# Learning concise constraint models from error-free data : studies on learning Boolean-arithmetic equations and short-term scheduling models

Ramiz Gindullin

COLLEGES SCIENCES
BRETAGNE POUR L'INGENIEUR
LOIRE ET LE NUMERIQUE

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

# THÈSE DE DOCTORAT DE

L'ÉCOLE NATIONALE SUPÉRIEURE
MINES-TÉLÉCOM ATLANTIQUE BRETAGNE
PAYS DE LA LOIRE – IMT ATLANTIQUE

ÉCOLE DOCTORALE N° 648
*Sciences pour l'Ingénieur et le Numérique*
Spécialité : *Informatique*

Par
**Ramiz GINDULLIN**

# Learning concise constraint models from error-free data : studies on learning Boolean-arithmetic equations and short-term scheduling models

**Thèse présentée et soutenue à IMT Atlantique, Nantes, le 25 mars 2024**
**Unité de recherche : Laboratoire LS2N**
**Thèse N° : 2024IMTA 0393**

## Rapporteurs avant soutenance :

| | |
|---|---|
| Nadjib Lazaar | HDR, Maître de Conférence, Université de Montpellier |
| Jean-Charles Régin | Professeur, Université Côte d'Azur, Nice |

## Composition du Jury :

| | | |
|---|---|---|
| Président : | François Clautiaux | Professeur, Institut mathématiques de Bordeaux |
| Examinateurs : | Nadjib Lazaar | HDR, Maître de Conférence, Université de Montpellier |
| | Jean-Charles Régin | Professeur, Université Côte d'Azur, Nice |
| Dir. de thèse : | Nicolas Beldiceanu | Professeur, IMT Atlantique, Nantes |

# Acknowledgement

I want to thank everyone who assisted me on this three-year journey.

First of all, I want to thank my supervisor, Nicolas Beldiceanu. He taught me a lot about the intricacies of constraint programming and research in general. He also showed me a lot of kindness and wisdom.

I also want to thank my parents, Vil Gindullin and Rimma Gindullina, and my sister, Elvina Gindullina, for supporting me during this time.

Last but not least, I want to thank my colleagues and friends. I made a lot of good memories during my work on the thesis. They gave me the strength to complete the thesis. I want to acknowledge my good friends (in no particular order): Jovial Cheukam-Ngouonou, Matthew Coyle, Charles Vernerey, Geo-Johns Anthony, Celia Kessassi, Michele Rigon, Luo Dan, Yan Xin, Antoine Omond and Chadia Ed-driouch.

# Table of Contents

# List of Tables

# Résumé

La grande majorité des techniques d'apprentissage automatique visent à faire des prédictions sur des données partielles contenant du bruit. Il en résulte des modèles probabilistes, souvent dans le continu, minimisant les erreurs de prédiction.

Cependant, dans certains cas cités ci-dessous, toutes les données fournies sont exemptes de bruit et d'erreurs, et l'objectif est de trouver un modèle qui décrit de manière concise et précise chaque instance de l'ensemble de données:

— Dans le cadre de données historiques portant sur des exemples de production à court terme, l'objectif est d'apprendre un modèle d'ordonnancement capable de produire de nouveaux ordonnancements valides,

— dans le cadre d'une table donne chaque entrée donne pour une combinaison de paramètres d'objet combinatoire la borne précise pouvant être prise par un autre paramètre, le but est d'apprendre des conjectures donnant la valeur de la borne en fonction des paramètres d'entrées.

Dans ce cadre, l'objectif d'apprendre un modèle portant sur des variables entières, valide qui décrit de manière concise les relations entre les éléments fournis par un ensemble d'exemples positifs.

En général, l'objectif n'est pas seulement de décrire l'ensemble de données en terme d'éléments d'un langage de modélisation, mais également de générer de nouveaux ensembles de données valides, par exemple la valeur d'une borne précise pour un nouveau jeu de paramètres d'entrées, ou un nouvel ordonnancement de production. Les techniques d'apprentissage automatique probabiliste sont insuffisantes pour de telles tâches, car elles produisent des modèles qui ne sont pas exacts à cent pour cent. Par exemple, les techniques probabilistes d'apprentissage automatique peuvent produire une conjecture ne correspondant pas à une borne précise pour une caractéristique de sortie donnée et un ensemble donné d'entrées.

Pour ces raisons, cela nous amène à utiliser différentes techniques d'apprentissage spécialisées pour travailler dans le domaine discret et produire un modèle expliquant les données dans leur intégralité sans introduire d'erreur. La principale difficulté est le surapprentissage, c'est-à-dire que lorsque l'on applique une approche de régression symbolique

pour rechercher une conjecture, on obtient souvent une formule polynomiale compliquée avec une douzaine de termes individuels qui n'ont aucune signification intuitive. Une telle conjecture sera souvent tout simplement erronée.

Par conséquent, il est nécessaire de choisir soigneusement les méthodes que l'on utilise dans un tel contexte. Le fait de savoir qu'il n'y a aucune d'erreurs dans l'ensemble de données permet la recherche de caractéristiques pertinentes en s'appuyant sur les dépendances fonctionnelles pour orienter le processus d'apprentissage.

Cette thèse étudie l'acquisition de modèles discrets à partir de données sans erreurs pour deux cas d'utilisation se recoupant en partie : (*i*) la recherche d'équations booléennes arithmétiques dans le contexte de conjectures pour des objets combinatoires, étendant le travail effectué par Beldiceanu *et al.* [1], et (*ii*) l'acquisition de modèle d'ordonnancement à court terme à partir d'un seul exemple d'ordonnancement valide. Dans le cadre de cette thèse, un certain nombre de techniques ont été mises au point. La plupart d'entre elles utilisent la programmation par contraintes et produisent finalement un modèle à contraintes discrètes.

## Programmation par contraintes et acquisition de modèles

La programmation par contraintes (PPC) est une approche visant à résoudre des problèmes combinatoires. Sa principale caractéristique, la distinguant d'autres techniques, est qu'elle met l'accent sur la faisabilité plutôt que sur l'optimisation. Ainsi, la PPC vise à résoudre des problèmes de satisfaction de contraintes (CSP), c'est-à-dire à trouver une solution qui satisfasse toutes les contraintes données [2]. Les solveurs de contraintes prennent en charge plusieurs types de contraintes :

— Des *contraintes arithmétiques*, c'est-à-dire des contraintes combinant des opérateurs arithmétiques sur des variables discrètes et des constantes avec un opérateur de comparaison '$\leq$', '$\geq$' ou '$=$'. Par exemple, une contrainte arithmétique typique d'un problème d'ordonnancement est la manière dont la fin d'une tâche $i$, $End[i]$, est calculée en fonction de son début, $Start[i]$, et de sa durée, $Duration[i]$ :

$$End[i] = Start[i] + Duration[i] \tag{1}$$

Un autre exemple de contrainte arithmétique est le suivant

$$f \leq v - \underline{p}, \tag{2}$$

où $f$ est le nombre de feuilles dans un arbre enraciné, $v$ est le nombre de sommets d'un arbre enraciné, et $\underline{p}$ est la profondeur minimale d'une feuille dans un l'arbre enraciné. La profondeur d'une feuille est le nombre d'arêtes entre la feuille et la racine de l'arbre.

Les contraintes arithmétiques sont également utilisées en dehors de la PPC, par exemple dans la programmation linéaire ou non linéaire en nombres entiers.

— Des *contraintes logiques ou réifiées*, c'est-à-dire des contraintes qui, en plus des opérations arithmétiques, utilisent également les opérateurs logiques ' $\implies$ ' et '$\equiv$'. Par exemple, si une paire de tâches $i$ et $j$ satisfait la condition $\mathit{Successor}[i] = j$, alors la condition $\mathit{End}[i] \leq \mathit{Start}[j]$ doit également être satisfaite. Cette expression peut s'exprimer à l'aide de la contrainte réifiée suivante :

$$\mathit{Successor}[i] = j \implies \mathit{End}[i] \leq \mathit{Start}[j] \tag{3}$$

— Les *contraintes globales*, c'est-à-dire des contraintes qui expriment une condition générale sur un ensemble de variables de manière concise [3]. Par exemple, une contrainte globale DISJUNCTIVE [4, 5] peut être utilisée pour imposer une restriction sur l'ensemble des tâches affectées à une même ressource de manière à ce qu'elles ne se chevauchent pas.

Les contraintes sont utiles dans une grande variété d'applications telles que la recherche opérationnelle (OR) [6], la bio-informatique [7], la chimie [8], et la vérification de logiciels [9]. Les contraintes sont un outil puissant qui allie flexibilité et performance et qui, contrairement aux approches de type "boîte noire", est facile à expliquer.

Un langage de modélisation populaire utilisé pour exprimer les modèles de contraintes de manière simple est MiniZinc [10]. MiniZinc est un langage de modélisation indépendant du solveur qui tire parti d'une vaste bibliothèque de contraintes, y compris les contraintes de planification des ressources les plus courantes [5]. Comme la plupart des solveurs de programmation par contraintes (par exemple, Google OR-Tools, CHOCO, SICStus Prolog) ou des solveurs MIP (par exemple, CPLEX, Gurobi) comprennent directement MiniZinc, il est possible d'exécuter un modèle MiniZinc avec n'importe lequel de ces solveurs.

La construction et l'utilisation efficace de ces contraintes dans la pratique exigent beaucoup d'expertise de la part d'un utilisateur, car il doit se familiariser avec la grande variété de contraintes pour savoir lesquelles seraient pertinentes pour une application donnée ou, le cas échéant, pour développer de nouveaux types de contraintes. C'est pourquoi un certain nombre d'outils d'acquisition de contraintes ont vu le jour. Leur objectif commun est à partir d'un ensemble de données, de déterminer quelle contrainte individuelle ou quel ensemble de contraintes individuelles décrit l'ensemble des données de manière pertinente et concise. Le deuxième objectif de l'acquisition de modèles est la recherche de modèles pouvant être ensuite utilisés pour produire de nouvelles solutions valides.

Chaque technique d'acquisition de modèle est limitée par les biais d'apprentissage qu'elle utilise. Un biais d'apprentissage, également connu sous le nom de biais inductif, est "l'ensemble des hypothèses que l'apprenant utilise pour expliquer les données fournies" [11]. La nécessité d'avoir un biais dans l'apprentissage a été étudiée dès 1980 par Mitchell dans [12]. Dans cette thèse, le biais d'apprentissage se réfère au type de formule que nous essayons d'apprendre, par exemple des formules polynomiales, des formules conditionnelles, des formules par cas, ou finalement des formules booléennes.

## Les contributions

La première contribution principale de cette thèse est que ni une conjecture complexe ni un modèle d'ordonnancement ne peuvent être appris en une seule étape :

1. *Les méthodes de décompositions basées sur les données* que nous avons introduites sont essentielles pour acquérir des conjectures complexes, c'est-à-dire des conjectures impliquant des *formules avec des sous-termes imbriqués mélangeant différents biais d'apprentissage*.

   De telles conjectures complexes peuvent survenir lors de la recherche de bornes précises pour les caractéristiques d'objets combinatoires. Considérons, par exemple, l'objet ARBRE ENRACINÉ. Un ARBRE ENRACINÉ $\mathcal{T}$ est "un graphe non orienté dans lequel deux sommets quelconques sont reliés par exactement un chemin, et dans lequel un sommet est appelé *racine*. Le *parent* d'un sommet $v$ est le sommet connecté à $v$ sur le chemin menant à la racine, et l'*enfant* d'un sommet $v$ est un sommet dont $v$ est le parent. Une *feuille* est un sommet sans enfant." [13]

   La conjecture (8) fournit des bornes inférieure et supérieure précises sur le nombre de feuilles $\ell$ d'un arbre enraciné $\mathcal{T}$ par rapport au nombre de sommets $n$ de $\mathcal{T}$,

le nombre minimum d'enfants $\underline{d}$ des sommets non enracinés de $\mathcal{T}$, et le nombre maximum d'enfants $\overline{d}$ des sommets non enracinés de $\mathcal{T}$, qui ont été trouvés en utilisant une approche de décomposition :

$$\underline{d} = 0 \Rightarrow \ell = 1 \wedge \underline{d} > 0 \Rightarrow \\ \ell \in \left[ \left\lceil \frac{n \cdot \underline{d} + \overline{d} - \underline{d} - n + 1}{\underline{d}} \right\rceil, \left\lfloor \frac{n \cdot \overline{d} + \underline{d} - \overline{d} - n + 1}{\overline{d}} \right\rfloor \right] \quad (4)$$

La validité de la conjecture (8) a été prouvée par Jovial Cheukam-Ngouonou dans [14].

2. *Consolider et combiner les informations acquises provenant de diverses sources* est crucial. Lors de l'acquisition de modèles de planification, il est essentiel de ne pas acquérir indépendamment les contraintes temporelles, les contraintes de ressources, les contraintes de calendrier et les contraintes fonctionnelles; en effet, cela permet de focaliser la recherche de modèles d'ordonnancement en s'appuyant sur les informations acquises à chaque étape.

La deuxième contribution principale de cette thèse est la création d'un nouveau biais d'apprentissage, à savoir les *formules arithmétiques booléennes*, dans le cadre du Bound Seeker [1]. Ce biais s'est avéré essentiel pour l'acquisition de conjectures sur des bornes précises, car de nombreuses conjectures consistent en plusieurs cas qui peuvent être encapsulés dans une seule formule utilisant une formule booléenne arithmétique. Lors de la mise en œuvre de ce biais, un modèle de contrainte de base a été développé, qui a ensuite été étendu en ajoutant des contraintes de cassage de symétrie, ainsi que des contraintes dites d'anti-réécriture. Une *base de données de contraintes d'anti-réécriture a été synthétisée de manière systématique par un programme basé sur les contraintes.*

La principale contribution pratique de cette thèse est de *considérer à la fois l'acquisition de formules complexes et l'acquisition de modèles d'ordonnancement au sein de la même plateforme d'acquisition*, à savoir l'extension du Bound Seeker à partir du travail de Beldiceanu *et al.* [1], et son amélioration pour concilier nos deux cas d'usage. Les contributions de cette thèse au Bound-Seeker préexistant à mon arrivée sont les suivantes :

1. Le développement d'un modèle à contraintes pour acquérir des équations booléennes arithmétiques à partir de données (voir Sections 6.2–6.4).
2. Le développement d'une méthode pour acquérir des expressions conditionnelles avec des conditions complexes qui sont trouvées en acquérant les équations booléennes

arithmétiques (voir Section 6.5.1).

3. Le développement d'une méthode d'acquisition de formules par cas en tant qu'arbres de décisions compacts qui sont trouvées en acquérant des équations booléennes arithmétiques (voir Section 6.5.2).
4. Le développement de quatre techniques de décomposition pour acquérir des conjectures complexes combinant plusieurs biais d'apprentissage (voir Chapitre 7).
5. La possibilité pour l'utilisateur de sélectionner les entrées et les sorties d'un modèle (voir Section 4.1.1.2).
6. La recherche automatisée des clés primaires et secondaires et la possibilité de fusionner deux tables décrivant respectivement un ensemble de tâches et un ensemble de ressources en une seule table (voir Sections 5.1, 5.2 et 4.1.4).
7. La capacité à classer les dépendances fonctionnelles candidates afin de sélectionner les dépendances fonctionnelles les plus probables (voir section 5.3).
8. L'acquisition de contraintes temporelles, c'est-à-dire de contraintes correspondant à des chaînes de précédences généralisées entre des tâches (voir Section 8.1).
9. L'acquisition de contraintes de ressources, c'est-à-dire des contraintes globales liées à l'affectation des tâches aux ressources (voir Section 8.2).
10. L'acquisition de contraintes de calendrier et d'utilisation de l'atelier par les équipes, c'est-à-dire des contraintes qui prennent en compte différents types de disponibilité ou d'indisponibilité (voir Section 8.3).
11. La capacité de générer un modèle d'ordonnancement MiniZinc qui peut être utilisé pour générer un nouvel ordonnancement valide (voir Chapitre 9).

En outre, la qualité d'acquisition des différentes contributions a été évaluée sur deux ensembles de données générés :

12. un ensemble de données utilisé pour l'acquisition de conjectures de bornes inférieures et supérieures sur diverses caractéristiques de huit objets combinatoires qui consiste en une collection de 252300 tables (voir Section 2.1) [15].
13. Un ensemble de données ouvertes de programmes de production à court terme composé de 48000 tables (voir Section 10.1) [16].

# Plan de la thèse

Cette thèse se compose des cinq parties suivantes:

— En gardant à l'esprit les deux cas d'utilisation de cette thèse, à savoir l'acquisition de conjectures et l'apprentissage de modèles d'ordonnancement, la partie I fournit les informations de base pertinentes pour cette thèse. Elle se compose d'un chapitre.

- Le chapitre 1 fournit une vue d'ensemble de l'état de l'art pour l'acquisition de contraintes, l'acquisition de conjectures, l'acquisition d'équations booléennes arithmétiques et d'arbres de décision.

— La partie II décrit les deux cas d'utilisation étudiés dans cette thèse, chacun faisant l'objet d'un chapitre.

- Le chapitre 2 décrit le premier cas d'utilisation dans lequel nous recherchons des conjectures pour des bornes inférieures et supérieures précises de diverses caractéristiques de huit objets combinatoires. En outre, ce chapitre contient des exemples de conjectures complexes que le système a pu trouver avec les nouveaux biais d'apprentissage introduits dans cette thèse. Certaines des conjectures complexes acquises ont été prouvées par Jovial Cheukam-Ngouonou dans [14], ce qui signifie que le système a trouvé des conjectures valides non triviales.

  Le chapitre 2 utilise (*i*) nos articles publiés [1, 15], (*ii*) un article récemment accepté [17], et notre (*iii*) article [18] qui, au moment de la rédaction de cette thèse, est en cours d'examen dans la revue Constraints. Notez que l'article [18] est une extension de l'article [15].

- Le chapitre 3 décrit le deuxième cas d'utilisation dans lequel nous acquérons un modèle de contrainte à partir d'un seul exemple d'ordonnancement à court terme valide.

— La partie III fournit une description du système dénommé le Bound Seeker présenté pour la première fois dans [1], puis étendu dans cette thèse. Il se compose de deux chapitres.

- Le chapitre 4 fournit une description détaillée de la fonctionnalité du système Bound Seeker. Une partie du système génère des métadonnées qui sont largement utilisées par le système.
- Le chaptitre 5 détaille les différentes contributions de cette thèse au processus de génération de métadonnées, c'est-à-dire la génération de clés primaires et secondaires et le classement des dépendances fonctionnelles, qui sont tous les

deux cruciaux pour l'acquisition d'un modèle d'ordonnancement où les données d'entrée sont généralement réparties sur plus d'une table.

— La partie IV décrit les différentes contributions de la thèse à la recherche de formules complexes pour les deux cas d'utilisation. Elle se compose de deux chapitres :

- C-Le chapitre 6 décrit le processus d'acquisition de nouveaux biais d'apprentissage ajoutés : équations booléennes arithmétiques, expressions conditionnelles avec des conditions complexes et arbres de décision simples.

  Le chapitre 6 utilise notre article publié [15] ainsi qu'un article [18] qui, au moment de la rédaction de cette thèse, est en cours d'examen.

- Le chapitre 7 décrit diverses techniques de décomposition pour combiner divers biais d'apprentissage.

  Le chapitre 7 utilise notre article accepté, au moment de la rédaction de cette thèse, à savoir [17].

— La partie V décrit comment nous acquérons des modèles d'ordonnancement à court terme à partir d'un seul exemple positif. Elle se compose de trois chapitres :

- Le chapitre 8 décrit comment acquérir diverses contraintes d'ordonnancement telles que les contraintes temporelles, les contraintes de ressources, les contraintes d'équipes et les contraintes de calendrier.
- Le chapitre 9 décrit comment convertir le modèle d'ordonnancement acquis en un modèle MiniZinc.
- Le chapitre 10 fournit une évaluation détaillée de l'acquisition de modèles de planification à court terme à partir d'un seul exemple.

Finalement, la Conclusion résume toutes les contributions décrites dans les chapitres 5–10 de cette thèse.

# Introduction

The vast majority of machine learning (ML) techniques aim to make predictions on partial and noisy data. This results in probabilistic models, often on continuous domains, which minimises predictive errors.

However, in some cases, all provided data is free of noise and errors, and the goal is to find a model that concisely and precisely describes each instance of the dataset, e.g.:

— a valid short-term production schedule (STPS) for a company, with the goal to learn a scheduling model which can produce new schedules,

— a table with invariants wrt to a given sharp bound of a mathematical object, with the goal to learn conjectures between invariants which can later be proved mathematically,

— a list of valid player moves from an unspecified game, with the goal of learning the rules of that game.

The main objective of these examples is to learn a valid model that describes the relationships between the elements provided for each input of the given dataset, often based on discrete models. Usually, the goal is not just to describe the dataset in a modelling language, but also to generate new valid datasets, e.g. a new production schedule. Probabilistic machine learning techniques are insufficient for these tasks as they produce models that are not 100% accurate. For example, probabilistic machine learning techniques may produce a conjecture that violates a sharp bound for a given output characteristic and a given set of input characteristic values of a learning dataset. This will render the acquired conjecture useless.

This leads us to utilise different ML techniques that are specialised to work in discrete domains and produce a model which describes the dataset as a whole without any error. The main difficulty is overfitting, i.e. if one applies a symbolic regression approach to search for conjectures, one often will get a complicated polynomial formula with a dozen of individual terms that have no mathematical meaning. An overfitted conjecture would often be simply wrong.

As a result, a careful choice of instruments is required, often designed to solve a given problem individually. Knowing that there are no errors in the dataset simplifies

the search for relevant features by allowing to use functional dependencies to focus the learning process.

This thesis studies the acquisition of meaningful models from error-free data for two, somewhat overlapping, use cases: (*i*) the search for Boolean-arithmetic equations in the context of conjectures for combinatorial objects, extending the work done by Beldiceanu *et al.* [1], and (*ii*) the acquisition of STPS from an example of a valid schedule. A number of techniques have been developed, most of which use constraint programming and eventually produce a constraint model.

# Constraint programming and model acquisition

Constraint programming (CP) is an approach aimed at solving combinatorial problems. Its key feature which puts it apart from other is a focus on feasibility rather than optimisation. Thus, CP is aimed at solving constraint satisfaction problem (CSP), i.e. finding a solution which satisfy all given constraints [2]. CP solvers supports several types of constraints:

— *arithmetic constraints*, i.e. constraints which combine arithmetic operators over variables and constants with a comparison operator '$\leq$', '$\geq$' or '$=$'. For example, a typical arithmetic constraint common in STPS is how the end time of a task $i$, $End[i]$, is calculated for given start time of the task $i$, $Start[i]$, and duration of the task $i$, $Duration[i]$:

$$End[i] = Start[i] + Duration[i] \tag{5}$$

Another example of an arithmetic constraint is

$$f \leq v - \underline{p}, \tag{6}$$

where $f$ is the number of leaves in a single tree, $v$ is a number of vertices of a single tree and $\underline{p}$ is a minimum depth of a leaf in the rooted tree. A depth of a leaf is the number of edges between the leaf and the root of the tree.

Arithmetic constraints are also used outside of CP, e.g. in mixed-integer linear or non-linear programming.

— *logical or reified constraints*, i.e. constraints which, in addition to arithmetical operations also uses logical operators ' $\Longrightarrow$ ' and '$\equiv$'. For example, if a pair of tasks

$i$ and $j$ satisfy the condition $Successor[i] = j$ then the condition $End[i] \leq Start[j]$ must also be satisfied. This statement can be expressed with a reified constraint:

$$Successor[i] = j \implies End[i] \leq Start[j] \tag{7}$$

— *global constraints*, i.e. constraints that express a condition over some variables in a concise manner independent from the context [3]. For example, a global constraint DISJUNCTIVE [4, 5] can be used to impose a restriction on a set of tasks assigned to a unique resource in such a way that they do not overlap with each other.

Constraints are useful in a wide variety of applications such as operations research (OR) [6], bioinformatics [7], chemistry [8], and software verification [9]. Constraints are a powerful tool that both combines flexibility and performance and, unlike 'black box' approaches, is easy to explain.

A popular modelling language used to express constraint models in an easy way is MiniZinc [10]. MiniZinc is a solver-independent modelling language which takes advantage of a large library of constraints including the most common resource scheduling constraints [5]. As most constraint programming solvers (e.g. Google OR-Tools, CHOCO, SICStus Prolog), or MIP solvers (e.g. CPLEX, Gurobi) understand MiniZinc, one can directly execute a MiniZinc model with any of these solvers.

To construct and effectively use such constraints in practice requires a lot of expertise from a researcher as one must be familiar with the wide variety of constraints to know which would be useful for a given application or, if needed, to develop new types of constraints. Thus a number of constraint acquisition tools were developed. Their goal is to read a given dataset and determine which individual constraint or which set of individual constraints describes the dataset in an effective manner, i.e. it is general enough to avoid overfitting while retaining the meaningful information, i.e. it is not too general. The aim of model acquisition is the search of models that can then be used to produce new valid solutions.

Each model acquisition technique is restricted by the learning biases it uses. A learning bias, also known as an inductive bias, is "the set of assumptions that the learner uses to predict outputs of given inputs" [11]. The need for bias in learning was explored as early as 1980, by Mitchell in [12]. In this thesis, the learning bias refers to the type of formula we are trying to learn, e.g. polynomial formulae, conditional formulae, case formulae, Boolean formulae.

# Contributions

The first main contribution of this thesis is that neither a complex conjecture nor a scheduling model can be learned in a single step, i.e. :

1. *Data-driven decompositions* are crucial to acquire complex conjectures, i.e. conjectures involving *formulae with nested sub-terms using different learning biases.*

   Such complex conjectures can occur during the search for sharp bounds for the characteristics of combinatorial objects. Consider, for example, a ROOTED TREE. A ROOTED TREE $\mathcal{T}$ is "an undirected graph in which any two vertices are connected by exactly one path, and in which one vertex has been called the root. The *parent* of a vertex $v$ is the vertex connected to $v$ on the path to the root, and the *child* of a vertex $v$ is a vertex of which $v$ is the parent. A *leaf* is a vertex without children." [13]

   Conjecture (8) provides a sharp lower bound and a sharp upper bound on the number of leaves $\ell$ of a rooted tree $\mathcal{T}$ wrt the number of vertices $n$ of $\mathcal{T}$, the minimum number of children $\underline{d}$ of the non-leave vertices of $\mathcal{T}$, and the maximum number of children $\overline{d}$ of the non-leave vertices of $\mathcal{T}$, that was found by using a decomposition approach:

   $$\underline{d} = 0 \Rightarrow \ell = 1 \wedge \underline{d} > 0 \Rightarrow \ell \in \left[ \left\lceil \frac{n \cdot \underline{d} + \overline{d} - \underline{d} - n + 1}{\underline{d}} \right\rceil, \left\lfloor \frac{n \cdot \overline{d} + \underline{d} - \overline{d} - n + 1}{\overline{d}} \right\rfloor \right] \quad (8)$$

   The validity of Conjecture (8) was proved by Jovial Cheukam-Ngouonou in [14].

2. *Consolidating and combining acquired information stemming from various sources.* When acquiring scheduling models, it is essential not to acquire temporal constraints, resource constraints, calendar constraints and functional constraints independently, as this permit focusing the search of scheduling models.

The second main contribution of this thesis is the creation of a new learning bias, namely *Boolean-arithmetic formulae*, for the Bound Seeker [1]. This has proved to be an essential bias in acquiring conjectures about sharp bounds, since many conjectures consist of several cases that can be encapsulated in a single formula using a Boolean-arithmetic formula. When implementing this bias, a core constraint model was developed, which then later, was enhanced by adding symmetry-breaking constraints, as well as so-called anti-rewriting constraints. *A database of anti-rewriting constraints* was synthesised in a systematic way by a separate constraint program.

The main practical contribution of this thesis is *to consider both the acquisition of complex formulae and the acquisition of scheduling models within the same acquisition platform*, namely extending the Bound Seeker from the work of Beldiceanu *et al.* [1], and enhancing it to accommodate our two use cases simultaneously. The contributions of this thesis to the system include:

1. The development of a CP model to acquire Boolean-arithmetic equations from data (see Sections 6.2–6.4).
2. The development of a method to acquire conditional expressions with complex conditions which are found by acquiring the Boolean-arithmetic equations (see Section 6.5.1).
3. The development of a method for acquiring case formulae as simple decisions trees which are found by acquiring Boolean-arithmetic equations (see Section 6.5.2).
4. The development of four decomposition techniques to acquire complex conjectures combining two or more learning biases (see Chapter 7).
5. The ability for the user to select inputs and outputs of the scheduling constraint model (see Section 4.1.1.2).
6. The automated search for primary and foreign keys and the ability to merge two tables into one (see Sections 5.1, 5.2 and 4.1.4).
7. The ability to rank candidate functional dependencies in order to select the most likely genuine functional dependency (see Section 5.3).
8. The acquisition of temporal constraints, i.e. constraints corresponding to chains of generalised precedences between tasks (see Section 8.1).
9. The acquisition of resource constraints, i.e. global constraints related to the assignment of tasks to resources (see Section 8.2).
10. The acquisition of calendar and shift constraints, i.e. constraints that take into account different types of availability or unavailability (see Section 8.3).
11. The ability to generate a MiniZinc scheduling model that can be later used to generate a new valid schedule (see Chapter 9).

Additionally the acquisition quality of the contributions was evaluated on two generated datasets:

12. a dataset used for acquiring conjectures lower and upper sharp bounds on various characteristics of eight combinatorial objects which consists of a collection of 252300 tables (see Section 2.1) [15].

13. an open dataset of short-term production schedules which consists of 48000 tables (see Section 10.1) [16].

# Outline of the thesis

This thesis consists of the following five parts:

— Having the two use-case of this thesis in mind, namely conjecture acquisition and learning scheduling models, Part I provides the background information relevant to this thesis. It consists of one chapter.

- Chapter 1 provides an overview of the state-of-the-art for constraint acquisition, acquisition of conjectures, acquisition of Boolean-arithmetic equations and decision trees.

— Part II describes the two use cases considered in this thesis, each with one chapter.

- Chapter 2 describes the first use case in which we search for conjectures for sharp lower and upper bounds of various characteristics of eight combinatorial objects. In addition, this chapter contains examples of some complex conjectures that the system acquires with new learning biases. Some of the acquired complex conjectures were proved by Jovial Cheukam-Ngouonou in [14], meaning that the system found some non trivial valid conjectures.

  Chapter 2 uses text from (*i*) our published papers [1, 15], (*ii*) from a recently accepted paper [17], and from (*iii*) paper [18] which, at the time of the writing of this thesis, are under review in the Constraints journal. Note that [18] is an extension of paper [15].

- Chapter 3 describes the second use case in which we acquire a constraint model from a single example of a valid short term schedule.

— Part III provides a description of the Bound Seeker system first introduced in [1], and then extended in this thesis. It consist of two chapters.

- Chapter 4 provides the detailed description of the functionality of the Bound Seeker system. One part of the system generates metadata which is extensively used by the system.

- Chapter 5 details the various contributions of this thesis to the metadata generation process, i.e. the generation of primary and foreign keys and the ranking

of functional dependencies, both of which are crucial to the acquisition of a scheduling model where the input data is typically spread over more than one table.

— Part IV describes various contributions of the thesis to the search of complex formulae for both use cases. It consists of two chapters:

- Chapter 6 describes the process of acquisition of new added learning biases: Boolean-arithmetic equations, conditional expressions with complex conditions and simple decision trees.

  Chapter 6 uses text from our published paper [15] and a paper [18] which, at the time of the writing of this thesis, is under review.

- Chapter 7 describes various decomposition techniques to combine various learning biases into one acquired formulae.

  Chapter 7 uses text from our accepted, at the time of the writing of this thesis, for publishing paper [17].

— Part V describes how we acquire short term scheduling models from a single valid example. It consists of three chapters:

- Chapter 8 describes how to acquire various scheduling constraints such as temporal constraints, resource constraints, shift constraints and calendar constraints.
- Chapter 9 describes how to convert the acquired scheduling model into a MiniZinc model.
- Chapter 10 provides the detailed evaluation of the acquisition of short term scheduling models from a single example.

The Conclusion summarises all the contributions described in Chapters 5–10 of this thesis.

Part I

# Background

Chapter 1

# Background

This chapter provides a state-of-the-art overview of the techniques relevant to this PhD thesis. Section 1.1 provides a description of the state of the art on various constraint acquisition techniques. Section 1.2 provides a description of the state-of-the-art techniques on the discovery of Boolean-arithmetic equations. Section 1.3 provides a description of the state-of-the-art on the discovery of conjectures describing various relations between invariants of mathematical objects.

## 1.1 An overview of constraint acquisition techniques

Constraint acquisition is a machine learning approach to acquire individual constraints and constraint networks from examples of solutions and non-solutions. A constraint network consists of a set of variables with their respective domain, and a set of constraints [19]. The majority of constraint acquisition techniques are aimed at the acquisition of constraint networks on integer domains.

One of the early example of constraint acquisition [20], applied plausible explanation-based learning (PEBL) to acquire constraint scheduling models using both partial and full schedules. They applied the approach on scheduling scenarios of the NASA Space Shuttles.

Padmanabhuni *et al.* [21] proposed a framework for learning and generalising constraints from positive examples. The aim of the proposed framework was the development of automatic constraint acquisition in discrete domains, in contrast to previous work in machine learning that focuses on continuous domains. The paper also highlights the relationship between constraint acquisition and earlier machine learning techniques, namely inductive logic programming (ILP). Some of the ideas used by ILP are relevant to constraint acquisition, primarely the idea stated by Haussler *et al.* that all learning algorithms use an inductive learning bias, "otherwise all possible classifications of unseen instances are equally possible, thus no inductive or, more generally, learning method is better than

a random guessing" [22]. In addition, paper [21] is one of the first mentions of the term 'automatic constraint acquisition'.

Model acquisition was formally initiated by the work of Freuder [23] in the early 2000s. Two directions were initially developed [24]: the *passive* approach relies on examples of labelled solutions, called *positive examples*, and non-solutions, named *negative examples*. The *active* approach assumes an oracle can tell whether an example is a solution or not, and it aims to reduce the number of queries required to learn a model [25]. Both approaches have advantages and limitations:

— Passive learning requires that we have enough diverse and informative labelled solutions and non-solutions; on the other hand, no oracle is needed during the acquisition process.

— Asking many questions to an oracle is not always realistic, especially if the oracle is a human being; on the other hand, if there is a lack of diverse labelled solutions, targeted questions can boost the learning process.

Sammut *et al.*[26] present a prototypical idea of the active learning approach, in which the program learns complex concepts by memorising simpler concepts and asking questions to a user to check whether they describe complex concepts correctly.

For problems with a strong structure, i.e. problems for which the model can be concisely defined using the global constraints [3, 27] available in the MiniZinc modelling language [10, 28], passive learning does not need a lot of labelled solutions and non-solutions.

### 1.1.1 Active learning for the matchmaking problem

An early approach for active learning is first suggested by Freuder [29] and later implemented by Freuder and Wallace [30]. In this work they state a constraint acquisition and satisfaction problem (CASP) and apply it to the matchmaker problem.

CASP is a problem where the constraint solver first acquires the information about the constraints which he later uses to solve the corresponding CSP. In the matchmaker problem, the matchmaking agents represent constraint solvers which provide customers with potential solutions, i.e. suggestions, based on the information provided by a client. The system assumes that the constraint solver and the customer are using constraints from the same 'universe', i.e. the universal set of constraints. The system works as follows:

1. The solver proposes to the customer a solution based on a set of constraints.

2. The customer then evaluates whether or not there are constraints that are violated. These violations are communicated to the solver. It is assumed that the customer is able to articulate each of the constraints if necessary, i.e. express them in a constraint language which is accepted by the system.
3. The solver incorporates these violations, produces a new set of constraints, solves the new CSP, and provides a new solution to the user.
4. Steps 1–3 are repeated until all customer's constraints are satisfied.

The paper [30] evaluates several strategies for producing new sets of constraints.

### 1.1.2 Interactive constraint acquisition and timid acquisition strategy

The idea of interactive constraint acquisition is proposed in [31] by O'Sullivan *et al.* This approach to constraint acquisition differs from that described in Section 1.1.1 in that it does not assume that the user knows how to formulate constraints. Instead, it asks the user for a positive example of a constraint to be acquired, then by applying the *List-Then-Eliminate* algorithm [32] it determines the version space for the user's constraint. A version space is "the set of all classifiers expressible in the language that correctly classify a given set of data" [33]. Then the system attempts to generalise the user's example to produce its own *qualifying example* which is presented to the user. Examples accepted and rejected by the user are used to refine the version space. If the qualifying example is rejected, the user is asked to provide another example.

The paper [34] evaluates several generalization strategies. It concludes that the *Least generalization* strategy works best in situations where the user provides helpful examples.

O'Connell *et al.* [35] extends the proposed approach in [31, 34] from acquiring a single constraint to the acquisition of a constraint network. To achieve this, the system considers only positive examples provided by the user. The reason why the authors do not consider negative examples is that a negative example means that one or more constraints are violated. Determining which constraints are violated is too complex for the constraint solver. There are ways to utilise negative examples in the context of the acquisition of multiple constraints (see Section 1.1.3) but they have their inherent limitations.

O'Connell *et al.* developed the timid acquisition strategy based on Berwick's *subset principle* [36, 37]. According to this principle, the system takes the smallest possible steps to avoid over-generalization. The paper [35] also evaluates different strategies for

selecting examples, and concludes that the *Maximum Differentiating* strategy gave the best results. Since the user cannot be expected to use such an approach, an example of a selection assistant was developed.

### 1.1.3 The CONACQ system

The *CONstraint ACQuisition* (CONACQ) algorithm was introduced by Coletta *et al.* [38] as a system for passive constraint acquisition. It uses positive and negative examples to generate the version space for the constraint network. The main weakness of the approach in [38] is the acquisition of constraint networks that contain redundant constraints: this has a negative impact on the acquisition process, preventing the algorithm from converging to the most specific hypothesises for the target constraint network. Bessiere *et al.* in [39] proposed a number of techniques to handle redundant constraints, namely *redundancy rules* and *backbone detection*.

Furthermore, Bessiere *et al.* reformulate the CONACQ algorithm as a SAT-based version space algorithm, which can also incorporate domain-specific knowledge, if necessary. The CONACQ algorithm was tested on a series of experiments, each with 12 variables and 12 domain values per variable. Bessiere *et al.* observed that the convergence rate improves when domain-specific knowledge was used with redundancy rules and backbone detection, but the computation time increases by up to 25 times: there is a trade-off between computation time and convergence rate.

To demonstrate a practical application of CONACQ's passive constraint acquisition, Paulin *et al.* used it to automatically design the behaviour of autonomous robots.

Bessiere *et al.* proposed in [40] a version of CONACQ which supports the active acquisition of constraint networks: the acquisition system assists the user to select the set of the examples. The system generates membership queries (see [41]) which the user must classify as a solution or a non-solution. A careful selection of queries significantly reduces the number of examples required to learn the target constraint network. The paper considers two strategies of query generation: optimal-in-expectation and optimistic. Their evaluation showed that the optimistic strategy works best in cases where the number of acquired constraints is small, otherwise it was better to use the optimal-in-expectation strategy. Shchekotykhin and Friedrich improved the query generation for CONACQ by allowing a domain expert to provide arguments, i.e. domain-specific knowledge, in addition to classifying examples.

Papers [24, 42] provide an overview of the different versions of CONACQ.

### 1.1.4 The QUACQ system

In [43], Bessiere *et al.* describe an active acquisition algorithm named *QUick ACQuisition* (QUACQ). This algorithm asks the user to classify partial queries. If a query is classified negatively, the algorithm uses this knowledge to converge to a constraint from a target constraint network. The advantages of the proposed approach are:

— it is guaranteed to converge to the target constraint network in a polynomial number of queries,

— queries are shorter than membership queries, making them easier for a user to handle,

— positive examples are not required for the algorithm to converge to a target constraint network.

The algorithm shows weakness during the acquisition of problems with dense constraint networks as the required number of queries becomes too large for a human user to handle, e.g. 8645 queries are needed to acquire a constraint model for a Sudoku puzzle. Bessiere *et al.* in [44] improves the algorithm by adding generalization queries, which allows the user to decide whether or not a learned constraint can be generalized to other scopes of variables of the same type. The new approach, called G-QUACQ, was able to reduce the number of queries by up to 50 times.

An alternative approach was proposed by Daoudi *et al.* in [45], where recommendation queries were introduced. A recommendation query asks a user whether or not a predicted constraint belongs to the target constraint network. To select which constraints are recommended, the PREDICT&ASK algorithm was proposed, as a part of the combined P-QUACQ system. The evaluation showed that P-QUACQ halves the number of queries compared to the QUACQ algorithm.

Unlike the QUACQ system, which produces a single explanation for each negative example, the MULTIACQ algorithm [46] learns multiple explanations for why a user classifies an example as negative The number of necessary queries is up to three times from what is required by QUACQ, meanwhile the size of a query is reduced up to 10 times. Tsouros *et al.* in [47] improve on the MULTIACQ algorithm by (*i*) lowering the complexity of learning multiple constraints after a negative example, (*ii*) focusing the scopes to reduce the number of queries, and (*iii*) generalizing partial queries in such a way that they can be used as examples posted to the users. The resulting algorithm, called MQUACQ, reduces the acquisition time up to 15 times compared to QUACQ. Tsouros

*et al.* in [48] introduced MQUACQ2 which further improves MQUACQ by focusing on the structure of the learned network and by completely removing the scope of the acquired constraint after a negative example.

Bessiere *et al.* in [25, 49] introduced the second version of the algorithm, QUACQ2. In it, the problem of constraint acquisition was reformulated to be in line with standard concept learning [50, 51], i.e. the target constraint network is a subset of the constraints in the basis. Since the original QUACQ algorithm did not adhere to this definition it could lead to the acquisition of a wrong constraint network, i.e. collapsing, or providing the user with redundant queries. In contrast, QUACQ2 never collapses. Additionally, QUACQ2 does not require normalized target constraint networks, i.e. it can learn any type of constraint networks.

### 1.1.5 Soft constraint acquisition systems

Rossi and Sperduti proposed in [52] an active constraint acquisition system which acquires a system of soft constraints and preferences. To achieve this, a learning module is added to a soft constraint solver. The learning module acquires the information about the preferences between provided examples from dialogue with the user. This information can be provided in two ways. The first way is when the user provides the exact ratings of each solution. This allows the learning module adjusting the soft constraints network by minimizing the error function with a gradient descent search. Otherwise, if the user only provides partial information about the solutions, the second way is to utilise reinforcement learning techniques to maximize the expected reward function.

### 1.1.6 The Model Seeker

Started first as a Constraint seeker [53], which acquires a single global constraint from the global constraints catalogue [3], the Model Seeker [54–56] is a passive constraint acquisition tool capable of acquiring a complete constraint network from a limited number of positive solutions, typically less than five. The tool requires that the solutions are provided as flat samples, i.e. as a flat list of integers, of the same size without no additional knowledge, i.e. hints, is given. The model seeker utilises both global constraints and constraint programming assuming that the target constraint network has a strong internal structure, i.e. it can be represented in a very compact way. The latter allows one to use basic templates to group relevant variables together. The workflow of the model seeker

tool is the following:

1. **Transformation** The sample solutions are converted to a normalised format which is compatible with the tool. This produces an input vector of fixed length.
2. **Candidates generation**
   (a) The argument creation phase creates different subsequences from the given input vector. A given subsequence is called a pattern.
   (b) For each pattern generated during step 2a, the tool calls the Constraint seeker to find all matching constraints for all matching patterns across all given samples. The tool supports the acquisition of around 70 constraints from the global constraints catalogue.
3. **Candidates simplification** The tool removes all candidate constraints that are implied by at least one of the candidates, i.e. a so-called dominance check phase. Afterwards, the tool performs the removal of trivial constraints, the simplification of patterns, and the simplification and the ranking of the remaining candidate constraints.
4. **Code generation** The tool generates code for the acquired constraint network.

### 1.1.7 Classifier-based constraint acquisition

Unlike previous supervised learning techniques, the CLASSACQ system proposed by Prestwich *et al.* is an example of an unsupervised machine learning technique. It proposes to use a naive Bayes classifier to categorise a set of given examples as solutions or non-solutions. The system learns simple linear constraints which are then simplified or approximated further, thus the system acquires a soft constraint network. The system showed the following advantages:

— robustness under noise,

— memory requirements are independent of bias size, as only the training data is stored in memory,

— evaluation has shown that it quickly acquires a constraint network,

— the system avoids learning weak constraints, i.e. constraints that are unlikely to be violated unless a large number of examples are provided.

### 1.1.8 Other constraint acquisition techniques

We present in this section a short overview of various constraint acquisition techniques which weren't covered in the previous sections. Some of the mentioned techniques utilise domain-specific knowledge. More information can be found in [51] which summarize state-of-the-art constraint acquisition techniques which were developed by 2018.

#### 1.1.8.1 Acquiring timetabling, scheduling and planning models

Picard-Cantin *et al.* [57, 58] proposed a Branch-and-Bound algorithm to learn parameters of global constraints such as SEQUENCE and AMONG, with a focus on constraints used in timetabling. Gregory and Lindsay [59] use constraint programming to learn planning domain models with action costs from examples. Senderovich *et al.* [60] developed a set of tools to generate basic scheduling models [61] from event data by conversion of acquired timed Petri nets to a constraint model.

#### 1.1.8.2 Inductive logic programming

Mizoguchi and Ohwada [62, 63] and later Kawamura [64] developed techniques to learn spatial constraints and linear algebra constraints in continuous domains. Lallouet *et al.* [65] proposed a framework which bridges the inductive logic programming (ILP) and CP. In it, the positive and negative solutions are presented as interpretations by first-order rules in a chosen logic language, which are then used to produce a CSP expressed in a chosen constraint language.

#### 1.1.8.3 Model acquisition in the context of combinatorial problems

Paulus *et al.* [66] trained neural networks to learn integer linear constraints for combinatorial problems. Pawlak and Krawiec [67, 68] reformulated a constraint acquisition problem as MILP, then applied a conventional MILP solver to synthesise constraints from negative and positive examples. Kolb *et al.* [69] and Paramonov *et al.* [70] introduced Tabular Constraint Learner (TaCLe) which reconstructs the constraints from tabular data in a spreadsheet. It operates in two stages. The first stage analyses blocks of columns and rows, the second stage investigates individual rows and columns. Kumar *et al.* [71, 72] proposed a constraint acquisition approach adapted to find constraints from tensors, as such data structures are often present in OR environments, such as scheduling, assignment, etc. Their approach acquires both first-order constraints and global constraints.

Coulombe and Quimper [73] proposed 'Constraint Acquisition Based on Solution Counting' (CABSC). CABSC reframes the constraint acquisition problem as a Meta-CSP, i.e. a combinatorial problem whose solution is a CSP. Instead of analysis of individual constraints, the system reasons globally across all constraints. This allows considering multiple different constraints at once.

#### 1.1.8.4 Miscellaneous

Lallouet and Legtchenko [74, 75] proposed a Consistency Checking Classifier, based on *open constraints*, i.e. constraints for which there are both positive and negative examples are available, which is later transformed into a constraint. To do this, classifiers are transformed into propagators ensuring the active behaviour of a constraint. Vu *et al.* [76] propose a general framework to unify acquisition algorithms for different types of problems. The framework allows reformulating constraint acquisition problems as optimisation problems and formulate new, more general, acquisition techniques. Belaid *et al.* [77] proposed an active constraint acquisition tool called 'Generic Qualitative Constraint Acquisition' (GEQCA). Its contribution is the development of a generic correct method to learn any kind of qualitative constraints between each pair of entities of a specific problem. Tsouros *et al.* [78] proposed an active constraint acquisition approach which uses guided queries and builds larger constraints from the bottom-up. This allowed to reduce both the waiting time for the user and the number of queries.

## 1.2 Acquiring Boolean-arithmetic expressions

Learning purely Boolean expressions from data is widely reported in the literature. A significant number of papers explore the acquisition of relevant features, often called the "*relevant features problem*" (RFP). Blum formalises the RFP in [79], and provides a survey of various algorithms in [80]. The RFP can be applied to features that are Boolean, integer or continuous, each of which requires its own approach [81, chapter 1.2]. Some of the works focusing on purely Boolean RFP are described in [82–84]. In [85], Mutlu and Oghaz provide a taxonomy of Boolean and non-Boolean feature extraction techniques applied to graphs. Other works present the acquisition of Boolean expressions as part of Boolean rule extraction methods for classification problems using SAT [86] or neural networks [87]. Lastly, some papers [88, 89] focus on the construction and the simplification of Boolean functions.

The acquisition of Boolean-arithmetic expressions is often used in the context of classification problems where random forest [90], decision trees [91–93], Bayesian rule lists [94], fuzzy association rules [95] and rough sets [96] approaches are used.

Most of the work considers the acquisition of relatively simple Boolean-arithmetic expressions of the type "*attribute has a value of*". The SEEN system [93] extracts more complex Boolean-arithmetic expressions that contain the $+$, $\times$ and / arithmetic operators in the context of searching for complex rules in soft decision trees: it calls this topic "*logical-arithmetic expression mining*".

Beyond the domain of Boolean formulae, synthesising formulae from data [97] mostly relies on a generate and test approach to produce candidate formulae of increasing complexity for a fixed grammar. In the context of this thesis, applying techniques that minimise an error function produces complicated formulae that are not verified wrt all input data. In [15] Gindullin *et al.* compared the approach presented in Chapter 6 to methods used for symbolic regression such as GPlearn [98] and ffx [99]: GPlearn generally found no formulae, while ffx discovered formulae with a large number of terms.

As shown above by papers [91–93], Boolean-arithmetic equations are often associated with decision trees. Additionally, Section 6.5.2 of this thesis looks into the acquisition of simple decision trees, mentioning Boolean-arithmetic expressions, and corresponding to case formulae. Thus, the next section provides a short overview of various state-of-the-art techniques used to construct decision trees.

### 1.2.1 Acquiring decision trees

The most common definition of decision trees (DT) is that they are "sequential models, which logically combine a sequence of simple tests; each test compares a numeric attribute against a threshold value or a nominal attribute against a set of possible values" [100]. It should be noted that this definition describes univariate DT, where each test involves only one single attribute. If at least one test considers more than one attribute, then the DT is called multivariate [101]. Multivariate DTs usually contain linear combinations of attributes, but sometimes non-linear tests are also used [102]. Papers [91–93] cited in Section 1.2 construct multivariate DTs.

A number of popular techniques were developed for both univariate and multivariate DTs. For univariate DTs these include C4.5 [103, pp 17–25], CART (Classification and Regression Trees) [104, chapters 2 and 3], SPRINT (Scalable PaRallelizable INduction of decision Trees) [105], SLIQ (Supervises Learning In Quest) [106], Rainforest [107].

Multivariate DTs are addressed by Ittner and Shclosser [108], Yildiz and Alpaydin [102], Altinçay [109], Djukova and Peskov [110], Tharwart *et al.* [111] and Sok *et al.* [112, 113].

Some works consider the acquisition of fuzzy DTs, i.e. DTs where the test on each node of a given DT produces a probabilistic result, in opposite of crispy DTs. Approaches that produce soft DTs are Fuzzy ID3 [114], soft DTs [115], polynomial-fuzzy DTs [116], G-FDT (Gini index based DTs) [117], SEEN (Soft dEcision Tree for logical arithmetic Expressions miNing) [93].

Costa and Pedreira provided a recent general survey on decision trees in [118]. Another recent survey is done by Carrisoza *et al.* [119] which focused on continuous optimization and mixed-integer linear optimization formulations for decision trees. Previous surveys have been carried out by Kotsiantis [100] and Murthy [120].

## 1.3 Automated discovery

The interest in automated discovery of hypothesises between mathematical concepts dates back to 1976. Lenat [121–123] developed two programs called 'AM' and 'EURISKO' to discover new mathematical concepts with a few simple heuristics which then build upon each other in a recursive manner. Later, in 1984, Valiant [124] formalised the concept of learning machines, i.e. machines capable of learning entire classes of non-trivial concepts in a polynomial number of steps, and showed that this was possible to achieve.

A more targeted research, concerned only with uncovering relations between invariants on graphs, was first done by Brigham and Dutton [125, 126] in 1983. They developed the so-called INteractive GRaph Invariant Delimiter (INGRID), a system for acquiring more precise information about partially specified graphs which can be used to derive new theorems or to test already derived theorems. They present the discovered relations between invariants on graphs in [127, 128].

Fajtlowicz [129] developed the computer program 'Graffity' suited to certain types of graphs that acquires about 60 invariants. Fajtlowicz also developed two procedures, IRIN and CNCL, to facilitate (*i*) the search of the conjectures between invariants and (*ii*) the removal of trivial conjectures and of conjectures that can be derived from other conjectures because of transitivity. Larson and Van Cleemput [130, 131] based their search for conjectures on the Dalmatian heuristic proposed in [129]. In [131] showed and proved five novel theorems acquired by their system.

Caporossi and Hansen [132, 133] developed 'AutoGraphiX' a program which does

the opposite task. From an invariant corresponding to a bound, it tries to construct an extremal or near-extremal graph using variable neighbourhood search to sharpen the initial bound. It is also able suggest a proof to or refute a given conjecture automatically. The system uses three approaches to automate acquisition of conjectures - numerical, geometrical and algebraic. In [132] it refuted 9 conjectures of Graffiti [129]. It suggested over 50 novel conjectures, 15 of which were proved. Aochiche *et al.* [134] provide the updated information on the number of uncovered, proven and disproven conjectures on graphs.

While Colton *et al.* [135] are not discovering relations between invariants, they propose both a semi-automated and automated qualitative searches to build and verify new theorems. These verified theorems classify algebras of a particular type and size into isomorphism classes. Colton *et al.* used a variety of techniques including Mace Model generator [136], C4.5 [103, pp 17–25], the Spass theorem prover [137], etc.

Davies *et al.* [138] has proposed, instead of a fully automated search for conjectures, an assisting system for mathematicians to facilitate the discovery of new conjectures. To do this, the mathematician can choose a pair of invariants, then the system uses neural networks to generate a dataset which includes both invariants. Then the system can check whether or not the relation between the invariants is statistically more accurate on further samples than the baseline chance of 25% . If it is, the neural network tries to acquire a conjecture on continuous domains, i.e. a polynomial with non-integer coefficients.

Beldiceanu *et al.* [1] proposed a system called the Bound Seeker which will be described in the Chapter 4, as the current thesis uses this system as a basis to build further.

It should be noted that the aforementioned approaches are not strictly limited to mathematical objects. If necessary, they can produce good results if applied on other kinds of data. e.g. Brooks *et al.* [139] used the Dalmatian heuristic and conjecturing techniques proposed in [129] on COVID-19 datasets to discover relations in the form of the Boolean expressions $(A \wedge B) \vee C \vee (D \wedge E)$, $(A \wedge B) \Rightarrow C$, etc.

Part II

# Use cases

Chapter 2

# First use case: Acquiring conjectures on combinatorial objects

In this chapter, first the problem of the acquisition of maps of conjectures for sharp bounds on combinatorial objects is described to provide the context. Next the reasons on why there is a focus on the acquisition of Boolean-arithmetic equations are provided.

## 2.1 Acquiring sharp bounds for maps of conjectures for combinatorial objects

Our work on acquiring Boolean-Arithmetic Equations is motivated by learning conjectures about sharp bounds on the characteristics of combinatorial objects. The learning process is based on tables, each entry of which is a positive example, specifying the sharp lower (resp. upper) bound of a characteristic of a combinatorial object based on a combination of values for other characteristics. As we look for sharp bounds, the learning process acquires equality, which in the end leads to inequality, since the conjectures concern lower and upper bounds. As all the entries in a table are error-free, we acquire formulae that match all table entries.

**Example 2.1.1.** *For example, a combinatorial object could be a digraph $\mathcal{G}$ whose characteristics are the number of vertices $v$, the number of arcs $a$, and the number of connected components $c$. A sharp upper bound on the number of arcs of $\mathcal{G}$ relative to its number of vertices and its number of connected components is $a \leq (v-(c-1))^2+(c-1)$. This sharp bound would be acquired from Table 2.1, which gives the sharp upper bound on the number of arcs $a$ wrt the number of vertices and connected components of $\mathcal{G}$.*

Although the sharp bound $(v-(c-1))^2+(c-1)$ in the Example 2.1.1 does not mention any Boolean conditions, the following section shows the relevance of Boolean conditions to learn sharp bounds.

We consider eight combinatorial objects for which we generated a dataset:

— **digraph (without isolated vertex):** a set of vertices $\mathcal{V}$ and a set of ordered pairs of vertices $\mathcal{A}$ with the restriction that each vertex of $\mathcal{V}$ occurs in at least one pair of $\mathcal{A}$ [140].

— **rooted tree:** a connected acyclic undirected graph where a vertex is designed as the "root" of the tree [141].

— **rooted forest:** a disjoint union of rooted trees [141]; we also consider a variant, **rooted forest2**, where all rooted trees have at least two vertices.

— **partition:** a partition of a set $\mathcal{S}$ is a collection of possibly empty subsets of $\mathcal{S}$ such that every element of $\mathcal{S}$ is in exactly one of the subsets of the collection. The use of a partition was motivated by the by fact that a partition can be interpreted as a solution to the conjunction of the NVALUE (i.e. the number of partition subsets, see [142]) and the BALANCE (i.e. the difference between the cardinalities of the largest and smallest subsets of the partition, see [53, pp 698–703]) constraints. Motivated by the extension of the BALANCE constraint, i.e. ALL_BALANCE [143], we also consider a version of partition named **partition0** where all subsets of $\mathcal{S}$ are non-empty.

— **stretch**: a solution of a stretch constraint on 0-1 variables, where a subsequence of 1 immediately preceded and followed by a 0 is called a *stretch* [144]; we also consider

Table 2.1 – Example of a table used to acquire the equation $a = (v - (c - 1))^2 + (c - 1)$ leading to the sharp upper bound $a \leq (v - (c - 1))^2 + (c - 1)$; the table gives the maximum number of arcs $a$ of a digraph in relation to its number of vertices $v$ and connected components $c$.

| $v$ | $c$ | $a$ |
|---|---|---|
| 1 | 1 | 1 |
| 2 | 1 | 4 |
| 2 | 2 | 2 |
| 3 | 1 | 9 |
| 3 | 2 | 5 |
| 3 | 3 | 3 |
| 4 | 1 | 16 |
| 4 | 2 | 10 |
| 4 | 3 | 6 |
| 4 | 4 | 4 |

Table 2.2 – Examples of characteristics (char.) of combinatorial objects and corresponding conjectures: (*i*) $c, s, oc, \underline{c}$ and $\overline{s}$: *number of connected components (cc), strongly connected components (scc), connected components with at least two vertices, size of the smallest cc* and *size of the largest scc* of a **digraph**; (*ii*) $c_0$: *denote* 0 *if all the cc have same maximal size, and* $\underline{c}$ *otherwise*, for a **digraph**; (*iii*) $v$ and $f$: *number of vertices* and *leaves* in a **rooted tree**; (*iv*) $\overline{d}$: *largest degree of a parent node* in a **rooted tree** or a **rooted forest**; (*v*) $\underline{p}$ and $\underline{t}$: *minimum depth* and *size of the smallest tree* in a **rooted forest**; (*vi*) $n$, $nval$, and $\underline{m}$: *number of elements*, *number of subsets*, and *cardinality of the smallest subset* in a **partition**; (*vii*) $sr$, $dr$, and $\underline{dm}$: *difference between the number of elements of the largest and smallest stretches*, *difference between the maximum and minimum distance of consecutive stretches*, and *minimum distance between consecutive stretches* in **stretch**; (*viii*) $n$, $ng$, and $osc$: *number of elements, total number of stretches, and number of stretches which have more than one element when the number of element of the largest stretch is maximal* in **cyclic stretch**.

| Combinatorial object | Number of char. | Some of the used char. |
|---|---|---|
| **digraph** | 20 | $c, \underline{c}, s, oc$ |
| **digraph** | 20 | $c_0, c, s, \overline{s}$ |
| **rooted tree** | 6 | $\overline{d}, v, f$ |
| **rooted forest** | 11 | $\underline{p}, \overline{d}, \underline{t}$ |
| **partition** | 14 | $n, nval, \underline{m}$ |
| **stretch** | 26 | $sr, dr, \underline{dm}$ |
| **cyclic stretch** | 26 | $osc, n, ng$ |

the variant named **cyclic stretch** where, when the sequence begins and terminates by 1, those two 1 belong to the same stretch.

The tool described in [1], called Bound Seeker, searches systematically for conjectures related to the acquisition of sharp bounds. The way it organises the search is following:

1. The data generation step. The input data consists of a collection of tables giving for any combinatorial object of size at most *size*, for any combination of at most three input parameters, for any feasible combinations of values of these input parameters, the sharp lower or the sharp upper bound of a given output parameter, e.g. Table (A) of Fig. 7.1 is an excerpt of such an input table. In addition, an input table may also mention auxiliary parameters, so-called secondary parameters, which are all functionally determined by the input parameters of the table.

   **Definition 1.** *For* Digraph, Rooted tree, Rooted Forest, *and* Rooted Forest2, size *denotes the number of vertices. For* Partition0 *and* Partitions, size *is the number of elements of the set we partition, and finally for* Stretch *and* Cyclic Stretch, size *is the sequence length.*

The data set used for acquiring conjectures in [15] and this thesis consists of a collection of 252300 tables representing 12 GB, giving for any combinatorial object of size at most *size*, for any combination of at most three input parameters, for any feasible combinations of values of these input parameters, the sharp lower or the sharp upper bound of a given output parameter.

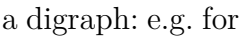
2. For each table the tool searches for relevant functional dependencies for a given output column, for which we search for conjectures, and stores them.
3. For each candidate dependency the tool builds a sub-table where it uses only columns mentioned in the selected FD as inputs with the selected output column.
4. For each sub-table the tool enumerates through a list of learning biases, i.e. a type of formula we attempt to learn. For each bias a corresponding CSP is created for the given sub-table, with constraints posted on each entry. If the CSP is solved successfully, the conjecture is found. The tool supported two types of biases: polynomial formulae with unary and binary operations and conditional formulae of the type $(cond\,?\,x : y)$, which is equel to $x$ if condition $cond$ holds, $y$ otherwise.

## 2.2 The relevance of Boolean-arithmetic equations for learning sharp bounds on combinatorial objects

To show the expressive power of Boolean-arithmetic equations (BAE), let us consider typical uses where they are relevant for acquiring sharp bounds, i.e. an inequality for which the equality holds for at least one example.

1. Using a BAE is a natural option when the codomain of $f(X_1, X_2, \ldots, X_n)$ is equal to $\{0, 1\}$ or more generally consists of only two distinct consecutive values $v$ and $v+1$. For example, let $v, a, \underline{os}$ and $\underline{s}$ be the number of vertices, the number of arcs, the number of of strongly connected components with the smallest number of vertices, and the size of the smallest strongly connected component of a digraph. As the CP model of [1] shows, when $a$ is maximal, we have the relation $\underline{os} = \lfloor \frac{v}{\max(-\underline{s}+v, \underline{s})} \rfloor$, which is equivalent to the relation $\underline{os} = 1 + [v = 2 \cdot \underline{s}]$, where the Boolean expression $[v = 2 \cdot \underline{s}]$ is used as an integer, i.e. either 0 for false or 1 for true.
2. Even when the number of distinct values $m$ of the codomain of $f(X_1, X_2, \ldots, X_n)$

is greater than two values, but still very small, we can use Boolean arithmetic expressions to capture concise formulae. This is done by summing up $m-1$ Boolean-arithmetic conditions as illustrated now: e.g. let $v, a, c_1$ and $\underline{c}$ be the number of vertices, of arcs, of connected components having more than one vertex and the size of the smallest connected component of a digraph. As the CP model of [1] shows, when $a$ is maximal, we have the relation $c_1 = \lfloor \frac{(v+\max(-\underline{c}+v,\underline{c}))}{(2\cdot\max(\max(-\underline{c}+v,\underline{c}),2)-\max(-\underline{c}+v,\underline{c})+1)} \rfloor$, which is subsumed by the BAE $c_1 = 2 - ([\underline{c} = 1] + [(v - \underline{c}) \leq 1])$.

3. Quite often, using BAE allows one simplifying formulae with min and max, as illustrated now. Let $v, c, c_{23}$ and $\overline{s}$ be the number of vertices, connected components, connected components with two or three vertices where the size of each strongly connected component is equal to one, and the size of the largest strongly connected component of a digraph: e.g. for the graph  we have $v = 7, c = 3, c_{23} = 2, \overline{s} = 2$. As discovered by the CP model of [1], when $c$ is minimal, we have $c_{23} = (v.\overline{s} \leq 3\,?\,\min(v-1,1) : 0)$, which can be replaced by the Boolean relation $c_{23} = [\overline{s} = 1 \wedge v \in [2,3]]$.

4. It may occur that a formula can provide an approximate bound with an error of at most 1 on a parameter in $\mathbb{Z}$. Then, one way to get a sharp bound, is to find a Boolean formula that precisely describes the bound discrepancy. For example, a non-sharp lower bound (with a deviation of at most 1) on the number of connected components $c$ of a digraph $\mathcal{G}$ wrt the number of vertices $v$ of $\mathcal{G}$, the size of the largest connected component $\overline{c}$ of $\mathcal{G}$, and the size of the smallest strongly connected component $\underline{s}$ of $\mathcal{G}$ is given by $c \geq \lceil \frac{v}{\overline{c}} \rceil$; but a sharp lower bound is given by $c \geq \lceil \frac{v}{\overline{c}} \rceil + [(\underline{s} < \mathrm{fmod}(v, \overline{c})) \wedge (2 \cdot \underline{s} > \overline{c})]$, where $\mathrm{fmod}(x, y)$ is defined by the conditional expression $(x \bmod y = 0\,?\,y : x \bmod y)$. This case is explained more in detail in the next section.

5. Certain bounds can be expressed by a conditional formula, where the 'then' and the 'else' parts are simple expressions that are easy to identify, but the condition is a Boolean arithmetic expression that must be acquired. For example, consider a forest $\mathcal{F}$ of trees, where $v$, $f$, $\underline{f}$ and $\overline{f}$ denote respectively the total number of vertices in the trees of $\mathcal{F}$, the total number of leaves in the trees of $\mathcal{F}$, and the minimum and maximum number of leaves in the different trees of $\mathcal{F}$. A sharp upper bound of the maximum number of leaves $\overline{f}$ of a forest $\mathcal{F}$ is given by the conditional expression $\Big(f = \underline{f} \vee v = f \;?\; \underline{f} \;:\; f - \underline{f}\Big)$, where $f = \underline{f} \vee v = f$ is a Boolean arithmetic expression.

6. Some bounds can be expressed as case formulae, where the conditions correspond to Boolean arithmetic expressions that must be acquired, and the terms that provide values for each branch are integer constants that must also be identified. For example, a sharp lower bound on the number of strongly connected components $s$ of a digraph wrt the size $\underline{c}$ of its smallest connected component and the size $\underline{s}$ of its smallest strongly connected component is given by the following case formula:

$$s \geq \begin{cases} 3 & \texttt{if } \lfloor \underline{c}/\underline{s} \rfloor = 1 \wedge (\underline{c} - \underline{s}) \geq 1 \\ 1 & \texttt{else if } \underline{c} = \underline{s} \\ 2 & \texttt{otherwise} \end{cases} \tag{2.1}$$

The following points are specific to our equation discovery context [145]:

— As our samples are error-free, we need to acquire formulae that *correctly represent all the samples* we have.

— As our samples correspond to instances of combinatorial objects reaching a sharp bound, this is why we search for equations rather than for inequalities.

— We keep the original columns of the tables, as using one-hot encoding considerably increases the number of columns and affects the interpretability [146].

The updated version of the system with BAE as the new learning bias is presented in the paper [15].

## 2.3 The relevance of formula synthesis for learning sharp bounds

The second goal of this use case is to show how the ability to synthesise complex formulae from different learning biases is useful to improve the quality of the search for the maps of conjectures on sharp bounds. While the problem of synthesising formulae from data [130] is central to many areas such as programming by example (e.g. finding formulae in spreadsheets [70, 147, 148]), program verification (e.g. identifying loop invariants [149]), and conjecture generation (e.g. proposing bounds for combinatorial objects [1, 130, 150]), acquisition techniques are limited when the learning bias, i.e. "the set of assumptions that the learner uses to predict outputs of given inputs" [11], is vast. In this thesis, these assumptions correspond to the type of formulae we acquire.

Besides the recent work of S.-M. Udrescu *et al.* [151] which applies to continuous functions, most approaches for acquiring discrete functions rely on a grammar to define a domain-specific learning bias. They use a generate-and-test method to produce candidate formulae of increasing complexity. Several improvements were made to limit the combinatorial explosion of candidate formulae. They include the use of probabilistic grammar or statistical methods [152] to focus on more likely candidate formulae first, to generate partially instantiated formulae whose coefficients are determined by a CP or a MIP model [1, 153], or to apply metaheuristics [154]. However, their main weakness is twofold: first, they usually deal with formulae from a *restricted domain-specific learning bias*; second, they try to *directly acquire a formula* that mentions all the relevant input parameters at once.

We generally do not know how to effectively combine learning biases for acquiring formulae, so a system that knows how to solve problems with the learning bias (A) and another system that knows how to solve problems with the learning bias (B) can be combined to handle not only problems with the learning bias (A) or (B), but can also acquire formulae that combine both learning biases in a nested manner. The question, then, is *how to find decomposition methods that facilitate the discovery and combination of multiple learning biases.*

The method we propose partly answers this question through the following observation. Although many formulae are complex, i.e. they involve various operators in different subterms of a same formula, some parameters only appear in a *few sub-terms*, and some sub-terms have a *very specific form.* We show that by analysing a minimal functional dependency (MFD) of a table, while relying on the input columns and the output column of the table, it is sometimes possible to identify different sub-terms of a formula to be learnt and the operators that connect its sub-terms. We carry out this analysis by using Constraint Programming (CP) to solve certain sub-problems that allows us to decompose the formula we are looking for, into its sub-terms, without knowing yet the formula to be found.

### 2.3.1 Context and Motivation

The system [1] used three different learning biases (*i*)–(*iii*) (the detailed description of each learning bias is provided in Section 4.1.2). Despite using these biases, the system [1] missed many conjectures. Rather than extending these biases further or introducing new biases, we propose to combine these biases to catch complex formulae. To avoid a combinatorial explosion, this is not done by assembling a merged grammar associated

with different biases, but rather by *devising some recursive decomposition techniques that identify the sub-terms of a formula and how these sub-terms are connected by arithmetic operators*. The base case of such recursion belongs to biases (*i*)–(*iii*). The next section provides a first insight on how this is achieved.

### 2.3.2 Running Examples and Intuition of the Decomposition Technique

Conj. (2.2)–(2.5) illustrate sharp lower bounds found by the decomposition method that could not be found before, as they were outside the scope of biases (*i*)–(*iii*). They are used as running examples and were proved in Appendix A to stress the fact that the decomposition method can find non-obvious conjectures that turn out to be true. In the first phase, the decomposition method identifies an incomplete formula, i.e. a formula for which some terms are still unknown and will be determined later in the 2nd phase by applying the decomposition method recursively. In Conj. (2.2)–(2.5), the right-hand side of each inequality matches terms, highlighted by a brace, connected by one or two arithmetic operators (or a conditional), where:

— Terms with no grey background refer to expressions matching biases (*i*)–(*iii*), that are found in the 1st phase.

— Terms with a grey background are found in the 2nd phase when the decomposition is applied recursively.

**Conjecture 1.** *Conj. (2.2) provides a sharp lower bound on the number of connected components $c$ of a digraph where every vertex is adjacent to at least an arc wrt its number of vertices $v$, the maximum number of vertices $\overline{c}$ inside a connected component, and the smallest number of vertices $\underline{s}$ in a strongly connected component (scc). The right-hand side of Inequality (2.2) consists of the terms (1.1) and (1.2), resp. referring to biases (i) and (iii), and linked by a sum.*

$$c \geq \underbrace{\lceil v/\overline{c} \rceil}_{\text{(1.1) binary function}} + \underbrace{[\neg((2 \cdot \underline{s} \leq \overline{c}) \vee (\underline{s} \geq (v \bmod \overline{c} = 0 \ ? \ \overline{c} \ : \ v \bmod \overline{c}))]}_{\text{(1.2) Boolean term}} \tag{2.2}$$

In Phase one, the decomposition method finds a formula of the form $g_{1,1}(v, \overline{c}) + g_{1,2}(v, \overline{c}, \underline{s})$, where $g_{1,1}$ has only 2 parameters, and where the codomain of $g_{1,2}$ is the set $\{0, 1\}$; in the 2nd phase, the method finds the functions $g_{1,1}$ and $g_{1,2}$.

**Conjecture 2.** *Conj. (2.3) gives a sharp lower bound $c$ of a digraph wrt its number of scc $s$, and the $\overline{c}$ and $\underline{s}$ characteristics introduced in Ex. 1. The term (2.1) is the isolated input parameter $s$ and the term (2.2), i.e. $\lfloor \overline{c}/\underline{s} \rfloor$, refers to bias (i). These terms are connected by an integer division rounded up.*

$$c \geq \left\lceil \underbrace{s}_{\text{(2.1) unary function}} \; / \; \underbrace{\lfloor \overline{c}/\underline{s} \rfloor}_{\text{(2.2) binary function}} \right\rceil \tag{2.3}$$

In Phase one, the method finds the formula $\left\lceil \frac{g_{2,1}(s)}{g_{2,2}(\overline{c},\underline{s})} \right\rceil$, with $g_{2,1}(s) = s$, and where $g_{2,2}$ has only 2 parameters; in the 2nd phase, the method finds the function $g_{2,2}$ itself.

**Conjecture 3.** *Conj. (2.4) depicts a sharp lower bound on the maximum number of vertices $\overline{s}$ inside an scc of a digraph wrt the $s$ and $\underline{s}$ characteristics previously introduced. Within Conj. (2.4), the term (3.1) is a conditional expression, i.e. bias (ii), the term (3.2) refers to a unary function, and the term (3.3), i.e. $[s = 1]$, corresponds to a Boolean expression, i.e. bias (iii). These terms are connected by the division rounded up and the sum operators.*

$$\overline{s} \geq \left\lceil \underbrace{(v = \underline{s} \; ? \; v : v - \underline{s})}_{\substack{\text{(3.1) binary function as} \\ \text{a conditional}}} \; / \; ( \; \underbrace{s - 1}_{\substack{\text{(3.2) unary} \\ \text{function}}} \; + \; \underbrace{[s = 1]}_{\substack{\text{(3.3) Boolean} \\ \text{term}}} \; ) \right\rceil \tag{2.4}$$

In Phase 1, the decomposition method finds a formula of the form $\left\lceil \frac{g_{3,1}(v,\underline{s})}{g_{3,2}(s)+g_{3,3}(s)} \right\rceil$, with $g_{3,2}(s) = s - 1$, and where $g_{3,1}$ has only 2 parameters, and where the codomain of $g_{3,3}$ is the set $\{0, 1\}$; in the 2nd phase, the method finds $g_{3,1}$ and $g_{3,3}$.

**Conjecture 4.** *Conj. (2.5) depicts a sharp lower bound on the maximum number of vertices $\overline{c}$ inside a connected component of a digraph wrt the $v$, $c$, $\underline{c}$ and $\overline{s}$ characteristics previously introduced. The right-hand side of Conj. (2.5) is a complex conditional expression outside the scope of bias (ii), as its 'else' part is too complicated. It consists of three parts:*

- *A simple condition $v = c \cdot \underline{c}$ denoted by (4.1);*
- *A 'then' part, $\underline{c}$, depicted by (4.2);*
- *An 'else' part, $\max\left(\overline{s}, \left\lceil \frac{v-\underline{c}}{c-1} \right\rceil\right)$, referring to a complex term labelled by (4.3).*

$$\overline{c} \geq \Bigg( \underbrace{v = c \cdot \underline{c}}_{\text{(4.1) condition}} \quad ? \quad \underbrace{\underline{c}}_{\substack{\text{(4.2) 'then'} \\ \text{part}}} \quad : \quad \underbrace{\max\left(\overline{s}, \left\lceil \frac{v - \underline{c}}{c - 1} \right\rceil\right)}_{\text{(4.3) 'else' part}} \Bigg) \tag{2.5}$$

The method first finds $(v = g_{4,1}(c, \underline{c}) \ ? \ g_{4,2}(\underline{c}) : g_{4,3}(v, c, \underline{c}, \overline{s}))$ with $g_{4,1}(c, \underline{c}) = c \cdot \underline{c}$, $g_{4,2}(\underline{c}) = \underline{c}$ ; then it finds function $g_{4,3}$ using the method in a recursive way:

— It finds $g_{4,3}(v, c, \underline{c}, \overline{s}) = \max(g_{4,3,1}(\overline{s}), g_{4,3,2}(v, c, \underline{c}))$, with $g_{4,3,1}(\overline{s}) = \overline{s}$, and where $g_{4,3,2}$ has 3 parameters;

— It then finds function $g_{4,3,2}$ using again the decomposition method in a recursive way:

  — It first finds $g_{4,3,2}(v, c, \underline{c}) = \left\lceil \frac{g_{4,3,2,1}(v, \underline{c})}{g_{4,3,2,2}(c)} \right\rceil$ with $g_{4,3,2,2}(c) = c - 1$;

  — It then finds function $g_{4,3,2,1}(v, \underline{c})$ directly as $v - \underline{c}$, a polynomial, i.e. bias $(i)$.

We saw four conjectures acquired by our decomposition method. They cover all the types of decompositions we found. Each function introduced in Phase 1 when searching for an incomplete formula is simpler than the function initially looked for: $(a)$ it has *fewer input parameters*, e.g. in Conj. 1, $g_{1,1}(v, \overline{c}) = \lceil v/\overline{c} \rceil$ does not mention the $\underline{s}$ parameter, or $(b)$ its codomain is *restricted to two values*, e.g. in Conj. 1, the codomain of $g_{1,2}$ is the set $\{0, 1\}$, or $(c)$ it *only holds if a given condition is met*, e.g. the 'else' part in Conj. (4) only holds if $v \neq c \cdot \underline{c}$.

Conjectures (1)–(4) were proven by Jovial Cheukam-Ngouonou [14].

Section 7 describes the acquisition process of four type of decompositions we introduced in this section.

Chapter 3

# Second use case: Acquiring short-term scheduling models

In this chapter we will look into the second use case – the acquisition of a constraint model from a single example of a short-term production schedule. We assume that the schedule is correct, i.e. error-free. First, we will provide the context of the use case and provide the scope of the proposed solution. Then, we will describe the inputs that are needed for the model acquisition. Next, since in practice the inputs have diverse formats we discuss how to unify input data with a preprocessing step. Lastly, we describe the intended output of model acquisition, i.e. types of constraints we want to acquire. During the section a running example will be presented and described in depth.

## 3.1 Context and scope

### 3.1.1 Context

Advanced short-term scheduling systems were introduced several decades ago to generate optimised production schedules automatically [155]. These technologies reduce costs and speed up the creation of production schedules. However, most companies still rely on spreadsheets or simple rules to manage their plants [156]. Humans cannot analyse the large number of scheduling alternatives, so the resulting production schedule is suboptimal. Consultancy costs are a major obstacle to the adoption of advanced scheduling systems. Implementing scheduling software requires experts to build the model, and such a software system often requires high costs to customise the software. Furthermore, expert model building is not in line with the Industry 4.0 context, where the shop floor is frequently reconfigured. This use case studies the acquisition of short-term scheduling models from data to generate these models automatically.

While Industry 4.0 heavily uses various AI techniques and OR models, there is little

work which takes advantage of AI to acquire short-term scheduling models [60] which can be directly run on standard industrial OR tools such as CPLEX [157] from IBM or OR-Tools [158] from Google to produce feasible schedules. Indeed, the use of AI techniques for Industry 4.0 was up to now mostly restricted to prediction and classification problems, e.g. [159, 160].

### 3.1.2 Scope

The scope of this use case is the acquisition of OR models, and the domain we consider is STPS in an industrial context: i.e. we do not limit ourselves to pure scheduling problems like flow shop problems, but we also consider industrial constraints like calendar or shift constraints. Our focus is to acquire short-term scheduling models, where a model identifies the variables (e.g. task starts, assigned resources) of the problem, and the constraints (e.g. precedence, disjunctive) between these variables. The acquired models are technology-agnostic (CP, MIP), expressed in the MiniZinc modelling language which can be executed on a variety of solvers. Note that our focus is not on creating models optimised for a specific technology, as MiniZinc takes this aspect into account by transparently adapting a model to a particular technology, for example by linearising certain constraints when running the model on a MIP solver [161].

— On the one hand, our model acquisition tool assumes that tasks and resources of a schedule are described using tables and that the data provided contains no errors. This latter assumption is made as learning from a single positive example containing errors is too challenging.

— On the other hand, it does not assume any specific format for these tables, i.e. the order of the columns in a table does not matter, the columns of these tables are not necessarily labelled, and some columns may not be relevant to the acquisition process. It is up to the acquisition system to interpret the meaning of the columns to extract the different constraints of a scheduling model. However, it is required to indicate which columns correspond to the input data of the model, and which columns correspond to output data: e.g. in some scheduling problems, the machine on which a task is pre-assigned is known (i.e. the machine corresponds to input data), and in some other scheduling problems, the solver would also have to assign each task to a machine (i.e. the machine corresponds to output data). This clean separation between the input data used by a model and the model itself allows one

to reuse an acquired model on some other input data set *without modifying the acquired model*.

Learning from a single example is not as radical as it might seem. In fact, in a real-life context, a schedule can involve anywhere from a hundred to a few thousand tasks, so even a single example can provide enough data.

## 3.2 Model acquisition input

The model acquisition tool uses a single positive instance of a short-term schedule as the basis for acquiring a model. In this work, we refer to *scheduling instance* as the input data of a scheduling problem with its associated solution, whereas in classical scheduling terminology a scheduling instance refers only to the input data. The scheduling instance typically consists of resources and task attributes that describe an instance of a feasible production schedule. A scheduling instance can (*i*) be generated by a digital twin [162], (*ii*) derived from historical data or data streams collected by sensors on the shop floor, or (*iii*) created manually.

**Example 3.2.1.** *As a running example, consider tables Table 3.1 and Table 3.2 which respectively describe the set of machines and the set of tasks of an instance of a schedule used to acquire a model. Each column of such tables corresponds to a machine or to a task attribute.*

Table 3.1 – An example of 'machines table' giving for each machine its identifier, its speed, and its unavailability

| Machine_id | Speed | Unavailability |
|---|---|---|
| MB_10 | 2 | {30..39,50..59} |
| MB_2 | 3 | {15..19} |
| MB_3 | 2 | {40..49,70..79} |
| MB_5 | 4 | {40..49,70..79} |

The type of configuration used in Example 3.2.1 is quite common for companies already using scheduling software such as an Excel spreadsheet, a basic ERP or an ontology-based digital twin.

Table 3.2 – An example of 'tasks table' providing for each task its attributes

| Task_id | Start | Quantity | Duration | Machine | Machines_set | Successor |
|---|---|---|---|---|---|---|
| AB10001 | 0 | 10 | 20 | MB_10 | {MB_5, MB_10} | AB10002 |
| AB10002 | 20 | 10 | 30 | MB_2 | {MB_2, MB_3} | AB10003 |
| AB10003 | 50 | 10 | 20 | MB_3 | {MB_3, MB_5} | AB10004 |
| AB10004 | 70 | 10 | 20 | MB_10 | {MB_10} | – |
| AB10005 | 0 | 4 | 12 | MB_2 | {MB_2, MB_5} | AB10006 |
| AB10006 | 20 | 4 | 8 | MB_10 | {MB_2, MB_10} | AB10007 |
| AB10007 | 28 | 4 | 8 | MB_3 | {MB_3, MB_5} | – |
| AB10008 | 0 | 9 | 36 | MB_5 | {MB_5, MB_10} | AB10009 |
| AB10009 | 50 | 9 | 27 | MB_2 | {MB_2} | – |

## 3.3 Preprocessing of the inputs for the model acquisition

The tables providing a scheduling instance are preprocessed to:

1. Select the relevant columns, namely the columns required to build a model. This is because the information regarding a scheduling instance is usually only a small fraction of the data available in the databases and spreadsheets of a company.
2. Mark columns corresponding to an input of the model as well as columns refereeing to model output, i.e. columns whose values will be determined by the model.
3. Convert data to a uniform format supported by the model acquisition part.

While Issues 1 and 2 are handled by a human who annotates every column of the tables, the automatic preprocessing phase described below solves Issue 3. The model acquisition tool works with tables whose cells correspond to an integer value, a list of integers, or a list of integer intervals. Therefore, cells corresponding to strings are automatically converted to an appropriate integer:

— As many identifiers (e.g. task identifiers, product identifiers, or resource identifiers) are alphanumeric, the preprocessing tool converts them to natural numbers.

— Sometimes an undefined value is represented by an alphanumeric string, and we convert them to an unused integer value. Similarly, an empty cell is also converted to an unused integer value.

— Cells may refer to task identifiers which are not part of the input data as the corresponding tables may only focus on a set of tasks associated with a restricted time horizon. Such references to missing task identifiers are converted to some unused integer value.

— Cells containing date and time information are converted into natural numbers. The earliest instant occurring in a table is set to 0, while the other time points are presented as a shift from the earliest instant.

**Example 3.3.1.** *After the preprocessing phase, Table 3.1 and Table 3.2 are respectively transformed into Table 3.3 and Table 3.4, where blue cells correspond to modified values. Object identifiers, i.e. machine and task identifiers, were changed to distinct integer values as shown by the first columns of Table 3.3 and Table 3.4; references to machine and task identifiers were updated accordingly as illustrated by the last three columns of Table 3.4. Within the 'Successor' column of Table 3.4, missing values were replaced by the otherwise unused integer value* $-1$*. In the rest of the thesis, to continue with the same running example, we will assume that all the columns in Table 3.3 are marked as input columns, and that only the 'Quantity', 'Machines_set' and 'Successor' columns in Table 3.4 are input columns. In other words, given the previous input columns, only columns 'Start', 'Duration' and 'Machine' in Table 3.4 need to be determined by the acquired OR model.*

Table 3.3 – Machines table obtained after preprocessing Table 3.1

| Machine_id | Speed | Unavailability |
|---|---|---|
| 10 | 2 | {30..39,50..59} |
| 2 | 3 | {15..19} |
| 3 | 2 | {40..49,70..79} |
| 5 | 4 | {40..49,70..79} |

Table 3.4 – Tasks table obtained after preprocessing Table 3.2

| Task_id | Start | Quantity | Duration | Machine | Machines_set | Successor |
|---|---|---|---|---|---|---|
| 10001 | 0 | 10 | 20 | 10 | {5, 10} | 10002 |
| 10002 | 20 | 10 | 30 | 2 | {2, 3} | 10003 |
| 10003 | 50 | 10 | 20 | 3 | {3, 5} | 10004 |
| 10004 | 70 | 10 | 20 | 10 | {10} | −1 |
| 10005 | 0 | 4 | 12 | 2 | {2, 5} | 10006 |
| 10006 | 20 | 4 | 8 | 10 | {2, 10} | 10007 |
| 10007 | 28 | 4 | 8 | 3 | {3, 5} | −1 |
| 10008 | 0 | 9 | 36 | 5 | {5, 10} | 10009 |
| 10009 | 50 | 9 | 27 | 2 | {2} | −1 |

## 3.4 Output of the model acquisition tool

The MiniZinc modelling language [10] provides a large library of predefined constraints where many of these constraints represent high-level modelling abstractions [3], which are

understood by most solvers. We acquire a MiniZinc constraint model, where a model consists of a conjunction of constraints of the following types: functional constraints, temporal constraints, resource constraints, calendar constraints, and shift constraints. The rest of this section reviews the type of constraints that the tool can acquire, whereas sections 4.1.2, 8.1, 8.2 and 8.3 describe the acquisition process.

### 3.4.1 Functional constraints

A functional constraint is a constraint of the form $y = f(x_1, x_2, \ldots, x_n)$, where $x_1$, $x_2$, $\ldots$, $x_n$, $y$ are columns of some table $T$, and $f$ is a function which determines the value of column $y$ from the values of columns $x_1$, $x_2$, $\ldots$, $x_n$ for every row of table $T$. Function $f$ corresponds to a linear expression or to a non-linear expression involving usual arithmetic operators such as 'min', 'max', '$\times$'. The acquisition process of functional constraints will be mentioned in Section 4.1.2.

**Example 3.4.1.** *From Tables 3.1 and 3.2 we observe the functional constraint (3.1) which computes the duration of a task i from the machine that processes the task and from the corresponding machine speed which denotes how fast a machine can process one item, i.e. the processing time per item.*

$$Duration[i] = Speed[Machine[i]] \times Quantity[i] \tag{3.1}$$

### 3.4.2 Temporal constraints

Given a sequence $S$ of tasks, a temporal constraint on the sequence $S$ enforces the same temporal relation between all pairs of consecutive tasks of the sequence $S$. A temporal relation between two consecutive tasks is a minimum, maximum, exact distance constraint between (*i*) the start or the end of the first task and (*ii*) the start or the end of the second task.

Identifying a temporal constraint requires (*i*) to recognise the associated sequences of tasks, (*ii*) to find which attributes of the task are involved (i.e. start, end, duration), (*iii*) to define whether it is a minimum, maximum or exact distance constraint, and finally (*iv*) to compute the constant corresponding to the distance between consecutive tasks of a sequence $S$. The acquisition process of temporal constraints will be explained in Section 8.1.

**Example 3.4.2.** *Within Table 3.2, using the 'Successor' column, we observe the following three sequences of tasks:*

- $S_1$ = *AB10001, AB10002, AB10003, AB10004,*
- $S_2$ = *AB10005, AB10006, AB10007, and*
- $S_3$ = *AB10008, AB10009.*

*These sequences correspond to precedence constraints between the end of task $i$ and the start of the next task with a minimum distance of* 0*, namely:*

$$Start[i] + Duration[i] + 0 \leq Start[Successor[i]] \tag{3.2}$$

### 3.4.3 Resource constraints

Resource constraints restrict the simultaneous resource utilisation by the tasks. We support the acquisition of the following two types of resource constraints, namely, DISJUNCTIVE [4, 5] and DIFFN [10, 163] constraints. Given a set of tasks $T$, where each task is defined by its start and end, the DISJUNCTIVE($T$) constraint prevents all pairs of tasks in $T$ from overlapping in time.

Given a set of rectangles $R$, where each rectangle is defined by the coordinates of its bottom left corner, its length, and its width, the DIFFN($R$) constraint requires that all pairs of rectangles in $R$ do not overlap, and that all rectangle sides are either perpendicular or parallel to the placement axes $x$ and $y$. In the context of scheduling problems, the $x$ and $y$ axes represent the time and the resource axes. Each rectangle $r$ of $R$ corresponds to a task, where the $x$-coordinate of $r$, the $y$-coordinate of $r$, the length of $r$, and the width of $r$ respectively refer to the start of the task, the resource to which the task is assigned, the duration of the task, and the constant 1, i.e. 1 as the task is assigned to one single resource.

Note that the DISJUNCTIVE constraint is a special case of the DIFFN constraint where the set of tasks to process on each resource is known. The DIFFN constraint applies when tasks are not pre-assigned to specific resources. In this latter case, the WITHIN constraint defines the set of resources that can process a task (see Equation 3.3). The acquisition process of resource constraints is described in Section 8.2.

**Example 3.4.3.** *First, observe that in Table 3.2, the 'Machine' attribute of the $i$-th task can only be selected from a corresponding 'Machines_set' attribute, i.e. we have the*

*following* WITHIN($Machine[i], Machine\_set[i]$) *constraint corresponding to:*

$$Machine[i] \in Machine\_set[i] \tag{3.3}$$

*Second, as tasks which are assigned to a same machine do not overlap, we have a* DIFFN *constraint where the i-th task corresponds to a rectangle whose coordinates of its bottom left corner, length and width respectively are* $(Start[i], Machine[i])$*,* $Duration[i]$ *and* 1*.*

### 3.4.4 Calendar constraints

When dealing with short-term scheduling, the execution of tasks is often restricted by calendar constraints [164, 165]. Depending on the resource it uses, a task can only run during certain periods when the resource is available. Each resource can have its own calendar, defined as a set of time intervals during which the resource is available or not.

Given a set of tasks $T$, where each task is defined by its start, its end and the calendar identifier it uses, and a set of calendars $C$, where each calendar is specified by a unique identifier and a sorted list of disjoint intervals, the CALENDAR($kind, T, C$) constraint holds iff

— $kind = 1$: each *task start* is in a calendar interval,

— $kind = 2$: each *task end minus one* is in a calendar interval,

— $kind = 3$: each *task* is included in a calendar interval,

— $kind = 4$: each *task start* does not belong to any calendar interval,

— $kind = 5$: each *task end minus one* does not belong to any calendar interval,

— $kind = 6$: each *task* does not overlap any calendar interval.

Note that we use the term ‘task end minus one’ rather than ‘task end’, because we want to refer to the last instant when the task is running.

### 3.4.5 Shift constraints

The working day in a workshop is often divided into working periods called shifts, with a team of workers assigned to each shift. The number of shifts usually varies from three to two, depending on whether it is a working day or a weekend. Certain tasks that need to be monitored by the same team must start and finish within the same shift.

Given a set of tasks $T$, where each task is defined by its start, its end and the calendar identifier it uses, and a set of calendars $C$, where each calendar is specified by a unique identifier and a sorted list of disjoint intervals, the SHIFT$(T, C)$ constraint holds iff

— there is no gap between two consecutive calendar intervals,

— each task is included in a calendar interval.

The SHIFT constraint can be seen as a special case of the CALENDAR constraint when $kind = 3$ and there is no gap between successive calendar intervals.

**Example 3.4.4.** *First, observe that in Table 3.2, the 'Machine' attribute of the i-th task can only be selected from a corresponding 'Machines_set' attribute, i.e. we have the following* WITHIN$(Machine[i], Machine_set[i])$ *constraint corresponding to the equation 3.3.*

*Second, as tasks which are assigned to a same machine do not overlap, we have a* DIFFN *constraint where the i-th task corresponds to a rectangle whose coordinates of its bottom left corner, length and width respectively are* $(Start[i], Machine[i])$*,* $Duration[i]$ *and* 1*.*

*Third, since each task i does not intersect any calendar interval of* $Unavailability[Machine[i]]$ *associated with the machine* $Machine[i]$ *to which task i is assigned, we have a* CALENDAR *constraint with* $kind = 6$*, where the i-th task corresponds to a task starting at* $Start[i]$*, ending at* $Start[i] + Duration[i]$*, and using the calendar of the machine* $Machine[i]$ *to which it is assigned.*

### 3.4.6 Wrapping up the constraints of the running example

This section first illustrates the solution associated with the example schedule given in Table 3.3 and Table 3.4. It then provides pseudocode for the model, which is generated from the contents of Table 3.3 and Table 3.4 by the model acquisition tool presented in Section 9.3.

Figure 3.1 illustrates the solution associated with the sample schedule given in Tables 3.3 and 3.4. Each task corresponds to a rectangle labelled with its identifier, whose coordinates in the bottom left corner are the start of the task and the machine to which the task is assigned. Tasks coloured in the same way are part of a chain of precedence constraints, for example task 10005 finishes before task 10006 starts, while task 10006 ends before task 10007. Hatched areas represent intervals between consecutive tasks al-

located to the same machine and thick red lines correspond to the periods of machine unavailability associated with the machine calendars.

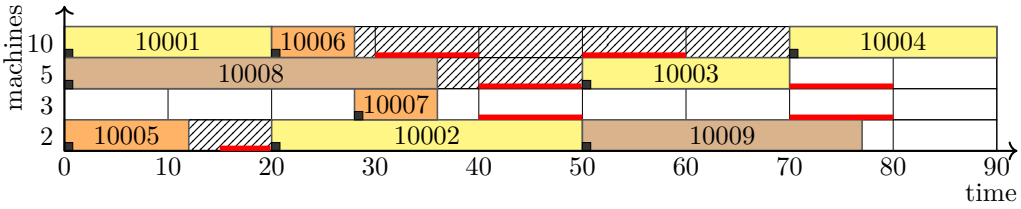



Figure 3.1 – Schedule corresponding to the running example

To be directly transferable to other sets of input data, a generated model always consists of two parts: the first part corresponding to the input data declaration, and the second part consisting of parameterised constraints. In the model shown below, the arrays `Task_id`, `Successor`, and `Machine_set` refer to concrete elements in the other arrays. All input data arrays are taken directly from the input schedule. Note that the start and duration of a task are not part of the input data, as they will be obtained by running the model.

The arrays `Calendar_id`, `Res`, `Low` and `Len` introduce the periods of unavailability of each machine; they correspond respectively to an unavailability period identifier, to the machine to which it corresponds, to when it starts and to how long it lasts. This information is generated from the 'Machine_id' and 'Unavailability' columns in Table 3.3. Since we have a CALENDAR constraint which requires that the start and end of the tasks do not overlap any calendar interval (i.e. the *kind* parameter is equal to 6), all downtimes were included in the DIFFN constraint in the form of dummy fixed rectangles. This ensures that each task will not overlap any unavailability period of the machine assigned to the task.

```
                                                INPUT DATA OF THE MODEL
Task_id      = [ 1, 2, 3, 4,5,6, 7,8, 9]
Successor    = [ 2, 3, 4,-1,6,7,-1,9,-1]
Quantity     = [10,10,10,10,4,4, 4,9, 9]
Machine_set  = [{1,4},{2,3},{3,4},{1},{2,4},{1,2},{3,4},
                {1,4},{2}]
Speed        = [2,3,2,4]
Calendar_id  = [ 1, 2, 3, 4, 5, 6, 7]
Res          = [ 1, 2, 2, 3, 3, 4, 4]
Low          = [15,40,70,40,70,30,50]
Len          = [ 5,10,10,10,10,10,10]
```

PARAMETERISED CONSTRAINTS OF THE MODEL

$\forall i \in \texttt{Task\_id}:$

$\texttt{Duration}[i] = \texttt{Speed}[\texttt{Machine}[i]] \times \texttt{Quantity}[i]$

$\forall i \in \texttt{Task\_id},\ \texttt{Successor}[i] \neq -1:$

$\texttt{Start}[i] + \texttt{Duration}[i] + 0 \leq \texttt{Start}[\texttt{Successor}[i]]$

$\forall i \in \texttt{Task\_id}:$

$\texttt{Machine}[i] \in \texttt{Machine\_set}[i]$

$\textsc{diffn}([\texttt{Start}[i], \texttt{Machine}[i], \texttt{Duration}[i], 1] \mid \forall i \in \texttt{Task\_id},$

$[\ \texttt{Low}[j],\ \texttt{Res}[j],\ \texttt{Len}[j], 1] \mid \forall j \in \texttt{Calendar\_id})$

Part III

# Overview of the model acquisition tool

Chapter 4

# The model acquisition tool

To achieve objectives stated in Chapters 2 and 3 a model acquisition tool was developed together with Nicolas Beldiceanu and Jovial Cheukam-Ngouonou. The tool is written in SICStus Prolog [166]. The tool uses extensively the Prolog's ability to backtrack to effectively generate and enumerate through the data tables, characteristics, the candidate formulae, etc. The tool also use the finite domain constraint solver [167] to acquire the different types of constraints from the input data.

For the first use case (see Chapter 2) the tool outputs the list of acquired conjectures on sharp bounds of the given characteristics. For the second use case (see Chapter 3) the tool outputs a model MiniZinc file and a data MiniZinc file which then can later be used to generate new valid schedules.

In this chapter we will describe each component of the tool to provide a global view of what was done prior to and during this thesis.

## 4.1 Description of the components of the acquisition tool

The tool consists of several components that are launched separately from each other. Each component designed to solve a particular problem. There are five main component examined in this section:

1. *Metadata generation component*, which analyses each tables to extract data which is used by other components. It also acquires temporal constraints (see sections 3.4.2 and 8.1).
2. *Functional constraint acquisition component*, which acquires conjectures for combinatorial objects (see Chapter 2) and functional constraints for STPS (see Section 3.4.1).

3. *Schedule constraint acquisition component*, which acquires resource and calendar constraits (see Sections 3.4.3 and 3.4.4 and Chapter 8).
4. *Table merging component*, which merges two table into one.
5. *Model conversion component*, which converts acquired constraints into an output MiniZinc files.

### 4.1.1 Metadata generation

The model acquisition process must access certain aggregated information several times to guide and focus on the acquisition of various types of constraints. For each merged table, this aggregated data is calculated once and for all and stored in an associated metadata file. There are three types of metadata that we describe in the next paragraphs, namely, (*i*) information valid for an entire table, (*ii*) information specific to each column in a table, and (*iii*) information describing a relationship between a column in a table and other columns in the same table.

#### 4.1.1.1 Metadata valid for an entire table

We have the following information.

— Name and number of columns of the table, and ranked candidate primary keys.

— All relevant information for computing temporal constraints is computed during the metadata generation phase. How this is done is described in Section 8.1.

#### 4.1.1.2 Metadata specific to each column in a table

For each column we compute the following information.

— Flags noting whether or not the column could be used as an input and/or output of a functional constraint. These flags are initially provided by the user in the table file and then stored in the metadata in a convenient format.

— The potential scheduling attributes to which the column name can correspond, i.e. *start*, *duration*, *end*. To do this, we normalise the names of the columns and look for the presence of certain sub-words that indicate that a column might correspond to a particular scheduling attribute.

— The data type of the cells of the column, namely 0–1 integer, integer or set of integers.

— The number of distinct values and the list of distinct values when the number of distinct values is small.

— The minimum value, the maximum value, and the sum of all entries in each column when the column data type is an integer.

#### 4.1.1.3 Information describing a relationship between a column and other columns

We gather the following relationships.

— The indication that a column is the same as another column in the table. When there are two or more identical columns, we consider only the first one and discard the others.

— The fact that for each entry in a table, the corresponding value in a given column is always smaller (or always larger) than the corresponding value in another given column. This information is used, for example, when acquiring functional constraints, see Section 4.1.2.

— The fact that for each entry in a table, the corresponding value in a given column is always included in the corresponding set of values in another given column. This indicates that a column is a part of a WITHIN constraint, see Section 8.2.

— To control the search for functional constraints looking for a formula expressing the value of a column $c$ as a function of other columns, we precompute for each column $c$ the list of its minimal functional dependencies [168], that we now define.

**Definition 2.** *Given a table $T$, a column $c$ of $T$, and a set of columns $\mathcal{C}$ of $T$ (with $c \notin \mathcal{C}$),* $\mathcal{C}$ functionally determines $c$ *if and only if each $\mathcal{C}$ value in $T$ is associated with one single $c$ value in $T$.*

**Definition 3.** *Given a table $T$, a* minimal functional dependency *for column $c$ is a set of columns $\mathcal{C}$ such that (*i*) $\mathcal{C}$ functionally determines $c$, and (*ii*) there is not subset $\mathcal{C}' \subset \mathcal{C}$ such that $\mathcal{C}'$ functionally determines $c$.*

**Example 4.1.1.** *Regarding the 'Duration' column, the merged table Table 4.1 contains 10 minimal functional dependencies that we compute and record:*

- {*'Task_id'*},
- {*'Start'*, *'Machine'*},
- {*'Start'*, *'Quantity'*},
- {*'Start'*, *'Successor'*},

- {*'Start', 'Speed'*},
- {*'Quantity', 'Machine'*},
- {*'Quantity', 'Machine_set'*},
- {*'Quantity', 'Successor'*},
- {*'Quantity', 'Speed'*},
- {*'Machine', 'Successor'*}.

— In some cases, focusing only on minimal functional dependencies is not enough to acquire a functional constraint. As a result, the model acquisition tool will also check all combinations of minimal functional dependencies with one or two additional columns. As the number of functional dependencies obtained can be too large (several hundreds), we rank them to select the best candidates. The detailed description of the ranking process is provided in Section 5.3.

### 4.1.2 Conjectures and functional constraints acquisition

The acquisition of *conjectures* and *functional constraints* learns equality constraints expressing the value of an output column relatively to other columns. For example, in the context of short-term scheduling models, such constraints may include calculating the duration of a task based on the size of the order and the speed of the machine to which the task is assigned.

In practice, there is no difference between the process of acquiring conjectures for combinatorial objects and functional constraints, as the same technique is used for both. For the sake of simplicity, the remainder of this section will refer to functional constraints only, unless otherwise specified.

The acquisition of functional constraints is organised as follows:

1. The model acquisition tool selects all output columns from the metadata.
2. For each output column:
   (a) Based on the list of ranked functional dependencies of the selected output column, we generate the list of candidate formulae. Each candidate formula includes ($a$) the list of input parameters of the functional dependency, ($b$) the type of formula, e.g. polynomial, arithmetic-Boolean expression, and ($c$) the parameters of the formula indicating its complexity, such as a communicative operator, a number of Boolean-arithmetic terms or a number of monomes, etc. The model acquisition tool ranks the candidate formulae from the simplest to the most complex.

   A candidate formula belongs to one of the following learning biases:

- (learning bias (*i*)) formulae that contain binary and unary functions within a polynomial function. The list of binary and unary functions include $+$, $-$, $\times$, $/$, min, max, mod and Boolean comparisons against constants (interpreted as 1 if a comparison is `true` and 0 otherwise). This learning bias was developed by Nicolas Beldiceanu and Jovial Cheukam-Ngouonou.

- (learning bias (*ii*)) a case formula of the format:

$$y = \begin{cases} f_1(x_1, x_2, \ldots, x_n) & \texttt{if } \mathcal{C}_1(x_1, x_2, \ldots, x_n) \\ f_2(x_1, x_2, \ldots, x_n) & \texttt{otherwise}, \end{cases},$$

  where:

  — $\mathcal{C}_1(x_1, x_2, \ldots, x_n)$ is a Boolean-arithmetic condition containing one of the following comparison operator $\leq$, $\geq$, $\in$, $\notin$, and that may include one or more arithmetic operators such as $+$, $-$, $\times$, $/$, min, max or mod;

  — $f_1(x_1, x_2, \ldots, x_n)$, $f_2(x_1, x_2, \ldots, x_n)$ are either an input column, a coefficient or a binary or a unary function that uses $+$, $-$, $\times$, $/$, min, max or mod.

  This learning bias was introduced by Jovial Cheukam-Ngouonou and Nicolas Beldiceanu.

- (learning bias (*iii*)) an arithmetic-Boolean expression (BAEx) $\mathcal{B}$ of the format $\mathcal{B} = g_{i=1}^{n} \mathcal{C}_i(x_1, \ldots, x_n)$, where:

  — $\mathcal{C}_i(x_1, x_2, \ldots, x_n)$ is an arithmetic-Boolean condition that contains one of the comparison operators $\leq$, $\geq$, $\in$, $\notin$ and can include one or more arithmetic operators such as $+$, $-$, $\times$, $/$, min, max or mod;

  — $n \geq 1$ is the number of arithmetic-Boolean conditions;

  — $g \in \{\wedge, \vee, =, \veebar, +\}$ is a single commutative logical operator or the sum operator.

  This learning bias was developed by the author of this thesis under the supervision of Nicolas Beldiceanu. The detailed description is presented in Chapter 6.

- (learning bias (*iv*)) a case formula of the form:

$$y = \begin{cases} c_1 \;\; if \;\; \mathcal{B}_1 \\ c_2 \;\; if \;\; \mathcal{B}_2 \wedge \neg\mathcal{B}_1 \\ c_3 \;\; if \;\; \mathcal{B}_3 \wedge \neg\mathcal{B}_2 \wedge \neg\mathcal{B}_1 \\ \dots \\ c_n \;\; otherwise, \end{cases},$$

where:

— $\mathcal{B}_j$ is an arithmetic-Boolean expression, $\mathcal{B}_j = g_{j=1}^{n}\mathcal{C}_j(x_1, x_2, \dots, x_n)$;

— $c_j$ is a label of a cluster $j$ corresponding to an integer value.

This learning bias was developed by the author of this thesis under the supervision of Nicolas Beldiceanu. The detailed description is presented in Section 6.5.

- four decomposition techniques (7.2), (7.3), (7.4), and (7.5), which combine learning biases (*i*)–(*iii*) together. This part was developed by the author of the thesis under the supervision of Nicolas Beldiceanu. The detailed description is presented in Chapter 7.

(b) The model acquisition tool enumerates through the list of candidate formulae:

i. It generates a constraint model corresponding to the current candidate formula [1].

ii. It attempts to solve the constraint model:

— if the model acquisition tool does not find a solution, it moves on to the next candidate formula,

— otherwise, if a solution was found, this solution is compared against previously found solutions for the output column. If the system considers the new solution to be simpler than any of the previously found solutions, or if there were no previously found solutions before, then the solution is recorded in a database. This allows the model acquisition tool to acquire increasingly simpler constraints.

Note that all the learning biases are checked in a particular order, one after the other. The following ideas were used to design this ordering. The first idea is to try simple formulae that can be checked quickly and are more likely to appear first. The second idea is to provide only those learning biases that can realistically be present in a use case.

*The ordering of the learning biases during the search for conjectures for combinatorial objects is the following*:

1. Boolean-arithmetic formulae: bias (*iii*) when the output column of the table 'tab' has only two values.
2. Simplest polynomial formulae: bias (*i*) with one monome.
3. Simple conditional formulae: bias (*ii*) and bias (*iv*) when $n = 2$.
4. Simple polynomial formulae: bias (*i*), two or three monomes.
5. The decompositions (7.2), (7.3), (7.4), and (7.5), in this order.
6. Complex polynomial formulae: bias (*i*), four to six monomes.
7. Complex case formulae: bias (*iv*) when $n \geq 3$.

*The order of learning biases during the search for functional constraints in STPS is*:

1. Boolean-arithmetic formulae: bias (*iii*) when the output column of the table 'tab' has only two values.
2. Simple polynomial formulae: bias (*i*) with one to three monomes.
3. Simple conditional formulae: bias (*ii*) and bias (*iv*) when $n = 2$.

### 4.1.3 Acquiring scheduling constraints

This module acquires resource and calendar constraints related to task scheduling. These constraints were described in sections 3.4.3 and 3.4.4.

The module was developed by the author of the thesis and Nicolas Beldiceanu. See Chapter 8 for the full description of the acquisition process.

### 4.1.4 Merging tables

Some scheduling constraints can only be obtained by examining two or more tables simultaneously. These tables will be related to each other as a parent table and a child table, and will be merged before acquiring any constraints. Merging tables removes the need to access to the entries of the child tables from the entries of the parent table as such entries will be directly available from the merged table. In addition, the constraint acquisition part is simpler when all constraints are acquired from a single table.

It should be noted that some input tables may have no explicitly stated primary key. In this case, there are two possibilities:

— if the user explicitly annotated that there are no primary keys, then the model acquisition system will not try to learn temporal constraints and will attempt to learn functional, resource and calendar constraints,

— otherwise, the model acquisition system will try to guess the best candidate for the primary key.

**Example 4.1.2.** *For instance, in Example 3.4.1, the functional constraint (3.1) expressing the duration of each task, besides using the 'Machine' and the 'Quantity' columns of the tasks table Table 3.4, also mentions the 'Speed' column of the machines table Table 3.3.*

To merge a child table and a parent table, we perform the following steps:

— We get the primary key of the parent table and the foreign key of the child table that connects the two tables.

— We create the columns of the merged table: the merged table includes all the columns of the child and parent tables, except for the primary key column of the parent table. The columns from the child table are listed first, before the columns of the parent tables.

**Example 4.1.3.** *The merged table created from the child table Table 3.4 and the parent table Table 3.3 has the following columns, where columns from the child (resp. parent) table are shown in blue (resp. cyan).*

| *Task_id* | *Start* | *Quantity* | *Duration* | *Machine* | *Machines_set* | *Successor* | *Speed* | *Unavailability* |
|---|---|---|---|---|---|---|---|---|

— For each row $c$ in the child table, we create a corresponding row in the merged table:

  — We identify the row $p$ in the parent table for which the primary key matches the foreign key value of the $c$-th row of the child table.

  — The merged table entry takes the $c$-th row of the child table, as well as the $p$-th row of the parent table except its primary key.

**Example 4.1.4.** *For example, for the first two rows of the child tasks table Table 3.4 the corresponding rows of the parent resource table Table 3.3 are rows* 1 *and* 3 *for which the machine speed is respectively* 2 *and* 3*. Consequently, the first two rows of the merged table are given below.*

| *Task_id* | *Start* | *Quantity* | *Duration* | *Machine* | *Machines_set* | *Successor* | *Speed* | *Unavailability* |
|---|---|---|---|---|---|---|---|---|
| *10001* | *0* | *10* | *20* | *10* | $\{5, 10\}$ | *10002* | *2* | *{30..39,50..59}* |
| *10002* | *20* | *10* | *30* | *2* | $\{2, 3\}$ | *10003* | *3* | *{15..19}* |

— We generate information which links back the columns of the merged table to its child and parents tables. This information will be used later when creating the MiniZinc model to synthesise the code that accesses the original child and parents tables.

**Example 4.1.5.** *The full merged table obtained after merging Table 3.4 and Table 3.3 is the Table 4.1 shown below.*

Table 4.1 – Merged table obtained after merging Table 3.3 and Table 3.4

| Task_id | Start | Quantity | Duration | Machine | Machines_set | Successor | Speed | Unavailability |
|---|---|---|---|---|---|---|---|---|
| 10001 | 0 | 10 | 20 | 10 | {5, 10} | 10002 | 2 | {30..39, 50..59} |
| 10002 | 20 | 10 | 30 | 2 | {2, 3} | 10003 | 3 | {15..19} |
| 10003 | 50 | 10 | 20 | 3 | {3, 5} | 10004 | 2 | {40..49, 70..79} |
| 10004 | 70 | 10 | 20 | 10 | {10} | −1 | 2 | {30..39, 50..59} |
| 10005 | 0 | 4 | 12 | 2 | {2, 5} | 10006 | 3 | {15..19} |
| 10006 | 20 | 4 | 8 | 10 | {2, 10} | 10007 | 2 | {30..39, 50..59} |
| 10007 | 28 | 4 | 8 | 3 | {3, 5} | −1 | 2 | {40..49, 70..79} |
| 10008 | 0 | 9 | 36 | 5 | {5, 10} | 10009 | 4 | {40..49, 70..79} |
| 10009 | 50 | 9 | 27 | 2 | {2} | −1 | 3 | {15..19} |

The module was developed by the author of the thesis.

### 4.1.5 Generating a MiniZinc model

The model conversion module is used to convert acquired by other modules constraints stored in an internal database into a MiniZinc file. The full description is given in Section 9. The module is fully developed by the author of the thesis.

## 4.2 Workflow of operations for acquiring sharp bounds on characteristics of combinatorial objects

The conjecture acquisition workflow consists of the following steps:

1. The input tables are generated (see Section 10.1).
2. The model acquisition system analyses independently each data table to produce metadata files.
3. The model acquisition system combines the metadata information extracted from each table in a meta-metadata file.
4. The system uses the meta-metadata file to identify the order in which to explore the the input tables and uses this order to focus on acquiring conjectures for sharp bounds of characteristics of combinatorial objects. It will store all found conjectures into a conjecture file.

## 4.3 Workflow of operations for acquiring STPS

The model acquisition workflow consists of the following steps:

1. The model acquisition system analyses independently each data table describing various aspects of a scheduling plan, e.g. the description of tasks and resources, to produce metadata files.
2. The model acquisition system combines information extracted from each table in some aggregated table from which the model will be acquired.
3. The model acquisition system produces the metadata file for the aggregated table. Temporal constraints are acquired in this step.
4. Using some identified MFDs, the model acquisition system focuses on acquiring functional constraints which explain the value of an attribute wrt to the values of some other attributes.
5. The model acquisition system acquires typical scheduling constraints, i.e. resource and calendar constraints.
6. The model acquisition system generates a MiniZinc model from the different types of constraints it learns in the previous steps. The model is generated in a way that each class of constraints is clearly commented and outlined in a generic way that allows one to change input data, e.g. to modify the set of available machines for a task.

Chapter 5

# Acquiring primary and foreign keys and ranking functional dependencies

This chapter presents contributions of the author of the thesis to the creation of metadata focusing on the acquisition of scheduling models. Sections 5.1 and 5.2 provide descriptions of generation of primary and foreign keys for the table respectively. Next, in Section 5.3, the process of ranking FDs for the formulae search is described.

## 5.1 Generation of primary keys of an input table

If the user does not provide the intended primary key (PK) of an input table then the model acquisition tool generates a candidate PK. There are two reasons for this:

— it can be used to combine two or more tables together,

— it is used to filter out columns that cannot be used during the constraint acquisition, as these columns usually contain information that is not numerical before converting the input tables.

Note that primary keys are only generated for tables corresponding to STPS. We do not generate PKs for tables that contain information about sharp bounds of characteristics of combinatorial objects.

In this section, we first present the problem statement for finding a candidate PK, as well as some conditions that help to simplify the search for candidate PKs. We then describe four search algorithms for candidate PKs and conclude with a quantitative evaluation of each algorithm.

### 5.1.1 Problem statement

The task is to find all viable candidates for the PK from a table with $n$ columns and $m$ rows, sorted by a cost function to rank them. We assume that all columns have at least two unique values, and that no column is the same as another, meaning that there is at least one row that has different values within those columns.

An algorithm that searches for candidate PKs presents its results as a list of sets $\{Cost, [B_1..B_n]\}$, where $B_i, i = 1..n$ is a list of Boolean variables. If $B_i = 1$, it means that column $i$ is a part of a candidate PK. If $B_i = 0$, then column $i$ is not a part of a candidate PK.

In order for candidate PK search algorithms to work correctly, it is necessary that the cost function satisfies conditions:

— the cost must not decrease if a column is added to a candidate;

— the cost must not change if the order of columns included in the candidate is changed.

Here are some examples of cost functions:

— $Cost = \sum_{i=1}^{n} ColRange_i \times B_i$,

— $Cost = \sum_{i=1}^{n} ColNVal_i \times B_i$,

— $Cost = \sum_{i=1}^{n} (ColRange_i - ColNVal_i) \times B_i$,

— $Cost = \prod_{i=1}^{n} ColRange_i^{B_i}$,

— $Cost = \prod_{i=1}^{n} ColNVal_i^{B_i}$,

— $Cost = \prod_{i=1}^{n} ColRange_i^{B_i} + \sum_{i=1}^{n} B_i$,

— $Cost = \prod_{i=1}^{n} (ColRange_i - ColNVal_i)^{B_i}$,

where:

— $ColRange_i$ is the range of values of column $i$, i.e. $ColRange_i = Max_i - Min_i + 1$, where $Min_i$ and $Max_i$ are minimal and maximal values of the column $i$,

— $ColNVal_i$ is the number of distinct values within column $i$.

We exclude dominated candidates from the results. The candidate $i$ is dominated by the candidate $j$ when both conditions are satisfied:

— candidate $j$ has a lower cost than $i$;

— the set of columns of candidate $j$ is the subset of columns of candidate $i$.

For all algorithms, it is possible to limit the maximum cardinality of a candidate, i.e. the maximum number of columns in the candidate. By default, the maximum cardinality for all algorithms is set to three as primary keys in the real life applications rarely use four or more columns.

### 5.1.2 Necessary conditions for a subset of columns to be a primary key candidate

For the set of columns $S$, it is possible to perform a preliminary check to determine as it is faster to first check necessary conditions whether the set $S$ can be or cannot be a candidate PK. If the necessary conditions for a subset of columns are satisfied, then we proceed with the thorough check in accordance with the selected algorithm, i.e. we apply the global constraint LEX_ALLDIFFERENT [169] on all projected entries of the table for the given subset of columns.

**Definition 4** (necessary condition 1)**.** *If a subset of columns $S$ is a candidate PK, then the maximum number of unique combinations of column distinct values must be equal or greater than the number of entries in the table:*

$$\prod_{i \in S} ColNVal_i \geq m$$

**Definition 5.** *$MaxOccurences_i$ for the column $i$ is the maximum number of rows where the same distinct value is presented.*

*MaxOccurences* for all columns are precalculated and stored in the metadata beforehand.

**Definition 6** (necessary condition 2)**.** *If a subset of columns $S$ is a candidate PK, then for every $S_1$ and $S_2$, such that $S_1 \cup S_2 = S, S_1 \cap S_2 = \varnothing$, the following inequality must be true:*

$$\sum_{i \in S_1} MaxOccurences_i - (|S_1| - 1) \times m \leq \prod_{i \in S_2} ColNVal_i$$

*Remark* 1. Necessary condition 1 is a particular case of necessary condition 2, when $S_1 = \varnothing$.

*Remark* 2. Necessary condition 2 can be generalised further, when not only most often occurrences are considered, but also second ones, third ones, etc. If $Occurrence_{i,j}$ is a j-most occurrence of a distinct value within the column $i$, $S$ is a candidate PK, then for every $S_1$ and $S_2$, $S_1 \cup S_2 = S$, $S_1 \cap S_2 = \varnothing$, the inequality must be true:

$$1 \leq l_1 \leq \min_{i \in S_1} NVal_i$$

$$\sum_{i \in S_1} \sum_{j=1}^{l_1} Occurences_{i,j} - (|S_1| - 1) \times m \leq l_1^{|S_1|} \times \prod_{i \in S_2} ColNVal_i$$

The computation experiments showed that using these generalised conditions for $l_1 \geq 2$ do not improve the performance of algorithms 1 and 2.

Using necessary conditions 1 and 2 to prune the initial set of candidate solutions helps reduce the search space for an algorithm that tries to produce the list of candidate solution. In Sections 5.1.3–5.1.6 four PK candidates search algorithms are described, two of which use the necessary conditions 1 and 2. The four algorithms are evaluated in Section 5.1.7.

Each of these algorithms takes a table and a list of candidate columns as an input. The output of each algorithm is a list of candidate PKs sorted in the ascending order WRT to the chosen cost function.

### 5.1.3 Algorithm 1 for searching candidate PKs

1. *Candidate generation*. Generate set of all column combinations and put them in the list of candidate PKs in increasing order of cardinality.
2. *Candidate elimination*:
   (a) select the first element of the list of candidate PKs. If the list is empty, stop the process;
   (b) check necessary conditions 1 and 2 for the chosen candidate PK. If they are satisfied, move to the step 2c. If one of the conditions is not satisfied, remove the selected candidate PK from the list and return to Step 2a.
   (c) for the selected candidate PK apply LEX_ALLDIFFERENT global constraint on all projected entries of the table:
      — if the candidate PK does not satisfy the constraint, remove it from the list of candidates and return to Step 2a;

— if the candidate satisfies the constraint, then put the candidate into the list of results and remove all dominated candidates from the list of candidate PKs. Return to Step 2a.

3. *Calculating costs*. Calculate the cost of each solution in the list of the results. Sort the list of solutions by cost.

The step *2b* is not required for the algorithm to function, but the usage of necessary conditions 1 and 2 improves the performance of the algorithm.

### 5.1.4 Algorithm 2 for searching candidate PKs

1. *Candidate generation*. Generate the constraint model. Results of this model will automatically satisfy necessary condition 1.

$$B_i = 0 \iff C_i = 1$$

$$V_i = 1 \iff C_i = NVal_i$$

$$\prod_{1}^{n} C_i \geq NRows,$$

where $B_i, i = 1..n$ - Boolean variables representing the inclusion of columns in the solution, and $C_i$ - integer variables.

Find all valid solutions and calculate their costs. Put the solutions in a list of all candidate PKs. Sort the list by cost and cardinality of a solution.

2. *Candidate elimination*:
   (a) take the first element of the list of candidates. If the list empty, stop the process;
   (b) check necessary conditions 2. If they are satisfied, move to the step 2c. If one of the conditions is not satisfied, remove the candidate from the list, take the next one and return to Step 2a.
   (c) apply LEX_ALLDIFFERENT global constraint on the first element of the list of candidates:
      — if the candidate does not satisfy the constraint, remove it from the list of candidates and return to Step 2a;
      — if the candidate satisfies the constraint, then put the candidate into the list of results and remove all dominated candidates from the list of candidates. Return to Step 2a.

3. *Result presentation.* Calculate the cost of each solution in the list of the results. Sort the list of solutions by cost.

### 5.1.5 Algorithm 3 for searching candidate PKs

1. *Generation of constraints.* For each pair of entries $i$ and $j$ of the input table generate a constraint:

$$\sum_{\forall k, \mathrm{tab}[i,k] \neq \mathrm{tab}[j,k]} B_k \geq 1, i \neq j, \tag{5.1}$$

where $k$ is a column index of a table 'tab'. The constraint ensures that at least one column for which can help differentiate between rows $i$ and $j$ is a part of a candidate PK. For example, for two rows of a table with five columns below:

| $B_1$ | $B_2$ | $B_3$ | $B_4$ | $B_5$ |
|---|---|---|---|---|
| 1 | 4 | 3 | 6 | 3 |
| 1 | 3 | 3 | 6 | 1 |

columns $B_1$, $B_3$ and $B_4$ are not necessarily part of a PK candidate. But either or both columns $B_2$ and $B_5$ must be a part of a candidate PK. To reflect this, a constraint $B_2 + B_5 \geq 1$ is posted.

2. *Solving the constraint model.* Solve the stated CSP and store every valid solution in a list. Calculate the cost of each solution in the list. Sort the list of solutions by cost.
3. *Candidate elimination.* Remove all dominated candidates and output from the final list of results.

### 5.1.6 Algorithm 4 for searching candidate PKs

1. *Generation of constraints.* The approach is similar to *Algorithm 3*. The difference from *Algorithm 3*, is that not all pairs of entries of the input table are used to save time.

   For each column:

   (a) the table is sorted by the selected column and the values of this column is taken as the key;

(b) all pairs of rows with the same value of the key are taken to form constraints, add them to the list of constraints.

2. *Solving the constraint model.* Solve the stated CSP and store every valid solution in a list. Calculate the cost of each solution in the list. Sort the list of solutions by cost.
3. *Candidate elimination.* Remove all dominated candidates and output from the final list of results.

### 5.1.7 Performance comparison

Table 5.1 compares search times of Algorithms 1, 2, 3 and 4. Datasets *testsales1000*, *testsales2000*, *testsales5000* are based on the table "Internet Sales" from AdventureWorks sample databases [170]. Datasets *testrandom1000* and *testrandom5000* are generated with random values with the exception of the first three columns, which are generated with a predetermined pattern.

Table 5.1 – Comparison between candidate PK search algorithms

| Dataset | Cols | Rows | Alg. 1 | Alg. 2 | Alg. 3 | Alg. 4 |
|---|---|---|---|---|---|---|
| *testsales1000* | 20 | 1000 | 99ms | 101ms | 8s | 9s |
| *testrandom1000* | 40 | 1000 | 7s | 18s | 1m30s | 3m10s |
| *testsales3000* | 20 | 3000 | 380ms | 383ms | 1m20s | 1m35s |
| *testsales5000* | 20 | 5000 | 629ms | 635ms | 4m55s | 3m20s |
| *testrandom5000* | 50 | 5000 | 1m45s | 2m25s | N/A | N/A |

Algorithms 3 and 4 took more than 24h to solve the *testrandom5000* problem and their results are not included in the table.

The model acquisition tool uses Algorithm 1 together with the cost function $Cost = \prod_{i=1}^{n} ColRange_i^{B_i} + \sum_{i=1}^{n} B_i$. This criteria minimises the number of possible unique values of a given PK, i.e. $\prod_{i=1}^{n} ColRange_i^{B_i}$, and, if there are two candidate PKs with the same number of possible unique values, it provides the preference to a candidate PK with a fewer number of columns by using $\sum_{i=1}^{n} B_i$.

## 5.2 Generation of foreign keys of an input table

To combine two tables into one, we need to know both PKs of both tables and a foreign key (FK) of the child table. If the user does not provide the FK, the model acquisition

tool searches for the best candidate FK. It could be done by acquisition of semantic constraints [171], which is outside of the scope of this thesis. Instead, in this section a simple algorithm which solves the stated problem is presented below.

Given the list of all tables, the system checks every pair of potential children and parents. For each Child and Parent tables we:

1. take the list of candidate PKs from the Parent table;
2. take a candidate PK from the list, project it on the entries of Child table (*Child PK entries*);
3. search for all subsets of columns of Parent table for which, after the projection on the entries of Parent table (*Parent FK entries*), the resulting entries contain every entry from *Child PK entries*;
4. assign cost for each found subset. The cost is the coverage of *Parent FK entries* by *Child PK entries*. The higher the coverage, the better;
5. return to step 2 and repeat the process until all candidate PKs are considered, in which case we move to the next step;
6. select the candidate FK with the highest cost and record it in the metadata file.

## 5.3 Ranking functional dependencies

As previously said in Section 4.1.1, it is required to select FDs which is more likely to produce a functional constraint. In the literature, it is referred as a genuine FD discovery problem [172, 173]. In order to do this we rank FDs according to the following three heuristic criteria in ascending lexicographic order:

1. The number of distinct row values within the selected functional dependency (see Section 5.3.1).
2. The maximal correlation coefficient between the selected functional dependency and the output column. The model acquisition tool first calculates either the Pearson correlation coefficient between a single input column and the output column, or an adjusted multiple correlation coefficient between several input columns and the output column. Second, based on the selected functional dependency, it calculates intermediate columns on number of formulae corresponding to some non-linear learning biases and calculates the Pearson correlation coefficients for between each interme-

diate column and the selected output column. Out of all The model acquisition then takes the maximal calculated correlation coefficient (see Section 5.3.2).

3. The number of columns within the selected functional dependency (see Section 5.3.3).

For each column in the merged table, we calculate and record a ranked list of functional dependencies.

Sections 5.3.1–5.3.3 provide the reasoning behind each criteria and detailed description on how the given criteria is calculated. Then, Section 5.3.4 provides a detailed explanation on how the ranking of FDs is done and how we select the best candidate FD from the list. Section 5.3.5 provides an empiric evaluation of the proposed ranking strategy.

### 5.3.1 Number of distinct vectors within a selected functional dependency

The reasoning behind this criteria is that the number of distinct rows within a genuine functional dependency will likely be close to the number of distinct values in the output column [168, 174]. An advantage of this criteria is that it can be used for both numerical and non-numerical data entries.

A limitation of this criteria is that there are situations where columns of an FD with a large number of unique values lead to outputs with a limited number of distinct values. e.g. if we have a column *duration* which is a result of the calculation $Duration[i] = End[i] - Start[i]$, then the column *Duration* can potentially have a small number of distinct values, while columns *Start* and *End* are likely to have a large number of distinct values each.

Nonetheless, preliminary tests showed that this is the strongest criteria out of three.

### 5.3.2 Correlation between a functional dependency and its output

The reasoning behind this criteria is following. If there is a correlation coefficient between inputs and an output it is more likely that there is a relation between them.

To do so we calculate either a Pearson's correlation coefficient between a single column FD and the output column or an adjusted multiple correlation coefficient between an FD with multiple columns and the output column. Unlike the first criteria, this criteria only works with numerical data. Pearson's correlation and multiple correlation coefficients are

only useful when searching for linear formulae. As a result, a further generalisation is needed to accommodate a variety of learning biases. To achieve this goal:

— for a single column FD we calculate:
  — Pearson's correlation coefficient between the FD and the output column,
  — Pearson's correlation coefficient between $1/X$, where X is a value in the row for FD, and the output column,

  and take the maximum absolute value out of two;

— for a two columns FD we assume that $X$ and $Y$ are values in an entry of the table for the FD in first and second columns respectively. Then we calculate:
  — adjusted multiple correlation coefficient between FD and the output column,
  — Pearson's correlation coefficient between the output column and the vector with values $X + Y$ for each entry of the table,
  — Pearson's correlation coefficient between the output column and the vector with values $X - Y$ for each entry of the table,
  — Pearson's correlation coefficient between the output column and the vector with values $X \times Y$ for each entry of the table,
  — Pearson's correlation coefficient between the output column and the vector with values $\lfloor X/Y \rfloor$ for each entry of the table,
  — Pearson's correlation coefficient between the output column and the vector with values $\lfloor Y/X \rfloor$ for each entry of the table,
  — Pearson's correlation coefficient between the output column and the vector with values $min(X, Y)$ for each entry of the table,
  — Pearson's correlation coefficient between the output column and the vector with values $max(X, Y)$ for each entry of the table,

  and take the maximum absolute value out of all of them;

— for FD with $n \geq 3$ columns, we calculate:
  — adjusted multiple correlation coefficient between the FD and the output column;
  — for each possible vector $V$, where $V_i \in \{-1, 1\}, i \in 1..n$, we:
    — produce a new vector of values $\sum_i^n X_i \times V_i$, where $X_i$ a value of an entry of the table in a column $i$ of the FD,

— calculate Pearson's correlation coefficient between calculated values and the output column.

— every possible adjusted multiple correlation coefficient between FD, with the values of one of the column being squared, and the output column.

and take the maximum absolute value out of all of them.

The higher the absolute value of the coefficient the more genuine FD appears to be.

*Remark* 3. Every Pearson's correlation coefficient and multiple correlation coefficient is tested for statistical significance for $p = 0.05$.

*Remark* 4. Specific correlations calculated for an FD can depend on the specific application domain. e.g. $X + Y$, $X - Y$ or $\sum_i^n X_i$ are likely to be encountered in scheduling models.

### 5.3.3 Number of columns within a functional dependency

This is the weakest criteria out of three. If we have two FD with identical values on first two criteria we have a preference for an FD with a smaller number of columns.

### 5.3.4 Selecting functional dependencies

For every FD all three criteria are calculated. FDs with maximum absolute value of a correlation coefficient lesser than 0.5 are discarded from consideration. Afterwards, all FDs are sorted in ascending lexicographic order. Then the top $N$ FDs are taken, where $N$ is preselected. The current version of the model acquisition tool sets $N = 20$.

**Example 5.3.1.** *After adding a column to the ten minimal functional dependencies listed in Example 4.1.1, and ranking the corresponding set, we obtain the following four best ranked functional dependencies sorted in ascending order wrt their respective cost vector.*

1. {*'Quantity', 'Speed'*} *with cost* $(6, -0.98, 2)$,
2. {*'Start', 'Quantity', 'Speed'*} *with cost* $(9, -0.98, 3)$,
3. {*'Start', 'Quantity'*} *with cost* $(9, -0.75, 2)$,
4. {*'Start', 'Speed'*} *with cost* $(9, 0, 2)$.

*Values* 6 *and* 2 *from the cost vector* $(6, -0.98, 2)$ *of the best ranked functional dependencies* {*'Quantity','Speed'*} *are respectively explained by:*

- *Value* 6 *corresponds to the number of distinct pairs of values in the 'Quantity' and 'Speed' columns of the merged table Table 4.1, namely* $(10, 2)$, $(10, 3)$, $(4, 3)$, $(4, 2)$, $(9, 4)$, $(9, 3)$.
- *Value* 2 *is the number of columns of the best ranked functional dependency, i.e.* {*'Quantity', 'Speed'*}.

*The best ranked functional dependency is used to find the functional constraint:*

$$\textit{'Duration'} = \textit{'Quantity'} \times \textit{'Speed'}.$$

### 5.3.5 Assessment of the functional dependencies ranking process

Forty test tables were generated with random formulae using random input columns. On average, about 400 minimal FDs and their derivatives were considered for each table. The proposed ranking criteria was applied. The results showed that:

— Two of the tables did not generate a required genuine FD. This can be solved by adding two columns to a minimal functional dependency instead of one.

— For two tables the intended genuine FD got ranked very low (85 and 132, respectively).

— For two tables the intended genuine FD got ranked between first 30 and first 40 positions.

— For the remaining 34 tables (34 tables) genuine FDs got ranked within first 30 positions.

While not perfect, it shows that the proposed ranking criteria helps localize the genuine FD in most cases.

Part IV

# Acquiring conjectures and functional constraints

Chapter 6

# New biases for functional constraints acquisition

Previously, in Chapter 2, we described several settings for the practical use of Boolean expressions that we observed in the context of sharp bound acquisition and in Section 1.2 we discussed the related work.

In this chapter, we provide the description of the new bias, Boolean-arithemtic expressions and its applications in the acquisition of case formulae conjectures. In Section 6.1, we define the Boolean-arithmetic formulae that we consider throughout this paper. In Section 6.2, we provide a core CP model for learning a BAE that explains an output column of a table from a set of input columns. In Section 6.5, we show how BAE acquisition can be used to acquire conditions for known clusters. We show in Section 6.3 various extensions of the core model to restrict the search space of Boolean-arithmetic expressions. We evaluate the core model and its extensions in Section 6.6.

## 6.1 Describing Boolean-arithmetic expressions

The BAEx we consider is dictated by two opposite objectives.

— On the one hand, we want to focus on concise expressions involving few variables and constants. This is motivated by the need to generate formulae that can be interpreted by a human being, and by the necessity to avoid a combinatorial explosion when searching for such formulae [152], for efficiency reasons. Consequently, we limit the number of variables and constants, as well as the number of sub-terms of Boolean-Arithmetic expressions.

— On the other hand, we aim at covering a variety of Boolean expressions which occurs in practice. This is done by allowing one to use a variety of arithmetic operators and Boolean functions.

To meet the above objectives, we use the following two-level description:

— First, we consider a Boolean-arithmetic condition (BAC) mentioning *no more than three variables and two constants*, the comparison operators $\leq$, $=$, $\geq$, $>$, $\in$ and a *variety of arithmetic operators* such as $+$, $-$, $\times$, $\lfloor\text{-}\rfloor$, $\lceil\text{-}\rceil$, mod, min, max. We have 57 elementary arithmetic conditions listed in Table 6.1, where $x$, $y$, $z$ represent variables and $c$, $d$ constants. Only the '$\in$' condition may be negated: in fact ($a$) in the classes A1,A2,...,B17 all inequalities already have a negated counterpart, and ($b$) we do not negate equality constraints as this leads to weak constraints, i.e. constraints that accept many solutions.

— Second, we build a Boolean-arithmetic term by feeding several arithmetic conditions, or their negation, to a *commutative and associative aggregation operator* such as $+$, $\vee$, $\wedge$, $\oplus$, eq, card1, voting, where:

  — $\oplus$ stands for XOR;

  — eq is equal to 1 iff all its conditions are evaluated to the same value;

  — card1 is equal to 1 iff only one of its conditions is evaluated to 1;

  — voting is equal to 1 iff the majority of its conditions are evaluated to 1.

The use of a commutative and associative aggregation operator simplifies the interpretability of a formula and reduces the combinatorics, as the order of the BACs within a Boolean-arithmetic term is irrelevant. It allows for a compact representation of some Boolean expressions, that would otherwise be large when expressed in conjunctive or disjunctive normal form without introducing new variables. For example, the n-ary XOR, i.e. $\oplus_{i=1}^{n} \ell_i$, is represented by a CNF consisting of $2^{n-1}$ clauses, where each clause mentions all literals $\ell_1, \ell_2, \ldots, \ell_n$. It also permits the use of the '$+$' operator in a natural way.

### 6.1.1 Limiting the complexity of Boolean-arithmetic expressions

To restrict the space of possible Boolean arithmetic expressions, which is huge since it results from the combination of 57 elementary arithmetic conditions listed in the second column of Table 6.1, we use the following empirical observations.

— It is rare for a Boolean arithmetic expression to mention several conditions using the same arithmetic operator, especially if the arithmetic operators mention many variables. To implement this idea, we classify conditions into three categories A, B, and C depending on the number of variables in the condition. Categories A, B,

and C respectively consist of different classes A1–A2, B1–B17, C1–C5. The number of occurrences of conditions in the same class is unlimited for A, restricted to two occurrences for B, or reduced to a single occurrence for C.

— Considering that (*i*) the arithmetic operators sum, difference, and product are fairly common, but that other operators such as mod are less common, and that (*ii*) the use of more than one condition involving three variables is rare, we impose that all classes B6–C5 are incompatible. For example, if we use a condition from B6, we will not use any condition from B7–C5.

Table 6.1 – List of the 57 considered conditions with their assigned respective cost that will be used in Section 6.4; within a condition, $x$, $y$, $z$ are variables, $\underline{y}$ stands for the minimum value of $y$, $c$ and $d$ are constants, and $(cond\,?\,e_1 : e_2)$ denotes expression $e_1$ if condition *cond* holds, expression $e_2$ otherwise.

| Class | Conditions (with their costs in parenthesis) |
|---|---|
| A1 | $x = c$ (1), $x \leq c$ (1), $x \geq c$ (1) |
| A2 | $x \in [c, d]$ (0) |
| B1 | $x = y$ (0), $x \leq y$ (0) |
| B2 | $x = c \cdot y$ (2), $x \leq c \cdot y$ (3) |
| B3 | $x = c \cdot y$ (2), $c \cdot x \leq y$ (3) |
| B4 | $x + y = c$ (2), $x + y \leq c$ (3), $x + y \geq c$ (3) |
| B5 | $x - y = c$ (2), $x - y \leq c$ (3), $x - y \geq c$ (3) |
| B6 | $x \bmod c = d$ (6), $x \bmod c \leq d$ (7), $x \bmod d \geq d$ (7) |
| B7 | $\lvert x - y \rvert = c$ (3), $\lvert x - y \rvert \leq c$ (4), $\lvert x - y \rvert \geq c$ (4) |
| B8 | $\min(x, y) = c$ (3), $\min(x, y) \leq c$ (4), $\min(x, y) \geq c$ (4) |
| B9 | $\max(x, y) = c$ (3), $\max(x, y) \leq c$ (4), $\max(x, y) \geq c$ (4) |
| B10 | $x \cdot y = c$ (4), $x \cdot y \leq c$ (5), $x \cdot y \geq c$ (6) |
| B11 | $\lfloor x/y \rfloor = c$ (4), $\lfloor x/y \rfloor \leq c$ (5), $\lfloor x/y \rfloor \geq c$ (5) |
| B12 | $\lfloor x/(\max(y - \underline{y}, 1)) \rfloor = c$ (7), $\lfloor x/(\max(y - \underline{y}, 1)) \rfloor \leq c$ (8), $\lfloor x/(\max(y - \underline{y}, 1)) \rfloor \geq c$ (8) |
| B13 | $\lceil x/y \rceil = c$ (4), $\lceil x/y \rceil \leq c$ (5), $\lceil x/y \rceil \geq c$ (5) |
| B14 | $x \bmod y = c$ (6), $x \bmod y \leq c$ (7), $x \bmod y \geq c$ (7) |
| B15 | $x - (y \bmod x) = c$ (7), $x - (y \bmod x) \leq c$ (8), $x - (y \bmod x) \geq c$ (8) |
| B16 | $x - (x \bmod y) = c$ (7), $x - (x \bmod y) \leq c$ (8), $x - (x \bmod y) \geq c$ (8) |
| B17 | $(x = 0\,?\,y : x \bmod y) = c$ (7), $(x = 0\,?\,y : x \bmod y) \leq c$ (8), $(x = 0\,?\,y : x \bmod y) \geq c$ (8) |
| C1 | $x + y \leq z$ (100) |
| C2 | $(x - y) \bmod z = 0$ (100) |
| C3 | $\lceil (x - y)/z \rceil \leq \lfloor (x - z)/y \rfloor$ (100) |
| C4 | $x \geq (y = 0 : z\,; y \bmod z)$ (100) |
| C5 | $x \bmod y > z \bmod y$ (100) |

## 6.2 A core model for acquiring Boolean-arithemtic equations

This section introduces a CP-based core model for acquiring BAE. First, the model relies on reified constraints to represent learned Boolean expressions that mention a re-

stricted number of arithmetic conditions taken from a large set of candidate conditions. Second, the model incorporates symmetry-breaking constraints resulting from the relaxation of arithmetic conditions. Section 6.3 will extend these symmetry-breaking constraints by considering also commutative arithmetic operators, as well as conditions that mention the same comparison and arithmetic operators with different attributes.

### 6.2.1 Problem description

Given a two-dimensional table $\text{tab}[1..r, 1..c]$ of integer values, consisting of $r$ distinct rows and $c$ distinct columns, where column $c$ is functionally determined by columns $1, 2, \ldots, c-1$, the problem is to come up with a constraint model to acquire an equality constraint of the form

$$\forall j \in [1, r] : \ \text{tab}[j, c] = f(\text{tab}[j, 1], \text{tab}[j, 2], \ldots, \text{tab}[j, c-1]) \tag{6.1}$$

i.e. a constraint that is valid for all rows of the table, where $f$ is a Boolean-arithmetic expression mentioning $c-1$ parameters and $n_{\text{AC}}$ conditions, where $n_{\text{AC}}$ will be called the *arity* of the Boolean-arithmetic Equation (6.1).

As we want to restrict the complexity of the acquired formulae, the expression $f$ is limited to $n_{\text{AC}} \in \{1, 2, 3\}$ conditions taken from a list of conditions $C$, where each condition is chosen from $m = 57$ potential distinct BACs introduced in Section 6.1 (where a small number of conditions may be duplicated using different constants), and a single commutative and associative aggregation operator $g$ selected from the set $\{\vee, \wedge, \oplus, +, \text{eq}, \text{card1}, \text{voting}\}$. As the acquisition system successively tries the different aggregation operators, we assume from now on that $g$ is fixed. As we search for Boolean-arithmetic expressions by increasing number of BACs, we also assume that $n_{\text{AC}}$ is fixed to a value in $\{1, 2, 3\}$.

Each potential candidate BAC $C_d$ of $f$ (with $d \in [1, m]$) mentioning $\ell_d$ columns of the table 'tab' (with $\ell_d \in [1, 3]$) and $\ell'_d$ coefficients (with $\ell'_d \in [0, 2]$) is represented by the term $C_d \begin{pmatrix} a_{d,1}, \ldots, a_{d,\ell_d}, \\ c_{d,1}, \ldots, c_{d,\ell'_d} \end{pmatrix}$, where:

- the variables $a_{d,1}, \ldots, a_{d,\ell_d}$ denote the indices of the distinct columns of the table $\text{tab}[1..r, 1..c]$ mentioned by condition $C_d$,
- the variables $c_{d,1}, \ldots, c_{d,\ell'_d}$ represent the coefficients used in the arithmetic expression of condition $C_d$.

The problem is to come up with a CP-based model which, given $(i)$ a commutative

and associative Boolean operator $g \in \{\vee, \wedge, \oplus, +, \text{eq}, \text{card1}, \text{voting}\}$, and (*ii*) a fixed number of conditions $n_{\text{AC}}$, extracts the subset of relevant conditions for the expression $f$ of Constraint (6.1), and finds for each used conditions all its parameters, i.e. which columns and which coefficient values it uses.

**Example 6.2.1.** *To illustrate the section on describing the problem, we provide an example of a table and the corresponding acquired BAE. On the page 101, the left-hand side of Table 6.3 provides a table* $\text{tab}[1..9, 1..4]$ *from which we acquire the following BAE* $x_4 = [(x_1 - x_2) = 2] \vee [x_3 \leq 4]$. *The acquisition process is now explained in Section 6.2.2.*

## 6.2.2 A CP core model

**Notation 1.** *Given a table* $\text{tab}[1..r, 1..c]$*, the* $j$*-th row of* $\text{tab}[1..r, 1..c]$ *is called a* negative entry *if* $\text{tab}[j, c] = 0$*, and a* positive entry *otherwise.*

### 6.2.2.1 Selecting the BACs used in $f$

To each potential BAC $C_d$ (with $d \in [1, m]$) of a Boolean-arithmetic expression $f$, we associate a variable $b_d$ such that:

- $b_d = -1$ means that neither condition $C_d$, nor condition $\neg C_d$ are used in $f$,
- $b_d = \ \ 0$ indicates that the condition $\neg C_d$ is used in $f$, i.e. $C_d$ is negated,
- $b_d = \ \ 1$ signifies that the condition $C_d$ occurs in $f$.

As $f$ should mention $n_{\text{AC}}$ BACs, we set up the following AMONG constraint [163] to specify that $m - n_{\text{AC}}$ conditions must be unused:

$$\text{AMONG}\,(m - n_{\text{AC}}, \langle b_1, b_2, \ldots, b_m \rangle, -1) \tag{6.2}$$

### 6.2.2.2 Selecting the attributes used in each BAC

For each potential condition $C_d \begin{pmatrix} a_{d,1}, \ldots, a_{d,\ell_d}, \\ c_{d,1}, \ldots, c_{d,\ell'_d} \end{pmatrix}$ (with $d \in [1, m]$), we set all its variables $a_{d,1}, a_{d,2}, \ldots, a_{d,\ell_d}$ to 0 when the condition $C_d$ is not used, i.e. when $b_d = -1$. We introduce the variables $a'_{d,1}, a'_{d,2}, \ldots, a'_{d,\ell_d}$ corresponding to $a_{d,1} + 1, a_{d,2} + 1, \ldots, a_{d,\ell_d} + 1$: we use the offset $+1$ as these variables will also be used in ELEMENT constraints [163] whose index starts at 1.

For each potential condition $C_d$, its variables $a'_{d,1}$, $a'_{d,2}$, ..., $a'_{d,\ell_d}$ should either be all distinct and greater than or equal to 2, or be all equal to 1. This is expressed by the next GLOBAL_CARDINALITY (GCC) constraint [175]:

$$\text{GCC}\left(\begin{array}{c}\left\langle a'_{d,1}, a'_{d,2}, \ldots, a'_{d,\ell_d}\right\rangle, \\ \langle 1:\{0,\ell_d\},\ 2:\{0,1\},\ \ldots\ ,\ c:\{0,1\}\rangle\end{array}\right) \tag{6.3}$$

When the condition $C_d$ is unused, i.e. $b_d = -1$, we set all its variables $a'_{d,1}, a'_{d,2}, \ldots, a'_{d,\ell_d}$ to 1 to break symmetry, i.e. to avoid enumerating over these variables:

$$\forall d \in [1,m], \forall k \in [1,\ell_d] : b_d = -1 \Leftrightarrow a'_{d,k} = 1 \tag{6.4}$$

To force the use of all attributes from 1 to $c-1$ of the table tab$[1..r, 1..c]$ across all selected conditions, i.e. those conditions $C_d$, (with $d \in [1,m]$) such that $b_d \neq -1$, we set up the following GCC constraint:

$$\text{GCC}\left(\begin{array}{cc}\left\langle\begin{array}{c}a'_{1,1}, a'_{1,2}, \ldots, a'_{1,\ell_1}, \\ a'_{2,1}, a'_{2,2}, \ldots, a'_{2,\ell_2}, \\ \ldots\ldots\ldots\ldots\ldots\ldots \\ a'_{m,1}, a'_{m,2}, \ldots, a'_{m,\ell_m}\end{array}\right\rangle, & \left\langle\begin{array}{c}2:o_2, \\ 3:o_3, \\ \ldots\ldots \\ c:o_c\end{array}\right\rangle\end{array}\right) \text{ with } o_i \in [1,m], \forall i \in [2,c] \tag{6.5}$$

#### 6.2.2.3 Restricting the coefficients of each BAC

When the BAC $C_d\left(\begin{array}{c}a_{d,1}, \ldots, a_{d,\ell_d}, \\ c_{d,1}, \ldots, c_{d,\ell'_d}\end{array}\right)$ is used, i.e. $b_d \neq -1$, the coefficient variables $c_{d,1}, \ldots, c_{d,\ell'_d}$ denote the coefficients used in the arithmetic expression related to $C_d$. As we look for simple formulae, the initial domain of such variables is initialised to a small interval, e.g. $[-9,+9]$, which turned out to be not too restrictive when considering sharp bounds on combinatorial objects.

Note that we are not interested in acquiring conditions that can be substituted by `true` or `false` as they could be simplified away. For some types of conditions, this would require additional constraints on the condition's coefficients, e.g. for $C_d(a_{d,1}, c_{d,1}, c_{d,2}) \equiv [(a_{d,1} \bmod c_{d,1}) = c_{d,2}]$ we post the additional constraints $b_d \neq -1 \Rightarrow c_{d,1} \geq 2$ and $b_d \neq -1 \Rightarrow c_{d,2} \in [0, c_{d,1}-1]$:

— If $c_{d,1} = 1$ then the condition $C_d(a_{d,1}, c_{d,1}, c_{d,2})$ is either always `true`, when $c_{d,2} = 0$, or always `false` when $c_{d,2} \neq 0$.

— Otherwise, if $c_{d,1} \geq 2$ and $c_{d,2} \notin [0, c_{d,1} - 1]$, then the condition $C_d(a_{d,1}, c_{d,1}, c_{d,2})$ is always `false` as $(a_{d,1} \bmod c_{d,1}) \in [0, c_{d,1} - 1]$.

When the condition $C_d$ is unused, we have $b_d = -1 \Rightarrow (c_{d,1} = \cdots = c_{d,\ell'_d} = 0)$ to avoid multiple solutions stemming from the coefficients of an unused condition. How to restrict further the initial domain of the coefficient variables wrt the entries of the table $\text{tab}[1..r, 1..c]$ to limit the search will be explained in Section 6.3.3.

#### 6.2.2.4 Setting row constraints

To evaluate each condition $C_d$ wrt the $j$-th row of the table $\text{tab}[1..r, 1..c]$, we create the variables $v_{d,j,k}$ for the values of its $k$-th attributes and a variable $b_{d,j}$ for the value of $C_d$. This is now described:

— For each condition $C_d$ (with $d \in [1, m]$), for each row $j$ (with $j \in [1, r]$), and for each argument $k$ (with $k \in [1, \ell_d]$) of condition $C_d$, we create a variable $v_{d,j,k}$ that gives, either the value of the $k$-th argument of condition $C_d$ wrt the $j$-th row of the table $\text{tab}[1..r, 1..c]$, or 0 if the condition $C_d$ is unused. This is expressed by ELEMENT constraint [163]:

- $\forall d \in [1, m]$, $\forall j \in [1, r]$, $\forall k \in [1, \ell_d]$ :
  $\text{ELEMENT}\left(a'_{d,k},\ \langle 0, \text{tab}[j, 1], \text{tab}[j, 2], \ldots, \text{tab}[j, c - 1]\rangle\ ,\ v_{d,j,k}\right)$.

— We also create a 0–1 variable $b_{d,j}$ which will be set to true iff condition $C_d$ holds for the $j$-th row of the table $\text{tab}[1..r, 1..c]$:

- $\forall d \in [1, m]$, $\forall j \in [1, r] : b_{d,j} \Leftrightarrow C_d\left(v_{d,j,1}, v_{d,j,2}, \ldots, v_{d,j,\ell_d}\right)$.

Now, based on the aggregator $g$, we state some row constraints for each used condition $C_d$ (with $d \in [1, m]$) and wrt each row of the table $\text{tab}[1..r, 1..c]$. These row constraints are related to the type of aggregator $g$ we are using. In this context, we distinguish the following types of aggregators I, II, and III:

I. Aggregators for which (*i*) positive and negative table entries have distinct row constraints and (*ii*) a single table entry may determine the value of each condition $C_d$. For example, if $g$ is the '$\wedge$' aggregator then on a positive entry, a condition $C_d$ which is false (with $d \in [1, m]$) falsifies the Boolean arithmetic expression $f$, which is impossible on the positive entries, meaning that each condition used in $f$ must be true. Aggregators '$\vee$' and '$\wedge$' belong to this class.

II. Aggregators for which (*i*) positive and negative table entries have distinct row constraints, and (*ii*) a single table entry cannot determine the value of each condition $C_d$. Aggregator 'eq' belongs to this class.

III. Aggregators for which (*i*) positive and negative table entries have the same row constraint, and (*ii*) a single table entry cannot determine the value of each condition $C_d$. Aggregators '+', '$\oplus$', 'card1', and 'voting' belong to this class.

Table 6.2 provides for each class in {I, II, III} of aggregator $g$ the corresponding row constraints that determine the value of the Boolean arithmetic expression $f$. As mentioned earlier, for the first two classes, these row constraints depend on whether we have a positive or negative table entry; for the third class, the same constraint applies for both a positive and a negative table entry. We now explain the constraints stated in Table 6.2 for the first aggregator of each class.

**[Case aggregator $g$ is '$\vee$' ]**

— For each positive row $j$ (with $j \in [1, r]$), we post the constraint $\vee_{d=1}^{m}[b_d = b_{d,j}]$ to ensure that *at least one condition is true* so that the disjunction of conditions holds.

— For each condition $C_d$ (with $d \in [1, m]$) and each negative row $j$ (with $j \in [1, r]$), we post the constraint $\textsc{table}(\langle(b_d, b_{d,j})\rangle, \langle(-1, 0), (-1, 1), (0, 1), (1, 0)\rangle)$ [3, pp 1400–1402]. When the condition $C_d$ is not used, i.e. $b_d = -1$, there is no restriction on $b_{d,j}$, i.e. $b_{d,j} \in \{0, 1\}$; otherwise, *each condition must be falsified*, i.e. $b_{d,j} = 1 - b_d$, so that the corresponding disjunction of conditions is not true.

**[Case aggregator $g$ is '$\wedge$' ]**

— For each condition $C_d$ (with $d \in [1, m]$) and each positive row $j$ (with $j \in [1, r]$), i.e. rows such that $\text{tab}[j, c] = 1$, we post the constraint $\textsc{table}(\langle(b_d, b_{d,j})\rangle, \langle(-1, 0), (-1, 1), (0, 0), (1, 1)\rangle)$. When the condition $C_d$ is not used, i.e. $b_d = -1$, there is no restriction on $b_{d,j}$; otherwise, the evaluation of the condition $C_d$ wrt the $j$-th row should be identical to the value $\text{tab}[j, c]$.

— For each negative row $j$ (with $j \in [1, r]$), i.e. rows such that $\text{tab}[j, c] = 0$, we post the constraint $\vee_{d=1}^{m}[b_c = \neg b_{d,j}]$ to falsify at least one condition so that the corresponding conjunction of conditions is not true.

**[Case aggregator $g$ is 'eq' ]**

— For each positive row $j$ (with $j \in [1, r]$), we post the constraint:

$$[\textstyle\sum_{d=1}^{m}[b_d = b_{d,j}] = n_{\text{AC}}] \ \vee \ [\sum_{d=1}^{m}[b_d = \neg b_{d,j}] = n_{\text{AC}}]$$

enforcing *either that all conditions hold or that all conditions are false.*

— For each negative row $j$ (with $j \in [1, r]$), we post the constraint:

$$[\textstyle\sum_{d=1}^{m}[b_d = b_{d,j}] < n_{\text{AC}}] \wedge [\sum_{d=1}^{m}[b_d = \neg b_{d,j}] < n_{\text{AC}}]$$

imposing that *at least one condition is false and at least one is true.*

Table 6.2 – Row constraints which are posted on a positive or a negative table entry for computing the value of a Boolean arithmetic expression $f$, depending on the used aggregator $g$ of classes I, II; for class III the same row constraint is posted for all entries.

| Class | $g$ | Positive entries ($\text{tab}[j,c]=1$) | Negative entries ($\text{tab}[j,c]=0$) |
|---|---|---|---|
| I | '$\vee$' | $\vee_{d=1}^{m}[b_d = b_{d,j}]$ | $\text{TABLE}\left(\langle(b_d, b_{d,j})\rangle, \left\langle \begin{array}{c}(-1,0),(-1,1),\\(\ \ 0,1),(\ \ 1,0)\end{array} \right\rangle\right)$ |
| | '$\wedge$' | $\text{TABLE}\left(\langle(b_d, b_{d,j})\rangle, \left\langle \begin{array}{c}(-1,0),(-1,1),\\(\ \ 0,0),(\ \ 1,1)\end{array} \right\rangle\right)$ | $\vee_{d=1}^{m}[b_d = \neg b_{d,j}]$ |
| II | 'eq' | $\bigvee \left( \begin{array}{l}\sum_{d=1}^{m}[b_d = \ \ b_{d,j}] = n_{\text{AC}},\\ \sum_{d=1}^{m}[b_d = \neg b_{d,j}] = n_{\text{AC}}\end{array} \right)$ | $\bigwedge \left( \begin{array}{l}\sum_{d=1}^{m}[b_d = \ \ b_{d,j}] < n_{\text{AC}},\\ \sum_{d=1}^{m}[b_d = \neg b_{d,j}] < n_{\text{AC}}\end{array} \right)$ |
| III | '+' | $\text{tab}[j,c] = \sum_{d=1}^{m}[b_d = b_{d,j}]$ | |
| | '$\oplus$' | $\text{tab}[j,c] = (\sum_{d=1}^{m}[b_d = b_{d,j}]) \bmod 2$ | |
| | 'card1' | $\text{tab}[j,c] = [(\sum_{d=1}^{m}[b_d = b_{d,j}]) = 1]$ | |
| | 'voting' | $\text{tab}[j,c] = [2 \cdot (\sum_{d=1}^{m}[b_d = b_{d,j}]) > n_{\text{AC}}]$ | |

**[Case aggregator $g$ is '+' ]** For each row $j$ (with $j \in [1, r]$), we post the constraint $\text{tab}[j,c] = \sum_{d=1}^{m}[b_d = b_{d,j}]$ to ensure that *the appropriate number of conditions are satisfied.*
**[Case $g$ is $\oplus$ ]** Using the observation that $x \oplus y$ is equal to $(x + y) \bmod 2$, and that $\oplus$ is associative, we post for each row $j$ (with $j \in [1, r]$) the constraint $\text{tab}[j,c] = (\sum_{d=1}^{m}[b_d = b_{d,j}]) \bmod 2$.

**[Case $g$ is** card1 **]** We post for each row $j$ (with $j \in [1, r]$) the constraint $\text{tab}[j,c] = [(\sum_{d=1}^{m}[b_d = b_{d,j}]) = 1]$.

**[Case $g$ is** voting **]** We post for each row $j$ (with $j \in [1, r]$) the constraint $\text{tab}[j,c] = [2 \cdot (\sum_{d=1}^{m}[b_d = b_{d,j}]) > n_{\text{AC}}]$.

Table 6.3 – Illustrating the core model on the table tab[1..9, 1..4] (with columns $x_1$, $x_2$, $x_3$, $x_4$) for acquiring a Boolean-arithmetic expression explaining $x_4$ wrt $x_1$, $x_2$, $x_3$ using the '$\vee$' aggregator with two conditions $C_1$ and $C_2$ selected from the following potential candidate conditions $C_1: \ x_i - x_j = cst$, $C_2: \ x_i \leq cst$ and $C_3: \ x_i = x_j$.

| | $j$ | table tab $x_1$ $x_2$ $x_3$ $x_4$ | $C_1 = [(x_1 - x_2) = 2]$ $b_1$=1 $a'_{1,1}$=2 $a'_{1,2}$=3 $b_{1,j}$ $v_{1,j,1}$ $v_{1,j,2}$ | $C_2 = [x_3 \leq 4]$ $b_2$=1 $a'_{2,1}$=4 $b_{2,j}$ $v_{2,j,1}$ | $C_3 = [x_{k_1} = x_{k_2}]$ $b_3$=−1 $a'_{3,1}$=1 $a'_{3,2}$=1 $b_{3,j}$ $v_{3,j,1}$ $v_{3,j,2}$ | row constraint satisfaction |
|---|---|---|---|---|---|---|
| positive entries | 1 | 4 2 5 **1** | 1 4 2 | 0 5 | 1 0 0 | `true` |
| | 2 | 3 4 4 **1** | 0 3 4 | 1 4 | 1 0 0 | `true` |
| | 3 | 1 1 3 **1** | 0 1 1 | 1 3 | 1 0 0 | `true` |
| | 4 | 3 1 5 **1** | 1 3 1 | 0 5 | 1 0 0 | `true` |
| | 5 | 4 1 2 **1** | 0 4 1 | 1 2 | 1 0 0 | `true` |
| negative entries | 6 | 2 4 5 **0** | 0 2 4 | 0 5 | 1 0 0 | `true` |
| | 7 | 4 1 5 **0** | 0 4 1 | 0 5 | 1 0 0 | `true` |
| | 8 | 4 3 5 **0** | 0 4 3 | 0 5 | 1 0 0 | `true` |
| | 9 | 3 5 5 **0** | 0 3 5 | 0 5 | 1 0 0 | `true` |

**Example 6.2.2** (Continuation of Example 6.2.1)**.** *Table 6.3 summarises the acquisition of the BAE* $x_4 = [(x_1 - x_2) = 2] \vee [x_3 \leq 4]$ *from the table* tab[1..9, 1..4]*: it provides the main variables introduced by the core model. First, note that only conditions* $C_1$ *and* $C_2$ *are selected, as* $b_3 = -1$*. For the first positive entry (i.e.* $j = 1$*) and the first negative entry (i.e.* $j = 6$*), we now show that the corresponding row constraints described in Table 6.2 are true:*

— *As row* 1 *is a positive entry, we post the constraint* $[b_1 = b_{1,1}] \vee [b_2 = b_{2,1}] \vee [b_3 = b_{3,1}]$ *which is true as* $b_1 = b_{1,1}$ *holds.*

— *As row* 6 *is a negative entry, we post the constraint* $\textsc{table}(\langle (b_d, b_{d,6}) \rangle, \mathcal{T})$*, with* $\mathcal{T} = \langle (-1, 0), (-1, 1), (0, 1), (1, 0) \rangle$*, for each condition* $C_d$ *(*$d \in \{1, 2, 3\}$*). All three constraints hold for the sixth row.*

Although the previous model encodes all the necessary constraints, it has two main weaknesses. First, it generates a large number of symmetric solutions corresponding to certain possible permutations of arguments or conditions. Second, it lacks of global view as all row constraints are loosely coupled to the original table. The aim of the next section is to address these weaknesses.

## 6.3 Enhancing the core model

In this section, we describe three ways of strengthening the core model presented in Section 6.2.

### 6.3.1 Linking the number of conditions, their arity, and the number of attributes

We introduce the following constraints to explicitly restrict the potential combinations of unary, binary and ternary conditions to consider.

**Notation 2.** *Within the expression $f$ formed by $n_{\text{AC}}$ conditions, let $n_{\text{AC},k}$ denote the number of conditions mentioning $k$ attributes.*

Since we restrict the Boolean-arithmetic expression $f$ to at most three conditions, we state the constraints $n_{\text{AC}} = \sum_{k=1}^{3} n_{\text{AC},k}$ and $\sum_{k=1}^{3} k \cdot n_{\text{AC},k} = \sum_{i=2}^{c} o_i$, where $o_i$ is the number of occurrences of value $i$ in the variables $a'_{d,1}, a'_{d,2}, \ldots, a'_{d,\ell_d}$, as stated by the GCC constraint (6.3) of the core model. We now state the lower and upper bounds on the number of distinct attributes $c-1$ appearing in the expression $f$ wrt $n_{\text{AC},k}$ (with $k \in [1,3]$):

- $c - 1 \geq \max_{k=1}^{3} (k \cdot \min(1, n_{\text{AC},k}))$,
- $c - 1 \leq \sum_{k=1}^{3} (k \cdot n_{\text{AC},k})$.

### 6.3.2 Symmetry breaking

As a formula may involve commutative arithmetic operators whose arguments can be interchanged, and mention several occurrences of the same condition which can be swapped, we show how to restrict the search space for formulae.

#### 6.3.2.1 Commutative arithmetic operators

For each BAC $C_d \begin{pmatrix} a_{d,1},\ a_{d,2}, \\ c_{d,1}, \ldots, c_{d,\ell'_d} \end{pmatrix}$ (with $d \in [1,m]$) mentioning two attributes $a_{d,1}$ and $a_{d,2}$, as well as a commutative arithmetic operator such as $+$, min, or max, we order its arguments only when the condition is used, by posting a constraint of the form $b_d \neq -1 \Rightarrow a'_{d,1} < a'_{d,2}$ on its variables $a'_{d,1}$ and $a'_{d,2}$, i.e. we use a strict inequality as a same attribute is used at most once in a condition $C_d$.

#### 6.3.2.2 Conditions mentioning the same comparison and arithmetic operators

In case the same condition would occur several times in the expression $f$, positively or negatively, or with different attributes, we post symmetry-breaking constraints to prevent generating equivalent subexpressions. We order the list of potential BACs $C_1, C_2, \ldots, C_m$ so that conditions that use the same comparison operator $\leq$, $=$, $\geq$, $\in$, as well as the same arithmetic operator $+$, $-$, $\times$, $\lfloor\text{-}\rfloor$, $\lceil\text{-}\rceil$, mod, min, max are located consecutively. For each pair of consecutive conditions $C_d \begin{pmatrix} a_{d,1}, \ldots, a_{d,\ell_d}, \\ c_{d,1}, \ldots, c_{d,\ell'_d} \end{pmatrix}$, $C_{d+1} \begin{pmatrix} a_{d+1,1}, \ldots, a_{d+1,\ell_{d+1}}, \\ c_{d+1,1}, \ldots, c_{d+1,\ell'_{d+1}} \end{pmatrix}$ (with $d \in [1, m-1]$) using the same comparison and arithmetic operators, we enforce the following symmetry-breaking constraint.

The idea is to impose a strict decreasing lexicographic ordering constraint (SLOC) between the variables of such consecutive conditions $C_d$ and $C_{d+1}$.

**Example 6.3.1.** *Assume we have two consecutive conditions $C_d$ and $C_{d+1}$ corresponding to an equality constraint, i.e. the first condition of class B1 of Table 6.1. To avoid gen-*

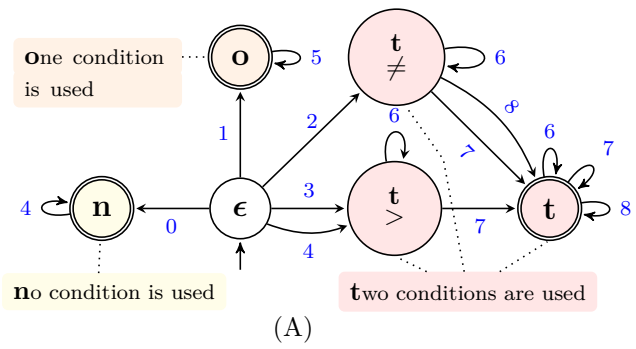
Table 6.4 – Definition of the input letters of the finite automaton depicted in Part (A) of Fig. 6.1 used for breaking symmetry between two consecutive conditions

| Input letter | Corresponding condition | Comment |
|---|---|---|
| $w_k = 0$ | $u_k = -1 \wedge v_k = -1$ | Both conditions are unused. |
| $w_k = 1$ | $(u_k = 0 \vee u_k = 1) \wedge v_k = -1$ | Only one condition is used. |
| $w_k = 2$ | $u_k = 1 \wedge v_k = 0$ | The 1st condition is used positively, and the negation of the 2nd condition is used. |
| $w_k = 3$ | $u_k = 0 \wedge v_k = 0$ | The negation of the 1st condition is used, and the negation of the 2nd condition is used. |
| $w_k = 4$ | $u_k = 1 \wedge v_k = 1$ | $k = 1$: both conditions are used positively, $k > 1$: attributes of both conditions are unused. |
| $w_k = 5$ | $u_k > 1 \wedge v_k = 1$ | $u_k$ is an attribute of the 1st condition, and $v_k$ an unused attribute of the 2nd condition, as the 2nd condition is unused. |
| $w_k = 6$ | $u_k > 1 \wedge v_k > 1 \wedge u_k = v_k$ | $u_k$ and $v_k$ are attributes of the two used conditions, such that $u_k = v_k$. |
| $w_k = 7$ | $u_k > 1 \wedge v_k > 1 \wedge u_k > v_k$ | $u_k$ and $v_k$ are attributes of the two used conditions, such that $u_k > v_k$. |
| $w_k = 8$ | $u_k > 1 \wedge v_k > 1 \wedge u_k < v_k$ | $u_k$ and $v_k$ are attributes of the two used conditions, such that $u_k < v_k$. |

*erating both the formula '$x = y \wedge y = z$' and the formula '$y = z \wedge x = y$', we order the conditions $C_d$ and $C_{d+1}$ wrt the variables involved in these two conditions: If we assume that the vector $(y, z)$ is lexicographically greater than the vector $(x, y)$, our symmetry-breaking constraint prevents us from generating the first formula '$x = y \wedge y = z$'.*

However, we need to consider the cases where these conditions are unused ($b_d = -1$, $b_{d+1} = -1$), negated ($b_d = 0$, $b_{d+1} = 0$) or positively used ($b_d = 1$, $b_{d+1} = 1$). We use the following idea to adapt the SLOC to our context: a SLOC can be described as a finite automaton whose input alphabet consists of letters that pairwise compare the k-th components of two vectors [169]. We compare the vectors $\overrightarrow{U} = (b_d, a'_{d,1}, a'_{d,2}, \ldots, a'_{d,\ell_d}) = (u_1, u_2, \ldots, u_{\ell_d+1})$ and $\overrightarrow{V} = (b_{d+1}, a'_{d+1,1}, a'_{d+1,2}, \ldots, a'_{d+1,\ell_{d+1}}) = (v_1, v_2, \ldots, v_{\ell_d+1})$. Recall from Section 6.2.2 that $(i)$ depending on whether condition $C_d$ is unused, negated or used positively, $b_d$ will be set to $-1$, 0 or 1 respectively, and that $(ii)$ the variables $a'_{d,1}, a'_{d,2}, \ldots, a'_{d,\ell_d}$ are all in the range $[2, c]$ as we applied the offset $+1$. By pairwise comparing the $k$-th components of vectors $\overrightarrow{U}$ and $\overrightarrow{V}$ (with $k \in [1, \ell_d + 1]$) we create the following vector $\overrightarrow{W} = (w_1, w_2, \ldots w_{\ell_d+1})$, where each component is defined by one of the nine letters $0, 1, \ldots, 8$ described in Table 6.4.

We then force the components of vector $\overrightarrow{W}$ to be accepted by the finite automaton given in Fig. 6.1. The three accepting states labelled by **n**, **o**, and **t** respectively correspond to the fact that $(i)$ **n**one of the conditions $C_d$, $C_{d+1}$ is used, $(ii)$ **o**nly the first condition $C_d$ is used, and $(iii)$ the **t**wo conditions $C_d$, $C_{d+1}$ are both used. The outgoing transitions from state $\boldsymbol{\epsilon}$ to states $\overset{\mathbf{t}}{\neq}$ and $\overset{\mathbf{t}}{>}$ enforce that, when using a condition and its negated form,




(A)

| $\overrightarrow{U}$ | $\overrightarrow{V}$ | $\overrightarrow{W}$ | Condition(s) |
|---|---|---|---|
| $(-1, 1)$ | $(-1, 1)$ | $(0, 4)$ | none |
| $(0, 2)$ | $(-1, 1)$ | $(1, 5)$ | $\neg C_d(1)$ |
| $(0, 3)$ | $(-1, 1)$ | $(1, 5)$ | $\neg C_d(2)$ |
| $(0, 3)$ | $(0, 2)$ | $(2, 7)$ | $\neg C_d(2)$ $\neg C_{d+1}(1)$ |
| $(1, 2)$ | $(-1, 1)$ | $(1, 5)$ | $C_d(1)$ |
| $(1, 3)$ | $(-1, 1)$ | $(1, 5)$ | $C_d(2)$ |
| $(1, 3)$ | $(0, 2)$ | $(2, 7)$ | $C_d(2)$ $\neg C_{d+1}(1)$ |
| $(1, 3)$ | $(1, 2)$ | $(4, 7)$ | $C_d(2)$ $C_{d+1}(1)$ |

(B)

Figure 6.1 – (A) Finite automaton for breaking symmetries between two consecutive conditions $C_d$, $C_{d+1}$ sharing the same comparison and arithmetic operators, where accepting states are denoted by a double circle; (B) Examples of vectors $\overrightarrow{U}$, $\overrightarrow{V}$, $\overrightarrow{W}$ and corresponding used conditions with their arguments (each condition mentions one single attribute).

the negated form is located in the second position. The two outgoing transitions of state $\overset{\mathbf{t}}{>}$ ensure that the arguments of the first used condition are lexicographically strictly greater than the arguments of the second condition, while the two outgoing transitions of state $\overset{\mathbf{t}}{\neq}$ force the two conditions not to use the same arguments.

**Example 6.3.2.** *We illustrate how the automaton of Figure 6.1 prevents generating the first formula 'x = y ∧ y = z' shown in Example 6.3.1. Without loss of generality, we assume that z is lexicographically greater than y, and that y is lexicographically greater than x. As both conditions 'x = y' and 'y = z' are used positively, their respective vectors* $\overrightarrow{U}$ *and* $\overrightarrow{V}$ *are respectively equal to* $(1, x, y)$ *and to* $(1, y, z)$*. Then, by pairwise comparing the three components of vectors* $\overrightarrow{U}$ *and* $\overrightarrow{V}$*, the corresponding vector* $\overrightarrow{W}$ *is equal to* $(4, 8, 8)$*, i.e.* 4 *as both conditions are used positively,* 8 *as x and y are attributes such that x is lexicographically less than y,* 8 *as y and z are attributes such that y is lexicographically less than z. As the finite automaton of Figure 6.1 rejects the sequence* $(4, 8, 8)$*, the formula 'x = y ∧ y = z' will not be generated.*

### 6.3.3 Pre-computing the combinations of possible values of the coefficients of a condition

Most BACs $C_d \begin{pmatrix} a_{d,1}, \ldots, a_{d,\ell_d}, \\ c_{d,1}, \ldots, c_{d,\ell'_d} \end{pmatrix}$ can be presented as a comparison of the form $C'_d(P) \lozenge c_{d,\ell'_d}$ (with $\lozenge \in \{\leq, =, \geq\}$), where $C'_d(P)$ is an arithmetic expression parameterised by $P = \begin{pmatrix} a'_{d,1}, \ldots, a'_{d,\ell_d}, \\ c_{d,1}, \ldots, c_{d,\ell'_d-1} \end{pmatrix}$. Such BACs in a Boolean formula $f$ must not be equivalent to `true` or `false`, as otherwise they could be simplified away from $f$. We also want to avoid generating a condition involving an inequality when an equality would suffice. For this purpose we proceed as follows.

— For each possible combination of values $p$ of parameter $P$ wrt the potential values of $a'_{d,1}, \ldots, a'_{d,\ell_d}, c_{d,1}, \ldots, c_{d,\ell'_d-1}$, we compute the feasible values of $C'_d(p)$ wrt all the table entries of tab$[1..r, 1..c]$. We denote by $\mathcal{V}_{d,p}$ such sets.

— Then, depending on the comparison operator $\lozenge$ used in condition $C_d$, we derive for each combination of values $p$ of parameter $P$, the set of values of coefficient $c_{d,\ell'_d}$ which does not make condition $C_d$ always true or always false. We denote such sets as $\mathcal{V}^{\lozenge}_{d,p}$. They are obtained from the sets $\mathcal{V}_{d,p}$ in the following way.

  — [$\lozenge$ is '=']: when the coefficient $c_{d,\ell'_d}$ is assigned a value outside $\mathcal{V}_{d,p}$ the condition

Table 6.5 – Example table for pre-computing the possible values of the coefficients for conditions $C_1$ and $C_2$

| $x_1$ | $x_2$ | $x_c$ | $[x_1 - x_2]$ | $[x_2 - x_1]$ | $[x_1 \bmod 3]$ | $[x_2 \bmod 3]$ |
|---|---|---|---|---|---|---|
| 1 | 2 | 0 | −1 | 1 | 1 | 2 |
| 2 | 1 | 0 | 1 | −1 | 2 | 1 |
| 1 | 2 | 1 | −1 | 1 | 1 | 2 |
| 1 | 3 | 1 | −2 | 2 | 1 | 0 |
| 1 | 4 | 1 | −3 | 3 | 1 | 1 |

$C_d$ would always be false; if the cardinality of $\mathcal{V}_{d,p}$ is 1 then $\mathcal{V}^{\Diamond}_{d,p} = \emptyset$ (i.e. if there is only one value the condition would always be true), otherwise $\mathcal{V}^{\Diamond}_{d,p} = \mathcal{V}_{d,p}$.

— [$\Diamond$ is '$\leq$' or '$\geq$']: let $\alpha$ and $\omega$ respectively be the smallest and the largest value of the set $\mathcal{V}_{d,p}$; then $\mathcal{V}^{\Diamond}_{d,p} = \mathcal{V}_{d,p} \setminus \{\alpha, \omega\}$. The intuition for $\Diamond =$'$\leq$' is as follows: if we keep $\alpha$ then $\Diamond =$'$\leq$' is equivalent to $\Diamond =$'='; if we keep $\omega$ the condition will always be true. For '$\geq$' the intuition is symmetrical.

— We may further reduce the set $\mathcal{V}^{\Diamond}_{d,p}$ by considering the aggregator $g$. First, for each possible combination of values $p$ of parameter $P$, we compute the feasible values of $C'_d(p)$ wrt all the positive (resp. negative) table entries of tab$[1..r, 1..c]$. We denote by $\mathcal{V}^{\text{pos}}_{d,p}$ (resp. $\mathcal{V}^{\text{neg}}_{d,p}$) such sets. From these sets, we compute the further restricted set $\mathcal{V}^{\Diamond,g}_{d,p}$ as follows:

— [$g$ is '$\wedge$']: if $\Diamond$ is '=' then $\mathcal{V}^{\Diamond,g}_{d,p} = \mathcal{V}^{\text{pos}}_{d,p}$ else $\mathcal{V}^{\Diamond,g}_{d,p} = \mathcal{V}^{\text{pos}}_{d,p} \cap \mathcal{V}^{\Diamond}_{d,p}$,

— [$g \in \{$' $\vee$ ',' + '$\}$]: $\mathcal{V}^{\Diamond,g}_{d,p} = \mathcal{V}^{\Diamond}_{d,p} \setminus \mathcal{V}^{\text{neg}}_{d,p}$,

— [$g \notin \{$' $\wedge$ ',' $\vee$ ',' + '$\}$]:$\mathcal{V}^{\Diamond,g}_{d,p} = \mathcal{V}^{\Diamond}_{d,p}$.

— Finally, we set up the table constraint $\textsc{table}\left(\left\langle \begin{array}{c} a'_{d,1}, \ldots, a'_{d,\ell_d}, \\ c_{d,1}, \ldots, c_{d,\ell'_d} \end{array} \right\rangle, \mathcal{S}\right)$ where $\mathcal{S}$ corresponds to the union of Cartesian products $\cup_{p \in P}(p \times \mathcal{V}^{\Diamond,g}_{d,p})$.

**Example 6.3.3.** *To illustrate the process, consider Table 6.5. There are two input columns 1 and 2 and the output column c. Consider the two conditions* $C_1 = [a_{1,1} - a_{1,2} = c_{1,1}]$ *and* $C_2 = [a_{2,1} \bmod c_{2,1} \geq c_{2,2}]$.

- *For* $C_1$ *we have only two options for* $p = \{a_{1,1}, a_{1,2}\}$*, namely:*

*1)* $p = \{1, 2\}$: $\left\{ \begin{array}{l} \mathcal{V}_{1,p} = \{-3, -2, -1, 1\},\ \mathcal{V}^{=}_{1,p} = \mathcal{V}_{1,p}, \\ \mathcal{V}^{=,\text{'}\wedge\text{'}}_{1,p} = \{-3, -2, -1\},\ \mathcal{V}^{=,\text{'}\vee\text{'}}_{1,p} = \mathcal{V}^{=}_{1,p} \setminus \{-1, 1\} = \{-3, -2\}. \end{array} \right.$

*2)* $p = \{2, 1\}$: $\left\{ \begin{array}{l} \mathcal{V}_{1,p} = \{-1, 1, 2, 3\},\ \mathcal{V}^{=}_{1,p} = \mathcal{V}_{1,p}, \\ \mathcal{V}^{=,\text{'}\wedge\text{'}}_{1,p} = \{1, 2, 3\}, \quad \mathcal{V}^{=,\text{'}\vee\text{'}}_{1,p} = \mathcal{V}^{=}_{1,p} \setminus \{-1, 1\} = \{2, 3\}. \end{array} \right.$

• *For $C_2$ we need to enumerate on $c_{2,1}$. w.l.o.g., we only consider the case $c_{2,1} = 3$. In this context, the options for $p = \{a_{2,1}, c_{2,1}\}$ are:*

*1)* $p = \{1, 3\}$*:* $\mathcal{V}_{2,p} = \{1, 2\}$, $\alpha = 1$, $\omega = 2$, $\mathcal{V}^{\geq}_{2,p} = \mathcal{V}_{2,p} \setminus \alpha, \omega = \emptyset$*, i.e. this set of options for this condition is not considered any further.*

*2)* $p = \{2, 3\}$*:* $\begin{cases} \mathcal{V}_{2,p} = \{0, 1, 2\},\ \alpha = 0, \omega = 2,\ \mathcal{V}^{\geq}_{1,p} = \mathcal{V}_{2,p} \setminus \{\alpha, \omega\} = \{1\}, \\ \mathcal{V}^{\text{pos}}_{2,p} = \{1\},\ \mathcal{V}^{\text{neg}}_{2,p} = \{1, 2\}, \\ \mathcal{V}^{\geq, \text{`}\wedge\text{'}}_{2,p} = \mathcal{V}^{\text{pos}}_{2,p} \cap \mathcal{V}^{\geq}_{2,p} = \{1\}, \mathcal{V}^{\geq, \text{`}\vee\text{'}}_{2,p} = \mathcal{V}^{\geq}_{2,p} \setminus \mathcal{V}^{\text{neg}}_{2,p} = \emptyset. \end{cases}$

## 6.4 Defining, acquiring, and applying anti-rewriting constraints to avoid generating simplifiable Boolean-arithmetic expressions

In some cases, the generated BAEs can be simplified, as it was done for example in [176], or when a condition is always true or always false. Also, we already handle this case in Section 6.3.3, there are many other cases where a formula could be possibly simplified, as we allow combining a wide range of conditions. For example, we should not generate a BAE like $(x_1 + x_3 = 5) \wedge (x_1 \leq x_2) \wedge (x_2 \leq x_1)$, as it can be simplified to $(x_1 + x_3 = 5) \wedge (x_1 = x_2)$. Our approach is to create and add a set of constraints that prevent the generation of simplifiable expressions from the very beginning of the acquisition process. We will refer to these constraints as *anti-rewriting constraints*.

Since there are a huge number of possible anti-rewriting constraints, we synthesise a database of anti-rewriting constraints once and for all, using a CP approach to capture such constraints systematically.

The main idea of this sectnio is to identify anti-rewriting constraints that limit the creation of a pair of conditions, which can then be simplified. To do this, we will identify the hypotheses under which a pair of conditions is simplifiable: these hypotheses concern both (*i*) the way in which two conditions share certain variables (because otherwise the two conditions would be independent), and (*ii*) the restrictions on the coefficients of certain variables in these conditions, i.e. restrictions on the domain of a coefficient, or constraints between two coefficients.

We first describe in Section 6.4.1 the type of hypotheses we use on the coefficients of the conditions of an anti-rewriting constraint. We then introduce in Section 6.4.2 three families of anti-rewriting constraints, and show in Section 6.4.3 how to use such anti-rewriting

constraints to restrict the search space. For each family of anti-rewriting constraints, we provide the algorithms used to acquire the corresponding anti-rewriting constraints in Section 6.4.4, and the way to select the most general anti-rewriting constraints in Section 6.4.5. Finally, we describe the database of anti-rewriting constraints in Section 6.4.6.

### 6.4.1 Hypothesis description for a pair of conditions of an anti-rewriting constraint

The three families of anti-rewriting constraints that prevent the generation of a simplifiable term of the form $C_1$ or $C_1\ g\ C_2$ all require a hypothesis on the coefficients of condition $C_1$ and possibly $C_2$. In the simplest case, a hypothesis can be the Boolean constant true, which means that the hypothesis imposes no restrictions on the coefficients of conditions $C_1$ and $C_2$. Otherwise, in the general case, to define a hypothesis, we introduce the notion of a comparison function on the coefficients of $C_1$ and $C_2$.

**Definition 7.** *Given condition* $C_1\begin{pmatrix} a_{1,1},\ldots,a_{1,\ell_1}, \\ c_{1,1},\ldots,c_{1,\ell'_1} \end{pmatrix}$ *and, possibly, condition* $C_2\begin{pmatrix} a_{2,1},\ldots,a_{2,\ell_2}, \\ c_{2,1},\ldots,c_{2,\ell'_2} \end{pmatrix}$*, the domain of a* comparison function *is one of the following expressions* $c_{1,i}\ cmp\ cst$*,* $c_{2,j}\ cmp\ cst$*,* $c_{1,i}+cst_1\ cmp\ c_{2,j}+cst_2$*, or* $cst_3\cdot c_{1,i}\ cmp\ cst_4\cdot c_{2,j}$*, (with* $i\in[1,\ell'_1]$*,* $j\in[1,\ell'_2]$ *and* $cmp\in\{=,\leq,\geq,\neq\}$*), where* $cst$*,* $cst_1$*,* $cst_2$*,* $cst_3$*, and* $cst_4$ *are natural numbers in* $[0,2]$*, where* $cst_1\cdot cst_2=0$*, and where* $cst_3\cdot cst_4=\max(cst_3,cst_4)>0$*, and the codomain is the set* $\{\texttt{true},\texttt{false}\}$*.*

**Definition 8.** *Given condition* $C_1\begin{pmatrix} a_{1,1},\ldots,a_{1,\ell_1}, \\ c_{1,1},\ldots,c_{1,\ell'_1} \end{pmatrix}$ *and, possibly, condition* $C_2\begin{pmatrix} a_{2,1},\ldots,a_{2,\ell_2}, \\ c_{2,1},\ldots,c_{2,\ell'_2} \end{pmatrix}$*, a* hypothesis $\hbar$ *is one of the following expressions* (*i*) *the Boolean constant* `true` *if there is no restriction on the coefficients of* $C_1$ *and* $C_2$*,* (*ii*) *one single comparison function or, when we have two conditions,* (*iii*) *the conjunction or the negation of the conjunction of two comparison functions, provided that one of the conditions* $C_1$ *or* $C_2$ *mentions at least one coefficient and the other includes at least two coefficients, i.e.* $\min(\ell'_1,\ell'_2)\geq 1$ *and* $\max(\ell'_1,\ell'_2)\geq 2$*.*

We call a hypothesis corresponding to the case (iii) of Definition 8 *compound*, and *simple* otherwise.

**Definition 9.** *Given condition* $C_k\left(\begin{array}{c} a_{k,1},\ldots,a_{k,\ell_k},\\ c_{k,1},\ldots,c_{k,\ell'_k}\end{array}\right)$, $k \in \{1,2\}$, *and condition* $C_3\left(\begin{array}{c} a_{3,1},\ldots,a_{3,\ell_3},\\ c_{3,1},\ldots,c_{3,\ell'_3}\end{array}\right)$, *a* link-hypothesis $\hbar_{k,3}$ *is either* (*i*) *the Boolean constant* `true`, (*ii*) *an expression* $c_{k,i}+cst_1 = c_{3,j}+cst_2$ (*with* $i \in [1,\ell'_k]$, $j \in [1,\ell'_3]$), *where* $cst_1$ *and* $cst_2$ *are natural numbers in* $[0,1]$, *where* $cst_1 \cdot cst_2 = 0$.

**Example 6.4.1.** *To illustrate Definitions (7)–(9) consider the two conditions '*$(x \bmod c_{1,1}) = c_{1,2}$*', and '*$\neg(x \in [c_{2,1}, c_{2,2}])$*'. The hypothesis '*$(c_{1,2}+1) \leq c_{2,1} \wedge 2\cdot c_{1,2} \geq c_{2,2}$*' matches case (iii) of Definition (8), as we have a conjunction of two comparison functions, where the two conditions use two coefficients each. This hypothesis will be used in the anti-rewriting constraint presented in Example 6.4.2.*

The next section shows how to exploit Definitions (7)–(9) to define three families of anti-rewriting constraints, while Section 6.4.3 details how such anti-rewriting constraints will be used.

### 6.4.2 Defining families of anti-rewriting constraints

As for the problem described in Section 6.2.1, we consider a table tab$[1..r, 1..c]$ of integer values, and a Boolean-arithmetic expression $f$ satisfying (6.1). As in Section 6.2.2.3, we also assume that each condition in $f$ can be made `true` or `false` by a combination of values in the domains of its variables. In addition, we suppose that $f$ uses the aggregation operator $g$, and one or two Boolean-arithmetic conditions $C_1^{\mathtt{ar}}\left(\begin{array}{c} a_{1,1},\ldots,a_{1,\ell_1},\\ c_{1,1},\ldots,c_{1,\ell'_1}\end{array}\right)$ (resp. $C_2^{\mathtt{ar}}\left(\begin{array}{c} a_{2,1},\ldots,a_{2,\ell_2},\\ c_{2,1},\ldots,c_{2,\ell'_2}\end{array}\right)$), where $A_1 = \{a_{1,1},\ldots,a_{1,\ell_1}\}$ (resp. $A_2 = \{a_{2,1},\ldots,a_{2,\ell_2}\}$) denotes the set of columns of table tab$[1..r, 1..c]$ used by $C_1^{\mathtt{ar}}$ (resp. $C_2^{\mathtt{ar}}$).

We define three families of anti-rewriting constraints, preventing generating a Boolean-arithmetic expression in which conditions $C_1^{\mathtt{ar}}$ (resp. $C_1^{\mathtt{ar}}$ and $C_2^{\mathtt{ar}}$) can be simplified if certain requirements are met. We have the following two types of needs for our three families:

— The first requirement is that, if we use both conditions $C_1^{\mathtt{ar}}$ and $C_2^{\mathtt{ar}}$ in $f$, conditions $C_1^{\mathtt{ar}}$ and $C_2^{\mathtt{ar}}$ are tightly coupled, since it makes little sense to find anti-rewriting constraints between almost independent conditions. To do this, we assume that

one of the conditions $C_1^{\mathtt{ar}}$ or $C_2^{\mathtt{ar}}$ refers to all the columns in the other condition (i.e. $A_1 \subseteq A_2$ or $A_2 \subseteq A_1$), and we denote the set of columns used by $A$ as $A_1 \cup A_2$.

— The second requirement is a hypothesis $\hbar$ imposed on the coefficients $c_{1,1}, \ldots, c_{1,\ell'_1}$ of $C_1^{\mathtt{ar}}$ if only one condition is used, or on the coefficients $c_{1,1}, \ldots, c_{1,\ell'_1}$ of $C_1^{\mathtt{ar}}$ and the coefficients $c_{2,1}, \ldots, c_{2,\ell'_2}$ of $C_2^{\mathtt{ar}}$ if both conditions are used in $f$.

We now define each family of anti-rewriting constraints as a condition involving (*i*) the aggregation operator $g$, (*ii*) the conditions $C_1^{\mathtt{ar}}$ and possibly $C_2^{\mathtt{ar}}$, and (*iii*) the hypothesis $\hbar$ on the coefficients of $C_1^{\mathtt{ar}}$ and $C_2^{\mathtt{ar}}$. For each family, we provide the condition that characterises it, the type of simplification it prevents, and an illustrative example.

**Definition 10.** *A Family 1 anti-rewriting constraint consists of an aggregator $g \in \{\wedge, \vee\}$, two non-equivalent conditions $C_1^{\mathtt{ar}}$ and $C_2^{\mathtt{ar}}$, and a hypothesis $\hbar$ on the coefficients of $C_1^{\mathtt{ar}}$ and $C_2^{\mathtt{ar}}$ satisfying the following statement:*

$$\hbar \Rightarrow ((C_1^{\mathtt{ar}} \Rightarrow C_2^{\mathtt{ar}}) \oplus (C_2^{\mathtt{ar}} \Rightarrow C_1^{\mathtt{ar}})) \tag{6.6}$$

If, assuming the hypothesis $\hbar$ holds, one and exactly one of the two conditions $C_1^{\mathtt{ar}}$ or $C_2^{\mathtt{ar}}$ is superseded by the other, then the subexpression $C_1^{\mathtt{ar}} g\, C_2^{\mathtt{ar}}$ would be simplified by using the strongest of the two conditions when $g =$‘$\wedge$’ (i.e. the condition on the left-hand side of the implication), or by employing the weakest condition when $g =$‘$\vee$’ (i.e. the condition on the right-hand side of the implication).

**Example 6.4.2.** *An example of an anti-rewriting constraint for Family 1 for nonnegative $x$, when $g =$‘$\wedge$’, is given the two conditions ‘$(x \bmod c_{1,1}) = c_{1,2}$’, and ‘$\neg(x \in [c_{2,1}, c_{2,2}])$’, and by the hypothesis ‘$(c_{1,2} + 1) \leq c_{2,1} \wedge 2 \cdot c_{1,2} \geq c_{2,2}$’.*

*This is because the conjunction ‘$(x \bmod c_{1,1}) = c_{1,2} \wedge \neg(x \in [c_{2,1}, c_{2,2}])$’ of the two conditions can be rewritten as ‘$(x \bmod c_{1,1}) = c_{1,2}$’ when the hypothesis ‘$(c_{1,2} + 1) \leq c_{2,1} \wedge 2 \cdot c_{1,2} \geq c_{2,2}$’ holds.*

*To prove this statement, we will prove that if the hypothesis ‘$(c_{1,2}+1) \leq c_{2,1} \wedge 2 \cdot c_{1,2} \geq c_{2,2}$’ is true then the condition ‘$(x \bmod c_{1,1}) = c_{1,2}$ implies the condition ‘$\neg(x \in [c_{2,1}, c_{2,2}])$’ but not the other way around:*

1. *if the condition ‘$(x \bmod c_{1,1}) = c_{1,2}$’ is false, then the condition ‘$\neg(x \in [c_{2,1}, c_{2,2}])$’ can be both positive or negative for the implication to be true. Thus we only concentrate on cases where the condition ‘$(x \bmod c_{1,1}) = c_{1,2}$’ is true.*

2. *For the condition '$(x \bmod c_{1,1}) = c_{1,2}$' to be true $x$ must be equal to $c_{1,2} + k \cdot c_{1,1}$, where $k \geq 0$. We must remember that $c_{1,1} > c_{1,2}$ as discussed in Section 6.2.2. It means that we can substitute $c_{1,1}$ as $c_{1,2} + cst$, where $cst \geq 1$. Thus $x = c_{1,2} + k \cdot c_{1,1}$ can be rewritten as $x = (k+1) \cdot c_{1,2} + k \cdot cst$.*

3. *If the hypothesis '$(c_{1,2}+1) \leq c_{2,1} \wedge 2 \cdot c_{1,2} \geq c_{2,2}$' is true then we consider three cases:*
   - *When $k = 0$, then $x = c_{1,2}$. Since $(c_{1,2} + 1) \leq c_{2,1}$ then the condition '$\neg(x \in [c_{2,1}, c_{2,2}])$' is true, as $x \leq c_{2,1} - 1$.*
   - *When $k = 1$, then $x = 2 \cdot c_{1,2} + cst$. As $2 \cdot c_{1,2} \geq c_{2,2}$ then $x \geq c_{2,2} + cst$ or, because $cst \geq 1$, $x \geq c_{2,2} + 1$ for which the condition '$\neg(x \in [c_{2,1}, c_{2,2}])$' will be true.*
   - *for $k \geq 2$, the proof is trivial.*

   *This shows that the condition '$(x \bmod c_{1,1}) = c_{1,2}$ implies the condition '$\neg(x \in [c_{2,1}, c_{2,2}])$'.*

4. *To show that the condition '$\neg(x \in [c_{2,1}, c_{2,2}])$' does not imply the condition '$(x \bmod c_{1,1}) = c_{1,2}$' we must remember that $c_{1,1} \geq 2$ as discussed in Section 6.2.2. Thus, if for $x$ the condition '$(x \bmod c_{1,1}) = c_{1,2}$' is true then for $x + 1$ the condition '$((x + 1) \bmod c_{1,1}) = c_{1,2}$' is false. When $k \geq 1$, as shown in the previous step, both conditions '$(x \bmod c_{1,1}) = c_{1,2}$' and '$\neg(x \in [c_{2,1}, c_{2,2}])$' are true when $x = (k+1) \cdot c_{1,2} + k \cdot cst$. If we take $x = 2 \cdot c_{1,2} + cst + 1$ then the condition '$(x \bmod c_{1,1}) = c_{1,2}$' is false while the condition '$\neg(x \in [c_{2,1}, c_{2,2}])$' remains true. Thus, the condition '$\neg(x \in [c_{2,1}, c_{2,2}])$' cannot imply the condition '$(x \bmod c_{1,1}) = c_{1,2}$'.*

**Definition 11.** *A Family 2 anti-rewriting constraint consists of an aggregator $g \in \{\vee, \wedge, \oplus, \mathrm{eq}\}$, two conditions $C_1^{\mathtt{ar}}$ and $C_2^{\mathtt{ar}}$, and a hypothesis $\hbar$ on the coefficients of $C_1^{\mathtt{ar}}$ and $C_2^{\mathtt{ar}}$ satisfying the following statement:*

$$\hbar \Rightarrow (C_1^{\mathtt{ar}} \ g \ C_2^{\mathtt{ar}}) \vee \neg(C_1^{\mathtt{ar}} \ g \ C_2^{\mathtt{ar}}) \tag{6.7}$$

When $g \in \{\vee, \wedge, \oplus, \mathrm{eq}\}$, assuming the hypothesis $\hbar$ holds, the subexpression $C_1^{\mathtt{ar}} g \, C_2^{\mathtt{ar}}$ is either always true or always false, we should not generate that subexpression.

**Example 6.4.3.** *An example of an anti-rewriting constraint for Family 2 when $g =$'$\wedge$' is given by the two conditions '$x \geq c_{1,1}$', and '$x \leq c_{2,1}$', and by the hypothesis '$c_{1,1} > c_{2,1}$'. In fact, if $c_{1,1} > c_{2,1}$ holds, then the conjunction $x \geq c_{1,1} \wedge x \leq c_{2,1}$ is always false.*

To describe the third family of anti-rewriting constraints, we must first introduce the notion of the cost of a condition, which heuristically reflects the complexity of the condition, i.e. the lower the cost of a condition, the simpler the condition. This is because, if some conditions are equivalent to others, we want to keep only the simplest condition.

**Definition 12.** *The cost of a condition $C$, denoted by $cost(C)$, is defined in the second column of Table 6.1. The cost of the negation of a condition $C$ is defined as $cost(C) + 1$.*

We also need to heuristically compare the complexity of an expression of the form $C_1^{\mathtt{ar}}\ g\ C_2^{\mathtt{ar}}$, where $C_1^{\mathtt{ar}}$ and $C_2^{\mathtt{ar}}$ are two conditions linked by an aggregator $g$, with a single condition $C_3^{\mathtt{ar}}$. To do this, we define two vectors which are then compared lexicographically.

**Definition 13.** *We associate to $C_1^{\mathtt{ar}}\ g\ C_2^{\mathtt{ar}}$ and to $C_3^{\mathtt{ar}}$ two vectors $v_{C_1^{\mathtt{ar}},g,C_2^{\mathtt{ar}}} = \langle a_1, a_2, a_3\rangle$ and $v_{C_3^{\mathtt{ar}}} = \langle b_1, b_2, b_3\rangle$ where:*

- *$a_1$ is the number of attributes used in $C_1^{\mathtt{ar}}$ and $C_2^{\mathtt{ar}}$, while $b_1$ is the number of attributes used in $C_3^{\mathtt{ar}}$.*
- *$a_2$ is defined as $cost(C_1^{\mathtt{ar}}) + c_g + cost(C_2^{\mathtt{ar}})$ (with $c_\wedge = 0$, $c_\vee = 1$, $c_\oplus = 2$, $c_g = 3$ if $g \notin \{\wedge, \vee, \oplus\}$), while $b_2$ is defined as $cost(C_3^{\mathtt{ar}})$.*
- *$a_3$ and $b_3$ correspond to the number of conditions of $C_1^{\mathtt{ar}}\ g\ C_2^{\mathtt{ar}}$ and $C_3^{\mathtt{ar}}$, namely 2 for $C_1^{\mathtt{ar}}\ g\ C_2^{\mathtt{ar}}$ and 1 for $C_3^{\mathtt{ar}}$.*

**Definition 14.** *A Family 3a anti-rewriting constraint consists of a single condition $C_1^{\mathtt{ar}}$, a hypothesis $\hbar$ on the coefficients of $C_1^{\mathtt{ar}}$, a condition $C_3^{\mathtt{ar}}$ referring to a subset of the columns of $C_1^{\mathtt{ar}}$, and a so-called link-hypothesis $\hbar_{1,3}$, between a coefficient of condition $C_1^{\mathtt{ar}}$ and a coefficient of condition $C_3^{\mathtt{ar}}$, satisfying the following statement:*

$$((\hbar \wedge \hbar_{1,3}) \Rightarrow (C_1^{\mathtt{ar}} \Leftrightarrow C_3^{\mathtt{ar}})) \wedge (cost(C_1^{\mathtt{ar}}) > cost(C_3^{\mathtt{ar}})) \tag{6.8}$$

If, assuming that the hypotheses $\hbar$ and $\hbar_{1,3}$ hold, the expressions $C_1^{\mathtt{ar}}$ and $C_3^{\mathtt{ar}}$ are equivalent and the cost of $C_1^{\mathtt{ar}}$ is greater than the cost of $C_3^{\mathtt{ar}}$, we should not generate the expression $C_1^{\mathtt{ar}}$, since $C_1^{\mathtt{ar}}$ could be rewritten as the simpler expression $C_3^{\mathtt{ar}}$. As shown by Example 6.4.4, Definition 14 uses a hypothesis $\hbar_{1,3}$ to get a tighter connection between the conditions $C_1^{\mathtt{ar}}$ and $C_3^{\mathtt{ar}}$, allowing an implication between the two parts of Formula 6.8.

**Example 6.4.4.** *Given $\hbar =$ '$c_{1,1} = 2$', $\hbar_{1,3} =$ '$c_{1,1} + 1 = c_{3,1}$', $x_1 > 0$ and $x_2 > 0$, the condition '$x_1 \cdot x_2 \leq c_{1,1}$' is equivalent to the simpler condition '$x_1 + x_2 \leq c_{3,1}$', i.e. '$(x_1 \cdot x_2 \leq 2) \Leftrightarrow (x_1 + x_2 \leq 3)$'; simpler as $cost($ '$x_1 \cdot x_2 \leq c_{1,1}$'$) = 5$ is greater than $cost($ '$x_1 + x_2 \leq c_{1,1} + 1$'$) = 3$.*

**Definition 15.** *A Family 3b anti-rewriting constraint consists of an aggregator* $g \in \{\vee, \wedge, \oplus, \text{eq}\}$*, two conditions* $C_1^{\texttt{ar}}$ *and* $C_2^{\texttt{ar}}$*, a hypothesis* $\hbar$ *on the coefficient* $C_1^{\texttt{ar}}$ *and* $C_2^{\texttt{ar}}$*, a condition* $C_3^{\texttt{ar}}$ *which is neither always true nor always false and referring to a subset of the columns of* $C_1^{\texttt{ar}}$ *or* $C_2^{\texttt{ar}}$*, a link-hypothesis* $\hbar_{1,3}$ *between a coefficient of condition* $C_1^{\texttt{ar}}$ *and a coefficient of condition* $C_3^{\texttt{ar}}$*, and a link-hypothesis* $\hbar_{2,3}$ *between a coefficient of condition* $C_2^{\texttt{ar}}$ *and a coefficient of condition* $C_3^{\texttt{ar}}$*, satisfying the following statement:*

$$(\hbar \wedge \hbar_{1,3} \wedge \hbar_{2,3}) \Rightarrow \begin{array}{l}((C_1^{\texttt{ar}}\ g\ C_2^{\texttt{ar}}) \Leftrightarrow C_3^{\texttt{ar}})) \wedge \\ (v_{C_1^{\texttt{ar}},g,C_2^{\texttt{ar}}} >_{\ell ex} v_{C_3^{\texttt{ar}}})\end{array} \tag{6.9}$$

If, assuming hypothesises $\hbar, \hbar_{1,3}$ and $\hbar_{2,3}$ hold, $C_1^{\texttt{ar}}\ g\ C_2^{\texttt{ar}}$ and $C_3^{\texttt{ar}}$ are equivalent, and the cost of $C_1^{\texttt{ar}}\ g\ C_2^{\texttt{ar}}$ is greater than the cost of $C_3^{\texttt{ar}}$, we should not generate the expression $C_1^{\texttt{ar}}\ g\ C_2^{\texttt{ar}}$, since $C_1^{\texttt{ar}}\ g\ C_2^{\texttt{ar}}$ could be rewritten as the simpler expression $C_3^{\texttt{ar}}$.

**Example 6.4.5.** *An example of an anti-rewriting constraint for Family 3 with no hypothesis at all, i.e.* $\hbar =$ `'true'`, $\hbar_{1,3} =$ `'true'` *and* $\hbar_{2,3} =$ `'true'`*, when* $g =$*'*$\wedge$*', is given by the two conditions '*$x_1 \leq x_2$*' and '*$x_1 \geq x_2$*'. This conjunction is equivalent to the condition '*$x_1 = x_2$*', where '*$x_1 = x_2$*' is simpler than '*$x_1 \leq x_2$*' and '*$x_1 \geq x_2$*' as the cost vector of* `'`$x_1 = x_2$`'` $= \langle 2, 0, 1\rangle$ *is lexicographically strictly less than the cost vector of* `'`$x_1 \leq x_2$`'` $\wedge$ `'`$x_1 = x_2$`'` $= \langle 2, 0+0+0, 2\rangle$.

**Example 6.4.6.** *An example of an anti-rewriting constraint for Family 3, when* $g =$*'*$\wedge$*', is given by the two conditions '*$x \geq c_{1,1}$*' and '*$x \leq c_{2,1}$*', and by the hypothesis '*$c_{1,1} = c_{2,1}$*'. In fact, if* $c_{1,1} = c_{2,1}$ *holds, then the conjunction '*$(x \geq c_{1,1}) \wedge (x \leq c_{2,1})$*' is equivalent to the condition '*$x = c_{1,1}$*', where '*$x = c_{1,1}$*' is simpler than '*$x \geq c_{1,1}$*' and '*$x \leq c_{2,1}$*' as the cost vector of* `'`$x = c_{1,1}$`'` $= \langle 1, 1, 1\rangle$ *is lexicographically strictly less than the cost vector of* `'`$(x \geq c_{1,1}) \wedge (x \leq c_{2,1})$`'` $= \langle 1, 1+0+1, 2\rangle$.

We perform a systematic search for anti-rewriting constraints for every pair of conditions we have in Table 6.1, retaining the most general rules as described in Section 6.4.5. All acquired anti-rewriting constraints are stored in a database which will later be consulted when acquiring BAE. For each identified anti-rewriting constraint, we store (*i*) the corresponding conditions $C_1^{\texttt{ar}}$ and, if used, $C_2^{\texttt{ar}}$, including whether they are negated or not, (*ii*) the attribute sets $A_1$ and, if used, $A_2$, (*iii*) the aggregator $g$, (*iv*) the hypothesis $h$, and (*v*) the set of matching attributes $\mathcal{M}$ that we now define. Note that we do not record the linked hypotheses attached to the anti-rewriting constraints of Family 3b, as they were only used to find the condition $C_3^{\texttt{ar}}$. Since the condition $C_3^{\texttt{ar}}$ is the simplified

form of the non-simplified expression $C_1^{\texttt{ar}}\ g\ C_2^{\texttt{ar}}$ specified by an anti-rewriting constraint of Family 3b, we do not store $C_3^{\texttt{ar}}$ either.

**Definition 16.** *Given an anti-rewriting constraint mentioning two conditions $C_1^{\texttt{ar}}$ and $C_2^{\texttt{ar}}$ (resp. one condition $C_1^{\texttt{ar}}$), we define the* matching set $\mathcal{M}$ *of that anti-rewriting constraint as the set of pairs $(t_1, t_2)$, where $t_1 \in [1, \ell_1]$ and $t_2 \in [1, \ell_2]$, for which $a_{1,t_1} = a_{2,t_2}$ (resp. the empty set).*

In other words, the matching set $\mathcal{M}$ establishes the correspondence between the shared attributes of conditions $C_1^{\texttt{ar}}$ and $C_2^{\texttt{ar}}$.

Table 6.6 contains several examples of acquired anti-rewriting constraints for aggregator $g$ in $\{\wedge, \vee\}$ and conditions 'A1', 'A2' and 'B1' from Table 6.1, assuming that each variable $x_i$ (with $i \in \{1, 2\}$) is greater than or equal to 0. More complex anti-rewriting constraints found by the system are given in Section 6.4.5.

Table 6.6 – Examples of simple anti-rewriting constraints acquired for some conditions in Table 6.1, where 'F' is a shortcut for 'Family'.

| $k$ | F | $C_1^{\texttt{ar}}$ | $C_2^{\texttt{ar}}$ | $A_1$ | $A_2$ | $g$ | $\hbar$ | $\mathcal{M}$ | Equivalent |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | $x_1 = c_{1,1}$ | $x_1 \leq c_{2,1}$ | $[x_1]$ | $[x_1]$ | $\wedge$ | $c_{1,1} \leq c_{2,1}$ | (1,1) | $x_1 = c_{1,1}$ |
| 2 | 1 | $x_1 = c_{1,1}$ | $x_1 \leq c_{2,1}$ | $[x_1]$ | $[x_1]$ | $\vee$ | $c_{1,1} \leq c_{2,1}$ | (1,1) | $x_1 \leq c_{2,1}$ |
| 3 | 1 | $x_1 = c_{1,1}$ | $x_1 \geq c_{2,1}$ | $[x_1]$ | $[x_1]$ | $\wedge$ | $c_{1,1} \geq c_{2,1}$ | (1,1) | $x_1 = c_{1,1}$ |
| 4 | 1 | $x_1 = c_{1,1}$ | $x_1 \geq c_{2,1}$ | $[x_1]$ | $[x_1]$ | $\vee$ | $c_{1,1} \geq c_{2,1}$ | (1,1) | $x_1 \geq c_{2,1}$ |
| 5 | 1 | $x_1 = x_2$ | $x_1 \leq x_2$ | $[x_1, x_2]$ | $[x_1, x_2]$ | $\wedge$ | `true` | (1,1), (2,2) | $x_1 = x_2$ |
| 6 | 1 | $x_1 = x_2$ | $x_1 \leq x_2$ | $[x_1, x_2]$ | $[x_1, x_2]$ | $\vee$ | `true` | (1,1), (2,2) | $x_1 \leq x_2$ |
| 7 | 1 | $x_1 = x_2$ | $x_2 \leq x_1$ | $[x_1, x_2]$ | $[x_2, x_1]$ | $\wedge$ | `true` | (1,2), (2,1) | $x_1 = x_2$ |
| 8 | 1 | $x_1 = x_2$ | $x_2 \leq x_1$ | $[x_1, x_2]$ | $[x_2, x_1]$ | $\vee$ | `true` | (1,2), (2,1) | $x_2 \leq x_1$ |
| 9 | 2 | $x_1 \leq c_{1,1}$ | $x_1 \geq c_{2,1}$ | $[x_1]$ | $[x_1]$ | $\wedge$ | $c_{1,1} + 1 \leq c_{2,1}$ | (1,1) | `false` |
| 10 | 2 | $x_1 \leq c_{1,1}$ | $x_1 \geq c_{2,1}$ | $[x_1]$ | $[x_1]$ | $\vee$ | $c_{2,1} \geq c_{2,1}$ | (1,1) | `true` |
| 11 | 2 | $x_1 \leq x_2$ | $x_2 \geq x_1$ | $[x_1, x_2]$ | $[x_2, x_1]$ | $\vee$ | `true` | (1,2), (2,1) | `true` |
| 12 | 3a | $x_1 \leq c_{1,1}$ | – | $[x_1]$ | – | $\wedge$,$\vee$ | $c_{1,1} = 0$ | – | $x_1 = 0$ |
| 13 | 3b | $x_1 = c_{1,1}$ | $x_1 \leq c_{2,1}$ | $[x_1]$ | $[x_1]$ | $\vee$ | $c_{1,1} = c_{2,1} + 1$ | (1,1) | $x_1 \leq c_{1,1}$ |
| 14 | 3b | $x_1 \geq c_{1,1}$ | $\neg(x_1 \in [c_{2,1}, c_{2,2}])$ | $[x_1]$ | $[x_1]$ | $\wedge$ | $c_{1,1} \geq c_{2,1} \wedge$ $c_{1,1} \leq c_{2,2}$ | (1,1) | $x_1 \geq c_{2,2} + 1$ |
| 15 | 3b | $x_1 \leq x_2$ | $x_2 \leq x_1$ | $[x_1, x_2]$ | $[x_2, x_1]$ | $\wedge$ | `true` | (1,2), (2,1) | $x_1 = x_2$ |

### 6.4.3 Using anti-rewriting constraints to restrict the search space

For acquiring a BAE mentioning a subset of the potential conditions $C_1, C_2, \ldots, C_m$ of Section 6.2.1 we state:

— the core model constraints introduced in Section 6.2.2,

— the enhanced model constraints described in Section 6.3, and finally

— the anti-rewriting constraints: we post the anti-rewriting constraints that mention a single condition $C_i$ (with $i \in [1, m]$), i.e. those of Family 3a, and the anti-rewriting constraints referencing two conditions $C_i$ and $C_j$ (with $i, j \in [1, m], i < j$), i.e. those of families 1, 2, and 3b.

**Anti-rewriting constraints that mention a single condition** For each condition $C_i$ of $C_1, C_2, \ldots, C_m$ (with $i \in [1, m]$), we collect the set $\mathcal{S}_i$ of stored constraints from the database of anti-rewriting constraints, i.e. those constraints of Family 3a, mentioning condition $C_i$. For each anti-rewriting constraint $k$ of the set $\mathcal{S}_i$, let $n_{i,k}$ be a 0–1 integer value defined as 1 (resp. 0) if condition $C_i$ is used positively (resp. negatively) in the $k$-th anti-rewriting constraint; we state the following constraint to avoid generating a formula mentioning a condition $C_i$ that could be simplified, where the variable $b_i$ was introduced in Section 6.2.2 to indicate how condition $C_i$ is used

$$(b_i = n_{i,k}) \Rightarrow \neg\hbar \tag{6.10}$$

Implication (6.10) can be interpreted as follows. If condition $C_i$ is used in the $k$-th anti-rewriting constraint, we ensure through implication that the hypothesis $\hbar$ is negated, as otherwise the condition $C_i$ could be simplified.

**Anti-rewriting constraints that mention two conditions** For each pair of conditions $C_i$, $C_j$ of $C_1, C_2, \ldots, C_m$ (with $i, j \in [1, m]$, $i < j$) we collect the set $\mathcal{S}_{i,j}$ of stored constraints from the database of anti-rewriting constraints, i.e. those constraints of families 1, 2 and 3b mentioning conditions $C_i$ and $C_j$. For each anti-rewriting constraint $k$ of the set $\mathcal{S}_{i,j}$ we state the following constraint

$$((b_i = n_{i,k}) \wedge (b_j = n_{j,k}) \wedge (\wedge_{(t_i,t_j)\in m}(a_{i,t_i} = a_{j,t_j}))) \Rightarrow \neg\hbar \tag{6.11}$$

Implication (6.11) can be interpreted as follows. If (*i*) condition $C_i$ is used as specified by the $k$-th anti-rewriting constraint, and similarly if (*ii*) condition $C_j$ is used appropriately, and if (*iii*) the attribute variables used in $C_i$ and $C_j$ correspond to the matching set associated with the $k$-th anti-rewriting constraint, we check that the conjunction of these three conditions implies the negation of hypothesis $\hbar$, as otherwise the term $C_i\ g\ C_j$ could be simplified.

**Example 6.4.7.** *Suppose we acquire a BAE corresponding to the conjunction of at most*

*three conditions of seven possible conditions* $C_1, C_2, \ldots, C_7$*, where*

$$C_1 \equiv \text{`}a_{1,1} = c_{1,1}\text{'}, \quad C_2 \equiv \text{`}a_{2,1} \leq c_{2,1}\text{'}, \quad C_3 \equiv \text{`}a_{3,1} \geq c_{3,1}\text{'}, \quad C_4 \equiv \text{`}a_{4,1} = a_{4,2}\text{'},$$
$$C_5 \equiv \text{`}a_{5,1} \leq a_{5,2}\text{'}, \quad C_6 \equiv \text{`}a_{6,1} \leq a_{6,2}\text{'}, \quad C_7 \equiv \text{`}a_{7,1} \in [c_{7,1}, c_{7,2}]\text{'},$$

*with respect to a database of anti-rewriting constraints consisting only of the anti-rewriting constraints in Table 6.6. For each condition* $C_i$ *(with* $i \in [1, 7]$*) (resp. for each pair of conditions* $C_i$*,* $C_j$ *(with* $i, j \in [1, 7]$*,* $i < j$*)), the left part of Figure 6.2 (resp. the right part of Figure 6.2) provides the set of anti-rewriting constraints mentioning condition* $C_i$ *(resp. conditions* $C_i$ *and* $C_j$*). Implication (6.10) (resp. Implication (6.11)) will be stated for each of the identified anti-rewriting constraints.*

(A)

| $C_1$ | $C_2$ | $C_3$ | $C_4$ | $C_5$ | $C_6$ | $C_7$ |
|---|---|---|---|---|---|---|
| ∅ | 12 | ∅ | ∅ | ∅ | ∅ | ∅ |

(B)

| | $C_1$ | $C_2$ | $C_3$ | $C_4$ | $C_5$ | $C_6$ | $C_7$ |
|---|---|---|---|---|---|---|---|
| $C_1$ | – | 1,13 | 3 | ∅ | ∅ | ∅ | ∅ |
| $C_2$ | – | – | 9 | ∅ | ∅ | ∅ | ∅ |
| $C_3$ | – | – | – | ∅ | ∅ | ∅ | 14 |
| $C_4$ | – | – | – | – | 5,7 | 5,7 | ∅ |
| $C_5$ | – | – | – | – | – | 15 | ∅ |
| $C_6$ | – | – | – | – | – | – | ∅ |

Figure 6.2 – Sets of anti-rewriting constraint identifiers (taken from the column named $k$ from Table 6.1) for (A) conditions $C_1, C_2, \ldots, C_7$ and (B) their corresponding pairs.

*For example, the following implications will be stated for the anti-rewriting constraints number 12, 1, 7 and 14.*

*12: The twelfth anti-rewriting constraint mentions only the condition* $C_2$*. Following Implication (6.10), we post the constraint* $(b_2 = 1) \Rightarrow \neg(c_{2,1} = 0)$*.*

*1: The first anti-rewriting constraint mentions the two conditions* $C_1$ *and* $C_2$*. Following Implication (6.11), we post the constraint* $(b_1 = 1 \wedge b_2 = 1 \wedge a_{1,1} = a_{2,1}) \Rightarrow \neg(c_{1,1} \leq c_{2,1})$*.*

*7: The seventh anti-rewriting constraint mentions the two conditions* $C_4$ *and* $C_5$ *with two pairs of matching attributes and a negated hypothesis. Following Implication (6.11), we post the constraint* $(b_4 = 1 \wedge b_5 = 1 \wedge a_{4,1} = a_{5,2} \wedge a_{4,2} = a_{5,1}) \Rightarrow \neg \texttt{true}$*.*

*14: The fourteenth anti-rewriting constraint mentions the two conditions $C_3$ and $C_7$, one of them being negated, and a negated compound hypothesis. Following Implication (6.11), we post the constraint* $(b_3 = 1 \wedge b_7 = 0 \wedge a_{3,1} = a_{7,1}) \Rightarrow \neg(c_{3,1} \geq c_{7,1} \wedge c_{3,1} \leq c_{7,2})$.

### 6.4.4 Method for acquiring anti-rewriting constraints

In this section we show how to systematically search for anti-rewriting constraints corresponding to the three definitions provided in Section 6.4.2. Ideally, the aim would be to prove the validity of each anti-rewriting constraint. Given that we obtain several thousand anti-rewriting constraints, we cannot envisage a manual proof for each of them. We use the following general approach to get around this difficulty. Instead of proving a logical formula associated with an anti-rewriting constraint, we use constraint programming to check that the negation of the formula has no solution by making an assumption about the range of unknowns in the formula, i.e. we check that we cannot find a counterexample that invalidates the logical formula: For the variables in the formula, we take a range of nonnegative values, while for the coefficients we take a small interval corresponding to the values used in the Bound Seeker. Note that with this method, the anti-rewriting constraints found are valid if the variables and coefficients are well within the respective ranges considered during the generation of the anti-rewriting constraints. If this were not the case, the anti-rewriting constraints found might prove to be too restrictive, preventing the Bound Seeker from finding certain formulae.

To explain the general methodology to acquire anti-rewriting constraints we first introduce some vocabulary. For the different anti-rewriting constraints (6.6), (6.7), (6.8), and (6.9) we call *in-condition* conditions $C_1^{\mathtt{ar}}$ and $C_2^{\mathtt{ar}}$, and *out-condition* condition $C_3^{\mathtt{ar}}$. To generate each anti-rewriting constraint for a given in-condition $C_1^{\mathtt{ar}}$ or two given in-conditions $C_1^{\mathtt{ar}}$ and $C_2^{\mathtt{ar}}$ of Table 6.1, and an aggregator $g$, we do as follows:

1. Using Definition 8, we enumerate over the possible hypotheses. To reduce the number of hypotheses to be considered, we check for families 1 and 3b that, for each selected hypothesis $\hbar$, the conjunction $\hbar \wedge C_1^{\mathtt{ar}} \wedge C_2^{\mathtt{ar}}$ admits at least one solution.
2. Using Table 6.1, we enumerate over the possible out-conditions if an out-condition is mentioned in the definition of the anti-rewriting constraint.
3. We check that the negation of the formula associated with the anti-rewriting constraint has no solution, provided the variables and the coefficients of the in-conditions

are in a given range. If this is the case, we record the in-conditions, the hypothesis, and the eventual out-condition of the found anti-rewriting constraint.

To improve the performance of Step 3 for Families 1 and 2, we split the implications (6.6) and (6.7) into two parts and check that only one of them is true, by verifying that only the negation of one of them has no solution.

— For Family 1, $\hbar \Rightarrow (C_1^{\mathtt{ar}} \Rightarrow C_2^{\mathtt{ar}})$ and $\hbar \Rightarrow (C_2^{\mathtt{ar}} \Rightarrow C_1^{\mathtt{ar}})$.

— For Family 2 (with $g \in \{\vee, \wedge, \oplus, \mathrm{eq}\}$), $\hbar \Rightarrow (C_1^{\mathtt{ar}}\ g\ C_2^{\mathtt{ar}})$ and $\hbar \Rightarrow \neg(C_1^{\mathtt{ar}}\ g\ C_2^{\mathtt{ar}})$.

### 6.4.5 Finding the set of most general anti-rewriting constraints

Among all anti-rewriting constraints sharing the same condition or the same pair of conditions, we need to extract the most general ones. To do this, we look for the most general set of hypotheses for a given aggregator and a condition or pair of conditions shared by a set of anti-rewriting constraints.

In practice, for each family 1, 2, 3a, and 3b many candidate hypotheses, denoted by $\mathcal{H}_{g,C_1^{\mathtt{ar}},C_2^{\mathtt{ar}}}$, are found for an aggregator $g$ and a given pair of conditions $C_1^{\mathtt{ar}}$ and $C_2^{\mathtt{ar}}$. If fact, for certain triples $g$, $C_1^{\mathtt{ar}}$, $C_2^{\mathtt{ar}}$, we found several hundred candidate hypotheses. Many pairs of hypotheses in $\mathcal{H}_{g,C_1^{\mathtt{ar}},C_2^{\mathtt{ar}}}$ are redundant, either because they are equivalent or because one hypothesis is implied by the other.

**Example 6.4.8.** *For example, with the candidate hypothesises $\hbar_1$ = '$c_{1,1} \geq c_{2,1}$' and $\hbar_2$ = '$c_{1,1} \geq c_{2,1} + 1$' we keep only $\hbar_1$ because it is implied by $\hbar_2$, i.e. if $\hbar_2$ is* `true`*, then $\hbar_1$ is also* `true`*.*

We now describe the method for eliminating redundant hypotheses from the set of candidate hypotheses $\mathcal{H}_{g,C_1^{\mathtt{ar}},C_2^{\mathtt{ar}}}$. We compare all pairs of hypotheses $\hbar_i$, $\hbar_j$ (with $i \neq j$) of $\mathcal{H}_{g,C_1^{\mathtt{ar}},C_2^{\mathtt{ar}}}$, and we remove one of the hypotheses if one of the logical expressions is met:

- $\hbar_i \Leftrightarrow \hbar_j$: if one of them is a simple hypothesis and the other is a compound hypothesis we keep the simple hypothesis, otherwise we keep hypothesis $\hbar_i$.
- $\neg(\hbar_i \Leftrightarrow \hbar_j) \wedge (\hbar_i \Rightarrow \hbar_j)$: we retain the more general hypothesis $\hbar_j$.

As before, to check a logical expression, we use a CP program to test that the negation of the logical expression has no solution.

**Example 6.4.9.** *The search for the most general hypotheses for the aggregators $\wedge$ or $\vee$, and for the pair of conditions '$x \bmod c_{1,2} = c_{1,1}$' and '$x \notin [c_{2,1}, c_{2,2}]$', (with $x \in [1, 100]$) reduces the number candidate hypothesises for Family 1 from 596 to 6 hypothesises:*

- $\hbar_1$: '$(c_{1,2} + 1 \leq c_{2,1}) \wedge (2 \cdot c_{1,2} \geq c_{2,2})$',
- $\hbar_2$: '$c_{1,1} \geq c_{2,2} + 1$',
- $\hbar_3$: '$(c_{1,2} \geq c_{2,2} + 1) \wedge (c_{1,1} + 1 \leq c_{2,1})$',
- $\hbar_4$: '$(c_{1,1} = 1) \wedge (c_{1,2} \geq c_{2,2})$',
- $\hbar_5$: '$(c_{1,2} \geq c_{2,2}) \wedge (2 \cdot c_{1,1} = c_{2,1})$',
- $\hbar_6$: '$(c_{1,2} \geq c_{2,2}) \wedge (c_{1,1} + 1 = c_{2,1})$'.

### 6.4.6 Characteristics of the generated database of anti-rewriting constraints

The described system generated 2,072 simple and compound constraints for Family 1 in less than an hour, 1,228 simple and compound constraints for Family 2 in less than an hour, and 468 simple constraints for Family 3 in about three days using a 2.6 GHz Intel Core i7.

Note that we did not automatically generate anti-rewriting constraints for conditions using ' mod ' operations, as this was too costly, since CP systems handle arithmetic expressions that mix the 'mod' operator with multiple occurrences of a same variable poorly.

The automatically generated anti-rewriting constraints were combined with 14 handwritten constraints covering cases outside the scope of the three families. Handwritten constraints are more general than automatically generated anti-rewriting constraints, as they can cover several cases at once. Handwritten constraints include:

— Constraints that consider information about the domains of the attributes involved in the expression $C_1^{\mathtt{ar}}\ g\ C_2^{\mathtt{ar}}$. For example, the expression '$(x_1 = c_{1,1}) \wedge (x_1 \leq x_2)$' can be simplified to '$x_1 = c_{1,1}$' under the hypothesis $\hbar =$ '$c_{1,1} \leq \min(x_2)$'. The system does not generate such constraints because it would be too costly to find them.

— Constraints with conditions using 'mod' operations.

## 6.5 Additional applications of Boolean-arithmetic expressions

We will look at two other applications of BAEs, where BAEs are used as a building block for learning more complex expressions:

— A first application where we extend the acquisition of conditional expressions introduced in [1], using the BAE to learn complex conditions rather than using a predefined set of simple conditions as before. An example of this situation was provided at the end of Section 2.2 by Item 5.

— A second application, where the number of distinct values in an output column is small, is the acquisition of case formulae: we use the acquisition of a BAE to find the conditions which determine which value, i.e. which branch of the case should be selected, as illustrated at the end of Section 2.2 by Item 6.

### 6.5.1 Using Boolean-arithmetic expressions to learn extended conditionals

We assume that we already know two simple functions $f_{\text{then}}(X_j)$ and $f_{\text{else}}(X_j)$ (with $X_j = \text{tab}[j,1], \text{tab}[j,1], \ldots, \text{tab}[j,c-1]$, $j \in [1,r]$) such that, for the output column $c$, we have either $\text{tab}[j,c] = f_{\text{then}}(X_j)$ or $\text{tab}[j,c] = f_{\text{else}}(X_j)$ for all $j \in [1,r]$. Under this hypothesis, we want to acquire a formula of the form $\text{tab}[j,c] = (cond(X_j)\,?\,f_{\text{then}}(X_j) : f_{\text{else}}(X_j))$. We use the BAE to enable the acquisition of more complex conditions for the '$cond(X_j)$' condition than the simple conditions used in [1].

To this end, we replace each output column value $\text{tab}[j,c]$ such that $\text{tab}[j,c] = f_{\text{then}}(X_j)$ and $\text{tab}[j,c] \neq f_{\text{else}}(X_j)$ by the intermediate Boolean value $\text{tab}'[j,c] = 1$; similarly, we replace each value $\text{tab}[j,c]$ such that $\text{tab}[j,c] = f_{\text{else}}(X_j)$ and $\text{tab}[j,c] \neq f_{\text{then}}(X_j)$ by $\text{tab}'[j,c] = 0$. We ignore those values for which $\text{tab}[j,c] = f_{\text{then}}(X_j) = f_{\text{else}}(X_j)$, as the value of the conditional expression $(cond(X_j)\,?\,f_{\text{then}}(X_j) : f_{\text{else}}(X_j))$ is, in this case, independent of the condition $cond(X_j)$. Finally, we search for a BAE for the entries of table $\text{tab}'$.

**Example 6.5.1.** *Consider the example introduced in Item 5 of Section 2.2, where we used the conditional expression* $\left(f = \underline{f} \vee v = f \;?\; \underline{f} \;:\; f - \underline{f}\right)$. *In this context, Table 6.7 provides, for an excerpt of the corresponding table* $tab[j,k]$ *(with* $j \in [1,6]$, $k \in [1,4]$*),*

— *the 'then' part of the conditional, i.e.* $f_{then}(\underline{f}) = \underline{f}$,

— *the 'else' part of the conditional, i.e.* $f_{else}(f, \underline{f}) = f - \underline{f}$, *and*

— *the corresponding entries of table* $tab'[j,k]$ *(with* $j \in [1,6]$, $k \in [1,4]$*) which will be used to find the condition* $f = \underline{f} \vee v = f$.

Table 6.7 – Illustrating the construction of Table tab$'[j,k]$ for finding the condition $f = \underline{f} \vee v = f$ of the extended conditional of the example introduced in Item 5 of Section 2.2

| v | f | $\underline{\text{f}}$ | tab$[j,c] = \overline{f}$ | $f_{\text{then}}(\underline{f})$ | $f_{\text{else}}(f,\underline{f})$ | tab$'[j,c]$ |
|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 0 | 1 |
| 2 | 1 | 1 | 1 | 1 | 0 | 1 |
| 2 | 2 | 1 | 1 | 1 | 1 | – |
| 3 | 1 | 1 | 1 | 1 | 0 | 1 |
| 3 | 2 | 2 | 2 | 2 | 0 | 1 |
| 3 | 2 | 1 | 1 | 2 | 1 | 0 |

### 6.5.2 Using Boolean-arithmetic expressions to learn case formulae as small decision trees

Decision trees are widely used for classification problems, see e.g. [91, 92, 177], as decision trees are often more interpretable compared to other machine learning approaches [177]. Some work such as [93] make very limited use of Boolean-arithmetic conditions in the nodes of a decision tree. However, to the best of our knowledge, no work on the acquisition of decision trees really considers the particularities associated with the search for small case formulae in the context of discrete combinatorial objects:

— While our input data may be large, the resulting decision tree should be small to get interpretable formulae.

— To get small decision trees, we need to combine various logical operators such as the 7 operators introduced in Section 6.1, with a fair variety of arithmetic conditions such as the 57 conditions depicted in Table 6.1.

— Knowing that our input data contains no errors and consists of integer values, we have to reconcile two conflicting objectives: (*i*) finding small decision trees (*ii*) that accurately cover all our input data.

Besides the example explained in Item 6 at the end of Section 2.2, typical examples of such case formulae derived from the acquisition of small decision trees are shown in Figure 6.3.

In the rest of this section, we describe our approach to learn small decision trees. If the number of distinct values in an output column is limited, i.e. less than or equal to four and greater than two, as otherwise we could consider a conditional expression, we use BAEs to build a small decision tree that classifies each distinct value of the output column. To illustrate the method for building such a decision tree, we will use a running example that we now present.

$$\begin{cases} 0 & \texttt{if } x_1 = 1 \\ 1 & \texttt{else if } x_1 = x_2 \vee 2 \cdot x_2 \leq x_1 \\ 2 & \texttt{otherwise} \end{cases} \quad \begin{cases} 0 & \texttt{if } x_1 = 1 \wedge x_2 \leq 2 \\ 1 & \texttt{else if } x_2 = x_1 \vee x_3 = 1 \\ 2 & \texttt{otherwise} \end{cases}$$

$$\begin{cases} 0 & \texttt{if } x_1 = 1 \wedge x_2 = 1 \\ 1 & \texttt{else if } x_1 = x_2 \\ 2 & \texttt{otherwise} \end{cases} \quad \begin{cases} 0 & \texttt{if } (x_1 + x_2) \leq 3 \\ 1 & \texttt{else if voting}(x_1 = x_2,\ x_2 = 1,\ (x_1 - x_3) \leq 2) \\ 2 & \texttt{else if } x_1 = x_3 \vee x_2 = 1 \vee (x_1 - x_2) \bmod x_3 = 0 \\ 3 & \texttt{otherwise} \end{cases}$$

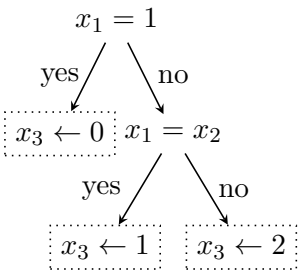
Figure 6.3 – Typical formulae we acquired using small decision trees with BAE

**Example 6.5.2.** *Consider the table shown in Part (A) of Figure 6.4, which consists of two input columns $x_1$ and $x_2$, and an output column $x_3$ containing three different values. Part (B) of Figure 6.4 represents the decision tree classifying the three output values, and Part (C) gives the corresponding case formula.*

(A)

| $x_1$ | $x_2$ | $x_3$ |
|---|---|---|
| 1 | 1 | 0 |
| 1 | 2 | 0 |
| 2 | 2 | 1 |
| 2 | 3 | 2 |
| 2 | 4 | 2 |
| 3 | 3 | 1 |
| 3 | 4 | 2 |

(B)




(C) $$x_3 = \begin{cases} 0 & \texttt{if } x_1 = 1 \\ 1 & \texttt{else if } x_1 = x_2 \\ 2 & \texttt{otherwise} \end{cases}$$

Figure 6.4 – (A) Example table for illustrating the acquisition of small decision trees, (B) corresponding decision tree, and (C) associated case formula.

Note that, within Example 6.5.2, the value of $x_3$ is functionally determined by the two input columns $x_1$ and $x_2$. However, observe that whether $x_3$ is assigned the value 0 or a value in the set $\{1, 2\}$ is determined solely by the input column $x_1$. This brings us back to the concept of conditional functional dependency [178] that we now recall.

**Definition 17.** *Given a two-dimensional table* $\text{tab}[1..r, 1..c]$ *of integer values, consisting of $r$ distinct rows and $c$ distinct columns, where column $c$ is functionally determined by columns $1, 2, \ldots, c-1$, the set of* Conditional Functional Dependency *(CFD) wrt two disjoint subsets of values $\mathcal{V}_0$ and $\mathcal{V}_1$ taken by column $c$ is the set of minimal functional dependencies determining column $c$, provided that all values in $\mathcal{V}_0$ and in $\mathcal{V}_1$ are respectively*

*replaced by* 0 *and* 1 *in the* $c$-th *column of table* $\mathrm{tab}[1..r, 1..c]$*, ignoring rows for which* $\mathrm{tab}[i, c] \notin \mathcal{V}_0 \cup \mathcal{V}_1$ *(with* $i \in [1, r]$*). Such a set of conditional functional dependencies is referred to as* $\mathtt{CFD}_{\mathcal{V}_0,\mathcal{V}_1}$.

At each level of a decision tree, we will use a conditional functional dependency to determine which input columns from column set $\{1, 2, \ldots, c-1\}$ should be used to find the corresponding BAE. We first present a naive approach to building such a decision tree and then refine it.

#### 6.5.2.1 A first deterministic approach for learning small decision trees

Let $\mathcal{C} = \{v_1, v_2, \ldots, v_p\}$ be the set of distinct output values, sorted in increasing order, used in the $c$-th column of table $\mathrm{tab}[1..r, 1..c]$. For each value $v_k$ of the set $\mathcal{C}$ (with $k \in [1, p]$), we perform the following steps.

1. set $\mathcal{V}_1$ to $\{v_k\}$ and $\mathcal{V}_0$ to $\{v_{k+1}, v_{k+2}, \ldots, v_p\}$
2. for each row $i$ of tab:
   if $\mathrm{tab}[i, c] \in \mathcal{V}_1$ then reset $\mathrm{tab}[i, c]$ to 1 else reset $\mathrm{tab}[i, c]$ to 0
3. let $\mathtt{cfd}_{\mathcal{V}_0,\mathcal{V}_1} \in \mathtt{CFD}_{\mathcal{V}_0,\mathcal{V}_1}$ be a CFD determining $\mathrm{tab}[i, c]$
4. run the Boolean-arithmetic acquisition process to find a condition which is true iff $\mathrm{tab}[i, c] = 1$ wrt the selected input columns in $\mathtt{cfd}_{\mathcal{V}_0,\mathcal{V}_1}$
5. remove from table tab all rows such that $\mathrm{tab}[i, c] \in \mathcal{V}_1$, and restore the initial output value of the output column of the remaining entries

Although the previous algorithm is simple, its main weakness is that, if Step 4 fails, the procedure will not find a decision tree. To add flexibility to the previous deterministic approach, we can do two things:

— To not just take the output values in ascending order, as in Step 1, but consider all possible permutations of the output values. Note that the conditional functional dependencies selected at Step 3 depend on the permutation being considered.

— To not just pick up one conditional functional dependency, as in Step 3, but consider all possible conditional functional dependencies.

#### 6.5.2.2 A non-deterministic approach for learning small decision trees

To consider all permutations of the output values and all conditional functional dependencies, we construct the following search tree, which will later be explored using a

breadth-first search to find a concrete decision tree. Note that the size of such a tree is rather limited, as we consider permutations of at most four output values.

— The state of each node $\mathtt{n}$ of the search tree consists of two disjoint sets of values $\mathcal{V}_0^{\mathtt{n}}$ and $\mathcal{V}_1^{\mathtt{n}}$, as well as possibly of one conditional functional dependency $\mathtt{cfd}^{\mathtt{n}}_{\mathcal{V}_0,\mathcal{V}_1} \in \mathtt{CFD}^{\mathtt{n}}_{\mathcal{V}_0,\mathcal{V}_1}$.

— The root node of the search tree, i.e. the node labelled with '1', consists of the sets $\mathcal{V}_0^1 = \mathcal{C}$ and $\mathcal{V}_1^1 = \emptyset$.

— Given a node $\mathtt{n}$ of the search tree, with the two sets of values $\mathcal{V}_0^{\mathtt{n}}$ and $\mathcal{V}_1^{\mathtt{n}}$ its children $\mathtt{n}'$ are defined in the following way:

$$\forall v \in \mathcal{V}_0^{\mathtt{n}}, \forall \mathtt{cfd}^{\mathtt{n}}_{\mathcal{V}_0^{\mathtt{n}}\setminus\{v\},\{v\}} \in \mathtt{CFD}^{\mathtt{n}}_{\mathcal{V}_0^{\mathtt{n}}\setminus\{v\},\{v\}} : \begin{cases} \mathcal{V}_0^{\mathtt{n}'} & = \mathcal{V}_0^{\mathtt{n}} \setminus \{v\} \\ \mathcal{V}_1^{\mathtt{n}'} & = \{v\} \\ \mathtt{cfd}^{\mathtt{n}'}_{\mathcal{V}_0^{\mathtt{n}'},\mathcal{V}_1^{\mathtt{n}'}} & = \mathtt{cfd}^{\mathtt{n}}_{\mathcal{V}_0^{\mathtt{n}}\setminus\{v\},\{v\}} \end{cases}$$

**Example 6.5.3.** *Figure 6.5 shows the precomputed search tree for Example 6.5.2. For example, the three arcs of the leftmost path from the root node labelled by the state* $\mathcal{V}_0^1$, $\mathcal{V}_1^1$ *to the leave nodes labelled by the state* $\mathcal{V}_0^2$ , $\mathcal{V}_1^2$ , $\mathtt{cfd}^2_{0,1}$ *respectively indicate:*

— *The arc from the root node* $\mathcal{V}_0^1$, $\mathcal{V}_1^1$ *to* $\mathcal{V}_0^2$, $\mathcal{V}_1^2$, $\mathtt{cfd}^2_{0,1}$ *indicates that if we replace the values in* $\mathcal{V}_1^2 = \{0\}$ *(resp. the values in* $\mathcal{V}_0^2 = \{1,2\}$*) in column* $x_3$ *in the table of the example given in Part (A) of Figure 6.4 with the value* 1 *(resp. the value* 0*), then* $x_3$ *is functionally determined by* $x_1$*.*

— *The arc from* $\mathcal{V}_0^2$, $\mathcal{V}_1^2$, $\mathtt{cfd}^2_{0,1}$ *to* $\mathcal{V}_0^3$, $\mathcal{V}_1^3$, $\mathtt{cfd}^3_{0,1}$ *indicates that, ignoring the value* 0 *of* $x_3$ *(i.e. values of* $x_3$ *not in* $\mathcal{V}_0^2$*), if we replace the values in* $\mathcal{V}_1^3 = \{1\}$ *(resp. the values in* $\mathcal{V}_0^3 = \{2\}$*) with value* 1*, (resp. the value* 0*), then* $x_3$ *is functionally determined by* $x_1$ *and* $x_2$*.*

— *Finally, the arc from* $\mathcal{V}_0^3$, $\mathcal{V}_1^3$, $\mathtt{cfd}^3_{0,1}$ *to the leaf node* $\mathcal{V}_0^4$, $\mathcal{V}_1^4$, $\mathtt{cfd}^4_{0,1}$ *indicates that we already consider all possible values of* $x_3$ *except value* 2 *as* $\mathcal{V}_0^4 = \emptyset$ *and* $\mathcal{V}_1^4 = \{2\}$*.*

*The three output values in* $\mathcal{V}_1^2$, $\mathcal{V}_1^3$, *and* $\mathcal{V}_1^4$ *and the variables of the corresponding conditional functional dependencies* $\mathtt{cfd}^2_{0,1}$, $\mathtt{cfd}^3_{0,1}$, *and* $\mathtt{cfd}^4_{0,1}$ *matches, from top to bottom, the output values and variables used in the case formula shown in Part (C) of Figure 6.4.*

**Heuristics used to explore the search tree** Given that we have to acquire a BAE for each node in the search tree that does not correspond to the root or a leaf, and that

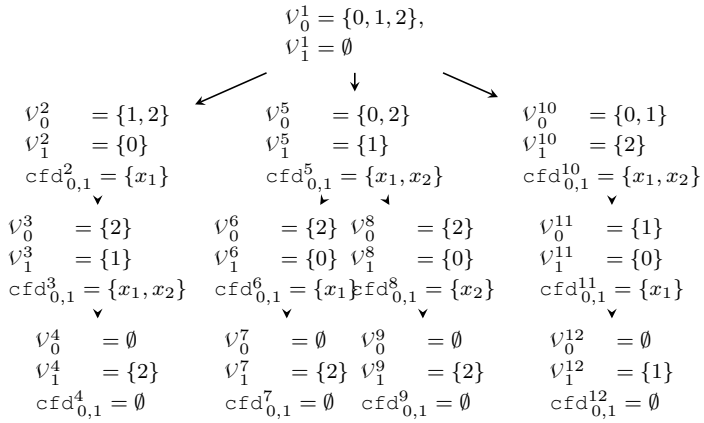



Figure 6.5 – Search tree of the running example, where $\mathtt{cfd}_{0,1}^i$ is a shortcut for $\mathtt{cfd}_{\mathcal{V}_0^i,\mathcal{V}_1^i}^i$ (with $i \in [1, 12]$)

acquiring a BAE can be expensive, we use a heuristic to select the next node on the frontier along which the breadth-first search algorithm is currently exploring. For the same reason as above, namely that it is expensive to acquire a BAE, we stop the search as soon as we have found all the BAEs along one of the branches of the search tree. The heuristic described below tends to produce smaller decision trees faster than many other search strategies we have also tested.

Each node *node* of the frontier is associated with a cost vector whose number of components $q$ is the maximum between, on the one hand, ($i$) the maximum possible arity over all BAE of the search tree, i.e. 3 in our case, and, on the other hand, ($ii$) the maximum number of parameters of the conditional functional dependencies. We select the node of the frontier whose cost vector is the smallest from a lexicographic point of view. The cost vector of a node on the frontier is the sum of three vectors, which we will now describe.

— For the part of the path $P$ from the root of the search tree to the node *node*, the cost corresponds to a vector whose $i$-th component (with $i \in [1, q]$) is the number of Boolean equations already found of arity $q - i + 1$ on $P$.

— For the sub-tree $S$ whose root is the node *node*, the cost corresponds to a vector which is the lowest cost from a lexicographical point of view associated with the different paths in the sub-tree $S$, starting from the node *node* and leading to a leaf of $S$. The cost of a given path corresponds to a vector of $q$ components, where the $i$-th component (with $i \in [1, q]$) is the number of conditional functional dependencies involving $q - i + 1$ columns.

— Finally, for the sub-tree $S$ whose root is the node *node*, we also have a vector of $q$ components, where the $i$-th component (with $i \in [2, q]$) is equal to zero, and where the first component is defined as follows. If there is at least one path from the node *node* to a leaf of $S$ where the numbers of entries associated with the different output values on that path are not increasing, then the first component is 1, otherwise it is 0. This is because we prefer the last output value to have the largest number of entries, as (*i*) we do not have to compute any BAE for the last output value, and as (*ii*) calculating a BAE for an output value with fewer entries usually results in a simpler formula.

The construction of the three previous vectors takes account of the fact that we assign the highest weights to the components with the lowest index, considering that we are ultimately selecting the smallest cost vector in the lexicographical sense. This favours the selection of a node for which there is a path to a leaf that minimises the number of variables located in the BAEs still to be found.

**Example 6.5.4.** *Consider the search tree for the running example shown in Figure 6.5. The costs of the nodes associated with the three nodes on the initial frontier of the search tree with sets* $\mathcal{V}_1^2$, $\mathcal{V}_1^5$, *and* $\mathcal{V}_1^{10}$ *are defined as the sum of three vectors, as explained above:*

- *for* $\mathcal{V}_1^2$ *we get* $\langle 0,0,0\rangle + \langle 0,1,1\rangle + \langle 0,0,0\rangle = \langle 0,1,1\rangle$.
- *for* $\mathcal{V}_1^5$ *we get* $\langle 0,0,0\rangle + \langle 0,1,1\rangle + \langle 1,0,0\rangle = \langle 1,1,1\rangle$.
- *for* $\mathcal{V}_1^{10}$ *we get* $\langle 0,0,0\rangle + \langle 0,1,1\rangle + \langle 1,0,0\rangle = \langle 1,1,1\rangle$.

*We therefore select the node associated with* $\mathcal{V}_1^2$ *and obtain the BAE* $x_1 = 1$. *The frontier now consists of the nodes* $\mathcal{V}_1^3$, $\mathcal{V}_1^5$, *and* $\mathcal{V}_1^{10}$, *where the cost of the node associated with* $\mathcal{V}_1^3$ *is equal to* $\langle 0,0,1\rangle + \langle 0,1,0\rangle + \langle 0,0,0\rangle = \langle 0,1,1\rangle$. *We therefore select the node associated with* $\mathcal{V}_1^3$ *and obtain the BAE* $x_1 = x_2$. *Now, as we reach a leaf of the search tree, we stop the search as we found a formula. This formula corresponds to the case formula shown in Part (C) of Figure 6.4.*

## 6.6 Evaluation

The CP core model introduced in Section 6.2 and its extensions described in Sections 6.3 and 6.4, as well as the extended conditionals and case formulae presented in Section 6.5, were evaluated in the context of the search for conjectures on sharp bounds on characteristics of several combinatorial objects. The source code is available

at https://github.com/cquimper/MapSeekerCPAIORExtended together with the instructions on how to utilise the code to replicate the results of this section.

We first review the different types of combinatorial objects and the corresponding input data used in this evaluation. We then examine the contribution of using Boolean-arithmetic equations, extended conditionals and case formulae for acquiring sharp bounds on the characteristics of combinatorial objects, besides the polynomials and conditional formulae used in [1]. Finally, we measure the enhancements introduced in this article in Sections 6.3 and 6.4, of the CP core model for acquiring BAEs presented in Section 6.2.

### 6.6.1 Describing the combinatorial objects and the data used in the experiments

We consider the following set of combinatorial objects for our experiments.

— **digraph (without isolated vertices):** a set of vertices $\mathcal{V}$ and a set of ordered pairs of vertices $\mathcal{A}$ with the restriction that each vertex of $\mathcal{V}$ occurs in at least one pair of $\mathcal{A}$ [140].

— **rooted tree:** a connected acyclic undirected graph where a vertex is designed as the "root" of the tree [141].

— **rooted forest:** a disjoint union of rooted trees [141]; we also consider a variant, **rooted forest2**, where all rooted trees have at least two vertices.

— **partition:** a partition of a set $\mathcal{S}$ is a collection of non-empty subsets of $\mathcal{S}$ such that every element of $\mathcal{S}$ is in exactly one of the subsets of the collection. The use of a partition was motivated by the fact that a partition can be interpreted as a solution to the conjunction of the NVALUE (i.e. the number of partition subsets, see [142]) and the BALANCE (i.e. the difference between the cardinalities of the largest and smallest subsets of the partition, see [3, pp 698–703]) constraints. Motivated by the extension of the BALANCE constraint, i.e. ALL_BALANCE [143], we also consider a version of partition named **partition0** where a subset may be empty.

— **stretch**: a solution of the stretch constraint on 0-1 variables $v_0, v_1, \ldots, v_{n-1}$ where a subsequence of 1 immediately preceded and followed by a 0 is called a *stretch* [144]; we also consider the variant named **cyclic stretch** where we use modular arithmetic wrt $n$ to define a cyclic stretch.

For DIGRAPH, ROOTED TREE, ROOTED FOREST, and ROOTED FOREST2, *size* denotes the number of vertices. For PARTITION and PARTITION0, *size* is the number of elements

of the set we partition, and for STRETCH and CYCLIC STRETCH, *size* is the sequence length. The data set [15] used for acquiring conjectures consists of a collection of 252300 tables representing 12 GB, giving for any combinatorial object of size at most *size*, for any combination of at most three input parameters, for any feasible combinations of values of these input parameters, the sharp lower or the sharp upper bound of a given output parameter. An input table may contain auxiliary parameters, called *secondary parameters*, all functionally determined by the table's input parameters. The programs generating all the data used in this section have been included in the code made available from https://github.com/cquimper/MapSeekerCPAIORExtended. In the context of digraphs, an example of input table was given in Chapter 2, see Table 2.1, with two input parameters corresponding to the number of vertices and the number of connected components, and one output parameter corresponding to the maximum number of arcs.

### 6.6.2 Evaluating the contribution of Boolean-arithmetic equations and their extensions for learning sharp bounds

#### 6.6.2.1 Illustrating the diversity of Boolean-arithmetic formulae found

To illustrate the diversity of Boolean-arithmetic formulae found, Table 6.8 shows for each combinatorial object and some of their characteristics, some conjectures discovered using the Boolean model described in this paper. hose conjectures are (*i*) either an equality expressing the value of a characteristic when another characteristic is reaching its sharp bound (bounded characteristic) (*ii*) either an inequality representing a sharp bound formulated wrt other characteristics (bounding characteristics) [1].

#### 6.6.2.2 Evaluating the contribution of Boolean-arithmetic equations to learn sharp bounds

We evaluate the number of formulae found by the full version of the model which includes all the contributions of Sections 6.2, 6.3, 6.4, and 6.5 (i.e. Model 2) which replace those formulae found by the model acquiring only polynomials or conditionals of [1] (i.e. the Model 1) and the number of new formulae which were not found by Model 1, as mentioned in Table 6.9, Table 6.10 and Table 6.11. To achieve this experiment the two models, written in SICStus 4.7.1, were executed on a cluster with Intel processors such as Silver 4216 Cascade Lake @ 2.1˜GHz, and E7-4809 v4 Broadwell @ 2.1˜GHz. We use

Table 6.8 – Examples of characteristics (char.) of combinatorial objects and corresponding conjectures: (*i*) $c, s, oc, \underline{c}$ and $\overline{s}$: *number of connected components (cc), strongly connected components (scc), connected components with at least two vertices, size of the smallest cc* and *size of the largest scc* of a **digraph**; (*ii*) $c_0$: *denote* 0 *if all the cc have the same maximal size, and* $\underline{c}$ *otherwise*, for a **digraph**; (*iii*) $v$ and $f$: *number of vertices* and *leaves* in a **rooted tree**; (*iv*) $\underline{d}$ and $\overline{d}$: *smallest degree and largest degree of a parent node* in a **rooted tree** or a **rooted forest**; (*v*) $\underline{p}, \overline{p}$ and $\underline{t}$: *minimum depth, maximum depth* and *size of the smallest tree* in a **rooted forest**; (*vi*) $n$, *nval*, and $\underline{m}$: *number of elements*, *number of subsets*, and *cardinality of the smallest subset* in a **partition**; (*vii*) *sr*, *dr*, *dmin* and *dmax*: *difference between the number of elements of the largest and smallest stretches*, *difference between the maximum and minimum distance of consecutive stretches*, *minimum distance between consecutive stretches* and *maximum distance between consecutive stretches* in **stretch**; (*viii*) $n$, *smax*, *ng*, and *osc*: *number of 0–1 variables, number of elements of the largest stretch, number of stretches, and number of stretches which have more than one element* in **cyclic stretch**.

| Combinatorial object | Kind of bounded char. | Bounding char. | Examples of discovered conjectures when the bounded char. achieves its bound |
|---|---|---|---|
| **digraph** | lower bound of $c$ | $\underline{c}, s, oc$ | $c \geq 1 + [\neg(s \leq \underline{c} \wedge oc \leq 1)]$ |
| **digraph** | lower bound of $\underline{c}$ | $c, s, \overline{s}$ | $c_0 = [\neg\,\text{voting}(c = s, c = 1, \min(c, \overline{s}) = 1)]$ |
| **rooted tree** | upper bound of $f$ | $\underline{d}, v, \underline{p}$ | $\overline{p} = (\underline{p} \geq 2 \vee v - \underline{d} = 1\,?\,\underline{p} : 2)$ |
| **rooted forest** | lower bound of $f$ | $\underline{p}, \overline{d}, \underline{t}$ | $\overline{d} = 2 - ([v + \underline{p} \leq 2] + [(\underline{t} - \underline{p}) = 1])$ |
| **partition** | lower bound of *nval* | $n, \underline{m}$ | $nval \geq 1 + [2 \cdot \underline{m} \leq n]$ |
| **stretch** | lower bound of *dmax* | $sr, dr$ | $dmin = [(sr + dr) \geq 1]$ |
| **cyclic stretch** | upper bound of *smax* | $n, ng$ | $osc = [\neg\,\text{card1}(n = 2 \cdot ng, n \cdot ng = 3, n \cdot ng \leq 3)]$ |

510 threads for the whole experiment and allocate a time out of 48 hours to each thread to find all the lower and upper bounds of the different combinatorial objects.

— Table 6.9 gives (*i*) the number of Boolean formulae found, i.e. 4171 (sum of second columns) (*ii*) the number of Boolean formulae replacing a formula with a polynomial or a conditional, i.e. 4091 (sum of third columns) and (*iii*) the number of new Boolean formulae, i.e. 80 (sum of fourth columns) discovered compared to the model described in [1], i.e. Model 1, which only looked for formulae with conditionals, polynomials and arithmetic functions involving two polynomials.

— Table 6.10 gives (*i*) the number of extended conditional formulae found, i.e. 237 (sum of second columns) (*ii*) the number of extended conditional formulae replacing a formula with a polynomial or a conditional, i.e. 213 (sum of third columns) and (*iii*) the number of new extended conditional formulae, i.e. 24 (sum of fourth columns).

— Table 6.11 gives (*i*) the number of cases formulae found, i.e. 295 (sum of second

columns) (*ii*) the number of cases formulae replacing a formula with a polynomial or a conditional, i.e. 266 (sum of third columns) and (*iii*) the number of new cases formulae, i.e. 29 (sum of fourth columns).

This experiment shows that by using the new model, i.e. Model 2, for learning sharp bounds, the Boolean-arithmetic equations and their extensions contribute a lot for the interpretability of formula as 4570 Boolean, extended conditionals and case formulae replace polynomials and conditionals of the model described in [1], i.e. Model 1. Model 2 also contributes identifying new formulae not found by the Model 1 as 133 Boolean, extended conditionals, and cases formulae were newly found in the context of learning sharp bound.

Table 6.9 – Contribution of the full version of the model with incorporated anti-rewriting constraints of Section 6.4, i.e. Model 2, which acquires Boolean formulae (BF), polynomials and conditionals, compared to searching only polynomial and conditional formulae with the model of [1], i.e. Model 1.

| Combinatorial object | Number of (BF) which are: found | replacing polynomials & conditionals | new |
|---|---|---|---|
| **digraph** | 515 | 464 | 51 |
| **rooted tree** | 61 | 61 | 0 |
| **rooted forest** | 436 | 431 | 5 |
| **rooted forest2** | 415 | 415 | 0 |

| Combinatorial object | Number of (BF) which are: found | replacing polynomials & conditionals | new |
|---|---|---|---|
| **partition** | 113 | 113 | 0 |
| **partition0** | 51 | 51 | 0 |
| **stretch** | 1373 | 1359 | 14 |
| **cyclic stretch** | 1207 | 1197 | 10 |

Table 6.10 – Contribution of the full version of the model with incorporated anti-rewriting constraints of Section 6.4, i.e. Model 2, which acquires extended conditionals formulae (EC) (described in Section 6.5.1), polynomials and conditionals, compared to searching only polynomial and conditional formulae with the model of [1], i.e. Model 1.

| Combinatorial object | Number of (EC) wich are: found | replacing polynomials & conditionals | new |
|---|---|---|---|
| **digraph** | 69 | 53 | 16 |
| **rooted tree** | 12 | 12 | 0 |
| **rooted forest** | 111 | 103 | 8 |
| **rooted forest2** | 36 | 36 | 0 |

| Combinatorial object | Number of (EC) which are found | replacing polynomials & conditionals | new |
|---|---|---|---|
| **partition** | 3 | 3 | 0 |
| **partition0** | 4 | 4 | 0 |
| **stretch** | 34 | 29 | 5 |
| **cyclic stretch** | 26 | 26 | 0 |

Table 6.11 – Contribution of the full version of the model with incorporated anti-rewriting constraints of Section 6.4, i.e. Model 2, which acquires case formulae (CF) (described in Section 6.5.2), polynomials and conditionals, compared to searching only polynomial and conditional formulae with the model of [1], i.e. Model 1.

| Combi-natorial object | Number of (CF) which are: found | replacing polynomials & conditionals | new |
|---|---|---|---|
| **digraph** | 32 | 28 | 4 |
| **rooted tree** | 3 | 3 | 0 |
| **rooted forest** | 11 | 8 | 3 |
| **rooted forest2** | 1 | 1 | 0 |

| Combi-natorial object | Number of (CF) which are: found | replacing polynomials & conditionals | new |
|---|---|---|---|
| **partition** | 6 | 4 | 2 |
| **partition0** | 3 | 3 | 0 |
| **stretch** | 107 | 100 | 7 |
| **cyclic stretch** | 74 | 66 | 8 |

### 6.6.3 Evaluating the enhancements of the CP core model for acquiring Boolean-arithmetic equations

In this section we evaluate the computing time spent by the core model of Section 6.2 (i.e. C. Model), by its enhanced version of the model of Section 6.3 (i.e. E. Model) and by the full version of the model with incorporated anti-rewriting constraints of Section 6.4 (i.e. F. Model) wrt (*i*) the kind of combinatorial object, and wrt (*ii*) the type of aggregator used in a BAE.

To achieve this, we took 1546 examples of BAEs we acquired in Section 6.6.2 to generate a corresponding test table for each. Then, for each generated test table we select the aggregator $g$ and the number of terms $n_{AC}$, both of which correspond to the selected acquired BAE, and execute one of the models. We tested the performance on a MacBookPro with a 2.6 GHz Core i7 and 16 GB of memory using SICStus 4.6.0. The aggregated results are presented in Tables 6.12 and 6.13.

Table 6.12 – Acquisition time for C. Model, E. Model and F. Model wrt combinatorial objects

| Combinatorial object | Number of conjectures | Average time per conjecture C.Model | E.Model | F.Model |
|---|---|---|---|---|
| **digraph** | 226 | 33s | 5.5s | 5.4s |
| **rooted tree** | 17 | 0.7s | 0.2s | 0.3s |
| **rooted forest** | 171 | 4.2s | 2.2s | 2.3s |
| **rooted forest2** | 106 | 5.3s | 3s | 2.8s |
| **partition** | 50 | 2.3s | 1.4s | 1.5s |
| **partition0** | 23 | 1.7s | 0.9s | 1s |
| **stretch** | 447 | 4.4s | 1.6s | 1.6s |
| **cyclic stretch** | 506 | 2.6s | 1.4s | 1.5s |
| Total | 1546 | 3.4h | 1h | 1h |

Table 6.12 shows that both the E. Model and the F. Model acquire a BAE with, on

Table 6.13 – Acquisition time for C. Model, E. Model and F. Model wrt aggregators

| $g$ | $n_{AC}$ | Number of conjectures | Average time per conjecture C.Model | E.Model | F.Model |
|---|---|---|---|---|---|
| $\wedge$ | 1 | 1148 | 0.49s | 0.3s | 0.3s |
| $\wedge$ | 2 | 52 | 15s | 7.8s | 8.4s |
| $\vee$ | 2 | 90 | 11s | 5.4s | 5.8s |
| $\vee$ | 3 | 1 | 2415s | 209s | 192s |
| eq | 2 | 73 | 22s | 13s | 14s |
| eq | 3 | 14 | 398s | 62s | 56s |
| $+$ | 2 | 160 | 1.7s | 0.9s | 1.2s |
| $+$ | 3 | 8 | 3.6s | 1.1s | 1.5s |

average, 72% less time than the C. Model. Additional tests showed that using just the constraints from Section 6.3.1 increases the speed of the C. Model by $\approx 5\%$, just the constraints from Section 6.3.2 – by $\approx 63\%$, and just the constraints from Section 6.3.3 – by $\approx 48\%$.

Table 6.13 shows that anti-rewriting constraints of Section 6.4 slightly slow down the performance during acquisition of simpler BAE, e.g. when $g = `\wedge'$ and $n_{AC} = 2$, as a little time is spent on posting the constraints. There is a small drawback of including the anti-rewriting constraints.

The benefit of using the anti-rewriting constraints is twofold ($i$) they reduce the time of acquisition of more complex BAE, e.g. for $g = `\text{eq}'$ and $n_{AC} = 3$ anti-rewriting rules reduced time by 9.6%, for $g = `\vee'$ and $n_{AC} = 3$ – by 8.1%, and ($ii$) for $\approx 2.8\%$ of tables they prevented generation of more complex equations. For example, for a table for the **graph**, the F. Model generated only one BAE, $s = 3 - ((\underline{c} = \underline{s}) + (\underline{c} = \overline{c}))$, while the E. Model also produced a more complex BAE, $s = 1 + (\underline{c} - \underline{s} \geq 1) + (\lceil \overline{c} \div \underline{c} \rceil \geq 2)$.

Chapter 7

# Decompositions as a recursive way to acquire complex formulae

Previously, in Section 2.3.1 we explored the motivation behind examination of various decomposition techniques to synthesise complex formulae which combine several learning biases and in Section 2.3.2 we gave concrete examples of complex formulae obtained using our new decomposition techniques. In Section 7.1 we provide the definition of what decomposition technique is. In Section 7.2 we describe the decomposition techniques introduced in this chapter. In Section 7.3, we evaluate our contribution to assess its performance and the relevance of the discovered conjectures. Some of the more complicated conjectures found by the system were proven by Jovial Cheukam-Ngouonou.

## 7.1 Decompositions definition

By decomposing the formula to acquire into an expression consisting of several simpler terms as mentioned in Section 2.3.2, we aim to solve an easier acquisition problem. To this end, we introduce four ways of decomposing a function. Before formally defining them, we explain each of them intuitively.

— [**Adding a Boolean expression**] The first decomposition breaks down a function as the sum of a function with fewer input parameters, and a function whose codomain is the set $\{0, 1\}$. As shown in Conj. (1), this decomposition was motivated by the possibility of approximating a bound by an error margin of at most 1.

— [**Isolating a parameter**] The 2nd decomposition is based on the idea that one may find a formula in which an input parameter only occurs in the outer term of the formula where no other input parameter is used. In Conj. (2), the parameter $s$ only occurs in the numerator of the top-level of binary function 'integer division rounded up'.

— [**Isolating a parameter and using a 0-1 slack**] The 3rd decomposition generalises the 2nd decomposition by introducing a 0-1 slack term that can refer to any subset of the input parameters. In Conj. (3), the input parameter $s$ does not occur in the numerator of the formula; indeed, it is only mentioned by the denominator of the top level function 'integer division rounded up'.

— [**Introducing a conditional**] The 4th decomposition is based on the intuition that it may be easier to find a formula that applies to a subset of the table entries rather than to all. The 4th decomposition divides the table entries into a 'then' and an 'else' set using a simple condition then, for each set, we have to identify a corresponding formula. We use a small set of predefined formulae with a subset of the condition's parameters for the 'then' set, while imposing no restriction on the 'else' set. Conj. (4) illustrates such conditional decomposition.

### 7.1.1 Problem statement

Given a table $\text{tab}[1..nrows, 1..ncol+1]$ of integer values, consisting of $nrows$ rows and $ncol+1$ columns, where columns $1, 2, \ldots, ncol$ form a mfd determining column $ncol+1$, we address the following question: How to decompose the problem of discovering a function $g$ satisfying all the following set of equalities

$$\forall j \in [1, nrows] : \ \text{tab}[j, ncol+1] = g(\text{tab}[j, 1], \ldots, \text{tab}[j, ncol]) \tag{7.1}$$

into a set of easier subproblems requiring finding a limited number of functions satisfying one of the decompositions (7.2)–(7.5) of Def. 18 that we now introduce.

**Definition 18** (Decomposition Types)**.**

$$\forall j \in [1, nrows] : \text{tab}[j, ncol+1] = g_1(\text{tab}[j, a_1], \ldots, \text{tab}[j, a_{\ell_1}]) + g_2(\text{tab}[j, b_1], \ldots, \text{tab}[j, b_{\ell_2}] \tag{7.2}$$

$$\forall j \in [1, nrows] : \text{tab}[j, ncol+1] = g_1(\text{tab}[j, a_1], \ldots, \text{tab}[j, a_{\ell_1}]) \oplus g_3(\text{tab}[j, b_1]) \tag{7.3}$$

$$\forall j \in [1, nrows] : \text{tab}[j, ncol+1] = g_1(\text{tab}[j, a_1], \ldots, \text{tab}[j, a_{\ell_1}]) \oplus (g_2(\text{tab}[j, b_1], \ldots, \text{tab}[j, b_{\ell_2}]) + g_3(\text{tab}[j, b_1])) \tag{7.4}$$

$$\forall j \in [1, nrows] : \mathrm{tab}[j, ncol + 1] = \begin{array}{r}(\mathrm{cond}(\mathrm{tab}[j, c_1], \ldots, \mathrm{tab}[j, c_{\ell_3}]) \ ?\\ g_4(\mathrm{tab}[j, d_1], \ldots, \mathrm{tab}[j, d_{\ell_4}]) :\\ g_1(\mathrm{tab}[j, a_1], \ldots, \mathrm{tab}[j, a_{\ell_1}]))\end{array} \quad (7.5)$$

1. $g_1 : \mathbb{Z}^{\ell_1} \to \mathbb{Z}$ *refers to one of the biases* (i)*–*(iii) *or to a formula obtained by one of the four decompositions;* $a_1, \ldots, a_{\ell_1}$ *are distinct indices from* $[1, ncol]$ *with* $\ell_1 \in [1, ncol - 1]$ *for (7.2)–(7.4) as* $g_1$ *does not involve all input parameters, and with* $\ell_1 \in [1, ncol]$ *for (7.5). For (7.3)–(7.4),* $b_1$ *is different from* $a_1, \ldots, a_{\ell_1}$.

2. $g_2 : \mathbb{Z}^{\ell_2} \to \{0, 1\}$ *matches bias* (iii)*;* $b_1, \ldots, b_{\ell_2}$ *are distinct indices from* $[1, ncol]$ *with* $\ell_2 \in [1, ncol]$. *Note that in (7.2), functions* $g_1$ *and* $g_2$ *may share some parameters.*

3. $g_3 : \mathbb{Z} \to \mathbb{Z}$ *is one of the unary functions* $A \cdot x^2 + B \cdot x + C$, $\lfloor \frac{A \cdot x^2 + B \cdot x}{D} \rfloor$, $\lceil \frac{A \cdot x^2 + B \cdot x}{D} \rceil$, $\min(A \cdot x + B, C)$, $\max(A \cdot x + B, \ C)$, $(A \cdot x + B) \bmod D$, $|A \cdot x + B|$, $[((x + A) \bmod D) = C]$, $[((x + A) \bmod D) \geq C]$, $[((x + A) \bmod D) \leq C]$, *with* $A$, $B$, $C \in \mathbb{Z}$, *and* $D \in \mathbb{Z}^+$. *To limit the search space, we consider unary functions involving up to* 3 *constants.*

4. *Within (7.3)–(7.4),* $\oplus$ *stands for one of the operators* '$+$', '$\cdot$', 'min', 'max', '$\lfloor \text{-} \rfloor$', *or* '$\lceil \text{-} \rceil$'.

5. *Within (7.5),* cond *is a condition mentioning at most* 3 *parameters, i.e.* $\ell_3 \in [1, 3]$, *of the form* '$x = \min(x)$', '$x = y$', '$x \leq y$', '$x \bmod y = 0$', '$x = y \cdot z$', '$A \cdot x \leq y$', *while* $g_4$ *is one of the functions* '$B$', '$x$', '$[x = \min(x)]$', '$[x > \min(x)]$', '$x \cdot y$', '$[x = y + B]$', '$x = y + z$', *with* '$A$', '$B$' $\in \mathbb{Z}$.

We introduce a fair number of functions and conditions for $g_3$, $g_4$, and cond in Def. 18. To avoid overfitting, they mention at most 3 coefficients whose range is restricted to $[-2, 2]$.

**Example 7.1.1.** *Within Sect. 2.3.2, the right-hand part of inequalities (2.2), (2.3), (2.4), and (2.5) resp. matches the following decomposition types:*

— *(7.2) with* $g_1(v, \overline{c}) = \lceil \frac{v}{\overline{c}} \rceil$ *and* $g_2(v, \overline{c}, \underline{s}) = [\neg((2 \cdot \underline{s} \leq \overline{c}) \vee (\underline{s} \geq (v \bmod \overline{c} = 0 \ ? \ \overline{c} \ : \ v \bmod \overline{c}))]$.

— *(7.3) with* $g_1(\overline{c}, \underline{s}) = \lfloor \overline{c} / \underline{s} \rfloor$, $g_3(s) = s$, *and* $\oplus =$ '$\lceil \text{-} \rceil$'.

— *(7.4) with* $g_1(v, \underline{s}) = (v = \underline{s} \ ? \ v : v - \underline{s})$, $g_2(s) = [s = 1]$, $g_3(s) = s - 1$, *and* $\oplus =$ '$\lceil \text{-} \rceil$'.

— *(7.5) with* $g_1(v, c, \underline{c}, \overline{s}) = \max\left(\overline{s}, \left\lceil \frac{v - \underline{c}}{c - 1} \right\rceil\right)$ *and* $g_4(\underline{c}) = \underline{c}$.

## 7.2 Implementing the different types of decompositions

The implementation combines ($a$) phases of generating a limited number of alternatives on the type of functions used in the decomposition and on which parameters these functions mention, and ($b$) test phases verifying certain simple conditions and solving a constraint model to find the values of the coefficients of the functions mentioned in the terms of the decomposition. We introduce some notation to refer to intermediate structures used to analyse the consequence of eliminating an input parameter from a mfd.

**Notation 3.** *Consider the table* $\mathrm{tab}[1..nrows, 1..ncol+1]$*, in which the first ncol columns, the input parameters, form a mfd determining column* $ncol+1$*, i.e. the output parameter.*

- *Let* $\mathrm{tab}_k^{\nearrow}[1..nrows, 1..ncol+1]$ *be the table obtained by sorting the rows of the table* $\mathrm{tab}[1..nrows,\ 1..ncol+1]$ *in increasing lexicographic order wrt columns* $1, \ldots, k-1, k+1, \ldots, ncol+1$*, i.e. column k is skipped; to make the correspondence between the entries of the tables* $\mathrm{tab}_k$ *and* $\mathrm{tab}_k^{\nearrow}$*, let* $\sigma_k$ *denote the permutation that maps the* $j$*-th row of the table* $\mathrm{tab}_k$ *to the* $\sigma_k(j)$*-th row of the table* $\mathrm{tab}_k^{\nearrow}$ *(with* $j \in [1, nrows]$*).*
- *Let* $\mathcal{I}$ *denote the parameters associated with columns* $1, 2, \ldots, ncol$ *of the table* tab*, and let* $\mathcal{I}_j$ *(with* $j \in [1, nrows]$*) represents the corresponding* parameter values *for the* $j$*-th row of the table* tab*, i.e. values* $\mathrm{tab}[j,1], \mathrm{tab}[j,2], \ldots, \mathrm{tab}[j, ncol]$*.*
- *Let* $\mathcal{I}^k$ *(with* $k \in [1, ncol]$*) denote the parameters associated with columns* $1, \ldots, k-1, k+1, \ldots, ncol$ *of the table* tab*, while let* $\mathcal{I}_j^k$ *(with* $j \in [1, nrows]$*) represents the corresponding* parameter values $\mathrm{tab}[j,1], \ldots, \mathrm{tab}[j,k-1], \mathrm{tab}[j,k+1], \ldots, \mathrm{tab}[j, ncol]$*.*

### 7.2.1 Decomposition of Type (7.2) [adding a Boolean expression]

**Question** We want to check whether there is a $k \in [1, ncol]$ such that $\forall j \in [1, nrows]$ : $\mathrm{tab}[j, ncol+1] = g_1(\mathcal{I}_j^k) + g_2(\mathcal{I}_j)$, with $g_2 : \mathbb{Z}^{\ell_2} \to \{0,1\}$; i.e. we seek an approximation with a maximum error of 1, by using a function $g_1$, without parameter $k$, and a correction term $g_2$.

**Steps for finding a decomposition of Type (7.2)**

1. [**Determining the parameters of** $g_1$] First, we successively select the $k$-th column (with $k \in [1, ncol]$) to remove from the input parameters of the function $g_1$, and we apply Steps 2. to 4. for each candidate column $k$.

| | $v$ | $\overline{c}$ | $\underline{s}$ | $c$ |
|---|---|---|---|---|
| 1 | 9 | 2 | 1 | 5 |
| 2 | 9 | 3 | 1 | 3 |
| 3 | **9** | **3** | **2** | **4** |
| 4 | 9 | 3 | 3 | 3 |
| 5 | 9 | 4 | 1 | 3 |
| 6 | 9 | 4 | 2 | 3 |
| 7 | 9 | 5 | 1 | 2 |
| 8 | 9 | 5 | 2 | 2 |
| 9 | 9 | 5 | 4 | 2 |

(A) tab[1..9,1..4]

$\sigma_3$

| | $v$ | $\overline{c}$ | $\underline{s}$ | $c$ |
|---|---|---|---|---|
| 1 | 9 | 2 | 1 | 5 |
| 2 | 9 | 3 | 1 | 3 |
| 3 | 9 | 3 | 3 | 3 |
| 4 | **9** | **3** | **2** | **4** |
| 5 | 9 | 4 | 1 | 3 |
| 6 | 9 | 4 | 2 | 3 |
| 7 | 9 | 5 | 1 | 2 |
| 8 | 9 | 5 | 2 | 2 |
| 9 | 9 | 5 | 4 | 2 |

(B1) $\text{tab}_3^{\nearrow}$[1..9,1..4]

| | |
|---|---|
| 1 | 5 |
| 2 | 3 |
| 3 | 3 |
| 4 | **3** |
| 5 | 3 |
| 6 | 3 |
| 7 | 2 |
| 8 | 2 |
| 9 | 2 |

(B2) $\min_3^{\nearrow}$[1..9]

| | $v$ | $\overline{c}$ | $g_1(v,\overline{c})$ |
|---|---|---|---|
| 1 | 9 | 2 | 5 |
| 2 | 9 | 3 | 3 |
| 3 | **9** | **3** | **3** |
| 4 | 9 | 3 | 3 |
| 5 | 9 | 4 | 3 |
| 6 | 9 | 4 | 3 |
| 7 | 9 | 5 | 2 |
| 8 | 9 | 5 | 2 |
| 9 | 9 | 5 | 2 |

(C1)

| | $v$ | $\overline{c}$ | $\underline{s}$ | $g_2(v,\overline{c},\underline{s})$ |
|---|---|---|---|---|
| 1 | 9 | 2 | 1 | 0 |
| 2 | 9 | 3 | 1 | 0 |
| 3 | **9** | **3** | **2** | **1** |
| 4 | 9 | 3 | 3 | 0 |
| 5 | 9 | 4 | 1 | 0 |
| 6 | 9 | 4 | 2 | 0 |
| 7 | 9 | 5 | 1 | 0 |
| 8 | 9 | 5 | 2 | 0 |
| 9 | 9 | 5 | 4 | 0 |

(C2)

Figure 7.1 – Tables used to find a decomposition of type 7.2 for Conj. (1); bold entries refer to Ex. 7.2.1.

2. **[Checking whether the codomain of $g_2$ is the set $\{0,1\}$]** Second, provided function $g_1$ does not use the $k$-th input parameter selected in Step 1, we analyse how this affects the codomain of function $g_2$, even if functions $g_1$ and $g_2$ are yet unknown.

   For each maximum interval of consecutive rows $[\ell, u]$ in the sorted table $\text{tab}_k^{\nearrow}[1..nrows, 1..ncol+1]$ for which columns $1,\ldots,k-1,k+1,\ldots,ncol$ have the same value, we get the maximum $max_{\ell,u}$ and minimum $min_{\ell,u}$ values in the $(ncol+1)$-th column, and we check that the difference $max_{\ell,u} - min_{\ell,u}$ does not exceed 1. In other words, we test for the table $\text{tab}_k^{\nearrow}$ that, for each combination of identical input parameters, from which the $k$-th input parameter is ignored, the corresponding output parameter varies by at most 1. When satisfied, this test ensures that the codomain of $g_2$ is in $\{0,1\}$.

   For each entry $j \in [\ell, u]$ of the table $\text{tab}_k^{\nearrow}$ we set $\min_k^{\nearrow}[j] = min_{\ell,u}$, where $\min_k^{\nearrow}$ is a one-dimensional table whose entries vary from 1 to $nrows$.

3. **[Determining the values of $g_1(\mathcal{G}_j^k)$ and $g_2(\mathcal{G}_j)$]** Third, for each combination of input parameters of functions $g_1$ and $g_2$ we compute their respective output values: $\forall j \in [1, nrows]$, $g_1(\mathcal{G}_j^k) = \min_k^{\nearrow}[\sigma_k(j)]$ and $g_2(\mathcal{G}_j) = \text{tab}_k^{\nearrow}[\sigma_k(j), ncol+1] - \min_k^{\nearrow}[\sigma_k(j)]$.

4. **[Using $g_1(\mathcal{G}_j^k)$ and $g_2(\mathcal{G}_j)$ for identifying functions $g_1$ and $g_2$]** We search for $g_1$ by using the CP solvers associated with biases $(i)$–$(iii)$ or by applying recursively one of the decompositions of this paper. To identify $g_2$ we call the Boolean solver associated with the bias $(iii)$.

**Example 7.2.1** (Illustrating the Search for a Decomposition of Type (7.2) for Conj. (2.2))**.** *Part (A) of Fig. 7.1 provides* 9 *entries of the table* 'tab' *with input parameters* $v$, $\overline{c}$, $\underline{s}$ *and the lower bound of the output parameter* $c$*, previously introduced. Assume we skip the third column of table* 'tab'*,* $k = 3$*, shown in grey in table* $\text{tab}_3^{\nearrow}$*, i.e. we ignore column* $\underline{s}$*.*

— *Parts (B1) and (B2) resp. show the tables introduced for finding a decomposition of Type (7.2), i.e. tables* 'tab'*,* $\text{tab}_3^{\nearrow}$*, and* $\min_3^{\nearrow}$*. The permutation* $\sigma_3$ *(with* $\sigma_3(3) = 4$*,* $\sigma_3(4) = 3$*, and* $\sigma_3(j) = j$ *otherwise) maps the entries of table* 'tab' *to the entries of table* $\text{tab}_3^{\nearrow}$*. The rows of* $\text{tab}_3^{\nearrow}$ *and* $\min_3^{\nearrow}$ *can be partitioned in four maximum intervals, depicted in dark and light grey, resp. corresponding to the pair of values* $(9, 2)$*,* $(9, 3)$*,* $(9, 4)$*, and* $(9, 5)$ *for the input parameters* $v$ *and* $\overline{c}$*. As for each of these four intervals the difference between the maximum and the minimum value of* $c$ *does not exceed one, we can compute the values of* $g_1(v, \overline{c})$ *and* $g_2(v, \overline{c}, \underline{s})$*.*

— *Parts (C1) and (C2) resp. give the tables used to acquire* $g_1(v, \overline{c})$ *and* $g_2(v, \overline{c}, \underline{s})$*, e.g. for* $j$=3*,* $g_1(\mathcal{G}_j^3)$=$g_1(\mathcal{G}_3^3)$=$g_1(9, 3)$=$\min_3^{\nearrow}[\sigma_3(j)]$=$\min_3^{\nearrow}[\sigma_3(3)]$= $\min_3^{\nearrow}[4]$=3*, and* $g_2(\mathcal{G}_j)$=$g_2(9, 3, 2)$=$\text{tab}_3^{\nearrow}[\sigma_3(j), 4] - \min_3^{\nearrow}[\sigma_3(j)]$=$\text{tab}_3^{\nearrow}[4, 4] - \min_3^{\nearrow}[4] = 4 - 3 = 1$.

### 7.2.2 Decompositions of Types (7.3) and (7.4) (isolating a parameter)

Using a binary operator $\oplus$, Decomposition (7.3) combines 2 sub-terms: a function $g_3$ involving a single input parameter, with a function $g_1$ mentioning only all other remaining input parameters. Decomposition (7.4) extends (7.3) a bit by adding an extra term whose codomain is the set $\{0, 1\}$. As identifying Decomposition (7.4) is very similar to identifying Decomposition (7.3), we focus on the latter for space reasons.

**Question** We want to check whether there is a $k \in [1, ncol]$ such that $\forall j \in [1, nrows]$ : $\text{tab}[j, ncol{+}1] = g_1(\mathcal{G}_j^k) \oplus g_3(\text{tab}[j, k])$, where $\oplus$ and function $g_3$ were defined in Def. 18; i.e. we want to see if we can express the formula we are looking for, by restricting one of its parameters to just one of the formula's sub-terms.

1. [**Selecting** $k$**,** $\oplus$**, and** $g_3$] To determine the images of function $g_1$ that will be used to find function $g_1$ itself in Step. 2 we successively consider the $ncol \times 10 \times 6$ combinations of triples $\langle k, g_3, \oplus \rangle$ (with $k \in [1, ncol]$), see Items 3–4 of Def. 18 for $g_3$ and $\oplus$. To find whether a combination of triples can be used or not to find the images of the function $g_1$, we apply the following steps:

| | $s$ | $\overline{c}$ | $\underline{s}$ | $c$ |
|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 |
| 2 | 2 | 1 | 1 | 2 |
| 3 | 2 | 3 | 2 | 2 |
| 4 | 2 | 9 | 3 | 1 |
| 5 | 3 | 1 | 1 | 3 |
| 6 | 3 | 3 | 2 | 3 |
| 7 | 3 | 9 | 3 | 1 |
| 8 | 4 | 3 | 2 | 4 |
| 9 | 4 | 9 | 3 | 2 |

(A) tab

| | $s$ | $\overline{c}$ | $\underline{s}$ | $c$ |
|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 |
| 2 | 2 | 1 | 1 | 2 |
| 3 | 3 | 1 | 1 | 3 |
| 4 | 2 | 3 | 2 | 2 |
| 5 | 3 | 3 | 2 | 3 |
| 6 | 4 | 3 | 2 | 4 |
| 7 | 2 | 9 | 3 | 1 |
| 8 | 3 | 9 | 3 | 1 |
| 9 | 4 | 9 | 3 | 2 |

(B) $\text{tab}_1^{\nearrow}$

| $g_3(s)=A\cdot s^2+B\cdot s+C$ | $g_1(\overline{c},\underline{s})$ | $\lceil g_3(s)/g_1(\overline{c},\underline{s})\rceil = c$ |
|---|---|---|
| $A+B+C$ | $y_1$ | $\lceil(A+B+C)/y_1\rceil=1$ |
| $4\cdot A+2\cdot B+C$ | $y_1$ | $\lceil(4\cdot A+2\cdot B+C)/y_1\rceil=2$ |
| $9\cdot A+3\cdot B+C$ | $y_1$ | $\lceil(9\cdot A+3\cdot B+C)/y_1\rceil=3$ |
| $4\cdot A+2\cdot B+C$ | $y_4$ | $\lceil(4\cdot A+2\cdot B+C)/y_4\rceil=2$ |
| $9\cdot A+3\cdot B+C$ | $y_4$ | $\lceil(9\cdot A+3\cdot B+C)/y_4\rceil=3$ |
| $16\cdot A+4\cdot B+C$ | $y_4$ | $\lceil(16\cdot A+4\cdot B+C)/y_4\rceil=4$ |
| $4\cdot A+2\cdot B+C$ | $y_7$ | $\lceil(4\cdot A+2\cdot B+C)/y_7\rceil=1$ |
| $9\cdot A+3\cdot B+C$ | $y_7$ | $\lceil(9\cdot A+3\cdot B+C)/y_7\rceil=1$ |
| $16\cdot A+4\cdot B+C$ | $y_7$ | $\lceil(16\cdot A+4\cdot B+C)/y_7\rceil=2$ |

(C)

| | $\overline{c}$ | $\underline{s}$ | $g_1(\overline{c},\underline{s})$ |
|---|---|---|---|
| 1 | 1 | 1 | 1 ($y_1$) |
| 2 | 1 | 1 | 1 ($y_1$) |
| 3 | 3 | 2 | 1 ($y_4$) |
| 4 | 9 | 3 | 3 ($y_7$) |
| 5 | 1 | 1 | 1 ($y_1$) |
| 6 | 3 | 2 | 1 ($y_4$) |
| 7 | 9 | 3 | 3 ($y_7$) |
| 8 | 3 | 2 | 1 ($y_4$) |
| 9 | 9 | 3 | 3 ($y_7$) |

(D)

Figure 7.2 – (A),(B),(D) Tables, and (C) Constraints used for finding a decomposition of Type 7.3 for Conj. (2); variables $y_1$, $y_4$, and $y_7$ in Tables (C) and (D) correspond to the 'value variables' for $g_1$.

(a) **[Creating the "value variables" for the images of $g_1$]** For each maximum interval of consecutive rows $[\ell, u]$ of the table $\text{tab}_k^{\nearrow}[1..nrows, 1..ncol+1]$ for which columns $1,\ldots,k-1,k+1,\ldots,ncol$ have the same value, we create a single domain variable $y_\ell$ *representing the value* of $g_1(\mathcal{G}^k_{\sigma_k^{-1}(\ell)})$, where $\sigma_k^{-1}$ denotes the inverse permutation of permutation $\sigma_k$.

(b) **[Stating the row constraints for finding the coefficients of $g_3$, and the value of $g_1$ for each row]** For each entry $j$ of a maximal interval of consecutive rows $[\ell, u]$ of the table $\text{tab}_k^{\nearrow}[1..nrows, 1..ncol+1]$ for which columns $1\ldots,k-1,k+1,\ldots,ncol$ have the same value, we create the constraint $y_\ell \oplus g_3(\text{tab}[\sigma_k^{-1}(j),k]) = \text{tab}[\sigma_k^{-1}(j), ncol+1]$.

(c) **[Solving the row constraints]** We solve the conjunction of constraints stated in Step 1b: we find the values of the coefficients of the unary function $g_3$, and the values of the "value variables" of $g_1$, while minimising the sum of the absolute value of the coefficients of $g_3$ using a CP solver.

Among all triples for which Step 1c found a solution, we keep those triples $\langle k, \oplus, g_3\rangle$ which minimise the sum of absolute values of the coefficients of $g_3$.

2. **[Identifying function $g_1$]** As for the decomposition of Type (7.2), we search for function $g_1$ by employing the existing CP solvers associated with biases (*i*)–(*iii*) or by applying recursively one of the 4 decompositions proposed in this paper. For this purpose we reuse the value of the "value variables" found for $g_1$ in Step 1c.

**Example 7.2.2** (Illustrating the Search for a Decomposition of Type (7.3) for Conj. (2.3))**.**

*Part (A) of Fig. 7.2 provides* 9 *entries of the data for the input parameters* $s$, $\overline{c}$, $\underline{s}$ *and the lower bound of the output parameter* $c$, *namely table* 'tab' *previously introduced. Assume we skip the first column of the table* 'tab', *i.e.* $k = 1$ *shown in grey in the table* $\text{tab}_1^{\nearrow}$, *that is we ignore the column labelled by* $s$. *The rows of Parts (B)–(C) of Fig. 7.2 can be partitioned in three maximum intervals, depicted in dark and light grey, resp. corresponding to the pair of values* $(1, 1)$, $(3, 2)$, *and* $(9, 3)$ *for the input parameters* $\overline{c}$ *and* $\underline{s}$.

— *Assuming we look for a function* $g_3$ *of Type* $A \cdot s^2 + B \cdot s + C$, *and for a binary operator* $\oplus$ *of the form* '$\lceil\text{-}\rceil$', *the three columns in Part (C) resp. show for each row of* $\text{tab}_1^{\nearrow}$, *(*p1*) the unary function* $g_3$, *(*p2*) the "value variables" for* $g_1$, *and (*p3*) the corresponding constraints.*

— *Part (D) gives the derived table used to acquire* $g_1(\overline{c}, \underline{s})$.

### 7.2.3 Decomposition of Type (7.5) (introducing a conditional)

Type (7.5) decomposition combines a simple condition and a simple function $g_4$ for the 'then' part of the condition, with a function $g_1$ corresponding to biases (*i*)–(*iii*), or obtained by one of the first three decompositions described in this paper, i.e. the conditional decomposition is not applied recursively, as this has proven to be very time-consuming. We use the following steps to search for a decomposition of Type (7.5):

1. [**Selecting** cond **and** $g_4$] We successively consider the $6 \times 7$ combinations of pairs $\langle \text{cond}, g_4 \rangle$, see Item 5 of Def. 18. To determine whether or not a combination of pairs can be used, we create this constraint model:
   (a) The variables $c_1, c_2, \ldots, c_n$ (resp. $d_1, d_2, \ldots, d_m$) denote the indices of the columns of the table $\text{tab}[1..nrows, 1..ncol+1]$ used in the condition cond (resp. in function $g_4$ of the 'then' part). These variables are in $[1, ncol]$ as they correspond to input parameters, i.e. we state the constraints $\forall i \in [1, n] : c_i \in [1, ncol]$, $\text{alldifferent}([c_1, c_2, \ldots, c_n])$, $\forall i \in [1, m] : d_i \in [1, ncol]$, and $\text{alldifferent}([d_1, d_2, \ldots, d_m])$.
   (b) For each entry $j$ (with $j \in [1, nrows]$) of the table $\text{tab}[1..nrows, 1..ncol+1]$, we state the constraints:
      i. $\forall k \in [1, n\,] : \text{element}(c_k, \text{tab}[j, 1..ncol], v_{j,k}\,)$, $\text{cond}(v_{j,1}, v_{j,2}, \ldots, v_{j,n}) \Leftrightarrow r_j, r_j \in [0, 1]$,
      ii. $\forall k \in [1, m] : \text{element}(d_k, \text{tab}[j, 1..ncol], w_{j,k})$,
      iii. $r_j = 1 \Rightarrow g_4(w_{j,1}, \ldots, w_{j,m}) = \text{tab}[j, ncol+1]$.

(c) By maximising the number of rows in the table tab[1..$nrows$, 1..$ncol+1$] for which condition 'cond' is met, we try to create a smaller subproblem for acquiring the 'else' part. This is done by stating the constraints $cost = \sum_{j\in[1,nrows]} r_j$, $cost > 0$, $cost < nrows$. The last two constraints require the 'then' (or 'else') part to contain at least one row for which condition 'cond' is true (or false) as we want to obtain a non-simplifiable conditional formula. We maximise *cost* wrt the posted constraints.

2. [**Identifying function** $g_1$] As for decompositions (7.2)–(7.4), using Item 5 of Def. 18, we search for function $g_1$ using the CP solvers related to biases $(i)$–$(iii)$, or by recursively applying decompositions (7.2)–(7.4). To do this, we focus only on all the $j$-th rows of the table tab[1..$nrows$, 1..$ncol+1$] for which condition cond$(v_{j,1}, v_{j,2}, \ldots, v_{j,n})$ does not hold, i.e. the 'else' part of the conditional.

## 7.3 Evaluation

### 7.3.1 Type of conjectures we are looking for

We search for:

(I) conjectures expressing a secondary parameter wrt input parameters,

(II) conjectures expressing sharp bound on an output parameter wrt input parameters,

(III) conjectures expressing a secondary parameter wrt both input and secondary parameters,

(IV) conjectures expressing sharp bound on an output parameter wrt both input and secondary parameters.

We prefer conjectures using input parameters only as it allows one to express sharp bounds directly wrt input parameters, i.e. without using secondary parameters. Focusing only on sharp bounds limits the number of conjectures learned, which now depends only on the number of characteristics considered for the combinatorial objects.

### 7.3.2 Experimental Setting

We compare 2 versions of the acquisition tool using SICStus 4.7.1 on an cluster with Intel processors such as Silver 4216 Cascade Lake @ 2.1GHz, and E7-4809 v4 Broadwell @

2.1Ghz. The source code is available at https://github.com/cquimper/MapSeekerAAAI24 together with the instructions on how to reproduce the results presented in this section; The 1st version uses biases (*i*)–(*iii*), while the 2nd version uses biases (*i*)–(*iii*) and the 4 decompositions (7.2)–(7.5). If one of these versions took more than 96 hours to complete the acquisition for an input table, that table is excluded from the result evaluation, unless otherwise stated. We acquire conjectures on tables of smaller sizes and test them on the largest tables using the method described in [1] on selecting table size. We exclude invalidated conjectures from our evaluation.

### 7.3.3 Experimental Results

Out of a total of 4469 tables, the 1st version had timeouts on 44 tables, the 2nd version on 49 tables, with almost no overlap. We remove these tables and use the remaining 4378 tables to compare 2 versions. The 4378 tables has 21797 secondary and output parameters. As cluster node performance varies and we cannot control allocation of tables over CPUs, we only compare the aggregated full acquisition time for both versions. The 1st version took 5888 hours in total, while the 2nd version took 25053 hours to complete. A total of 26 (resp. 54) conjectures acquired by the 1st (resp. 2nd) version were not validated.

Table 7.1 shows the detailed results of the experiment. The 1st and the 2nd versions resp. found conjectures for 16078 and 17255 secondary or output parameters. The 2nd version found conjectures of types I–IV (resp. I–II) for 5% more (resp. 14% more) secondary and output parameters compared to the 1st version. The 14% increase reflects the fact that the 2nd version expresses more conjectures with input parameters only, which is one of our goals. Including all 4469 tables, the 2nd version found conjectures for 7% more secondary and output parameters than the 1st version.

The 2nd version found 6% more conjectures of types II and IV, i.e. sharp bounds. In the 2nd version, 2857 secondary or output parameters (16.5% of the 17255 parameters) have conjectures that use decompositions (7.2)–(7.5); 26% of them use several decompositions in one conjecture.

In [14], we proved Conjectures (2.2)–(2.5) to show that our decomposition methods find non-trivial sharp bounds, as well as some non-obvious conjecture found for Rooted Tree.

Table 7.1 – Detailed experimental results for the 1st and the 2nd versions of the acquisition tool, where $n_o$ is the number of secondary and output parameters across all tables, $n_t$ is the number of acquired conjectures by the 1st or the 2nd version, $n_a$ is the number of secondary and output parameters for which the 1st or the 2nd version could acquire at least a conjecture, $n_b$ is the number of output parameters for which the 1st or the 2nd version could acquire at least a conjecture, $n_i$ is the number of secondary and output parameters for which the 1st or the 2nd version could acquire at least a conjecture input parameters only, $n_e$ is the number of conjectures invalidated on the largest available size of a combinatorial object, $n_d$ is the number of output parameters for which the 2nd version could acquire at least a conjecture using decompositions (7.2)–(7.5), $n_{7.2}$, $n_{7.3}$, $n_{7.4}$, $n_{7.5}$ are the number of output parameters for which we could resp. acquire at least a conjecture using (7.2), (7.3), (7.4), (7.5), $n_{>1}$ is the number of output parameters for which the 2nd version found at least a conjecture using more than one decompositions.

| combinatorial object | $n_o$ | 1st version | | | | | 2nd version | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | $n_t$ | $n_a$ | $n_b$ | $n_i$ | $n_e$ | $n_t$ | $n_a$ | $n_b$ | $n_i$ | $n_e$ | $n_d$ | $n_{7.2}$ | $n_{7.3}$ | $n_{7.4}$ | $n_{7.5}$ | $n_{>1}$ |
| Digraph | 2861 | 3270 | 2637 | 434 | 1789 | 2 | 3412 | 2702 | 447 | 1940 | 6 | 328 | 89 | 71 | 52 | 156 | 66 |
| Rooted Tree | 185 | 225 | 138 | 67 | 119 | 0 | 240 | 152 | 77 | 133 | 1 | 39 | 10 | 12 | 6 | 18 | 8 |
| Rooted Forest | 2088 | 2343 | 1577 | 562 | 1250 | 4 | 2672 | 1697 | 613 | 1428 | 6 | 433 | 122 | 131 | 112 | 168 | 121 |
| Rooted Forest2 | 2861 | 2404 | 1639 | 569 | 1372 | 1 | 2563 | 1700 | 607 | 1459 | 9 | 361 | 71 | 108 | 94 | 103 | 65 |
| Partition | 562 | 572 | 436 | 78 | 279 | 0 | 586 | 453 | 78 | 303 | 0 | 89 | 11 | 27 | 7 | 34 | 9 |
| Partition0 | 235 | 238 | 189 | 37 | 134 | 0 | 282 | 209 | 38 | 162 | 0 | 65 | 14 | 12 | 10 | 20 | 13 |
| Stretch | 6416 | 6481 | 4978 | 556 | 2157 | 4 | 6981 | 5237 | 582 | 2473 | 0 | 660 | 146 | 220 | 118 | 357 | 161 |
| Cyclic Stretch | 6589 | 5964 | 4484 | 521 | 2041 | 15 | 7011 | 5105 | 561 | 2486 | 32 | 882 | 103 | 299 | 131 | 636 | 309 |
| **total** | 21797 | 21497 | 16078 | 2824 | 9141 | 26 | 23747 | 17255 | 3003 | 10384 | 54 | 2857 | 566 | 880 | 530 | 1492 | 752 |

Part V

# Acquiring and generating short-term production scheduling models

Chapter 8

# Acquiring scheduling constraints

In this chapter we will describe the process of acquisition of various schedule constraints relevant to the use case described in Chapter 3. In Section 8.1 we describe the process of acquisition of temporal constraints put on subsequent tasks within a chain of tasks. In Section 8.2 we describe the process of acquisition of scheduling constraints put on the resources. In Section 8.3 we describe the process of acquisition of calendar constraints, i.e. the constraints that put limits on possible time periods for each task.

## 8.1 Acquiring temporal constraints

Using the merged table, the acquisition of temporal constraints is done in three steps:

1. Identify disjoint sequences of tasks.
2. Recognise columns that refer to the start, the duration, or the end of a task, i.e. temporal task attributes.
3. Solve a discrete optimisation problem to identify temporal constraints that hold for all pairs of adjacent tasks of the identified sequences.

### 8.1.1 Identifying disjoint sequences of tasks

A column $s$ of the merged table is a disjoint sequence of tasks if it contains two types of values: ($i$) values corresponding to distinct primary keys in the merged table, namely values representing distinct task identifiers, and ($ii$) an additional unique value that does not match any primary key in the merged table, namely a value that represents the end of a sequence of tasks. Such a column is called a *link* column, and the corresponding pairs of linked tasks are denoted by $\mathcal{L}$.

**Example 8.1.1.** *Within the merged tasks table, Table 4.1, only the 'Successor' column fulfils the above two conditions: the 'Successor' column only contains task identifiers or the single value* $-1$. *The corresponding set of linked tasks*

*is* $\mathcal{L} = \{(10001, 10002), (10002, 10003), (10003, 10004), (10005, 10006), (10006, 10007), (10008, 10009)\}$.

### 8.1.2 Identifying temporal task attributes

to identify potential temporal attributes we first calculate two sets – $\mathcal{T}_1$ and $\mathcal{T}_2$. We compute a first set of potential temporal attributes $\mathcal{T}_1$ by considering all the columns of the merged table except the columns corresponding to a primary key, a link column, and a set of values. Using the column names from the merged table, we compute a second set of potential temporal attributes $\mathcal{T}_2$. We compute a second set of potential temporal attributes $\mathcal{T}_2$ by using the column names. To achieve this, the model acquisition tool uses a dictionary of common words used to denote temporal characteristics of a task, e.g. 'duration', 'time', etc. If the column name contains a word from such dictionary the model acquisition tool puts it in the set $\mathcal{T}_2$. The dictionary can by adapted for different settings if necessary.

If the intersection of the sets $\mathcal{T}_1$ and $\mathcal{T}_2$ contains fewer than two elements (two as a temporal constraint mentions at least two columns of the merged table), the set $\mathcal{T}_1$ is the set of potential temporal attributes (as the column names could not be used to identify the temporal attributes); otherwise, the set $\mathcal{T}_1 \cap \mathcal{T}_2$ is the set of potential temporal attributes. We split the set of potential temporal attributes into two subsets $\mathcal{T}_{\text{in}}$ and $\mathcal{T}_{\text{out}}$ respectively corresponding to input and output attributes given in the metadata (see Section 3.3).

**Example 8.1.2.** *Within the merged tasks table, i.e. Table 4.1, the system identifies the set of input temporal attributes* $\mathcal{T}_{in} = \emptyset$*, and the set of output temporal attributes* $\mathcal{T}_{out} =$ {*'Start','Duration'*}.

### 8.1.3 Identifying temporal constraints

A temporal constraint consists of an inequality $\leq$, (resp. $\geq$), [1] where the left-hand side is a sum of temporal attributes to which we may add a nonnegative distance $d_{\leq}$ (resp. $d_{\geq}$), and the right-hand side is a sum of temporal attributes. Such temporal constraint has to hold for all entries of the merged table for which the value of the link column is a task identifier. Using the link column and the sets of potential temporal attributes identified in the previous steps we need to find out the following information:

1. Two symmetrical temporal constraints using $\leq$ and $\geq$ are converted to an equality.

— The set of attributes used in the left-hand side and right-hand side of the temporal constraint.

— The kind of comparison operator used in the temporal constraint.

— The distance used in the left-hand side of the temporal constraint.

We use the following constraint model to acquire these three elements:

$$\begin{aligned}
&\text{minimise } d_{\leq} + d_{\geq} \text{ such that:} \\
&\quad \forall k \in \mathcal{T}_{\text{in}} \cup \mathcal{T}_{\text{out}} : b_k^{\ell} \in \{0,1\},\ b_k^{r} \in \{0,1\}, \\
&\quad \forall (i,j) \in \mathcal{L} : \sum_{k \in \mathcal{T}_{\text{in}} \cup \mathcal{T}_{\text{out}}} x_{i,k} \cdot b_k^{\ell} + d_{i,j} \ = \sum_{k \in \mathcal{T}_{\text{in}} \cup \mathcal{T}_{\text{out}}} x_{j,k} \cdot b_k^{r}, \\
&\quad d_{\leq} \geq 0, \quad d_{\leq} = \min_{(i,j) \in \mathcal{L}} (d_{i,j}), \quad d_{\geq} \geq 0, \quad d_{\geq} = \max_{(i,j) \in \mathcal{L}} (d_{i,j})
\end{aligned} \tag{8.1}$$

where $x_{i,k}$ (resp. $x_{j,k}$) denotes the value of the $k$-th column of task $i$ (resp. $j$) in the merged table, and where the 0–1 variables $b_k^{\ell}$ and $b_k^{r}$ (with $k \in \mathcal{T}_{\text{in}} \cup \mathcal{T}_{\text{out}}$) indicate which temporal attributes (i.e. columns) are selected or not. We search for the combination of temporal attributes that minimises $d_{\leq} + d_{\geq}$ as smaller values for $d_{\leq}$ and $d_{\geq}$ produce less restrictive constraints, thus avoiding overfitting.

To focus the search of temporal constraints, we add the following restrictions to the constraints depicted in (8.1):

1. The left-hand (resp. right-hand) side of a temporal constraint should mention at most two temporal attributes from $\mathcal{T}_{\text{in}} \cup \mathcal{T}_{\text{out}}$. This is because both the left-hand and the right-hand sides of a temporal constraint never simultaneously mention the start, the duration and the end attributes.
2. The left-hand (resp. right-hand) side of a temporal constraint should mention at least a temporal attribute from the $\mathcal{T}_{\text{in}}$ set. This is because it is extremely unlikely that the left-hand (resp. right-hand) side of a temporal constraint only mention input columns of the merged table.
3. If the left-hand (resp. right-hand) side of a temporal constraint mentions only one single attribute $a_{\text{in}}$ from the $\mathcal{T}_{\text{in}}$ set, and if the right-hand (resp. left-hand) side mentions more than one attribute from $\mathcal{T}_{\text{in}}$, then the right-hand (resp. left-hand) side should also reuse the $a_{\text{in}}$ attribute.
4. If both, the left-hand and the right-hand sides of a temporal constraint mention more than one temporal attribute from the $\mathcal{T}_{\text{in}}$ set, then there is at least one attribute

$a_\ell \in \mathcal{T}_{\text{in}}$ of the left-hand side of a temporal constraint and there is at least one attribute $a_r \in \mathcal{T}_{\text{in}}$ of the right-hand side of a temporal constraint, such that for all entries of the merged table, the value of $a_\ell$ is always less than or equal (resp. greater than or equal to) to the value of $a_r$.

The choice of which temporal constraint to keep, i.e. a 'less than or equal to' or a 'greater than or equal to', depends on for how many pairs $(i, j)$ of $\mathcal{L}$, the distance $d_{i,j} = \sum_{k \in \mathcal{T}_{\text{in}} \cup \mathcal{T}_{\text{out}}} \left( x_{j,k} \cdot b_k^r - x_{i,k} \cdot b_k^\ell \right)$ is closer to $d_\leq$ or to $d_\geq$:

— If the number of pairs which are closer to $d_\leq$ is greater, then the model acquisition tool keeps the temporal constraint that uses the '$\leq$' comparison operator.

— If the number of pairs which are closer to $d_\geq$ is greater, then the model acquisition tool keeps the temporal constraint that uses the '$\geq$' comparison operator.

— Otherwise, the model acquisition tool keeps both temporal constraints.

**Example 8.1.3.** *First, using the link column, i.e. column 'Successor', and the potential temporal attributes {'Start','Duration'}, we create the set of potential temporal constraints listed in Table 8.1.*

*Second, we search for the values of the Boolean variables $b_2^\ell$, $b_4^\ell$, $b_2^r$, $b_4^r$ which minimise the sum of the two distances $d_\leq + d_\geq$ while satisfying all temporal constraints of Table 8.1. We obtain the solution $b_2^\ell = 1$, $b_4^\ell = 1$, $b_2^r = 1$, $b_4^r = 0$, $d_\leq = 0$, $d_\geq = 14$, meaning that we select the attributes 'Start' and 'Duration' for the left-hand side of the potential temporal constraint and we only select the attribute 'Start' for the right-hand side.*

*Third, we find out which of the comparisons $\leq$ or $\geq$ leads to a tighter temporal constraint. As within the distances $d_{10001,10002} = 20-(0+20) = 0$, $d_{10002,10003} = 50-(20+30) = 0$, $d_{10003,10004} = 70-(50+20) = 0$, $d_{10005,10006} = 20-(0+12) = 8$, $d_{10006,10007} = 28-(20+8) = 0$, $d_{10008,10009} = 50-(0+36) = 14$ depicted in column 'distance' of Table 8.1, the majority of them, i.e. four, are closer to $d_\leq = \min \begin{pmatrix} d_{10001,10002}, d_{10002,10003}, d_{10003,10004}, \\ d_{10005,10006}, d_{10006,10007}, d_{10008,10009} \end{pmatrix} = 0$ than to $d_\geq = \max \begin{pmatrix} d_{10001,10002}, d_{10002,10003}, d_{10003,10004}, \\ d_{10005,10006}, d_{10006,10007}, d_{10008,10009} \end{pmatrix} = 14$, we select the '$\leq$' comparison operator and obtain the temporal constraint (3.2) quoted in Example 3.4.2.*

Table 8.1 – Main model part used for acquiring temporal constraints: state an equality constraint for each task $i$ which is linked to a task $j$ via the 'Successor' column; the left (resp. right) term of a potential temporal constraint is a weighted sum of the temporal attributes of task $i$ (resp. $j$) using the Boolean variables $b_2^\ell$, $b_4^\ell$ and the distance $d_{i,j}$ (resp. $b_2^r$ and $b_4^r$) corresponding to the selection of the temporal attributes within the potential temporal constraint.

| left-hand side | | right-hand side | link | distance $d_{i,j}$ |
|---|---|---|---|---|
| $0 \cdot b_2^\ell + 20 \cdot b_4^\ell + d_{10001,10002}$ | = | $20 \cdot b_2^r + 30 \cdot b_4^\ell$ | 10001 linked to 10002 | $d_{10001,10002} = 0$ |
| $20 \cdot b_2^\ell + 30 \cdot b_4^\ell + d_{10002,10003}$ | = | $50 \cdot b_2^r + 20 \cdot b_4^\ell$ | 10002 linked to 10003 | $d_{10002,10003} = 0$ |
| $50 \cdot b_2^\ell + 20 \cdot b_4^\ell + d_{10003,10004}$ | = | $70 \cdot b_2^r + 20 \cdot b_4^\ell$ | 10003 linked to 10004 | $d_{10003,10004} = 0$ |
| $0 \cdot b_2^\ell + 12 \cdot b_4^\ell + d_{10005,10006}$ | = | $20 \cdot b_2^r + 8 \cdot b_4^\ell$ | 10005 linked to 10006 | $d_{10005,10006} = 8$ |
| $20 \cdot b_2^\ell + 8 \cdot b_4^\ell + d_{10006,10007}$ | = | $28 \cdot b_2^r + 8 \cdot b_4^\ell$ | 10006 linked to 10007 | $d_{10006,10007} = 0$ |
| $0 \cdot b_2^\ell + 36 \cdot b_4^\ell + d_{10008,10009}$ | = | $50 \cdot b_2^r + 27 \cdot b_4^\ell$ | 10008 linked to 10009 | $d_{10008,10009} = 14$ |

## 8.2 Acquiring resource constraints

In Section 3.4.3 we provided the description of resource constraints such as DISJUNCTIVE [4, 179] or DIFFN. We also introduced the WITHIN constraint is used in a scheduling problems to express that a task must be assigned to a resource taken from a given set of resources. The acquisition of WITHIN constraints is done during the metadata generation as this is a straightforward task, i.e. see Example 3.4.3.

Resource constraints such as DISJUNCTIVE [4, 179] or DIFFN [163, 180], which express that certain tasks must not overlap in time, are learned after acquiring temporal and functional constraints. In fact, the columns in the merged table that are part of the temporal and functional constraints are used to restrict the candidate columns involved in the resource constraints. To this end, the model acquisition tool follows the steps below:

— It generates a list of columns that can be interpreted as a resource column, namely a column corresponding to a resource to which some tasks are assigned. These columns must not be part of the primary key (see Section 5.1) nor be present in a temporal constraint (see Section 8.1) or in any of the acquired functional constraints (see Section 4.1.2).

— It collects temporal column candidates. It uses (*i*) the columns in the acquired temporal constraints, as well as (*ii*) the columns that are not part of the primary and foreign keys of the merged table, that do not correspond to sets of values, and that are not part of a WITHIN constraint.

— For each candidate resource column:

— From the list of temporal column candidates, it considers each pair of columns where one acts as the start time of the tasks, and the other as the duration or end time of the tasks.

— It creates a constraint model to check that tasks assigned to a same resource do not overlap.

— It moves to the next temporal column candidate if two tasks that are assigned to a same machine overlap; otherwise, it records the sum of time differences between adjacent tasks as the resource constraint cost. For each candidate resource column, the model acquisition tool selects the pair of temporal columns that minimise the cost, e.g. see the hatched areas in Figure 3.1.

— After selecting the pair of temporal columns with minimal cost, it checks whether the candidate resource column is part of a WITHIN constraint or not. If yes, the model acquisition tool acquires a DIFFN constraint as the tasks are not pre-assigned; otherwise, the model acquisition tool acquires a DISJUNCTIVE constraint as all tasks were initially pre-assigned to a resource.

**Example 8.2.1.** *To find a resource constraint, the model acquisition tool first tries to identify a resource column in the merged table. It discards (*i*) column 'Task_id' as it corresponds to a primary key of the merged table, (*ii*) columns 'Duration', 'Quantity', and 'Speed' as they are part of a functional constraint (3.1) presented in Example 3.4.1, (*iii*) column 'Start' as it occurs in the temporal constraint (3.2) shown in Example 3.4.2, (*iv*) column 'Successor' as it is a link column, and (*v*) column 'Machine_set' as it is a set of integer values. The only remaining potential resource column is the 'Machine' column, which is a word in the dictionary of the model acquisition tool assigned to denote resource columns.*

*For the 'Machine' column and the temporal columns 'Start' and 'Duration' the system finds out that the tasks which are assigned to the same machine do not overlap in time. The cost of this constraint is* 64 *corresponding to the sum of the gap between consecutive tasks on machines* 2, 3 *and* 10 *as illustrated by the hatched areas in Figure 3.1. Finally, as we have a* WITHIN *constraint involving the 'Machine' column, the model acquisition tool generates a* DIFFN *constraint stating that two tasks that will be assigned to the same machine should not overlap in time. In this* DIFFN *constraint, each task is represented as a rectangle whose bottom left corner coordinates, length, and width respectively correspond to the 'Start' column, to the 'Machine' column, to the 'Duration' column, and to* 1.

## 8.3 Acquiring calendar constraints

To acquire the CALENDAR constraints introduced in Section 3.4.4 the model acquisition tool proceeds as follows:

— It first collects resource column candidates and temporal column candidates in the same way as described in Section 8.2 for acquiring resource constraints.

— It then selects potential calendar columns, i.e. columns whose entries correspond to a sorted list of non-overlapping intervals.

— Finally, for each combination of resource column candidates, temporal column candidates, potential calendar columns, and of the *kind* parameter of a potential CALENDAR constraint, it checks whether the corresponding constraint holds or not.

**Example 8.3.1.** *As in Example 8.2.1, the unique resource column candidate is the 'Machine' column, and the temporal column candidates are the 'Start' and the 'Duration' columns. The only calendar column candidate is the 'Unavailability' column as it consists of sets of disjoint intervals. The model acquisition tool learns a* CALENDAR *constraint with a kind parameter set to* 6*, i.e. a constraint that prevents each task from overlapping any unavailability period associated with the machine to which the task is assigned.*

Chapter 9

# Generating MiniZinc models

## 9.1 Generating a MiniZinc model wrt the acquired constraints

The last step of the model acquisition tool generates an executable model as stated in Section 3.4 from all the constraints acquired in the previous step. The MiniZinc code generator successively generates:

— the *variable and data arrays* for each column,

— the input data and the output variables,

— the *foreign key constraints* between tables,

— the acquired *temporal constraints* for chains of tasks,

— the acquired *functional constraints*,

— the acquired *resource constraints*,

— the acquired *calendar constraints*,

— the acquired *shift constraints*.

Apart from the *calendar constraints* and the *shift constraints*, all other acquired constraints are available in the MiniZinc modelling language and can therefore be translated directly to MiniZinc. As the generation of the MiniZinc code for the SHIFT constraint is done in the same way as the generation of the code for the CALENDAR constraint, we will only show how to reformulate the different variants of the CALENDAR constraint introduced in Section 8.3.

## 9.2 Reformulating the calendar constraint

The reformulation of an acquired CALENDAR constraint depends on several conditions, on how the input data was provided, and on the nature of the acquired resource

constraints. In the following, $n$ denotes the number of tasks.

1. [**We received a single table describing the tasks of a schedule**] In this case, each task $t$ has its own list of sorted intervals of availability or unavailability denoted $\mathcal{C}a\ell_t$; $\overline{\mathcal{C}a\ell_t}$ denotes the complement of the list of sorted periods $\mathcal{C}a\ell_t$.
   - $kind = 1 \Rightarrow \forall t \in [1, n] : s_t \in \mathcal{C}a\ell_t$,
   - $kind = 2 \Rightarrow \forall t \in [1, n] : s_t + d_t - 1 \in \mathcal{C}a\ell_t$,
   - $kind = 3 \Rightarrow \forall t \in [1, n] :$ DISJUNCTIVE$(\overline{\mathcal{C}a\ell_t} \cup \{[s_t, d_t]\})$, i.e. task $t$ is completely included in one of the intervals of the CALENDAR constraint.
   - $kind = 4 \Rightarrow \forall t \in [1, n] : s_t \in \overline{\mathcal{C}a\ell_t}$,
   - $kind = 5 \Rightarrow \forall t \in [1, n] : s_t + d_t - 1 \in \overline{\mathcal{C}a\ell_t}$,
   - $kind = 6 \Rightarrow \forall t \in [1, n] :$ DISJUNCTIVE$(\mathcal{C}a\ell_t \cup \{[s_t, d_t]\})$, i.e. task $t$ does not overlap any calendar interval.
2. [**We got two tables describing the tasks and resources involved in a schedule**]
   (a) [**The periods of availability or unavailability were defined in the tasks table, i.e. they are specific to each task**] This means that we can directly generate MiniZinc code that restricts the start or the end of each task wrt the set of intervals that describe the calendar. This case is treated in the same way as Case 1.
   (b) [**The periods of availability or unavailability were defined in the resource table, i.e. they are specific to each resource**]
      i. [**The tasks were pre-assigned to the resources**] In this context, let $r_t$ denotes the resource assigned to task $t$. This case is very similar to Case (1) as we only need to replace $\mathcal{C}a\ell_t$ by $\mathcal{C}a\ell_{r_t}$, and $\overline{\mathcal{C}a\ell_t}$ by $\overline{\mathcal{C}a\ell_{r_t}}$.
      ii. [**The tasks were not pre-assigned to the resources**] In this context, let $r_t$ denotes the resource that will be assigned to task $t$, which is a variable of the model, and let $\mathcal{R}$ (resp. $\overline{\mathcal{R}}$) denotes the rectangles derived from the intervals (resp. complement of the intervals) of the calendars attached to all the resources. Now, depending on whether we already have a DIFFN constraint using the same attributes as the CALENDAR constraint, we have two cases.

A. **[We have already a diffn constraint]** We simply add to the existing DIFFN constraint the list of fixed rectangles $\overline{\mathcal{R}}$ (resp. $\mathcal{R}$) corresponding to the different calendars when $kind = 3$ (resp. $kind = 6$).

B. **[We do not have a diffn constraint]** We need to create a DIFFN constraint for each task, so that tasks assigned to the same resource can possibly overlap.

   — $kind = 1 \Rightarrow \forall t \in [1, n] : \text{DIFFN}(\overline{\mathcal{R}} \cup \{[s_t, r_t, 1, 1]\})$,

   — $kind = 2 \Rightarrow \forall t \in [1, n] : \text{DIFFN}(\overline{\mathcal{R}} \cup \{[s_t + d_t - 1, r_t, 1, 1]\})$,

   — $kind = 3 \Rightarrow \forall t \in [1, n] : \text{DIFFN}(\overline{\mathcal{R}} \cup \{[s_t, r_t, d_t, 1]\})$,
   i.e. task $t$ is completely included in one of the intervals of the calendar of the resource to which it is assigned.

   — $kind = 4 \Rightarrow \forall t \in [1, n] : \text{DIFFN}(\mathcal{R} \cup \{[s_t, r_t, 1, 1]\})$,

   — $kind = 5 \Rightarrow \forall t \in [1, n] : \text{DIFFN}(\mathcal{R} \cup \{[s_t + d_t - 1, r_t, 1, 1]\})$,

   — $kind = 6 \Rightarrow \forall t \in [1, n] : \text{DIFFN}(\mathcal{R} \cup \{[s_t, r_t, d_t, 1]\})$,
   i.e. task $t$ does not overlap any calendar interval of the resource to which it is assigned.

## 9.3 Illustrating the constraint model for Tables 3.3 and 3.4

To demonstrate the generation of MiniZinc models, we will take the example from tables 3.3 and 3.4, run the acquisition process accordingly to 4.3 and then convert it to a MiniZinc file. The resulting constraint model consists of:

— A *foreign key* constraint to link the task and machine tables (line 50).

— A PRECEDENCE constraint (line 70) where the sum of the start time and the duration of the task must be less than or equal to the start time of its successor task.

— A constraint expressing the duration of each task as the product of the quantity produced by the task and the speed of the machine to which the task is assigned (line 53). This constraint uses columns from both the task and machine tables.

— A WITHIN constraint stating for each task the machines to which the task can be assigned (line 16).

— A DIFFN constraint ensuring that, for each machine, downtime periods and assigned tasks do not overlap (line 66).

Listing 9.1 – The model file code

```
1 % the user cannot change the values for the number of
2 % columns in the tables, but can change other values
3 % consistently.
4
5 include "table.mzn";
6 include "globals.mzn";
7
8 % declare variables and constraints for table 'task.pl'
9 set of int: ROWS1 = 1..9;
10 array[ROWS1] of int: tasks_task_id;
11 array[ROWS1] of var 0..70: tasks_start_time;
12 array[ROWS1] of int: tasks_quantity;
13 array[ROWS1] of var 8..36: tasks_duration;
14 array[ROWS1] of var {2,3,5,10}: tasks_machine;
15 array[ROWS1] of set of int: tasks_machines_set;
16 constraint forall(i in ROWS1) (tasks_machine[i] in tasks_machines_set[i]);
17 array[ROWS1] of int: tasks_successor;
18
19 set of int: COLUMNS1 = 1..6;
20 array[ROWS1,COLUMNS1] of var int: tasks;
21 constraint tasks =
22 [| tasks_task_id[1], tasks_start_time[1], tasks_quantity[1],
    tasks_duration[1], tasks_machine[1], tasks_successor[1],
23  | tasks_task_id[2], tasks_start_time[2], tasks_quantity[2],
     tasks_duration[2], tasks_machine[2], tasks_successor[2],
24  | tasks_task_id[3], tasks_start_time[3], tasks_quantity[3],
     tasks_duration[3], tasks_machine[3], tasks_successor[3],
25  | tasks_task_id[4], tasks_start_time[4], tasks_quantity[4],
     tasks_duration[4], tasks_machine[4], tasks_successor[4],
26  | tasks_task_id[5], tasks_start_time[5], tasks_quantity[5],
     tasks_duration[5], tasks_machine[5], tasks_successor[5],
27  | tasks_task_id[6], tasks_start_time[6], tasks_quantity[6],
     tasks_duration[6], tasks_machine[6], tasks_successor[6],
28  | tasks_task_id[7], tasks_start_time[7], tasks_quantity[7],
     tasks_duration[7], tasks_machine[7], tasks_successor[7],
29  | tasks_task_id[8], tasks_start_time[8], tasks_quantity[8],
     tasks_duration[8], tasks_machine[8], tasks_successor[8],
```

```
30  | tasks_task_id[9], tasks_start_time[9], tasks_quantity[9],
      tasks_duration[9], tasks_machine[9], tasks_successor[9],
31  |];
32
33 % declare variables and constraints for table 'machines.pl'
34 set of int: ROWS2 = 1..4;
35 array[ROWS2] of int: machines_machine_id;
36 array[ROWS2] of int: machines_machine_speed;
37
38 set of int: COLUMNS2 = 1..2;
39 array[ROWS2,COLUMNS2] of var int: machines;
40 constraint machines =
41 [| machines_machine_id[1], machines_machine_speed[1],
42  | machines_machine_id[2], machines_machine_speed[2],
43  | machines_machine_id[3], machines_machine_speed[3],
44  | machines_machine_id[4], machines_machine_speed[4],
45  |];
46
47
48 array[ROWS1] of var int: fk_machines;
49 % costraint to connect tables "tasks" and "machines" with a foreign key
     "fk_machines":
50 constraint forall(i in ROWS1) (machines_machine_id[fk_machines[i]] =
     tasks_machine[i]);
51
52
53 constraint forall (i in ROWS1) (tasks_duration[i] = ((1*tasks_quantity[i]*
     machines_machine_speed[fk_machines[i]])));
54
55 array[int] of int: machines_unavailability_start;
56 array[int] of int: machines_unavailability_resource;
57 array[int] of int: machines_unavailability_duration;
58 array[int] of int: machines_unavailability_one;
59
60
61 % args: diffn(List of task start times,
62 %              List of machines for each task,
63 %              List of task durations,
64 %              List of 1s)
65 % this DIFFN constraint also includes CALENDAR constraint.
66 constraint (diffn([tasks_start_time[i] | i in ROWS1] ++
```

```
    machines_unavailability_start,[tasks_machine[i] | i in ROWS1] ++
    machines_unavailability_resource, [tasks_duration[i] | i in ROWS1] ++
    machines_unavailability_duration, [1 | i in ROWS1] ++
    machines_unavailability_one));

array[ROWS1] of int: tasks_task_id_successor;
% PRECEDENCE constraint:
constraint forall(i in ROWS1 where tasks_task_id_successor[i] != -1)
    ((tasks_start_time[i]+tasks_duration[i]) <=
    (tasks_start_time[tasks_task_id_successor[i]]));
```

Listing 9.2 – The data file code

```
% The user can modify any integer or set value.
% The size of each array must be modified consistently.

tasks_task_id = [10001,10002,10003,10004,10005,10006,10007, 10008,10009];
tasks_quantity = [10,10,10,10,4,4,4,9,9];
tasks_machines_set = [{5,10},{2,3},{3,5},{10},{2,5},{2,10},{3,5},
  {5,10},{2}];
tasks_successor = [10002,10003,10004,-1,10006,10007,-1,10009,-1];
machines_machine_id = [10,2,3,5];
machines_machine_speed = [2,3,2,4];
machines_unavailability_start = [30,15,40,70,40,70];
machines_unavailability_resource = [10,2,3,3,5,5];
machines_unavailability_duration = [9,4,9,9,9,9];
machines_unavailability_one = [1,1,1,1,1,1];
tasks_task_id_successor = [2,3,4,-1,6,7,-1,9,-1];
```

If the user opens this file in MiniZinc IDE and executes it, the resulting schedule will likely be not optimised because the generated model does not have a KPI for which it can optimize a schedule. The user must add KPI to the model manually. e.g. the user wants to add a KPI that is the sum of all make-spans for all three products at the end of the model file:

Listing 9.3 – Added MiniZinc code to the model file

```
var int: max_makespan;
constraint max_makespan = max([tasks_start_time[4] +
  tasks_duration[4], tasks_start_time[7] + tasks_duration[7],
  tasks_start_time[9] + tasks_duration[9]]);
solve minimize max_makespan;
```

After solving the constraint model with the new KPI in MiniZinc IDE the user would obtain a new schedule that reduces maximum make-span from 90 to 89 (see also Table 9.1 and Figure 9.1):

Listing 9.4 – The generated schedule with an acquired constraint model presented in Listings 9.1, 9.2 and 9.3

```
1  tasks =
2  [| 10001,  0, 10, 20, 10, 10002
3   | 10002, 20, 10, 20,  3, 10003
4   | 10003, 49, 10, 20,  3, 10004
5   | 10004, 69, 10, 20, 10,    -1
6   | 10005,  0,  4, 12,  2, 10006
7   | 10006, 19,  4, 12,  2, 10007
8   | 10007, 49,  4, 16,  5,    -1
9   | 10008,  0,  9, 36,  5, 10009
10  | 10009, 36,  9, 27,  2,    -1
11  |];
```

The user can also modify input data and rerun the model. e.g. he can edit line 6 to change the list of available resources for each task or 9 to change machines' speeds and unavailability periods.

Table 9.1 – A new schedule minimizing the maximum make-span of the production schedule from the acquired constraint model presented in Listings 9.1, 9.2 and 9.3

| Task_id | Start | Quantity | Duration | Machine | Machines_set | Successor | Speed | Unavailability |
|---|---|---|---|---|---|---|---|---|
| 10001 | 0 | 10 | 20 | 10 | $\{5, 10\}$ | 10002 | 2 | $\{30..39, 50..59\}$ |
| 10002 | 20 | 10 | 30 | 3 | $\{2, 3\}$ | 10003 | 2 | $\{40..49, 70..79\}$ |
| 10003 | 49 | 10 | 20 | 3 | $\{3, 5\}$ | 10004 | 2 | $\{40..49, 70..79\}$ |
| 10004 | 69 | 10 | 20 | 10 | $\{10\}$ | $-1$ | 2 | $\{30..39, 50..59\}$ |
| 10005 | 0 | 4 | 12 | 2 | $\{2, 5\}$ | 10006 | 3 | $\{15..19\}$ |
| 10006 | 19 | 4 | 8 | 2 | $\{2, 10\}$ | 10007 | 3 | $\{15..19\}$ |
| 10007 | 49 | 4 | 8 | 5 | $\{3, 5\}$ | $-1$ | 4 | $\{40..49, 70..79\}$ |
| 10008 | 0 | 9 | 36 | 5 | $\{5, 10\}$ | 10009 | 4 | $\{40..49, 70..79\}$ |
| 10009 | 36 | 9 | 27 | 2 | $\{2\}$ | $-1$ | 3 | $\{15..19\}$ |

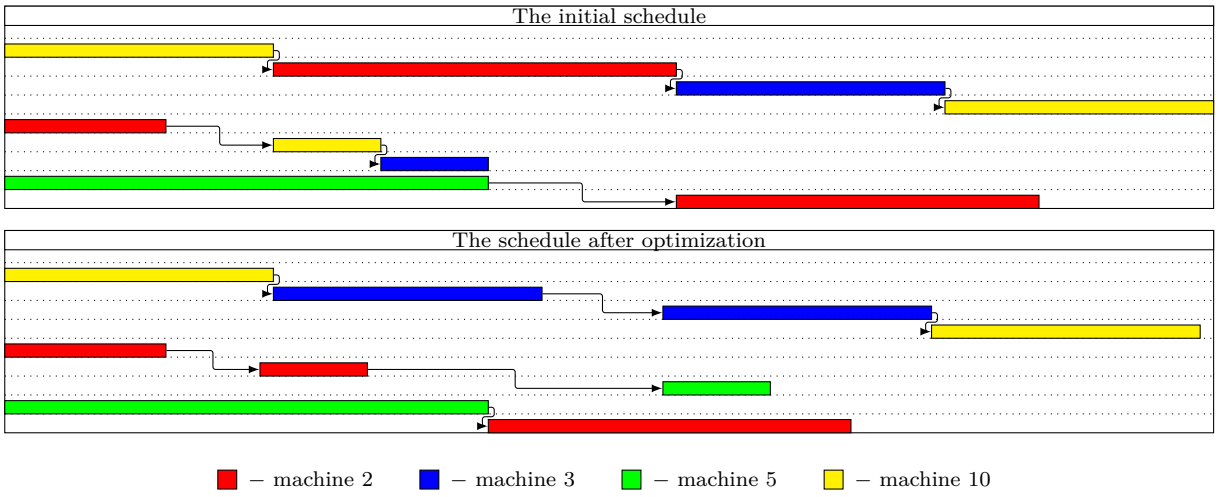



Figure 9.1 – The comparison between schedule from Tables 3.3 and 3.4 and the schedule generated from the acquired constraint model presented in Listings 9.1, 9.2 and 9.3

Chapter 10

# Evaluating the acquisition of the short-term schedules

In this section, we evaluate our model acquisition tool on a large set of instances with a variety of scheduling constraints, a varying number of tasks, and the addition or omission of data not involved in the desired model. First, we present our instance generator, which is used to generate the input data for the learning process. Secondly, we analyse the models obtained by our model acquisition tool by comparing them with the models used to generate the training data: For each type of constraint, we report the number of constraints actually learned, as well as the number of learned constraints not present in the initial model.

The source code is available at https://github.com/cquimper/MapSeekerScheduling together with the instructions on how to utilise the code to replicate the results of this chapter.

## 10.1 Generating a variety of input tables for the acquisition tool

To test the robustness of our model acquisition tool in a variety of situations, we generated 48,000 instances of schedules with variations in the following five dimensions: 1. task description, 2. temporal constraints, 3. resource constraints, 4. the introduction or absence of noisy columns, and 5. the number of tasks and resources in a schedule. For each combination of these five dimensions, we generate ten examples. The corresponding generated dataset is available in the Zenodo repository[16]. We will now describe each of these five dimensions.

1. *Different ways of describing a task.* Although each task $i$ necessarily has a start time $start\_time_i$, an end time $end\_time_i$ and a duration $duration_i$, a tasks table

may contain only two of these attributes, the missing attribute being obtained from the equality $start\_time_i + duration_i = end\_time_i$. We therefore provide two or three attributes for the tasks in the models we generate.

In the models we produce, $start\_time_i$ is an attribute that the solver must determine, while $end\_time_i$ is calculated based on $start\_time_i$ and $duration_i$. The duration of tasks is either stated explicitly if the duration attribute is present in the table, or implicit if the start time and end time attributes are both present in the table. We consider two variants for the duration attribute: The first variant is where $duration_i$ is just an input to the model and has a pre-assigned value. The second variant is where $duration_i$ is both an input used to calculate $end\_time_i$ and an output determined by a formula using pre-assigned input values, for example $duration_i = qty_i \cdot resource\_speed_i$.

As a result, we have eight ways of describing a task, namely:

(a) only the start time and duration columns are part of the table, the task duration is a pre-assigned input parameter,

(b) only the start time and end time columns are part of the table, the task duration is a pre-assigned input parameter,

(c) only the duration and end time columns are part of the table, the task duration is a pre-assigned input parameter,

(d) all three columns are present in the table, the task duration is a pre-assigned input parameter,

(e) the start time, duration and end time columns are all part of the table, and the task duration is calculated using a formula,

(f) only the start time and end time columns are part of the table, and the task duration is calculated using a formula,

(g) only the duration and end time columns are part of the table, and the task duration is calculated using formula,

(h) all three columns are present in the table, and the task duration is calculated using a formula.

2. *Different ways of expressing temporal constraints between task $i$ and its successor $j$ as described in Section 8.1, where all constants $cst$, $cst_1$, $cst_2$ are nonnegative:*

(a) no temporal constraints at all,

(b) $start_i + cst \leq start_j$,

(c) $start_i + cst \geq start_j$,

(d) $start_i + cst_1 \leq start_j$, $start_i + cst_2 \geq start_j$ $(cst_2 \neq cst_1)$,

(e) $start_i + cst = start_j$,

(f) $start_i + cst \leq end\_time_j$,

(g) $start_i + cst \geq end\_time_j$,

(h) $start_i + cst_1 \leq end\_time_j$, $start_i + cst_2 \geq end\_time_j$ $(cst_2 \neq cst_1)$,

(i) $start_i + cst = end\_time_j$,

(j) $end\_time_i + cst \leq start_j$,

(k) $end\_time_i + cst \geq start_j$,

(l) $end\_time_i + cst_1 \leq start_j$, $end\_time_i + cst_2 \geq start_j$ $(cst_2 \neq cst_1)$,

(m) $end\_time_i + cst = start_j$,

(n) $end\_time_i + cst \leq end\_time_j$,

(o) $end\_time_i + cst \geq end\_time_j$,

(p) $end\_time_i + cst_1 \leq end\_time_j$, $end\_time_i + cst_2 \geq end\_time_j$ $(cst_2 \neq cst_1)$,

(q) $end\_time_i + cst = end\_time_j$.

Note that we only generate temporal constraints that mention the start time, i.e. 2b–2m, if the start time attribute is part of the table, i.e. not in the cases 1c or 1g. Items for which the restriction only mentions the '$\leq$' (resp. '$\geq$') comparison operator correspond to maximum (resp. minimum) distance constraints, while items using only the '=' comparison operator match an exact distance constraint. In the items mentioning both '$\leq$' and '$\geq$', the two constants $cst_1$ and $cst_2$ are distinct as, otherwise, we would have an exact distance constraint. Minimum and maximum distance constraints are for instance linked to some minimum/maximum waiting time between consecutive tasks. Some constraints, e.g. Item 2b, express a pipelining constraint where consecutive tasks can partially overlap.

3. *Different ways of expressing resource scheduling constraints as described in Sections 8.2 and 8.3:*

   (a) no scheduling constraints at all,

   (b) a DISJUNCTIVE constraint for each subset of tasks using the same resource,

(c) a DIFFN constraint on all tasks, so that there is no overlap between tasks that will be assigned to the same resource,

(d) a SHIFT constraint that forces the start and end times of each task to be within the same availability period, with no gap between two consecutive availability periods,

(e) a CALENDAR constraint that forces the start and end time of each task assigned to a given resource $r$ to fall within the same period of availability of the resource $r$,

(f) a set of DISJUNCTIVE constraints and a SHIFT constraint,

(g) a DIFFN and a SHIFT constraint,

(h) a set of DISJUNCTIVE constraints and a CALENDAR constraint,

(i) a DIFFN and a CALENDAR constraint.

The combination of certain temporal and resource constraints may lead to infeasibility. For instance, in a temporal constraint of type 2c, the two corresponding tasks may overlap, which is incompatible with a DISJUNCTIVE constraint between these tasks, i.e. a constraint of type 3b. Therefore, we do not generate scheduling instances that mix the dimensions 2c, 2d, 2e, 2g, 2h, 2i, 2o, 2p, 2q, with the dimensions 3b, 3c, 3f, 3g, 3h, 3i. Note that the number of resources generated varies according to the number of tasks, as explained in Item 5.

4. *Creating noisy columns or not*:

   (a) no additional noisy columns,

   (b) three extra columns with random values standing for noise.

5. *Number of tasks and resources referenced by the schedule*:

   (a) 10 tasks and 2 resources,

   (b) 100 tasks and 10 resources,

   (c) 1,000 tasks and 20 resources,

   (d) 10,000 tasks and 100 resources.

Since any combination of 1a–1d with 2a and with 3a has no constraints, we ignore these combinations.

## 10.2 Summary of the results and detailed discussion

### 10.2.1 Summary of the results

We tested whether the model acquisition tool learns the intended model for each of the 48,000 examples. Tables 10.1–10.7 provide detailed statistics for each type of acquired constraint, which will be discussed in detail in Section 10.2.2.

The model acquisition tool captures 48% of *precedence* constraints accurately and 40% partially, 99% of *resource* constraints, 98% of *calendar* and *shift* constraints, and 90% of *functional* constraints exactly.

The number of rows in the input tables affects the quality of the *functional* (see tables 10.4–10.5) and *precedence* (see Table 10.1) constraints: the lower the number of rows, the greater the chance of getting an incorrect constraint. In all cases, however, the results stabilise from a hundred tasks upwards.

If the number of rows is small, there is a slight risk of acquiring incorrect *resource*, *shift* and *calendar* constraints, especially if there are noisy columns in the tables (see tables 10.2–10.3 and 10.6–10.7).

On average, using SICStus Prolog 4.6.0 on a 2019 MacBook Pro Core i7 with 6 cores and 16 GB, it took 4 seconds to acquire a MiniZinc model from a schedule with 10 tasks, 10 seconds for 100 tasks, 21 seconds for 1,000 tasks and 27 seconds for 10,000 tasks.

Table 10.1 – Statistics of acquisition of PRECEDENCE constraints

| Input tables | Ctrs to find | Exact ctrs, % | Partial ctrs, % | Wrong ctrs, % |
|---|---|---|---|---|
| 10 rows | 10960 | 32,92 | 53,22 | 11,74 |
| 100 rows | 10960 | 52,64 | 37,12 | 2,35 |
| 1000 rows | 10960 | 53,68 | 34,89 | 1,11 |
| 10000 rows | 10960 | 53,74 | 34,52 | 0,55 |
| w/ noise cols | 21920 | 47,95 | 41,05 | 3,04 |
| w/o noise cols | 21920 | 48,53 | 38,83 | 4,84 |
| all tables | 43840 | 48,24 | 39,94 | 3,94 |

### 10.2.2 Detailed discussion

Within this section, we refer to a specific example by using its unique identifier used in the Zenodo repository[16]. For each type of constraint, we analyse the discrepancy

Table 10.2 – Statistics of acquisition of DISJUNCTIVE constraints

| Input tables | Ctrs to find | Exact ctrs, % | Partial ctrs, % | Wrong ctrs |
|---|---|---|---|---|
| 10 rows | 2880 | 96,11 | 0 | 20 |
| 100 rows | 2880 | 99,51 | 0 | 0 |
| 1000 rows | 2880 | 99,72 | 0 | 0 |
| 10000 rows | 2880 | 99,97 | 0 | 0 |
| w/ noise cols | 5760 | 99,2 | 0 | 20 |
| w/o noise cols | 5760 | 98,42 | 0 | 0 |
| all tables | 11520 | 98,82 | 0 | 20 |

Table 10.3 – Statistics of acquisition of DIFFN constraints

| Input tables | Ctrs to find | Exact ctrs, % | Partial ctrs, % | Wrong ctrs |
|---|---|---|---|---|
| 10 rows | 2880 | 97,68 | 0,34 | 0 |
| 100 rows | 2880 | 98,61 | 0 | 0 |
| 1000 rows | 2880 | 98,47 | 0 | 0 |
| 10000 rows | 2880 | 98,82 | 0 | 0 |
| w/ noise cols | 5760 | 98,4 | 0,07 | 0 |
| w/o noise cols | 5760 | 98,38 | 0,1 | 0 |
| all tables | 11520 | 98,39 | 0,09 | 0 |

Table 10.4 – Statistics of acquisition of formulae for *end_time*

| Input tables | Ctrs to find | Exact ctrs, % | Partial ctrs, % | Wrong ctrs, % |
|---|---|---|---|---|
| 10 rows | 5920 | 81,1 | 12,06 | 0,02 |
| 100 rows | 5920 | 93,18 | 5,95 | 0 |
| 1000 rows | 5920 | 93,48 | 5,59 | 0 |
| 10000 rows | 5920 | 93,36 | 5,84 | 0 |
| w/ noise cols | 11840 | 90,41 | 6,79 | 0,01 |
| w/o noise cols | 11840 | 90,14 | 7,93 | 0 |
| all tables | 23680 | 90,28 | 7,36 | 0 |

between the expected constraints and the acquired constraints.

#### 10.2.2.1 Precedence constraints

In Table 10.1, each acquired PRECEDENCE constraint is classified in one of the following categories:

Table 10.5 – Statistics of acquisition of formulae for *duration*

| Input tables | Ctrs to find | Exact ctrs, % | Partial ctrs, % | Wrong ctrs, % |
|---|---|---|---|---|
| 10 rows | 4060 | 66,43 | 17,86 | 1,87 |
| 100 rows | 4060 | 97,68 | 0 | 0 |
| 1000 rows | 4060 | 97,8 | 0 | 0 |
| 10000 rows | 4060 | 97,73 | 0 | 0 |
| w/ noise cols | 8120 | 91,1 | 3,78 | 0,19 |
| w/o noise cols | 8120 | 88,73 | 5,14 | 0,73 |
| all tables | 16240 | 89,91 | 4,46 | 0,47 |

Table 10.6 – Statistics of acquisition of SHIFT constraints

| Input tables | Ctrs to find | Exact ctrs, % | Partial ctrs, % |
|---|---|---|---|
| 10 rows | 3960 | 95,63 | 2,14 |
| 100 rows | 3960 | 98,89 | 0,2 |
| 1000 rows | 3960 | 99,37 | 0,1 |
| 10000 rows | 3960 | 99,52 | 0,05 |
| w/ noise cols | 7920 | 98,78 | 0,5 |
| w/o noise cols | 7920 | 97,92 | 0,74 |
| all tables | 15840 | 98,35 | 0,62 |

Table 10.7 – Statistics of acquisition of CALENDAR constraints

| Input tables | Ctrs to find | Exact ctrs, % | Partial ctrs, % |
|---|---|---|---|
| 10 rows | 3960 | 94,47 | 0,1 |
| 100 rows | 3960 | 98,86 | 0 |
| 1000 rows | 3960 | 99,6 | 0 |
| 10000 rows | 3960 | 99,95 | 0 |
| w/ noise cols | 7920 | 98,89 | 0,05 |
| w/o noise cols | 7920 | 97,55 | 0 |
| all tables | 15840 | 98,22 | 0,03 |

— *An exact constraint*, i.e. a constraint that is correctly identified.

— *A partial constraint*, if only part of the constraint is correctly identified, i.e. if one of the following statements is true:

  — A coefficient or comparison operator in a PRECEDENCE constraint is incorrectly identified;

e.g. for the table 'schedule_robustness_a_b_a_a_d_9.pl', instead of the constraint

$$start_i + 5 \leq start_j$$

the model acquisition tool learned

$$start_i + 5 = start_j \tag{10.1}$$

— The PRECEDENCE constraint uses the start time instead of the end time, or vice versa;

e.g. for the table 'schedule_robustness_b_k_b_a_a_0.pl', instead of the constraint

$$end_i + 3 \geq start_j \tag{10.2}$$

the model acquisition tool learned

$$start_i + 17 \leq end_j \tag{10.3}$$

— The model acquisition tool learns either one constraint instead of two, or two constraints instead of one;

e.g. for table 'schedule_robustness_b_b_c_b_a_4.pl', instead of a constraint

$$start_i + 2 \leq start_j \tag{10.4}$$

the model acquisition tool learned

$$start_i + 2 \leq start_j \wedge start_i + 56 \geq start_j \tag{10.5}$$

A partial constraint may be a stricter version of an intended constraint, such as Constraint (10.1), or, depending on task durations, it may or may not contradict the intended constraint; e.g. Constraint (10.3) will not contradict Constraint (10.2): i.e. if each task duration is greater than or equal to 7 then Constraint (10.3) can be rewritten as $end_i + 3 \leq start_j$.

— *A wrong constraint*, i.e. either:

— The constraint uses an unexpected attribute;

e.g. for the table 'schedule_robustness_c_o_a_a_a_0.pl', instead of a con-

straint

$$end_i + 3 \geq end_j \tag{10.6}$$

the model acquisition tool learned

$$end_i + 17 \leq end_j + duration_j \tag{10.7}$$

— The constraint was not expected for this table.

#### 10.2.2.2 Resource, calendar and shift constraints

Each acquired *resource* constraint, i.e. DISJUNCTIVE and DIFFN, and each *calendar* or *shift* constraint is classified in one of the following categories:

— *An exact constraint*, i.e. a constraint that is correctly identified.

— *A partial constraint*, where the resource (resp. calendar or shift) constraint mentions the proper resource (resp. calendar) attribute, but one or more temporal attributes, i.e. start time, end time or duration, are used in the wrong place.

— *A wrong constraint*, if one of the following statements is true:

  — The resource or the calendar attribute was not correctly identified.

  — A non-temporal attribute is used in place of a temporal attribute.

  — The constraint is not supposed to be acquired: for example, if the tool learns a set of DISJUNCTIVE constraints instead of a single DIFFN constraint because it mistakenly assumes that tasks are pre-assigned, we consider that these are incorrectly learned DISJUNCTIVE constraints.

#### 10.2.2.3 Functional constraints

Each acquired functional constraint is classified in one of the following categories:

— *An exact constraint*, i.e. a constraint that is correctly identified.

— *A partial constraint*, when the tool learns a formula with only a subset of the expected input attributes, which can happen if some unused attributes are constants; e.g. for the table 'schedule_robustness_d_f_d_b_ a_8.pl', instead of the constraint

$$end_i = start_i + duration_i \tag{10.8}$$

the model acquisition tool learned

$$end_i = duration_i + 1, \tag{10.9}$$

because all start times are equal to one.

— *A wrong constraint*, when the model acquisition tool learns a formula from a different family or a formula that uses attributes that are not expected;
e.g. for the table 'schedule_robustness_e_k_i_a_a_1.pl', instead of the constraint

$$duration_i = quantity_i \cdot speed_i + 7 \tag{10.10}$$

the model acquisition tool learned

$$duration_i = 11 + \neg[quantity_i = 4] \tag{10.11}$$

# Conclusion

The aim of this thesis was to propose a variety of ML techniques related to the acquisition of constraint models from error-free data in the context of two use cases.

*The first use case* is the search for conjectures for eight different combinatorial objects which was established in [1]. Two primary contributions are

— A new learning bias, *Boolean-arithmetic equations* (BAE). To acquire a BAE, a constraint model is created and then solved.

The proposed constraint model consists of three parts:

— *The core model*. The model is necessary to select which Boolean-arithmetic conditions (BAC) would be part of the final BAE for a given logical operator.

— *Constraints enhancing the core model*. These constraints are aimed at improving the performance of the core model by imposing symmetry-breaking constraints and by limiting the search tree to only non-simplifiable solutions.

— *The anti-rewriting constraints*. Their goal is to prevent the generation of simplifiable BAE. These constraints are generated automatically by a separate constraint program and stored into a reusable database. The anti-rewriting constraints are categorised into several families to simplify their generation.

The new learning bias is also used to construct case formulae by creating sub-tables from the main table.

— An idea of *formula synthesis* which allows combining multiple learning biases into one conjecture. The way this is done is by analysing a given table to see different ways it can be decomposed rather than using a brute-force approach. Four decomposition techniques were developed which allow decomposing any given table into sub-tables. We then can apply any learning bias or any decomposition technique on each sub-table recursively. This helps to acquire complex conjectures in a modular manner.

Both contributions were evaluated on the dataset containing information about sharp bounds for eight combinatorial objects. The evaluation showed that both contributions help acquire a large number of missing conjectures for the dataset.

*The second use case* is the acquisition of a constraint model from a single valid short-term production schedule (STPS). The goal is to acquire the relevant scheduling constraints such as functional constraints between columns, temporal constraints within chains of tasks, constraints put on the usage of resources and constraints which align each tasks with the provided calendar. The acquired constraints are then combined into a single model for future use. The thesis takes the tool developed in [1] and proposes various enhancements to accommodate this use case:

— The automated search for *candidate primary and foreign keys* for cases when none are provided.

— The *ranking of functional dependencies* to select the most likely candidate functional dependencies which can produce a valid functional constraint.

— The acquisition of *temporal constraints*. These constraints ensure that each subsequent tasks within a given chain are aligned properly against each other.

— The acquisition of *resource constraints*. These constraints ensure that tasks assign for each resource do not intersect. To ensure this, global constraints DISJUNCTIVE and DIFFN are used.

— The proposal and the acquisition of *calendar and shift constraints*, i.e. the constraints which ensure that each task is placed accordingly to a given calendar, be it periods of availability or unavailability of an individual resource or a given shift schedule.

— The module which converts acquired constraints into a single constraint model in a MiniZinc file. This constraint model then can later be used together with a variety of solvers to produce new schedules, optimised wrt different criteria.

This thesis also provides an evaluation of the acquisition quality of the developed tool against a dataset of 48000 schedules. The evaluation showed good results where the vast majority of constraints was identified correctly.

In conclusion, this thesis achieves our initial goals: it shows that ML approaches specifically designed to work with error-free data can provide good results in many situations.

Future work may include exploring the following topics:

— acquring new scheduling constraints to the system such as conditional scheduling constraints;

— coming with new decomposition techniques, e.g. when we can infer specific properties on the shape of the conjecture;

— acquiring constraints when it is known or expected to have a very limited number of errors.

# BIBLIOGRAPHY


1. Beldiceanu, N., Cheukam-Ngouonou, J., Douence, R., Gindullin, R. & Quimper, C.-G., *Acquiring Maps of Interrelated Conjectures on Sharp Bounds in 28th International Conference on Principles and Practice of Constraint Programming, CP 2022, July 31 to August 8, 2022, Haifa, Israel* (ed Solnon, C.) **235** (Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022), 6:1–6:18, `https://doi.org/10.4230/LIPIcs.CP.2022.6` (cit. on pp. 14, 17, 19, 22, 24–26, 41, 46–50, 73, 120, 127–131, 142, 171, 172).
2. Freuder, E. C. & Mackworth, A. K., *in Foundations of artificial intelligence* 13–27 (Elsevier, 2006) (cit. on pp. 14, 22).
3. Beldiceanu, N., Carlsson, M. & Rampon, J.-X., *Global constraint catalog, (revision a)* 2012 (cit. on pp. 15, 23, 31, 35, 59, 99, 127).
4. Carlier, J., The one-machine sequencing problem, *European Journal of Operational Research* **11,** 42–47 (1982) (cit. on pp. 15, 23, 61, 150).
5. Baptiste, P., Pape, C. L. & Nuijten, W., *Constraint-Based Scheduling: Applying Constraint Programming to Scheduling Problems* (Kluwer, 2012) (cit. on pp. 15, 23, 61).
6. Shaw, P., *Using constraint programming and local search methods to solve vehicle routing problems in International conference on principles and practice of constraint programming* (1998), 417–431 (cit. on pp. 15, 23).
7. Barahona, P. & Krippahl, L., Constraint programming in structural bioinformatics, *Constraints* **13,** 3–20 (2008) (cit. on pp. 15, 23).
8. Peng, X. & Solnon, C., *Using Canonical Codes to Efficiently Solve the Benzenoid Generation Problem with Constraint Programming in 29th International Conference on Principles and Practice of Constraint Programming (CP 2023)* (2023) (cit. on pp. 15, 23).
9. Gotlieb, A., TCAS software verification using constraint programming, *The Knowledge Engineering Review* **27,** 343–360 (2012) (cit. on pp. 15, 23).

10. Stuckey, P. J., Marriott, K. & Tack, G., *The MiniZinc Handbook* 2020, https://www.minizinc.org/doc-2.6.4/en/index.html (cit. on pp. 15, 23, 31, 59, 61).

11. Wikimedia Commons, *Inductive Bias* https://en.wikipedia.org/wiki/Inductive_bias, Accessed August 4th, 2023, 2003 (cit. on pp. 16, 23, 49).

12. Mitchell, T. M., The need for biases in learning generalizations (1980) (cit. on pp. 16, 23).

13. Wikimedia Commons, *Tree (graph theory)* https://en.wikipedia.org/wiki/Tree_(graph_theory), Accessed January 2th, 2024, 2004 (cit. on pp. 16, 24).

14. Cheukam-Ngouonou, J., Gindullin, R., Beldiceanu, N., Douence, R. & Quimper, C.-G., *Proving Conjectures Acquired by Composing Multiple Biases* 2023, arXiv: 2312.08990 [cs.AI] (cit. on pp. 17, 19, 24, 26, 53, 142).

15. Gindullin, R., Beldiceanu, N., Cheukam-Ngouonou, J., Douence, R. & Quimper, C.-G., *Boolean-Arithmetic Equations: Acquisition and Uses in Integration of AI and OR Techniques in Constraint Programming - 20th International Conference, CPAIOR 2023, Nice, France, May 29-June 1, 2023, Proceedings* (ed Andre, C.) (Springer, 2023) (cit. on pp. 18–20, 25–27, 39, 47, 49, 128).

16. Gindullin, R. & Beldiceanu, N., *A set of positive examples consisting of short-term schedules for testing the acquisition of MiniZinc scheduling models* (Zenodo, Sept. 2023), https://doi.org/10.5281/zenodo.8340001 (cit. on pp. 18, 26, 161, 165).

17. Gindullin, R., Beldiceanu, N., Cheukam-Ngouonou, J., Douence, R. & Quimper, C.-G., *Composing Biases by Using CP to Decompose Minimal Functional Dependencies for Acquiring Complex Formulae (accepted) in Thirty-Eighth AAAI Conference on Artificial Intelligence (AAAI-24), Vancouver, Canada, February 20-27, 2024, Proceedings* (2024) (cit. on pp. 19, 20, 26, 27).

18. Gindullin, R., Beldiceanu, N., Cheukam-Ngouonou, J., Douence, R. & Quimper, C.-G., Boolean-Arithmetic Equations: Acquisition and Uses (submitted), *Constraint Programming* (cit. on pp. 19, 20, 26, 27).

19. Bessiere, C., Coletta, R., Koriche, F. & O'Sullivan, B., *A SAT-based version space algorithm for acquiring constraint satisfaction problems in Machine Learning: ECML 2005: 16th European Conference on Machine Learning, Porto, Portugal, October 3-7, 2005. Proceedings 16* (2005), 23–34 (cit. on p. 30).

20. Zweben, M. *et al.,* Learning to improve constraint-based scheduling, *Artificial Intelligence* **58,** 271–296 (1992) (cit. on p. 30).

21. Padmanabhuni, S., You, J.-H. & Ghose, A., *A framework for learning constraints in Proceedings of the Fourth Pacific Rim International Conference on Artificial Intelligence (PRICAI-96) Workshop on Induction of Complex Representations* (1996) (cit. on pp. 30, 31).

22. Haussler, D., Quantifying inductive bias: AI learning algorithms and Valiant's learning framework, *Artificial intelligence* **36,** 177–221 (1988) (cit. on p. 31).

23. Freuder, E. C., *Constraint Acquisition in Artificial Intelligence, Automated Reasoning, and Symbolic Computation, Joint International Conferences, AISC 2002 and Calculemus 2002, Marseille, France, July 1-5, 2002, Proceedings* (eds Calmet, J., Benhamou, B., Caprotti, O., Henocque, L. & Sorge, V.) **2385** (Springer, 2002), 1, https://doi.org/10.1007/3-540-45470-5%5C_1 (cit. on p. 31).

24. Bessière, C., Koriche, F., Lazaar, N. & O'Sullivan, B., Constraint acquisition, *Artif. Intell.* **244,** 315–342, https://doi.org/10.1016/j.artint.2015.08.001 (2017) (cit. on pp. 31, 33).

25. Bessiere, C. *et al.,* Learning constraints through partial queries, *Artificial Intelligence* **319,** 103896 (2023) (cit. on pp. 31, 35).

26. Sammut, C. & Banerji, R. B., Learning concepts by asking questions, *Machine learning: An artificial intelligence approach* **2,** 167–192 (1986) (cit. on p. 31).

27. Beldiceanu, N., Carlsson, M., Demassey, S. & Petit, T., Global Constraint Catalogue: Past, Present and Future, *Constraints An Int. J.* **12,** 21–62, https://doi.org/10.1007/s10601-006-9010-8 (2007) (cit. on p. 31).

28. Nethercote, N. *et al., MiniZinc: Towards a Standard CP Modelling Language in Principles and Practice of Constraint Programming - CP 2007, 13th International Conference, CP 2007, Providence, RI, USA, September 23-27, 2007, Proceedings* (ed Bessière, C.) **4741** (Springer, 2007), 529–543, https://doi.org/10.1007/978-3-540-74970-7%5C_38 (cit. on p. 31).

29. Freuder, E. C., *Active learning for constraint satisfaction in Active Learning. AAAI-95 Fall Symposium Series, Working Notes* (1995), 34–35 (cit. on p. 31).

30. Freuder, E. C. & Wallace, R. J., *Suggestion strategies for constraint-based matchmaker agents in International Conference on Principles and Practice of Constraint Programming* (1998), 192–204 (cit. on pp. 31, 32).

31. O'Sullivan, B., Freuder, E. C. & O'Connell, S., *Interactive constraint acquisition in Proceedings of Workshop on User-Interaction in Constraint Processing at the CP-2001* (2001) (cit. on p. 32).

32. Mitchell, T., Concept learning and the general-to-specific ordering, *Machine Learning,* 20–51 (1997) (cit. on p. 32).

33. Hirsh, H., Generalizing version spaces, *Machine learning* **17,** 5–46 (1994) (cit. on p. 32).

34. O'Connell, S., O'Sullivan, B. & Freuder, E. C., *A study of query generation strategies for interactive constraint acquisition in Applications and Science in Soft Computing* (2004), 225–232 (cit. on p. 32).

35. O'Connell, S., O'Sullivan, B. & Freuder, E. C., *Timid acquisition of constraint satisfaction problems in Proceedings of the 2005 ACM symposium on Applied computing* (2005), 404–408 (cit. on p. 32).

36. Berwick, R. C., Learning from positive-only examples: The subset principle and three case studies, *Machine Learning: An Artificial Intelligence Approach* **2,** 625–645 (1986) (cit. on p. 32).

37. Ayoun, D., The subset principle in second language acquisition, *Applied Psycholinguistics* **17,** 185–213 (1996) (cit. on p. 32).

38. Coletta, R. *et al., Constraint acquisition as semi-automatic modeling in International Conference on Innovative Techniques and Applications of Artificial Intelligence* (2003), 111–124 (cit. on p. 33).

39. Bessiere, C., Coletta, R., Freuder, E. C. & O'Sullivan, B., *Leveraging the learning power of examples in automated constraint acquisition in International Conference on Principles and Practice of Constraint Programming* (2004), 123–137 (cit. on p. 33).

40. Bessiere, C., Coletta, R., O'Sullivan, B. & Paulin, M., *Query-driven constraint acquisition in IJCAI'07: International Joint Conference on Artificial Intelligence* (2007), 44–49 (cit. on p. 33).

41. Angluin, D., Queries revisited, *Theoretical Computer Science* **313,** 175–194 (2004) (cit. on p. 33).

42. O'Sullivan, B., *Automated modelling and solving in constraint programming in Proceedings of the AAAI Conference on Artificial Intelligence* **24** (2010), 1493–1497 (cit. on p. 33).

43. Bessiere, C. *et al., Constraint acquisition via partial queries in IJCAI: International Joint Conference on Artificial Intelligence* (2013), 475–481 (cit. on p. 34).

44. Bessiere, C. *et al., Boosting Constraint Acquisition via Generalization Queries. in ECAI* (2014), 99–104 (cit. on p. 34).

45. Daoudi, A., Mechqrane, Y., Bessiere, C., Lazaar, N. & Bouyakhf, E. H., *Constraint acquisition using recommendation queries in IJCAI: International Joint Conference on Artificial Intelligence* (2016), 720–726 (cit. on p. 34).

46. Arcangioli, R., Bessiere, C. & Lazaar, N., *Multiple constraint aquisition in IJCAI: International Joint Conference on Artificial Intelligence* (2016), 698–704 (cit. on p. 34).

47. Tsouros, D. C., Stergiou, K. & Sarigiannidis, P. G., *Efficient methods for constraint acquisition in Principles and Practice of Constraint Programming: 24th International Conference, CP 2018, Lille, France, August 27-31, 2018, Proceedings 24* (2018), 373–388 (cit. on p. 34).

48. Tsouros, D. C., Stergiou, K. & Bessiere, C., *Structure-driven multiple constraint acquisition in Principles and Practice of Constraint Programming: 25th International Conference, CP 2019, Stamford, CT, USA, September 30–October 4, 2019, Proceedings 25* (2019), 709–725 (cit. on p. 35).

49. Bessiere, C. *et al.,* Partial queries for constraint acquisition, *arXiv preprint arXiv:2003.06649* (2020) (cit. on p. 35).

50. Angluin, D., Frazier, M. & Pitt, L., Learning conjunctions of Horn clauses, *Machine Learning* **9,** 147–164 (1992) (cit. on p. 35).

51. De Raedt, L., Passerini, A. & Teso, S., *Learning constraints from examples in Proceedings of the AAAI Conference on Artificial Intelligence* **32** (2018) (cit. on pp. 35, 37).

52. Rossi, F. & Sperduti, A., Acquiring both constraint and solution preferences in interactive constraint systems, *Constraints* **9,** 311–332 (2004) (cit. on p. 35).

53. Beldiceanu, N. & Simonis, H., *A constraint seeker: Finding and ranking global constraints from examples in Principles and Practice of Constraint Programming–CP 2011: 17th International Conference, CP 2011, Perugia, Italy, September 12-16, 2011. Proceedings 17* (2011), 12–26 (cit. on pp. 35, 45).

54. Beldiceanu, N. & Simonis, H., *Learning Structured Constraint Models: a First Attempt in The 22nd Irish Conference on Artificial Intelligence and Cognitive Science, AICS'11* (2011) (cit. on p. 35).

55. Beldiceanu, N. & Simonis, H., *A Model Seeker: Extracting Global Constraint Models from Positive Examples in Principles and Practice of Constraint Programming - 18th International Conference, CP 2012, Québec City, QC, Canada, October 8-12, 2012. Proceedings* (ed Milano, M.) **7514** (Springer, 2012), 141–157, `https://doi.org/10.1007/978-3-642-33558-7%5C_13` (cit. on p. 35).

56. Beldiceanu, N. & Simonis, H., Modelseeker: Extracting global constraint models from positive examples, *Data Mining and Constraint Programming: Foundations of a Cross-Disciplinary Approach,* 77–95 (2016) (cit. on p. 35).

57. Picard-Cantin, É., Bouchard, M., Quimper, C.-G. & Sweeney, J., *Learning parameters for the sequence constraint from solutions in Principles and Practice of Constraint Programming: 22nd International Conference, CP 2016, Toulouse, France, September 5-9, 2016, Proceedings 22* (2016), 405–420 (cit. on p. 37).

58. Picard-Cantin, É., Bouchard, M., Quimper, C.-G. & Sweeney, J., *Learning the Parameters of Global Constraints Using Branch-and-Bound in Principles and Practice of Constraint Programming: 23rd International Conference, CP 2017, Melbourne, VIC, Australia, August 28–September 1, 2017, Proceedings 23* (2017), 512–528 (cit. on p. 37).

59. Gregory, P. & Lindsay, A., *Domain model acquisition in domains with action costs in Proceedings of the International Conference on Automated Planning and Scheduling* **26** (2016), 149–157 (cit. on p. 37).

60. Senderovich, A., Booth, K. E. C. & Beck, J. C., *Learning Scheduling Models from Event Data in Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling, ICAPS 2018, Berkeley, CA, USA, July 11-15, 2019* (eds Benton, J., Lipovetzky, N., Onaindia, E., Smith, D. E. & Srivastava, S.) (AAAI Press, 2019), 401–409, https://ojs.aaai.org/index.php/ICAPS/article/view/3504 (cit. on pp. 37, 56).

61. Van der Aalst, W. M., Petri net based scheduling, *Operations-Research-Spektrum* **18,** 219–229 (1996) (cit. on p. 37).

62. Mizoguchi, F., *Constraint-directed generalization for learning spatial relations in Proceedings of International Workshop on Inductive Logic Programming* (1992) (cit. on p. 37).

63. Mizoguchi, F. & Ohwada, H., Constrained relative least general generalization for inducing constraint logic programs, *New Generation Computing* **13,** 335–368 (1995) (cit. on p. 37).

64. Kawamura, T., *Towards inductive generalization in constraint logic programs in IJCAI-93 Workshop on Inductive Logic Programming* (1993) (cit. on p. 37).

65. Lallouet, A., Lopez, M., Martin, L. & Vrain, C., *On learning constraint problems in 2010 22nd IEEE International Conference on Tools with Artificial Intelligence* **1** (2010), 45–52 (cit. on p. 37).

66. Paulus, A., Rolínek, M., Musil, V., Amos, B. & Martius, G., *CombOptNet: Fit the Right NP-Hard Problem by Learning Integer Programming Constraints* 2021, arXiv: 2105.02343 [cs.LG] (cit. on p. 37).

67. Pawlak, T. P. & Krawiec, K., Automatic synthesis of constraints from examples using mixed integer linear programming, *European Journal of Operational Research* **261,** 1141–1157 (2017) (cit. on p. 37).

68. Pawlak, T. P. & Krawiec, K., Synthesis of Constraints for Mathematical Programming With One-Class Genetic Programming, *IEEE Trans. Evol. Comput.* **23,** 117–129, https://doi.org/10.1109/TEVC.2018.2835565 (2019) (cit. on p. 37).

69. Kolb, S., Paramonov, S., Guns, T. & De Raedt, L., Learning constraints in spreadsheets and tabular data, *Machine Learning* **106,** 1441–1468 (2017) (cit. on p. 37).

70. Paramonov, S., Kolb, S., Guns, T. & Raedt, L. D., *TaCLe: Learning Constraints in Tabular Data in Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, CIKM 2017, Singapore, November 06 - 10, 2017* (eds Lim, E.-P. *et al.*) (ACM, 2017), 2511–2514, https://doi.org/10.1145/3132847.3133193 (cit. on pp. 37, 49).

71. Kumar, M., Teso, S., De Causmaecker, P. & De Raedt, L., *Automating Personnel Rostering by Learning Constraints Using Tensors in 2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI)* (2019), 697–704 (cit. on p. 37).

72. Kumar, M., Kolb, S. & Guns, T., *Learning constraint programming models from data using generate-and-aggregate in 28th International Conference on Principles and Practice of Constraint Programming (CP 2022)* (2022) (cit. on p. 37).

73. Coulombe, C. & Quimper, C.-G., *Constraint Acquisition Based on Solution Counting in 28th International Conference on Principles and Practice of Constraint Programming (CP 2022)* (2022) (cit. on p. 38).

74. Lallouet, A., Legtchenko, A., Monfroy, E. & Ed-Dbali, A., Solver learning for predicting changes in dynamic constraint satisfaction problems, *Changes* **4** (2004) (cit. on p. 38).

75. Lallouet, A. & Legtchenko, A., *Two contributions of constraint programming to machine learning in European Conference on Machine Learning* (2005), 617–624 (cit. on p. 38).

76. Vu, X.-H. & O'Sullivan, B., A unifying framework for generalized constraint acquisition, *International Journal on Artificial Intelligence Tools* **17,** 803–833 (2008) (cit. on p. 38).

77. Belaid, M.-B., Belmecheri, N., Gotlieb, A., Lazaar, N. & Spieker, H., *Geqca: Generic qualitative constraint acquisition in Proceedings of the AAAI Conference on Artificial Intelligence* **36** (2022), 3690–3697 (cit. on p. 38).

78. Tsouros, D., Berden, S. & Guns, T., Guided Bottom-Up Interactive Constraint Acquisition, *arXiv preprint arXiv:2307.06126* (2023) (cit. on p. 38).

79. Blum, A., *Relevant examples and relevant features: Thoughts from computational learning theory in AAAI Fall Symposium on 'Relevance* **5** (1994), 1 (cit. on p. 38).

80. Blum, A. L. & Langley, P., Selection of relevant features and examples in machine learning, *Artificial intelligence* **97,** 245–271 (1997) (cit. on p. 38).

81. Guyon, I. & Elisseeff, A., *in Feature extraction* 1–25 (Springer, 2006) (cit. on p. 38).

82. Nguifo, E. M. & Njiwoua, P., *in Feature Extraction, Construction and Selection* 205–218 (Springer, 1998) (cit. on p. 38).

83. Mossel, E., O'Donnell, R. & Servedio, R. A., Learning functions of k relevant variables, *Journal of Computer and System Sciences* **69,** 421–434 (2004) (cit. on p. 38).

84. Forman, G. & Kirshenbaum, E., *Extremely fast text feature extraction for classification and indexing in Proceedings of the 17th ACM conference on Information and knowledge management* (2008), 1221–1230 (cit. on p. 38).

85. Mutlu, E. C. & Oghaz, T. A., Review on graph feature learning and feature extraction techniques for link prediction, *arXiv preprint arXiv:1901.03425* (2019) (cit. on p. 38).

86. Yu, J., Ignatiev, A., Stuckey, P. J. & Le Bodic, P., *Computing optimal decision sets with SAT in International Conference on Principles and Practice of Constraint Programming* (2020), 952–970 (cit. on p. 38).

87. Mereani, F. & Howe, J. M., *Exact and approximate rule extraction from neural networks with Boolean features in Proceedings of the 11th International Joint Conference on Computational Intelligence* (2019), 424–433 (cit. on p. 38).

88. Jakobovic, D., Picek, S., Martins, M. S. & Wagner, M., Toward more efficient heuristic construction of Boolean functions, *Applied Soft Computing* **107,** 107327 (2021) (cit. on p. 38).

89. Golia, P., Slivovsky, F., Roy, S. & Meel, K. S., *Engineering an efficient Boolean functional synthesis engine in 2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)* (2021), 1–9 (cit. on p. 38).

90. Jun, S., Lee, S. & Chun, H., Learning dispatching rules using random forest in flexible job shop scheduling problems, *International Journal of Production Research* **57,** 3290–3310 (2019) (cit. on p. 39).

91. Aung, M. H. *et al., Comparing analytical decision support models through Boolean rule extraction: A case study of ovarian tumour malignancy in International Symposium on Neural Networks* (2007), 1177–1186 (cit. on pp. 39, 121).

92. Barbareschi, M., Barone, S. & Mazzocca, N., Advancing synthesis of decision tree-based multiple classifier systems: an approximate computing case study, *Knowledge and Information Systems* **63,** 1577–1596 (2021) (cit. on pp. 39, 121).

93. Kosman, E., Kolchinsky, I. & Schuster, A., *Mining Logical Arithmetic Expressions From Proper Representations in Proceedings of the 2022 SIAM International Conference on Data Mining (SDM)* (2022), 621–629 (cit. on pp. 39, 40, 121).

94. Yang, H., Rudin, C. & Seltzer, M., *Scalable Bayesian rule lists in International conference on machine learning* (2017), 3921–3930 (cit. on p. 39).

95. Au, W.-H. & Chan, K. C., Mining fuzzy association rules in a bank-account database, *IEEE Transactions on Fuzzy Systems* **11,** 238–248 (2003) (cit. on p. 39).

96. Lambert-Torres, G., *Application of rough sets in power system control center data mining in 2002 IEEE Power Engineering Society Winter Meeting. Conference Proceedings (Cat. No. 02CH37309)* **1** (2002), 627–631 (cit. on p. 39).

97. Alur, R., Singh, R., Fisman, D. & Solar-Lezama, A., Search-based program synthesis, *Commun. ACM* **61,** 84–93, `https://doi.org/10.1145/3208071` (2018) (cit. on p. 39).

98. Tanemura, K., Tachibana, Y., Tokuni, Y., Manabe, H. & Miyadera, R., *Application of Generic Programming to Unsolved Mathematical Problems in 2022 IEEE 11th Global Conference on Consumer Electronics (GCCE)* (2022), 845–849 (cit. on p. 39).

99. McConaghy, T., FFX: Fast, scalable, deterministic symbolic regression technology, *Genetic Programming Theory and Practice IX,* 235–260 (2011) (cit. on p. 39).

100. Kotsiantis, S. B., Decision trees: a recent overview, *Artificial Intelligence Review* **39,** 261–283 (2013) (cit. on pp. 39, 40).

101. Brodley, C. E. & Utgoff, P. E., Multivariate decision trees, *Machine learning* **19,** 45–77 (1995) (cit. on p. 39).

102. Yildiz, C. & Alpaydin, E., Omnivariate decision trees, *IEEE Transactions on Neural Networks* **12,** 1539–1546 (2001) (cit. on pp. 39, 40).

103. Quinlan, J. R., *C4. 5: programs for machine learning* (Morgan Kaufmann Publishers Inc., 1993) (cit. on pp. 39, 41).

104. Breiman, L., Friedman, J., Olshen, R. & Stone, C., *Classification and Regression Trees* (Wadsworth, 1984) (cit. on p. 39).

105. Shafer, J., Agrawal, R., Mehta, M., *et al., SPRINT: A scalable parallel classifier for data mining in Vldb* **96** (1996), 544–555 (cit. on p. 39).

106. Mehta, M., Agrawal, R. & Rissanen, J., *SLIQ: A fast scalable classifier for data mining in Advances in Database Technology—EDBT'96: 5th International Conference on Extending Database Technology Avignon, France, March 25–29, 1996 Proceedings 5* (1996), 18–32 (cit. on p. 39).

107. Gehrke, J., Ramakrishnan, R. & Ganti, V., RainForest—a framework for fast decision tree construction of large datasets, *Data Mining and Knowledge Discovery* **4,** 127–162 (2000) (cit. on p. 39).

108. Ittner, A. & Schlosser, M., *Discovery of Relevant New Features by Generating Non-Linear Decision Trees. in KDD* **1996** (1996), 108–113 (cit. on p. 40).

109. Altınçay, H., Decision trees using model ensemble-based nodes, *Pattern recognition* **40,** 3540–3551 (2007) (cit. on p. 40).

110. Djukova, E. & Peskov, N., A classification algorithm based on the complete decision tree, *Pattern Recognition and Image Analysis* **17,** 363–367 (2007) (cit. on p. 40).

111. Tharwat, A., Gaber, T., Ibrahim, A. & Hassanien, A. E., Linear discriminant analysis: A detailed tutorial, *AI communications* **30,** 169–190 (2017) (cit. on p. 40).

112. Sok, H. K., Ooi, M. P.-L. & Kuang, Y. C., Sparse alternating decision tree, *Pattern Recognition Letters* **60,** 57–64 (2015) (cit. on p. 40).

113. Sok, H. K., Ooi, M. P.-L., Kuang, Y. C. & Demidenko, S., Multivariate alternating decision trees, *Pattern Recognition* **50,** 195–209 (2016) (cit. on p. 40).

114. Wang, X., Chen, B., Qian, G. & Ye, F., On the optimization of fuzzy decision trees, *Fuzzy Sets and Systems* **112,** 117–125, ISSN: 0165-0114, https://www.sciencedirect.com/science/article/pii/S0165011497003862 (2000) (cit. on p. 40).

115. Olaru, C. & Wehenkel, L., A complete fuzzy decision tree technique, *Fuzzy Sets and Systems* **138,** 221–254, ISSN: 0165-0114, https://www.sciencedirect.com/science/article/pii/S0165011403000897 (2003) (cit. on p. 40).

116. Mugambi, E. M., Hunter, A., Oatley, G. & Kennedy, L., *Polynomial-fuzzy decision tree structures for classifying medical data in Research and Development in Intelligent Systems XX: Proceedings of AI2003, the Twenty-third SGAI International Conference on Innovative Techniques and Applications of Artificial Intelligence* (2004), 155–167 (cit. on p. 40).

117. Chandra, B. & Varghese, P. P., Fuzzifying Gini Index based decision trees, *Expert Systems with Applications* **36,** 8549–8559 (2009) (cit. on p. 40).

118. Costa, V. G. & Pedreira, C. E., Recent advances in decision trees: An updated survey, *Artificial Intelligence Review* **56,** 4765–4800 (2023) (cit. on p. 40).

119. Carrizosa, E., Molero-Rio, C. & Romero Morales, D., Mathematical optimization in classification and regression trees, *Top* **29,** 5–33 (2021) (cit. on p. 40).

120. Murthy, S. K., Automatic Construction of Decision Trees from Data: A Multi-Disciplinary Survey, *Data Mining and Knowledge Discovery* **2,** 345–389 (1999) (cit. on p. 40).

121. Lenat, D. B., The ubiquity of discovery, *Artificial Intelligence* **9,** 257–285 (1977) (cit. on p. 40).

122. Lenat, D. B., EURISKO: a program that learns new heuristics and domain concepts: the nature of heuristics III: program design and results, *Artificial intelligence* **21,** 61–98 (1983) (cit. on p. 40).

123. Lenat, D. B. & Brown, J. S., Why AM and EURISKO appear to work, *Artificial intelligence* **23,** 269–294 (1984) (cit. on p. 40).

124. Valiant, L. G., A theory of the learnable, *Communications of the ACM* **27,** 1134–1142 (1984) (cit. on p. 40).

125. Brigham, R. & Dutton, R., Ingrid: A software tool for extremal graph theory research, *Congr. Numer* **39,** 337–352 (1983) (cit. on p. 40).

126. Dutton, R. D., Brigham, R. C. & Gomez, F., INGRID: A graph invariant manipulator, *Journal of symbolic computation* **7,** 163–177 (1989) (cit. on p. 40).

127. Brigham, R. C. & Dutton, R. D., A compilation of relations between graph invariants, *Networks* **15,** 73–107 (1985) (cit. on p. 40).

128. Brigham, R. C. & Dutton, R. D., A compilation of relations between graph invariants—supplement I, *Networks* **21,** 421–455 (1991) (cit. on p. 40).

129. Fajtlowicz, S., *in Annals of Discrete Mathematics* 113–118 (Elsevier, 1988) (cit. on pp. 40, 41).

130. Larson, C. E. & Van Cleemput, N., Automated conjecturing I: Fajtlowicz's Dalmatian heuristic revisited, *Artificial Intelligence* **231,** 17–38 (2016) (cit. on pp. 40, 49).

131. Larson, C. E. & Van Cleemput, N., Automated conjecturing iii, *Annals of Mathematics and Artificial Intelligence* **81,** 315–327 (2017) (cit. on p. 40).

132. Hansen, P. & Caporossi, G., Autographix: An automated system for finding conjectures in graph theory, *Electronic Notes in Discrete Mathematics* **5,** 158–161 (2000) (cit. on pp. 40, 41).

133. Caporossi, G. & Hansen, P., Variable neighborhood search for extremal graphs: 1 The AutoGraphiX system, *Discrete Mathematics* **212,** 29–44 (2000) (cit. on p. 40).

134. Aouchiche, M., Caporossi, G., Hansen, P. & Laffay, M., AutoGraphiX: a survey, *Electronic Notes in Discrete Mathematics* **22,** 515–520 (2005) (cit. on p. 41).

135. Colton, S., Meier, A., Sorge, V. & McCasland, R., *Automatic Generation of Classification Theorems for Finite Algebras in International Joint Conference on Automated Reasoning* (2004), 400–414 (cit. on p. 41).

136. McCune, W., Mace4 reference manual and guide, *arXiv preprint cs/0310055* (2003) (cit. on p. 41).

137. Weidenbach, C. *et al., Spass Version 2.0 in Automated Deduction—CADE-18: 18th International Conference on Automated Deduction Copenhagen, Denmark, July 27–30, 2002 Proceedings 18* (2002), 275–279 (cit. on p. 41).

138. Davies, A. *et al.,* Advancing mathematics by guiding human intuition with AI, *Nature* **600,** 70–74, `https://doi.org/10.1038/s41586-021-04086-x` (2021) (cit. on p. 41).

139. Brooks, J. P., Edwards, D. J., Larson, C. E. & Van Cleemput, N., Conjecturing-based computational discovery of patterns in data, *arXiv preprint arXiv:2011.11576* (2020) (cit. on p. 41).

140. Beldiceanu, N., *Global Constraints as Graph Properties on a Structured Network of Elementary Constraints of the Same Type in Principles and Practice of Constraint Programming - CP 2000, 6th International Conference, Singapore, September 18-21, 2000, Proceedings* (ed Dechter, R.) **1894** (Springer, 2000), 52–66, ISBN: 3-540-41053-8, https://doi.org/10.1007/3-540-45349-0%5C_6 (cit. on pp. 45, 127).

141. Knuth, D., *Art of Computer Programming, Volume 4, Generating All Trees, pp. 461-462* 461–462 (Addison-Wesley, 2006) (cit. on pp. 45, 127).

142. Pachet, F. & Roy, P., *Automatic Generation of Music Programs in Principles and Practice of Constraint Programming - CP'99, 5th International Conference, Alexandria, Virginia, USA, October 11-14, 1999, Proceedings* (ed Jaffar, J.) **1713** (Springer, 1999), 331–345, https://doi.org/10.1007/978-3-540-48085-3%5C_24 (cit. on pp. 45, 127).

143. Bessière, C. *et al., The Balance Constraint Family in Principles and Practice of Constraint Programming - 20th International Conference, CP 2014, Lyon, France, September 8-12, 2014. Proceedings* (ed O'Sullivan, B.) **8656** (Springer, 2014), 174–189 (cit. on pp. 45, 127).

144. Pesant, G., *A Filtering Algorithm for the Stretch Constraint in Principles and Practice of Constraint Programming — CP 2001* (ed Walsh, T.) (Springer Berlin Heidelberg, Berlin, Heidelberg, 2001), 183–195, ISBN: 978-3-540-45578-3 (cit. on pp. 45, 127).

145. Todorovski, L., *in Encyclopedia of Machine Learning* (eds Sammut, C. & Webb, G. I.) 327–330 (Springer US, Boston, MA, 2010), ISBN: 978-0-387-30164-8 (cit. on p. 49).

146. Schelldorfer, J. & Wuthrich, M. V., Nesting classical actuarial models into neural networks, *Available at SSRN 3320525* (2019) (cit. on p. 49).

147. Gulwani, S., *Automating string processing in spreadsheets using input-output examples in Proceedings of the 38th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2011, Austin, TX, USA, January 26-28, 2011* (eds Ball, T. & Sagiv, M.) (ACM, 2011), 317–330, https://doi.org/10.1145/1926385.1926423 (cit. on p. 49).

148. Gulwani, S., Harris, W. R. & Singh, R., Spreadsheet data manipulation using examples, *Commun. ACM* **55,** 97–105, https://doi.org/10.1145/2240236.2240260 (2012) (cit. on p. 49).

149. Srivastava, S., Gulwani, S. & Foster, J. S., Template-based program verification and program synthesis, *Int. J. Softw. Tools Technol. Transf.* **15,** 497–518, https://doi.org/10.1007/s10009-012-0223-4 (2013) (cit. on p. 49).

150. Aouchiche, M., Caporossi, G., Hansen, P. & Laffay, M., AutoGraphiX: a survey, *Electron. Notes Discret. Math.* **22,** 515–520, https://doi.org/10.1016/j.endm.2005.06.090 (2005) (cit. on p. 49).

151. Udrescu, S.-M. *et al., AI Feynman 2.0: Pareto-optimal symbolic regression exploiting graph modularity in Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual* (eds Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M.-F. & Lin, H.-T.) (2020), https://proceedings.neurips.cc/paper/2020/hash/33a854e247155d590883b93bca53848a-Abstract.html (cit. on p. 50).

152. Brence, J., Todorovski, L. & Džeroski, S., Probabilistic grammars for equation discovery, *Knowledge-Based Systems* **224,** 107077, ISSN: 0950-7051, https://www.sciencedirect.com/science/article/pii/S0950705121003403 (2021) (cit. on pp. 50, 92).

153. Ligęza, A. *et al., Explainable Artificial Intelligence. Model Discovery with Constraint Programming in International Symposium on Methodologies for Intelligent Systems* (2020), 171–191 (cit. on p. 50).

154. Hansen, P. & Caporossi, G., AutoGraphiX: An Automated System for Finding Conjectures in Graph Theory, *Electron. Notes Discret. Math.* **5,** 158–161, https://doi.org/10.1016/S1571-0653(05)80151-9 (2000) (cit. on p. 50).

155. Stadtler, H., *in Modern Concepts of the Theory of the Firm: Managing Enterprises of the New Economy* 285–300 (Springer, 2004) (cit. on p. 55).

156. De Man, J. C. & Strandhagen, J. O., Spreadsheet application still dominates enterprise resource planning and advanced planning systems, *Ifac-Papersonline* **51,** 1224–1229 (2018) (cit. on p. 55).

157. Cplex, I. I., V12. 1: User's Manual for CPLEX, *International Business Machines Corporation* **46,** 157 (2009) (cit. on p. 56).

158. Perron, L. & Furnon, V., *OR-Tools* version v9.6, Computer software, Google, 2023, https://developers.google.com/optimization/ (cit. on p. 56).

159. Patil, R. J., Using ensemble and metaheuristics learning principles with artificial neural networks to improve due date prediction performance, *International Journal of Production Research* **46,** 6009–6027 (Nov. 2008) (cit. on p. 56).

160. Schneckenreither, M., Haeussler, S. & Gerhold, C., Order release planning with predictive lead times: a machine learning approach, *International Journal of Production Research* **59,** 3285–3303 (Dec. 2020) (cit. on p. 56).

161. Belov, G., Stuckey, P. J., Tack, G. & Wallace, M., *Improved Linearization of Constraint Programming Models in Principles and Practice of Constraint Programming - 22nd International Conference, CP 2016, Toulouse, France, September 5-9, 2016, Proceedings* (ed Rueher, M.) **9892** (Springer, 2016), 49–65, https://doi.org/10.1007/978-3-319-44953-1%5C_4 (cit. on p. 56).

162. Listl, F. G. *et al., Decision Support on the Shop Floor Using Digital Twins: Architecture and Functional Components for Simulation-Based Assistance in Advances in Production Management Systems. Artificial Intelligence for Sustainable and Resilient Production Systems: IFIP WG 5.7 International Conference, APMS 2021, Nantes, France, September 5–9, 2021, Proceedings, Part I* (2021), 284–292 (cit. on p. 57).

163. Beldiceanu, N. & Contejean, E., Introducing Global Constraints in CHIP, *Mathl. Comput. Modelling* **20,** 97–123 (1994) (cit. on pp. 61, 96, 98, 150).

164. Beldiceanu, N., *Parallel machine scheduling with calendar rules in 6th International Workshop on Project Management and Scheduling (PMS'98),Istanbul, Turkey* (1998) (cit. on p. 62).

165. Kreter, S., Schutt, A. & Stuckey, P. J., *Modeling and Solving Project Scheduling with Calendars in Principles and Practice of Constraint Programming - 21st International Conference, CP 2015, Cork, Ireland, August 31 - September 4, 2015, Proceedings* (ed Pesant, G.) **9255** (Springer, 2015), 262–278, https://doi.org/10.1007/978-3-319-23219-5%5C_19 (cit. on p. 62).

166. Carlsson, M. & Mildner, P., SICStus Prolog - The first 25 years, *Theory Pract. Log. Program.* **12,** 35–66, https://doi.org/10.1017/S1471068411000482 (2012) (cit. on p. 68).

167. Carlsson, M., Ottosson, G. & Carlson, B., *An Open-Ended Finite Domain Constraint Solver in Programming Languages: Implementations, Logics, and Programs, 9th International Symposium, PLILP'97, Including a Special Trach on Declarative Programming Languages in Education, Southampton, UK, September 3-5, 1997, Proceedings* (eds Glaser, H., Hartel, P. H. & Kuchen, H.) **1292** (Springer, 1997), 191–206, https://doi.org/10.1007/BFb0033845 (cit. on p. 68).

168. Wei, Z. & Link, S., *Discovery and ranking of functional dependencies in 2019 IEEE 35th International Conference on Data Engineering (ICDE)* (2019), 1526–1537 (cit. on pp. 70, 87).

169. Beldiceanu, N., Carlsson, M. & Petit, T., *Deriving Filtering Algorithms from Constraint Checkers in Principles and Practice of Constraint Programming - CP 2004, 10th International Conference, CP 2004, Toronto, Canada, September 27 - October 1, 2004, Proceedings* (ed Wallace, M.) **3258** (Springer, 2004), 107–122, https://doi.org/10.1007/978-3-540-30201-8%5C_11 (cit. on pp. 81, 104).

170. *AdventureWorks sample databases* Accessed June 5th, 2023. https://learn.microsoft.com/en-us/sql/samples/adventureworks-install-configure?view=sql-server-ver16 (cit. on p. 85).

171. Hartmann, S., Link, S. & Trinh, T., Constraint acquisition for Entity-Relationship models, *Data & Knowledge Engineering* **68,** 1128–1155 (2009) (cit. on p. 86).

172. Yeh, P. Z. & Puri, C. A., *Discovering conditional functional dependencies to detect data inconsistencies in Proceedings of the Fifth International Workshop on Quality in Databases at VLDB2010* **176** (2010) (cit. on p. 86).

173. Discovery of genuine functional dependencies from relational data with missing values, *Proceedings of the VLDB Endowment* **11,** 880–892 (2018) (cit. on p. 86).

174. Zhang, Y., Guo, Z. & Rekatsinas, T., *A statistical perspective on discovering functional dependencies in noisy data in Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data* (2020), 861–876 (cit. on p. 87).

175. Jean-Charles, R. E., Generalized arc consistency for global cardinality constraint, *American Association for Artificial Intelligence (AAAI 1996),* 209–215 (1996) (cit. on p. 97).

176. Hong, H., *Simple solution formula construction in cylindrical algebraic decomposition based quantifier elimination in Papers from the international symposium on Symbolic and algebraic computation* (1992), 177–188 (cit. on p. 107).

177. Kingsford, C. & Salzberg, S. L., What are decision trees?, *Nature biotechnology* **26,** 1011–1013 (2008) (cit. on p. 121).

178. Medina, R. & Nourine, L., *Conditional functional dependencies: An FCA point of view in Formal Concept Analysis: 8th International Conference, ICFCA 2010, Agadir, Morocco, March 15-18, 2010. Proceedings 8* (2010), 161–176 (cit. on p. 122).

179. Aggoun, A. & Beldiceanu, N., Extending CHIP in order to Solve Complex Scheduling and Placement Problems, *Mathl. Comput. Modelling* **17,** 57–73 (1993) (cit. on p. 150).

180. Beldiceanu, N. & Carlsson, M., *Sweep as a Generic Pruning Technique Applied to the Non-Overlapping Rectangles Constraints in Principles and Practice of Constraint Programming (CP'2001)* (ed Walsh, T.) **2239,** Preprint available as SICS Tech Report T2001-13, `http://soda.swedish-ict.se/2384/2/SICS-T--2001-13--SE.pdf` (Springer-Verlag, 2001), 377–391 (cit. on p. 150).

COLLEGES SCIENCES
BRETAGNE POUR L'INGENIEUR
LOIRE ET LE NUMERIQUE

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

**Titre :** Apprentissage de modèles à contraintes concis à partir de données sans erreurs: études sur l'acquisition d'équations arithmétiques booléennes et de modèles d'ordonnancement à court terme

**Mot clés :** programmation par contraintes, acquisition de modeles, limites nettes

**Résumé :** Utilisant la programmation logique par contrainte, l'objectif de cette thèse est de développer plusieurs techniques d'acquisition de contraintes pour les situations où nous disposons de données sans erreur. De telles situations rendent la majorité des techniques de ML inutilisables et de nouvelles approches sont nécessaires.

Les techniques d'acquisition de contraintes proposées sont appliquées à deux cas d'utilisation : la recherche de nouvelles conjectures de limites fortes pour huit objets combinatoires et l'acquisition de contraintes à partir d'un calendrier de production à court terme unique et valide.

Les contributions de la thèse comprennent (i) un modèle de contrainte pour acquérir des expressions booléennes-arithmétiques à partir de données, (ii) une base de données générée automatiquement de contraintes anti-réécriture qui empêchent la génération d'équations booléennes-arithmétiques simplifiables, (iii) un certain nombre de techniques de synthèse de formules qui peuvent acquérir une formule unique combinant plusieurs biais d'apprentissage, (iv) l'acquisition d'une variété de contraintes d'ordonnancement telles que les contraintes temporelles, de ressources, de calendrier et d'équipes, et dans ce dernier cas (v) la génération d'un modèle d'ordonnancement MiniZinc.

**Title:** Learning concise constraint models from error-free data: studies on learning Boolean-arithmetic equations and short-term scheduling models

**Keywords:** constraint programming, model acquisition, sharp bounds

**Abstract:** Using constraint logic programming, the goal of this thesis is to develop several constraint acquisition techniques for the situations where we have error-free data. Such situations render majority of ML techniques unusable and new approaches are required.

The proposed constraint acquisition techniques are applied for two use cases: search for new sharp bounds conjectures for eight combinatorial objects and the constraint acquisition from a single valid short-term production schedule.

The contributions of the thesis include (i) a constraint model to acquire Boolean-arithmetic expressions from data, (ii) an automatically generated database of anti-rewriting constraints that prevent the generation of simplifiable Boolean-arithmetic equations, (iii) a number of formulae synthesis techniques which can acquire a single formula combining several learning biases, (iv) the acquisition of a variety of scheduling constraints such as temporal, resource, calendar and shift constraints, and in this later case (v) the generation of a MiniZinc scheduling model.