ψ be an allocation for R. Given two sets of researchers $\mathcal{S}_1, \mathcal{S}_2 \subseteq \mathcal{R}$ with $\mathcal{S}_1 \subseteq \mathcal{S}_2$, the marginal contribution of $\mathcal{S}_1$ to $\mathcal{S}_2$ (in R and ψ) is the value:

$$marg_\psi(\mathcal{S}_1, \mathcal{S}_2) = best_\psi(\mathcal{S}_2) - best_\psi(\mathcal{S}_2 \setminus \mathcal{S}_1).$$

Intuitively, the marginal contribution $marg_\psi(S_1, S_2)$ quantifies the loss of VQR score for the group of researchers $\mathcal{S}_2$ (e.g., a substructure) if the groups of researchers $\mathcal{S}_1$ were not part of it. In particular,$marg_\psi(\{r\}, \mathcal{R})$ measures the loss for the whole structure R, if the single researcher r were not part of it.

We are now ready to define the last fairness property, which takes care of the actual contribution of individuals and groups.

Definition 6.27 (marginality)**.** A division rule γ must be such that, for each group of researchers $\mathcal{S} \subseteq \mathcal{R}$ and each allocation ψ, $\gamma_\psi(\mathcal{S}) \geq marg_\psi(\mathcal{S}, \mathcal{R})$. Therefore, every group is granted at least its marginal contribution to the performance of the structure R.

In particular, the above property entails that groups without interactions with other researchers, e.g., departments without collaborations with other departments of the same university, get precisely the total scores of the products assigned to them according to ψ. We now describe a practical example with the aim to summarize and clarify the division rules and the issues discussed so far.

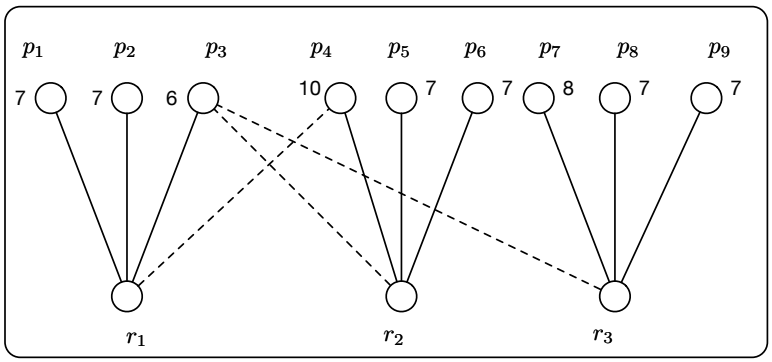


Figure 6.2: Running example

6.5.4 A simple scenario

Consider the simple scenario depicted in Figure 6.2 and let us exploit an intuitive graphical notation based on a weighted bipartite graph, whose vertices are the researchers and the products, and whose weights are the VQR scores of the products. Assume that there are just three researchers, r_1, r_2 and r_3, affiliated to R. Moreover, consider the allocation ψ such that $\psi(r_1) = \{p_1, p_2, p_3\}, \psi(r_2) = \{p_4, p_5, p_6\}$ and $\psi(r_3) = \{p_7, p_8, p_9\}$. Thus, we have that $\mathcal{P}_\psi = \{p_1, \ldots, p_9\}$ is the set of publications submitted for the evaluation. In particular, we also assume that $products(r_1) \cap \mathcal{P}_\psi = \{p_1, p_2, p_3, p_4\}$, $products(r_2) \cap \mathcal{P}_\psi = \{p_3, p_4, p_5, p_6\}$, and $products(r_3) \cap \mathcal{P}_\psi = \{p_3, p_6, p_7, p_8\}$. Thus, p_3 is co-authored by r_1, r_2 e r_3, while p_4 is co-authored by r_1 and r_2. Then, according to the VQR score assigned to each product (i.e., the weight of the corresponding vertex in Figure 6.2), we get that $products_{\mathrm{VQR}}(\psi) = 66$.

Suppose now that we want to use the rule *proj* to assign the scores. Recall that *proj* is defined as follows.

$$proj_\psi(r) = \sum_{p \in \psi(r)} score_{\mathrm{VQR}}(p).$$

Then, according to *proj* the researchers would get $proj_\psi(r_1) = 20, proj_\psi(r_2) = 24$

and $proj_\psi(r_3) = 22$. This is unfair for the researcher r_1 since even if she and r_2 have co-authored p_3 and p_4, they get a different payoff. Therefore, as we said, r_1 could complain with her structure if ψ is selected.

Now, suppose were using the rule *trivial*,

$$trivial_\psi(r) = \begin{cases} \sum_{p \in \mathcal{P}_\psi} score_{\text{VQR}}(p), & \text{if } r = r_1 \\ 0 & \text{if } r \in \mathcal{R} \setminus \{r_1\} \end{cases}$$

it is easy to check that $score(r_1) = 66$ and $score(r_i) = 0\ \forall r_i \in \mathcal{R} \setminus \{r_1\}$. Clearly *trivial* is unsatisfying.

Finally, if we consider

$$uniform_\psi(r) = \frac{score_{\text{VQR}}(R)}{|\mathcal{R}|}.$$

we have that each researcher gets the score $\frac{20+24+22}{3} = 22$, that is uniform, but it does not capture our intuition that a division rule should reflect the actual contribution of each researcher to the overall evaluation of the structure.

We can show that the concept of marginal contribution captures this intuition. We proceed as follows. Initially, we compute the best scores that all the possible groups of researches could get if they were the only one in the research structure.

- $best_\psi(\{r_1\}) = 24$, $best_\psi(\{r_2\}) = 24$ and $best_\psi(\{r_3\}) = 22$;
- $best_\psi(\{r_1, r_2\}) = 44$, $best_\psi(\{r_1, r_3\}) = 46$ and $best_\psi(\{r_2, r_3\}) = 46$;
- $best_\psi(\{r_1, r_2, r_3\}) = 66$.

Then, by recalling the formula of marginal contributions,

$$marg_\psi(\mathcal{S}_1, \mathcal{S}_2) = best_\psi(\mathcal{S}_2) - best_\psi(\mathcal{S}_2 \setminus \mathcal{S}_1).$$

we compute them, first with respect to the whole set $\mathcal{R}$ of researchers in R, and then with respect to all the possible pairs of researchers $\{r_i, r_j\}$:

- $marg_\psi(\{r_1\}, \mathcal{R}) = 66 - 46 = 20$;
- $marg_\psi(\{r_2\}, \mathcal{R}) = 66 - 46 = 20$;
- $marg_\psi(\{r_3\}, \mathcal{R}) = 66 - 44 = 22$;
- $marg_\psi(\{r_1,\, r_2\}, \mathcal{R}) = 66 - 22 = 44$;
- $marg_\psi(\{r_1,\, r_3\}, \mathcal{R}) = marg_\psi(\{r_2,\, r_3\}, \mathcal{R}) = 66 - 24 = 42$;
- $marg_\psi(\{r_1\}, \{r_1,\, r_2\}) = marg_\psi(\{r_2\}, \{r_1,\, r_2\}) = 44 - 24 = 20$;
- $marg_\psi(\{r_1\}, \{r_1,\, r_3\}) = marg_\psi(\{r_2\}, \{r_2,\, r_3\}) = 46 - 22 = 24$;
- $marg_\psi(\{r_3\}, \{r_1,\, r_3\}) = marg_\psi(\{r_3\}, \{r_2,\, r_3\}) = 46 - 24 = 22$.

From the computed marginal contributions, we can see that researchers r_1 and r_2 are completely interchangeable.

6.5.5 Using the Shapley value as a division rule

In this section we show that a division rule enjoying all the above fairness properties actually exists and it is the Shapley value. Indeed, it is natural to formalize the considered application as a coalitional game where the agents are the researchers, and the worth of any coalition is the best result that the researchers in that coalition may achieve if acting in isolation. Formally, for a research structure R and an allocation ψ, define the coalitional game $\mathcal{G}_\psi = (\mathcal{R}, best_\psi)$. Then, we propose the following division rule, which is easily seen to be precisely the Shapley value of $\mathcal{G}_\psi$:

$$\gamma^*_\psi(r) = \sum_{\mathcal{S} \subseteq \mathcal{R}} \frac{(|\mathcal{R}| - |\mathcal{S}|)!(|\mathcal{S}| - 1)!}{|\mathcal{R}|!} marg_\psi(r, \mathcal{S}), \quad \forall r \in \mathcal{R}.$$

This rule, satisfies all the nice fairness properties discussed so far as the following theorem states:

Theorem 6.28 (cf. [Greco and Scarcello, 2013b]). *Let ψ be any allocation for a given structure R. Then, γ^*_ψ is a division rule satisfying the budget-balance, the independence of the product allocation, the impartiality and the marginality fairness properties.*

6.5.6 Discussion

So far, we focused on a setting where an allocation is selected by the research structure R and we are in charge of specifying a division rule γ^*_ψ that enjoys all the fairness properties discussed earlier. However, fairness depends also on the preliminary selection of the products $\mathcal{P}_\psi$. Indeed we cannot guarantee fairness if the given allocation is unfair itself. In the first phase of the evaluation process, the structure R (i.e., the department) is in charge of evaluating the scientific production of its researchers. However, a detailed evaluation of the whole production is not an easy task, indeed the simplest solution for R is to ask its researchers to perform a self-evaluation of their products. Therefore, the goal of R will be achieved if authors correctly/truthfully self-evaluate their products. But what happens if this assumption is removed? It is natural at all to remove this assumption? While it is clear that the main goal for R is to maximize the total value of the products submitted to ANVUR, whenever the same product has different co-authors some strategic issues come into play, and co-authors personal interest may induce them to cheat on the quality of their products. To see why, recall that ANVUR did not specify a division rule. Therefore, researchers consider *proj* as the division rule. Thus if *proj* is selected, and if (as we think) it is then used to reward R and hence researchers, co-authors may be competitors and might want to act strategically to improve their own score. It is clear that these kinds of manipulations, should be prevented by a good division

rule. Indeed this issues is addressed by a branch of game theory called *Mechanism design*. Note that, *proj* is definetely not fair. Thus, under the assumption that this rule will be used, researchers act strategically in order to have allocated the best products.

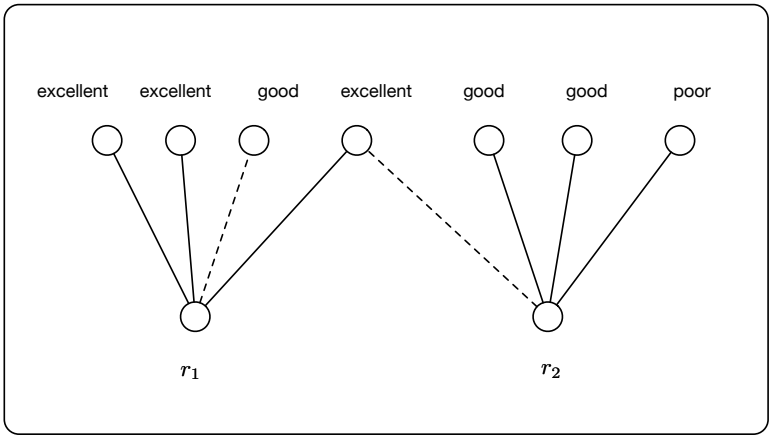


Figure 6.3: A clooser look at strategical behaviours.

This is exemplified in Figure 6.3. For example, researcher r_1 could manipulate the allocation process by being not truthful in order to get the best publications for herself and thus forcing r_2 to get the bad ones. The result is that the global optimum for R is missed (i.e., no efficiency) and moreover there is no fairness at all. Note that this strategic behavior is likely to happen as soon as two researchers from different departments co-authored some work, and each of them is interested in providing as much as contribution as possible to her/his department. For instance, in Figure 6.4 reports data about co-authorship at University of Calabria while in Figure 6.5 we illustrate the number of research groups with their size.

From these pictures we see that a very particular interactions between researchers emerges. Indeed 30 components out of 14 consist of only two researchers. Moreover, for the largest component which comprise 27 researchers we discovered that treewidth is really low Figure 6.6. We will see in the next chapter that these kinds of information

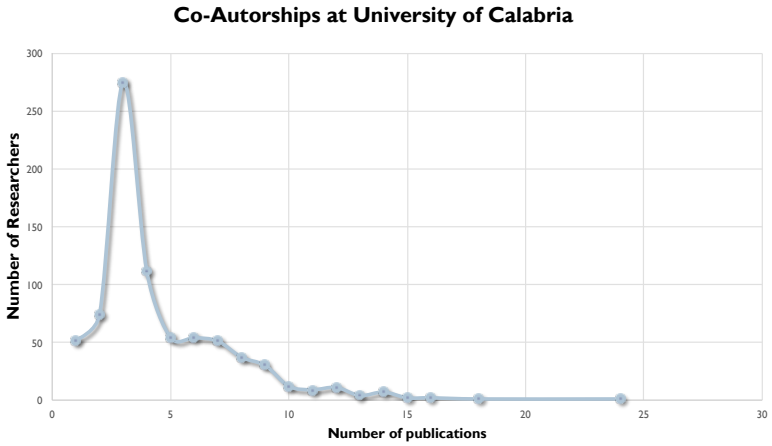


Figure 6.4: Co-autorships at University of Calabria. Researchers against the number of co-authored products

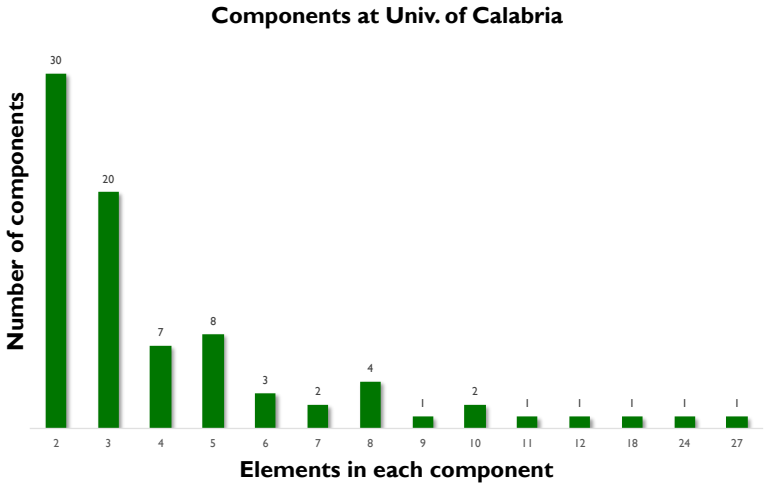


Figure 6.5: The number of components at University of Calabria

are of great interest (specifically, to the complexity of computing the Shapley value in this setting). We conclude the section by referring the reader to [Greco and Scarcello, 2013b] for more on this subject.

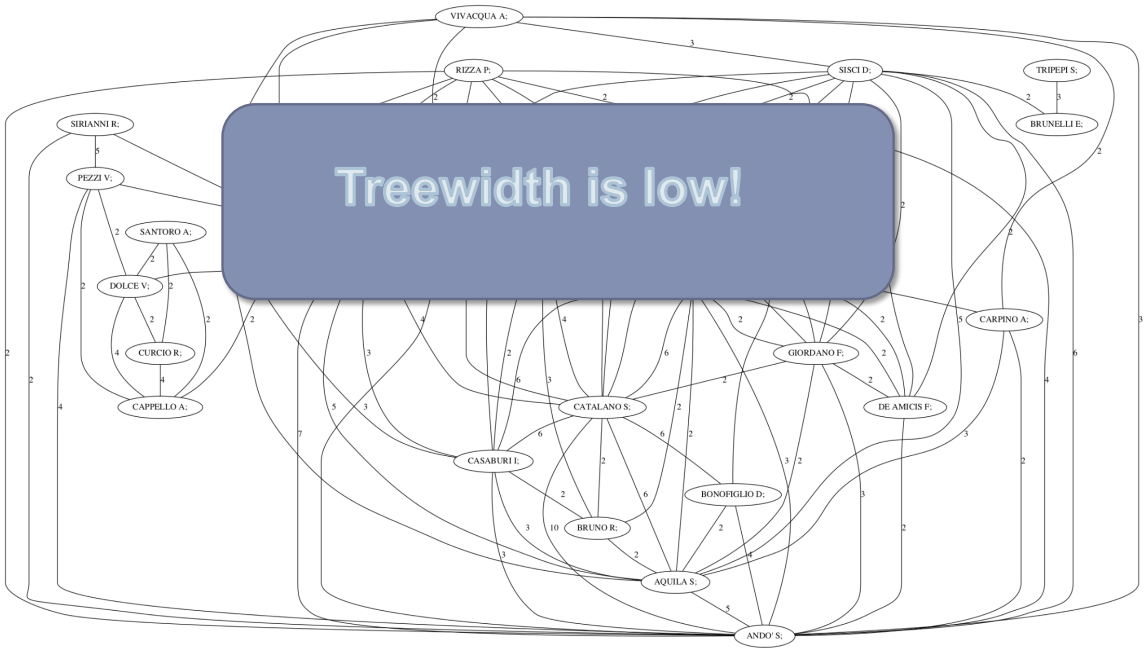


Figure 6.6: An example component

CHAPTER 7

Structural Tractability of Shapley and Banzhaf Values in Allocation Games

7.1 Introduction

It is often required to reward agents according to their actual contribution to the total worth of their organization. A notable case is when a set of indivisible goods (or tasks, or whatever) should be allocated to a given set of agents, and a monetary compensation is possible. Coalitional games, and in particular allocation games, properly model such applications, and the Shapley value and the Banzhaf value are two well-known solution concepts that provide fair and hence acceptable agent worths. Unfortunately, it turns out that their computation is a #P-hard problem, for which only quite trivial tractable classes are known.

In this chapter we identify large islands of tractability for both solution concepts, by exploiting the structure of the interactions among agents. The main result is that for any class of games having interaction graphs of bounded treewidth, both the Shapley value and the Banzhaf value are computable in polynomial time. Tractability also holds for games where each good is owned by at most two agents (independently of agents' interactions). These results allow us to deal with games having hundreds of agents in real-world applications. Moreover, we believe that the underlying technical ingredients can be exploited for different kinds of coalitional games, too.

For a better understanding of the problem, we first strengthen the known hardness results to the case of goods with one possible value only. Then, we look for islands of tractability of allocations problems. To this end, we provide a characterization of the marginal contribution of an agent to any coalition in terms of certain properties of good allocations, which are not required to be optimal ones. Such a technical tool allows us to point out the tractability of allocation games where every good is shared (or claimed for) by two agents at most.

Our main result, also based on the tool discussed in Chapter 5 and on further ingredients exploiting constraint satisfaction techniques, is a polynomial-time algorithm for the computation of the Shapley value and the Banzhaf value in allocation games where agent interactions have a tree-like structure—formally, have bounded *treewidth* [Robertson and Seymour, 1984]. These games capture scenarios of practical interest; for instance, in Chapter 6 we analyzed the data of a concrete instantiation for the setting described by [Greco and Scarcello, 2014b]. The setting refers to an allocation problem arising in the Italian Research Assessment program. The instantiation refers to the data of the University of Calabria by discovering that the treewidth of the associated graph, consisting of more than 500 nodes, is just 9. Moreover, the main result and the technical tools used to get it have their own theoretical interest, and the analysis of the complexity of reasoning problems related to coalitional games over classes of instances having some useful structural property is an active topic of research in artificial intelligence. In fact, even if we focus on the Shapley and the Banzhaf values, the methodology we propose may have a wider spectrum of applicability, as its salient ideas can be reused in other contexts.

7.2 Intractability of computation

Computing the Shapley value is a problem that has been shown to be #P-complete on different classes of games (see, e.g., [Deng and Papadimitriou, 1994; Nagamochi *et al.*, 1997; Bachrach and Rosenschein, 2009; Aziz and de Keijzer, 2014]), including the allocation games [**?**]. In particular, hardness has been shown to hold even on instances whose goods have three possible values. Below, we improve the result by showing that there is no advantage in focusing on scenarios where all goods have the same value. To this end, we first prove the #P-hardness of computing the *Banzhaf value*.

Theorem 7.1. *Computing the Banzhaf value is* #P*-hard on allocation games (under Turing reductions), even for scenarios* $\mathcal{A} = \langle N, \mathbb{G}, \Omega, \mathtt{val} \rangle$ *such that* $|\{\mathtt{val}(g) \mid g \in \mathbb{G}\}| = 1$.

Proof Sketch. Let $(S \cup I, E)$ be a bipartite graph, hence with $S \cap I = \emptyset$ and $E \subseteq S \times I$. Computing the number of subsets $C \subseteq S$ of vertices to which all vertices in I can be *matched* is #P-hard [Colbourn *et al.*, 1995].

Based on $(S \cup I, E)$, let us build the allocation scenario $\mathcal{A} = \langle S \cup \{|S|+1\}, I, \Omega, \mathtt{val} \rangle$ where nodes in S (resp., I) are transparently viewed as the agents (resp., goods), where $\mathtt{val}(g) = 1$ for each $g \in I$, and where $\Omega(i) = \{g \in I \mid \{i, g\} \in E\}$ while $\Omega(|S| + 1) = I$. Consider then the allocation game $\mathcal{G}_{\mathcal{A}} = \langle N, v_{\mathcal{A}} \rangle$ with $N = S \cup \{|S| + 1\}$, and the *Banzhaf value* $\beta_{|S|+1}(\mathcal{G}_{\mathcal{A}})$. Observe that, for any given coalition $C \subseteq S = N \setminus \{|S| + 1\}$, $v(C \cup \{|S| + 1\}) - v(C) = 0$ if, and only if, $C \subseteq S$ is a set of vertices to which all vertices in I can be matched. Eventually, $\beta_{|S|+1}(\mathcal{G}_{\mathcal{A}}) \times 2^{|S|}$ is the number of subsets $C \subseteq S$ for which some vertex in I cannot be matched, and $2^{|S|} - \beta_{|S|+1}(\mathcal{G}_{\mathcal{A}}) \times 2^{|S|}$ is the desired number, which can be computed in polynomial

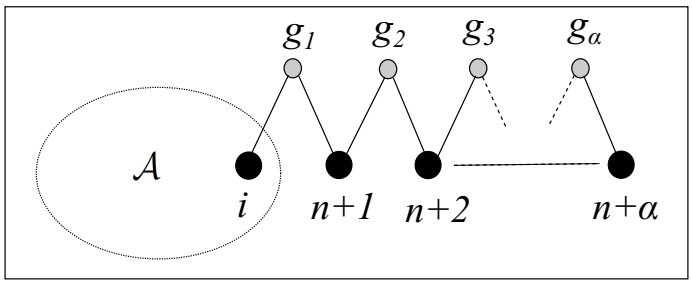


Figure 7.1: Illustration in the proof of Theorem 7.1: Construction of the scenario $\mathcal{A}_\alpha$ based on $\mathcal{A}$.

time once the Banzhaf value $\beta_{|S|+1}(\mathcal{G}_\mathcal{A})$ is known. □

Now, we show that the Banzhaf value of allocation games can be computed in polynomial time based on the knowledge of the Shapley value, so that this latter concept turns out to be #P-hard too. This property was already known to hold over (certain) *simple* games [Aziz *et al.*, 2009]. For its proof, we shall exploit the fact that, for each agent $i \in N$, the Shapley value can be rewritten as follows:

$$\phi_i(\mathcal{G}_\mathcal{A}) = \sum_{h=0}^{n-1} \frac{h!(n-h-1)!}{n!} \beta_i(\mathcal{G}_\mathcal{A}, h), \tag{7.1}$$

where, for each $h \in \{0, ..., n-1\}$,

$$\beta_i(\mathcal{G}_\mathcal{A}, h) = \sum_{C \subseteq N \setminus \{i\}, |C|=h} (v(C \cup \{i\}) - v(C)).$$

Theorem 7.2. *Computing the Shapley value is* #P*-hard on allocation games (under Turing reductions), even for scenarios* $\mathcal{A} = \langle N, \mathbb{G}, \Omega, \mathtt{val} \rangle$ *such that* $|\{\mathtt{val}(g) \mid g \in \mathbb{G}\}| = 1$.

Proof Sketch. Assume that $|\{\mathtt{val}(g) \mid g \in \mathbb{G}\}| = 1$; in particular, w.l.o.g., assume that $\mathtt{val}(g) = 1$ holds, for each $g \in \mathbb{G}$. Let i be an agent in N, and for each $\alpha \in \{1, ..., n\}$, consider the allocation scenario $\mathcal{A}_\alpha = \langle \{1, ..., n+\alpha\}, \mathbb{G} \cup \{g_1, ..., g_\alpha\}, \Omega_\alpha, \mathtt{val}_\alpha \rangle$ defined as follows:

- on the agents in $N \setminus \{i\}$, Ω_α coincides with Ω; moreover, $\Omega_\alpha(n+j) = \{g_j, g_{j+1}\}$, for each $j \in \{1, ..., \alpha - 1\}$; $\Omega_\alpha(i) = \Omega(i) \cup \{g_1\}$; and $\Omega_\alpha(n + \alpha) = \{g_\alpha\}$;
- on the goods in $\mathbb{G}$, $\texttt{val}_\alpha$ coincides with $\texttt{val}$; moreover, $\texttt{val}_\alpha(g_j) = 1$ for each $j \in \{1, ..., \alpha\}$.

Let $C \subseteq N \cup \{n + 1, ..., (n + \alpha - 1)\}$, and consider the illustration in Figure 7.1 to help the intuition. Observe that if $\{i, n + 1, ..., (n + \alpha - 1)\} \not\subseteq C$, then $v_{\mathcal{A}_\alpha}(C \cup \{n + \alpha\}) - v_{\mathcal{A}_\alpha}(C) = 1$. Otherwise, i.e., if $\{i, n + 1, ..., (n + \alpha - 1)\} \subseteq C$, then $v_{\mathcal{A}_\alpha}(C \cup \{n+\alpha\}) - v_{\mathcal{A}_\alpha}(C) = v_{\mathcal{A}}(C' \cup \{i\}) - v_{\mathcal{A}}(C')$, where $C' = C \setminus \{i, n+1, ..., n+\alpha\}$. Therefore, for each $h \in \{0, ..., (n + \alpha - 1)\}$, we have that:

$$\beta_{n+\alpha}(\mathcal{G}_{\mathcal{A}_\alpha}, h) = \begin{cases} \kappa_{n+\alpha}(h) + \beta_i(\mathcal{G}_{\mathcal{A}}, h - \alpha) & \text{if } h \geq \alpha \\ \kappa_{n+\alpha}(h) & \text{if } h < \alpha \end{cases}$$

where $\kappa_{n+\alpha}(h)$ is the number of coalitions C such that $|C| = h$ and such that $\{i, n + 1, ..., (n + \alpha - 1)\} \not\subseteq C$.

Note that $\kappa_{n+\alpha}(h)$ can be computed in polynomial time.

By using the above expression for $\beta_{n+\alpha}(\mathcal{G}_{\mathcal{A}_\alpha}, h)$ in Equation (7.1), we can derive that:

$$\phi_{n+\alpha}(\mathcal{G}_{\mathcal{A}_\alpha}) = \sum_{h=0}^{n+\alpha-1} \frac{h!(n + \alpha - h - 1)!}{(n + \alpha)!} \kappa_{n+\alpha}(h) + \sum_{h=0}^{n-1} \frac{(h + \alpha)!(n - h - 1)!}{(n + \alpha)!} \beta_i(\mathcal{G}_{\mathcal{A}}, h).$$

Eventually, by instantiating α with each value in $\{1, ..., n\}$, we get a system of n linear equations over the variables $\beta_i(\mathcal{G}_{\mathcal{A}}, 0), ..., \beta_i(\mathcal{G}_{\mathcal{A}}, n - 1)$. By multiplying by $(n + \alpha)!$ all the terms of the corresponding equation, we get a system where $(h + \alpha)!(n - h - 1)!$ is the coefficient of any term of the form $\beta_i(\mathcal{G}_{\mathcal{A}}, h)$. Given the form of these coefficients, the equations are easily seen to be linear independent—see [Aziz and de Keijzer, 2014; Aziz *et al.*, 2009].

Therefore, the values of $\beta_i(\mathcal{G}_\mathcal{A}, 0), ..., \beta_i(\mathcal{G}_\mathcal{A}, n-1)$ for which the system admits a solution are univocally determined. Moreover, they might be computed in polynomial time (e.g., by Bareiss's implementation of Gaussian elimination [Bareiss, 1968]), if we were able to compute in polynomial time $\phi_{n+\alpha}(\mathcal{G}_{\mathcal{A}_\alpha})$, for each $\alpha \in \{1, ..., n\}$. To conclude, just notice that the Banzhaf value of agent i in $\mathcal{G}_\mathcal{A}$ is precisely given by $\beta_i(\mathcal{G}_\mathcal{A}) = \frac{1}{2^{n-1}} \sum_{h=0}^{n-1} \beta_i(\mathcal{G}_\mathcal{A}, h)$. □

From these results, it turns out that acting on the values of goods does not help very much in identifying tractable classes of instances. So, we next consider different kinds of restrictions based on the "interactions" that emerge among agents.

7.3 Characterizations of the Shapley value

Throughout the section, assume that an allocation scenario $\mathcal{A} = \langle N, \mathbb{G}, \Omega, \mathtt{val} \rangle$ is given. Let $\{w_1, ..., w_m\} = \{\mathtt{val}(g) \mid g \in \mathbb{G}\} \cup \{0\}$ be the set of all values associated with goods in $\mathbb{G}$ (plus the null value 0, if not present), and assume that $w_1 > w_2 > \cdots > w_m$. W.l.o.g, assume also that $w_m = 0$.

7.3.1 A closer look at marginal contributions

In this section, we elaborate and discuss useful characterizations of the Shapley value of allocation games.

We start with a simple reformulation. Let $i \in N$ be an agent, let $h \in \{0, ..., n-1\}$, let $\ell \in \{1, ..., m\}$, and let us denote by $\#\mathrm{col}^i_\ell(\mathcal{G}_A, h)$ the number of coalitions C such that $|C| = h$ and $v_\mathcal{A}(C \cup \{i\}) - v_\mathcal{A}(C) \geq w_\ell$. Then, the coefficients $\beta_i(\mathcal{G}_\mathcal{A}, h)$ in the expression for the Shapely value illustrated in Equation (7.1) can be rewritten as follows.

Theorem 7.3. *For each agent $i \in N$ and $h \in \{0, ..., n-1\}$,*

$$\beta_i(\mathcal{G}_\mathcal{A}, h) = w_1 \times \ \#\mathrm{col}_1^i(\mathcal{G}_\mathcal{A}, h) + \sum_{\ell=2}^{m} w_\ell \times \left(\#\mathrm{col}_\ell^i(\mathcal{G}_\mathcal{A}, h) - \#\mathrm{col}_{\ell-1}^i(\mathcal{G}_\mathcal{A}, h)\right).$$

Proof. Recall that, for each $i \in N$ and $h \in \{0, ..., n-1\}$,

$$\beta_i(\mathcal{G}_\mathcal{A}, h) = \sum_{C \subseteq N \setminus \{i\}, |C|=h} (v(C \cup \{i\}) - v(C)).$$

Consider an arbitrary coalition $C \subseteq N \setminus \{i\}$ with $|C| = h$. Let $\bar{\pi}$ be an optimal allocation for $\mathcal{A}[C \cup \{i\}]$, and observe that because of the monotonicity of allocation games, there is an optimal allocation $\bar{\pi}_{\text{-}i}$ for $\mathcal{A}[C]$ such that $\texttt{img}(\bar{\pi}_{\text{-}i}) \subseteq \texttt{img}(\bar{\pi})$. If the two images coincide, then we clearly have that $v_\mathcal{A}(C \cup \{i\}) - v_\mathcal{A}(C) = 0$. Otherwise, $\texttt{img}(\bar{\pi}) \setminus \texttt{img}(\bar{\pi}_{\text{-}i})$ is a singleton $\{g\}$, and $v_\mathcal{A}(C \cup \{i\}) - v_\mathcal{A}(C) = \texttt{val}(\bar{\pi}) - \texttt{val}(\bar{\pi}_{\text{-}i}) = \texttt{val}(g)$. In both cases, the marginal contribution is a value w_ℓ with $\ell \in \{1, ..., m\}$. Then, we can rewrite the expression for the coefficients as $\beta_i(\mathcal{G}_\mathcal{A}, h) = \sum_{\ell=1}^{m} w_\ell \times M_\ell^i(\mathcal{G}_\mathcal{A}, h)$, where $M_\ell^i(\mathcal{G}_\mathcal{A}, h)$ is the number of coalitions C with $|C| = h$ and such that $v_\mathcal{A}(C \cup \{i\}) - v_\mathcal{A}(C) = w_\ell$. By definition of $\#\mathrm{col}_l^i(\mathcal{G}_\mathcal{A}, h)$, we get $M_1^i(\mathcal{G}_\mathcal{A}, h) = \#\mathrm{col}_1^i(\mathcal{G}_\mathcal{A}, h)$ and, $\forall \ell \geq 2$, $M_l^i(\mathcal{G}_\mathcal{A}, h) = \#\mathrm{col}_\ell^i(\mathcal{G}_\mathcal{A}, h) - \#\mathrm{col}_{\ell-1}^i(\mathcal{G}_\mathcal{A}, h)$. □

Hence, counting the number of coalitions to which an agent i provides some marginal contribution is deeply related to the computation of the Shapley (and the Banzhaf) value of allocation games. We now further explore this specific task.

For each "level" $\ell \in \{1, ..., m\}$ over the possible values, an agent $i \in N$ is said to be *dependent at level* ℓ (short: ℓ-dependent) if for each $g \in \Omega(i)$ with $\texttt{val}(g) \geq w_\ell$, there is an agent $j \in N \setminus \{i\}$ such that $g \in \Omega(j)$. In particular, note that any agent having goods with value at least w_ℓ and which are not shared with any other agent is not dependent at level ℓ; in fact, all her marginal contributions are at least w_ℓ, independently of the coalition to be considered. Let $G_\ell = (N_\ell, E_\ell)$ be the undirected

graph where N_ℓ is the set of all ℓ-dependent agents and where $\{i,j\} \in E_\ell$ if, and only if, there is a good $g \in \Omega(i) \cap \Omega(j)$ with $\texttt{val}(g) \geq w_\ell$. Then, a coalition $R \subseteq N_\ell$ of agents is called a *component at level* ℓ (short: ℓ-component) if the subgraph of G_ℓ induced by the nodes in R is connected. As a special case, if there is no good $g \in \Omega(i)$ with $\texttt{val}(g) \geq w_\ell$, then $\{i\}$ is an ℓ-component.

Example 7.4. *Consider the scenario $\mathcal{A}_0$ reported in Figure 6.1. We have $w_1 = \texttt{val}(g_1)$. Moreover, $\{1,2\}$ and $\{3\}$ are the only subset-maximal components at level 1. Indeed, $g_1 \in \Omega(1) \cap \Omega(2)$, and there is no good in $\Omega(3)$ with value w_1.* $\lhd$

The concept of ℓ-component plays a crucial role to characterize useful properties of marginal contributions.

Lemma 7.5. *Let $R \cup \{i\}$ be an ℓ-component, with $R \subseteq N_\ell \setminus \{i\}$. Then, the following statements are equivalent:*

(1) $v_{\mathcal{A}}(R \cup \{i\}) - v_{\mathcal{A}}(R) \geq w_\ell$;

(2) there is an optimal allocation $\bar{\pi}$ for $\mathcal{A}[R \cup \{i\}]$ such that $\texttt{val}(\bar{\pi}) \geq w_\ell$, for each $j \in R \cup \{i\}$.

Proof Sketch. (1)⇒(2) Assume that (2) does not hold. So, we are guaranteed about the existence of an optimal allocation $\bar{\pi}$ for $\mathcal{A}[R \cup \{i\}]$ and of an agent $j' \in R \cup \{i\}$ such that $\texttt{val}(\bar{\pi}(j')) < w_\ell$. Consider the following two possible cases.

First, assume that $\texttt{val}(\bar{\pi}(i)) < w_\ell$. Since the restriction of $\bar{\pi}$ over the agents in R is a feasible allocation for $\mathcal{A}[R]$, then we immediately get that $v_{\mathcal{A}}(R) \geq \texttt{val}(\bar{\pi}) - \texttt{val}(\bar{\pi}(i)) > \texttt{val}(\bar{\pi}) - w_\ell$, and hence $v_{\mathcal{A}}(R \cup \{i\}) - v_{\mathcal{A}}(R) < w_\ell$.

Second, assume that $\texttt{val}(\bar{\pi}(i)) \geq w_\ell$. We start by observing that, due to the optimality of $\bar{\pi}$, for each agent $j' \in R \cup \{i\}$ with $\texttt{val}(\bar{\pi}(j')) < w_\ell$, $\{g \mid g \in \Omega(j') \wedge \texttt{val}(g) \geq w_\ell\} \subseteq \texttt{img}(\bar{\pi})$. That is, goods that might be in principle allocated to an

agent $j' \in R \cup \{i\}$ with $\texttt{val}(\bar{\pi}(j')) < w_\ell$ and having value at least w_ℓ are actually allocated to some different agent in $R \cup \{i\}$. Given that $R \cup \{i\}$ is an ℓ-component (and that, in particular, each agent is ℓ-dependent), we are guaranteed about the existence of a succession $i = j'_1, j'_2, ..., j'_h$ such that $\bar{\pi}(j'_x) \cap \Omega(j'_{x+1}) \neq \emptyset$, for each $x \in \{1, ..., h\text{-}1\}$; and $\texttt{val}(\pi(j'_h)) < w_\ell$. Consider then the function $\bar{\pi}_{\text{-}i} : R \to \mathbb{G} \cup \{\emptyset\}$ with $\bar{\pi}_{\text{-}i}(j'_{x+1}) = \bar{\pi}(j'_x)$, for each $x \in \{1, ..., h\text{-}1\}$; and $\bar{\pi}_{\text{-}i}(j'') = \bar{\pi}(j'')$, for each $j'' \in R \setminus \{j'_2, ..., j'_h\}$. Then, $\bar{\pi}_{\text{-}i}$ is an allocation for $\mathcal{A}[R]$ and we have that $\texttt{val}(\bar{\pi}_{\text{-}i}) = \texttt{val}(\bar{\pi}) - \texttt{val}(\bar{\pi}(j'_h))$. Hence, $v_{\mathcal{A}}(R) \geq \texttt{val}(\bar{\pi}_{\text{-}i}) = \texttt{val}(\bar{\pi}) - \texttt{val}(\bar{\pi}(j'_h)) > \texttt{val}(\bar{\pi}) - w_\ell$. That is, $v_{\mathcal{A}}(R \cup \{i\}) - v_{\mathcal{A}}(R) < w_\ell$.

In both cases, we have concluded that (1) does not hold.

(2)⇒(1) Let $\bar{\pi}$ be an optimal allocation for $\mathcal{A}[R \cup \{i\}]$ such that $\texttt{val}(\bar{\pi}) \geq w_\ell$, for each $j \in R \cup \{i\}$. Because of the allocation monotonicity property, there is an optimal allocation $\bar{\pi}_{\text{-}i}$ for $\mathcal{A}[R]$ such that $\texttt{img}(\bar{\pi}_{\text{-}i}) \subseteq \texttt{img}(\bar{\pi})$. Hence, for each $j \in R$, $\texttt{val}(\bar{\pi}_{\text{-}i}(j)) \geq w_\ell$. Now, observe that $v_{\mathcal{A}}(R \cup \{i\}) = \texttt{val}(\texttt{img}(\bar{\pi}))$ and $v_{\mathcal{A}}(R) = \texttt{val}(\texttt{img}(\bar{\pi}_{\text{-}i}))$. So, $v_{\mathcal{A}}(R \cup \{i\}) - v_{\mathcal{A}}(R)$ coincides with the value of one of the goods in $\texttt{img}(\bar{\pi})$, and $v_{\mathcal{A}}(R \cup \{i\}) - v_{\mathcal{A}}(R) \geq w_\ell$. □

Example 7.6. *In our running example,* $\{1, 2\}$ *is a component at level 1, and* g_1 *is the only good having value (at least)* w_1*. So, there is no allocation* $\bar{\pi}$ *for* $\mathcal{A}_0[\{1, 2\}]$ *satisfying condition (2) in Lemma 7.5 for* $\ell = 1$*, and hence* $v_{\mathcal{A}_0}(\{1, 2\}) - v_{\mathcal{A}_0}(\{2\}) < w_1$*. Eventually, by moving to the second level, the allocation for* $\mathcal{A}_0[\{1, 2\}]$ *depicted in Figure 6.1 witnesses, again by Lemma 7.5, that* $v_{\mathcal{A}_0}(\{1, 2\}) - v_{\mathcal{A}_0}(\{2\}) \geq w_2$*.* ◁

We now generalize Lemma 7.5 to arbitrary coalitions and allocations that are not required to be optimal. To this end, if $C \subseteq N$ is a coalition and $i \notin C$ is an agent, then we denote by $\mathbf{p}^i_\ell(C)$ the *ℓ-part* of C w.r.t. i. This is defined as the emptyset

if $i \notin N_\ell$; otherwise, $\mathbf{p}^i_\ell(C)$ is the subset-maximal (in fact, unique) ℓ-component $R \subseteq C \cup \{i\}$ with $i \in R$.

Theorem 7.7. *Let $C \subseteq N$ be a coalition and let $i \in N \setminus C$ be an agent. Then, the following statements are equivalent:*

(1) $v_{\mathcal{A}}(C \cup \{i\}) - v_{\mathcal{A}}(C) \geq w_\ell$;

(2) there is an allocation $\bar{\pi}$ for $\mathcal{A}[\mathbf{p}^i_\ell(C)]$ such that $\mathtt{val}(\bar{\pi}(j)) \geq w_\ell$, for each $j \in \mathbf{p}^i_\ell(C)$.

Proof. Let us address preliminary the case where $i \notin N_\ell$. In this case, $\mathbf{p}^i_\ell(C) = \emptyset$ and (2) trivially holds. Moreover, since agent i is not ℓ-dependent, its marginal contribution to any coalition is at least w_ℓ. So, (1) holds trivially, too. In the remaining, consider the case where $i \in N_\ell$, so that $i \in \mathbf{p}^i_\ell(C)$. Let $R = \mathbf{p}^i_\ell(C) \setminus \{i\}$ and let $S = C \setminus R$.

(1)⇒(2) By submodularity, we know that $v_{\mathcal{A}}(R \cup \{i\}) - v_{\mathcal{A}}(R) \geq v_{\mathcal{A}}(R \cup \{i\} \cup S) - v_{\mathcal{A}}(R \cup S)$ holds. Hence, we get that $v_{\mathcal{A}}(R \cup \{i\}) - v_{\mathcal{A}}(R) \geq w_\ell$. Then, by Lemma 7.5, we derive that there is an allocation $\bar{\pi}$ for $\mathcal{A}[R \cup \{i\}]$ such that $\mathtt{val}(\bar{\pi}(j)) \geq w_\ell$, for each $j \in R \cup \{i\}$.

(2)⇒(1) Assume that π is an allocation for $\mathcal{A}[R \cup \{i\}]$ such that $\mathtt{val}(\pi(j)) \geq w_\ell$, for each $j \in R \cup \{i\}$. We first claim that this property must hold for every optimal allocation, too. Assume by contradiction that there is an optimal allocation $\bar{\pi}$ for $\mathcal{A}[R \cup \{i\}]$ with $\mathtt{val}(\bar{\pi}(j')) < w_\ell$ for some agent $j' \in R \cup \{i\}$. Because $\mathtt{img}(\bar{\pi}) \neq \mathtt{img}(\pi)$, there exists a succession of agents $j' = j'_1, j'_2, ..., j'_h$ with the assignment to the last agent (possibly, just j') such that $\pi(j'_h) \notin \mathtt{img}(\bar{\pi})$ and $\pi(j'_x) = \bar{\pi}(j'_{x+1})$, for each $x \in \{1, ..., h\text{-}1\}$. Then, consider the function $\bar{\pi}^* : R \cup \{i\} \to \mathbb{G} \cup \{\emptyset\}$ such that $\bar{\pi}^*(j'_x) = \pi(j'_x)$, for each $x \in \{1, ..., h\}$; and $\bar{\pi}^*(j'') = \bar{\pi}(j'')$, for each $j'' \in (R \cup \{i\}) \setminus \{j'_1, ..., j'_h\}$.

Note that $\bar{\pi}^*$ is an allocation for $\mathcal{A}[R \cup \{i\}]$ and $\texttt{val}(\bar{\pi}^*) > \texttt{val}(\bar{\pi})$, because the only difference is that the new image contains the good $\pi(j'_h)$, whose value is at least w_ℓ, instead of the good $\bar{\pi}(j'_1)$ for which $\texttt{val}(\bar{\pi}(j'_1)) < w_\ell$ holds. However, this contradicts the optimality of $\bar{\pi}$.

So far, we have shown that there is an optimal allocation $\bar{\pi}$ for $\mathcal{A}[R \cup \{i\}]$ such that $\texttt{val}(\bar{\pi}(j)) \geq w_\ell$, for each $j \in R \cup \{i\}$. By Lemma 7.5, this entails that $v_{\mathcal{A}}(R \cup \{i\}) - v_{\mathcal{A}}(R) \geq w_\ell$ holds. Moreover, by monotonicity, there is an optimal allocation $\bar{\pi}_{\text{-}i}$ for $\mathcal{A}[R]$ such that $\texttt{img}(\bar{\pi}_{\text{-}i}) \subseteq \texttt{img}(\bar{\pi})$. Hence, for each $j \in R$, $\texttt{val}(\bar{\pi}_{\text{-}i}(j)) \geq w_\ell$. Given these properties of the optimal allocations for $\mathcal{A}[R \cup \{i\}]$ and $\mathcal{A}[R]$, and given the definition of S, we conclude that $\texttt{opt}(\mathcal{A}[R \cup \{i\} \cup S]) = \texttt{opt}(\mathcal{A}[R \cup \{i\}]) + \texttt{opt}(\mathcal{A}[S])$ and $\texttt{opt}(\mathcal{A}[R \cup S]) = \texttt{opt}(\mathcal{A}[R]) + \texttt{opt}(\mathcal{A}[S])$. So, $v_{\mathcal{A}}(R \cup \{i\} \cup S) - v_{\mathcal{A}}(R \cup S) = v_{\mathcal{A}}(R \cup \{i\}) - v_{\mathcal{A}}(R) \geq w_\ell$. □

Example 7.8. *By continuing with Example 7.6, note that* $\{1, 2\} = \mathbf{p}^1_\ell(\{2, 3\})$ *holds, for* $\ell \in \{1, 2\}$. *Therefore, the allocation for* $\mathcal{A}_0[\{1, 2\}]$ *depicted in Figure 6.1 witnesses, by Theorem 7.7, that* $v_{\mathcal{A}_0}(\{1, 2, 3\}) - v_{\mathcal{A}_0}(\{2, 3\}) \geq w_2$. ◁

7.4 Islands of tractability

7.4.1 Bounded sharing

Our analysis intensively uses Theorem 7.7. The first outcome is an island of tractability based on the notion of bounded sharing. Formally, for a given level ℓ, define the *sharing degree* of an allocation scenario $\mathcal{A}$, denoted by $sd_\ell(\mathcal{A})$, as the maximum, over all goods g with $\texttt{val}(g) \geq w_\ell$, of $|\{j \in N \mid g \in \Omega(j)\}|$. Intuitively, it measures the maximum number of agents competing for the same good (with value at least w_ℓ).

Lemma 7.9. *Assume that* $sd_\ell(\mathcal{A}) \leq 2$, *for a value level* ℓ. *Let* i *be an agent in* N,

and let $h \in \{0, ..., n-1\}$. *Then,*

$$\#\mathrm{col}^i_\ell(\mathcal{G}_\mathcal{A}, h) = \frac{(n-1)!}{(n-1-h)!h!} - \mathcal{X}, \text{ where}$$

- $\mathcal{X} = 0$, *if* $h < |\mathbf{p}^i_\ell(N \setminus \{i\})| - 1$; *or if* i *is not* ℓ*-dependent, or the subgraph of* G_ℓ *induced by the nodes in* $\mathbf{p}^i_\ell(N \setminus \{i\})$ *contains a cycle, or there are two agents* j *and* j' *in* $\mathbf{p}^i_\ell(N \setminus \{i\})$ *with* $|\Omega(j) \cap \Omega(j') \cap \{g \mid \mathtt{val}(g) \geq w_\ell\}| > 1$.
- $\mathcal{X} = \frac{(n-|\mathbf{p}^i_\ell(N\setminus\{i\})|)!}{(n-h-1)!(h+1-|\mathbf{p}^i_\ell(N\setminus\{i\})|)!}$, *otherwise.*

Proof Sketch. Let $C \subseteq N$ be any coalition with $i \notin C$. We claim that: (1) $v_\mathcal{A}(C \cup \{i\}) - v_\mathcal{A}(C) \geq w_\ell$ holds if, and only if, (2) either i is not ℓ-dependent; or the subgraph of G_ℓ induced by the nodes in $\mathbf{p}^i_\ell(N \setminus \{i\})$ contains a cycle; or there are agents j and j' in $\mathbf{p}^i_\ell(N \setminus \{i\})$ with $|\Omega(j) \cap \Omega(j') \cap \{g \mid \mathtt{val}(g) \geq w_\ell\}| > 1$; or $\mathbf{p}^i_\ell(N \setminus \{i\}) \neq \mathbf{p}^i_\ell(C)$.

(1)⇒(2) Assume that (2) does not hold. So, the subgraph of G_ℓ induced by the nodes in $\mathbf{p}^i_\ell(N \setminus \{i\})$ is acyclic and $\mathbf{p}^i_\ell(N \setminus \{i\}) = \mathbf{p}^i_\ell(C)$ holds, with $i \in \mathbf{p}^i_\ell(C)$. Moreover, for each pair j, j' of agents in $\mathbf{p}^i_\ell(C)$, we have $|\Omega(j) \cap \Omega(j') \cap \{g \mid \mathtt{val}(g) \geq w_\ell\}| \leq 1$. Therefore, by definition of G_ℓ and given that $sd_\ell(\mathcal{A}) \leq 2$, the goods in $\bigcup_{j \in \mathbf{p}^i_\ell(C)} \Omega(j)$ having value at least w_ℓ have a one-to-one correspondence with the edges of the subgraph of G_ℓ induced by the nodes in $\mathbf{p}^i_\ell(C)$, which is a tree. Hence, the number of these goods is $|\mathbf{p}^i_\ell(C)| - 1$. Therefore, there is no allocation π such that $\mathtt{val}(\pi(j)) \geq w_\ell$, for each $j \in \mathbf{p}^i_\ell(C)$. By Theorem 7.7, we conclude that (1) does not hold.

(2)⇒(1) If i is not ℓ-dependent, then it is immediate that $v_\mathcal{A}(C \cup \{i\}) - v_\mathcal{A}(C) \geq w_\ell$. So assume that i is ℓ-dependent.

Assume that there are two agents j' and j'' in $\mathbf{p}^i_\ell(C)$ such that $\Omega(j') \cap \Omega(j'') \cap \{g \mid \mathtt{val}(g) \geq w_\ell\} \supseteq \{g', g''\}$ with $g' \neq g''$. Consider a spanning tree T of the subgraph of G_ℓ induced by the nodes in $\mathbf{p}^i_\ell(C)$, and let us root it at agent j'. Consider then the

allocation $\bar{\pi}$ built as follows: $\bar{\pi}(j') = g'$, $\bar{\pi}(j'') = g''$; and for each agent $j_c \neq j''$ whose parent in T is j_p, let $\bar{\pi}(j_c)$ be any arbitrary good in $\Omega(j_c) \cap \Omega(j_p) \cap \{g \mid \texttt{val}(g) \geq w_\ell\}$. Since $sd_\ell(\mathcal{A}) \leq 2$, $\bar{\pi}$ is well-defined. Moreover, $\bar{\pi}$ satisfies condition (2) in Theorem 7.7, and we can conclude that $v_{\mathcal{A}}(C \cup \{i\}) - v_{\mathcal{A}}(C) \geq w_\ell$.

In the rest, assume that for each pair j, j' of agents in $\mathbf{p}^i_\ell(N \setminus \{i\})$, $|\Omega(j) \cap \Omega(j') \cap \{g \mid \texttt{val}(g) \geq w_\ell\}| \leq 1$. Consider the case where $\mathbf{p}^i_\ell(N \setminus \{i\}) \neq \mathbf{p}^i_\ell(C)$, and let j be an agent in $\mathbf{p}^i_\ell(C)$ adjacent to an agent $j' \in \mathbf{p}^i_\ell(N \setminus \{i\}) \setminus \mathbf{p}^i_\ell(C)$. Consider the spanning tree T of the subgraph of G_ℓ induced by the nodes in $\mathbf{p}^i_\ell(C)$, and let us root it at agent j. Consider then the allocation $\bar{\pi}$ built as follows: $\bar{\pi}(j) = g$ with $\{g\} = \Omega(j) \cap \Omega(j') \cap \{g \mid \texttt{val}(g) \geq w_\ell\}$; and for each agent j_c whose parent in T is j_p, let $\bar{\pi}(j_c) = g_c$ with $\{g_c\} = \Omega(j_c) \cap \Omega(j_p) \cap \{g \mid \texttt{val}(g) \geq w_\ell\}$. Since $sd_\ell(\mathcal{A}) \leq 2$, $\bar{\pi}$ is well-defined. Moreover, $\bar{\pi}$ satisfies condition (2) in Theorem 7.7, and we conclude that $v_{\mathcal{A}}(C \cup \{i\}) - v_{\mathcal{A}}(C) \geq w_\ell$.

Finally, assume that $\mathbf{p}^i_\ell(N \setminus \{i\}) = \mathbf{p}^i_\ell(C)$ and that for each pair j, j' of agents in $\mathbf{p}^i_\ell(N \setminus \{i\})$, $|\Omega(j) \cap \Omega(j') \cap \{g \mid \texttt{val}(g) \geq w_\ell\}| \leq 1$. Assume that the subgraph of G_ℓ induced by the nodes in $\mathbf{p}^i_\ell(N \setminus \{i\})$ contains a cycle, say $j_1, ..., j_k, j_1$. Consider the spanning tree T of the subgraph of G_ℓ induced by the nodes in $\mathbf{p}^i_\ell(C)$. Assume, w.l.o.g., that the edge (j_{k-1}, j_1) is not in T and let us root T at j_1. Consider the allocation $\bar{\pi}$ such that: $\bar{\pi}(j_1) = g$ with $\{g\} = \Omega(j_1) \cap \Omega(j_{k-1}) \cap \{g \mid \texttt{val}(g) \geq w_\ell\}$; and for each agent j_c whose parent in T is j_p, let $\bar{\pi}(j_c) = g_c$ with $\{g_c\} = \Omega(j_c) \cap \Omega(j_p) \cap \{g \mid \texttt{val}(g) \geq w_\ell\}$. As above, $\bar{\pi}$ is well-defined and we have that $v_{\mathcal{A}}(C \cup \{i\}) - v_{\mathcal{A}}(C) \geq w_\ell$.

In the light of the above relationship, observe that if i is not ℓ-dependent, or the subgraph of G_ℓ induced by the nodes in $\mathbf{p}^i_\ell(N \setminus \{i\})$ contains a cycle, or there are agents j and j' in $\mathbf{p}^i_\ell(N \setminus \{i\})$ with $|\Omega(j) \cap \Omega(j') \cap \{g \mid \texttt{val}(g) \geq w_\ell\}| > 1$, or $h + 1 < |\mathbf{p}^i_\ell(N \setminus \{i\})|$ (so that $\mathbf{p}^i_\ell(N \setminus \{i\}) \neq \mathbf{p}^i_\ell(C)$), then any coalition $C \subseteq N \setminus \{i\}$

with $|C| = h$ is such that $v_{\mathcal{A}}(C \cup \{i\}) - v_{\mathcal{A}}(C) \geq w_\ell$. Indeed, these conditions are independent of C. Therefore, $\#\mathrm{col}^i_\ell(\mathcal{G}_{\mathcal{A}}, h)$ just consists of the overall number of coalitions C of cardinality h that can be built out of $n-1$ agents. This is given by $\frac{(n-1)!}{(n-1-h)!h!}$.

It remains to consider what happens if the conditions above that are independent of h do not hold, and the cardinality h of the considered coalitions is such that $h > |\mathbf{p}^i_\ell(N \setminus \{i\})| - 1$. In this case, for all coalitions C with $|C| = h$ obtained by extending $\mathbf{p}^i_\ell(N \setminus \{i\}) \setminus \{i\}$ with further $h - (|\mathbf{p}^i_\ell(N \setminus \{i\})| - 1)$ agents, $v_{\mathcal{A}}(C \cup \{i\}) - v_{\mathcal{A}}(C) < w_\ell$ holds. Therefore, $\#\mathrm{col}^i_\ell(\mathcal{G}_{\mathcal{A}}, h)$ is obtained by subtracting the number of such coalitions from the whole number of cardinality-h coalitions without agent i. □

Because of this closed form characterization and of Theorem 7.3, the following is immediately established.

Theorem 7.10. *The Shapley value (and the Banzhaf value) of allocation games $\mathcal{G}_{\mathcal{A}}$ can be computed in polynomial time on scenarios $\mathcal{A} = \langle N, G, \Omega, \mathtt{val}\rangle$ such that $sd_\ell(\mathcal{A}) \leq 2$, for each value level ℓ.*

Assessing whether tractability still holds with higher values of sharing degree is left as an open problem. In fact, one may recall that computing the number of perfect matching in a graph (cf. proof of Theorem 7.1) is #P-hard even on graph whose nodes have degree 3 [Dagum and Luby, 1992]. Hence, a negative answer to the question would be hardly surprising.

7.4.2 Bounded treewidth

In this subsection, we analyze the complexity of the Shapley value over allocation games where the interactions among agents have a tree-like structure. We use the

technical tools provided in Section 7.3, by combining them with CSP techniques that are of interest in their own.

For any scenario $\mathcal{A} = \langle N, \mathbb{G}, \Omega, \mathtt{val} \rangle$, its *interaction graph* is the undirected graph $G(\mathcal{A}) = (N, E)$ such that $\{i, j\} \in E$ if, and only if, there is a good $g \in \Omega(i) \cap \Omega(j)$.

Recall that a *tree decomposition* of a graph $G = (N, E)$ is a pair $\langle T, \chi \rangle$, where $T = (V, F)$ is a tree, and χ is a function assigning to each vertex $p \in V$ a set of nodes $\chi(p) \subseteq N$, such that the following conditions are satisfied: (1) $\forall b \in N$, $\exists p \in V$ such that $b \in \chi(p)$; (2) $\forall \{b, d\} \in E$, $\exists p \in V$ such that $\{b, d\} \subseteq \chi(p)$; (3) $\forall b \in N$, the set $\{p \in V \mid b \in \chi(p)\}$ induces a connected subtree of T. The *width* of $\langle T, \chi \rangle$ is $\max_{p \in V} |\chi(p) - 1|$, and the *treewidth* of G (short: $tw(G)$) is the minimum width over all its tree decompositions (see, e.g., [Robertson and Seymour, 1984]).

Example 7.11. *Consider again the allocation scenario $\mathcal{A}_0$ that is reported in Figure 6.1. Then, $G(\mathcal{A}_0)$ is a clique over the nodes in $\{1, 2, 3\}$. The treewidth of $G(\mathcal{A}_0)$ is 2, as it is witnessed by the tree decomposition $\mathtt{TD}_0 = \langle T_0, \chi_0 \rangle$ where T_0 consists of one vertex v only, and $\chi_0(v) = \{1, 2, 3\}$.* ◁

7.5 CSP encoding (for the Banzhaf value)

Recall that, given a CSP instance $\mathcal{I}$, deciding whether there is a solution (and compute one, if any) is generally NP-hard, but it is known to be feasible in polynomial time on classes of CSP instances $\mathcal{I}$ whose associated graphs have treewidth bounded by some given constant [Gottlob *et al.*, 2013]. In particular, we already pointed out (in Chapter 2) that these kinds of *structural tractability results* have been generalized to counting problems too Theorem 2.4.

In order to show the tractability of the Shapley value over allocation games having associated graphs with bounded treewidth, we shall encode the computation of the

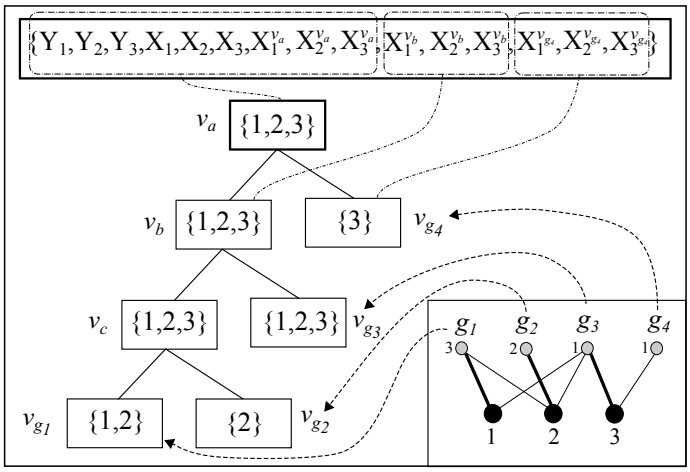


Figure 7.2: Decomposition in Example 7.12—the label of the root modified as in the proof of Theorem 7.13 is on the top.

coefficients $\#\mathrm{col}^i_\ell(\mathcal{G}_\mathcal{A}, h)$—see Theorem 7.3—in terms of a counting problem over a suitably defined CSP instance and we shall then make use of Theorem 2.4.

We recall that the challenge to be faced is to use a constant number of possible values for the auxiliary variables. For instance, the natural encoding where some variable X_j (associated with an agent $j \in N$) can take as values the goods in $\Omega(j)$ is not useful here. In fact, we propose an encoding that uses both the given allocation scenario $\mathcal{A}$ and a tree decomposition $\mathtt{TD} = \langle T, \chi \rangle$ of $G(\mathcal{A})$. The idea is that each good is associated with some distinguished vertex of T, while suitable variables in the labels of the tree encode the roadmaps to reach such goods. In particular, their domain just contains the needed road signs (five values are enough). This is detailed below.

We start by building a tree decomposition with certain desirable properties. Let $\langle T', \chi' \rangle$ be a tree decomposition of $G(\mathcal{A})$ whose width is $k > 0$. Note that, for each good $g \in \mathbb{G}$, we are guaranteed about the existence of a vertex v'_g in T' such that $\chi(v'_g) \supseteq \{j \mid g \in \Omega(j)\}$. Indeed, the agents in $\{j \mid g \in \Omega(j)\}$ form a clique in $G(\mathcal{A})$.

In a pre-processing step, we modify $\langle T', \chi' \rangle$ by adding a fresh vertex v_g as a child

of v'_g, whose label is $\chi(v_g) = \chi(v'_g) \cap \{j \mid g \in \Omega(j)\}$. By iterating over all goods, we get the desired tree where each good g is associated with a distinguished vertex (in fact, leaf) v_g labeled by the agents to whom g can be allocated. Of course, the transformation is feasible in polynomial time. Eventually, we further transform the decomposition by making it binary: For each vertex v with children $v_1, ..., v_n$, we can create a novel vertex $\bar{v}$ as a child of v and with its label, by subsequently appending under it all these children but v_1. Let $\mathtt{TD} = \langle T, \chi \rangle$ be the resulting tree decomposition, having the same width as $\langle T', \chi' \rangle$.

Example 7.12. *Figure 7.2 illustrates a width-2 tree decomposition* $\bar{\mathtt{TD}}_0$ *of* $G(\mathcal{A}_0)$*, by evidencing the vertices that are univocally associated with the goods in* $\{g_1, g_2, g_3, g_4\}$*. Moreover, note that the decomposition is defined over a binary tree.* $\lhd$

The input to our encoding is the allocation scenario $\mathcal{A}$, the agent $i \in N$, the natural number ℓ, and the tree decomposition $\mathtt{TD} = \langle T, \chi \rangle$. Note that, for the moment, we do not consider the size h. Then, we define the encoding ξ such that $\xi(\mathcal{A}, i, \ell, \mathtt{TD})$ is the CSP instance $\langle Var, U, \mathbf{C} \rangle$, where

- $Var = \bigcup_{j \in N} \{X_j, Y_j\} \cup \{X_j^v \mid v \text{ is in } T \wedge j \in \chi(v)\}$;
- $U = \{0, 1, \odot, \swarrow, \searrow, \uparrow\}$;

and where $\mathbf{C}$ is defined as follows, with constraint relations being reported in tabular form in Figure 7.3:

1. For each agent $j \in N$ and vertex v in T with $j \in \chi(v)$, there is a constraint $(S_{v,j}, r_{v,j})$ with $S_{v,j} = \{X_j^v, X_j\}$;
2. For each good g with $\mathtt{val}(g) \geq w_\ell$ and each $j \in \chi(v_g)$, there is a constraint $(S_{g,j}, r_{g,j})$ such that $S_{g,j} = \{X_j^{v_g}\}$;
3. For each agent $j \in N$, there is a constraint (S_j, r_j) such that $S_j = \{Y_j, X_j\}$;

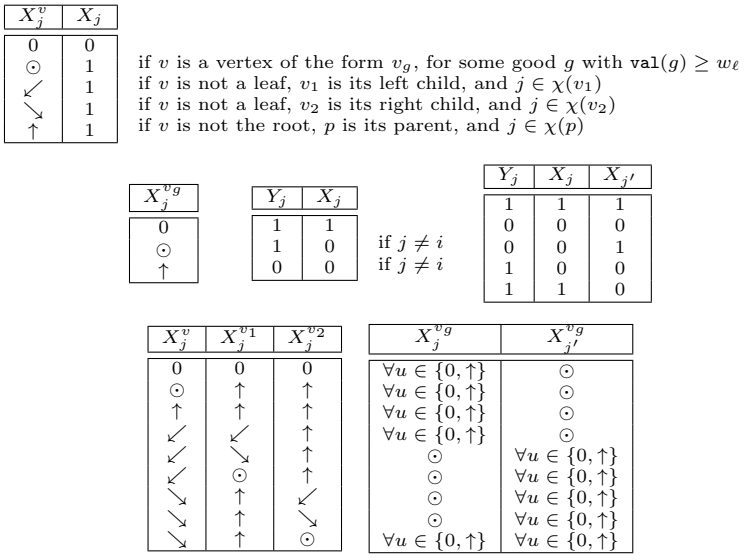


Figure 7.3: CSP encoding in Section 7.5.

4. For each pair of agents $j \in N$ and $j' \in N$ that are adjacent in G_ℓ, there is a constraint $(S_{j,j'}, r_{j,j'})$ such that $S_{j,j'} = \{Y_j, X_j, X_{j'}\}$;

5. For each non-leaf vertex v whose left (resp., right) child is v_1 (resp., v_2), and for each $j \in \chi(v)$, there is a constraint $(S'_{v,j}, r'_{v,j})$ such that $S'_{v,j} = \{X_j^v, X_j^{v_1}, X_j^{v_2}\}$;

6. For each good g with $\mathtt{val}(g) \geq w_\ell$ and for each pair $j, j' \in \chi(v_g)$ with $g \in \Omega(j) \cap \Omega(j')$, there is a constraint $(S_{g,j,j'}, r_{g,j,j'})$ such that $S_{g,j,j'} = \{X_j^{v_g}, X_{j'}^{v_g}\}$;

7. No further constraint is in $\mathbf{C}$.

Theorem 7.13. *The following properties hold:*

(a) $\xi(\mathcal{A}, i, \ell, \mathtt{TD})$ *can be built in polynomial time;*

(b) the domain of each variable in $\xi(\mathcal{A}, i, \ell, \mathtt{TD})$ *consists of at most 5 distinct elements;*

(c) $tw(G(\xi(\mathcal{A}, i, \ell, \mathtt{TD}))) \leq 5 \times (tw(G(\mathcal{A})) + 1)$*;*

(d) if θ *is a solution to* $\xi(\mathcal{A}, i, \ell, \mathtt{TD})$*, then* $R_\theta = \{j \mid \theta(Y_j) = 1 \wedge j \neq i\}$ *is such that* $v_{\mathcal{A}}(R_\theta \cup \{i\}) - v_{\mathcal{A}}(R_\theta) \geq w_\ell$*;*

(e) if $R \subseteq N \setminus \{i\}$ *is such that* $v_{\mathcal{A}}(R \cup \{i\}) - v_{\mathcal{A}}(R) \geq w_\ell$*, then there is a solution* θ *to* $\xi(\mathcal{A}, i, \ell, \mathtt{TD})$ *with* $R = R_\theta$*;*

(f) $\sum_{h=0}^{n-1} \#\mathrm{col}_\ell^i(\mathcal{G}_{\mathcal{A}}, h) = |\Theta(\xi(\mathcal{A}, i, \ell, \mathtt{TD}), \{Y_1, ..., Y_n\})|$.

Proof Sketch. Property *(a)* and Property *(b)* are immediate.

Concerning Property *(c)* note that, if $\mathtt{TD} = \langle T, \chi \rangle$, then the tuple $\langle T, \chi_\xi \rangle$ such that for each vertex v in T, $\chi_\xi(v) = \bigcup_{j \in \chi(v)} \{X_j, Y_j, X_j^v\} \cup \{X_j^{v_1}, X_j^{v_2} \mid v \text{ is not a leaf}\}$ is a tree decomposition of $G(\xi(\mathcal{A}, i, \ell, \mathtt{TD}))$. Note that the decomposition does not depend on i and ℓ. As an example, the modified label associated with the root node of the tree decomposition of the graph $G(\mathcal{A}_0)$ in Example 7.11 is shown in Figure 7.2.

Concerning Property *(d)*, assume that θ is a solution to $\xi(\mathcal{A}, i, \ell, \mathtt{TD})$ and let $\bar{R}_\theta$ be the set $\{j \mid \theta(X_j) = 1 \wedge j \neq i\}$.

Let j be any agent in $\bar{R}_\theta \cup \{i\}$. First, we claim that there is a vertex v^* in T such that $\theta(X_j^{v^*}) = \odot$. By contradiction, assume there is no such vertex. Let v be the vertex in $\mathtt{TD}$ that is the closest to the root with $j \in \chi(v)$. Because of the constraint $(S_{v,j}, r_{v,j})$ of type 1, we have that $\theta(X_j^v) \in \{\swarrow, \searrow\}$. If v is a non-leaf vertex, then because of constraint $(S'_{v,j}, r'_{v,j})$ of type 5, we have that $\theta(X_j^w) \in \{\swarrow, \searrow\}$ holds, with $w \in \{v_1, v_2\}$ being one of its two children in T. In particular, given the constraint of type 1, for the child w it must be the case that $j \in \chi(w)$ holds. Therefore, we can apply the argument again on w, and so top-down from w we can eventually reach a leaf $\bar{v}$ such that $\theta(\bar{v}) \in \{\swarrow, \searrow\}$. But, this is impossible by the constraint $(S_{\bar{v},j}, r_{\bar{v},j})$ of type 1. So, we know that for each $j \in \bar{R}_\theta \cup \{i\}$, there is a vertex v^* in T such that $\theta(X_j^{v^*}) = \odot$. Moreover, for each vertex w in the path connecting v and v^*, $\theta(w) \in \{\swarrow, \searrow\}$. Therefore, because of constraints of type 5, for each vertex u with $j \in \chi(u)$ and not occurring in this path, it is the case that $\theta(u) = \uparrow$.

Hence, for each $j \in \bar{R}_\theta \cup \{i\}$, there is precisely one vertex v^* such that $\theta(X_j^{v^*}) = \odot$.

Because of the constraints of type 1 and 2, it holds that $v^* = v_g$, for some good $g \in \Omega(j)$ with $\texttt{val}(g) \geq w_\ell$. Moreover, because of the constraints of type 6, there is no other agent j' such that $\theta(X_{j'}^{v^*})=\odot$. In the light of these properties, the function $\pi : N \to \mathbb{G}$ such that $\pi(j) = g$, for each $j \in \bar{R}_\theta \cup \{i\}$ with $\theta(X_j^{v_g})=\odot$, and $\pi(\bar{j}) = \emptyset$, for each other agent $\bar{j}$, is well-defined and is an allocation. In particular, for each agent $j \in \bar{R}_\theta \cup \{i\}$, $\texttt{val}(\pi(j)) \geq w_\ell$.

Let now $R_\theta = \{j \mid \theta(Y_j) = 1 \wedge j \neq i\}$. Observe that, because of the constraints of type 3, it holds that $\bar{R}_\theta \subseteq R_\theta$. In particular, $\theta(X_i) = 1$. Moreover, note that because of the constraints of type 4, whenever $\theta(X_{j'}) = 1$ and $\theta(Y_j) = 1$ with j and j' being adjacent in G_ℓ, then $\theta(X_j) = 1$ holds, too. Hence, $\bar{R}_\theta \cup \{i\} \supseteq \mathbf{p}_\ell^i(R_\theta)$. It follows that we can apply Theorem 7.7 on the coalition R_θ, and we conclude that $v_{\mathcal{A}}(R_\theta \cup \{i\}) - v_{\mathcal{A}}(R_\theta) \geq w_\ell$.

Consider now Property *(e)*. If $R \subseteq N \setminus \{i\}$ is a coalition with $v_{\mathcal{A}}(R \cup \{i\}) - v_{\mathcal{A}}(R) \geq w_\ell$, then by Theorem 7.7 there is an allocation π such that $\texttt{val}(\pi(j)) \geq w_\ell$, for each $j \in \mathbf{p}_\ell^i(R)$. Consider the substitution θ such that: $\theta(Y_j) = 1$ iff $j \in R \cup \{i\}$; $\theta(X_j) = 1$ iff $j \in \mathbf{p}_\ell^i(R)$; $\theta(X_j^{v_g}) = \odot$ iff $\pi(j) = g$; $\theta(X_j^v) = 0$ iff $j \notin R$; $\theta(X_j^v) = \uparrow$ iff $\theta(X_j^{v_g}) = \odot$ holds for a vertex v_g that is not in the subtree of T rooted at v; $\theta(X_j^v) = \swarrow$ (resp., $\theta(X_j^v) = \searrow$) if $\theta(X_j^{v_g}) = \odot$ for a vertex v_g that occurs in the subtree rooted at the left (resp., right) child of v. By inspecting the constraints, it can be checked that θ is in fact a solution to $\xi(\mathcal{A}, i, \ell, \texttt{TD})$.

Finally, Property *(f)* derives by Property *(d)*, by Property *(e)*, by the fact that $\theta(Y_i) = \theta(X_i) = 1$ holds in any solution, and by the definition of $\#\mathrm{col}_\ell^i(\mathcal{G}_{\mathcal{A}}, h)$. □

By combining Theorem 7.13 and Theorem 2.4, the following can be easilly established.

Corollary 7.14. *The Banzhaf value of allocation games $\mathcal{G}_{\mathcal{A}}$ can be computed in polynomial time on scenarios $\mathcal{A} = \langle N, G, \Omega, \mathtt{val}\rangle$ such that $tw(G(\mathcal{A}))$ is bounded by a constant.*

Proof. Recall that the Banzhaf value of $\mathcal{G}_{\mathcal{A}}$ can be written as $\beta_i(\mathcal{G}_{\mathcal{A}}) = \frac{1}{2^{n-1}} \sum_{h=0}^{n-1} \beta_i(\mathcal{G}_{\mathcal{A}}, h)$. Given the expressions for $\beta_i(\mathcal{G}_{\mathcal{A}}, h)$ in Theorem 7.3, to compute the Banzhaf value in polynomial time, we need a method to efficiently evaluate $\sum_{h=0}^{n-1} \#\mathrm{col}_\ell^i(\mathcal{G}_{\mathcal{A}}, h)$, for each agent i and level ℓ. By Property (f) in Theorem 7.13, this is possible on scenarios $\mathcal{A}$ where $tw(G(\mathcal{A}))$ is bounded by some constant k. Indeed, a width-k tree decomposition can be computed in linear time. □

7.6 From the Banzhaf value to the Shapley value

The encoding ξ discussed so far does not take h as a parameter. In fact, it just provides us a way to compute the value $\sum_{h=0}^{n-1} \#\mathrm{col}_\ell^i(\mathcal{G}_{\mathcal{A}}, h)$ and, hence, the Banzhaf value. In order to compute the contribution $\#\mathrm{col}_\ell^i(\mathcal{G}_{\mathcal{A}}, h)$ for each cardinality of the coalitions (and, hence, the Shapley value by Theorem 7.3), we need a way to filter, out of all possible solutions, those θ such that $|R_\theta| = h$. This is not immediate (by preserving the structural properties and the bound on the domains), so that a careful construction is in order.

Let $\mathcal{I}$ be the instance $\langle Var, U, \mathbf{C}\rangle$ with $\{Y_1, ..., Y_n\} \subseteq Var$, let $\mathtt{TD}' = \langle T', \chi'\rangle$ be a binary tree decomposition of $G(\mathcal{I})$, and let $h \in \{0, ..., n\text{-}1\}$. Then, consider the modified CSP instance $\zeta(\mathcal{I}, \{Y_1, ..., Y_n\}, h, \mathtt{TD}') = \langle Var', U', \mathbf{C}'\rangle$ such that:

- $Var' = Var \cup \{W_v \mid v \text{ is in } T'\}$;
- $U' = U \cup \{0, ..., h\}$;
- $\mathbf{C}' = \mathbf{C} \cup \{(S_v, r_v) \mid v \text{ is in } T'\}$.

In particular, for each non-leaf vertex v in T' with children v_1 and v_2, we have $S_v = \chi'(v) \cup \{W_v, W_{v_1}, W_{v_2}\}$. Moreover, r_v contains all possible substitutions θ over the variables in S_v such that $\theta(W_v), \theta(W_{v_1}), \theta(W_{v_2}) \in \{0, ..., h\}$ and $\theta(W_v) - |\{Y_j \in S_v \mid v = \mathtt{cr}(j) \wedge \theta(Y_j) = 1\}| = \theta(W_{v_1}) + \theta(W_{v_2})$, where $\mathtt{cr}(j)$ is the vertex v^* that is the closest to the root and such that $Y_j \in \chi'(v^*)$. Additionally, if v is the root of T', then we require that $\theta(W_v) = h + 1$ holds.

Instead, if v is a leaf, then $S_v = \chi'(v) \cup \{W_v\}$, and r_v contains all possible substitutions θ over S_v such that $\theta(W_v) = |\{Y_j \in S_v \mid v = \mathtt{cr}(j) \wedge \theta(Y_j) = 1\}|$.

Theorem 7.15. *Let k be a fixed natural number, and assume that $\mathtt{TD}'$ has treewidth k. Then, the following properties hold:*

(a) $\zeta(\mathcal{I}, \{Y_1, ..., Y_n\}, h, \mathtt{TD}')$ can be built in polynomial time;

(b) $tw(G(\zeta(\mathcal{I}, \{Y_1, ..., Y_n\}, h, \mathtt{TD}'))) \leq k + 3$;

(c) if θ_h is a solution to $\zeta(\mathcal{I}, \{Y_1, ..., Y_n\}, h, \mathtt{TD}')$, then there is exactly one solution θ to $\mathcal{I}$ with $|\{Y_j \mid \theta(Y_j) = 1\}| = h$ and $\theta_h \supseteq \theta$.

(d) if θ is a solution to $\mathcal{I}$ such that $|\{Y_j \mid \theta(Y_j) = 1\}| = h$, then there is exactly one solution θ_h to $\zeta(\mathcal{I}, \{Y_1, ..., Y_n\}, h, \mathtt{TD}')$ with $\theta_h \supseteq \theta$;

Proof Sketch. Property *(a)* and Property *(b)* are immediate, by inspection of the construction. For Property *(c)*, just observe that if θ_h is a solution to $\zeta(\mathcal{I}, \{Y_1, ..., Y_n\}, h, \mathtt{TD}')$, then $|\{Y_j \mid \theta_h(Y_j) = 1\}| = h$ and its restriction to the variables in Var, say θ, is a solution to $\mathcal{I}$. Finally, concerning Property *(d)*, note that if θ is a solution to $\mathcal{I}$ with $|\{Y_j \mid \theta(Y_j) = 1\}| = h$, then we can build a solution $\theta_h \supseteq \theta$ to $\zeta(\mathcal{I}, \{Y_1, ..., Y_n\}, h, \mathtt{TD})$ in a way that, for each variable $W_v \in (Var' \setminus Var)$, $\theta_h(W_v)$ is the number of variables Y_j such that $\theta(Y_j) = 1$ and $\mathtt{cr}(j)$ occurs in the subtree of T rooted at v. In particular, note that there are no further assignments for these variables leading to

a solution. □

By combining the above result with Theorem 7.13, the following can be established.

Theorem 7.16. *The Shapley value of allocation games can be computed in polynomial time on all allocation scenarios whose interaction graphs have bounded treewidth.*

Proof. Consider such a class of allocation scenarios $\mathcal{A}$ with $tw(G(\mathcal{A})) \leq k$, for some fixed natural number k. Then, a width-k tree decomposition $\mathtt{TD}$ of $G(\mathcal{A})$ and the encoding $\mathcal{I} = \xi(\mathcal{A}, i, \ell, \mathtt{TD})$ can be computed in polynomial time. Let $\mathtt{TD}'$ be a tree decomposition of $G(\mathcal{I})$ whose width is bounded by $5 \times (k+1)$ (cf. Theorem 7.13). For each $h \in \{0, ..., n\text{-}1\}$, let $\mathcal{I}' = \zeta(\mathcal{I}, \{Y_1, ..., Y_n\}, h+1, \mathtt{TD}')$. Because of Theorem 7.15 and again Theorem 7.13, we know that $|\Theta(\mathcal{I}', \{Y_1, ..., Y_n\})|$ coincides with the value $\#\mathrm{col}^i_\ell(\mathcal{G}_A, h)$. We then get the Shapley value by using Theorem 7.3, and Equation (7.1).

Unfortunately, tractability does not follow here by just using Theorem 2.4 on $\mathcal{I}'$ and $\{Y_1, ..., Y_n\}$, since the auxiliary variables of the form W_v do not have a bounded domain. However, because of Property *(c)* and Property *(d)*, we can add such variables to the output variables without altering the number of solutions, because $|\Theta(\mathcal{I}', \{Y_1, ..., Y_n\})| = |\Theta(\mathcal{I}', \{Y_1, ..., Y_n\} \cup (\mathit{Var}' \setminus \mathit{Var})|$ holds. Thus, we actually apply Theorem 2.4 on $\mathcal{I}'$ with output variables $\{Y_1, ..., Y_n\} \cup (\mathit{Var}' \setminus \mathit{Var})$, and we still conclude that $|\Theta(\mathcal{I}', \{Y_1, ..., Y_n\})|$ can be computed in polynomial time. □

7.7 Summary

We have studied the problem of computing the Shapley value (and the Banzhaf value) of allocation games, which are coalitional games implicitly (and succinctly) specified in terms of an underlying allocation scenario. We have shown that the

problem is #P-complete, even in stringent settings. Motivated by this bad news, we identified islands of tractability by focusing either on scenarios with sharing degree at most 2 or such that the interactions among agents have a tree-like structure. This way, real world applications with useful structural properties can efficiently be dealt with, even if many agents and goods are involved. Moreover, the technical tools used to get the results may have a wider spectrum of applicability, beyond allocation problems.

A variant of the proposed framework considers scenarios where agents must necessarily get some good. In this case, it makes sense to have goods with negative values, too. In fact, we remark that our algorithms can be extended to manage these cases as well, by just considering suitable negative levels. Our work leaves open the technical question of whether tractability still holds over scenarios with sharing degree bounded by some constant greater than 2 (e.g., $sd_\ell(\mathcal{A}) \geq 3$). Moreover, it might stimulate further research to analyze the complexity of other solution concepts over allocation games, such as the *nucleolus* [Schmeidler, 1969].

Part III

CSP and Mining

CHAPTER 8

Process Mining

Process mining aims to discover, monitor and improve real processes by extracting knowledge from the event logs that are made available by today's information systems van der Aalst [2011]. A prominent process mining task is *process discovery*, whose goal is to facilitate the (re-)design phase and the implementation of complex process models. Indeed, process discovery algorithms are devoted to automatically deriving a model that can explain all the episodes recorded in an event *log* related to the execution of the activities of an underlying process. An approach to accomplish this task is presented which can benefit from the background knowledge that, in many cases, is available to the analysts taking care of the process (re-)design. The approach is based on encoding the information gathered from the log and the (possibly) given background knowledge in terms of *precedence constraints*, i.e., of constraints over the topology of the resulting process models. Mining algorithms are eventually formulated in terms of reasoning problems over precedence constraints, and the computational complexity of such problems is thoroughly analyzed by tracing their tractability frontier. Solution algorithms are proposed and their properties analyzed. These algorithms have been implemented in a prototype system, and results of a thorough experimental activity are discussed.

8.1 An overview of process discovery

Process discovery has emerged as a powerful approach to support the analysis and the design of complex processes. It consists of analyzing a set of traces registering the sequence of tasks performed along several enactments of a transactional system, in order to build a process model that can explain all the episodes recorded over them. Eventually, the "mined" model can be used to design a detailed process model suited to be supported in a workflow management system, or to shed light on the actual process behavior for optimization purposes (for instance, by singling out deviations between the intended conceptual model and the behavior that is actually registered).

In abstract terms, discovery algorithms carry out two different sub-tasks. First, they analyze the *traces* registering the sequences of activity executions over each enactment of the process, by mining the causal dependencies that are likely to hold among them. In particular, they present these dependencies in form of *dependency graphs*, that is, directed graphs whose nodes one-to-one correspond with the activities and where, intuitively, edges ingoing into an activity a are meant to model that the possibility of executing a is conditioned by the execution of the activities from which these edges originate. Second, they enrich dependency graphs with advanced facets of process enactments (such as synchronization constructs, branching constructs, duplicate activities, and invisible activities, just to name a few) and return process models formalized in expressive modeling languages. These concepts are next exemplified.

Example 8.1. *Consider the set* $\mathcal{A} = \{a, b, c, d, e\}$ *of activities, and the structures depicted in Figure 8.1. We have three dependency graphs,* $\mathcal{G}_0$*,* $\mathcal{G}_1$*, and* $\mathcal{G}_2$*, which are meant to encode the causal relationships that hold over the activities in* $\mathcal{A}$*. Moreover,*

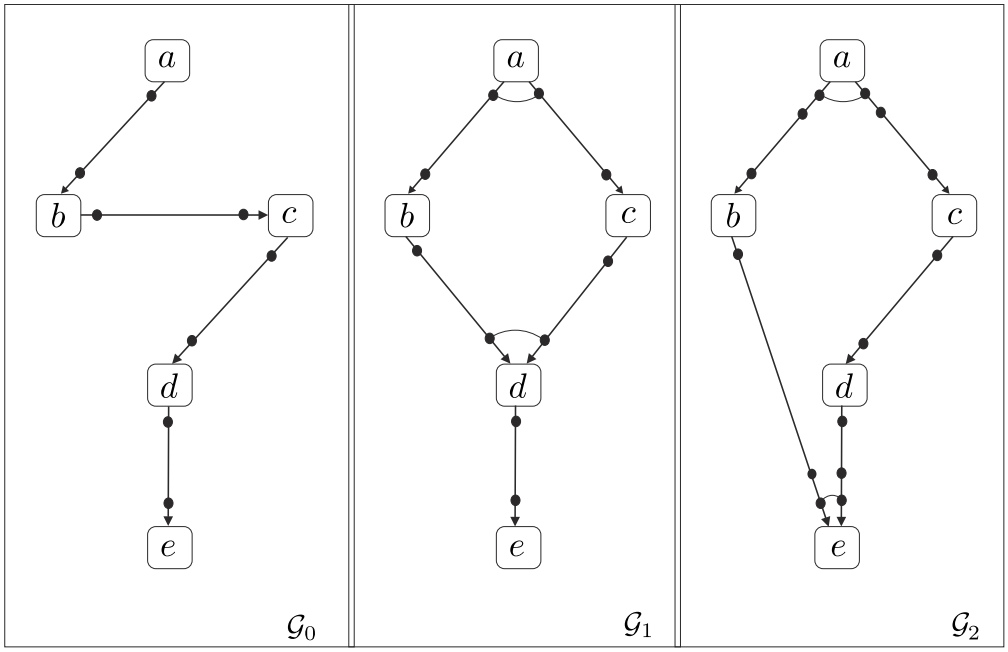


Figure 8.1: Process models in the running example.

these graphs are adorned with an intuitive notation that expresses routing constructs over them. For instance, in $\mathcal{G}_1$, the flow of execution is split after a over two branches that are to be executed in parallel, and which are eventually synchronized by the activity d—see Section 8.3 for the formalization of the notation and of its associated semantics.

Assume now that the two traces abcde and acbde over $\mathcal{A}$ are given as input to a process discovery algorithm. Then, the graph $\mathcal{G}_0$ will be hardly returned as output, as it does not model the flow associated with the trace acbde, where b is executed after c.

Instead, $\mathcal{G}_1$ and $\mathcal{G}_2$ are good candidates for a dependency graph supporting the two traces. The crucial observation here is that b is executed before c in one trace, while b is executed after c in the other. This likely witnesses that the two activities are not related by any causal dependency, and that they are executed in "parallel", i.e., over different branches of the process. Moreover, one might argue that the graph $\mathcal{G}_1$,

together with the associated routing constraints, is more appropriate than $\mathcal{G}_2$ for being returned as output. Indeed, there is heuristic evidence in the two given traces that b and d are not parallel, since b seems to be a pre-requisite for the execution of d. $\lhd$

Several process models can, in general, be associated with a log of traces given as input. Ideally, one might select the most appropriate one among them under the assumption that the log is *complete*, i.e., that it registers all the possible traces for the underlying process. Indeed, in this case, if an activity always precedes another, then we can safely discard those models where such activities can be executed over different branches. For instance, if we assume that the traces *abcde* and *acbde* of Example 8.1 are the only possible ones, then we can safely conclude that b and d are not parallel activities, hence discarding the graph $\mathcal{G}_2$ in Figure 8.1.

Log completeness has received considerable attention in the literature, and it is a crucial assumption under which a number of discovery methods can be proven to be correct (see, e.g., van der Aalst *et al.* [2004, 2002]). In fact, process discovery is often carried out via heuristics approaches, for which the quality of the resulting models grows with the fraction of the traces given at hand w.r.t. all possible traces for the process. It follows that the quality can be rather poor in those cases where logs are far from being complete due, for instance, to the following reasons:

Temporal bias: First, two parallel activities may always appear in the same relative order, simply because one of them always finishes after the other, due to a different duration of them or of some of their predecessor activities. For instance, even for a process that conforms with the schema $\mathcal{G}_2$, we may find no trace where d occurs before b, just because c is time consuming, so that b is always completed before d.

Combinatorial explosion: Second, log completeness might not hold just because the process has not been enacted a sufficient number of times. Indeed, the number

of possible traces grows exponentially w.r.t. the number of activities that can be executed in parallel. For instance, for a process with n branches each one involving m activities, we have more than $n!^m$ possible traces. Therefore, for real processes even with just 2 branches and about 20 activities, the number of combinations immediately leads to more than one billion enactments, which in many application domains are definitively more than the enactments actually registered.

Access restrictions: Finally, even when logs are in principle complete, analysts might have a restricted access to them due to a number of reasons, ranging from privacy reasons, to specific policies adopted within the enterprise, and to issues related to the technologic infrastructure. In particular, these latter issues often occur when the process is not (fully) automatized within a unique environment, so that logs are not immediately available and they need to be built by integrating the raw data produced by different information platforms (which can be difficult, time-consuming, or even just impossible if one of these platforms does not support logging mechanisms). In fact, it frequently happens in real-world applications that only a few representative traces are available (or can be reconstructed) for analysis purposes (for instance, related to a given time range or to some specific use cases of the overall process).

Because of the issues illustrated above, process discovery techniques are still at an early stage of adoption within enterprises. Indeed, analysts are likely to prefer traditional "top-down" design approaches, where models are eventually built by refining and formalizing a number of desiderata and specifications reflecting the prior knowledge they possess about the process to be automatized. For example, by a-priori knowing that b and d are parallel activities, the analyst can immediately discard $\mathcal{G}_1$ in Example 8.1, even though no trace is given where d actually occurred before b.

8.2 Bottom-up vs top-down design methods

Several process discovery approaches have already been proposed in the literature, differing in the mining capabilities, in their goals, in the adopted mining methods, and in the kinds of modeling language they support. For example, processes are often intuitively represented via pure directed graphs Agrawal et al. [1998]; Weijters and van der Aalst [2001]; Weijters and van der Aalst [2003]; Weijters *et al.* [2006]; Greco *et al.* [2006]; Chen and Yun [2003], while more expressive representations are used in other proposals, ranging from expression tree models Schimm [2003] and block-structured workflow models Herbst and Karagiannis [2000, 2003]; Hammori et al. [2006], to special classes of Petri nets van der Aalst and van Hee [2002]; van der Aalst *et al.* [2002, 2004]; de Medeiros et al. [2004]; Medeiros et al. [2007]. Moreover, moving from the observation that extracting a single process model may lead to over-generalized models mixing different usage scenarios, classical discovery algorithms have been often combined with methods for clustering log traces, so that a set of process models can be returned as output. This improves the precision of the underlying algorithms by capturing—from an abstract perspective—constructs that are beyond the expressiveness of the given modeling languages (cf. Greco *et al.* [2006]).

Despite the technical differences that emerge from the non-exhaustive list of proposals discussed above, it must be pointed out that all of them share the idea of mining process models by gathering statistics from the data and by processing them via heuristics. In particular, these "bottom-up" discovery techniques are not capable to take into account prior knowledge on the underlying process. As a result, over logs that are not complete, mined models may well violate conceptual specifications

and domain-constraints (recall that, in Example 8.1, $\mathcal{G}_1$ would be the most probable outcome of a discovery algorithm, but the analyst might a-priori know that b and d are parallel activities), hence turning out to be useless in real-life applications.

Recently, the need of defining process discovery algorithms that can take advantage of prior knowledge has been pointed out by [Goedertier et al., 2009]. In particular, they presented the AGNEs (Artificial Generation of Negative Events) technique, which is a mining algorithm founding on an Inductive Logic Programming (ILP) classification-oriented learner. The algorithm is articulated in four steps: first, temporal constraints are extracted from the input log, in order to capture local dependencies, non-local dependencies, and parallelism relationships. Second, the input log and the temporal constraints are used to generate negative examples, i.e., for each prefix of any trace, negative events are generated stating which activities are not allowed to be executed later on in that trace. Third, by using input log traces (as positive examples) together with the artificial negative events, a logic program is induced to predict whether any given activity is allowed to occur at a given position of a given sequence. Finally, the logic program is translated into a Petri net. An important peculiarity of AGNEs is that the first of the above four phases allows experts to define a "top-down" partial specification of the process, in terms of background knowledge that will be subsequently used to guide the mining algorithm. In particular, it is possible to state that two activities are parallel (resp., not parallel), and that one precedes/succeeds (resp., does not precede/succeed) the other.

Another process discovery method (partially) taking into account domain knowledge has been proposed by [van der Werf *et al.*, 2009]. As a reference model, again Petri nets are considered—moreover, the user can specify that the mined model must belong to some desired subclass of Petri nets (for instance, avoiding self loops or with

a fixed upper bound on the number of places/transitions). Starting with the most liberal (and overgeneralized) net for a given log, with as many transitions as the process activities and with no place, a more refined process model is obtained by iteratively adding a new place for restricting the allowed behavior. Each place is chosen greedily, by solving a system of integer linear inequalities, asking for a place with a minimal (resp., maximal) number of incoming edges (resp., outgoing edges). To curb the growth of the mined model, the search of the places is guided by the causal dependencies derived from the log (by using the metrics of [van der Aalst *et al.*, 2004]). Moreover, in its implementation in the *ProM* framework [van Dongen *et al.*, 2005], users are allowed to enforce finer grain constraints, by manually modifying the basic activity dependencies extracted from the log, prior to deriving a novel (refined) workflow model. By this way, the learning process can benefit from "top-down" specifications made available as domain knowledge expressed in terms of edge constraints and of constraints enforcing parallelism between pairs of activities.

8.3 Causal nets and logs

Several languages have been proposed in the literature which are tailored to the design and the analysis of processes, such as *Petri nets* [van der Aalst, 1998] or *event driven process chains* EPCs [van der Aalst *et al.*, 2002]. Given the focus of the thesis, it is convenient to adopt a language that is closer to the needs of process mining applications. Accordingly, our choice is to consider the language of *causal nets* [van der Aalst *et al.*, 2011], providing a favorable representational bias for such applications while having the same expressiveness as Petri nets.

Let us hereinafter assume that $\mathcal{A}$ is a given alphabet of symbols, univocally identifying the *activities* of some underlying process. The set $\mathcal{A}$ contains two distinguished

activities $a_\perp$ and $a_\top$, called the *starting* and the *terminating* activity, respectively.

A *dependency graph* (over $\mathcal{A}$) is a directed graph $\mathcal{G} = (V, E)$ whose nodes are the activities in the set $V \subseteq \mathcal{A}$, with $V \supseteq \{a_\perp, a_\top\}$, and whose edges in $E \subseteq V \times V$ encode the causal relationships that hold over them. In particular, for each activity $a \in V \setminus \{a_\perp, a_\top\}$, it must be the case that a occurs in some path from $a_\perp$ to $a_\top$. Moreover, $a_\perp$ and $a_\top$ have no ingoing and outgoing edges, respectively.

Example 8.2. *Consider again the three graphs $\mathcal{G}_0$, $\mathcal{G}_1$, and $\mathcal{G}_2$ depicted in Figure 8.1. It is immediate to check that they are dependency graphs over the set $\mathcal{A} = \{a, b, c, d, e\}$ of activities. In particular, a (resp., e) is the starting (resp., terminating) activity.* $\lhd$

A *causal net* (over $\mathcal{A}$) is a tuple $\mathcal{C} = \langle \mathcal{G}, \mathcal{I}, \mathcal{O} \rangle$ where $\mathcal{G} = (V, E)$ is a dependency graph and where $\mathcal{I}$ and $\mathcal{O}$ are two functions such that:

- $\mathcal{I}$ maps each activity $a \in V$ to the set $\mathcal{I}(a)$ of the *input bindings* for a. For any $a \in V \setminus \{a_\perp\}$, an input binding $ib \in \mathcal{I}(a)$ is a non-empty set of edges such that $ib \subseteq \{(x, a) \mid (x, a) \in E\}$, while the empty set $\emptyset$ is the only input binding for $a_\perp$, i.e., $\mathcal{I}(a_\perp) = \{\emptyset\}$. For each $a \in V$, $\bigcup_{ib \in \mathcal{I}(a)} ib = \{(x, a) \mid (x, a) \in E\}$ must hold.
- $\mathcal{O}$ maps each activity $a \in V$ to the set $\mathcal{O}(a)$ of the *output bindings* for a. For any $a \in V \setminus \{a_\top\}$, an output binding $ob \in \mathcal{O}(a)$ is a non-empty set of edges such that $ob \subseteq \{(a, y) \mid (a, y) \in E\}$, while the empty set $\emptyset$ is the only output binding for $a_\top$, i.e., $\mathcal{O}(a_\top) = \{\emptyset\}$. For each $a \in V$, $\bigcup_{ob \in \mathcal{O}(a)} ob = \{(a, y) \mid (a, y) \in E\}$ must hold.

Intuitively, input bindings are meant to encode the pre-conditions for the execution of an activity, while output bindings are meant to encode the effects of this execution.

Example 8.3. *Consider the causal net* $\mathcal{C}_0 = \langle \mathcal{G}_0, \mathcal{I}_0, \mathcal{O}_0 \rangle$, *where* $\mathcal{G}_0 = (\{a, b, c, d, e\}, E_0)$ *is the graph in Figure 8.1, and where* $\mathcal{I}_0(z) = \{ib_z\}$ *and* $\mathcal{O}_0(z) = \{ob_z\}$ *are such that* $ib_z = \{(x, z) \mid (x, z) \in E_0\}$ *and* $ob_z = \{(z, y) \mid (z, y) \in E_0\}$, *for each* $z \in \{a, b, c, d, e\}$. *The causal net models that the execution of* z *can start as soon as its predecessor activity in* $\mathcal{G}_0$ *is completed—of course, this is immaterial for the starting activity* a, *which is such that* $ib_a = \emptyset$. *In fact, after its execution and if* $z \neq e$, *we have that* z *enables the execution of its unique successor in* $\mathcal{G}_0$.

Similarly, consider the causal net $\mathcal{C}_1 = \langle \mathcal{G}_1, \mathcal{I}_1, \mathcal{O}_1 \rangle$, *where* $\mathcal{G}_1 = (\{a, b, c, d, e\}, E_1)$ *is the graph in Figure 8.1, and where* $\mathcal{I}_1(z) = \{ib'_z\}$ *and* $\mathcal{O}_1(z) = \{ob'_z\}$, *for each activity* z, *are such that:* $ib'_a = \emptyset$, $ib'_b = \{(a, b)\}$, $ib'_c = \{(a, c)\}$, $ib'_d = \{(b, d), (c, d)\}$, $ib'_e = \{(d, e)\}$, $ob'_a = \{(a, b), (a, c)\}$, $ob'_b = \{(b, d)\}$, $ob'_c = \{(c, d)\}$, $ob'_d = \{(d, e)\}$, *and* $ob'_e = \emptyset$. *Note that, after its execution, the activity* a *enables both* b *and* c *(in parallel), and that the flow is synchronized by* d, *which can be executed only once* b *and* c *are both completed.*

Note also that the symbols adorning the edges of the graphs in Figure 8.1 precisely correspond to the input and the output bindings. Formally, if $\langle \mathcal{G}, \mathcal{I}, \mathcal{O} \rangle$ *is a causal net, then each input binding* $ib \in \mathcal{I}(z)$ *(resp., output binding* $ob \in \mathcal{O}(z)$*) is represented by marking the edges in* ib *(resp., in* ob*) and by linking all such markings with a line. For instance, in the graph* $\mathcal{G}_1$, *the edges* (a, b) *and* (a, c) *are marked, and these markings are linked together, hence meaning that the output binding* $\{(a, b), (a, c)\}$ *occurs in* $\mathcal{O}(a)$.

For completeness, we point out that any given activity can be associated with more than one input/output binding, in general. As an example, consider the causal net $\mathcal{C}_2 = \langle \mathcal{G}_2, \mathcal{I}_2, \mathcal{O}_2 \rangle$ *associated with the graph* $\mathcal{G}_2$ *shown in Figure 8.1. Then, we may note, for instance, that* $\mathcal{O}_3(a) = \{\{(a, b)\}, \{(a, c)\}, \{(a, b), (a, c)\}\}$ *holds. Indeed, the*

edges (a, b) *and* (a, c) *are linked together as in the case of* $\mathcal{G}_1$*, but we additionally have the singleton markings now. Thus,* a *can either activate* b*, or* c*, or even both of them.* $\lhd$

Formally, the semantics of causal nets is next given in terms of which instantiations are "globally valid". This is different from the operational semantics of design-oriented modeling languages, such as the token-game semantics of Petri nets.

A *binding activity* for the causal net $\mathcal{C}$ is a tuple $\langle a, ib, ob\rangle$, where $a \in V$ is an activity and such that $ib \in \mathcal{I}(a)$ and $ob \in \mathcal{O}(a)$ hold. A sequence σ of binding activities $\langle a_1 = a_\perp, ib_1, ob_1\rangle, ..., \langle a_n = a_\top, ib_n, ob_n\rangle$ is called a *binding sequence*. The *state* S_j^σ of $\mathcal{C}$ at the j-th step of σ is defined inductively as the multi-set[1] of edges such that $S_0^\sigma = \emptyset$, and $S_j^\sigma = S_{j-1}^\sigma \cup ob_j \setminus ib_j$, for each $j \in \{1, ..., n\}$. The sequence σ is *valid* w.r.t. $\mathcal{C}$ if $S_n^\sigma = \emptyset$ and $ib_j \subseteq S_{j-1}^\sigma$, for each $j \in \{1, ..., n\}$.

Example 8.4. *Consider the net* $\mathcal{C}_1$ *discussed in Example 8.3, and the binding sequence* $\sigma = \sigma_1\sigma_2...\sigma_5$ *such that:* $\sigma_1 = \langle a, \{\}, \{(a, b), (a, c)\}\rangle$, $\sigma_2 = \langle b, \{(a, b)\}, \{(b, d)\}\rangle$, $\sigma_3 = \langle c, \{(a, c)\}, \{(c, d)\}\rangle$, $\sigma_4 = \langle d, \{(b, d), (c, d)\}, \{(d, e)\}\rangle$, *and* $\sigma_5 = \langle e, \{(d, e)\}, \{\}\rangle$. *Note that* $S_0^\sigma = \emptyset$, $S_1^\sigma = \{(a, b), (a, c)\}$, $S_2^\sigma = \{(a, c), (b, d)\}$, $S_3^\sigma = \{(b, d), (c, d)\}$, $S_4^\sigma = \{(d, e)\}$, *and* $S_5^\sigma = \emptyset$. *In fact,* σ *is valid w.r.t.* $\mathcal{C}_1$. $\lhd$

Transactional systems store partial information about binding sequences, by tracing the events related to the execution of the various activities. Formally, a *trace* t (over $\mathcal{A}$) has the form $t[1]t[2]...t[n]$, with $t[i] \in \mathcal{A}$ being an activity, for each $i \in \{1, ..., n\}$, and with n being the *length* of t. W.l.o.g., we shall hereinafter assume that $t[1] = a_\perp$, $t[n] = a_\top$, and that for each $i \in \{2, ..., n-1\}$, $t[i] \cap \{a_\perp, a_\top\} = \emptyset$. Indeed, we may possibly view $a_\perp$ and $a_\top$ as two virtual activities, which we add to t in order to satisfy this requirement. The length of t is also denoted as $len(t)$. A

[1] Hereinafter, set operations are transparently applied to multi-sets with the usual intended meaning.

multi-set L of traces is hereinafter just called a *log*, and the set of all the activities occurring over the traces in L is denoted by $\mathcal{A}(L)$.

Note that, as commonly done in the literature, we are considering here an abstract view of a log, by focusing on what order the various activities were executed (specifically, completed) and by getting rid of all information about *(i)* timings (e.g., starting times and durations) and about *(ii)* the data involved in them.

Concerning the first assumption, note that our process modeling language is not capable to support temporal information. In fact, very few mining approaches have been proposed that are able to discover timed models, like stochastic Petri nets (see, e.g, Anastasiou et al. [2011]; Hu et al. [2011]). Therefore, in our setting, information about timings in the log can be helpful to a limited extent only. For instance, we can immediately conclude that two activities are parallel if one starts after the other, but before the completion of it. Moreover, for two activities a and b that are not parallel, in order to assess how likely a is a pre-requisite for the execution of b, in addition of their relative positions (see Section 9.6), we can consider the time elapsed between the completion of a and the starting of b. In the paper, however, we will not expand on these standard heuristics (conceived for models that do not support temporal information), by referring the interested reader, e.g., to the work by [Wen *et al.*, 2009]. Here, we stress instead that it is an interesting avenue for further research to extend the features of causal nets by directly incorporating timing information and to explore how our techniques can be modified as to deal with the resulting model.

Concerning the second assumption, we point out that even the adaptation of basic process discovery algorithms to *multi-dimensional* settings, where modeling languages have again to be extended (in this latter case in order to deal with data about activity executions), has been only partially explored in the literature (see,

e.g., Greco *et al.* [2007]). In fact, this is still largely an open research issue, so that exporting our results to such richer settings is outside the scope of the paper, while constituting another interesting avenue for further research.

Now that we have clarified the assumptions underlying our setting, we can proceed to formalize when a log can be considered as the result of the enactments of a given process model. To this end, we say that a causal net $\mathcal{C}$ *supports* a trace t if there is a binding sequence σ that is valid w.r.t. $\mathcal{C}$ and where the j-th binding activity $\langle a_j, ib_j, ob_j \rangle$ of σ, for each $j \in \{1, ..., len(t)\}$, is such that $a_j = t[j]$. Moreover, we say that the causal net $\mathcal{C}$ *supports* a log L, denoted by $\mathcal{C} \vdash L$, if $\mathcal{C}$ supports each trace $t \in L$.

Example 8.5. *The causal net $\mathcal{C}_1$ discussed in Example 8.3 supports the trace abcde, as it is witnessed by the binding sequence σ illustrated in Example 8.4. Moreover, it can be checked that $\mathcal{C}_1$ also supports the trace acbde, and that no further trace is supported.*

Consider instead the causal net $\mathcal{C}_2$, again discussed in Example 8.3. Recall that a can activate either b, or c, or both of them. Moreover, by looking at the dependency graph $\mathcal{G}_2$ depicted in Figure 8.1, observe that b and d are parallel activities. Thus, the traces that $\mathcal{C}_2$ supports are abe, acde, abcde, acbde, and acdbe. ◁

We leave the section by pointing out that a useful extension of causal nets consists in allowing input and output bindings to be multi-sets, rather than just sets. With this extended model, for instance, an activity x can activate two different instances of an activity y, for which (x, y) is an edge in the underlying dependency graph.[2] A causal net enriched with this capability will be hereinafter called an *extended causal*

[2] Alternatively, we might think that input and output bindings are sets as usual, but that x activates two *hidden activities*, say h_1 and h_2, which both activate y in their turn. While hidden activities play a role in the enactments, they are not registered in the log. In fact, in process discovery applications, hidden activities are frequently used to enrich the basic expressivity of the process modeling languages.

net.

8.4 Dependency graphs and process mining: basic results

In process mining, a log L is given and the goal is to derive a process model supporting its traces. We next show that the main task for achieving this goal is essentially the discovery of the underlying dependency graph. In fact, based on this property, we can contextually show that the whole semantics of causal nets can be recast in simple graph-theoretic terms, which is convenient for our subsequent elaborations.

Definition 8.6. *Let L be a log. A dependency graph $\mathcal{G}$* acyclically supports *L, denoted by $\mathcal{G} \vdash_a L$, if for each trace $t \in L$, there is a subgraph $\mathcal{G}_t$ of $\mathcal{G}$ such that:*

- *$\mathcal{G}_t$ is an acyclic dependency graph over $\mathcal{A}(\{t\})$, and*
- *t is a* topologic sort *of $\mathcal{G}_t$, i.e., for each edge $(t[i], t[j])$ in $\mathcal{G}_t$, we have that $i < j$.*

□

Example 8.7. *Consider again the graph $\mathcal{G}_2$ in Figure 8.1, and note that $\mathcal{G}_2 \vdash_a \{abe, acde, abcde, acbde, acdbe\}$. For instance, given the trace abe, the subgraph of $\mathcal{G}_2$ induced over the activities $\{a, b, e\}$ is an acyclic dependency graph and abe is a topologic sort of it. Moreover, for any log L including a trace not in $\{abe, acde, abcde, acbde, acdbe\}$, we can check that $\mathcal{G}_2 \vdash_a L$ does not hold.* ◁

At this point, it is interesting to observe that the set of traces supported by the causal net $\mathcal{C}_2$ (see Example 8.5) precisely coincides with the set of traces acyclically supported by the dependency graph $\mathcal{G}_2$ on top of which $\mathcal{C}_2$ was built. This is not by chance, and is intimately related to the fact that none of the given traces contains repetitions of the same activity. We next formalize this property.

First, we recall that a log L is *linear* if there is no trace in L containing repetitions

of the same activity, i.e., for each $t \in L$ and for each $i, j \in \{1, ..., len(t)\}$ with $i \neq j$, $t[i] \neq t[j]$ holds. Then, we derive the following result.

Theorem 8.8. *Let L be a linear log, and let $\mathcal{G}$ be a dependency graph. Then, $\mathcal{G} \vdash_a L$ if, and only if, there is a causal net $\mathcal{C} = \langle \mathcal{G}, \mathcal{I}, \mathcal{O} \rangle$ such that $\mathcal{C} \vdash L$.*

Proof. (if part). Let $\mathcal{C} = \langle \mathcal{G}, \mathcal{I}, \mathcal{O} \rangle$ be a causal net such that $\mathcal{C} \vdash L$. Let t be any trace in L. Let $\mathcal{G}_t = (V_t, E_t)$ be the subgraph of $\mathcal{G} = (V, E)$ such that $V_t = \{t[1], ..., t[len(t)]\}$ and $E_t = \{(t[i], t[j]) \in E \mid 1 \leq i < j \leq len(t)\}$. Hence, $\mathcal{G}_t$ is the subgraph of $\mathcal{G}$ induced over the activities occurring in t, where we keep those edges of E that conform with the ordering of the activities in t. Since L is linear, $\mathcal{G}_t$ is acyclic and t is a topologic sort of it. According to Definition 8.6, to conclude, it is then sufficient to prove that $\mathcal{G}_t$ is a dependency graph. In fact, $a_\perp$ and $a_\top$ have no ingoing and outgoing edges, respectively, by construction of $\mathcal{G}_t$. The final requirement to be checked is now that, for each activity $a \in V \setminus \{a_\perp, a_\top\}$, it must be the case that a occurs in some path from $a_\perp$ to $a_\top$. In particular, since $\mathcal{G}_t$ is acyclic, we can equivalently check that each activity $a \in V \setminus \{a_\top\}$ (resp., $a \in V \setminus \{a_\perp\}$) has at least one outgoing (resp., ingoing) edge.

Let σ be a valid binding sequence such that $\sigma_j = \langle t[j], ib_j, ob_j \rangle$ is the j-th binding activity of σ, for each $j \in \{1, ..., len(t)\}$. Note that σ exists, since $\mathcal{C} \vdash L$. Consider first an activity $a \in V \setminus \{a_\top\}$, i.e, $a = t[j]$ where $j \in \{1, ..., len(t) - 1\}$. As σ is a binding sequence, $ob_j \in \mathcal{O}(t[j])$ holds and hence we have that $ob_j \neq \emptyset$. In particular, there is an edge $(t[j], y) \in ob_j$ such that $(t[j], y) \in S^\sigma_j = S^\sigma_{j-1} \cup ob_j \setminus ib_j$. Moreover, again because σ is valid, $S^\sigma_{len(t)} = \emptyset$ holds, and hence there is an index $i \in \{j + 1, ..., len(t)\}$ such that $(t[j], y) \in ib_i$. It follows that $y = t[i]$ and thus $(t[j], t[i])$ belongs to E (hence, to E_t since $j < i$). So, each activity $a \in V \setminus \{a_\top\}$ has at least one outgoing edge.

We conclude by claiming that, for each $j \in \{2, ..., len(t)\}$, there is an index $i \in \{1, ..., j-1\}$ such that $(t[i], t[j])$ is in E (hence, in E_t). In order to prove the claim, let again σ be a valid binding sequence such that $\sigma_j = \langle t[j], ib_j, ob_j \rangle$ is the j-th binding activity of σ, for each $j \in \{1, ..., len(t)\}$. By definition of binding sequence, we know that $S^\sigma_j \subseteq S^\sigma_{j-1} \cup ob_j$ holds, for each $j \in \{1, ..., len(t)\}$. In particular, since σ is valid, $S^\sigma_0 = \emptyset$ holds, and hence $S^\sigma_j \subseteq \bigcup_{h=1}^{j} ob_h$, for each $j \in \{1, ..., len(t)\}$. Recall now from the definition of causal net that $ob_j \in \mathcal{O}(t[j])$ implies $ob_j \subseteq \{(t[j], y) \mid (t[j], y) \in E\}$. Therefore, we conclude that:

$$S^\sigma_j \subseteq \bigcup_{h=1}^{j} \{(t[h], y) \mid (t[h], y) \in E\}, \text{for each } j \in \{1, ..., len(t)\}. \tag{8.1}$$

Let us exploit again the fact that σ is valid, in order to derive that $ib_j \subseteq S^\sigma_{j-1}$ holds, for each $j \in \{1, ..., len(t)\}$. Let now j be an index in the set $\{2, ..., len(t)\}$. Observe that, by definition of causal net, $ib_j \in \mathcal{I}(t[j])$ implies that $ib_j \subseteq \{(x, t[j]) \mid (x, t[j]) \in E\}$, with ib_j being, in particular, non empty. Therefore, by looking again at Equation 8.1 above and recalling that $ib_j \subseteq S^\sigma_{j-1}$, we can eventually conclude that an index $i \in \{1, ..., j-1\}$ exists such that $(t[i], t[j])$ occurs in E.

(only-if part). Assume that, for each trace $t \in L$, there is a subgraph $\mathcal{G}_t$ of $\mathcal{G} = (V, E)$ such that $\mathcal{G}_t$ is an acyclic dependency graph and t is a topologic sort of $\mathcal{G}_t$. For each trace t in L and for each index $j \in \{1, ..., len(t)\}$, define the input binding $ib_{t,j} = \{(x, t[j]) \in E \mid x \in \{t[1], ..., t[j-1]\}\}$ and the output binding $ob_{t,j} = \{(t[j], y) \in E \mid y \in \{t[j+1], ..., t[len(t)]\}\}$. Note that, for each $j \in \{2, ..., len(t)\}$, $ib_{t,j} \neq \emptyset$ holds. Indeed, as $\mathcal{G}_t$ is a dependency graph, for each $j \in \{2, ..., len(t)\}$, $t[j]$ has at least one ingoing edge $(x, t[j])$ in $\mathcal{G}_t$. Moreover, since t is a topologic sort of $\mathcal{G}_t$, we actually have that $x \in \{t[1], ..., t[j-1]\}$. Similarly, it can be seen that, for each $j \in \{1, ..., len(t) - 1\}$, $ob_{t,j} \neq \emptyset$ holds. Finally, it is immediate to check that

$ib_{t,1} = ob_{t,len(t)} = \emptyset$.

Now, consider the functions $\mathcal{I}$ and $\mathcal{O}$ such that, for each $a \in V$,

- $\mathcal{I}(a) = \bigcup_{t \in L, j | t[j]=a} ib_{t,j} \cup I_a$, where $I_a = \{(x, a) \mid (x, a) \in E\}$;
- $\mathcal{O}(a) = \bigcup_{t \in L, j | t[j]=a} ob_{t,j} \cup O_a$, where $O_a = \{(a, y) \mid (a, y) \in E\}$.

Since $\mathcal{G}$ is a dependency graph and given the definition of I_a and O_a, for each $a \in V$, it is immediate to check that $\mathcal{C} = \langle \mathcal{G}, \mathcal{I}, \mathcal{O} \rangle$ is a causal net. It remains to show that $\mathcal{C} \vdash L$, i.e., that for each trace t in L, there is a valid binding sequence σ such that $\langle t[j], ib_j, ob_j \rangle$ is the j-th binding activity of σ, for each $j \in \{1, ..., len(t)\}$. Let $t[1]...t[n]$ be a trace, and consider the sequence $\sigma = \langle t[1], ib_{t,1}, ob_{t,1} \rangle, ..., \langle t[n], ib_{t,n}, ob_{t,n} \rangle$. We have to show that $S^\sigma_n = \emptyset$ and $ib_{t,j} \subseteq S^\sigma_{j-1}$, for each $j \in \{1, ..., n\}$.

To prove the result, we first claim that, for each $j \in \{0, 1, ..., n\}$, $S^\sigma_j = \bigcup_{i=1}^{j} \{(t[i], t[i']) \mid (t[i], t[i']) \in E, i' > j\}$. In fact, the base case holds because $S^\sigma_0 = \emptyset$, by definition of the state of a causal net. Now, assume that the property holds up to the index $h < j$. We have to show that $S^\sigma_{h+1} = \bigcup_{i=1}^{h+1} \{(t[i], t[i']) \in E \mid i' > h + 1\}$. To this end, recall that $S^\sigma_{h+1} = S^\sigma_h \cup ob_{t,h+1} \setminus ib_{t,h+1}$, so that $S^\sigma_{h+1} = \bigcup_{i=1}^{h} \{(t[i], t[i']) \in E \mid i' > h\} \cup ob_{t,h+1} \setminus ib_{t,h+1}$ holds, by inductive hypothesis. Eventually, the result derives by the above expression and the fact that $ob_{t,h+1} = \{(t[h+1], y) \in E \mid y \in \{t[h+2], ..., t[n]\}\}$ and $ib_{t,h+1} = \{(x, t[h + 1]) \in E \mid x \in \{t[1], ..., t[h]\}\}$.

Armed with the above property, we can now resume the proof. First, we have to show that $S^\sigma_n = \emptyset$. In fact, we have that $S^\sigma_n = \bigcup_{i=1}^{n} \{(t[i], t[i']) \in E \mid i' > n\}$, which coincides with the empty set as n is the length of t. Second, we have to show that $ib_{t,j} \subseteq S^\sigma_{j-1}$, for each $j \in \{1, ..., n\}$. To this end, recall that $ib_{t,j} = \{(x, t[j]) \in E \mid x \in \{t[1], ..., t[j - 1]\}\}$, and eventually just check that $ib_{t,j} \subseteq \bigcup_{i=1}^{j-1} \{(t[i], t[i']) \in E \mid i' > j - 1\} = S^\sigma_{j-1}$. □

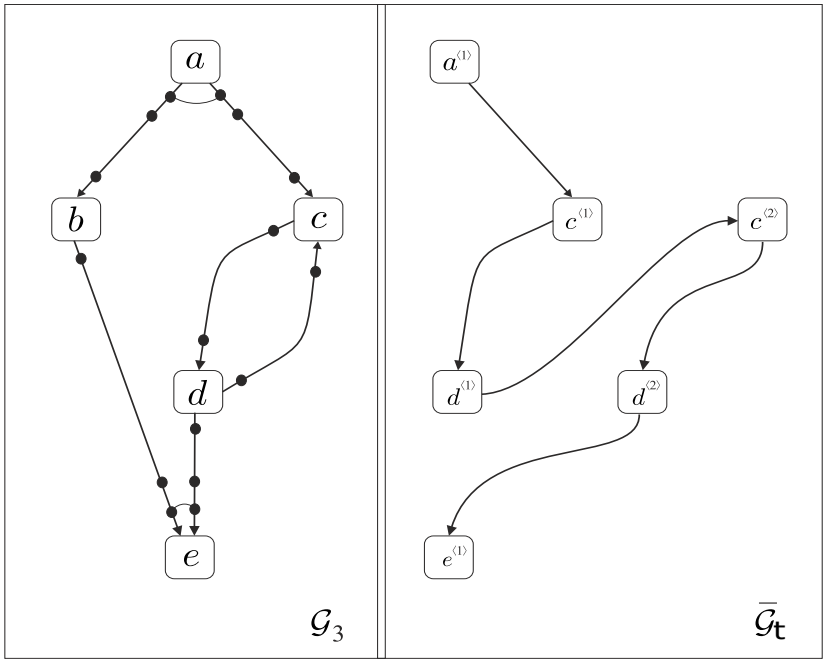


Figure 8.2: A process model involving a cycle, with an example unfolding.

To deal with arbitrary logs, we next introduce a mechanism for virtually unfolding cycles. For each trace t, let $\mathit{unfold}(t)$ denote the trace obtained from t by substituting the i-th occurrence in t of any activity a with the fresh (virtual) activity $a^{\langle i\rangle}$. Moreover, let $\mathit{unfold}(\mathcal{A})$ denote the (infinite) set of all virtual activities that can be built based on $\mathcal{A}$. The starting and terminating activity in $\mathit{unfold}(\mathcal{A})$ are $a_{\perp}^{\langle 1\rangle}$ and $a_{\top}^{\langle 1\rangle}$, respectively.

If L is a log, then we define $\mathit{unfold}(L)$ as the linear log $\{\mathit{unfold}(t) \mid t \in L\}$. Moreover, if $\bar{\mathcal{G}} = (\bar{V}, \bar{E})$ is a graph where $\bar{V} \subseteq \mathit{unfold}(\mathcal{A})$, then we define the *folding* of $\bar{\mathcal{G}}$ as the directed graph $\mathit{fold}(\bar{\mathcal{G}}) = (V, E)$ such that $V = \{x \in \mathcal{A} \mid \exists x^{\langle i\rangle} \in \mathit{unfold}(\mathcal{A}) \text{ s.t. } x^{\langle i\rangle} \in \bar{V}\}$ and $E = \{(x, y) \in \mathcal{A} \times \mathcal{A} \mid \exists x^{\langle i\rangle}, y^{\langle j\rangle} \in \mathit{unfold}(\mathcal{A}) \text{ s.t. } (x^{\langle i\rangle}, y^{\langle j\rangle}) \in \bar{E}\}$.

Definition 8.9. *Let L be a log. A dependency graph $\mathcal{G}$* supports *L, denoted by $\mathcal{G} \vdash L$,*

if for each trace $t \in L$, there is a graph $\bar{\mathcal{G}}_t$ such that $fold(\bar{\mathcal{G}}_t)$ is a subgraph of $\mathcal{G}$, $\bar{\mathcal{G}}_t \vdash_a \{unfold(t)\}$, and the following two conditions hold:

(1) there is no pair of edges $(x^{\langle i\rangle}, y^{\langle j\rangle})$, $(x^{\langle i\rangle}, y^{\langle j'\rangle})$ in $\bar{\mathcal{G}}_t$ such that $j \neq j'$, and

(2) there is no pair of edges $(x^{\langle i\rangle}, y^{\langle j\rangle})$, $(x^{\langle i'\rangle}, y^{\langle j\rangle})$ in $\bar{\mathcal{G}}_t$ such that $i \neq i'$. □

Example 8.10. *Consider the log consisting of the trace $t = acdcde$, and note that $unfold(acdcde) = a^{\langle 1\rangle}c^{\langle 1\rangle}d^{\langle 1\rangle}c^{\langle 2\rangle}d^{\langle 2\rangle}e^{\langle 1\rangle}$. Moreover, check that the graph $\bar{\mathcal{G}}_t$ depicted on the right part of Figure 8.2 is such that $\bar{\mathcal{G}}_t \vdash_a \{unfold(acdcde)\}$. In fact, $fold(\bar{\mathcal{G}})$ is a subgraph of the graph $\mathcal{G}_3$ depicted in the left part of the figure, and it is easily seen that conditions (1) and (2) of Definition 8.9 hold on $\bar{\mathcal{G}}_t$. Hence, $\mathcal{G}_3 \vdash \{acdcde\}$ holds.*

Consider now the trace $t' = abbe$ and $unfold(abbe) = a^{\langle 1\rangle}b^{\langle 1\rangle}b^{\langle 2\rangle}e^{\langle 1\rangle}$. Let $\bar{\mathcal{G}}_{t'}$ be the graph consisting of the edges $(a^{\langle 1\rangle}, b^{\langle 1\rangle})$, $(a^{\langle 1\rangle}, b^{\langle 2\rangle})$, $(b^{\langle 1\rangle}, e^{\langle 1\rangle})$, and $(b^{\langle 2\rangle}, e^{\langle 1\rangle})$. Then, $\bar{\mathcal{G}}_{t'} \vdash_a \{unfold(abbe)\}$, and $fold(\bar{\mathcal{G}}_{t'})$ is a subgraph of $\mathcal{G}_3$. However, $\bar{\mathcal{G}}_{t'}$ violates conditions (1) and (2) in Definition 8.9. Therefore, $\mathcal{G}_3$ does not support $\{abbe\}$. ◁

Note that whenever the log L is linear, the above definition reduces to Definition 8.6 and, in particular, conditions (1) and (2) are immaterial. More formally, in this case, $\mathcal{G} \vdash L$ holds if, and only if, $\mathcal{G} \vdash_a L$ holds. Hence, for linear logs, we are in the position of applying Theorem 8.8 with '$\vdash$' in place of '$\vdash_a$'. More generally, the following result is established in order to relate the notion of support over dependency graphs with the notion of support over causal nets. The result is of interest in its own, as it allows to restate the semantics of causal nets in pure graph-theoretic terms. In fact, it will play an important role in our subsequent elaborations.

Theorem 8.11. *Let L be a log, and let $\mathcal{G}$ be a dependency graph. Then, $\mathcal{G} \vdash L$ if, and only if, there is a causal net $\mathcal{C} = \langle \mathcal{G}, \mathcal{I}, \mathcal{O}\rangle$ such that $\mathcal{C} \vdash L$.*

Proof. (if part). Let $\mathcal{C} = \langle \mathcal{G}, \mathcal{I}, \mathcal{O} \rangle$ be a causal net such that $\mathcal{C} \vdash L$, with $\mathcal{G} = (N, E)$. This means that, for each trace t in L, there is a binding sequence σ such that σ is valid w.r.t. $\mathcal{C}$, and the j-th element of σ has the form $\langle t[j], ib_j, ob_j \rangle$, for each $j \in \{1, ..., len(t)\}$. Let S_j^σ denote the state of $\mathcal{C}$ at the j-th step of σ, and consider the sequence $\bar{\sigma}$ having the same length $len(t)$ as σ and whose j-th element $\langle \bar{a}_j, \bar{ib}_j, \bar{ob}_j \rangle$, for each $j \in \{1, ..., len(t)\}$, is obtained from $\langle t[j], ib_j, ob_j \rangle$ as follows:

- $\bar{a}_j$ is the virtual activity $t[j]^{\langle k \rangle}$, where k is the number of occurrences of the activity $t[j]$ in $t[1]...t[j]$, i.e., the number of binding activities defined over $t[j]$ in the first j elements of σ;
- For each output binding $(t[j], y) \in ob_j$, let α_y denote the number of binding activities occurring up to the j-th step of σ where $(t[j], y)$ occurs as an element of the output binding. Note that, since σ is a valid sequence w.r.t. $\mathcal{C}$ (and since bindings are sets, i.e., multiple occurrences are not allowed), we are guaranteed about the existence of α_y binding activities where these output bindings are consumed, i.e., where they are taken as input. Let $next(j, y)$ be the index of the α_y-th activity binding of this kind, hence, in particular with $(t[j], y) \in ib_{next(j,y)}$, and note that $next(j, y) > j$. Then, we define $\bar{ob}_j = \{(\bar{a}_j, \bar{a}_{next(j,y)}) \mid (t[j], y) \in ob_j\}$.
- For each input binding $(x, t[j]) \in ib_j$, let β_y denote the number of binding activities occurring up to the j-th step of σ, where $(x, t[j])$ is taken as input. Note that, since σ is a valid sequence w.r.t. $\mathcal{C}$, we are guaranteed about the existence of β_y binding activities where these input bindings are produced. Let $prev(j, x)$ be the index of the β_y-th activity binding of this kind, hence, in particular with $(x, t[j]) \in ob_{prev(j,x)}$, and note that $prev(j, x) < j$. Then, we define $\bar{ib}_j = \{(\bar{a}_{prev(j,x)}, \bar{a}_j) \mid (x, t[j]) \in ib_j\}$.

Define now $\bar{\mathcal{G}}_t = (\bar{V}_t, \bar{E}_t)$ as the graph where $\bar{V}_t = \mathcal{A}(\{unfold(t)\})$ and where $\bar{E}_t = \bigcup_{j=1}^{len(t)}(i\bar{b}_j \cup o\bar{b}_j)$. We claim that $\bar{\mathcal{G}}_t$ is a dependency graph. Indeed, note first that $\bar{\mathcal{G}}_t$ is acyclic because each edge in $\bar{E}_t$ has the form $(\bar{a}_i, \bar{a}_j)$, with $i < j$. In particular, $t[1]^{\langle 1 \rangle}$ and $t[len(t)]^{\langle 1 \rangle}$ play the role of the starting and the terminating activity, respectively. Then, consider an activity $\bar{a}_j \in \bar{V}_t \setminus \{t[1]^{\langle 1 \rangle}\}$ (resp., $\bar{a}_j \in \bar{V}_t \setminus \{t[len(t)]^{\langle 1 \rangle}\}$). Note that $\bar{a}_j$ has at least one ingoing (resp., outgoing) edge in $\bar{E}_t$, because there is at least an edge of the form $(x, t[j])$ (resp., $(t[j], y)$) in ib_j (resp., ob_j), by the fact that $\mathcal{C} = \langle \mathcal{G}, \mathcal{I}, \mathcal{O} \rangle$ is a causal net (hence, $ib_j \neq \emptyset$ and $ob_j \neq \emptyset$ hold) and that σ is a binding sequence. Since $\bar{\mathcal{G}}_t$ is acyclic, the above properties entail that $\bar{a}_j$ occurs in a path from $t[1]^{\langle 1 \rangle}$ to $t[len(t)]^{\langle 1 \rangle}$.

Consider now the functions $\bar{\mathcal{I}}_t$ and $\bar{\mathcal{O}}_t$ such that $\bar{\mathcal{I}}_t(\bar{a}_j) = \{i\bar{b}_j\}$ and $\bar{\mathcal{O}}_t(\bar{a}_j) = \{o\bar{b}_j\}$, for each $\bar{a}_j \in \bar{V}_t$. Since $\bar{\mathcal{G}}_t$ is a dependency graph and given the construction of its edges, we derive that $\langle \bar{\mathcal{G}}_t, \bar{\mathcal{I}}_t, \bar{\mathcal{O}}_t \rangle$ is a causal net (over $\mathcal{A}(\{unfold(t)\})$). Moreover, we claim that $\bar{\sigma}$ is valid w.r.t. $\langle \bar{\mathcal{G}}_t, \bar{\mathcal{I}}_t, \bar{\mathcal{O}}_t \rangle$. To prove the claim, note first that, given the above construction for $\bar{\sigma}$, if $i = prev(j, x)$ (resp., $i = next(j, y)$), then $x = t[i]$ (resp., $y = t[j]$) and $j = next(i, t[j])$ (resp., $j = prev(i, t[j])$). Then, consider the state $S_j^{\bar{\sigma}}$ of $\langle \bar{\mathcal{G}}_t, \bar{\mathcal{I}}_t, \bar{\mathcal{O}}_t \rangle$ at the j-th step of $\bar{\sigma}$, and observe that the following properties hold.

- For each $j \in \{1, ..., len(t)\}$, $i\bar{b}_j \subseteq S_{j-1}^{\bar{\sigma}}$. Indeed, consider an element $(\bar{a}_{prev(j,x)}, \bar{a}_j)$ in $i\bar{b}_j$, let $i = prev(j, x)$, and recall that $(\bar{a}_{prev(j,x)}, \bar{a}_j) = (\bar{a}_i, \bar{a}_{next(i,t[i])})$. Therefore, $(\bar{a}_{prev(j,x)}, \bar{a}_j)$ occurs in $o\bar{b}_i$ and, hence, in $S_i^{\bar{\sigma}} = S_{i-1}^{\bar{\sigma}} \cup o\bar{b}_i \setminus i\bar{b}_i$. Eventually, as $\bar{a}_j$ occurs only at the j-th step of $\bar{\sigma}$ and $i < j$, we have that $(\bar{a}_{prev(j,x)}, \bar{a}_j) \in S_{j-1}^{\bar{\sigma}}$.
- $S_{len(t)}^{\bar{\sigma}} = \emptyset$. Indeed, assume by contradiction that $(\bar{x}, \bar{y})$ occurs in $S_{len}^{\bar{\sigma}}(t)$. Then, there is an index j such that $\bar{x} = \bar{a}_j$ and $(\bar{a}_j, \bar{y})$ occurs in $o\bar{b}_j$. By construction, we therefore have that $(\bar{x}, \bar{y}) = (\bar{a}_j, \bar{a}_{next(j,y)})$. However, by letting $i = next(j, y)$, we can write that $(\bar{a}_j, \bar{a}_{next(j,y)}) = (\bar{a}_{prev(i,t[j])}, \bar{a}_i) \in i\bar{b}_i$. Since $i > j$, it follows

that $(\bar{x}, \bar{y}) \notin S_i^{\bar{\sigma}}$, as $\bar{x}$ occurs only at the j-th step of $\bar{\sigma}$. We conclude that $(\bar{x}, \bar{y}) \notin S_{len(t)}^{\bar{\sigma}}$. Contradiction.

By putting together the above results, we have so far shown that $\langle \bar{\mathcal{G}}_t, \bar{\mathcal{I}}_t, \bar{\mathcal{O}}_t \rangle$ is a causal net with $\langle \bar{\mathcal{G}}_t, \bar{\mathcal{I}}_t, \bar{\mathcal{O}}_t \rangle \vdash \{unfold(t)\}$, for each t in L. It follows that we can apply Theorem 8.8 on $\langle \bar{\mathcal{G}}_t, \bar{\mathcal{I}}_t, \bar{\mathcal{O}}_t \rangle$, and we derive that $\bar{\mathcal{G}}_t \vdash_a \{unfold(t)\}$ holds, for each t in L.

Finally, consider the graph $\bar{\mathcal{G}}_t$ and note that conditions (1) and (2) in Definition 8.9 are trivially satisfied, by construction of its edges. Indeed, all edges outgoing from $\bar{a}_j$, with $j \in \{1, ..., len(t)\}$, have the form $(\bar{a}_j, \bar{a}_{next(j,y)})$, where $(t[j], y) \in ob_j$. In particular, $(\bar{a}_j, \bar{a}_{next(j,y)})$ is univocally determined by y, so that $(\bar{a}_j, \bar{a}_{next(j,y)}) \neq (\bar{a}_j, \bar{a}_{next(j,y')})$ implies that $y \neq y'$ and hence $\bar{a}_{next(j,y)}$ and $\bar{a}_{next(j,y')}$ are virtual activities built from different true activities. A similar line of reasoning applies to the edges incoming into $\bar{a}_j$. Moreover, $fold(\bar{\mathcal{G}}_t)$ is a subgraph of $\mathcal{G}$. Indeed, $\bar{E}_t$ is the union of all bindings in $\langle \bar{\mathcal{G}}_t, \bar{\mathcal{I}}_t, \bar{\mathcal{O}}_t \rangle$, and any element in these bindings is of the form $(x^{\langle i \rangle}, y^{\langle j \rangle})$, where (x, y) occurs in a binding of $\mathcal{I}$ or $\mathcal{O}$ and, hence, is an edge in E. Therefore, according to Definition 8.9, we have shown that $\mathcal{G} \vdash L$ holds.

(only-if part). Assume that $\mathcal{G} \vdash L$ holds, with $\mathcal{G} = (V, E)$. Thus, for each trace $t \in L$, there is a graph $\bar{\mathcal{G}}_t = (\bar{V}_t, \bar{E}_t)$ such that $fold(\bar{\mathcal{G}}_t)$ is a subgraph of $\mathcal{G}$, $\bar{\mathcal{G}}_t \vdash_a \{unfold(t)\}$, and the following two conditions hold:

(1) there is no pair of edges $(x^{\langle i \rangle}, y^{\langle j \rangle})$, $(x^{\langle i \rangle}, y^{\langle j' \rangle})$ in $\bar{\mathcal{G}}_t$ such that $j \neq j'$, and

(2) there is no pair of edges $(x^{\langle i \rangle}, y^{\langle j \rangle})$, $(x^{\langle i' \rangle}, y^{\langle j \rangle})$ in $\bar{\mathcal{G}}_t$ such that $i \neq i'$.

By Theorem 8.8 applied on the fact that $\bar{\mathcal{G}}_t \vdash_a \{unfold(t)\}$ holds, we derive that there is a causal net $\bar{\mathcal{C}}_t = \langle \bar{\mathcal{G}}_t, \bar{\mathcal{I}}_t, \bar{\mathcal{O}}_t \rangle$ such that $\bar{\mathcal{C}}_t \vdash \{unfold(t)\}$ holds. This means that there is binding sequence $\bar{\sigma}$ that is valid w.r.t. $\bar{\mathcal{C}}$ and where the j-th binding

activity $\langle \bar{a}_j, \bar{ib}_j, \bar{ob}_j \rangle$ of $\bar{\sigma}$, for each $j \in \{1, ..., len(t)\}$, is such that $\bar{a}_j$ is the symbol $t[j]^{\langle k \rangle}$, with k being the number of occurrences of $t[j]$ in $t[1]...t[j]$.

Let $\mathcal{I}_t$ (resp., $\mathcal{O}_t$) be the function such that, for each node z in $fold(\bar{\mathcal{G}}_t)$, $\mathcal{I}_t(z) = \{ib_z \mid \text{ exists } j \text{ s.t. } \bar{ib}_z \in \bar{\mathcal{I}}_t(z^{\langle j \rangle})\}$ (resp., $\mathcal{O}_t(z) = \{ob_z \mid \text{ exists } i \text{ s.t. } \bar{ob}_z \in \bar{\mathcal{O}}_t(z^{\langle i \rangle})\}$), where ib_z (resp., ob_z) is the binding obtained from $\bar{ib}_z$ (resp., $\bar{ob}_z$) by stripping off the instantiation numbers of the virtual symbols. Similarly, define σ as the sequence obtained from $\bar{\sigma}$ by stripping off the instantiation numbers of the virtual symbols. Note that because of the conditions (1) and (2) above, and since bindings are defined over the edges of $\bar{\mathcal{G}}_t$, $|\bar{ib}_z| = |ib_z|$ and $|\bar{ob}_z| = |ob_z|$ hold, for each activity z. Therefore, if $S^{\bar{\sigma}}_j$ is the state of $\bar{\mathcal{C}}_t$ at the j-th step of $\bar{\sigma}$, then the state S^{σ}_j of $\mathcal{C}_t$ at the j-th step of σ can be obtained from $S^{\bar{\sigma}}_j$ by just stripping off the instantiation numbers of the symbols it contains. Hence, since $\bar{\sigma}$ is valid w.r.t. $\bar{\mathcal{C}}_t$, we can conclude that σ is valid w.r.t. $\mathcal{C}_t = \langle fold(\bar{\mathcal{G}}_t), \mathcal{I}_t, \mathcal{O}_t \rangle$. This witnesses that $\mathcal{C}_t \vdash \{t\}$ holds.

Let now $\bar{\mathcal{G}} = (\bar{V}, \bar{E})$ be the graph such that $\bar{V} = \bigcup_{t \in L} \bar{V}_t$ and $\bar{E} = \bigcup_{t \in L} \bar{E}_t$, and let $\mathcal{I}$ and $\mathcal{O}$ be the functions such that $\mathcal{I}(z) = \bigcup_{t \in L} \mathcal{I}_t(z)$ and $\mathcal{O}(z) = \bigcup_{t \in L} \mathcal{O}_t(z)$, for each z in $fold(\bar{\mathcal{G}})$. Note that $fold(\bar{\mathcal{G}})$ is a subgraph of $\mathcal{G}$, and that $\mathcal{C} = \langle fold(\bar{\mathcal{G}}), \mathcal{I}, \mathcal{O} \rangle$ is a causal net such that $\mathcal{C} \vdash L$. Therefore, in the case where $fold(\bar{\mathcal{G}}) = \mathcal{G}$, then we have derived that there is a causal net $\mathcal{C} = \langle \mathcal{G}, \mathcal{I}, \mathcal{O} \rangle$ such that $\mathcal{C} \vdash L$. To conclude the proof, the only remaining case to be analyzed is when $fold(\bar{\mathcal{G}}) = (V_f, E_f)$ is a proper subgraph of $\mathcal{G}$. In this case, consider the functions $\mathcal{I}'$ and $\mathcal{O}'$ such that, for each $z \in V \cap V_f$, $\mathcal{I}'(z) = \mathcal{I}(z) \cup I_z$ and $\mathcal{O}'(z) = \mathcal{O}'(z) \cup O_z$, and for each $z \in V \setminus V_f$, $I'(z) = I_z$ and $\mathcal{O}'(Z) = O_z$, where $I_z = \{\{(x, z)\} \mid (x, z) \in E\}$ and $O_z = \{\{(z, y)\} \mid (z, y) \in E\}$. As $\mathcal{G}$ is a dependency graph, by construction of $\mathcal{I}'$ and $\mathcal{O}'$, we trivially have that $\langle \mathcal{G}, \mathcal{I}', \mathcal{O}' \rangle$ is a causal net, which generalizes the behavior of $\mathcal{C} = \langle fold(\bar{\mathcal{G}}), \mathcal{I}, \mathcal{O} \rangle$ over the nodes and the edges in $\mathcal{G}$ that do not occur in $fold(\bar{\mathcal{G}})$.

Since $\mathcal{C} \vdash L$, we conclude that $\langle \mathcal{G}, \mathcal{I}', \mathcal{O}' \rangle \vdash L$. Indeed, if σ is a sequence valid w.r.t. $\mathcal{C}$, then it is also valid w.r.t.$\langle \mathcal{G}, \mathcal{I}', \mathcal{O}' \rangle$. □

Example 8.12. *Consider the causal net* $\mathcal{C}_3 = \langle \mathcal{G}_3, \mathcal{I}_3, \mathcal{O}_3 \rangle$, *where* $\mathcal{G}_3$ *is the dependency graph shown in Figure 8.2, and where* $\mathcal{I}_3$ *and* $\mathcal{O}_3$ *are the functions such that:* $\mathcal{I}_3(a) = \{\emptyset\}$, $\mathcal{I}_3(b) = \{\{(a,b)\}\}$, $\mathcal{I}_3(c) = \{\{(b,c)\}, \{d,c\}\}$, $\mathcal{I}_3(d) = \{\{(c,d)\}\}$, $\mathcal{I}_3(e) = \{\{(b,e)\}, \{(c,e)\}, \{(b,e),(c,e)\}\}$, $\mathcal{O}_3(a) = \{\{(a,b),(a,c)\}, \{(a,b\}, \{(a,c)\}\}$, $\mathcal{O}_3(b) = \{\{(b,d)\}\}$, $\mathcal{O}_3(c) = \{\{(c,d)\}\}$, $\mathcal{O}_3(d) = \{\{(d,e)\}\}$, *and* $\mathcal{O}_3(e) = \{\emptyset\}$. *Note that* $\mathcal{C}_3 \vdash \{acdcde\}$ *holds. Hence, by the above result, we can conclude that* $\mathcal{G}_3 \vdash \{acdcde\}$ *also holds, as we have in fact already observed in Example 8.10.* ◁

Note that if conditions (1) and (2) in Definition 8.9 are not guaranteed to hold, then a weaker variant of Theorem 8.11 can be still established.

Theorem 8.13. *Let* L *be a log, and let* $\mathcal{G}$ *be a dependency graph such that for each trace* $t \in L$, *there is a graph* $\bar{\mathcal{G}}_t$ *such that* $fold(\bar{\mathcal{G}}_t)$ *is a subgraph of* $\mathcal{G}$ *and* $\bar{\mathcal{G}}_t \vdash_a \{unfold(t)\}$. *Then, there is a possibly extended causal net* $\mathcal{C} = \langle \mathcal{G}, \mathcal{I}, \mathcal{O} \rangle$ *such that* $\mathcal{C} \vdash L$.

Proof. In the case where conditions (1) and (2) in Definition 8.9 are not guaranteed to hold, by inspecting the proof of the only-if part of Theorem 8.11, it can be checked that the structure $\mathcal{C}$ built there over $\mathcal{G}$ is still such that $\mathcal{C} \vdash L$ holds. The only difference is that $\mathcal{C}$ might be an extended causal net. □

CHAPTER 9

Precedence Constraints: Mining Problems and Complexity

9.1 Introduction

In this chapter, we propose and analyze a framework to specify additional properties on the process models that can be produced as output by process discovery algorithms. In particular, we formalize the concept of precedence constraints, discuss their application to the problem of mining causal nets, and analyze their complexity.

9.2 Syntax and Semantics

A *precedence constraint* is an assertion aimed at expressing a relationship of precedence among some of the activities in the underlying set $\mathcal{A}$. The language of precedence constraints is next defined in order to smoothly allow the formalization of the kinds of prior knowledge that are usually available to the analyst, such as, parallelism, locality, or exclusivity of activities (cf. Goedertier et al. [2009]).

Definition 9.1. *A* positive precedence *constraint π over $\mathcal{A}$ is either*

- *an expression of the form $S \to T$, called* edge *constraint, or*
- *an expression of the form $S \rightsquigarrow T$, called* path *constraint,*

where $S, T \subseteq \mathcal{A}$, with $|S| \geq 1$ and $|T| \geq 1$, are non-empty sets of activities.

For a positive constraint π, $\neg(\pi)$ is a negative *precedence constraint.* □

Precedence constraints are interpreted over directed graphs as follows.

Definition 9.2. *Let $\mathcal{G} = (V, E)$ be a directed graph such that $V \subseteq \mathcal{A}$. Then,*

(1) $\mathcal{G}$ satisfies an edge constraint $S \to T$, if there is an edge $(x, y) \in E$ with $x \in S$ and $y \in T$;

(2) $\mathcal{G}$ satisfies a path constraint $S \rightsquigarrow T$, if there is a sequence $x = a_0, a_1, ..., a_n = y$, with $n > 0$, such that $x \in S$, $y \in T$ and $(a_i, a_{i+1}) \in E$, for each $i \in \{0, ..., n-1\}$;

(3) $\mathcal{G}$ satisfies $\neg(\pi)$, if $\mathcal{G}$ does not satisfy π.

If $\mathcal{G}$ satisfies each constraint in a set Π of precedence constraints, then $\mathcal{G}$ is a model *of Π, denoted by $\mathcal{G} \models \Pi$. The set of all activities occurring in the constraints in Π is hereinafter denoted by $\mathcal{A}(\Pi)$.* □

A foundational task in process mining consists in automatically building a model that can explain the behavior registered in all the traces of some log L given as input. In this context, precedence constraints can formalize additional requirements that the model discovered from L has to satisfy. This gives rise to the following "Causal Net Mining" (short: CN-Mining) problem.

CN-Mining: Given a set $\mathcal{A}$ of activities, a log L with $\mathcal{A}(L) \subseteq \mathcal{A}$, and a set Π of precedence constraints with $\mathcal{A}(\Pi) \subseteq \mathcal{A}$, compute a possibly extended causal net $\mathcal{C} = \langle \mathcal{G}, \mathcal{I}, \mathcal{O} \rangle$ over $\mathcal{A}$ such that $\mathcal{C} \vdash L$ and $\mathcal{G} \models \Pi$, or check that no net with these properties exists.

Note that, for $\Pi = \emptyset$, the problem above reduces to the standard one considered in the literature. Moreover, observe that we are not considering (for the moment) quality measures for the resulting net. This issue will be explored later on. Finally, note that extended causal nets are allowed as solutions to the problem. In fact,

throughout the chapter, we shall explicitly discuss and show how our results extend to the more stringent setting where the focus is on (standard) causal nets.

Example 9.3. *Consider the set* $\Pi = \{ \neg(\{b\} \rightsquigarrow \{d\}), \neg(\{d\} \rightsquigarrow \{b\})\}$ *of precedence constraints. We have two negative path constraints, stating that* b *and* d *must be executed over "parallel" branches of the given process.*

Consider then the traces $abcde$ *and* $acbde$ *within the setting of Example 8.1, plus the causal nets* $\mathcal{C}_1$ *and* $\mathcal{C}_2$ *discussed in Example 8.3 and depicted in Figure 8.1. Without additional constraints, we have already noticed that* $\mathcal{C}_1$ *and* $\mathcal{C}_2$ *are such that* $\mathcal{C}_1 \vdash \{abcde, acbde\}$ *and* $\mathcal{C}_2 \vdash \{abcde, acbde\}$. *However, the dependency graph* $\mathcal{G}_2$ *(associated with* $\mathcal{C}_2$*) is a model of* Π*, while* $\mathcal{G}_1$ *is not, as it violates the constraint* $\neg(\{b\} \rightsquigarrow \{d\})$. *Thus,* $\mathcal{C}_2$ *is a solution to* CN-Mining *on input* $\{abcde, acbde\}$ *and* Π. $\lhd$

As a further remark, note that a solution to the mining problem might involve activities that do not occur in the log and in the constraints, i.e., in $\mathcal{A} \setminus \mathcal{A}(L) \setminus \mathcal{A}(\Pi)$. As an example, consider the set $\{\neg(\{a\} \rightarrow \{b\}), \{a\} \rightsquigarrow \{b\}\}$ of constraints prescribing the existence of a path from a to b, but forbidding the existence of a direct connection. If the log provided as input is empty, then any solution has to be defined over some "fresh" activities (at least one) ensuring the existence of a path from a to b of this kind. However, according to the formulation of CN-Mining, any "fresh" activity (in $\mathcal{A} \setminus \mathcal{A}(L) \setminus \mathcal{A}(\Pi)$) has to be explicitly provided as input to the mining problem, too. Therefore, one might wonder whether scenarios exist where all solutions need exponentially many fresh activities (w.r.t. $|\mathcal{A}(L) \cup \mathcal{A}(\Pi)|$), so that explicitly listing all of them would artificially blow up the size of the input. We next show that this is not possible, since a "small" solution always exists if the problem admits any solution. Hence, there is no loss of generality, but rather we gain

flexibility, by taking as input the set $\mathcal{A}$ possibly implicitly, i.e., by just specifying the upper bound on the number of activities that is defined in the statement of the result below.

Theorem 9.4. *Let L be a log, let Π be a set of precedence constraints, and let $\mathcal{C} = \langle \mathcal{G}, \mathcal{I}, \mathcal{O} \rangle$ be a causal net (resp., an extended causal net) with $\mathcal{C} \vdash L$ and $\mathcal{G} \models \Pi$. Then, a causal net (resp., an extended causal net) $\mathcal{C}' = \langle \mathcal{G}', \mathcal{I}', \mathcal{O}' \rangle$ exists such that $\mathcal{C}' \vdash L$, $\mathcal{G}' \models \Pi$, and $|V'| \leq |\mathcal{A}(L) \cup \mathcal{A}(\Pi)|^2 + |\mathcal{A}(L) \cup \mathcal{A}(\Pi)|$, with V' being the set of nodes of $\mathcal{G}'$. Moreover, if $\mathcal{G}$ is acyclic, then $\mathcal{G}'$ is acyclic, too.*

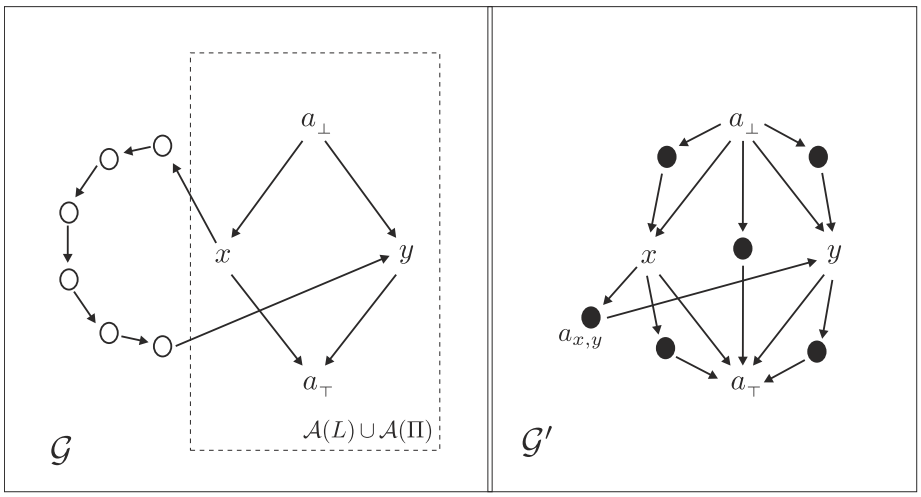


Figure 9.1: Example construction in the proof of Theorem 9.4.

Proof. Let $\mathcal{C} = \langle \mathcal{G}, \mathcal{I}, \mathcal{O} \rangle$, with $\mathcal{G} = (V, E)$, be a (resp., an extended) causal net such that $\mathcal{C} \vdash L$ and $\mathcal{G} \models \Pi$. Consider the graph $\mathcal{G}' = (V', E'_1 \cup E'_2)$ built as follows. The set V' consists of all the activities in $\mathcal{A}(L) \cup \mathcal{A}(\Pi)$ plus a fresh activity $a_{x,y}$ for each pair of activities $x, y \in \mathcal{A}(L) \cup \mathcal{A}(\Pi)$ such that there is a path in $\mathcal{G}$ from x to y. The set E'_1 consists of all the edges in E defined over the activities in $\mathcal{A}(L) \cup \mathcal{A}(\Pi)$, i.e., $E'_1 = \{(x, y) \in E \mid \{x, y\} \subseteq \mathcal{A}(L) \cup \mathcal{A}(\Pi)\}$. The set E'_2 contains the edges $(x, a_{x,y})$ and $(a_{x,y}, y)$, for each fresh activity $a_{x,y} \in V'$ (hence, $a_{x,y} \notin V$), and no further edge is in E'_2.

As an example, consider the graph $\mathcal{G}$ reported on the left of Figure 9.1. The graph is defined over the activities in $\mathcal{A}(L) \cup \mathcal{A}(\Pi) = \{a_\perp, a_\top, x, y\}$ plus 6 additional activities, which are depicted as circles. The graph $\mathcal{G}'$ that is built based on $\mathcal{G}$ is illustrated on the right part of the same figure. All nodes in $\mathcal{G}'$ that do not occur in $\mathcal{G}$ are depicted as black circles. In particular, observe that the node $a_{x,y}$ is responsible for preserving the connectivity that is supported in $\mathcal{G}$ by the nodes not occurring in $\mathcal{A}(L) \cup \mathcal{A}(\Pi)$.

Let us now analyze the properties of $\mathcal{G}'$. First, note that the activities $a_\perp$ and $a_\top$ are in V', as in fact they occur in $\mathcal{A}(L)$. Moreover, since $\mathcal{G}$ is a dependency graph, no edge ingoing into $a_\perp$ (resp., outgoing from $a_\top$) occurs in E'_2. Hence, given the construction of the edges in E'_1, we conclude that $a_\perp$ and $a_\top$ have no ingoing and outgoing edges in $\mathcal{G}'$.

Now, we claim that for each pair of activities $x, y \in \mathcal{A}(L) \cup \mathcal{A}(\Pi)$, there is a path from x to y in $\mathcal{G}$ if, and only if, there is a path from x to y in $\mathcal{G}'$. Indeed, if there is a path from x to y in $\mathcal{G}$, then the edges $(x, a_{x,y})$ and $(a_{x,y}, y)$ occur in E'_2. Conversely, assume that there is a path π from x to y in $\mathcal{G}'$, and for the sake of contradiction that there is no path from x to y in $\mathcal{G}$. As all edges of E defined over the activities in $\mathcal{A}(L) \cup \mathcal{A}(\Pi)$ are in E'_1, it must be the case that two edges occur in E'_2 having the form $(\bar{x}, a_{\bar{x},\bar{y}})$ and $(a_{\bar{x},\bar{y}}, \bar{y})$ and such that there is no path from $\bar{x}$ to $\bar{y}$ in $\mathcal{G}$. However, this is impossible given the construction of the edges in E'_2.

In the light of the above property, it follows that if $\mathcal{G}$ is acyclic, then $\mathcal{G}'$ is acyclic, too. Indeed, just notice that any cycle in $\mathcal{G}'$ must necessarily include a node in $\mathcal{A}(L) \cup \mathcal{A}(\Pi)$. Moreover, we can conclude that each activity $a \in \mathcal{A}(L) \cup \mathcal{A}(\Pi) \setminus \{a_\perp, a_\top\}$ is in a path in $\mathcal{G}'$ from $a_\perp$ to $a_\top$. Consider then an activity of the form $a_{x,y}$, which occurs in $V' \setminus V$. Since x (resp., y) either coincides with $a_\perp$ (resp., $a_\top$)

or it is reachable from $a_\perp$ (can reach $a_\top$) in $\mathcal{G}'$, because this property holds in fact on $\mathcal{G}$, we also conclude that $a_{x,y}$ is in a path in $\mathcal{G}'$ from $a_\perp$ to $a_\top$. By putting the above observations together, it follows that $\mathcal{G}'$ is a dependency graph over the set V' of activities. Moreover, $|V'| \leq |\mathcal{A}(L) \cup \mathcal{A}(\Pi)|^2 + |\mathcal{A}(L) \cup \mathcal{A}(\Pi)|$.

Recall now that $\mathcal{G}'$ preserves all the edges defined over the activities in $\mathcal{A}(L)$, and that $\mathcal{C} = \langle \mathcal{G}, \mathcal{I}, \mathcal{O} \rangle$ is a causal net (resp., extended causal net) such that $\mathcal{C} \vdash L$. It follows that there is a (resp., an extended) causal net $\mathcal{C}' = \langle \mathcal{G}', \mathcal{I}', \mathcal{O}' \rangle$ such that $\mathcal{C}' \vdash L$, where $\mathcal{I}'$ and $\mathcal{O}'$ just extend $\mathcal{I}$ and $\mathcal{O}$ as to include, for each given activity, a binding defined over the whole set of its ingoing and outgoing edges, respectively. Note that $\mathcal{C}'$ supports all the traces in L, even without using such fresh bindings. In fact, the construction of $\mathcal{I}'$ and $\mathcal{O}'$ is just required to ensure that $\mathcal{C}'$ is formally a (possibly extended) causal net.

In order to conclude, we have then to show that $\mathcal{G}' \models \Pi$. To this end, note that edge constraints and negated edge constraints are satisfied by $\mathcal{G}'$, because they are satisfied by $\mathcal{G}$ and since the two graphs coincide over the activities in $\mathcal{A}(\Pi)$. Eventually, recall that, for each pair of activities $x, y \in \mathcal{A}(L) \cup \mathcal{A}(\Pi)$, there is a path from x to y in $\mathcal{G}$ if, and only if, there is a path from x to y in $\mathcal{G}'$. Hence, also path constraints and negated path constraints are satisfied by $\mathcal{G}'$, because they are satisfied by $\mathcal{G}$. That is, $\mathcal{G}' \models \Pi$. □

9.3 Precedence Constraints

From a conceptual viewpoint, the problem defined above comprises a mining task, i.e., mining a process model supporting a given log, and a reasoning task, i.e., to check whether the model additionally satisfies some precedence constraints. We next show that even the mining task can be declaratively formulated in terms of reasoning

Function `computeBindings`($\mathcal{G}$, L), with $\mathcal{G} = (\mathcal{A}, E)$;

$\forall a \in \mathcal{A}$, $\mathcal{I}(a) := I_a$ and $\mathcal{O}(a) := O_a$, where
  $I_a = \{(x, a) \mid (x, a) \in E\}$ and $O_a = \{(a, y) \mid (a, y) \in E\}$;
for each trace t in L, and for each $i \in \{1, ..., len(t)\}$ **do**
  $\mathcal{I}(t[i]) := \mathcal{I}(t[i]) \cup \{(t[j], t[i]) \in E \mid j < i\}$; (*treated as multi-sets*)
  $\mathcal{O}(t[i]) := \mathcal{O}(t[i]) \cup \{(t[i], t[j]) \in E \mid i < j\}$; (*treated as multi-sets*)
end for
return $\langle \mathcal{G}, \mathcal{I}, \mathcal{O} \rangle$;

Figure 9.2: Computation of a possibly extended causal net in Theorem 9.6.

about precedence constraints. In other words, the language of precedence constraints is enough expressive as to capture the semantics of causal nets.

Definition 9.5. *Let L be a log. For each trace $t \in L$, the set of precedence constraints* induced by *t is defined as follows:*

$$\pi(t) = \{\ \{t[1], ..., t[i-1]\} \rightarrow \{t[i]\} \mid 1 < i \leq len(t)\ \} \cup \{\ \{t[i]\} \rightarrow \{t[i+1], ..., t[len(t)]\} \mid 1 \leq i < len(t)\ \}.$$

The set of precedence constraints induced by L is defined as $\pi(L) = \bigcup_{t \in L} \pi(t)$. □

Intuitively, we have stated that each activity in t can be directly reached by at least one of its predecessors in t, and it can directly reach at least one of its successors. Based on this definition, we establish the following crucial result.

Theorem 9.6. *Let $\mathcal{G}$ be a dependency graph over a set $\mathcal{A}$ of activities, let L be a log with $\mathcal{A}(L) \subseteq \mathcal{A}$, and let Π be a set of precedence constraints with $\mathcal{A}(\Pi) \subseteq \mathcal{A}$. The following statements are equivalent:*

(1) *The graph $\mathcal{G}$ is a model of $\Pi \cup \pi(L)$.*

(2) *There is a possibly extended causal net $\mathcal{C} = \langle \mathcal{G}, \mathcal{I}, \mathcal{O} \rangle$ that is a solution to* CN-Mining *on input $\mathcal{A}$, L, and Π.*

Moreover, input and output bindings witnessing that the "(1)⇒(2)"-part holds can be built via the function reported in Figure 9.2. In particular, if there is no trace in

L containing repetitions of the same activity (for short, we hereinafter say that L is linear), then the function actually builds a (standard) causal net.

Since the proof is rather involved, as a base case we show that the result holds in absence of user-defined constraints and when the focus is on linear logs only. Note that in Chapter 8 we have introduced a number of technical ingredients which will use in this proof.

Theorem 9.7. *Let L be a linear log and let $\mathcal{G}$ be a dependency graph. Then, $\mathcal{G} \models \pi(L)$ if, and only if, $\mathcal{G} \vdash_a L$.*

Proof. (if part). Assume that $\mathcal{G} \vdash_a L$, i.e., for each $t \in L$, there is a subgraph $\mathcal{G}_t = (V_t, E_t)$ of $\mathcal{G} = (V, E)$ such that $\mathcal{G}_t$ is an acyclic dependency graph, and t is a topologic sort of $\mathcal{G}_t$. Therefore, for each $i \in \{2, ..., len(t)\}$ (resp., $i \in \{1, ..., len(t)-1\}$), there is a path from $t[1]$ (resp., $t[i]$) to $t[i]$ (resp., $t[len(t)]$) in $\mathcal{G}_t$, by definition of dependency graph. In particular, since t is a topologic sort of $\mathcal{G}_t$, we are guaranteed about the existence of an edge in E_t (and then in E) having the form $(t[j], t[i])$ (resp., $(t[i], t[j'])$) and such that $j < i$ (resp., $i < j'$) holds. Hence, the set $\pi(t)$ of the precedence constraints induced by t are satisfied by $\mathcal{G}$, for each trace t in L. That is, $\mathcal{G} \models \pi(L)$.

(only-if part). Assume that $\mathcal{G} \models \pi(L)$, with $\mathcal{G} = (V, E)$. Let t be a trace in L, and let $\mathcal{G}_t = (V_t, E_t)$ be the graph such that $V_t = \{t[1], ..., t[len(t)]\}$ and $E_t = \{(t[i], t[j]) \in E \mid 1 \leq i < j \leq len(t)\}$. Since L is linear, we can note that $\mathcal{G}_t$ is acyclic, that $t[1] = a_\perp$ has no ingoing edges, and that $t[len(t)] = a_\top$ has no outgoing edges. We now claim that each activity $t[i] \in V_t \setminus \{t[1]\}$ can be reached from $t[1]$. The above property can be shown by induction on the index $i > 1$. In the case where $i = 2$, $(t[1], t[2])$ must belong to E (and hence to E_t), in order to satisfy the constraint

$\{t[1]\} \to \{t[2]\}$ in $\pi(t)$. Assume now that the activities in the set $\{t[2], ..., t[i-1]\}$ can be reached from $t[1]$. Then, because of the constraint $\{t[1], ..., t[i-1]\} \to \{t[i]\}$ in $\pi(t)$, we again have that $t[i]$ can be reached from $t[1]$. Similarly, it can be checked that the terminating activity $t[len(t)]$ can be reached by each activity $t[i] \in V_t \setminus \{t[len(t)]\}$, by using this time the fact that $\{t[i]\} \to \{t[i+1], ..., t[len(t)]\}$ is in $\pi(t)$. Hence, $\mathcal{G}_t$ is an acyclic dependency graph. Moreover, for each edge $(t[i], t[j])$ in E_t, we have that $i < j$ holds by construction. Thus, t is a topologic sort of $\mathcal{G}_t$. As $\mathcal{G}_t$ is a subgraph of $\mathcal{G}$, we then have $\mathcal{G} \vdash_a L$. □

Theorem 9.7 and Theorem 8.8 imply the following corollary, where process mining over linear logs is restated in terms of reasoning about precedence constraints.

Corollary 9.8. Let $\mathcal{G}$ be a dependency graph over a set $\mathcal{A}$ of activities, let L be a linear log with $\mathcal{A}(L) \subseteq \mathcal{A}$, and let Π be a set of precedence constraints with $\mathcal{A}(\Pi) \subseteq \mathcal{A}$. Then, the followings are equivalent: *(1)* The graph $\mathcal{G}$ is a model of $\Pi \cup \pi(L)$; *(2)* There is a causal net $\mathcal{C} = \langle \mathcal{G}, \mathcal{I}, \mathcal{O} \rangle$ that is a solution to CN-Mining on input $\mathcal{A}$, L, and Π.

Example 9.9. *Let* Π *be the set of constraints in Example 9.3, and consider the novel set* $\Pi' = \Pi \cup \pi(\{abcde, acbde\})$. *It can be checked that the dependency graph* $\mathcal{G}_2$ *in Figure 8.2 is a model of* Π'. *Thus, by Corollary 9.8, we are guaranteed about the existence of a causal net that can be defined on top of* $\mathcal{G}_2$ *and that is a solution to* CN-Mining *on input* $\{abcde, acbde\}$ *and* Π. *In fact, we already know that the causal net* $\mathcal{C}_2$ *defined in Example 8.3 is a solution.* ◁

Note that, the above result completely characterizes the cases where the problem CN-Mining admit solutions. Indeed, point (2) above can be equivalently restated as the existence of a possibly extended causal net that is a solution to the mining

problem. This is because of the following fact, which trivially holds since the ability of extended nets to activate multiple instances of the same activity is useless over linear logs.

Fact 9.10. Over linear logs, CN-Mining admits a solution if, and only if, it admits a causal net as a solution.

Moreover, it is useful to remark that the '(1)⇒(2)'-part of Corollary 9.8 can be stated constructively. That is, the causal net $\langle \mathcal{G}, \mathcal{I}, \mathcal{O} \rangle$ can be efficiently built given the graph $\mathcal{G}$. The method is just based on inspecting the proofs for the results in Section 8.4, and in fact it coincides with the one algorithmically reported in Figure 9.2.

At this point, the natural question is whether we can extend Theorem 9.7 to deal with arbitrary logs. An answer, which is however only partially positive, is stated below.

Theorem 9.11. *Let L be a log and let $\mathcal{G}$ be a dependency graph. Then, $\mathcal{G} \vdash L$ implies that $\mathcal{G} \models \pi(L)$.*

Proof. Assume that $\mathcal{G} \vdash L$ holds, with $\mathcal{G} = (V, E)$. By Definition 8.9, for each trace $t \in L$, there is a graph $\bar{\mathcal{G}}_t = (\bar{V}_t, \bar{E}_t)$ such that, in particular, $\mathit{fold}(\bar{\mathcal{G}}_t)$ is a subgraph of $\mathcal{G} = (V_t, E_t)$ and $\bar{\mathcal{G}}_t \vdash_a \{\mathit{unfold}(t)\}$. Then, we apply Theorem 9.7 on $\bar{\mathcal{G}}_t$, and we conclude that $\bar{\mathcal{G}}_t \models \pi(\mathit{unfold}(t))$. Moreover, we note that $\mathit{fold}(\bar{\mathcal{G}}_t) \models \pi(t)$ also holds. Indeed, for each $i \in \{1, ..., \mathit{len}(t)\}$, whenever $(\mathit{unfold}(t)[j], \mathit{unfold}(t)[i])$ (resp., $(\mathit{unfold}(t)[i], \mathit{unfold}(t)[h])$) is in $\bar{E}_t$, then $(t[j], t[i])$ (resp., $(t[i], t[h])$) is an edge of $\mathit{fold}(\bar{\mathcal{G}}_t)$ with $j < i$ (resp., $h > i$). Finally, define $\bar{\mathcal{G}} = (\bigcup_{t \in L} \bar{V}_t, \bigcup_{t \in L} \bar{E}_t)$. Then, we claim that $\mathit{fold}(\bar{\mathcal{G}}) \models \pi(L)$. Indeed, the constraints induced by L are positive ones, so that, since $\mathit{fold}(\bar{\mathcal{G}}_t) \models \pi(t)$, the graph $\mathit{fold}(\bar{\mathcal{G}})$ (of which $\mathit{fold}(\bar{\mathcal{G}}_t)$ is a subgraph) is

still such that $fold(\bar{\mathcal{G}}) \models \pi(t)$. To conclude, we can eventually observe that $fold(\bar{\mathcal{G}})$ is a subgraph of $\mathcal{G}$, and hence $\mathcal{G} \models \pi(L)$. □

This is the best one can hope to do. Indeed, we can see that $\mathcal{G} \models \pi(L)$ does not imply $\mathcal{G} \vdash L$, by just looking again at Example 8.10. There, we have noticed that the graph $\mathcal{G}_3$ does not support $\{abbe\}$. However, it can be checked that $\mathcal{G}_3 \models \pi(\{abbe\})$ holds.

Despite the above limitation, the counterpart of Corollary 9.8 for arbitrary logs can still be obtained by moving to possibly extended causal nets—as with Corollary 9.8, the '(1)⇒(2)'-part below is formalized algorithmically in Figure 9.2.

Theorem 9.12. *Let $\mathcal{G}$ be a dependency graph over a set $\mathcal{A}$ of activities, let L be a log with $\mathcal{A}(L) \subseteq \mathcal{A}$, and let Π be a set of precedence constraints with $\mathcal{A}(\Pi) \subseteq \mathcal{A}$. The followings are equivalent:* (1) *The graph $\mathcal{G}$ is a model of $\Pi \cup \pi(L)$;* (2) *There is a possibly extended causal net $\mathcal{C} = \langle \mathcal{G}, \mathcal{I}, \mathcal{O} \rangle$ that is a solution to* CN-Mining *on input $\mathcal{A}$, L, and Π.*

Proof. (1)⇒(2). Assume that $\mathcal{G}$ is a model of $\Pi \cup \pi(L)$, with $\mathcal{G} = (V, E)$. Let $\bar{\mathcal{G}} = (\bar{V}, \bar{E})$ be the graph such that $\bar{V} = \mathcal{A}(unfold(L))$ and $\bar{E} = \{(x^{\langle i \rangle}, y^{\langle j \rangle}) \mid (x, y) \in E, x^{\langle i \rangle} \in \bar{V}, y^{\langle j \rangle} \in \bar{V}\}$. Since $\mathcal{G}$ is a dependency graph, it is also the case that $\bar{\mathcal{G}}$ is a dependency graph (over $\mathcal{A}(unfold(L))$). Moreover, $\bar{\mathcal{G}} \models \pi(unfold(L))$ holds. Indeed, just note that for each trace $t \in L$ and for each $i \in \{1, ..., len(t)\}$, if $(t[j], t[i])$ (resp., $(t[i], t[h])$) is in E with $j < i$ (resp., $h > i$), then $(unfold(t)[j], unfold(t)[i])$ (resp., $(unfold(t)[i], unfold(t)[h])$) is in $\bar{E}$ by construction. Thus, we can apply Theorem 9.7 in order to conclude that $\bar{\mathcal{G}} \vdash_a unfold(L)$. Eventually, we observe that $fold(\bar{\mathcal{G}})$ is clearly a subgraph of $\mathcal{G}$. Then, we distinguish two cases. In the case where $\bar{\mathcal{G}}$ satisfies conditions (1) and (2) in Definition 8.9, then we can apply Theorem 8.11

and conclude that a causal net $\mathcal{C}$ can be built over $\mathcal{G}$ such that $\mathcal{C} \vdash L$. Instead, in the case where one of the above conditions does not hold, we can apply Theorem 8.13 and conclude that there is a possibly extended causal net $\mathcal{C}$ built over $\mathcal{G}$ and such that $\mathcal{C} \vdash L$.

(2)⇒(1). Assume that $\mathcal{C} = \langle \mathcal{G}, \mathcal{I}, \mathcal{O} \rangle$ is a solution to CN-Mining on input $\mathcal{A}$, L, and Π, and that, for the sake of contradiction, $\mathcal{G}$ is not a model of $\Pi \cup \pi(L)$. The fact that $\mathcal{G}$ is not a model of Π is trivially impossible. Thus, assume that $\mathcal{G}$ is not a model of $\pi(L)$, i.e., there is a trace $t \in L$ such that $\mathcal{G}$ does not satisfy the constraints in $\pi(t)$. According to Definition 9.5, there are two possible cases: (a) there is an index $i \in \{2, ..., len(t)\}$ such that there is no edge in $\mathcal{G}$ from one of the activities in $\{t[1], ..., t[i-1]\}$ to $t[i]$; (b) there is an index $i \in \{1, ..., len(t) - 1\}$ such that there is no edge in $\mathcal{G}$ from $t[i]$ to one of the activities in $\{t[i+1], ..., t[len(t)]\}$.

Now, as $\mathcal{C}$ is a solution, it must be the case that $\mathcal{C}$ supports the trace t, i.e., there is a binding sequence σ that is valid w.r.t. $\mathcal{C}$ and where the j-th binding activity $\langle a_j, ib_j, ob_j \rangle$ of σ, for each $j \in \{1, ..., len(t)\}$, is such that $a_j = t[j]$. In particular, the i-th activity is $\langle t[i], ib_i, ob_i \rangle$. In the case (a) above, the set ib_i, which is a non-empty set of edges ingoing into $t[i]$, is a subset of the set $\{(x, t[i]) \mid x \notin \{t[1], ..., t[i-1]\}\}$, while the state S^{σ}_{i-1} of the causal net consists of edges having the form (x', y), with $x' \in \{t[1], ..., t[i-1]\}$. It follows that ib_i is not contained in S^{σ}_{i-1}, and σ is not valid, which is impossible as σ is valid. In the case (b), the set ob_i, which is a non-empty set of edges outgoing from $t[i]$, is included in the set $\{(t[i], y) \mid y \notin \{t[1+1], ..., t[len(t)]\}\}$. Thus, S^{σ}_{i} necessarily includes an element having the form $(t[i], y)$, with $y \notin \{t[1+1], ..., t[len(t)]\}$. However, as there is no edge in $\mathcal{G}$ from $t[i]$ to one of the activities in $\{t[i+1], ..., t[len(t)]\}$, this element will occur at the last step of the execution of σ, i.e., $S^{\sigma}_{len(t)} \neq \emptyset$, which again is impossible. □

By combining the above result with Corollary 9.8, we have established the result in Theorem 9.6. In particular, the constructive "(1)⇒(2)"-part is seen to hold by inspecting the construction in the proofs and by observing that input and output bindings are constructed there precisely according to the function reported in Figure 9.2.

As an example application, since $\mathcal{G}_3 \models \pi(\{abbe\})$ holds, we are guaranteed that a net supporting this trace can be built on top of $\mathcal{G}_3$. To be concrete, we can consider the extended causal net $\mathcal{C}'_3 = \langle \mathcal{G}_3, \mathcal{I}'_3, \mathcal{O}'_3 \rangle$ where $\mathcal{I}'_3(a) = \{\emptyset\}$, $\mathcal{I}'_3(b) = \{\{(a,b)\}\}$, $\mathcal{I}'_3(e) = \{\{(b,e),(b,e)\}\}$, $\mathcal{O}'_3(a) = \{\{(a,b),(a,b)\}\}$, $\mathcal{O}'_3(b) = \{\{(b,e)\}\}$, and $\mathcal{O}'_3(e) = \{\emptyset\}$—the specification of the bindings over the activities c and d is irrelevant here.

Note that Theorem 9.6 reformulates the mining problem and the dynamics of causal nets in purely "static" terms, i.e., in terms of reasoning about the satisfaction of precedence constraints with no reference to the concepts of bindings and of trace/log support. This will be very useful for designing our algorithms and for conducting the complexity analysis. Moreover, the result has an immediate concrete application. Indeed, the reasoning problem can be easily encoded via a "standard" *constraints satisfaction problem*, CSP for short (see, e.g., Dechter [1992]), so that existing constraint programming platforms can be reused to compute solutions, in the spirit of the works by [De Raedt et al., 2008] and [Nijssen et al., 2009]. A preliminary version of this approach was discussed by [Greco *et al.*, 2012], however it was not designed to deal with causal nets (i.e., with full process models), but its focus was on the discovery of the underlying dependence graphs only.

An advantage of the CSP-based framework is that it can transparently handle arbitrary sets of precedence constraints. The price to be paid, however, is that computing a solution might be very challenging from a computational viewpoint, as we

shall formally illustrate in Section 9.4. In this dissertation, we decided therefore to propose solution algorithms that are specific for some classes of constraints (over which solutions can be efficiently computed), as well as methods that handle the general setting heuristically. In fact, from our experimentation, the heuristics methods emerged to be definitively much faster (orders of magnitude) than the CSP-based method, while being capable to end up with an exact solution in most cases. Accordingly, for those cases where a heuristic solution is acceptable (i.e., where some constraints might be violated), our earlier approach has to be considered as pragmatically superseded by the methods proposed here. However, it is an interesting avenue of further research to define different encodings (possibly, still CSP-based ones).

9.4 Complexity Analysis

We now complete the picture by studying the computational complexity of the problem CN-Mining, which is an important step towards developing effective algorithms for its solution. In the analysis that follows, we take into account various qualitative properties regarding the kinds of constraint being allowed, by tracing the tractability frontier w.r.t. them. Formally, let $\mathtt{S}$ be a subset of the following set of symbols $\{\rightarrow, \rightsquigarrow, \not\rightarrow, \not\rightsquigarrow\}$. Let $\mathcal{C}[\mathtt{S}]$ denote all the possible constraints that can be built in a way that if $\rightarrow \notin \mathtt{S}$ (resp., $\rightsquigarrow \notin \mathtt{S}$, $\not\rightarrow \notin \mathtt{S}$, $\not\rightsquigarrow \notin \mathtt{S}$), then no edge (resp., path, negated edge, negated path) constraint is in $\mathcal{C}[\mathtt{S}]$). Let CN-Mining$[\mathtt{S}]$ denote the restriction of the problem over any set Π of precedence constraints such that $\Pi \subseteq \mathcal{C}[\mathtt{S}]$. And, finally, let CN-Existence$[\mathtt{S}]$ be the *decision version* of this problem, where we have just to decide whether a solution exist at all, and we are not asked to compute a solution, if any. Then, our results can be summarized as it is stated next (see also Figure 9.3).

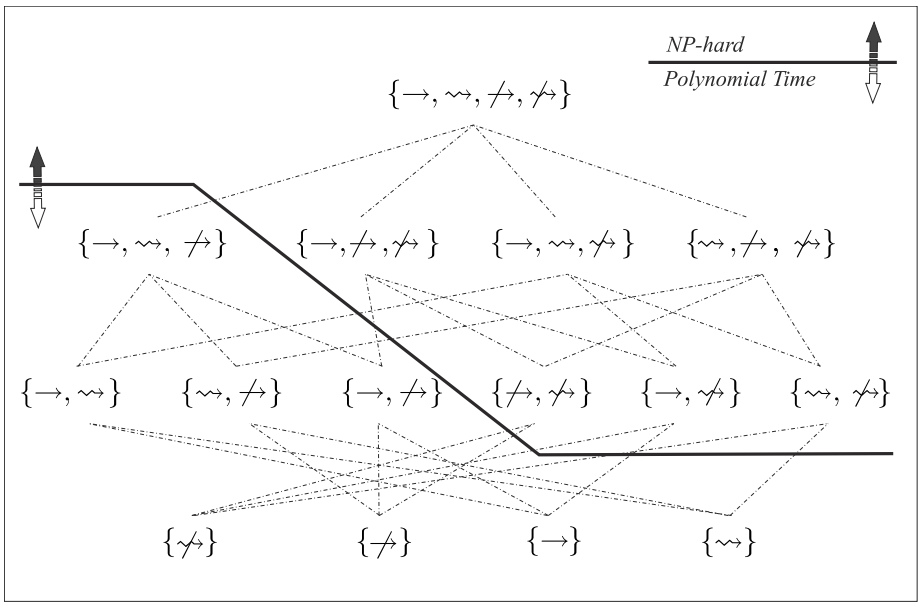


Figure 9.3: Tractability frontiers. A set $\mathtt{S} \subseteq \{\rightarrow, \rightsquigarrow, \not\rightarrow, \not\rightsquigarrow\}$ above (resp., below) the frontier means that the corresponding problem is NP-hard (resp., feasible in polynomial time) on $\mathcal{C}[\mathtt{S}]$.

Theorem 9.13. *If* $\mathtt{S} \subseteq \{\rightarrow, \rightsquigarrow, \not\rightarrow\}$ *or* $\mathtt{S} \subseteq \{\not\rightsquigarrow\}$*, then* CN-Mining[S] *is feasible in polynomial time. Otherwise, it is even intractable to check whether there is a solution at all (formally, the problem* CN-Existence[S] *is* NP*-complete).*

We now provide the proof of the hardness results in Theorem 9.13. Note that these proofs are based on linear logs only. Hence, according to Fact 9.10, for these results it is immaterial whether CN-Mining is defined over extended causal nets or over (standard) causal nets.

For the analysis that follows, we find convenient to introduce a variant of the problem CN-Mining, which we call Acyclic-CN-Mining. Given a set $\mathcal{A}$ of activities, a log L with $\mathcal{A}(L) \subseteq \mathcal{A}$, and a set Π of precedence constraints with $\mathcal{A}(\Pi) \subseteq \mathcal{A}$, the problem asks to compute a possibly extended causal net $\mathcal{C} = \langle \mathcal{G}, \mathcal{I}, \mathcal{O} \rangle$ over $\mathcal{A}$ such that $\mathcal{G}$ is acyclic, $\mathcal{C} \vdash L$, and $\mathcal{G} \models \Pi$, or to check that no net with these properties exists.

We start the proofs of the hardness results stated in Theorem 9.13, by focusing

on edge and path constraints imposed over the problem Acyclic-CN-Existence.

Lemma 9.14. Acyclic-CN-Existence$/\{\rightarrow\}/$ is NP-hard.

Proof. Consider the *monotone one-in-three 3SAT* problem defined as follows. We are given as input a Boolean formula in conjunctive normal form $\Phi = c_1 \wedge \ldots \wedge c_m$ over the variables $X_1, \ldots, X_n$, where each clause c_j, with $j \in \{1, ..., m\}$, has the form $(t_{j,1} \vee t_{j,2} \vee t_{j,3})$, and $t_{j,1}, t_{j,2}$, and $t_{j,3}$ are three distinct variables. Note that Φ is always satisfiable by the truth assignment where all variables evaluate true. In fact, the problem asks whether there is a satisfying truth assignment where each clause has exactly one variable evaluating true (and hence two variables evaluating false). This problem is known to be NP-complete Schaefer [1978].

Based on any formula Φ as above, we build the set $\mathcal{A}(\Phi)$ consisting of the variables in Φ, which are transparently viewed as activities, plus the three distinguished activities c, $a_\perp$ and $a_\top$, where $a_\perp$ and $a_\top$ are as usual the starting and the terminating activity, respectively. So, we formally have $\mathcal{A}(\Phi) = \{c\} \cup \{X_1, ..., X_n\} \cup \{a_\perp, a_\top\}$.

Moreover, we build the set $\Pi(\Phi) \subseteq \mathcal{C}[\{\rightarrow\}]$ of edge constraints as follows. For each clause c_j, $\Pi(\Phi)$ contains the constraints $\{t_{j,1}, t_{j,2}, t_{j,3}\} \rightarrow \{c\}$, $\{c\} \rightarrow \{t_{j,1}, t_{j,2}\}$, $\{c\} \rightarrow \{t_{j,1}, t_{j,3}\}$, and $\{c\} \rightarrow \{t_{j,2}, t_{j,3}\}$. No further constraint is in $\Pi(\Phi)$.

We now claim that: There is a satisfying truth assignment to the variables of Φ such that each clause has exactly one variable evaluating true $\Leftrightarrow$ there is an acyclic dependency graph $\mathcal{G}$ (over $\mathcal{A}(\Phi)$) such that $\mathcal{G} \models \Pi(\Phi)$.

($\Rightarrow$) Assume that σ is a satisfying truth assignment such that each clause has exactly one variable evaluating true. Consider the graph $\mathcal{G}(\Phi, \sigma) = (\mathcal{A}(\Phi), E)$ whose set of edges is defined as follows. For each variable X_h, with $h \in \{1, ..., n\}$, the edges $(a_\perp, X_h)$ and $(X_h, a_\perp)$ are in E. For each clause c_j and each variable $t_{j,i}$ evaluating

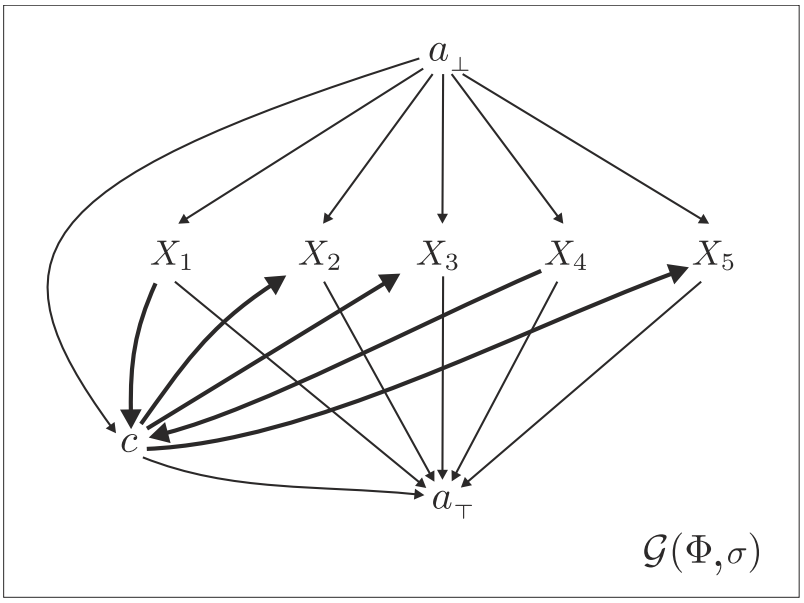


Figure 9.4: Example reduction in the proof of Lemma 9.14.

true (resp., false) in σ, the edge $(t_{j,i}, c)$ (resp., $(c, t_{j,i})$) is in E. The edges $(a_\bot, c)$ and $(c, a_\top)$ are in E, and no further edge is in E. As an example, the graph $\mathcal{G}(\Phi, \sigma)$ associated with the formula $\Phi = (X_1 \vee X_2 \vee X_3) \wedge (X_3 \vee X_4 \vee X_5)$ and the truth assignment σ, where X_1 and X_4 are the only variables evaluating true, is reported in Figure 9.4. In particular, note that the edges whose definition depend on σ are depicted in bold.

We first show that $\mathcal{G}(\Phi, \sigma)$ is a dependency graph. Indeed, $a_\bot$ and $a_\top$ have no ingoing and outgoing edges, respectively. Moreover, for each activity $a \in \mathcal{A}(\Phi) \setminus \{a_\bot, a_\top\}$, the edges $(a_\bot, a)$ and $(a, a_\top)$ are in E, so that a occurs in a path from $a_\bot$ to $a_\top$.

Then, we show that $\mathcal{G}(\Phi, \sigma)$ satisfies all the constraints in $\Pi(\Phi)$. Recall that σ is a satisfying truth assignment such that each clause has exactly one variable evaluating true. Thus, for each clause c_j, with $j \in \{1, ..., m\}$, by construction there is an edge of the form $(t_{j,i}, c)$, so that the constraint $\{t_{j,1}, t_{j,2}, t_{j,3}\} \to \{c\}$ is satisfied. Moreover, there are also two edges of the form $(c, t_{j,i'})$ and $(c, t_{j,i''})$, where $i' \neq i$,

$i'' \neq i$, $i' \neq i''$, and $\{i', i''\} \subseteq \{1, 2, 3\}$. Thus, the constraints $\{c\} \rightarrow \{t_{j,1}, t_{j,2}\}$, $\{c\} \rightarrow \{t_{j,1}, t_{j,3}\}$, and $\{c\} \rightarrow \{t_{j,2}, t_{j,3}\}$ are also satisfied, for each clause c_j. It follows that all constraints in $\Pi(\Phi)$ are satisfied by $\mathcal{G}(\Phi, \sigma)$, i.e., $\mathcal{G}(\Phi, \sigma) \models \Pi(\Phi)$.

In order to conclude, we now just need to point out that $\mathcal{G}(\Phi, \sigma)$ is acyclic. To this end, assume for the sake of contradiction that a cycle exists in $\mathcal{G}(\Phi, \sigma)$. Note that this cycle is necessarily defined over the variables and the distinguished activity c. Therefore, the set E of edges must contain an edge of the form (c, X_h) and an edge of the form (X_h, c). By construction, the existence of the edge (c, X_h) implies that X_h is a variable evaluating false in σ. However, the existence of the edge (X_h, c) implies that X_h is a variable evaluating true in σ. Contradiction.

($\Leftarrow$) Assume that $\mathcal{G} = (V, E)$ is an acyclic dependency graph satisfying all the constraints in $\Pi(\Phi)$, and define the truth assignment $\sigma_{\mathcal{G}}$ such that X_h evaluates true if, and only if, the edge (X_h, c) occurs in E. We first show that $\sigma_{\mathcal{G}}$ is satisfying. Indeed, for each clause c_j, with $j \in \{1, ..., m\}$, consider the associated constraint $\{t_{j,1}, t_{j,2}, t_{j,3}\} \rightarrow \{c\}$, and note that since $\mathcal{G} \models \Pi(\Phi)$, we are guaranteed about the existence of an edge from one of the variables in $\{t_{j,1}, t_{j,2}, t_{j,3}\}$ to c. Hence, for each clause c_j, at least one of the variables occurring in c_j evaluates true in $\sigma_{\mathcal{G}}$, by definition of this assignment, which is therefore satisfying.

Now, we show that $\sigma_{\mathcal{G}}$ is a such that each clause has exactly one variable evaluating true. Assume, for the sake of contradiction, that a clause c_j exists such that two variables, say $t_{j,i'}$ and $t_{j,i''}$ with $i' \neq i''$, evaluate true in $\sigma_{\mathcal{G}}$. Thus, the edges $(t_{j,i'}, c)$ and $(t_{j,i''}, c)$ are both in E. Consider then the constraint $\{c\} \rightarrow \{t_{j,i'}, t_{j,i''}\}$ associated with the clause c_j, and note that it prescribes that at least one of the edges in $\{(c, t_{j,i'}), (c, t_{j,i''})\}$ occurs in E. Assume, w.l.o.g., that $(c, t_{j,i'})$ is in E and observe that we have eventually a cycle over c and $t_{j,i'}$. Contradiction.

By Corollary 9.8 and Fact 9.10, the above entails that ACYCLIC-CN-MINING$[\{\rightarrow\}]$ on input $\mathcal{A}(\Phi)$, the empty log, and the set $\Pi(\Phi)$ has a solution if, and only if, there is a satisfying truth assignment to the variables of Φ such that each clause has exactly one variable evaluating true. As the reduction is feasible in polynomial time, it follows that ACYCLIC-CN-EXISTENCE$[\{\rightarrow\}]$ is NP-hard. □

A straightforward adaptation of the above proof, where each edge constraint is replaced by a path constraint over the same sets of activities, can be used to show that the problem remains intractable if we consider path constraints in place of edge constraints. The proof is reported below, for the sake of completeness.

Lemma 9.15. ACYCLIC-CN-EXISTENCE$[\{\rightsquigarrow\}]$ is NP-hard.

Proof. Consider again the setting in the proof of Lemma 9.14 and, for any formula Φ, define $\Pi'(\Phi) \subseteq \mathcal{C}[\{\rightsquigarrow\}]$ as the set of constraints obtained from $\Pi(\Phi)$ by replacing each edge constraint with the analogous path constraint. Therefore, for each clause c_j, $\Pi'(\Phi)$ contains the constraints $\{t_{j,1}, t_{j,2}, t_{j,3}\} \rightsquigarrow \{c\}$, $\{c\} \rightsquigarrow \{t_{j,1}, t_{j,2}\}$, $\{c\} \rightsquigarrow \{t_{j,1}, t_{j,3}\}$, and $\{c\} \rightsquigarrow \{t_{j,2}, t_{j,3}\}$. We now claim that: There is a satisfying truth assignment to the variables of Φ such that each clause has exactly one variable evaluating true $\Leftrightarrow$ there is an acyclic dependency graph $\mathcal{G}$ (over $\mathcal{A}(\Phi)$) such that $\mathcal{G} \models \Pi'(\Phi)$.

($\Rightarrow$) Assume that σ is a satisfying truth assignment such that each clause has exactly one variable evaluating true, and consider the graph $\mathcal{G} = (\Phi, \sigma)$ built in the proof of Lemma 9.14. Recall that $\mathcal{G} \models \Pi(\Phi)$. Hence, we trivially have that $\mathcal{G} \models \Pi'(\Phi)$, in particular because all path constraints are satisfied by direct connections.

($\Leftarrow$) Assume that $\mathcal{G} = (V, E)$ is an acyclic dependency graph satisfying all the constraints in $\Pi'(\Phi)$, and define the truth assignment $\sigma'_{\mathcal{G}}$ such that X_h evaluates true

if, and only if, there is a path from X_h to c in E. Because of the constraints $\{t_{j,1}, t_{j,2}, t_{j,3}\} \rightsquigarrow \{c\}$ occurring in $\Pi'(\Phi)$ and associated with the clause c_j, for each $j \in \{1, ..., m\}$, $\sigma'_{\mathcal{G}}$ is satisfying. In order to conclude, we need to show that $\sigma'_{\mathcal{G}}$ is a such that each clause has exactly one variable evaluating true. Assume, for the sake of contradiction, that a clause c_j exists such that two variables, say $t_{j,i'}$ and $t_{j,i''}$ with $i' \neq i''$, evaluate true in $\sigma_{\mathcal{G}}$. Thus, there is a path from $t_{j,i'}$ to c and a path from $t_{j,i''}$ to c are both in E. Consider then the constraint $\{c\} \rightsquigarrow \{t_{j,i'}, t_{j,i''}\}$ associated with the clause c_j, and note that it prescribes that there is a path from c to at least one of the nodes in $\{t_{j,i'}, t_{j,i''}\}$. Assume, w.l.o.g., that a path from c to $t_{j,i'}$ exists. Then, we have a cycle involving the activities c and $t_{j,i'}$. Contradiction.

By the above result, Corollary 9.8, Fact 9.10, and the fact that the reduction is feasible in polynomial time, it follows that Acyclic-CN-Existence$[\{\rightsquigarrow\}]$ is NP-hard. □

Let us now turn to the case of negated edge constraints, but still focusing on the acyclic variant of the mining problem.

Lemma 9.16. Acyclic-CN-Existence$[\{\not\rightsquigarrow\}]$ is NP-hard.

Proof. Let $\Pi = \{\{b_i^1, ..., b_i^{k_i}\} \rightarrow \{a_i^1, ..., a_i^{h_i}\} \mid i \in \{1, ..., m\}\} \subseteq \mathcal{C}[\{\rightarrow\}]$ be a (non-empty) set of edge constraints, such that $\{b_i^1, ..., b_i^{k_i}\} \cap \{a_i^1, ..., a_i^{h_i}\} = \emptyset$, for each $i \in \{1, ..., m\}$. For each $i \in \{1, ..., m\}$, let c_i denote a fresh activity not in $\mathcal{A}(\Pi)$. Moreover, let $a_{\perp}$ and $a_{\top}$ be two activities not in $\mathcal{A}(\Pi)$ playing the role of the starting and terminating activity, respectively. Then, consider the log $L(\Pi) = \{t_1, ..., t_m\}$ such that $t_i = a_{\perp} b_i^1 ... b_i^{k_i} c_i a_i^1 ... a_i^{h_i} a_{\top}$, for each $i \in \{1, ..., m\}$. Moreover, consider the set $\Pi^{\neg}$ of negated edge constraints such that $\Pi^{\neg} = \{\neg(\{a_{\perp}\} \rightarrow \{c_i\}), \neg(\{c_i\} \rightarrow \{a_{\top}\}) \mid i \in \{1, ..., m\}\}$.

We claim that: There is an acyclic dependency graph $\mathcal{G}$ over the set $\mathcal{A}(\Pi) \cup \{a_\perp, a_\top\}$ of activities and such that $\mathcal{G} \models \Pi \Leftrightarrow$ there is an acyclic dependency graph $\mathcal{G}'$ over the set $\mathcal{A}(\Pi) \cup \{a_\perp, a_\top\} \cup \{c_1, ..., c_m\}$ of activities and such that $\mathcal{G}' \models \pi(L(\Pi)) \cup \Pi^\neg$.

($\Rightarrow$) Assume that $\mathcal{G} = (V, E)$ is an acyclic dependency graph over the set $\mathcal{A}(\Pi) \cup \{a_\perp, a_\top\}$ of activities and such that $\mathcal{G} \models \Pi$. Consider the graph $\mathcal{G}' = (V', E \cup E'_1 \cup E'_2)$ where $V' = V \cup \{c_1, ..., c_m\}$ and whose set of edges is built as follows. The set E'_1 contains the edges $(a_\perp, a)$ and $(a, a_\top)$, for each activity $a \in \mathcal{A}(\Pi)$, and no further edge is in E'_1. Moreover, for each $i \in \{1, ..., m\}$, and for each edge $(x, y) \in E$ such that $x \in \{b_i^1, ..., b_i^{k_i}\}$ and $y \in \{a_i^1, ..., a_i^{h_i}\}$, E'_2 includes the edges (x, c_i) and (c_i, y). No further edge is in E'_2. It is immediately checked that $\mathcal{G}'$ is an acyclic dependency graph.

Now, we first claim that $\mathcal{G}' \models \Pi^\neg$. Indeed, $\{a_\perp, a_\top\} \cap \mathcal{A}(\Pi) = \emptyset$, so that E'_2 does not include any edge of the form $(a_\perp, c_i)$ or of the form $(c_i, a_\top)$, with $i \in \{1, ..., m\}$. Let now t_i be a trace in $L(\Pi)$, with $i \in \{1, ..., m\}$. Consider the set $\pi(t_i)$ of constraints derived from t_i according to Definition 9.5. Because of the edges in E'_1, all constraints in $\pi(t_i)$ are trivially satisfied by $\mathcal{G}'$, but the two constraints $\{a_\perp, b_i^1, ..., b_i^{k_i}\} \rightarrow \{c_i\}$ and $\{c_i\} \rightarrow \{a_i^1, ..., a_i^{h_i}, a_\top\}$, because there is no edge in $\mathcal{G}'$ from $a_\perp$ to c_i, and from c_i to $a_\top$. Now, recall that $\mathcal{G}$ satisfies Π and hence, an edge (x, y) occurs in E such that $x \in \{b_i^1, ..., b_i^{k_i}\}$ and $y \in \{a_i^1, ..., a_i^{h_i}\}$. By construction of the edges in E'_2, we therefore have that (x, c_i) and (c_i, y) are edges of $\mathcal{G}'$, which proves that the two constraints are satisfied, too. Hence, $\mathcal{G}' \models \pi(L(\Pi))$.

($\Leftarrow$) Assume there is an acyclic dependency graph $\mathcal{G}' = (V', E')$ over the set $\mathcal{A}(\Pi) \cup \{a_\perp, a_\top\} \cup \{c_1, ..., c_m\}$ of activities and such that $\mathcal{G}' \models \pi(L(\Pi)) \cup \Pi^\neg$. Let t_i be a trace in $L(\Pi)$, with $i \in \{1, ..., m\}$, and consider the two constraints $\{a_\perp, b_i^1, ..., b_i^{k_i}\} \rightarrow \{c_i\}$

and $\{c_i\} \to \{a_i^1, ..., a_i^{h_i}, a_\top\}$ in the set $\pi(t_i)$, which are satisfied by $\mathcal{G}'$. Since $\mathcal{G}' \models \Pi^\neg$, from the above we conclude that $\mathcal{G}' \models \{\{b_i^1, ..., b_i^{k_i}\} \to \{c_i\}, \{c_i\} \to \{a_i^1, ..., a_i^{h_i}\}\}$ holds.

Consider now the graph $\mathcal{G} = (V, E)$ where $V = V' \setminus \{c_1, ..., c_m\}$ and where the set of edges is defined as follows. The set E contains the edges $(a_\bot, a)$ and $(a, a_\top)$, for each activity $a \in V$. Moreover, for each $i \in \{1, ..., m\}$, E includes all edges of the form (x, y) such that $\{(x, c_i), (c_i, y)\} \subseteq E'$, $x \in \{b_i^1, ..., b_i^{k_i}\}$, and $y \in \{a_i^1, ..., a_i^{h_i}\}$. Note that, in the light of the above observation, at least one edge of this kind is included in E, so that $\mathcal{G} \models \Pi$ holds. Note also that $\mathcal{G}$ is a dependency graph. Eventually, to conclude the proof, just note that $\mathcal{G}$ is acyclic, as the existence of a cycle in $\mathcal{G}$ would immediately entail the existence of a cycle in $\mathcal{G}'$.

Observe now that the log $L(\Pi)$ is linear, so that we are in the position of applying Corollary 9.8 and Fact 9.10 on the result proven above. By this way, we conclude that Acyclic-CN-Mining$[\{\to\}]$ on input $\mathcal{A}(\Pi) \cup \{a_\bot, a_\top\}$, the empty log, and the set Π has a solution if, and only if, Acyclic-CN-Mining$[\{\not\to\}]$ on input $\mathcal{A}(\Pi) \cup \{a_\bot, a_\top\} \cup \{c_1, ..., c_m\}$, the log $L(\Pi)$, and the set $\Pi^\neg$ has a solution. Eventually, by inspecting the proof of Lemma 9.14, note that the constraints used to prove the NP-hardness of Acyclic-CN-Mining$[\{\to\}]$ are precisely of the form considered here for the set Π, and that the result is established even for logs that are empty. Therefore, we have reduced Acyclic-CN-Mining$[\{\not\to\}]$ to Acyclic-CN-Mining$[\{\to\}]$, so that the problem is hence shown to be NP-hard, too. □

To complete the picture, we now move to the case of arbitrary process models, i.e., of models that are not required to be acyclic. In this context, the picture is easily completed, as negated path constraints can be used to enforce acyclicity.

Theorem 9.17. *The problems* CN-MINING$[\{\rightarrow, \not\rightarrow\}]$, CN-MINING$[\{\rightsquigarrow, \not\rightarrow\}]$, *and* CN-MINING$[\{\not\Rightarrow, \not\rightarrow\}]$ *are* NP*-hard.*

Proof. Let Π be a set of constraints in $\mathcal{C}[\{\rightarrow\}]$ (resp., $\mathcal{C}[\{\rightsquigarrow\}]$, $\mathcal{C}[\{\not\Rightarrow\}]$). Based on Π, we build the set Π' of constraints including all the constraints in Π, plus the novel constraint $\neg(\{a\} \not\rightarrow \{a\})$, for each activity a taken from the underlying set of symbols $\mathcal{A}$. Of course, Π' belongs to $\mathcal{C}[\{\rightarrow, \not\rightarrow\}]$ (resp., $\mathcal{C}[\{\rightsquigarrow, \not\rightarrow\}]$, $\mathcal{C}[\{\not\Rightarrow, \not\rightarrow\}]$). In particular, the novel constraints just enforce that the resulting process model is acyclic. Then, ACYCLIC-CN-MINING$[\{\rightarrow\}]$ (resp., ACYCLIC-CN-MINING$[\{\rightsquigarrow\}]$, ACYCLIC-CN-MINING$[\{\not\Rightarrow\}]$) has a solution on input the set $\mathcal{A}$ of activities, a log L, and the set Π of constraints if, and only if, CN-MINING$[\{\rightarrow, \not\rightarrow\}]$ (resp., CN-MINING$[\{\rightsquigarrow, \not\rightarrow\}]$, CN-MINING$[\{\not\Rightarrow, \not\rightarrow\}]$) has a solution on input $\mathcal{A}$, L, and Π'. The result therefore follows from Lemma 9.14, Lemma 9.15, and Lemma 9.16. □

We now show that the decision version belongs to the complexity class NP. Combined with the NP-hardness results, this entails the corresponding NP-completeness results.

Theorem 9.18. CN-EXISTENCE[S] *is in* NP*, for each set* $\mathtt{S} \subseteq \{\rightarrow, \rightsquigarrow, \not\Rightarrow, \not\rightarrow\}$.

Proof. We need to decide about the existence of an extended causal net $\mathcal{C} = \langle \mathcal{G}, \mathcal{I}, \mathcal{O} \rangle$ over $\mathcal{A}$ such that $\mathcal{C} \vdash L$ and $\mathcal{G} \models \Pi$, where L and Π are the log and the set of constraints, respectively, provided as input. Thus, we can build a non-deterministic Turing machine that guesses a graph $\mathcal{G} = (V, E)$, with $V \subseteq \mathcal{A}$, and checks that $\mathcal{G} \models \Pi$. Moreover, for each trace $t \in L$, the machine guesses a sequence σ_t of binding activities $\langle t[1], ib_1, ob_1 \rangle, ..., \langle t[len(t)], ib_n, ob_n \rangle$ and checks that σ_t is valid w.r.t. $\mathcal{C}$. By Theorem 9.4, we can assume w.l.o.g. that the size of $\mathcal{G}$ is polynomially bounded. Thus, the overall size of the structures guessed by the machine is polynomially bounded.

Moreover, the operations performed on them are feasible in polynomial time. Hence, the problem belongs to NP. For completeness, note that if one restricts the problem to causal nets only, then the problem is still feasible in NP, as the machine has just to additionally check that $\mathcal{I}$ and $\mathcal{O}$ are sets rather than multi-sets as in the extended model. □

9.5 An exact solution approach for computing process models

The complexity analysis we have conducted evidenced that polynomial time algorithms are unlikely to exists for the problem of computing models (and minimal models) of sets of precedence constraints. This bad news calls for the design of sophisticated *exact* solution approaches that perform well in practice, and of *heuristic* methods that further speed up the computation These complementary solution approaches will be explored in the following two sections.

In this section, we start by considering the issue of designing exact solution approaches. To this end, we propose to encode precedence constraints in terms of "standard" *constraints satisfaction problems* (short: CSPs) and to reuse existing constraint programming platforms to compute models of them. Indeed, such platforms have been developed to solve NP-hard problems declaratively specified in terms of CSPs, and embody sophisticated solution algorithms (see, e.g., Apt . [2003]) allowing them to scale over large datasets, as their application in data mining contexts have already demonstrated De Raedt et al. [2008]; Nijssen et al. [2009].

Various kinds of constraints are supported by constraint programming systems in the literature. To our ends, we just need to consider two kinds of constraints defined over binary domains. Let $X_1, ..., X_n$ be n variables in Var, let $U(X_i) = \{0, 1\}$, and let $w_1, ..., w_n, \gamma$ be $n + 1$ real numbers. Then,

(1) A *summation constraint* is an expression of the form $\sum_{i=1}^{n} w_i \times X_i \geq \gamma$. The constraint is satisfied by an assignment θ if $\sum_{i=1}^{n} w_i \times \theta(X_i) \geq \gamma$ holds.

(2) A *reified (summation) constraint* is an expression of the form $\sum_{i=1}^{n} w_i \times X_i \geq \theta \leftrightarrow X$, where X is a variables such that $U(X) = \{0, 1\}$. The constraint is satisfied by an assignment θ if $\sum_{i=1}^{n} w_i \times \theta(X_i) \geq \gamma$ holds if, and only, if $\theta(X) = 1$.

In some cases, among all the possible solutions, we are interested in the one optimizing some given criterium. This gives rise to a *constraint satisfaction optimization problem* (short: CSOP) instance, that is, a quadruple $(Var, U, \mathcal{C}, f)$ where $(Var, U, \mathcal{C})$ is the underlying CSP instance, and where f is a valuation function assigning a real number $f(\theta)$ to each solution θ to $(Var, U, \mathcal{C})$. A solution θ to the CSOP instance is then a solution to the CSP instance such that $f(\theta) \leq f(\theta')$, for each other CSP solution θ'.

In several constraint programming platforms, the function f can be specified in terms of a linear minimization constraint. Let $X_1, ..., X_n$ be n variables in Var, let $U(X_i) = \{0, 1\}$, and let $w_1, ..., w_n$ be n real numbers. Then, a *linear minimization constraint* is an expression of the form $\sum_{i=1}^{n} w_i \times X_i$. Under this function, the value associated with a CSP solution θ is $f(\theta) = \sum_{i=1}^{n} w_i \times \theta(X_i)$.

9.5.1 CSP encoding for precedence constraints

Let $\mathcal{A} = \{a_1, ..., a_n\}$ be any set of activities, and let Π be any set of precedence constraints such that $\mathcal{A}(\Pi) \subseteq \mathcal{A}$. Figure 9.5 illustrates an algorithm, named PC-TOCSP, to encode $\mathcal{A}$ and Π into a CSP instance $(Var, U, \mathcal{C})$. The instance defined by the algorithm is such that, for each pair a_i and a_j of activities taken from $\mathcal{A}$, Var contains an "edge" variable, denoted as $e[a_i, a_j]$, plus $n + n^2$ "path" variables,

Input: A set $\mathcal{A} = \{a_1, ..., a_n\}$ of activities,
a set Π of precedence constraints with $\mathcal{A}(\Pi) \subseteq \mathcal{A}$, and
the TYPE $\in$ {ARBITRARY, ACYCLIC} of the problem;
Output: A CSP instance $(Var, U, \mathcal{C})$;

1. **let** $Var = \{e[a_i, a_j], p[a_i, a_j]^\ell, p[a_i, a_k, a_j]^\ell \mid a_i, a_j, a_k \in \mathcal{A}, \ell \in \{1, ..., n\}\}$;
2. **let** $U(X) = \{0, 1\}$ for each $X \in Var$;
3. **let** $\mathcal{C} = \emptyset$;
4. **for each** edge constraint $S \to a_j$ in Π **do**
   $\mathcal{C} := \mathcal{C} \cup \{\sum_{a_i \in S} e[a_i, a_j] \geq 1\}$;
5. **for each** path constraint $S \rightsquigarrow a_j$ in Π **do**
   $\mathcal{C} := \mathcal{C} \cup \{\sum_{a_i \in S} p[a_i, a_j]^n \geq 1\}$;
6. **for each** negated edge constraint $\neg S \to a_j$ in Π **do**
   $\mathcal{C} := \mathcal{C} \cup \{e[a_i, a_j] = 0 \mid a_i \in S\}$;
7. **for each** negated path constraint $\neg S \rightsquigarrow a_j$ in Π **do**
   $\mathcal{C} := \mathcal{C} \cup \{p[a_i, a_j]^n = 0 \mid a_i \in S\}$;
8. **if** TYPE=ACYCLIC **then**
   $\mathcal{C} := \mathcal{C} \cup \{p[a_i, a_j]^n + p[a_j, a_i]^n \leq 1 \mid \{a_i, a_j\} \subseteq \mathcal{A}\}$;
9. **for each** pair of distinct activities a_i and a_j **do**
   $\mathcal{C} := \mathcal{C} \cup \{e[a_i, a_j] \geq 1 \leftrightarrow p[a_i, a_j]^1\}$;
   $\mathcal{C} := \mathcal{C} \cup \{e[a_i, a_k] + p[a_k, a_j]^{\ell-1} \geq 2 \leftrightarrow p[a_i, a_k, a_j]^\ell \mid a_k \in \mathcal{A}, \ell \in \{2, ..., n\}\}$;
   $\mathcal{C} := \mathcal{C} \cup \{\sum_k p[a_i, a_k, a_j]^\ell \geq 1 \leftrightarrow p[a_i, a_j]^\ell \mid \ell \in \{2, ..., n\}\}$;
10. **for each** activity a_i, with $i \notin \{1, n\}$ **do**
    $\mathcal{C} := \mathcal{C} \cup \{p[a_1, a_i]^n + p[a_i, a_n]^n \geq 2\}$;
11. $\mathcal{C} := \mathcal{C} \cup \{\sum_{a_k \in \mathcal{A}} p[a_k, a_1]^n + \sum_{a_k \in \mathcal{A}} p[a_1, a_k]^n \leq 0\}$;
12. **return** $(Var, U, \mathcal{C})$;

Figure 9.5: Algorithm PCtoCSP.

denoted as $p[a_i, a_j]^\ell$ and $p[a_i, a_k, a_j]^\ell$, with $a_k \in \mathcal{A}$ and $\ell \in \{1, ..., n\}$. All variables are defined over the binary domain $\{0, 1\}$.

To help the intuition, we anticipate that $e[a_i, a_j]$ is meant to encode the existence of an edge from a_i to a_j, while $p[a_i, a_j]^\ell$ is meant to encode the existence of a path of length at most ℓ.

Steps 4, 5, 6, and 7 in the algorithm in Figure 9.5 encode in a straightforward manner edge, path, negated edge, and negated path constraints of Π, respectively. For example, for each edge constraint $S \to T$ occurring in Π, the constraint $\sum_{a_i \in S} \sum_{a_j \in T} e[a_i, a_j] \geq 1$ is added to $\mathcal{C}$ stating that at least one of the edge variables from any activity in S to any activity in T must be mapped to 1.

Step 8 is responsible of enforcing acyclicity if the additional input parameter TYPE is ACYCLIC. Indeed the constraint states that for each pair of distinct activities a_i and a_j, at most one path variable in $\{p[a_i, a_j]^n, p[a_j, a_i]^n\}$ can be set to 1.

Step 9, is responsible of providing the semantics to the variable of the form $p[a_i, a_j]^n$. Indeed, it enforces that $p[a_i, a_j]^n$ is mapped to 1 if, and only if, $e[a_i, a_j]$ is mapped to 1 or there is an intermediate activity a_k with $e[a_i, a_k] + p[a_k, a_j]^{n-1} \geq 1$.

Note that, the definition is recursive, and is explicitly unfolded in the encoding as the maximum length of any path is n. The base case occurs for variables of the form $p[a_i, a_j]^1$, whose value coincides with that of the corresponding edge variables. More generally, the definition exploits the auxiliary variables having the form $p[a_i, a_k, a_j]^\ell$, encoding the existence of an edge from a_i to a_k and of a path of length $\ell - 1$ at most from a_k to a_j.

Finally, step 10 and 11 enforce the conditions required to hold over dependency graphs, where a_1 and a_n play the role of the starting and terminating activity, respectively. In particular, each activity a_i must occur in a path connecting a_1 and a_n, and a_1 (resp., a_n) does not have ingoing (resp., outgoing) edges.

To analyze the algorithm, for any solution θ to the CSP $(Var, U, \mathcal{C})$ computed by PCToCSP (on input $\mathcal{A}, \Pi$ and TYPE), let $\mathcal{G}_\theta = (\mathcal{A}, E)$ be the graph such that (a_i, a_j) is in E if, and only if, $\theta(e[a_i, a_j]) = 1$. With this notation, the crucial properties of the encodings are stated below.

Theorem 9.19. *Let θ be a solution to $(Var, U, \mathcal{C})$. Then, $\mathcal{G}_\theta$ is a dependency graph such that $\mathcal{G}_\theta \models \Pi$. Moreover, if* TYPE=ACYCLIC*, then $\mathcal{G}$ is acyclic.*

Theorem 9.20. *Let $\mathcal{G}$ be a dependency graph (resp., acyclic dependency graph) such that $\mathcal{G} \models \Pi$. If* TYPE=ARBITRARY *(resp.,* ACYCLIC*), then there is a solution θ to $(Var, U, \mathcal{C})$ such that $\mathcal{G} = \mathcal{G}_\theta$.*

By combining Theorem 9.19, Theorem 9.20 and Corollary 9.8 (with the function `computeBindings` discussed in the previous section), we get that PCToCSP is an effective method to solve CN-Mining and Acyclic-CN-Mining over linear logs. In order to deal with arbitrary logs, we need and extension of the algorithm illustrated in Figure 9.5. This extension is the algorithm DGToCSP illustrated in Figure 9.6.

Input: A log L, a set Π of precedence constraints over $\mathcal{A}(L) = \{a_1, ..., a_n\}$, and the TYPE of the problem;
Output: A CSP instance $(Var, U, \mathcal{C})$;

1. **let** $(Var_1, U_1, \mathcal{C}_1)$ be the output of PCToCSP($\pi(\bar{L})$,ACYCLIC);
2. **let** $(Var_2, U_2, \mathcal{C}_2)$ be the output of PCToCSP(Π,TYPE);
3. **let** $Var = Var_1 \cup Var_2 \cup \{eProjected[a_i, a_j] \mid a_i, a_j \in \mathcal{A}\}$;
4. **let** $U(X) = \{0, 1\}$ for each $X \in Var$;
5. **let** $\mathcal{C} = \mathcal{C}_1 \cup \mathcal{C}_2$;
6. **for each** pair of distinct activities a_i and a_j **do**
   $\mathcal{C} := \mathcal{C} \cup \{e[a_i, a_j] \geq 1 \leftrightarrow eProjected[a_i, a_j]\}$;
   $\mathcal{C} := \mathcal{C} \cup \{\sum_{h,k} e[a_i\langle h\rangle, a_j\langle k\rangle] \geq 1 \leftrightarrow eProjected[a_i, a_j]\}$;
7. **return** $(Var, U, \mathcal{C})$;

Figure 9.6: Algorithm DG-DiscoveryToCSP.

Corollary 9.21. Let L be a log where no trace contains multiple occurrences of the same activity. Let $\mathcal{G}$ be a graph (resp., acyclic graph) over $\mathcal{A}(L)$, and Π be a set of precedence constraints over $\mathcal{A}(L)$. Then, the followings are equivalent:

(1) $\mathcal{G}$ is a solution to CN-Mining (resp., Acyclic-CN-Mining) on input L and Π.

(2) $\mathcal{G} = \mathcal{G}_\theta$ where θ is a solution to the *CSP* computed by PCToCSP on input $\pi(L) \cup \Pi$ with TYPE=ARBITRARY (resp., ACYCLIC).

The algorithm receives as input a set $\mathcal{A}$ of activities, a log L with $\mathcal{A}(L) \subset \mathcal{A}$, a set Π of precedence constraints with $\mathcal{A}(\Pi) \subset \mathcal{A}$, and a TYPE.

First, it computes the CSP instances $(Var_1, U_1, \mathcal{C}_1)$ and $(Var_2, U_2, \mathcal{C}_2)$ produced by PCToCSP on input $(\mathcal{A}(unfold(L)), \Pi(unfold(L))$,ACYCLIC) and $(\mathcal{A}, \Pi$,TYPE), respectively. Then, a novel CSP instance $(VAR, U, \mathcal{C})$ is defined as the union of these two instances, and the set $\mathcal{C}$ of constraints is eventually enlarged by including some constraints that server to relate the variables in Var_1 with the variables in Var_2. In particular, observe that the CSP instance $(Var_1, U_1, \mathcal{C}_1)$ is build given as input the set $\mathcal{A}(unfold(L))$ of fresh virtual activities, and recall from section 8.4 that any activity in $\mathcal{A}(unfold(L))$ has the form where a is an activity in $\mathcal{A}$ and k is the number of its occurrences in some trace of the log (up to some given position). Then the constraints added in step 6 guarantee that the variable $e[a_i, a_j] \in Var_2$ is mapped to 1, for each pair of activities a_i and a_j if, and only if, there is at least one variable

of the form $e[a_i\langle h\rangle, a_j\langle k\rangle] \in Var_1$ being mapped to 1. In words, variables in Var_2 are defined as the "projection" of the corresponding variables in Var_1, i.e., by just stripping off the subscripts denoting the number of occurrences.

Proposition 9.22. *Let L be a log, let $\mathcal{G}$ be a graph over $\mathcal{A}(L)$, and let Π be a set of precedence constraints over $\mathcal{A}$. Then, the followings are equivalent:*

(1) *$\mathcal{G}$ is a solution to* CN-Mining *(resp.,* Acyclic-CN-Mining*) on input L and Π.*

(2) *$\mathcal{G} = \mathcal{G}_\theta$ where θ is a solution to the* CSP *computed by* DG-DiscoveryToCSP *on input L, Π, and* Type=arbitrary *(resp.,* acyclic*).*

Proof. Note that the constraints in step 6 guarantees that $\mathcal{G}_\theta$ is the folding of the graph $\bar{\mathcal{G}}_\theta$, for each solution θ. Thus, the result follows from the correctness of algorithm PCtoCSP (cf. Theorem 9.19 and Theorem 9.20) guaranteeing that $\bar{\mathcal{G}}_\theta \models \pi(\bar{L})$, and by the application of Corollary 9.8. □

9.5.2 Structural optimization

Several process models (in principle, up to exponentially many) might be built as solutions to CN-Mining and Acyclic-CN-Mining given a log and a set of precedence constraints provided as input. Returning an arbitrary one is usually not enough in practical applications, and we might rather want to formalize an objective function over the candidate solutions, as compute the best one over them. In order to formalize an optimality criterium, we next propose to focus on those models whose dependencies are maximal w.r.t. some (reflexive and transitive) order Let $\mathcal{A}$ denote, as usual, a given set of activities. Moreover, assume that a *weighting function* $w : \mathcal{A}\times\mathcal{A} \longmapsto \mathbb{R}$ is given, which associates a value $w(e) \in \mathbb{R}$ with each element $e \in \mathcal{A}\times\mathcal{A}$, Then, for any pair of directed graphs $\mathcal{G} = (V, E)$ and $\mathcal{G}' = (V', E')$ over $\mathcal{A}$, we write

$\mathcal{G} \sqsubseteq_w \mathcal{G}'$ if $\sum_{e \in E} w(e) \leq \sum_{e' \in E'} w(e')$. Hence, $\mathcal{G} \sqsubseteq_w \mathcal{G}'$ means that the overall weight associated with $\mathcal{G}$ is at most the overall weight associated with $\mathcal{G}'$, A solution $\mathcal{C} = \langle \mathcal{G}, \mathcal{I}, \mathcal{O} \rangle$ to CN-Mining (resp., Acyclic-CN-Mining) is $\sqsubseteq_w$ $-maximal$ if, for each other solution $\mathcal{C}' = \langle \mathcal{G}', \mathcal{I}', \mathcal{O}' \rangle$ to CN-Mining (resp., Acyclic-CN-Mining), it holds that $\mathcal{G}' \sqsubseteq_w \mathcal{G}$. In words, by considering $\sqsubseteq_w$ $-maximal$ solutions, we look for dependency graphs having associated the maximum total weight.

Example 9.23. As a very simple weighting function, one might consider the constant function $1 : \mathcal{A} \times \mathcal{A} \longmapsto \{-1\}$ assigning a unitary negative weight to each element in $\mathcal{A} \times \mathcal{A}$. Then, $\sqsubseteq_1$ $-maximal$ solutions are those with the minimum possible number of edges. Consider, for instance, the graphs $\mathcal{G}_2$ and $\mathcal{G}_3$ depicted in Figure 8.1. It is easily seen that $\mathcal{G}_3 \sqsubseteq_1 \mathcal{G}_2$, because $\mathcal{G}_2$ consists of 5 edges, while $\mathcal{G}_3$ consists of 6 edges. As a further example, consider the graph $\mathcal{G}_1$ depicted in Figure 8.1, which consists of 5 edges. Therefore, $\mathcal{G}_1 \sqsubseteq_1 \mathcal{G}_2$ and $\mathcal{G}_2 \sqsubseteq_1 \mathcal{G}_1$ hold. $\lhd$

By using linear maximization constraints on top of the rewriting in Figure 9.6, we may compute $\sqsubseteq_w$ $-maximal$ solutions. In particular, if we consider solutions θ to the CSOP $(Var, U, \mathcal{C}, \sum_{a_i, a_j} w((a_i, a_j)) \times e[a_i, a_j])$, where $(Var, U, \mathcal{C})$ is the CSP computed by DG-DiscoveryToCSP, then solutions in point (2) of Theorem 9.6 are $\sqsubseteq_w$ $-maximal$ ones. In fact, this is a simple consequence of the semantics of linear maximization constraints. Eventually, we stress that as a useful weighting function, we might want to consider here the *causal score* defined in Section 9.6.

9.6 Classes of Tractable Precedence Constraints and Algorithms

In this section, we define two algorithms for efficiently solving CN-Mining over the classes $\mathcal{C}[\{\rightarrow, \rightsquigarrow, \not\rightarrow\}]$ and $\mathcal{C}[\{\not\rightarrow\}]$, respectively. The algorithms have been conceived as to make them still applicable over larger classes of constraints, by providing heuris-

tic solution methods in these (NP-hard) cases. The efficiency of the methods and their efficacy as heuristics will be eventually assessed in Section 9.7.

Hereinafter, we assume that a set $\mathcal{A}$ of activities is given, together with a log L such that $\mathcal{A}(L) \subseteq \mathcal{A}$ and with a set Π of precedence constraints such that $\mathcal{A}(\Pi) \subseteq \mathcal{A}$. Accordingly, to simplify the notation, we shall omit to indicate $\mathcal{A}$, L, and Π, unless we want to explicitly point out some dependency on some of them. Moreover, we shall denote by $\Pi^{\rightarrow}$ and $\Pi^{\rightsquigarrow}$ (resp., $\Pi^{\not\rightarrow}$ and $\Pi^{\not\rightsquigarrow}$) the sets of all positive (resp., negated) edge and path constraints in Π, respectively.

Both algorithms are based on a succession of graph manipulations, i.e., insertions and deletions of edges, starting with an initial (dependency) graph built from the log. In order to facilitate reasoning about such graph manipulations, for any directed graph $\mathcal{G} = (V, E)$ and for any set $E' \subseteq V \times V$ of edges, we define $\mathcal{G} \oplus E'$ as the graph $(V, E \cup E')$, and $\mathcal{G} \ominus E'$ as the graph $(V, E \setminus E')$. Moreover, we observe that, while performing these operations, it is practically relevant to (resp., exclude) only those edges that very likely (resp., hardly) witness the existence of true causal relationships. In order to provide a formal measure of the 'quality' of an edge, we consider here the notion of *causal score* inspired by the works of [Weijters and van der Aalst, 2001], [Weijters and van der Aalst, 2003], and [Weijters *et al.*, 2006].

Let δ be a real number with $0 < \delta < 1$. Then, the causal score (w.r.t. δ) is defined as the function $\mathbf{cs}_\delta : \mathcal{A} \times \mathcal{A} \mapsto \mathbb{R}$ such that $\mathbf{cs}_\delta(a_i, a_j) = D(a_i, a_j)/|\{t \in L \mid a_i = t[k], \text{ for some index } k\}|$, and where:

$$D(a_i, a_j) = \sum_{t \in L | a_i \text{precedes } a_j \text{ in } t} \delta^{\mathbf{cs}^+(t,a_i,a_j)} - \sum_{t \in L | a_j \text{precedes } a_i \text{ in } t} \delta^{\mathbf{cs}^-(t,a_i,a_j)},$$

with

- $\mathbf{cs}^+(t, a_i, a_j) = \min\{k - h - 1 \mid a_i = t[h] \wedge a_j = t[k] \wedge h < k\}$ being the minimum

number of symbols between an occurrence of a_i and a subsequent occurrence of a_j, and

- $\mathbf{cs}^{-}(t, a_i, a_j) = \max\{k-h-1 \mid a_i = t[k] \wedge a_j = t[h] \wedge h < k\}$ being the minimum number of symbols between an occurrence of a_j and a subsequent occurrence of a_i.

To illustrate the above definition, note that, for each trace $t[1]...t[n]$, $D(a_i, a_j)$ is incremented by a term δ^{k-h-1} if a_i occurs $k-h$ positions before an occurrence of a_j (and this is the minimum distance between the symbols in the given order), and decremented by the same term (in absolute value) if a_i occurs $k-h$ positions after an occurrence of a_j (again, with this distance being the minimum one). Moreover, the positive and the negative terms exponentially decrease at the growing of the distance between a_i and a_j in the traces. Note that $-1 \leq \mathbf{cs}_\delta(a_i, a_j) \leq 1$ holds, since $0 < \delta < 1$.

9.6.1 Precedence constraints without negated paths

We start by illustrating an algorithm to solve CN-Mining over $\mathcal{C}[\{\rightarrow, \rightsquigarrow, \not\rightarrow\}]$. The algorithm is designed so that, over linear logs (cf. Theorem 9.6), a causal net is always returned, whenever a solution in fact exists. Instead, over arbitrary logs, the algorithm might well return an extended causal net.

Conceptually the algorithm is organized in three main phases:

- First, a dependency graph is built by guaranteeing that each trace in the given log can be supported by properly enriching the graph with suitable bindings. While doing so, we avoid to make use of edges that are forbidden because of negated edge constraints. If this is not possible, then (we shall show that) no solution exists at all.

- Second, we incrementally add to the dependency graph edges that are not forbidden, until all positive edge and path constraints are satisfied. Note that, in principle, all edges that are not forbidden might be added to the dependency graph (and if some constraint is still violated, then we are again guaranteed that no solution exists). Therefore, in order to minimize the size of the resulting model (and the probability of introducing spurious dependencies), edges are selected in the algorithm based on their causal scores.
- Finally, the dependency graph resulting from the above manipulations is equipped with bindings guaranteeing that all the traces can be supported. This leads to a causal net that is returned as output.

These steps are now detailed in the rest of the section.

Precedence Graphs

The starting point of the algorithm is the construction of the *precedence graph* $\texttt{PG}(L, \Pi)$ over $\mathcal{A}(L)$. This graph is built by avoiding the inclusion of the edges that are forbidden because of negated edge constraints in Π, i.e., the edges in $\texttt{FE}(\Pi) = \{(x, y) \mid x \in S, y \in T, \neg(S \rightarrow T) \in \Pi^{\not\rightarrow}\}$. We start with an exemplification.

Example 9.24. *Consider a log L including only the trace abcde, and assume that $\Pi = \{\neg(\{c\} \rightarrow \{d\})\}$. The precedence graph $\texttt{PG}(L, \Pi)$ is illustrated in the left part of Figure 9.7. Intuitively, each activity $x \in \{a, b, c, d, e\}$ has an edge incoming (resp., outgoing) to any activity that precedes (resp., follows) x in some trace. However, we avoid the edges that are forbidden according to the constraints. In particular, the graph does not contain the edge (c, d), and the connectivity of d is guaranteed via the edge (b, d), i.e., d is reached by the node closest to it in the trace abcde and for which no violation in $\Pi^{\not\rightarrow}$ occurs. Moreover, c can reach the final activity via the edge (c, e).*

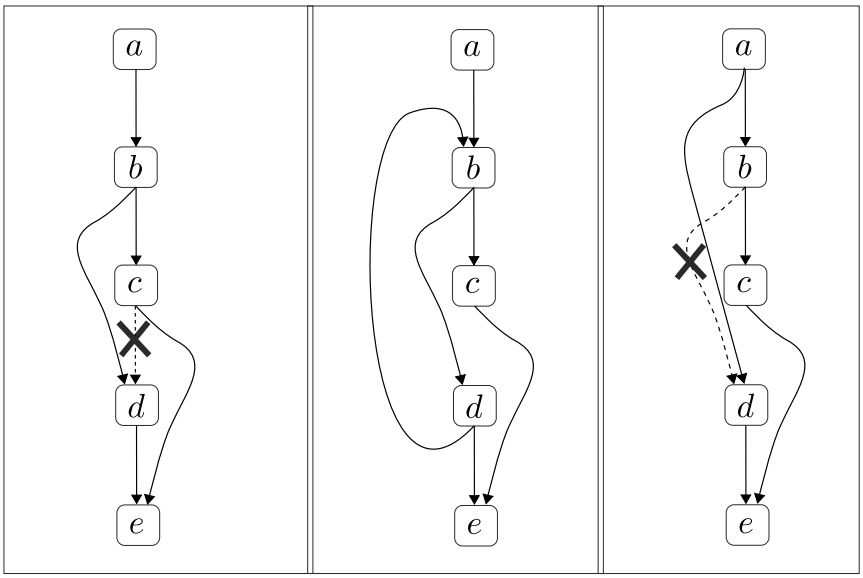


Figure 9.7: Precedence graphs for the examples in Section 9.6.

◁

Formally, $\texttt{PG}(L, \Pi) = (V, E)$ is defined as the directed graph where $V = \mathcal{A}(L)$ and where the set E of its edges is built as follows. For each trace $t \in L$ and for each $i \in \{2, ..., len(t)\}$ (resp., $i \in \{1, ..., len(t) - 1\}$), if there is an index $j \in \{1, ..., i - 1\}$ (resp., $h \in \{i + 1, ..., len(t)\}$) such that $(t[j], t[i]) \notin \texttt{FE}(\Pi)$ (resp., $(t[i], t[h]) \notin \texttt{FE}(\Pi)$), then E contains the edge $(t[j^*], t[i])$ (resp., $(t[i], t[h^*])$) witnessing that the property holds (i.e., $(t[j^*], t[i]) \notin \texttt{FE}(\Pi)$ (resp., $(t[i], t[h^*]) \notin \texttt{FE}(\Pi)$)) and having the highest causal score.[1] No further edge is in E.

Lemma 9.25. Assume that $\texttt{PG}(L, \Pi) \not\models \pi(L)$ holds. Then, there is no dependency graph $\mathcal{G}$ such that $\mathcal{G} \models \pi(L) \cup \Pi$. Otherwise, i.e., if $\texttt{PG}(L, \Pi) \models \pi(L)$, then $\texttt{PG}(L, \Pi)$ is a dependency graph over $\mathcal{A}(L)$ such that $\texttt{PG}(L, \Pi) \models \Pi^{\not\rightarrow}$.

Proof. Assume that $\texttt{PG}(L, \Pi) \not\models \pi(L)$. Then, there is a trace $t \in L$ such that $\texttt{PG}(L, \Pi) \not\models \pi(t)$. By Definition 9.5, there are two possible cases. First, there might be an index $i \in \{2, ..., len(t)\}$ such that $\{t[1], ..., t[i - 1]\} \rightarrow \{t[i]\}$ is not satisfied by

[1] We will say that the edges $(t[j^*], t[i])$ and $(t[i], t[h^*])$ are supported by t.

$\texttt{PG}(L, \Pi)$. By construction of $\texttt{PG}(L, \Pi)$, this means that $(t[j], t[i]) \in \texttt{FE}(\Pi)$ holds, for each $j \in \{1, ..., i-1\}$. Consider then a dependency graph $\mathcal{G}$ such that $\mathcal{G} \models \pi(L)$. Note that $\mathcal{G}$ must include an edge $(t[j^*], t[i])$, with $j^* \in \{1, ..., i-1\}$, in order to satisfy the above constraint. However, $(t[j^*], t[i])$ is in $\texttt{FE}(\Pi)$, and there is a negated edge constraint $\neg(S \rightarrow T) \in \Pi$ such that $t[j^*] \in S$ and $t[i] \in T$. So, if $\mathcal{G} \models \pi(L)$, then $\mathcal{G} \not\models \Pi$.

Assume now that $\texttt{PG}(L, \Pi) \models \pi(L)$. We first observe that $\texttt{PG}(L, \Pi)$ contains the two activities $a_{\perp}$ and $a_{\top}$, playing the role of the starting and terminating activity. In particular, $a_{\perp}$ and $a_{\top}$ have no ingoing and outgoing edges, respectively. Consider then any other activity a occurring in $\texttt{PG}(L, \Pi)$, and note that there is a trace $t \in L$ and an index $i \in \{2, ..., len(t)-1\}$ such that $t[i] = a$. Since $\texttt{PG}(L, \Pi) \models \pi(L)$, we are then guaranteed about the existence of two edges having the form $(t[j], t[i])$ and $(t[i], t[h])$, with $j \in \{1, ..., i-1\}$ and $h \in \{i+1, ..., len(t)\}$. By structural induction on the index i, it then follows that $t[i]$ occurs in a path connecting $t[1] = a_{\perp}$ to $t[len(t)] = a_{\top}$. Hence, $\texttt{PG}(L, \Pi)$ satisfies all the conditions for being a dependency graph over $\mathcal{A}(L)$. Then, in order to conclude the proof, we need to show that $\texttt{PG}(L, \Pi) \models \Pi^{\not\rightarrow}$. Indeed, assume by contradiction that a negated edge constraint $\neg(S \rightarrow T)$ exists in Π such that $x \in S$, $y \in T$, and the edge (x, y) is in $\texttt{PG}(L, \Pi)$. Hence, $(x, y) \in \texttt{FE}(\Pi)$, which is impossible as all edges of $\texttt{PG}(L, \Pi)$ do not belong to $\texttt{FE}(\Pi)$, by construction. □

Positive Precedence Constraints

According to the above result, precedence graphs can be used as a preliminary representation of inter-activity dependencies. However, these graphs do not guarantee that positive constraints are satisfied. Therefore, we define a method for identifying the edges needed to satisfy positive constraints in Π.

Example 9.26. *Consider the precedence graph* $\texttt{PG}(L, \Pi)$ *discussed in Example 9.24*

and assume that Π *also contains the constraint* $\{d\} \rightsquigarrow \{b\}$. *Note that* $\texttt{PG}(L, \Pi)$ *does not satisfy the positive constraint* $\{d\} \rightsquigarrow \{b\}$. *Thus, we have to update the graph by including a path starting from* d *and terminating into* b *and where the edge* (c, d) *does not occur in it. Of course, in this case we can just simply add an edge from* d *to* b, *as it is shown in the central part of Figure 9.7.* $\lhd$

Let $\mathcal{G} = (V, E)$ be a dependency graph with $V \supseteq \mathcal{A}(\Pi)$. For each pair of nodes $x, y \in V$, define the *weight of* (x, y) *for* $\mathcal{G}$ *w.r.t.* δ, as the real number $w_\delta(\mathcal{G}, x, y)$ such that[2]:

$$w_\delta(\mathcal{G}, x, y) = \begin{cases} 0 & \text{if } (x, y) \in E \\ +\infty & \text{if } x = a_\top \text{; or } y = a_\bot \text{; or } x \in S,\ y \in T, \text{ and } \neg(S \rightarrow T) \in \Pi \\ 2 - \mathbf{cs}_\delta(x, y) & \text{in the remaining cases} \end{cases}$$

If $a_1, ..., a_h$ is a sequence of nodes forming a path in $\mathcal{G}$, with $h \geq 2$, then we define its weight $w_\delta(\mathcal{G}, a_1, ..., a_h)$ as the value $\sum_{i=1}^{h-1} w_\delta(\mathcal{G}, a_i, a_{i+1})$. Note that $w_\delta(\mathcal{G}, a_1, ..., a_h) \geq 0$, because $\mathbf{cs}_\delta(x, y) \leq 1$ holds, for each pair x, y. Moreover, for each path constraint $S \rightsquigarrow T$ (resp., edge constraint $S \rightarrow T$) in Π, we define $\texttt{BestPath}_\delta(\mathcal{G}, S \rightsquigarrow T) = a_1, ..., a_h$ (resp. $\texttt{BestEdge}_\delta(\mathcal{G}, S \rightarrow T) = a_1, a_2$) as the minimum-weight path (resp., minimum-weight edge) such that $a_1 \in S$ and $a_h \in T$. Note that, if there is a precedence constraint $S \rightsquigarrow T$ (resp., $S \rightarrow T$) such that the weight of $\texttt{BestPath}_\delta(\mathcal{G}, S \rightsquigarrow T)$ (resp., $\texttt{BestEdge}_\delta(\mathcal{G}, S \rightarrow T)$) is $+\infty$, then all the paths (resp., edges) connecting any activity in S to any activity in T must include an edge that cannot occur in a model of $\Pi^{\not\Delta}$. Hence, the following is immediately established.

Lemma 9.27. Let $\mathcal{G}$ be a graph such that $\mathcal{G} \models \Pi^{\not\Delta}$. If there is a path constraint $S \rightsquigarrow T$ (resp., edge constraint $S \rightarrow T$) such that the weight of $\texttt{BestPath}_\delta(\mathcal{G}, S \rightsquigarrow T)$ (resp., $\texttt{BestEdge}_\delta(\mathcal{G}, S \rightarrow T)$) is $+\infty$, then CN-Mining has no solution (on $\mathcal{A}$, L,

[2]Here, $+\infty$ stands for any large enough positive real number, e.g., $+\infty > |\mathcal{A}|^2 \times \max_{(x,y)\in E} \mathbf{cs}_\delta(x, y)$.

Input: A set $\mathcal{A}$ of activities, with $a_\perp$ ($a_\top$) being the starting (terminating) one, a log L with $\mathcal{A}(L) \subseteq \mathcal{A}$, and a set $\Pi \in \mathcal{C}[\{\rightarrow, \rightsquigarrow, \not\rightarrow\}]$ of precedence constraints with $\mathcal{A}(\Pi) \subseteq \mathcal{A}$;
Parameters: Real numbers $1 > \delta > 0$ and $1 \geq \tau \geq 0$;
Output: A triple $\langle \mathcal{G}, \mathcal{I}, \mathcal{O} \rangle$, or 'no';

1. $\Pi := \Pi \cup \{\{a_\perp\} \rightsquigarrow \{a\}, \{a\} \rightsquigarrow \{a_\top\} \mid a \in \mathcal{A} \setminus \mathcal{A}(L)\}$;
2. **if** $\texttt{PG}(L, \Pi) \not\models \pi(L)$ **then return** 'no' (and `HALT`);
3. **let** $\mathcal{G}_1 = (\mathcal{A}, E_1)$ be the graph where E_1 consists of the edges in $\texttt{PG}(L, \Pi)$, $k := 1$;
4. $\mathcal{G}_1 := \mathcal{G}_1 \ominus \{(x, y) \mid (x, y)$ is supported by (at least) $\tau \times |L|$ traces$\}$; (*see Footnote 1*)
5. $\mathcal{F} := \{S \rightarrow T \in \Pi \mid \mathcal{G}_k \not\models S \rightarrow T\}$;
6. **while** $\mathcal{F} \neq \emptyset$ **do**
7. **let** $S \rightarrow T$ be in $\mathcal{F}$, and **let** $\texttt{BestEdge}_\delta(\mathcal{G}_k, S, T) = a_1, a_2$;
8. **if** $w_\delta(\mathcal{G}_k, a_1, a_2) \geq +\infty$ **then return** 'no' (and `HALT`);
9. $\mathcal{G}_{k+1} := \mathcal{G}_k \oplus \{(a_1, a_2)\}$, $k := k + 1$;
10. $\mathcal{F} := \{S \rightarrow T \in \Pi \mid \mathcal{G}_k \not\models S \rightarrow T\}$;
11. **end while**
12. $\mathcal{F} := \{S \rightarrow T \in \Pi \mid \mathcal{G}_k \not\models S \rightsquigarrow T\}$;
13. **while** $\mathcal{F} \neq \emptyset$ **do**
14. **let** $S \rightsquigarrow T$ be in $\mathcal{F}$, and **let** $\texttt{BestPath}_\delta(\mathcal{G}_k, S \rightsquigarrow T) = a_1, ..., a_h$;
15. **if** $w_\delta(\mathcal{G}_k, a_1, ..., a_h) \geq +\infty$ **then return** 'no' (and `HALT`);
16. $\mathcal{G}_{k+1} := \mathcal{G}_k \oplus \{(a_i, a_{i+1}) \mid i \in \{1, ..., h-1\}\}$, $k := k + 1$;
17. $\mathcal{F} := \{S \rightsquigarrow T \in \Pi \mid \mathcal{G}_k \not\models S \rightsquigarrow T\}$;
18. **end while**
19. **return** `computeBindings`($\mathcal{G}_k$,L); (* see Figure 9.2*)

Figure 9.8: Algorithm COMPUTE-CN (on $\mathcal{C}[\{\rightarrow, \rightsquigarrow, \not\rightarrow\}]$).

and Π).

If the hypothesis in the above lemma does not hold, in order to satisfy a path constraint $S \rightsquigarrow T$ (resp., edge constraint $S \rightarrow T$) we can just update the graph $\mathcal{G}$ as to include $\texttt{BestPath}_\delta(\mathcal{G}, S \rightsquigarrow T)$ (resp., $\texttt{BestEdge}_\delta(\mathcal{G}, S \rightarrow T)$) as a path (resp., an edge).

Putting Things Together

Now that we have discussed all the salient ingredients, we can illustrate the algorithm COMPUTE-CN shown in Figure 9.8.

The algorithm starts in step 1 by adding to Π a set of path constraints stating that each activity $a \in \mathcal{A} \setminus \mathcal{A}(L)$, i.e., not occurring in the log, has still to occur in a path from the starting activity to the terminating one. Step 2 is responsible for checking whether the precedence graph "supports" the log. Note that we directly

check the satisfaction of the constraints induced by the log. Indeed, if this graph does not satisfy the condition, then we report that no solution exists at all.

In step 3, a graph $\mathcal{G}_1$ is initially built over the nodes in $\mathcal{A}$ and the edges in $\texttt{PG}(L, \Pi)$. In the subsequent steps, $\mathcal{G}_k$ denotes the graph obtained from $\mathcal{G}_1$ after having performed $k-1$ manipulations on it. In fact, the process starts with step 4, which is a heuristic step that removes any edge that is not supported by a sufficient number of traces, i.e., by $\tau \times |L|$ traces, where τ is a threshold received as an additional input parameter. In fact, it can be checked that for $\tau = 0$, this is immaterial and the graph remains unchanged. Then, step 5–11 (resp., 12–18) are responsible for adding a number of edges to the precedence graph, as to satisfy all edge constraints (resp., path constraints), according to the strategy described in Section 9.6.1. A failure in this step leads the algorithm to exit while reporting that no solution exists at all.

Finally, a (possibly extended) causal net $\langle \mathcal{G}, \mathcal{I}, \mathcal{O} \rangle$ is built and returned as output. This latter step is carried out by the function in Figure 9.2 that just implements the strategy of including an input (resp., output) binding for each trace t, and of defining this binding with all predecessor (resp., successor) activities in t. Moreover, in order to formally guarantee that $\langle \mathcal{G}, \mathcal{I}, \mathcal{O} \rangle$ is a (possibly extended) causal net, we add the binding I_a (resp., O_a) to $\mathcal{I}(a)$ (resp., $\mathcal{O}(a)$), for each activity a in $\mathcal{G}$, being defined as the union of all incoming (resp., outgoing) edges.

The correctness of the whole algorithm is stated next. Note that the result explicitly differentiates the case when L is a linear log from the case when L is not linear. In the former case we are guaranteed that the output $\langle \mathcal{G}, \mathcal{I}, \mathcal{O} \rangle$ returned by the algorithm is a causal net, whereas in the latter case, $\langle \mathcal{G}, \mathcal{I}, \mathcal{O} \rangle$ might well be an extended causal net. In both cases, however, $\langle \mathcal{G}, \mathcal{I}, \mathcal{O} \rangle \vdash L$ and $\mathcal{G} \models \Pi$ hold.

Theorem 9.28. *The following properties hold on* COMPUTE-CN*, receiving as input*

$\mathcal{A}$, L, $\Pi \in \mathcal{C}[\{\rightarrow, \rightsquigarrow, \not\rightarrow\}]$, the parameter δ, and for $\tau = 0$:

- *if it returns 'no', then there is no solution;*
- *if it returns $\langle \mathcal{G}, \mathcal{I}, \mathcal{O} \rangle$ and L is (resp., is not) a linear log, then $\langle \mathcal{G}, \mathcal{I}, \mathcal{O} \rangle$ is a (resp., a possibly extended) causal net such that $\langle \mathcal{G}, \mathcal{I}, \mathcal{O} \rangle \vdash L$ and $\mathcal{G} \models \Pi$.*

Proof. By Lemma 9.25 and Theorem 9.6, if the algorithm returns 'no' at step 2, then we are guaranteed that there is no solution. Assume that we are not in this case. Then, by Lemma 9.25, we know that $\texttt{PG}(L, \Pi)$ is a dependency graph over $\mathcal{A}(L)$ with $\texttt{PG}(L, \Pi) \models \pi(L)$ and $\texttt{PG}(L, \Pi) \models \Pi^{\not\rightarrow}$. So, $\mathcal{G}_1$ is such that $\mathcal{G}_1 \models \pi(L)$ and $\mathcal{G}_1 \models \Pi^{\not\rightarrow}$.

Consider now the steps 5–18. Note that whenever the current graph $\mathcal{G}_k$ is updated (in steps 9 and 16), we are guaranteed that any edge (x, y) that is inserted does not violate any negated edge constraint, for otherwise the weight of $\texttt{BestEdge}_\delta(\mathcal{G}, S \rightarrow T)$ or $\texttt{BestPath}_\delta(\mathcal{G}, S \rightsquigarrow T)$ would be $+\infty$. Moreover, in the remaining steps, no edge is added. Therefore, $\mathcal{G}_k \models \Pi^{\not\rightarrow}$ holds, for each $k \geq 1$. In fact, steps 6–18 try to enforce the satisfaction of the edge and the path constraints. By Lemma 9.27 and since $\mathcal{G}_k \models \Pi^{\not\rightarrow}$ holds, for each $k \geq 1$, we derive that if the algorithm returns 'no', then we are guaranteed that no solution exists at all. Again, let us assume that this is not the case, in order to complete the analysis. So, we have now reached step 19, where we are guaranteed that the current graph $\mathcal{G}_k$ is such that $\mathcal{G}_k \models \Pi$. Moreover, as we have just added edges (with no edge incoming into the starting activity or outgoing from the terminating one) and initially $\mathcal{G}_1 \models \pi(L)$ holds, then $\mathcal{G}_k \models \pi(L)$ holds, too. In particular, the subgraph of $\mathcal{G}_k$ induced over the nodes in $\mathcal{A}(L)$ is a dependency graph. Moreover, because of the constraints added in step 1, every other activity in $a \in \mathcal{A} \setminus \mathcal{A}(L)$ is also in a path from the starting to the terminating activity. Hence, $\mathcal{G}_k$ is a dependency graph.

Finally, step 19 invokes a function that equips $\mathcal{G}_k$ with the sets $\mathcal{I}$ and $\mathcal{O}$. The

reader may check that the function coincides with the constructive implementation stated in Theorem 9.6. Therefore, the tuple $\mathcal{C} = \langle \mathcal{G}_k, \mathcal{I}, \mathcal{O} \rangle$ returned as output is such that $\mathcal{C} \vdash L$ and $\mathcal{G}_k \models \Pi \setminus \Pi^{\not\rightarrow}$. Hence, if $\Pi \in \mathcal{C}[\{\rightarrow, \rightsquigarrow, \not\rightarrow\}]$, then we have actually computed a solution, which is a (standard) causal net when L is linear. □

Implementation issues and computation time analysis Let $n_t = |L|$ and $n_a = |\mathcal{A}|$ be the number of traces and activities in input, respectively, and let l_t be the maximal trace length. Let $n^{\rightarrow}$ (resp., $n^{\not\rightarrow}$, $n^{\rightsquigarrow}$) be the number of constraints of type $\mathcal{C}[\{\rightarrow\}]$ (resp., $\mathcal{C}[\{\not\rightarrow\}]$, $\mathcal{C}[\{\rightsquigarrow\}]$) given as input. Moreover, let n_c be the total number of input constraints (independently of the type), and let k be the maximum number of elements in either side of them all, i.e., the maximum size of all their associated sets S and T.

In the implementation, constraints are indexed with a number in $\{1, \ldots, n_c\}$, and two arrays of lists of activity identifiers are used to keep trace of the activities appearing in the left and right sides, respectively, of each constraint. Causal scores, forbidden edges, and dependency graphs are all represented via $n_a \times n_a$ matrices. The initialization of these structures and steps 1-5 can be done in $O(n_t \times l_t^2 + n_c \times k + n_a^2)$ time, where in particular the leftmost term corresponds to computing the causal score matrix and $\texttt{PG}(L, \Pi)$ plus assessing whether this latter satisfies $\pi(L)$.

The cost of the first loop (steps 6-11) is $O(n^{\rightarrow} \times k^2)$. This accounts for checking the satisfaction of all positive edge constraints, and for the cost of computing the "best edge" for each of them. Since no edge is removed, each constraint is considered at most in one iteration, and will remain satisfied in the subsequent ones. The second loop (steps 13-18) resembles the first one, except for the focus on computing "best paths" rather than "best edges". For each constraint, the task can be carried out via

Input: A set $\mathcal{A}$ of activities, with $a_\perp$ ($a_\top$) being the starting (terminating) one,
a log L with $\mathcal{A}(L) \subseteq \mathcal{A}$, and
a set $\Pi \in \mathcal{C}[\{\not\rightsquigarrow\}]$ of precedence constraints with $\mathcal{A}(\Pi) \subseteq \mathcal{A}$;
Parameters: Real numbers $1 > \delta > 0$, and $1 \geq \tau \geq 0$;
Output: A triple $\langle \mathcal{G}, \mathcal{I}, \mathcal{O} \rangle$, or 'no';

1. $\mathcal{G}_1 := \texttt{PG}(L, \emptyset)$;
2. $\mathcal{G}_1 := \mathcal{G}_1 \ominus \{(x, y) \mid (x, y)$ is supported by (at least) $\tau \times |L|$ traces$\}$, $k := 1$, *removed* $:= \emptyset$;
3. **while** $\texttt{FakeEdge}(\mathcal{G}_k) \neq \emptyset$ **do**
4. $\quad (x^*, y^*) := \arg\min_{(x,y) \in \texttt{FakeEdge}(\mathcal{G}_k)} \{\mathbf{cs}_\delta(x, y)\}$, *removed* $:=$ *removed* $\cup \{(x^*, y^*)\}$;
5. $\quad$ **let** z, w be any pair of nodes in $\mathcal{G}_k$ **such that**
$\quad\quad$ **(c1)** $z \in \texttt{succ}(\mathcal{G}_k, x^*)$, $w \in \texttt{pred}(\mathcal{G}_k, y^*)$, $\{(x^*, z), (w, y^*)\} \cap$ *removed* $= \emptyset$;
$\quad\quad$ **(c2)** $\forall z', w'$ satisfying **(c1)**, $\mathbf{cs}_\delta(x^*, z) + \mathbf{cs}_\delta(w, y^*) \geq \mathbf{cs}_\delta(x^*, z') + \mathbf{cs}_\delta(w', y^*)$;
6. $\quad \mathcal{G}_{k+1} := \mathcal{G}_k \oplus \{(x^*, z), (w, y^*)\} \ominus \{(x^*, y^*)\}$, $k := k + 1$;
7. **end while**
8. **if** $\mathcal{G}_k \not\models \Pi^{\not\rightsquigarrow}$ **then return** 'no' (and HALT);
9. **return** $\texttt{computeBindings}(\mathcal{G}_k, L)$;

Figure 9.9: Algorithm COMPUTE-CN (on $\mathcal{C}[\{\not\rightsquigarrow\}]$).

the classical Djikstra algorithm (possibly provided with an artificial "super"-source node, linked to all activities in the lefthand set of the constraint), with a cost $O(n_a^2)$. Since $k \leq n_a$ and we have $n^{\rightsquigarrow}$ constraints, the overall cost is $O(n^{\rightsquigarrow} \times n_a^2)$.

Finally, the input and output bindings of each activity can be stored in two dictionaries, whose entries are multi-sets of activities, with each activity identifying the other vertex of an associated incoming/outgoing edge. Such multi-sets (which reduce to sets in the case of pure causal nets) are simply stored as vectors, encoding edges' multiplicity. Regarding such vectors as strings of length n_a (i.e., the sequence of occurrence counts, one per edge), these dictionaries can be implemented as tries (i.e., prefix trees), which can be built in $O(n_t \times l_t \times n_a)$. This cost accounts for *(i)* scanning each input trace s in both directions (forward and backward) with an index, say i, while incrementally building the multi-set of all activities in its first (resp., last) i positions, and for *(ii)* generating the resulting multi-set and adding it to the input (resp., output) bindings of $s[i]$. In total, we get $O(n_t \times l_t \times \max(l_t, n_a) + n_a^2 \times (1 + n^{\rightsquigarrow}) + n^{\rightarrow} \times k^2 + n^{\not\rightarrow} \times k)$.

9.6.2 The case of negated path constraints

We now present an algorithm to solve CN-Mining on the class $\mathcal{C}[\{\not\rightsquigarrow\}]$, which is illustrated in Figure 9.9. Most of the ingredients discussed so far will still play a role, but the approach is substantially different. The algorithm starts again with the construction of the precedence graph, but this time without taking care of negated edge constraints—as usual, note that edges with low causal scores are removed. That is, we start with the graph $\texttt{PG}(L, \emptyset)$, whose main properties are stated below and are easily seen to hold by inspecting the proof of Lemma 9.25.

Lemma 9.29. $\texttt{PG}(L, \emptyset)$ is a dependency graph over $\mathcal{A}(L)$ and $\texttt{PG}(L, \emptyset) \models \pi(L)$.

The algorithm subsequently breaks any path that witnesses a violation of the negated path constraints in Π. Formally, if $\mathcal{G} = (V, E)$ is a dependency graph over $\mathcal{A}$, then the set of all *fake edges* is defined as $\texttt{FakeEdge}(\mathcal{G}) = \{(x, y) \in E \mid (x, y)$ occurs in a path from an activity in S to an activity in T, with $\neg(S \rightsquigarrow T) \in \Pi\} \setminus \{(x, y) \in E \mid x = a_{\perp}$ or $y = a_{\top}\}$. Note that edges outgoing from the starting activity and incoming to the terminating one are treated differently. Intuitively, we would like to remove all fake edges from the graph. However, while doing so we might miss the ability of supporting the log L, so that we might need to repair the connectivity.

Let $\mathcal{G} = (V, E)$ be a graph over $\mathcal{A}$, and let y be an activity in V. The set $\texttt{pred}(\mathcal{G}, y)$ of the *causal predecessors* of y in L is defined as the set of all the activities $w \in V$ such that there is a path from $a_{\perp}$ to w in $\mathcal{G} \setminus \{(x', y) \mid (x', y) \in E\}$ or $a_{\perp} = w$, and for each trace $t \in L$ where y occurs, i.e., $y = t[i]$ for some $i \in \{1, ..., len(t)\}$, then w also occurs in t before y, i.e., $w = t[j]$ where $j < i$ holds. Symmetrically, let x be an activity in V. The set $\texttt{succ}(\mathcal{G}, x)$ of the *causal successors* of x in L is the set of all the activities

$z \in V$ such that there is a path from z to $a_{\top}$ in $\mathcal{G} \setminus \{(x, y') \mid (x, y') \in E\}$ or $a_{\top} = z$, and for each trace $t \in L$ where x occurs, i.e., $x = t[i]$ for some $i \in \{1, ..., len(t)\}$, then z also occurs in t after x, i.e., $z = t[j]$ where $j > i$ holds. Note that the following is immediate.

Lemma 9.30. Let $\mathcal{G} = (V, E)$ be a dependency graph such that $\mathcal{G} \models \pi(L)$, and let (x, y) be in E, hence with $x \neq a_{\top}$ and $y \neq a_{\perp}$. Then, $a_{\perp} \in \texttt{pred}(\mathcal{G}, y)$ and $a_{\top} \in \texttt{succ}(\mathcal{G}, x)$.

Proof. Note first that, since $\mathcal{G}$ is a dependency graph, it must be the case that $x \neq a_{\top}$ and $y \neq a_{\perp}$. Moreover, it can be also established that $w \cap \{y, a_{\top}\} = \emptyset$ and $z \cap \{x, a_{\perp}\} = \emptyset$, by definition of causal predecessor and successor. Therefore, the graph $\mathcal{G}' = \mathcal{G} \oplus \{(w, y), (x, z)\} \ominus \{(x, y)\}$ is such that the starting activity $a_{\perp}$ and the terminating activity $a_{\top}$ have no ingoing and outgoing edges, respectively. Consider now an activity $a \in V \setminus \{a_{\perp}, a_{\top}\}$. Note that, since $\mathcal{G}$ is a dependency graph, either *(i)* there is a path in $\mathcal{G} \ominus \{(x, y)\}$ from $a_{\perp}$ to a, or *(ii)* the edge (x, y) occurs in each path in $\mathcal{G}$ from $a_{\perp}$ to a. In the case *(i)*, we immediately conclude that there is a path from $a_{\perp}$ to a in $\mathcal{G}'$, too. Hence, let us focus on case *(ii)*. Recall that in $\mathcal{G}'$ we have the edge (w, y) where $w \neq x$. In particular, there is a path from $a_{\perp}$ to w in $\mathcal{G} \setminus \{(x', y) \mid (x', y) \in E\}$, or $w = a_{\perp}$. Hence, we have derived that there is a path from $a_{\perp}$ to a in $\mathcal{G}'$, too. By symmetric arguments, we derive also that there is a path from a to $a_{\top}$ in $\mathcal{G}'$. Hence, $\mathcal{G}'$ is a dependency graph.

In order to conclude the proof, we have now to show that $\mathcal{G}' \models \pi(L)$, too. Indeed, assume by contradiction that a trace t exists in L such that $\mathcal{G}'$ does not model $\pi(t)$. Given the differences between $\mathcal{G}$ and $\mathcal{G}'$ and since the constraints induced by the traces are only positive ones, it must be the case that the removal of the edge (x, y) is the source of the violation. Formally, we can be in one of the following two

scenarios:

(1) It holds that $y = t[i]$ and $x = t[j]$, with $j < i$, and $\mathcal{G}'$ does not satisfy the constraint $\{t[1], ..., t[i-1]\} \to \{y\}$, which is instead satisfied by $\mathcal{G}$ precisely because of the edge (x, y). However, we recall that the edge (w, y) occurs in $\mathcal{G}'$ and that w occurs before y in any trace where y occurs. That is, there is an index $j' \in \{1, ..., i-1\}$ such that $w = t[j']$ where $j' < i$ holds. Hence, $\mathcal{G}'$ satisfies the constraint. Contradiction.

(2) It holds that $x = t[i]$ and $y = t[j]$, with $i < j$, and $\mathcal{G}'$ does not satisfy the constraint $\{x\} \to \{t[i+1], ..., t[len(t)]\}$, which is instead satisfied by $\mathcal{G}$ precisely because of the edge (x, y). However, we recall that the edge (x, z) occurs in $\bar{\mathcal{G}}'$ and that z occurs after x in any trace where x occurs. That is, there is an index $j' \in \{i+1, ..., len(t)\}$ such that $z = t[j']$ where $j' > i$ holds. Hence, $\mathcal{G}'$ satisfies the constraint. Contradiction.

Therefore, $\mathcal{G}' \models \pi(L)$ holds. □

Lemma 9.31. Let $\mathcal{G} = (V, E)$ be a dependency graph such that $\mathcal{G} \models \pi(L)$, let (x, y) be in E, and let $w \neq x$ and $z \neq y$ be in $\texttt{pred}(\mathcal{G}, y)$ and $\texttt{succ}(\mathcal{G}, x)$, respectively. Then, $\mathcal{G}' = \mathcal{G} \oplus \{(w, y), (x, z)\} \ominus \{(x, y)\}$ is a dependency graph such that $\mathcal{G}' \models \pi(L)$.

Example 9.32. *Consider the trace abcde and the constraints* $\neg(\{c\} \rightsquigarrow \{d\})$ *and* $\neg(\{b\} \rightsquigarrow \{d\})$. *In this setting, for the dependency graph* $\texttt{PG}(\{abcde\}, \emptyset)$, *the edges* (c, d) *and* (b, c) *are fake ones. The left part of Figure 9.7 evidences how the graph has to be updated when removing the edge* (c, d)*—in fact, this coincides with the graph built when the edge* (c, d) *is forbidden, as we already discussed in Example 9.24. In the resulting graph,* (b, c) *is no longer fake. However, the graph does not still satisfy the constraints, and* (b, d) *is a fake edge. On the right part of Figure 9.7, a further*

update is reported accommodating the deletion of (b, d). $\triangleleft$

The specific strategy adopted to select causal predecessors and causal successors is formalized in the steps 3–7. Eventually we return the causal net built on top of $\mathcal{G}_k$ via the function `computeBindings`. The correctness of the whole approach is shown below. Note that, similarly to Theorem 9.28, the result explicitly differentiates the case when L is a linear log from the case when L is not linear. As usual, in the former case, $\langle \mathcal{G}, \mathcal{I}, \mathcal{O} \rangle$ is a causal net, while in the latter case it might be an extended causal net.

Theorem 9.33. *The following properties hold on* COMPUTE-CN*, receiving as input* $\mathcal{A}$*,* L*,* $\Pi \in \mathcal{C}[\{\not\rightarrowtail\}]$*, the parameter* δ*, and for* $\tau = 0$*:*

- *if it returns 'no', then there is no solution;*
- *if it returns* $\langle \mathcal{G}, \mathcal{I}, \mathcal{O} \rangle$ *and* L *is (resp., is not) a linear log, then* $\langle \mathcal{G}, \mathcal{I}, \mathcal{O} \rangle$ *is a (resp., possibly extended) causal net such that* $\langle \mathcal{G}, \mathcal{I}, \mathcal{O} \rangle \vdash L$ *and* $\mathcal{G} \models \Pi$*.*

Proof. Because of Lemma 9.29, we know that $\mathcal{G}_1$ is a dependency graph over $\mathcal{A}(L)$ and $\mathcal{G}_1 \models \pi(L)$. Consider then all the update operations performed in the steps 3–7. Because of Lemma 9.30, these operations are well-defined: for each fake edge detected, we can always find a predecessor and a successor (coinciding, at most, with the starting and terminating activities, respectively) that allow for removing it. Moreover, by Lemma 9.31, the graph $\mathcal{G}_k$ is still a dependency graph with $\mathcal{G}_k \models \pi(L)$.

Let us now focus on step 8. Note that when all fake edges are removed, the only edges that remain and that can violate a negated path constraints have the form $(a_\perp, a)$, $(a, a_\top)$, or $(a_\perp, a_\top)$. However, if there is a constraint preventing the existence of a path from $a_\perp$ to a, or from a to $a_\top$, or from $a_\perp$ to $a_\top$, then there can be no solution because of the definition of dependency graph. Hence, it is correct

that the algorithm halts there. Otherwise, for the analysis of the last step, recall that the function `computeBindings` is a constructive implementation of the proof of Theorem 9.6, in order to build a possibly extended causal net from the given graph $\mathcal{G}_k$. In particular, whenever L is linear, we end up with a causal net. □

Implementation issues and computation time analysis In addition to the notation used to analyze the costs of the algorithm of Figure 9.8, let us denote by $n^{\not\rightarrow}$ the number of constraints of type $\mathcal{C}[\{\not\rightarrow\}]$ that are taken as input, and by $m^{\not\rightarrow}$ the number of (distinct) paths prohibited by them, i.e., $m^{\not\rightarrow} = |\{(x, y) \in \mathcal{A} \times \mathcal{A} \mid \exists\ \neg(S \not\rightarrow T)$ such that $x \in S$ and $y \in T\}|$. Note that typically $m^{\not\rightarrow}$ is much smaller than k^2, where k still denotes the maximum number of elements appearing in either side of any constraint.

The data structures are the same as those discussed for the algorithm of Figure 9.8. In order to speed up the calculation of $\texttt{pred}(\mathcal{G}_k, x)$ and $\texttt{succ}(\mathcal{G}_k, x)$, for each activity x, we precompute a relaxed version of both sets, denoted by $\texttt{pred}(x)$ and $\texttt{succ}(x)$, only accounting for the ordering of activities in the log traces. More precisely, for any activity y, we have $y \in \texttt{succ}(x)$ (resp., $y \in \texttt{pred}(x)$) if, and only if, for each trace $t \in L$ where x occurs, then y also occurs in t after (resp., before) x. All such sets of (potential) successors and predecessors are stored as boolean vectors, one for each activity.

Initializing the data structures and performing the first two steps in the algorithm takes $O(n_t \times l_t^2 + n^{\not\rightarrow} \times k + n_a^2)$ time.

The loop spanning over steps 3-7 can be iterated $m^{\not\rightarrow}$ times at most, seeing as each iteration removes at least one of the fake edges and one of the paths violating some constraint—which are n_a^2 and $m^{\not\rightarrow}$, at most, respectively (with $m^{\not\rightarrow} \leq k^2 \leq n_a^2$).

The computation of all fake edges, at the beginning of each iteration, can be accomplished in $O(n^{\not\rightarrow} \times n_a^2)$ time as follows. For each constraint that is still unsatisfied, two symmetric multiple-source visits of $\mathcal{G}_k$ are carried out, starting from all the activities in its left and right sides, respectively; in particular, in the latter case, edges are considered as they were reversed. By reckoning all edges traversed in both directions as fake edges, we compute that with the minimum causal score—(x^*, y^*) in the figure.

We can then compute the transitive closure $\mathcal{G}_k^+$ of the current dependency graph via a matrix-multiplication method, in $O(n_a^\omega)$ time, and materialize it into a matrix. In our current implementation, based on the famous Strassen's method, it is $\omega = 2.8074$. Anyway, answering path queries against $\mathcal{G}_k^+$, in $O(n_a)$ time we can find the nodes z and w mentioned in step 5. To this end, we can simply select the activity in $\texttt{succ}(x^*)$ (resp., in $\texttt{pred}(y^*)$) with the maximal score $\mathbf{cs}_\delta(x^*, z)$ (resp., $\mathbf{cs}_\delta(w, y^*)$), among those satisfying all involved path constraints—involving the existence of a path from z to $a_\top$ (resp., from $a_\bot$ to w), which is a required property of causal successors (resp., predecessors). Therefore, the overall cost of the loop in Figure 9.9 is $O((n_a^\omega + n_a^2 \times n^{\not\rightarrow}) \times m^{\not\rightarrow})$.

A cost of $O(n_a)$ is enough for accomplishing all remaining (edge update) operations in the loop. The same result holds for step 8, where we just need to check whether all constraints are marked as satisfied. As explained previously, a $O(n_t \times l_t \times n_a)$ cost is needed for computing all bindings (while storing them in a concise form).

In conclusion, we get a total cost of $O(n_t \times l_t \times \max(l_t, n_a) + (n_a^\omega + n_a^2 \times n^{\not\rightarrow}) \times m^{\not\rightarrow})$.

9.7 Experimental evaluation

The algorithms proposed in this chapter have been implemented as a plug-in for the well-known process mining suite *ProM* van Dongen *et al.* [2005]. The plug-in receives as input a log file in the (standard) MXML or XES format[3] plus the constraints that users can define by using an intuitive XML-based specification language. As output, it produces a (possibly extended) causal net that, according to the philosophy of *ProM*, is made available and can be re-used in the suite for subsequent elaborations. In fact, *ProM* does not natively support bindings defined as multisets, and therefore the mined models are made available by flattening all multisets into standard sets. This means that, when logs are not linear (cf. Theorem 9.28 and Theorem 9.33), some loss of information might occur when re-using in *ProM* the models mined with our plug-in. However, the true mined models are always exported into a file encoding dependencies and bindings (again) via an XML-based specification language. The plug-in together with documentation on its usage, plus all datasets illustrated in this section are publicly available at http://staff.icar.cnr.it/wfmining/http/public_html/cnmining/.

Internally, the plug-in combines the computation schemes described in Figure 9.8 and Figure 9.9. In particular, the latter scheme is adopted when user-defined constraints belong to the class $\mathcal{C}[\{\not\rightsquigarrow\}]$, whereas the former has been slightly generalized,[4] in order to provide a heuristic solution approach in all remaining cases (while still being an exact solution approach over the class $\mathcal{C}[\{\rightarrow, \not\rightarrow, \rightsquigarrow\}]$). The generalization consists of a post-processing procedure applied to the discovered causal net returned by step 19 in Figure 9.8. The procedure removes "useless" edges with the intended goal

[3] See http://www.processmining.org for details on the mining suite and on its usage.

[4] We also implemented a generalization of the algorithm in Figure 9.9, but results of experimentation evidenced that its efficacy as a heuristic was not satisfying.

Symbol	Meaning
`ILP`	The ILP-based mining algorithm defined in van der Werf *et al.* [2009]
`AGNEs`	The *AGNEs* mining algorithm in Goedertier et al. [2009]
α	The α mining algorithm defined in van der Aalst *et al.* [2004]
`HM`	The *Heuristics* mining algorithm defined in Weijters *et al.* [2006]
`GM`	The *Genetics* mining algorithm defined in Medeiros et al. [2007]
`Here`	The COMPUTE-CN algorithm defined in Figure 9.8 and Figure 9.9

Table 9.1: Process discovery algorithms used in the experiments: legend of symbols.

of returning a more compact model, by increasing the chances of satisfying negated path constraints. Moreover, note that the removal of edges is consistently reflected by suitably updating the associated bindings where they occur.

Formally, the procedure iteratively removes the edge with the lowest causal score over all the edges (x, y) satisfying the following three conditions:

(i) none of the input (resp., output) bindings associated with y (resp., x) coincides with $\{(x, y)\}$, i.e., the edge does not appear as a singleton binding;

(ii) the removal of (x, y) will not violate any user-defined precedence constraint;

(iii) $\frac{\mathbf{cs}_\delta(x,y)}{\min\{\mathbf{cs}_\delta(x,y^*),\mathbf{cs}_\delta(x^*,y)\}} < \tau_{r2b}$, where (x, y^*) (resp., (x^*, y)) is the outgoing edge of x (resp., the incoming edge of y) with the highest causal score. Notice that τ_{r2b} is used here as a relative (lower) threshold to check the strength of any edge (x, y), relatively to those of the best y's predecessor and of the best x's successor—it is similar to the "relative to best" threshold proposed in Weijters *et al.* [2006].

In the tests described in the remainder of the chapter, we fixed the values $\delta = 0.9$, $\tau = 0.05$, and $\tau_{r2b} = 0.75$. These values were chosen pragmatically based on a series of tests conducted on a wide range of synthesized data.

The performances of our implementation have been compared with those of the approaches listed in Table 9.7. Note that we considered the approaches proposed by [van der Werf *et al.*, 2009] (`ILP`) and [Goedertier et al., 2009] (`AGNEs`), which we have already discussed in Section 8.2 and which can in fact incorporate a-priori knowl-

edge on the existence of activity dependencies. Moreover, we considered some classical discovery approaches, very popular in the Process Mining community: "Heuristics Miner" Weijters *et al.* [2006] (`HM`), "Genetic Miner" Medeiros et al. [2007] (`GM`), and the algorithm α van der Aalst *et al.* [2004]. The performances of these methods are discussed here just to delineate baselines suitable for assessing the gain that can be obtained by empowering mining methods with the capability to exploit knowledge on the real structure of the process under analysis. In fact, it will come with no surprise that these "traditional" methods are outperformed in presence of background knowledge (that they are not able to exploit). Hence, it is not sensible to consider a wider range of methods of this kind in the analysis that follows.

In order to test the competitors, we exploited their implementations available in the release 6.3 of the *ProM* framework van Dongen *et al.* [2005]. In fact, for `AGNEs`, `ILP`, and `GM` the implementations in the version 5.2 were used. Indeed, `AGNEs` is only available up to this earlier version, while the implementation of `ILP` in the latest release lacks of the ability of improving the models discovered by expressing parallelism and direct dependency relationships (which is the feature we are interested in). Concerning `GM`, we just obtained better performances with the version 5.2. Moreover, since an earlier implementation of α in the version 5.2 of *ProM* also allows users to express relationships of parallelism and direct dependencies, we have included this method too in all the tests performed in presence of background knowledge. Default settings were used for the various methods. For `GM`, we took the best model derived from an initial population of 100 models by means of 100 iterations (of genetic operations).

In the rest of the section, we shall illustrate results of experimental activity conducted over our method and its competitors. In particular, in Section 9.7.1 we

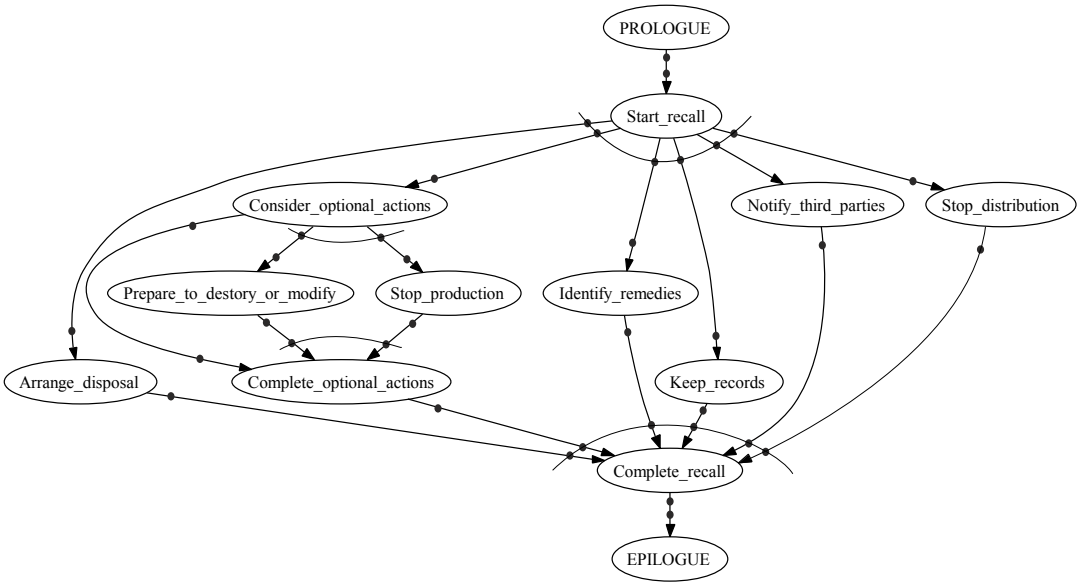


Figure 9.10: Causal net of the *ProductRecall* process.

consider an archetypical application scenario, in Section 9.7.2 we illustrate results on benchmark logs, and in Section 9.7.3 we present results on synthesized logs. A usage scenario (not directly related to process mining) is discussed in Section **??**. Experiments have been performed on a dedicated machine, equipped with an Intel core i7-3770k 3.5 GHz processor with 12 GB (DDR3 1600 MHz) of RAM, and running Windows 8 Professional.

9.7.1 Case study: a product-recall process

Let us consider the product recall process defined by [Wynn *et al.*, 2009], in accordance to the guidelines established by several public institutions (e.g., in Australia, New Zealand, USA, and EU). This is an archetypical application scenario, and its discussion is meant here to assess the importance of having background knowledge when an incomplete sample of the possible traces is given at hand.

Testbed Description

The process concerns the main activities that must be performed by a recall sponsor (usually the manufacturer of a suspect product), in response to a recall incident, which can be possibly triggered by consumer complaints, supplier notifications, or failed quality tests. Specifically, the reported problem has to be investigated and a comprehensive risk analysis must be done (macro-activity `PROLOGUE`—see Wynn *et al.* [2009] for details on the sequence of activities comprised in it), in order to decide whether the product should be recalled or not. The model associated with the activities occurring in the former case is reported in Figure 9.10: after starting the recall procedure, a case can proceed along a number of concurrent threads, including the following tasks: (i) stopping the distribution of the product, (ii) identifying remedies, (iii) arranging the disposal of items already distributed, (iv) keeping records for subsequent monitoring and analysis purposes, and (v) notifying third parties about the recall. In addition, depending on the kinds of product and of defect involved, it can be necessary to halt the production of the product and to destroy/modify other products that might have been contaminated. Once these recall actions have been completed, a sequence of finalizing activities must be performed (macro-activity `EPILOGUE`), ranging from monitoring the effectiveness of the process, to implementing changes to prevent similar problems in the future, to preparing reports for regulatory authorities and/or other third parties. Notice that, as pointed out by [Wynn *et al.*, 2009], the need of handling product recall operations, while taking care of traceability and notification issues, arises in a wide variety of real applications.

Evaluation Setting

In order to assess the quality of findings, we contrast the set D_{out} of causal dependencies discovered by the mining methods towards the set D_{in} of the real dependencies existing in the a-priori known process model, by resorting to the classical *F-measure* metric, defined as $\frac{2\times P\times R}{P+R}$, where P (standing for *precision*) is the fraction of the dependencies in the mined model that do exist in the real model, i.e., $P = |D_{out} \cap D_{in}|/|D_{out}|$, whereas R (standing for *Recall*) is the fraction of real dependencies captured by the mined model, i.e., $R = |D_{out} \cap D_{in}|/|D_{in}|$.

Test with variable amounts of log traces

First, we generated a log (of 10000 traces) that covers a significant number of all the possible behaviors and over which the various mining methods are able to perfectly reconstruct the process model for this case study. This step was just meant to check that all methods can be in principle effective to face the given scenario. Then, we progressively filtered the log, by therefore increasing its level of incompleteness. We stopped the process with a resulting log L of about 250 traces (of which about 200 are distinct ones), over which `HM` (the simplest discovery method) is still able to perfectly reconstruct the underlying model based on them. Experiments have been conducted over logs that are obtained from L by taking $x\%$ of its traces, for $x \in \{10, 20, ..., 100\}$. In particular, for each x, we built 10 samples. Each sample has been built by (uniformly at random) removing one trace, until a sub-log of the desired size of $\frac{x}{100} \times |L|$ is obtained. Experiments are performed on each of them, so that average values (for any given value of x) will be discussed in our analysis. Table 9.7.1 summarizes our findings.

Let us first consider the columns reporting results for the scenario where no con-

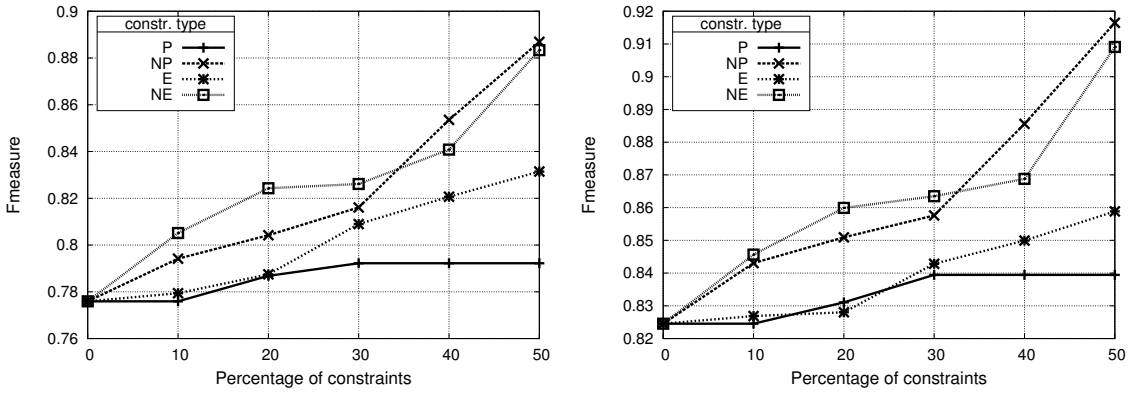
trace %	**without** constraints						**with** constraints			
	HM	α	GM	ILP	AGNEs	Here	α	ILP	AGNEs	Here
10	0.657	0.733	0.553	0.848	0.674	**0.929**	0.772	0.848	0.667	**0.956**
20	0.869	0.871	0.578	0.924	0.727	**0.987**	0.899	0.924	0.736	**0.992**
30	0.901	0.925	0.603	0.914	0.677	**0.984**	0.950	0.914	0.720	**1.000**
40	0.927	0.944	0.658	0.914	0.720	**0.992**	0.983	0.914	0.730	**1.000**
50	0.964	0.969	0.591	0.904	0.748	**1.000**	0.968	0.904	0.727	**1.000**
60	0.974	0.968	0.628	0.903	0.745	**1.000**	0.970	0.903	0.774	**1.000**
70	0.979	0.990	0.584	0.882	0.774	**1.000**	0.990	0.882	0.763	**1.000**
80	0.982	0.993	0.629	0.893	0.763	**1.000**	0.993	0.893	0.779	**1.000**
90	**1.000**	**1.000**	0.652	0.882	0.763	**1.000**	**1.000**	0.882	0.779	**1.000**
100	**1.000**	**1.000**	0.684	0.882	0.763	**1.000**	**1.000**	0.882	0.782	**1.000**
Avg	0.911	0.939	0.616	0.897	0.735	**0.989**	0.952	0.897	0.752	**0.995**

Table 9.2: F-measure scores obtained by algorithm COMPUTE-CN and its competitors, for different percentages of traces of process *ProductRecall* (Figure 9.10), without (left) and with (right) a-priori knowledge on parallelism relationships. For both cases, maximal scores on each trace percentage are written in bold.

straints are provided as input. It is easily seen that the performances of all methods are rather poor against very small log samples, and tend to improve when augmenting the number of input traces. Such an effect is evident in the case of the classical methods HM and α, which get full accuracy when provided with at least 90% of the log traces. Interestingly, our approach managed to reconstruct the real set of task dependencies already with 50% samples of the log, hence outperforming the competitors even without being provided as input with additional background knowledge. This confirms the validity of the processing scheme of Figure 9.8 and Figure 9.9 and the efficacy of the post-processing method illustrated in this section, where different heuristics (mainly considered in isolation in earlier approaches) are synergically integrated and exploited within an original computation scheme. In particular, note that in most of the classical approaches, a dependency from an activity x to an activity y can be added to the model only when x is a direct predecessor of y in some trace and the pair (x, y) is associated with a high causal score value. Our algorithms, instead, are more liberal because the definition of precedence graphs (see Section 9.6.1) is

entirely based on causal scores, so that a dependency from x to y can be added even when x never occurs as a direct predecessor of y. On the other hand, recall that dependencies are then pruned based on their support in the log with the threshold τ and by using the "relative to best" threshold proposed in Weijters *et al.* [2006]. The results in Table 9.7.1 evidence the efficacy of the combined usage of these ingredients. In fact, opposed to these good news related to our method, it emerges that low quality scores (with no evident gain w.r.t. the given percentage of traces) are achieved when using `ILP` and `AGNEs`, which are to be seen as our more direct competitors.

The four rightmost columns in Table 9.7.1 report the results obtained by providing domain knowledge to the discovery methods. To this end, we assumed that the analyst knows that the activity `Complete_optional_actions` is parallel with each of the activities in {`Identify_remedies`, `Keep_records`, `Stop_distribution`, `Arrange_disposal`, `Notify_third_parties`}. Note that this knowledge can be incorporated in the two competitors `AGNEs`, and `ILP`, as well as in the α algorithm, so that we are not exploiting for the moment the richer expressiveness of our framework. By looking at the table, it emerges that our method is able to fully exploit these constraints, by getting impressive accuracy results even on very small samples. Instead, the benefits of the background knowledge are lower for α and especially for `AGNEs`. Moreover, note that `ILP` is unable to exploit at all the given knowledge. In fact, in our experiments, we have noticed that the method is quite often not capable to benefit from knowledge about parallelism, while significant improvements can be obtained in presence of knowledge about relationships of precedences, as we shall see later.



(a) F-measure scores on *3%* log samples

(b) F-measure scores on *6%* log samples

Figure 9.11: F-measure scores obtained by COMPUTE-CN when varying the percentages of positive edge/path constraints and of negative edge/path constraints, for four different families of log samples, corresponding to 3% (a) and 6% (b) of traces in the log L, respectively.

Varying the quantity and type of background knowledge

In order to assess, in a deeper and more systematic manner, the capability of our approach to exploit a-priori knowledge, further tests were carried out on the same scenario while using different amounts of precedence constraints. The focus is to understand the impact of each of the types of precedence constraints on the quality of the mining results. Therefore, only very small portions of the log were used, containing 3% and 6% of the traces of the log L (of about 250 traces). For each trace percentage x% (with $x \in \{3, 6\}$), four samples were generated (as discussed in Section 9.7.1) and, as usual, average results are discussed.

Different inter-activity dependencies were extracted directly from the known model of the process, shown in Figure 9.10. Regarding this model as a dependency graph, four relations over its activities can be defined and computed trivially, each encoding some basic kind of (singleton body/head) precedence constraints: edges and paths (i.e., pairs of activities, where the second one depends on the first either directly or indirectly, respectively), and the associated complementary (w.r.t. all possible activ-

ity pairs) relations of negative edges and negative paths. These sets are denoted in the figures discussed below by E, P, NE, and NP, respectively. And, they consists of 19, 46, 150, and 123 different constraints, respectively. Then, in order to automatically generate different sets of precedence constraints, while controlling the relative amount of each type of them, a sample of elements is extracted randomly from each of the core relations discussed above, by using some given percentage value for each of them. In fact, five samples were generated and results were averaged over them.

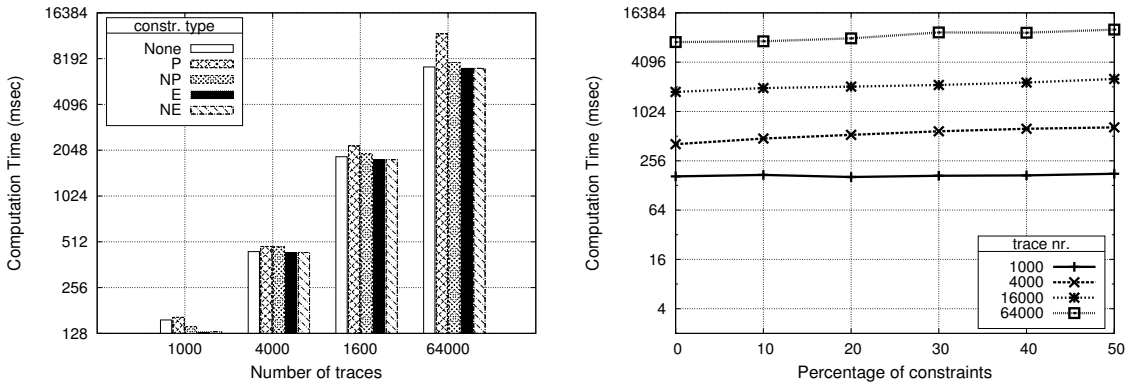
Figure 9.11 reports the average F-measure scores obtained by our solution approach against the different percentages of log traces, while using one of the above described kinds of pairwise constraints per time. For each kind of constraints, different amounts were considered, ranging from 0 to 50% of its whole population. It clearly emerges that the use of the background knowledge improves the performances of the algorithm. Indeed, higher F-measure scores are achieved when increasing the amount of whichever kind of constraints. However, a noticeably boost seems to be given to the accuracy when increasing the percentage of negative (edge or path) constraints, no matter of how many traces are taken as input. Conversely, lower improvements are obtained when only positive constraints are used, especially when these are expressed on paths. Intuitively, this is due to the fact that negative constraints are able to reduce the number of spurious flows of execution, which negatively impact on the quality of the resulting process model. This finding was confirmed over all our tests. Hence, as a practical guideline, the users of our plug-in are encouraged to introduce as much as possible negated constraints in the specification of the mining problem.

Note that Figure 9.11 evidences that the level of improvement is neatly higher when working on smaller log samples (hardly capturing all actual process behaviors).

Moreover, note that even in absence of constraints and with very few traces at hand (3%), our method is capable of achieving good performances in reconstructing the underlying model. This is in line with our findings discussed in Section 9.7.1.

Rate of unsatisfied constraints

So far our algorithm has been tested in scenarios where precedence constraints of different kinds are not mixed together. Therefore, according to the results discussed in Section 9.6, it provides an exact solution in these cases, i.e., no constraint can be violated by the resulting process model. Hence, in order to study the efficacy of the method as a heuristic, we performed an additional series of experiments with heterogeneous combinations of precedence constraints, mixing up negative path constraints with other kinds of constraints. To this end, we applied our algorithm to the same log samples built as in the previous subsection, while providing it with variable amounts of negative path constraints, and fixing the percentage of any other kind of constraints to 25%. Note that, given the adaptation discussed at the beginning of this section, the algorithm returns a process model that still satisfies all positive edge constraints, path constraints, and negated edge constraints. However, the satisfaction of negated path constraints is just greedily enforced by the post-processing phase, and it is therefore not guaranteed. In fact, we computed the rate of negated path constraints left unsatisfied by our approach, in correspondence of different amounts of distinct traces and of negated path constraints provided as input (expressed as percentages w.r.t. the size of their respective populations). In the worst case, just 0.8% of all negated path constraints given as input were not satisfied, in the average. This rate shrinked of about a half when using either 10% samples or 50% samples of negative paths. A similar trend was observed for trace samples with a 6% of distinct process traces. The amount of violated constraints became negligible when bigger



(a) Computation Time vs Log Size

(b) Computation Time vs Constraints' Amounts

Figure 9.12: Computation time spent by algorithm COMPUTE-CN with different amounts of traces and constraints' percentages in input. A base-2 logarithmic scale is used for the vertical axis in both figures, as well as for the horizontal axis in the left-hand figure.

samples of log traces (we tested both 12% or 24%), no matter how many negated paths are passed to the algorithm.

Running time

As we have pointed out in the Introduction, it often happens in real process mining applications that only a "small" set of (representative) traces are available for analysis purposes. For this reason, scalability plays generally a secondary role in the design of process mining algorithms, and many works just omit at all this kind of analysis. Nonetheless, we feel that it is relevant to complete the picture by providing the reader with a general idea of the scaling of our algorithms, hence complementing the theoretical analysis carried out in Section 9.6. To this end, we performed a series of tests on logs of different sizes. Logs were generated again with a variable level of completeness, by extracting different amounts of distinct traces out of the original log. First of all, we randomly extracted a sample of (distinct) traces out of the complete log, to produce different logs with size $s \in \{1000, 4000, 16000, 64000\}$. In particular, when traces have to be added to obtain a log bigger than the original

one, we simply duplicated the original traces in a balanced way (i.e., all traces are replicated approximatively the same number of times).[5]

Figure 9.12 reports the computation time (measured in milliseconds) when varying the amounts of traces and constraints in input. In particular, on the left, it shows results grouped according to the different kinds of constraints we have considered (i.e., E, P, NE, NP) and averaged over the samples built when considering $p\%$, with $p \in \{0, 10, 20, 30, 40, 50\}$, of constraints derived from the population of the given kind (within the same setting as in Section 9.7.1).

On the right, results are reported where, for each particular percentage $p\%$ of constraints, all the sets of different constraints built for p have been merged together. For each value of p, five different heterogeneous sets of constraints were generated, containing $p\%$ constraints for each constraint type, extracted at random from their respective populations. Four trials were performed over different logs built as discussed above, and as usual average measures are reported. Notably, whatever percentage is chosen, the computation time scales basically linearly with respect to the number of input traces. On the other hand, a positive correlation seems to exist as well between the overall running time and the quantity of constraints taken as input—if ignoring the case of 1000-sized logs, where the times measured are too low to safely infer any general significant trend of behavior. Anyway, the impact of constraints on times is negligible if compared with that of the log size.

Finally, we notice that the computation time of algorithm COMPUTE-CN is comparable to that of the standard process mining methods α, `ILP`, and `HM`, and neatly lower than those spent by `GM`, `AGNEs`. Indeed, the latter method took about 600 times longer than COMPUTE-CN to compute a model, in the average, while the

[5] Duplicated traces might be processed efficiently by "weighting" each trace with the number of its occurrences in the log. We avoid this trick, as we want to test the algorithms at the varying of the log size.

log name	distinct activities	pairs of $\|\|$ activities	control-flow constructs	distinct traces	log size
$parallel5$	10	10	–	107	3000
$a10skip$	12	1	$skip$	6	2665
$a12$	14	2	–	5	2492
$a5$	7	1	$loop$	13	2189
$a6nfc$	8	1	$nf\text{-}choice$	3	2040
$a7$	9	4	–	14	2023
$a8$	10	1	–	4	1803
$choice$	12	0	–	16	2400
$driversLicense$	9	0	–	2	2100
$herbstFig3p4$	12	3	$loop$	32	5637
$herbstFig6p18$	7	0	$loop$	153	9844
$herbstFig6p36$	12	0	$nf\text{-}choice$	2	3000
$herbstFig6p37$	16	36	–	132	4800
$herbstFig6p41$	16	4	–	12	3600
$herbstFig6p45$	8	4	–	12	2400
$l2l$	6	0	$loop$	10	4932
$l2lOptional$	6	0	$loop, skip$	9	2622
$l2lSkip$	6	0	$loop$	8	4554

Table 9.3: Benchmark logs: structural characteristics and statistics (see also Weerdt et al. [2012]). Each log consists of 300 (not necessarily distinct) traces.

computation time of `GM` was about 4500 times that of COMPUTE-CN when using the default population size of 1000 (the ratio only decreased to 600:1 with 100-model populations).

9.7.2 Comparative analysis on benchmark data

In order to assess the capability of our approach to discover a *good-quality* process model in a wider range of settings, we performed a series of tests on some benchmark logs, while measuring the accuracy of each discovered model by way of several "log-conformance" metrics, very popular in the fields of Process Mining and Business Process Analysis. Differently from the pure (edge-oriented) F-measure employed in the previous section, to contrast a discovered model to the true (a-priori known) one, these metrics allow for evaluating how much the behaviors registered in a given log comply with those allowed by the model under analysis.

Testbed: logs and conformance metrics

Our experimental activities were carried out over some of the benchmark logs provided with the ProM framework van Dongen *et al.* [2005], which have been widely used in the literature (see, e.g., Medeiros et al. [2007]; Goedertier et al. [2009];

De Weerdt et al. [2011]; Weerdt et al. [2012]) in order to evaluate process mining approaches—this benchmark is also congenial to our ends, since the underlying "true" models are available, and can be used both for comparison purposes and for automatically extracting background knowledge. In particular, owing to our special interest toward incompleteness issues, we focused on those logs exhibiting the highest degree of non-determinism and concurrency. The logs were generated from models including a number of routing constructs/patterns, such as (non free) choices, skips, and loops—the reader interested in expanding on these concepts is referred to the above literature. Table 9.7.2 summarizes their features, by reporting in particular the number of distinct activities composing the process, the number of pairs of activities belonging to different parallel branches ("parallel || pairs"), the presence of special constructs, the number of traces, and the log size measured as the number of *episodes* stored in it, i.e., as the sum of the occurrences of the activities over all traces. Note that six logs derive from process models that contains loops, and in fact they are not linear logs as they contain traces with multiple occurrences of the same activity.

In order to assess the capability of a process model to accurately capture the behavior recorded in a given log, several alternative *conformance* metrics have been proposed in the literature. In our analysis, given the focus of our approach, we consider *precision* metrics and *recall* metrics. Precision metrics attempt to estimate the amount of the "extra" (unseen and likely unwanted) behavior allowed by the model, with respect to that actually registered in the log, whereas recall metrics try to evaluate how much of the behavior recorded in a log is really captured by the model. All these metrics range over the real interval $[0, 1]$ and have been defined in the literature for Petri-net models. In order to use them with a model represented

Metric	*True*	AGNEs	α	GM	HM	ILP	Here
Fitness Rozinat and van der Aalst [2008]	1.000	0.995	0.988	**1.000**	0.994	**1.000**	0.994
Alignment Based Fitness Adriansyah et al. [2011]	1.000	**0.968**	0.830	**0.978**	**0.958**	0.889	**0.995**
Behavioral Recall Goedertier et al. [2009]	0.997	**0.992**	**0.978**	**0.989**	0.846	**1.000**	**0.989**
Proper Completion Rozinat and van der Aalst [2008]	1.000	0.938	0.851	**0.999**	**0.933**	**1.000**	0.931
Adv. Behav. Appropriateness Rozinat and van der Aalst [2008]	0.814	**0.823**	**0.854**	0.802	0.783	**0.856**	0.809
Alignment Based Precision van der Aalst *et al.* [2012]	0.921	**0.926**	**0.948**	0.874	0.871	**0.912**	**0.939**
Behavioral Specificity Goedertier et al. [2009]	1.000	**0.992**	0.978	**0.997**	0.990	**1.000**	0.986
(Wtd) Behavioral Precision De Weerdt et al. [2011]	0.915	**0.884**	0.882	**0.890**	0.748	**0.891**	0.887
Negative Event Precision Goedertier et al. [2009]	0.970	0.927	0.927	**0.953**	**0.943**	0.934	**0.937**

Table 9.4: Average conformance measures obtained, on benchmark logs, by different discovery methods—including the one proposed in this thesis (*Here*). For each row, the best average score (excluding ground-truth models) is underlined, while the results that were **recognized as significantly better than the average** of the outcomes over the various methods (for the same metrics and setting) are in bold.

in another language (for instance, a causal net), we preliminary translated the given model into a Petri net, with the help of suitable conversion plug-ins available in the ProM framework. The actual computation of the metrics was carried out by taking advantage of the *CoBeFra* tool, recently proposed van den Broucke *et al.* [2013] as a practical support to conformance analysis. In fact, we noticed that slight different measures can be obtained over different runs of the tool (but deviations are very limited), so that hereinafter we refer to measures averaged over three runs. A summary of all the considered metrics is reported in the first column of Table 9.7.2.

Results

For each of the conformance metrics described above, Table 9.7.2 reports the average value obtained by applying each of the methods to each of the logs in Table 9.7.2. As a term of comparison, column *True* reports the conformance results obtained for the process models that were actually used to generate the logs (which are all available). Since many methods got very similar results over several metrics, a statistical testing procedure was carried out, in order to check whether their behaviors are really different. To this end, for each of the considered metrics, a paired one-tail Student's test was applied to compare the outcomes of each method with the average of those obtained by all the methods. Notably, for each of the metrics, we did

not find significantly enough differences, apart from a small number of cases, which are emphasized in bold in the table—in almost all cases we could not reject (with a 95% level of confidence) the null hypothesis that a method behaves identically to the "average" one, rather than improving the latter. In fact, since all methods achieve good performances and differences are not statistically relevant, it is not informative to devote further space to analytically illustrate these performances over each of the logs. Accordingly, we just focus below on illustrating Table 9.7.2, where a synthectic picture is depicted by averaging the results over the whole family of benchmark logs.

"Traditional" approaches are able to reconstruct the originating models with high levels of precision and recall (this is hardly surprising given that these methods have been designed precisely to face the challenges in such kinds of benchmarks logs). However, `AGNEs`, `ILP`, and our method also exhibit good performances over these logs, and in some cases they perform better than the competitors even though no background knowledge is available. In any case, we observe that our evaluation procedure tends to disadvantage a model that was not originally built as a Petri net (as in our case), because the conversion plug-ins often produce results with hidden/duplicated transitions, which are considered as a source of extra-log behaviors by certain metrics.[6]

As a further comment, we stress here that, in terms of computation time, our approach is in line with the most efficient process discovery techniques, and definitely faster than both `AGNEs` and `GM`. The average values and standard deviations computed over running times (in milliseconds) are: 21851.2 ± 12334.6 for `AGNEs`, 341012.2 ± 385997.7 for `GM`, 95.6 ± 73.4 for `HM`, 203.1 ± 53.4 for `ILP`, and 138.2 ± 85.3 for our approach.

[6] In fact, since even the original models are not available as Petri nets, they get lower scores, for some metrics, than those obtained for methods returning true Petri nets.

original benchmark log	(%) of traces in the sub-log
herbstFig3p4	41%
herbstFig6p37	4%
herbstFig6p41	23%
herbstFig6p45	18%
parallel5	3%

Table 9.5: Statistics on the "critical" sub-log extracted for each of the benchmark logs in Table 9.7.2.

Tests on (critical) sublogs

In order to test all discovery methods in a more challenging setting, we extracted a *critical* sample out of each log, defined as an incomplete collection of traces, covering a limited portion of the whole variety of the behaviors in the original log. An empirical iterative procedure was devised to this purpose, where increasing amounts of traces are randomly removed from a given benchmark log, until we register a loss of 15% in the F-measure of the models discovered with algorithm `HM`. In order to further emphasize the effect of log incompleteness, we focused our attention on a subset of the logs in Table 9.7.2, which allowed all the tested discovery methods to perfectly rediscover the associated model (i.e., to achieve a maximal value of 1 over both F-measure and all of the recall metrics). The number of traces of each of these sub-logs is reported in the Table 9.7.2.

Given the incompleteness of the resulting logs, experiments were conducted also in the presence of background knowledge. In this case, for each critical sub-log and for each of the tested methods, we considered the background knowledge consisting of the three constraints extracted from the given true model that allow us to obtain the best performances w.r.t. the F-measure. Note that here we exploit the full expressiveness of our approach. In fact, it emerged that the best results are obtained in presence of negative constraints only.

Results for these tests are reported in Table 9.7.2, whose precision and recall scores

Metric	**without** constraints						**with** constraints			
	AGNEs	α	GM	HM	ILP	*Here*	α	AGNEs	ILP	*Here*
Fitness	0.900	0.852	0.950	0.882	0.941	0.951	0.865	0.950	0.969	**1.000**
Behavioral Recall	0.903	0.792	0.948	0.881	0.945	0.939	0.806	0.955	0.972	**1.000**
Alignment Based Fitness	0.858	0.744	0.948	0.834	**0.945**	0.948	0.799	0.919	0.572	1.000
Proper Completion	0.081	0.102	0.523	0.081	0.459	**0.543**	0.340	0.400	0.655	**1.000**
Adv. Behav. Appropriateness	0.625	0.711	0.901	0.580	0.828	0.910	0.788	0.579	0.880	0.928
Alignment Based Precision	0.952	0.957	0.966	0.956	0.963	0.965	0.967	0.966	0.973	0.972
Behavioral Specificity	0.903	0.792	0.948	0.881	**0.945**	0.949	0.806	0.955	0.972	**1.000**
(Wtd) Behavioral Precision	0.634	0.701	0.794	0.615	**0.794**	0.776	0.764	0.843	0.788	**0.966**
Negative Event Precision	0.678	0.746	0.827	0.658	0.833	0.822	0.815	0.872	0.816	**0.976**
F-measure	0.741	0.730	0.855	0.735	0.831	0.852	0.806	0.941	0.915	**1.000**

Table 9.6: Results on "critical" samples without and with background knowledge. For each row, the best average score is underlined, while the results that were **recognized as significantly better than the average** of the outcomes over the various methods (for the same metrics and setting) are in bold.

were computed by comparing each model discovered (from the sub-logs described above) with the respective (entire) benchmark log. Moreover, the F-measure values are also reported (see Section 9.7.1). As above, a one-tail Student's test was carried out to assess the significance of the performance differences. The results obtained in absence of background knowledge further confirm the validity of our approach and, in particular, the effectiveness of its underlying (causal-score driven) heuristics for pruning useless/unlikely edges. The advantage of using additional background knowledge clearly emerges. In particular, our method perfectly reconstructs the true models of all benchmark logs (see the lowermost row), and achieves full recall without incurring into overgeneralization.

9.7.3 Further tests on synthesized data

Another series of tests was conducted on different synthesized logs, in order to assess the effectiveness of the proposed approach against logs following different data distributions, as far as concerns the structure of the processes producing them.

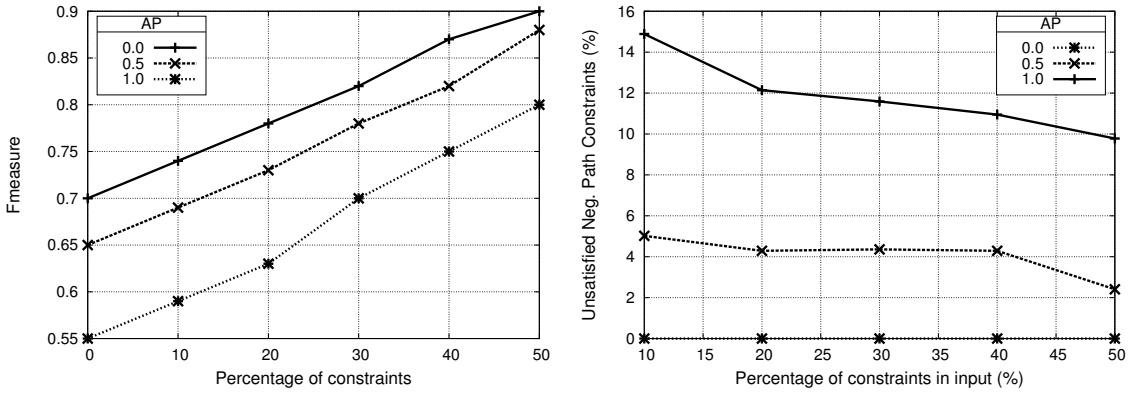
Generation of process models

The data used in this experimentation were produced with the help of a log generator (based upon the one described in Burattin and Sperduti [2011]), which

allows for both constructing a process model randomly, and generating a log of random execution traces out of it. The synthesis of the model is carried out via an incremental block-oriented procedure, where an initially empty model is extended iteratively by adding a new subprocess, until a given number of elementary activities is obtained. Each subprocess can be either an elementary activity, a sequence of activities, or a more complex control-flow block. The whole procedure is governed by a combination of gaussian and multinomial probability distributions, and it can be controlled by way of a number of parameters, which are listed below:

- *Task Number* (TN): the number of elementary activities to be generated;
- *Branching Factor* (BF): the number of branches in any branching structure;
- *Singleton Probability* (SP): probability that any current subprocess will be instantiated with just one activity;
- *Fork Probability* (FP): probability that the subprocess is a "fork" structure (i.e., a number of subprocesses that can be executed in parallel or that are mutually exclusive), given that it is not an elementary activity;
- *AND Probability* (AP): probability that the branches of any fork structure are in parallel, rather than in mutual exclusion.

In order to prevent models from having an excessively unbalanced shape, one further parameter can be set, named the *Maximum Nesting Depth* (MND), stating the maximum level of nesting of any activity with respect to all the control-flow blocks it is enclosed within—precisely, whenever the nesting level of a subprocess equals MND, the subprocess is forced to take the form of a single activity (i.e., it cannot give rise to a complex control-flow block).



(a) Accuracy w.r.t. true dependencies

(b) Rate of unsatisfied neg. path constraints

Figure 9.13: Results on synthesized log data, with different degrees of parallelism (AP): accuracy of discovered process models (in terms of F-measure w.r.t. real activity dependencies), and rates of unsatisfied (negative path) constraints. Both measures are reported for different amounts of a-priori constraints of all types (expressed as percentages w.r.t. the sizes of their respective population).

In our experiments, we fixed $SP = 0.1$ and $MND = 4$, and simulated three different levels of concurrency, hinged on different values of the parameter $AP \in \{0.0, 0.5, 1.0\}$. In order to consider a wide enough variety of process schemas, the remaining parameters were chosen in a random manner. Specifically, we picked up TN from a truncated normal distribution ranging between 10 and 30, with location 25 and scale 8. Instead, BF and FP were extracted from two uniform distributions ranging over the sets $\{2, 4, 6\}$ and $\{10, 20, \ldots, 80\}$, respectively. Overall, we generated 25 different schemas, and produced 4 logs for each of them, by simulating 1000 random enactments of the schema for each of them (as in Burattin and Sperduti [2011]). In this way, 100 different logs were eventually obtained, each one consisting of 1000 traces.

Results

Figure 9.13 reports the results of tests performed by applying our approach to the logs described above, while providing it with heterogenous sets of precedence

constraints, and hence expressing partial information on real activity relationships. As in Section 9.7.1, these constraints were randomly extracted from the model which generated the log, according to different levels of coverage (w.r.t. the entire population of constraints that can be derived from the model), expressed in terms of a percentage $p \in \{0, 10, 20, ..., 50\}$. In more details, for each log L and each value of p, we randomly generated 5 sets of constraints, by sampling $p\%$ pair from each of the binary relations (of the form E, P, NE, NP) extracted from the a-priori known model of L. Note that the binary relations E, P, NE, and NP vary depending on the specific model from which they have been extracted. As discussed above, we have considered 25 different schemas. Their sizes, averaged over the 25 different scenarios, are of 24, 140, 440, and 324 elements, respectively. The maximum (resp., minimum) sizes over the scenarios are 32, 251, 752, and 533 (resp., 14, 48, 130, and 94), respectively.

Each value was computed by averaging all the results obtained when using *(i)* any log generated with the given value of AP, and *(ii)* any constraint set covering $p\%$ of each (positive/negative direct/indirect) precedence relation. Clearly, higher levels of concurrency (i.e., higher values of AP) tend to lead to lower accuracy results, as far as concerns the recognition of real activity dependencies. Providing increasing amounts of constraints definitely helps to improve the average accuracy of the resulting models.

As a further effectiveness indicator (connected with the capability of our approach to really satisfy the constraints it was provided with), we finally present, in the right side of Figure 9.13, the rate of unsatisfied negative path constraints, i.e., the percentage of constraints of this kind that were taken as input in some run of the algorithm, but were not fulfilled by the causal net discovered in the same run—recall that all other constraints are necessarily satisfied. Again, these results are reported for different values of both p and AP. It is good news that the rates of unsatisfied

negative path constraints are always relatively small, especially in the case of low concurrency levels. In fact, all the models that were induced from concurrency-free logs do not violate any constraint at all. Higher violation rates affect the models extracted from choice-free logs ($AP = 1$), i.e., logs produced by a model where all the branches going out of a fork node are always executed concurrently. However, even in such an extreme case, the rate is quite low, reaching at most 15%, precisely when $p\% = 10\%$. Therefore, the absolute number of constraints violated is still very small, and only represents a 0.15% of all possible constraints associated with the underlying models. Moreover, note that increasing the amount of input constraints seems to help our approach reduce violation rates. This can be explained by observing that, as the availability of more background knowledge makes each discovered model more similar to the true one, it is more likely that the former will eventually satisfy a higher amount of all the a-priori constraints associated with the latter.

9.8 Summary

Current research is rather active in proposing process mining techniques supporting increasingly expressive modeling languages. However, most of the approaches proposed in the literature mine the causal dependencies that hold over the activities, while completely ignoring the prior knowledge that in many cases is available to the analyst. This dissertation moves a systematic step to fill the gap, by proposing and analyzing a constraint-based framework for process mining, where background knowledge can be encoded in the form of precedence constraints. The computational complexity of the framework has been studied, and the whole approach has been implemented in a prototype system, which has been tested on different log data.

Note that the use of constraints is currently gaining attention in a different con-

text, yet still related to process management, namely in the context of developing *declarative* approaches to support business automation. In fact, traditional modeling languages, such as Petri nets or causal nets, are *procedural* ones for they explicitly represent all the allowed behavior of the process, according to a "closed world" assumption. Opposed to this approach, recent research focused on developing declarative models where any possible enactment is allowed unless a constraint (expressed in some suitable formal logic) is known to hold and explicitly forbids it (see, e.g., van der Aalst *et al.* [2009]; Sadiq et al. [2005]; Reichert et al. [2009]). Following a number of earlier attempts to use logic-based languages for the specification of business processes (see, e.g., Bonner [1999]; Senkul et al. [2002]; Dourish et al. [1996]; Joeris [2000]; Wainer and de Lima Bezerra [2003]; Lu et al. [2006]; Attie et al. [1996]), *Declare* Pesic et al. [2009] is nowadays the most solid platform adopting a declarative perspective for process modeling, where the semantics of each constraint is provided in terms of an associated Linear Temporal Logic (short: LTL) formula, whose (finite) models are precisely the set of all the allowed traces Pesic et al. [2007, 2010].

Interestingly, the problem of automatically inferring a process model from a given log available at hand has been considered within these declarative frameworks for process management, too. In particular, techniques specifically designed to infer Declare constraints have been presented by [Maggi et al., 2011] and [Maggi et al., 2012] and subsequently enhanced by [Maggi et al., 2013], in order to discover data-aware models. Another approach has been proposed by [Di Ciccio and Mecella, 2012], where constraints are eventually expressed in terms of *regular expressions* rather than in terms of LTL formulas. By sharing the spirit of the above proposals, but focusing on a slightly different problem, [Chesani et al., 2009] proposed a methodology to analyze a log whose traces are labeled as compliant or non-compliant, and whose goal

is to learn a classification model defined as a set of rules/constraints expressed in the SCIFF Alberti et al. [2008] language (eventually mapped in the Declare notation). Other approaches to building rule-based classification models have been proposed, for instance, by [Bellodi et al., 2010] and [Ferreira and Ferreira, 2006].

By looking at the above body of literature, it clearly emerges that the use of constraints in these works is completely different from ours. Indeed, in these declarative frameworks, constraints are used to specify which traces are allowed, while in our approach they help define the set of those possible process models that are of interest to the analyst. In fact, our perspective is the traditional one where the analyst wants to end up with a procedural model, and where constraints are used to prune the search space of all the models that could be induced from a given log. Accordingly, we adopted a language to specify topological constraints on the process model, rather than a language (such as LTL) tailored to define constraints on traces.

Even though the two approaches are completely orthogonal, we stress however that the research reported in this thesis might have also an impact in the context of developing mining approaches for declarative process models. Indeed, in this setting, the idea of incorporating a-priori knowledge in the mining phase has been largely unexplored, and constitutes a promising avenue of further research. Currently, analysts are provided with the rules induced via mining methods and might refine them to incorporate their knowledge in a post-processing phase. Instead, it would be interesting to define approaches where analysts can a-priori formalize their knowledge and where rule/constraints adhering with it are automatically inferred from the available log.

Finally, we recall from Section 8.3 that logs have been viewed in the chapter as multi-set of traces, by disregarding the data involved over the activities. Therefore,

it is natural to look for extensions of the proposed mining techniques that could deal with context information (about, e.g., parameters and functional features of the activities) and process ontologies. Moreover, we observe that we have implicitly considered throughout the chapter a setting such that, whenever an activity is executed, then the corresponding event is registered in the log. In some cases, however, we might have to deal with *hidden* activities, i.e., activities that exist in the process model but that are not registered in the log. This is very often the case for activities corresponding to routing constructs, which therefore give rise to logs that are "incomplete" in a sense that is completely orthogonal to the one we have considered in this thesis. Indeed, algorithms tailored to deal with hidden activities have been also proposed in the literature, such as the $\alpha\#$ algorithm Wen *et al.* [2010], which however assumes (as usual) that the log registers all the possible traces for the underlying process. In our experiments, we have considered $\alpha\#$ too, and we observed that it is outperformed by the standard α algorithm—this is not surprising given that our experimental setting does not consider models with hidden activities (where $\alpha\#$ might be more effective). In fact, extending our approach with techniques inspired by $\alpha\#$ and conducting experimentations over process models with hidden activities are further interesting avenues of research.

CHAPTER 10

An Application Scenario (Beyond Process Mining): Urban Congestion

10.1 Introduction

So far, we have analyzed our algorithms within standard process mining settings, but it is not difficult to envisage that they have wider applicability. In this chapter, we complete the picture by discussing a different kind of application motivated by the TETRis research project,[1] funded by the Italian Ministry of Education, University and Research and within the general frame of reference "Internet of Things" supporting Smart City/Smart Territory, in which the acquisition of data by objects can be applied to large territorial areas by the widespread availability of communication networks [European Commission, 2012], [http://www.internet- of-things-research.eu/documents.htm]. In more detail, the activities described in this chapter are related to the identification of scenarios for the realization of innovative services aimed at an intelligent management of a urban and suburban territory.

Within these activities innovative solutions and technology platforms have been identified in order to enable a new way of working for the business entities of the area of interest (municipalities, provinces, regions, universities, etc.) and for citizens and operators involved. Infact, its extended and enhanced the wealth of information

[1] PON Project 01_00451.

acquired on the territory through the use of sensors and devices interconnected by local and remote communications systems that support high added value services to improve the quality of the territory itself in terms of livability and sustainability through the involvement of citizens who become the main tutors of the territory, the so-called "social sensors", for the detection of critical situations related to the context in which they live.

10.2 Project goals

TETRis project main objective is to create high added value services within Smart City and Smart Territory context [Kanter and Litow, 2009], [Komninos et al, 2011] also extending, wherever possible, the functions of TETRA communication system and acting along three main axes: The evolution and the opening of the application fields of TETRA communication system in order to define new information services for operators, exploiting new models and open source tools for the interconnection of TETRA with other networks and the identification of new type of devices obtained through TETRA system interoperability with existing sensors and sensor networks; The modeling and prototyping of an Open Source framework that allows to define a Smart Objects cooperation model and a Smart Objects management within the related Smart Environments; The identification of scenarios and application models in the perspective of Smart City/Smart territory services through the definition of Smart Environments and high added value Open Source services applied to territory monitoring, emergency management, urban and suburban mobility and services to citizens.

The specific objectives of TETRis project can be read as follows: Bring economic and social benefits to the community through more targeted and effective actions by

Public Administration and Public Security operators in different application scenarios such as emergency management, environmental protection, mobility and services to citizens, with the contribution of the same citizens through the sharing of information and the use of innovative tools for social networking; Extend the pervasiveness and effectiveness of public administration services, instrumental bodies, local police, health operators, transport companies in the reference areas; Improving the quality of life and the sense of safety of citizens through the spreading of safe and reliable technology infrastructures "always on".

10.3 Urban log mining

Within this project, the goal of our application is to discover knowledge about congestion dynamics in a urban transportation network. The analysis was performed on data extracted from a large collection of raw mobility records, concerning the circulation of buses (basically, over non-preferential roads/streets) across a network of two neighboring Italian cities (Cosenza and Rende) during year 2012. Each of these records registers the fact that a certain bus (following a certain route) occupied a given geographical position (identified via GPS coordinates) at a certain time.

The analysis of congestion dynamics was carried out by preliminary converting such raw data in the form of a process log, as explained next. First, we transformed the original data into a collection of bus *moves*, each indicating the transfer of a bus from one bus stop to the subsequent one. For any move m, let $pos_{from}(m)$ and $pos_{to}(m)$ denote its initial and final positions (i.e., bus stops), respectively, let $time(m)$ be the time when the move was completed, and let $duration(m)$ be the travel time of m. The set of all the pairs $\langle pos_{from}(m), pos_{to}(m)\rangle$ over all the moves m constitutes the alphabet of the symbols for the log we are going to construct. Each

pair is also called a *route segment*. For each bus, the sequence of route segments associated with the movements of the bus in a given day constitutes one trace in a base version of the log.

The base version is subsequently filtered by removing from each trace the segments that are *not* associated with an abnormally long travel time. In particular, a move m is considered as an anomalous event if $duration(m)$ is greater than 1.3 times the average duration of the moves over the same segment. Note that while removing the moves that are not anomalous, it might happen that two consecutive moves m_1 and m_2 in a trace result to be completely unrelated, in that either $pos_{from}(m_2) \neq pos_{to}(m_1)$, or $time(m_2) - duration(m_2) - time(m_1) > \Delta_t$, with Δ_t being a threshold fixed to 15 minutes in our experiments. In these cases, the trace is broken in two sub-traces, one ending with m_1 and the other starting with m_2, and the analysis is repeated until a fixed point is reached. Eventually, two additional dummy activities are placed at the beginning and at the end of each trace.

Starting with a setting with about 300000 moves and 3000 segments, by exploiting the procedure described above, we obtained a non-linear log over 1743 activities and 2883 traces. Note that the log is focused on the segments over which abnormal travel times have been registered (which witness some traffic congestion) and, in particular, each trace is meant to report the propagation of the congestion across the transportation network. In fact, by analyzing the resulting log with a process mining method, it is now easily seen that one can derive a (conceptual) process model representing how congestions "flow" over the network. In fact, the approach is close in the spirit to the work by [Liu et al., 2011]. The main difference is that in our setting the resulting flow model is not limited to a tree as in Liu et al. [2011] (which basically covers only branching constructs modeling scenarios where a congestion

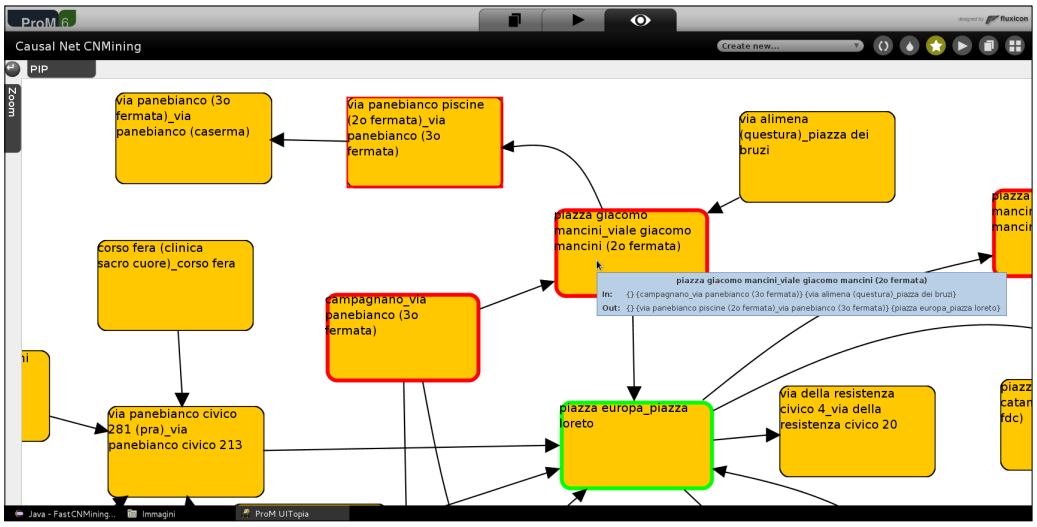


Figure 10.1: A screenshot of the plug-in *CNMining* implementing the algorithms presented in the paper. The causal net is drawn by exploiting the *Flex* interface available in ProM, where bindings are displayed inside tooltips activated by hovering the mouse over the corresponding node.

is split over two streets), but it can be an arbitrary process model with different routing constructs (in particular, synchronizing constructs modeling scenarios where congestions with different origins are merged together).

Figure 10.1 shows (portions of) one causal net discovered from the events recorded during April and May 2012, within a specific geographical area with diameter of about 8 Km—in this case, we have 108 activities and 52 traces. Each node in the model of Figure 10.1 is labeled with the names of the two bus stops it identifies. According to a process-oriented interpretation of anomaly traces, an edge in the model indicates that a delay in one route segment (the origin of the edge) is likely to propagate to another segment (the ending point). Therefore, in order to be adherent to this semantics, there cannot be an edge in the model from a segment to a subsequent adjacent segment along the same (oriented) route. Similarly, there cannot be an edge from a segment to a non-adjacent one. For these reasons, it comes with no surprise that standard approaches to process discovery might be not appropriate in

this application scenario, as they are not able to properly accommodate this kind of background knowledge.

Instead, after they were equipped with this knowledge, our algorithms produced models that truly reflect the semantics of the congestions. Moreover, discovered models emerged to highlight the problems inherent to the network. Actually, in this case, there is no "true" congestion model to be used to quantify the quality of the findings and to conduct experiments on the measures reported in Table 9.7.2. On the other hand, the setting is congenial to stress the scalability of the algorithms w.r.t. the number of the activities (which is large differently from classical applications of process mining). Indeed, recall from Section 9.6 that a factor in the scaling is $n_t \times n_a \times l_t$, where n_t is the number of traces, n_a the number of activities, and l_t the trace length. While in general l_t might linearly grow with n_a,[2] the scaling we registered (when varying the diameter of the area) was actually linear w.r.t. n_a, too. The reason is that even over the whole original log, 90% of the traces have length 6 at most, so that l_t acts as a fixed constant. This comes with no surprise, as we are considering causal correlations over anomalies.

[2]The quadratic scaling w.r.t. the number of activities is intrinsic to the computation of the causal scores, and hence characterizes most of the process discovery algorithms in the literature. In general, it might be avoided by considering less accurate heuristics (fixing the size of a sliding window over the trace), or decomposition approaches as the one recently proposed by [van der Aalst, 2012].

Part IV

Conclusions

CHAPTER 11

Conclusion and Future Research

11.1 Contributions

In this dissertation we studied Constraint satisfaction as a general framework that allows us to describe many computational problems arising in various areas of research, such as database theory, game theory and process mining. We look for classes of instances where such problems are tractable (islands of tractability), as well as for heuristics that are able to deal with real-world instances, possibly exploiting suitable properties of the instance structure. We summarize next the main contributions discussed in the dissertation.

11.1.1 Databases

In this area of research, we studied the problem of counting solutions in SQL queries specifying "COUNT" aggregates. We proposed a novel structural decomposition notion that extends the framework of greedy tree projections by adding weights. An experimental evaluation of the implementation is conducted, where the algorithm is contrasted with the available algorithms for computing hypertree decompositions (because there are no further algorithms dealing with the general framework). We also designed optimization algorithms which exploit our novel structural notion to identify classes of queries that are structurally tractable. We integrated our hybrid

optimizer inside a well-known relational DBMS and we compared the performances of our approach with those of the traditional optimizer. The results evidenced the effectiveness of our tool.

11.1.2 Game theory

We study coalitional game theory, which provides a solid mathematical framework to analyze scenarios where agents con obtain higher payoffs by collaborating with each other rather than by acting in isolation. In particular, we consider the class of allocation games, which has been originally defined by [Moulin, 1992] as a way to analyze fair division problems where monetary compensations are allowed and utilities are quasi-linear. In these contexts (and when monetary transfers are possible), the prototypical solution concept for them is the Shapley value, whose computation is known to be intractable, formally #P-complete. Motivated by this bad news, the dissertation investigates structural requirements that can be used to isolate islands of tractability. We study parameters of the underlying allocation scenarios which allow us to single out islands of tractability. In more detail, we show that computing the Shapley value, as well as the related concept of Banzhaf value, is feasible in polynomial time when each good is of interest for at most two agents, or when interactions among agents have a tree-like structure. We observe that these games capture scenarios of practical interest. For instance, we analyzed the data of one concrete instantiation for the setting described by [Greco and Scarcello, 2014b]. The setting refers to an allocation problem arising in the Italian Research Assessment program. The instantiation refers to the data of the University of Calabria, by discovering that the treewidth of the associated graph, consisting of more than 500 nodes, is 9 only. Moreover, the result has its own theoretical interest, as the analysis of the complexity of reasoning problems (even specifically related to coalitional games) over structures

of bounded treewidth is an active topic of research in artificial intelligence. Note that the tractability result has been established by encoding the problem of computing the Shapley value of allocation games $\mathcal{G}_A$ in terms of counting problems over suitably defined CSP instances [Dechter, 2003], and by using known tractability results for CSP instances having bounded treewidth. Indeed, based on the notion of treewidth, islands of tractability for counting problems have been recently singled out [Pichler and Skritek, 2013]. Our results evidence that subtle issues come into play when trying to reuse such recent results. In fact, while being focused on the Shapley value, the methodology we propose has a wider spectrum of applicability, and its salient ideas might be profitably reused in other contexts. Part of this work appeared as: Gianluigi Greco, Francesco Lupia and Francesco Scarcello Structural Tractability of Shapley and Banzhaf Values in Allocation Games in the Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI-15).

11.1.3 Process mining

While studying process mining applications, it emerged from our analysis that top-down design methods and bottom-up process discovery techniques have been almost separate worlds, so far. Indeed, while the use of background knowledge for improving the quality of results has already been considered in a number of traditional data mining tasks (on relational data and on sequence data) including pattern mining and clustering (see, e.g., [Guns et al., 2011; de Amo and Furtado, 2007]), designing counter-parts of such techniques in the context of process discovery is still largely an open issue. In this thesis, we move a further step to synergically integrate top-down design methods and bottom-up process discovery techniques. Indeed, we propose a "hybrid" approach to process discovery, where a mining method is conceived which

	Traditional	Goedertier et al. [2009]	van der Werf *et al.* [2009]	Here
edge constraints	□	■	■	■
path constraints	□	□	□	■
constraints for parallelism	□	■	■	■
constraints over sets	□	□	□	■

Table 11.1: Summary of constraint support.

can take into account a wide variety of constraints over the causal dependencies that are possibly available to the analyst. This is particularly useful for circumventing the problems emerging when log completeness does not hold. In more detail,

(1) We propose a formal framework to specify additional properties on the process models that can be produced as output by process discovery algorithms. The framework is based on defining a set of *precedence constraints* over the activities, and it supports the kind of prior knowledge that is usually available to the analyst. Table 11.1.3 summarizes the features of our framework, by comparing them with those supported (to some extent) by earlier approaches in the literature. Note that "traditional" process discovery methods do not provide any support to deal with prior knowledge.

(2) In the light of our formulation, process discovery can be conceptually viewed as a *mining task* (i.e., building all possible models for a given input log) followed by a *reasoning task* (i.e., filtering out those models that do not satisfy the precedence constraints defined by the analyst). However, exponentially many process models might be built in general as a result of the mining phase, hence a literal implementation of such a two-phase approach is unfeasible. In fact, we identify relevant classes of constraints where the two tasks can be addressed synergically, and we propose efficient algorithms to deal with them.

(3) We analyze the computational complexity of the proposed setting, by taking into account various qualitative properties regarding the kinds of constraints being al-

lowed, and by tracing the tractability frontier w.r.t. them. In particular, we show that for the classes of constraints that are not covered by the algorithms discussed in the point **(2)** above, an efficient solution algorithm is unlikely to exist at all, because process discovery turns out to be NP-hard over them.

(4) All the algorithms discussed in the thesis have been implemented and integrated in a prototype system, which is made available as a plug-in for the well-known process mining suite *ProM* van Dongen *et al.* [2005]. In particular, to face the above intractability results, our efficient solution algorithms (originally conceived for special cases only) are generalized by making them applicable as heuristic solution approaches for arbitrary classes of constraints. Case studies are illustrated. Results for the experimental activity we have conducted in order to validate the effectiveness of the proposed approach are also reported.

Part of this work has appeared as: Gianluigi Greco, Antonella Guzzo, Francesco Lupia and Luigi Pontieri. Process Discovery under Precedence Constraints. ACM Transactions on Knowledge Discovery from Data, 9(4): 32 (2015).

11.2 Future research directions

The arguments discussed in this dissertation give us natural directions for future research. In general, our idea is that is possible to develop new hybrid solvers that can effectively use structural properties to address the problem of computing all solutions to CSPs efficiently. In more detail, the most important direction for future research and applications to the database setting, is the one that integrates structural approaches with new developments in join processing, see, e.g., [Ngo *et al.*, 2014].

In the field of game theory, and in particular in coalitional games, a future research of direction consists in extending our structural tractability results to other solutions

concept such as the nucleolus.

Finally in process mining, an important direction for future research consists in extending the framework of precedence constraints in the setting of block-structured process models.

Bibliography

A. Atserias, M. Grohe, and D. Marx. Size bounds and query plans for relational joins. *SIAM Journal on Computing*, 42(4):1737–1767, 2013.

G. Bagan, A. Durand, and E. Grandjean. On acyclic conjunctive queries and constant delay enumeration. In *Proc. of CSL'07*, pp. 208–222, 2007.

A. Bulatov, V. Dalmau, M. Grohe, and D. Marx. Enumerating Homomorphisms. *Journal of Computer and System Sciences*, 78(2): 638–650, 2012.

Isolde Adler. Tree-related widths of graphs and hypergraphs. *SIAM J. Discret. Math.*, 22(1):102–123, February 2008.

Martin Grohe. The complexity of homomorphism and constraint satisfaction problems seen from the other side. *Journal of the ACM*, 54(1), Article 1, 2007.

G. Greco and F. Scarcello. Structural tractability of enumerating CSP solutions. *Constraints*, 18(1):8–74, 2013.

Hung Q Ngo, Christopher Ré, and Atri Rudra. Skew strikes back: New developments in the theory of join algorithms. *SIGMOD Rec.*, 42(4):5–16, February 2014.

Francesco Scarcello, Gianluigi Greco, and Nicola Leone. Weighted hypertree decompositions and optimal query plans. *J. Comput. Syst. Sci.*, 73(3):475–506, 2007.

R. Dechter and J. Pearl. Network-based heuristics for constraint-satisfaction problems. *Artif. Intell.*, 34(1):1–38, December 1987.

Georg Gottlob, Nicola Leone, and Francesco Scarcello. A comparison of structural CSP decomposition methods. *Artificial Intelligence*, 124(2):243 – 282, 2000.

G. Greco and F. Scarcello. Fair division rules for funds distribution: The case of the italian research assessment program (vqr 2004-2010). *Intelligenza artificiale*, 7 (1):45–56, 2013.

G. Greco and F. Scarcello. The power of tree projections: Local consistency, greedy algorithms, and larger islands of tractability. In *Proceedings of the Twenty-ninth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '10, pages 327–338, New York, NY, USA, 2010. ACM.

Tractable hypergraph properties for constraint satisfaction and conjunctive queries. In *Proceedings of the Forty-second ACM Symposium on Theory of Computing*, STOC '10, pages 735–744, New York, NY, USA, 2010. ACM.

H.L. Bodlaender and F.V. Fomin. A Linear-Time Algorithm for Finding Tree Decompositions of Small Treewidth. SIAM Journal on Computing, 25(6), pp. 1305–1317, 1996.

Georg Gottlob, Nicola Leone, and Francesco Scarcello. Hypertree decompositions and tractable queries. *Journal of Computer and System Sciences*, 64(3):579 – 627, 2002.

Isolde Adler, Georg Gottlob, and Martin Grohe. Hypertree width and related hypergraph invariants. *European Journal of Combinatorics*, 28(8):2167 – 2181, 2007.

EuroComb 05 - Combinatorics, Graph Theory and ApplicationsEuroComb 05 - Combinatorics, Graph Theory and Applications.

Georg Gottlob, Zoltán Miklós, and Thomas Schwentick. Generalized hypertree decompositions: Np-hardness and tractable variants. *J. ACM*, 56(6):30:1–30:32, September 2009.

P.A. Bernstein and N. Goodman. The power of natural semijoins. *SIAM Journal on Computing*, 10(4), pp. 751–771, 1981.

R. Fagin. Degrees of acyclicity for hypergraphs and relational database schemes. *Journal of the ACM*, 30(3):514–550, 1983.

Leonid Libkin. *Elements Of Finite Model Theory (Texts in Theoretical Computer Science. An Eatcs Series)*. SpringerVerlag, 2004.

Surajit Chaudhuri. An overview of query optimization in relational systems. In *Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, PODS '98, pages 34–43, New York, NY, USA, 1998. ACM.

E. F. Codd. A relational model of data for large shared data banks. *Commun. ACM*, 13(6):377–387, June 1970.

S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

J. D. Ullman. *Principles of Database and Knowledge Base Systems*. Computer Science Press, 1989.

Phokion G. Kolaitis and Moshe Y. Vardi. Conjunctive-query containment and constraint satisfaction. *J. Comput. Syst. Sci.*, 61(2):302–332, October 2000.

Marc Gyssens, Peter G. Jeavons, and David A. Cohen. Decomposing constraint satisfaction problems using database techniques. *Artif. Intell.*, 66(1):57–89, March 1994.

H. Aziz and B. de Keijzer. Shapley meets shapley. In *Proc. of STACS'14*, pp. 99–111.

H. Aziz, O. Lachish, M. Paterson, and R. Savani. Power indices in spanning connectivity games. In *Proc. of AAIM'09*, pp. 55–67.

Y. Bachrach and Y. S. Rosenschein. Power in threshold network flow games. pages 106–132, 2009.

J.F. Banzhaf. Weighted Voting Doesn't Work: A Mathematical Analysis. *Rutgers Law Rev.*, 19:317–343, 1965.

E. H. Bareiss. Sylvester's Identity and Multistep Integer-Preserving Gaussian Elimination. *Mathematics of Computation*, 22:565–565, 1968.

C. J. Colbourn, J.S. Provan, and D. Vertigan. The complexity of computing the tutte polynomial on transversal matroids. *Combinatorica*, 15(1):1–10, 1995.

P. Dagum and M. Luby. Approximating the permanent of graphs with large factors. *Theor. Comput. Sci.*, 102(2):283–305, 1992.

Xiaotie Deng and Christos H. Papadimitriou. On the complexity of cooperative solution concepts. *Mathematics of Operations Research*, 19:257–266, May 1994.

G. Gottlob, G. Greco, and F. Scarcello. Treewidth and hypertree width. In *Tractability: Practical Approaches to Hard Problems*. Cambridge University Press, 2013.

G. Gottlob, S. Tien Lee, G. Valiant, and P. Valiant. Size and Treewidth Bounds for Conjunctive Queries. *Journal of the ACM*, 59(3), 2012.

G. Greco and F. Scarcello. Counting solutions to conjunctive queries: structural and hybrid tractability. In *Proc. of PODS'14*, pp. 132–143.

G. Greco and F. Scarcello. Mechanisms for fair allocation problems: No-punishment payment rules in verifiable settings. *Journal of Artificial Intelligence Research*, 49:403–449, 2014.

F. Maniquet. A characterization of the Shapley value in queueing problems. *Journal of Economic Theory*, 109(1):90–103, 2003.

D. Mishra and B. Rangarajan. Cost sharing in a job scheduling problem. *Social Choice and Welfare*, 29(3):369–382, 2007.

H. Moulin. An application of the Shapley value to fair division with money. *Econometrica*, 60(6):1331–49, 1992.

H. Nagamochi, D.-Z. Zeng, N. Kabutoya, and T. Ibaraki. Complexity of the minimum base game on matroids. *Mathematics of Operations Research*, 22:146–164, 1997.

N. Nisan, T. Roughgarden, É. Tardos, and V. V. Vazirani. *Algorithmic Game Theory*. Cambridge University Press, Cambridge, UK, 2007.

M. J. Osborne and A. Rubinstein. *A Course in Game Theory*. The MIT Press, Cambridge, MA, USA, 1994.

R. Pichler and S. Skritek. Tractable counting of the answers to conjunctive queries. *Journal of Computer and System Sciences*, 79(6):984–1001, 2013.

G. Pesant. Counting solutions of CSPs: a structural approach. In *Proc. of IJCAI'05*, pp. 260–265, 2005.

N. Robertson and P.D. Seymour. Graph minors iii: Planar tree-width. *Journal of Combinatorial Theory, Series B*, 36(1):49–64, 1984.

D. Schmeidler. The nucleolus of a characteristic function game. *SIAM Journal of Applied Mathematics*, 17(6):1163–1170, 1969.

L. S. Shapley. A value for n-person games. *Contributions to the theory of games*, 2:307–317, 1953.

Arya Adriansyah, Boudewijn F. van Dongen, and Wil M. P. van der Aalst. 2011. Conformance Checking Using Cost-Based Fitness Analysis. In *Proc. of EDOC'11*. 55–64.

Rakesh Agrawal, Dimitrios Gunopulos, Frank Leymann. 1998. Mining process models from workflow logs. In *Proc. of EDBT'98*. 469–483.

Marco Alberti, Federico Chesani, Marco Gavanelli, Evelina Lamma, Paola Mello, and Paolo Torroni. 2008. Verifiable agent interaction in abductive logic programming: The SCIFF framework. *ACM Transactions on Computational Logic* 9 (2008), 29:1–29:43. Issue 4.

Nikolas Anastasiou, Tzu-Ching Horng, and William Knottenbelt. 2011. Deriving generalised stochastic Petri net performance models from high-precision location tracking data. In *Proc. of ValueTools'11*. 91–100.

Krzysztof Apt. 2003. *Principles of Constraint Programming*. Cambridge University Press, New York, NY, USA, 2003.

Paul C. Attie, Munindar P. Singh, E. Allen Emerson, Amit P. Sheth, and Marek Rusinkiewicz. 1996. Scheduling workflows by enforcing intertask dependencies. *Distributed Systems Engineering* 3, (1996), 222–238.

Elena Bellodi, Fabrizio Riguzzi, and Evelina Lamma. 2010. Probabilistic Declarative Process Mining. In *Proc. of KSEM'10*. 292–303.

Anthony J. Bonner. 1999. Workflow, Transactions, and Datalog. In *Proc. of PODS'99*. 294–305.

A.A. Bulatov, M. Dyer, L.A. Goldberg, M. Jerrum, and C. Mcquillan. The expressibility of functions on the boolean domain, with applications to counting CSPs. *Journal of the ACM*, 60(5), Article 32, 2013.

A.A. Bulatov. The complexity of the counting constraint satisfaction problem. *Journal of the ACM*, 60(5), Article 34, 2013.

Andrea Burattin and Alessandro Sperduti. 2011. PLG: A Framework for the Generation of Business Process Models and Their Execution Logs. In *BPM Workshops*. 214–219.

Kevin C. W. Chen, David Y. Y. Yun. 2003. Discovering Process Models from Execution History by Graph Matching. In *Proc. of IDEAL'03*. 887–892.

Federico Chesani, Evelina Lamma, Paola Mello, Marco Montali, Fabrizio Riguzzi, and Sergio Storari. 2009. Exploiting Inductive Logic Programming Techniques for Declarative Process Mining. *Transactions on Petri Nets and Other Models of Concurrency* 2 (2009), 278–295.

Sandra de Amo and Daniel A. Furtado. 2007. First-order temporal pattern mining with regular expression constraints. *Data & Knowledge Engineering* 62 (2007), 401–420. Issue 3.

Ana Karla Alves de Medeiros, Boudewijn F. van Dongen, Wil M. P. van der Aalst, and A. J. M. M. Weijters. 2004. *Process Mining: Extending the α-algorithm to Mine Short Loops*. Technical Report. University of Technology, Eindhoven.

Ana Karla Alves de Medeiros, A.J.M.M. Weijters, and Wil M. P. van der Aalst. 2007. Genetic process mining: an experimental evaluation. *Data Mining and Knowledge Discovery* 14 (2007), 245–304.

Luc De Raedt, Tias Guns, and Siegfried Nijssen. 2008. Constraint programming for itemset mining. In *Proc. of KDD'08*. 204–212.

Jochen De Weerdt, Manu De Backer, Jan Vanthienen and Bart Baesens. 2011. A robust F-measure for evaluating discovered process models. In *Proc. of CIDM'11*. 148–155.

Jochen De Weerdt, Manu De Backer, Jan Vanthienen, and Bart Baesens. 2012. A multi-dimensional quality assessment of state-of-the-art process discovery algorithms using real-life event logs. *Information Systems* 37, 7 (2012), 654 – 676.

Rina Dechter. 1992. Constraint networks. *Encyclopedia of Artificial Intelligence* (1992), 276–285.

Claudio Di Ciccio and Massimo Mecella. 2012. Mining Constraints for Artful Processes. In *Proc. of BIS'12*. 11–23.

Paul Dourish, Jim Holmes, Allan MacLean, Pernille Marqvardsen, and Alex Zbyslaw. 1996. Freeflow: Mediating Between Representation and Action in Workflow Systems. In *Proc of CSCW'96*. 190–198.

Hugo M. Ferreira and Diogo R. Ferreira. 2006. An Integrated Life Cycle for Workflow Management Based on Learning and Planning. *International Journal on Cooperative Information Systems* 15, 4 (2006), 485–505.

Michael R. Garey and David S. Johnson. 1979. *Computers and Intractability. A Guide to the Theory of* NP*-completeness*. Freeman and Comp., NY, USA.

Stijn Goedertier, David Martens, Jan Vanthienen, and Bart Baesens. 2009. Robust Process Discovery with Artificial Negative Events. *Journal of Machine Learning Research* 10 (2009), 1305–1340.

Gianluigi Greco, Antonella Guzzo, and Luigi Pontieri. 2007. An Information-Theoretic Framework for Process Structure and Data Mining. *International Journal of Data Warehousing and Mining* 3, 4 (2007), 99–119.

Gianluigi Greco, Antonella Guzzo, and Luigi Pontieri. 2012. Process Discovery via Precedence Constraints. In *Proc. of ECAI'12*. 366–371.

Gianluigi Greco, Antonella Guzzo, Luigi Pontieri, and Domenico Saccà. 2006. Discovering Expressive Process Models by Clustering Log Traces. *IEEE Transactions on Knowledge and Data Engineering* 18, 8 (2006), 1010–1027.

C.P. Gomes, W.J. Van Hoeve, A. Sabharwal, and B. Selman. Counting CSP solutions using generalized XOR constraints. In *Proc. of AAAI'07*, pp. 204–209, 2007.

Tias Guns, Siegfried Nijssen, and Luc De Raedt. 2011. Itemset mining: A constraint programming perspective. *Artificial Intelligence* 175 (2011), 1951–1983.

Markus Hammoria, Joachim Herbsta, and Niko Kleinerb. 2006. Interactive workflow mining – requirements, concepts and implementation. *Data & Knowledge Engineering* 56, 1 (2006), 41–63.

Joachim Herbst and Dimitris Karagiannis. 2000. Integrating Machine Learning and Workflow Management to Support Acquisition and Adaptation of Workflow Models. *Journal of Intelligent Systems in Accounting, Finance and Management* 9 (2000), 67–92.

Joachim Herbst and Dimitris Karagiannis. 2003. Workflow mining with InWoLvE. *Computers in Industry. Special Issue: Process/Workflow Mining* 53, 3 (2003), 245–264.

Haiyang Hu, Jianen Xie, and Hua Hu. 2011. A Novel Approach for Mining Stochastic Process Model from Workflow Logs. *Journal of Computational Information Systems* 7, 9 (2011), 3113–3126.

C. Koch. Processing queries on tree-structured data efficiently. In *Proc. of PODS'06*, pp. 213–224, 2006.

B. Kimelfeld and Y. Sagiv. Incrementally computing ordered answers of acyclic conjunctive formulas. In *Proc. of NGITS'06*, pp. 33–38, 2006.

Gregor Joeris. 2000. Decentralized and Flexible Workflow Enactment Based on Task Coordination Agents. In *Proc. of AOIS'00+CAiSE'00*. 41–62.

Wei Liu, Yu Zheng, Sanjay Chawla, Jing Yua and Xie Xing. 2011. Discovering spatio-temporal causal interactions in traffic data streams. In *Proc. of KDD'11*. 1010–1018.

Ruopeng Lu, Shazia Sadiq, Vineet Padmanabhan, and Guido Governatori. 2006. Using a Temporal Constraint Network for Business Process Execution. In *Proc. of ADC'06*. 157–166.

Fabrizio Maria Maggi, R. P. Jagadeesh Chandra Bose, and Wil M. P. van der Aalst. 2012. Efficient Discovery of Understandable Declarative Process Models from Event Logs. In *Proc. of CAiSE'12*. 270–285.

Fabrizio Maria Maggi, Marlon Dumas, Luciano Garcia-Banuelos, and Marco Montali.

2013. Discovering Data-Aware Declarative Process Models from Event Logs. In *Proc. of BPM'13*. 81–96.

Fabrizio Maria Maggi, Arjan J. Mooij, and Wil M. P. van der Aalst. 2011. User-guided discovery of declarative process models. In *Proc. of CIDM'11*. 192–199.

Siegfried Nijssen, Tias Guns, and Luc De Raedt. 2009. Correlated itemset mining in ROC space: a constraint programming approach. In *Proc. of KDD'09*. 647–656.

Maja Pesic, Dragan Bosnacki, and Wil M. P. van der Aalst. 2010. Enacting Declarative Languages Using LTL: Avoiding Errors and Improving Performance. In *Proc. of SPIN'10*. 146–161.

Maja Pesic, Helen Schonenberg, and Wil M. P. van der Aalst. 2009. DECLARE Demo: A Constraint-based Workflow Management System. In *Proc. of BPM'09 (Demos)*.

Maja Pesic, M. H. Schonenberg, Natalia Sidorova, and Wil M. P. van der Aalst. 2007. Constraint-Based Workflow Models: Change Made Easy. In *Proc. of OTM'07*. 77–94.

Manfred Reichert, Stefanie Rinderle-Ma, and Peter Dadam. 2009. Transactions on Petri Nets and Other Models of Concurrency II. Springer-Verlag, Berlin, Heidelberg, Chapter Flexibility in Process-Aware Information Systems, 115–135.

Anna Rozinat and Wil M. P. van der Aalst. 2008. Conformance checking of processes based on monitoring real behavior. *Information Systems* 33, 1 (2008), 64–95.

Shazia Wasim Sadiq, Maria E. Orlowska, and Wasim Sadiq. 2005. Specification and validation of process constraints for flexible workflows. *Information Systems* 30, 5 (2005), 349–378.

Thomas J. Schaefer. 1978. The complexity of satisfiability problems. In *Proce. of STOC'78*. 216–226.

Guido Schimm. 2003. Mining Most Specific Workflow Models from Event-Based Data. In *Proc. of BPM'03*. 25–40.

Pinar Senkul, Michael Kifer, and Ismail H. Toroslu. 2002. A logical Framework for Scheduling Workflows Under Resource Allocation Constraints. In *Proc. of VLDB'02*. 694–702.

Seppe K.L.M. van den Broucke, Jochen De Weerdt, Jan Vanthienen, and Bart Baesens. 2013. A comprehensive benchmarking framework (CoBeFra) for conformance analysis between procedural process models and event logs in ProM. In *Proc. of CIDM'13*. 254–261.

Wil M. P. van der Aalst. 1998. The Application of Petri Nets to Worflow Management. *Journal of Circuits, Systems, and Computers* 8, 1 (1998), 21–66.

Wil M. P. van der Aalst. 2011. *Process Mining: Discovery, Conformance and Enhancement of Business Processes* (1st ed.). Springer Publishing Company, Incorporated.

Wil M. P. van der Aalst. 2012. Decomposing Process Mining Problems Using Passages. In *Proc. of ICATPN'12*. 72–91.

Wil M. P. van der Aalst, Arya Adriansyah, and Boudewijn F. van Dongen. Replaying history on process models for conformance checking and performance analysis. *Wiley Interdisc. Rew.: Data Mining and Knowledge Discovery* 2, 2 (2012), 182–192.

Wil M. P. van der Aalst, Arya Adriansyah, and Boudewijn F. van Dongen. 2011. Causal nets: a modeling language tailored towards process discovery. In *Proc. of CONCUR'11*. 28–42.

Wil M. P. van der Aalst, Jörg Desel, and Ekkart Kindler. 2002. On the Semantics of EPCs: A Vicious Circle. In *Proc. of EPK'02*. 71–80.

Wil M. P. van der Aalst, Maja Pesic, and Helen Schonenberg. 2009. Declarative workflows: Balancing between flexibility and support. *Computer Science - R&D* 23, 2 (2009), 99–113.

Wil M. P. van der Aalst, Boudewijn F. van Dongen, Joachim Herbst, Laura Maruster, Guido Schimm, and A.J.M.M. Weijters. 2003. Workflow Mining: A Survey of Issues and Approaches. *Data & Knowledge Engineering* 47, 2 (2003), 237–267.

Wil M. P. van der Aalst and Kees M. van Hee. 2002. *Workflow Management: Models, Methods, and Systems*. MIT Press.

Wil M. P. van der Aalst, A.J.M.M. Weijters, and Laura Maruster. 2004. Workflow Mining: Discovering Process Models from Event Logs. *IEEE Transactions on Knowledge and Data Engineering* 16, 9 (2004), 1128–1142.

Jan Martijn van der Werf, Boudewijn F. van Dongen, Cor A. J. Hurkens, and Alexander Serebrenik. 2009. Process Discovery using Integer Linear Programming. *Fundamenta Informaticae* 94 (2009), 387–412. Issue 3-4.

Boudewijn F. van Dongen, Ana Karla Alves de Medeiros, H.M.W. Verbeek, A.J.M.M. Weijters, and Wil M. P. van der Aalst. 2005. The ProM Framework: A New Era in Process Mining Tool Support. In *Proc. of ATPN'05*, 1105–1116.

Jacques Wainer and Fabio de Lima Bezerra. 2003. Constraint-Based Flexible Workflows. In *Proc. of CRIWG'03*, 151–158.

A.J.M.M. Weijters and Wil M. P. van der Aalst. 2001. Process Mining: Discovering Workflow Models from Event-Based Data. In *Proc. of BNAIC'01*. 283–290.

A.J.M.M. Weijters, Wil M. P. van der Aalst, and Ana Karla Alves de Medeiros. 2006. *Process Mining with the HeuristicsMiner Algorithm*. Technical Report. Eindhoven University of Technology, Eindhoven.

A.J.M.M. Weijters and Wil M. P. van der Aalst. 2003. Rediscovering workflow models from event-based data using Little Thumb. *Integrated Computer-Aided Engineering* 10, 2 (2003), 151–162.

Lijie Wen, Jianmin Wang, Wil M. P. van der Aalst, Biqing Huang, and Jiaguang Sun. 2009. A novel approach for process mining based on event types. *J. on Intelligent Information Systems* 32, 2 (2009), 163–190.

Lijie Wen, Jianmin Wang, Wil M. P. van der Aalst, Biqing Huang, and Jiaguang Sun. 2010. A novel approach for process mining based on event types. *Data & Knowledge Engineering* 69, 10 (2010), 999–1021.

Moe Thandar Wynn, Chun Ouyang, Arthur H.M. ter Hofstede, and Colin J. Fidge. 2009. *Workflow Support for Product Recall Coordination*. Technical Report. BPM-center.org.

M. Yannakakis. Algorithms for acyclic database schemes. In *Proc. of VLDB'81*, pp. 82–94, 1981.