# Constraint Programming Models for Conceptual Clustering: Application to an ERP Configuration Problem

Maxime Chabert

N° d'ordre : 2018LYSEI118

**THÈSE DE DOCTORAT DE L'UNIVERSITÉ DE LYON**
**opérée au sein de l'INSA de Lyon**

**École doctorale n° 512**
**InfoMaths**

**Spécialité de doctorat :**
**Informatique**

**Soutenue publiquement le 18/12/2018, par**

Maxime CHABERT

---

# Constraint Programming Models for Conceptual Clustering : Application to an ERP Configuration Problem

---

Devant le jury composé de :

| | | | |
|---|---|---|---|
| Mme Thi-Bich-Hanh DAO | Maître de Conférences HDR | Université d'Orléans | Rapporteur |
| M. Christian BESSIÈRE | Directeur de Recherche | CNRS | Rapporteur |
| Mme Élisa FROMONT | Professeure des Universités | Université de Rennes 1 | Examinatrice |
| Mme Valérie BOTTA-GENOULAZ | Professeure des Universités | INSA-LYON | Examinatrice |
| M. Christian SCHULTE | Professeur des Universités | KTH Royal Institute of Technology | Examinateur |
| Mme Christine SOLNON | Professeure des Universités | INSA-LYON | Directrice de thèse |
| M. Pierre-Antoine CHAMPIN | Maître de Conférences HDR | Université Claude Bernard Lyon 1 | Co-directeur de thèse |
| Mme Amélie CORDIER | Maître de Conférences | Hoomano | Co-directeur de thèse |
| M. Justinian OPRESCU | Docteur en Informatique | Infologic | Invité |

# Département FEDORA - INSA Lyon - Écoles doctorales
# Quinquennal 2016 - 2020

| SIGLE | ÉCOLE DOCTORALE | NOM ET COORD. RESPONSABLE |
|---|---|---|
| CHIMIE | **CHIMIE DE LYON**<br>http://www.edchimie-lyon.fr<br>Sec. : Renée EL MELHEM<br>Bât. Blaise PASCAL, 3e étage<br>secretariat@edchimie-lyon.fr<br>INSA : R. GOURDON | M. Stéphane DANIELE<br>Institut de recherches sur la catalyse et l'environnement de Lyon<br>IRCELYON-UMR 5256<br>Équipe CDFA<br>2 Avenue Albert EINSTEIN<br>69 626 Villeurbanne CEDEX<br>directeur@edchimie-lyon.fr |
| E.E.A | **ÉLECTRONIQUE, ÉLECTROTECHNIQUE, AUTOMATIQUE**<br>http://edeea.ec-lyon.fr<br>Sec. : M.C. HAVGOUDOUKIAN<br>ecole-doctorale.eea@ec-lyon.fr | M. Gérard SCORLETTI<br>École Centrale de Lyon<br>36 Avenue Guy DE COLLONGUE<br>69 134 Écully<br>☎ 04.72.18.60.97 🖷 04.78.43.37.17<br>gerard.scorletti@ec-lyon.fr |
| E2M2 | **ÉVOLUTION, ÉCOSYSTÈME, MICROBIOLOGIE, MODÉLISATION**<br>http://e2m2.universite-lyon.fr<br>Sec. : Sylvie ROBERJOT<br>Bât. Atrium, UCB Lyon 1<br>☎ 04.72.44.83.62<br>INSA : H. CHARLES<br>secretariat.e2m2@univ-lyon1.fr | M. Fabrice CORDEY<br>CNRS UMR 5276 Lab. de géologie de Lyon<br>Université Claude Bernard Lyon 1<br>Bât. Géode<br>2 Rue Raphaël DUBOIS<br>69 622 Villeurbanne CEDEX<br>☎ 06.07.53.89.13<br>cordey@univ-lyon1.fr |
| EDISS | **INTERDISCIPLINAIRE SCIENCES-SANTÉ**<br>http://www.ediss-lyon.fr<br>Sec. : Sylvie ROBERJOT<br>Bât. Atrium, UCB Lyon 1<br>☎ 04.72.44.83.62<br>INSA : M. LAGARDE<br>secretariat.ediss@univ-lyon1.fr | Mme Emmanuelle CANET-SOULAS<br>INSERM U1060, CarMeN lab, Univ. Lyon 1<br>Bâtiment IMBL<br>11 Avenue Jean CAPELLE INSA de Lyon<br>69 621 Villeurbanne<br>☎ 04.72.68.49.09 🖷 04.72.68.49.16<br>emmanuelle.canet@univ-lyon1.fr |
| INFOMATHS | **INFORMATIQUE ET MATHÉMATIQUES**<br>http://edinfomaths.universite-lyon.fr<br>Sec. : Renée EL MELHEM<br>Bât. Blaise PASCAL, 3e étage<br>☎ 04.72.43.80.46 🖷 04.72.43.16.87<br>infomaths@univ-lyon1.fr | M. Luca ZAMBONI<br>Bât. Braconnier<br>43 Boulevard du 11 novembre 1918<br>69 622 Villeurbanne CEDEX<br>☎ 04.26.23.45.52<br>zamboni@maths.univ-lyon1.fr |
| MATÉRIAUX | **MATÉRIAUX DE LYON**<br>http://ed34.universite-lyon.fr<br>Sec. : Marion COMBE<br>☎ 04.72.43.71.70 🖷 04.72.43.87.12<br>Bât. Direction<br>ed.materiaux@insa-lyon.fr | M. Jean-Yves BUFFIÈRE<br>INSA de Lyon<br>MATEIS<br>Bât. Saint-Exupéry<br>7 Avenue Jean CAPELLE<br>69 621 Villeurbanne CEDEX<br>☎ 04.72.43.71.70 🖷 04.72.43.85.28<br>ed.materiaux@insa-lyon.fr |
| MEGA | **MÉCANIQUE, ÉNERGÉTIQUE, GÉNIE CIVIL, ACOUSTIQUE**<br>http://edmega.universite-lyon.fr<br>Sec. : Marion COMBE<br>☎ 04.72.43.71.70 🖷 04.72.43.87.12<br>Bât. Direction<br>mega@insa-lyon.fr | M. Philippe BOISSE<br>INSA de Lyon<br>Laboratoire LAMCOS<br>Bâtiment Jacquard<br>25 bis Avenue Jean CAPELLE<br>69 621 Villeurbanne CEDEX<br>☎ 04.72.43.71.70 🖷 04.72.43.72.37<br>philippe.boisse@insa-lyon.fr |
| SCSO | **SCSO** *<br>http://ed483.univ-lyon2.fr<br>Sec. : Viviane POLSINELLI<br>Brigitte DUBOIS<br>INSA : J.Y. TOUSSAINT<br>☎ 04.78.69.72.76<br>viviane.polsinelli@univ-lyon2.fr | M. Christian MONTES<br>Université Lyon 2<br>86 Rue Pasteur<br>69 365 Lyon CEDEX 07<br>christian.montes@univ-lyon2.fr |

* ScSo : Histoire, Géographie, Aménagement, Urbanisme, Archéologie, Science politique, Sociologie, Anthropologie.

# Remerciements

Une des nombreuses vertus de cette thèse est qu'elle m'a appris à dire "nous" au lieu de "je" car je me suis très vite rendu compte que je n'irais pas très loin tout seul. Ce "nous" désigne bien sûr les personnes avec qui je travaille au quotidien comme mes encadrants mais aussi l'ensemble des personnes qui par une simple discussion, si simple soit-elle, font avancer notre réflexion et nous inspirent. Ce "nous" semble très paradoxal avec la solitude que j'ai pu parfois ressentir face à cette thèse mais l'aboutissement est bien un travail collectif qui s'est enrichi grâce à de nombreuses personnes que je veux remercier chaleureusement.

Je tiens tout d'abord à remercier Infologic qui a été à l'origine de ce projet et particulièrement mon père qui a toujours su avoir des idées avant-gardistes et ambitieuses pour l'entreprise. Ce projet de thèse m'a permis de découvrir l'entreprise sous un angle que je ne connaissais pas. Je remercie l'ensemble des personnes qui ont cru dans ce projet ou qui m'ont permis d'avancer.

Je remercie Christine, mon encadrante de thèse, qui n'a pas hésité une seconde pour se lancer et s'investir dans ce projet. Merci pour ton exigence constante qui m'a toujours poussé à aller plus loin, même quand c'était difficile. Merci également pour ton soutien et ton investissement dans les moments clés de cette thèse.

Je remercie mes co-encadrants Amélie et Pierre-Antoine pour leurs idées toujours pertinentes. Même si je me suis éloigné de vos domaines de recherche, vos remarques ont toujours été justes et constructives pour faire avancer ce projet.

Je remercie le jury qui me fait l'honneur de lire et évaluer cette thèse. J'ai la chance d'avoir le jury que je souhaitais et j'en suis très heureux. Un merci particulier aux rapporteurs d'avoir pris le temps de me faire des rapports riches et pertinents.

Je remercie bien évidemment ma famille qui m'a toujours soutenu dans ce que je fais. Merci de me donner une stabilité indispensable pour pouvoir avancer sereinement dans mes études et dans ma vie. C'est en grande partie grâce à vous, par mon éducation et mes valeurs, que je termine cette thèse aujourd'hui.

Un merci très spécial à Perrine, ma compagne, qui a su me supporter et me soutenir durant cette dernière année si particulière.

Merci à mes amis qui ont toujours été là quand j'en avais besoin.

Je tiens également à remercier les doctorants et chercheurs rencontrés durant ces trois années avec qui j'ai pu partager de très bons moments, notamment durant les différentes conférences auxquelles j'ai eu la chance de participer.

Cette thèse m'a permis d'apprendre beaucoup de choses sur les plans scientifiques et professionnels bien évidemment mais aussi sur moi même. Cette thèse aura été une expérience d'une richesse incroyable sur laquelle je pourrai m'appuyer dans ma vie professionnelle et personnelle.

# Contents

# Résumé en français

## Motivations

Les progiciels de gestion intégrés ou ERP (Enterprise Resource Planning) sont essentiels pour les entreprises industrielles pour structurer, automatiser et piloter leurs processus métiers afin d'améliorer leur compétitivité. Les ERP sont les leaders incontestés des systèmes d'information des entreprises industrielles. Cependant, ce sont des logiciels génériques conçus pour être utilisés par une grande diversité d'entreprises qui ont des processus métiers et des besoins différents. Par conséquent, les ERP proposent de nombreuses options de paramétrage pour s'adapter aux différentes entreprises. Le déploiement d'un ERP est le processus qui consiste à paramétrer l'ERP selon les besoins d'une entreprise : il détermine le comportement exact de l'ERP et les processus métiers supportés par le système dans une entreprise. La réussite du déploiement d'un ERP dans une entreprise dépend souvent de facteurs organisationnels comme la gestion du projet, la préparation au changement ou la formation du personnel sur le nouveau système. Cependant, des facteurs opérationnels comme l'élicitation des besoins d'une entreprise ou le paramétrage d'un ERP sont des sujets de recherche constants et ne sont toujours pas maîtrisés.

Infologic est une entreprise française qui développe et installe son propre ERP, appelé *Copilote*, spécialisé dans l'agro-alimentaire. La principale force de *Copilote* est son hyper-adaptabilité qui permet de supporter un large éventail de besoins métiers. *Copilote* a plusieurs dizaines de milliers de paramètres qui sont utilisés pour adapter le plus finement possible le système aux besoins d'une entreprise. Cependant, cette flexibilité rend le système long et difficile à installer : une profonde connaissance de *Copilote* et de ses fonctionnalités est nécessaire pour déployer le système dans une entreprise. Les intégrateurs (les employés d'Infologic qui déploient *Copilote*) doivent être formés au moins 8 mois pour pouvoir être autonomes sur *Copilote*. En effet, Infologic ne capitalise pas sur les déploiements antérieurs de *Copilote* et les compétences des intégrateurs dépendent donc directement de leur expérience sur le système. C'est pourquoi réduire la complexité du déploiement de *Copilote* est un enjeu crucial pour Infologic qui doit intégrer rapidement et efficacement de nouveaux intégrateurs pour répondre à la demande de nouveaux clients qui augmente considérablement.

Dans cette thèse, nous étudions le processus de déploiement de *Copilote* pour comprendre les principales difficultés rencontrées par Infologic. Nous nous concentrons en particulier sur l'étape de paramétrage et sur l'architecture des paramètres de *Copilote*. Une difficulté majeure rencontrée pendant le paramétrage d'un besoin métier est de trouver rapidement les bons paramètres à modifier, *i.e.* les paramètres qui impactent la logique métier du système à ajuster, et ensuite d'assigner les bonnes valeurs à ces paramètres. Notre défi est de fournir un outil qui assiste les intégrateurs lorsqu'ils paramètrent *Copilote* selon les besoins d'une entreprise.

Nous proposons de réduire le temps et la complexité du paramétrage en réutilisant des morceaux de paramétrage. En effet, comme le souligne Daneva dans [Era+15], les entreprises ont souvent des besoins similaires, et le taux de besoins précédemment rencontrés dans une autre entreprise peut être considérablement élevé dans certains cas. Plus précisément, nous proposons une approche qui vise à extraire un catalogue de morceaux de paramétrage à partir des paramétrages existants de *Copilote*, et à associer chaque morceau de paramétrage avec le besoin métier auquel il correspond. Ce catalogue permet de capitaliser sur l'expérience de tous les intégrateurs et de réutiliser cette expérience pour les prochains déploiements de *Copilote*.

La partie la plus difficile de notre approche est d'utiliser des techniques de fouille de données pour identifier des morceaux de paramétrages qui correspondent à des besoins métiers. Nous proposons d'utiliser l'*Analyse de Concept Formel* (FCA) et plus précisément le clustering conceptuel pour regrouper les paramétrages qui correspondent au même besoin métier et extraire le morceau de paramétrage correspondant. Nous proposons d'utiliser la programmation par contraintes (PPC) pour faire du clustering conceptuel. En effet, les experts de *Copilote* ne peuvent pas définir une mesure idéale pour évaluer la pertinence d'un morceau de paramétrage. Le cadre déclaratif de la PPC nous permet de facilement intégrer les retours des experts par le biais de nouvelles contraintes et de critères à optimiser afin d'améliorer progressivement la pertinence des morceaux de paramétrages extraits.

## Objectifs et contributions

Notre objectif est d'extraire un catalogue de morceaux de paramétrage à partir des déploiements passés d'un ERP pour simplifier les prochains déploiements du système. Pour atteindre cet objectif, une première contribution est l'introduction d'une nouvelle abstraction appelée *carte d'unités fonctionnelles* qui permet de structurer le processus de déploiement. Une seconde contribution est la définition d'un processus interactif de fouille de données pour construire le catalogue, où des experts peuvent interactivement affiner leurs contraintes et critères utilisés pour extraire des morceaux de paramétrages. Une troisième contribution est la définition de nouveaux modèles PPC et de contraintes globales pour résoudre efficacement les problèmes de clustering conceptuel sous contraintes présents durant le processus de fouille.

Nous avons utilisé *Copilote* pour illustrer et évaluer nos contributions. Cependant, le processus que nous proposons pour extraire un catalogue de paramétrages à partir des déploiements passés est générique, et nous pensons qu'il pourrait être appliqué à d'autres ERP.

**Structuration du processus de déploiement sur la carte d'unités fonctionnelles.** Pour être capable d'extraire un catalogue de morceaux de paramétrage à partir des paramétrages existants de *Copilote*, nous avons besoin d'associer les paramètres avec la logique métier du système qu'ils impactent. Ces relations ne sont pas explicitement définies et seuls les intégrateurs expérimentés les connaissent. C'est pourquoi le premier objectif est d'extraire ces connaissances pour pouvoir capitaliser dessus. Pour cela, nous introduisons une nouvelle abstraction appelée carte d'unités fonctionnelles. Cette carte divise la logique fonctionnelle de *Copilote* en *unités fonctionnelles* structurées dans un Graphe Orienté Acyclique et nous permet d'associer les

paramètres avec le périmètre de logique fonctionnelle qu'ils impactent. Cette carte détaille l'ensemble des parties de *Copilote* qui peuvent être paramétrées selon les besoins d'une entreprise. Elle peut être considérée comme la modélisation de l'ensemble des fonctionnalités couvertes par *Copilote*. Elle peut être utilisée de différentes manières durant le processus de déploiement, pour guider les intégrateurs pendant l'analyse des besoins d'une entreprise et associer chaque besoin avec l'unité fonctionnelle concernée, et pour aider les intégrateurs à trouver les paramètres concernés quand ils paramètrent un besoin métier. Dans nos travaux, cette carte nous permet de considérer les morceaux de paramétrages qui concernent une même unité fonctionnelle ce qui est essentiel pour extraire des morceaux de paramétrage pertinents.

**Processus de fouille intéractif.** Nous proposons d'extraire des morceaux de paramétrage pertinents à partir des déploiements antérieurs de *Copilote*. Pour cela, nous avons collecté dans une base de données l'ensemble des paramétrages existants. Nous montrons comment extraire des morceaux de paramétrage pertinents, *i.e.*, des sous-ensembles de valeurs de paramètres qui correspondent à un besoin métier, en solutionnant des problèmes de clustering conceptuel. Comme la difficulté majeure est de définir les contraintes et critères nécessaires pour extraire des morceaux de paramétrage pertinents, nous proposons un processus interactif qui intègre les retours des experts en ajoutant de nouvelles contraintes ou critères d'optimisation pour améliorer progressivement la pertinence des morceaux de paramétrage extraits.

**Nouveaux modèles PPC pour le clustering conceptuel sous contraintes.** Le clustering conceptuel est un problème $\mathcal{NP}$-complet qui consiste à partitionner un ensemble d'objets de telle sorte que les objets regroupés dans un même cluster partagent des propriétés communes. Dans notre contexte où les retours des experts sont itérativement intégrés, nous avons besoin d'un outil flexible et efficace pour résoudre ce problème. Nous proposons d'utiliser la PPC, et les principales contributions de cette thèse sont de nouveaux modèles PPC et des contraintes globales pour résoudre les problèmes de clustering conceptuel sous contraintes en optimisant divers critères.

Il existe plusieurs approches PPC pour résoudre les problèmes de clustering conceptuel [Gun15 ; Dao+15a]. Cependant, ces approches supposent que le nombre de clusters est connu par avance ce qui n'est pas le cas dans notre contexte. Ces approches peuvent être étendues au cas où le nombre de clusters n'est pas fixé mais elles ne passent pas à l'échelle. Par conséquent, nous proposons deux nouveaux modèles de PPC pour résoudre les problèmes de clustering conceptuel quand le nombre de clusters n'est pas connu. Le premier modèle peut être considéré comme une extension du modèle de Dao et *al.* [Dao+15a]. Le deuxième modèle résout le problème en deux étapes, comme proposé par Ouali et *al.* dans [Oua+16] : la première étape consiste à extraire l'ensemble des clusters possibles à l'aide d'un outil spécialisé comme LCM [Uno+04] ; la deuxième étape consiste à sélectionner un ensemble de clusters qui partitionnent les objets à clusteriser. Nous avons expérimentalement comparé nos modèles PPC avec les approches déclaratives existantes sur un jeu de données classique d'apprentissage automatique et sur un nouveau jeu de données que nous avons généré à partir des paramétrages de *Copilote*. Nous montrons que nos modèles passent mieux à l'échelle quand le nombre de clusters n'est pas fixé et que nous optimisons un seul critère.

La deuxième étape de l'approche en deux temps proposée dans [Oua+16] revient à résoudre un problème de couverture exacte sous contraintes. Ce problème est $\mathcal{NP}$-complet et concerne plusieurs applications. Nous introduisons une nouvelle contrainte globale pour résoudre efficacement ce problème en utilisant une structure de données backtrackable introduite par Knuth dans [Knu09]. Nous proposons deux extensions à cette contrainte globale : la première permet de contraindre le nombre de sous-ensembles sélectionnés, et la deuxième permet de contraindre les coûts minimaux et maximaux associés aux sous-ensembles sélectionnés. Nous introduisons plusieurs algorithmes de propagation pour ces contraintes globales et nous montrons qu'ils sont plus efficaces que les approches déclaratives existantes pour les problèmes d'optimisation mono et multi-critères.

## Plan de la thèse

La première partie présente le contexte applicatif de nos travaux. Dans le chapitre 2, nous décrivons les principes de base des ERP et nous détaillons les particularités de *Copilote*. Dans le chapitre 3, nous décrivons comment les intégrateurs paramètrent *Copilote* durant le déploiement du système dans une nouvelle entreprise, et nous étudions la structure du paramétrage. Dans le chapitre 4, nous introduisons notre contribution applicative qui est une nouvelle approche pour guider les intégrateurs durant le paramétrage du système basée sur une carte d'unités fonctionnelles.

La deuxième partie décrit le contexte technique de nos travaux. Dans le chapitre 5, nous introduisons les principes de bases de la PPC. Dans le chapitre 6, nous décrivons le problème de clustering conceptuel et les approches déclaratives existantes pour résoudre ce problème. Dans le chapitre 7, nous décrivons le problème de couverture exacte et la structure de données *Dancing Links* introduite par Knuth pour résoudre ce problème. Nous décrivons et comparons également les approches déclaratives existantes pour résoudre ce problème. Dans le chapitre 8, nous décrivons les deux jeux de données que nous utilisons pour nos expérimentations : un jeu de données existant pour évaluer les algorithmes d'apprentissage automatique, et un nouveau jeu de données issus des paramétrages de *Copilote*.

La troisième partie décrit nos contributions techniques. Dans le chapitre 9, nous introduisons deux nouveaux modèles PPC pour résoudre les problèmes de clustering conceptuel et nous les comparons avec les approches déclaratives existantes. Dans le chapitre 10, nous introduisons une nouvelle contrainte globale *exactCover*, et nous décrivons et comparons trois algorithmes de propagation pour cette contrainte. Nous étendons *exactCover* au cas où des mesures d'utilité sont associées aux sous-ensembles pour permettre de contraindre l'utilité minimale et maximale des sous-ensembles sélectionnés. Dans le chapitre 11, nous introduisons la contrainte globale *exactCoverK* qui étend *exactCover* au cas où le nombre de sous-ensembles sélectionnés est contraint. Nous décrivons trois algorithmes de propagation pour cette contrainte. Dans le chapitre 12, nous évaluons l'intérêt de nos contraintes globales pour résoudre différents problèmes de clustering conceptuel. Nous considérons dans un premier temps le cas où un seul critère doit être optimisé puis le problème d'optimisation multi-critères qui consiste à calculer le front Pareto des solutions non-dominées sur un ensemble de critères. Nous introduisons une nouvelle stratégie dynamique pour résoudre ces problèmes

d'optimisation multi-critères et nous la comparons avec les stratégies statiques et dynamiques existantes. Dans le chapitre 13, nous introduisons un outil de fouille qui utilise nos modèles PPC pour extraire des morceaux de paramétrage pertinents et intègre itérativement les retours de l'expert en adaptant nos modèles. Nous listons différents retours que nous avons collectés lors de l'utilisation de notre outil par un expert sur une unité fonctionnelle de *Copilote* pour démontrer la faisabilité de notre approche.

Dans le chapitre 14, nous terminons nos travaux avec une discussion sur plusieurs perspectives.

# Chapter 1

# Introduction

## Motivations

Enterprise Resource Planning (ERP) systems are essential for industrial companies to structure, automatize and monitor their business processes in order to boost their competitiveness. ERPs are undisputed leaders for information systems in the industry sector. However, ERP systems are generic software which are designed to serve a large variety of companies with different business processes and needs. Therefore, they have many configuration options to support various business processes used in different companies. The *implementation process* of an ERP system is the process that consists in assigning values to ERP parameters according to the company requirements: It determines the exact operations and processes supported by the system in the specific company. This process involves many organizational issues such as project management, readiness for change or training of the staff on the new system. Operational issues related to the elicitation of the requirements of a company or the configuration of the ERP system have been widely studied. However, they are still not well understood.

Infologic is a French company that develops and integrates their own ERP system, called *Copilote*, specialized for agri-food industry. The main strength of *Copilote* is that it is highly customizable to handle many business requirements. It has tens of thousands of parameters that are used to adapt it as precisely as possible to customer requirements. However, this flexibility makes the implementation of *Copilote* a time consuming task that requires a deep knowledge of its functionalities and parameters. System integrators (employees of Infologic who implement *Copilote*) need at least eight months of training and practice to be autonomous for implementing *Copilote*. Indeed, Infologic does not capitalize on previous implementations of *Copilote* and system integrators skills are based on their own experience on the system. Reducing the complexity of the implementation of *Copilote* is a critical issue for Infologic who needs to integrate quickly and efficiently new system integrators to meet the demand of new customers which is significantly increasing.

In this thesis, we study the implementation process of *Copilote* in order to understand the main issues encountered by Infologic. In particular, we focus on the configuration step and on the architecture of *Copilote* parameters. Most of the time, a critical issue when configuring a business requirement is to find the right parameters to assign, *i.e.*, the parameters that impact the right business logic of *Copilote*, and then to find the right values to assign to these parameters. Our challenge is to provide a tool that assists system integrators when they configure *Copilote* according to business requirements.

To this aim, we propose to reduce the configuration time by reusing parts of previous configurations. Indeed, as pointed out by Daneva in [Era+15], companies have many common requirements, and the rate of requirement reuse may be remarkably high in some cases. More precisely, we propose an approach for extracting a catalog of configuration parts from existing configurations of *Copilote*, and we propose to associate with every configuration part of the catalog the business requirement it fulfills. This catalog capitalizes on the past experience of all integrators and will allow us to reuse this experience for next implementations of *Copilote*.

The most challenging part of our approach is to use data mining techniques to identify relevant configuration parts that may correspond to a business requirement. We propose to use *Formal concept Analysis* (FCA) and conceptual clustering to group together configurations that fulfill the same requirement and identify the corresponding part of configuration. To compute conceptual clusterings, we propose to use *Constraint Programming* (CP). Indeed, Copilote integration experts are not able to formally define all constraints and criteria needed to mine relevant configuration parts in an *a priori* way. The declarative framework of CP allows us to easily integrate feedbacks of experts by means of new constraints and optimization criteria in order to increase the relevancy of the mined configuration parts.

## Goals and contributions

Our goal is to extract a catalog of configuration parts from existing implementations of an ERP system in order to simplify future implementations of this system. To achieve this very general objective, a first contribution is the definition of a new abstraction, called *business unit map*, which allows us to structure the whole implementation process. A second contribution is the definition of an interactive mining process for building the catalog, where experts may interactively refine constraints and criteria used to mine relevant configuration parts. A third contribution is the definition of new CP models and global constraints to efficiently solve constrained conceptual clustering problems that occur during the mining process.

The ERP system used to illustrate and evaluate our contributions is *Copilote*. However, the process we propose for extracting a catalog of configuration parts from existing implementations of an ERP system is rather generic, and we believe it may be applied to other ERP systems as well.

**Structuration of the implementation process with a business unit map.** To be able to extract a catalog of configuration parts from existing configurations of *Copilote*, we need to relate parameters with the business logic of *Copilote* they impact. These relationships are not explicitly defined, and only experimented *Copilote* integrators have a good knowledge of them. Hence, a first goal is to extract this knowledge and capitalize it. To this aim, we introduce a new abstraction called business unit map. This map divides the business logic of *Copilote* into *business units* structured in a Directed Acyclic Graph (DAG) and allows us to relate parameters with the business logic scope of *Copilote* they impact. This map details all the parts of *Copilote* that may be configured according to business logic requirements. It may be seen as the model of all capabilities of *Copilote*. It may be used in many different ways during the implementation process, to guide system integrators during the requirement analysis

and relate every requirement to the concerned *Copilote* business unit, and to help system integrators focus on the right parameters when configuring a requirement. In our work, this map allows us to focus on configuration parts that concern a single business unit which is essential to extract relevant configuration parts.

**Interactive Mining Process.** We propose to extract relevant configuration parts from existing implementations of *Copilote*. To this aim, we have created a database that contains all previous configurations of the system. We show how to mine relevant configuration parts, *i.e.*, subsets of parameter values that implement business requirements, by solving constrained conceptual clustering problems. As it is difficult to define the constraints and criteria that lead to relevant configuration parts, we propose an interactive mining process, where expert feedbacks are iteratively added, by means of constraints and optimization criteria, in order to progressively improve relevancy.

**New CP approaches for constrained conceptual clustering.** Conceptual clustering is an $\mathcal{NP}$-hard problem which basically involves partitioning a set of objects in such a way that objects within a same cluster share some properties. In our context where expert feedbacks are integrated by means of constraints and optimization criteria, we need both an efficient and flexible tool for solving this problem. To this aim, we propose to use CP, and the main technical contributions of this thesis are new CP models and global constraints for efficiently solving conceptual clustering problems under various constraints and criteria.

There exist several CP approaches to solve conceptual clustering problems [Gun15; Dao+15a]. However, these approaches assume that the number of clusters is fixed and known *a priori*, which is not the case in our context. These models may be extended to the case where the number of clusters is not fixed but they do not scale well in this case. Therefore, we propose two new CP models to solve conceptual clustering problems when the number of clusters is not known *a priori*. The first model may be seen as an extension of the model of Dao et *al.* [Dao+15a]. The second model solves the problem in two steps, as proposed by Ouali et *al.* in [Oua+16]: In a first step, all possible clusters are efficiently extracted by using a dedicated mining tool such as LCM [Uno+04]; In a second step, a subset of clusters that defines a partition is selected by using a CP model. We experimentally compare our new CP models with existing declarative approaches on classical machine learning instances and on a new benchmark we have generated from existing configurations of *Copilote*. We show that our new CP models scale well when the number of clusters is not fixed and when considering a single criterion to optimize.

The second step of the two-step process proposed in [Oua+16] basically involves solving a constrained exact cover problem. This problem is $\mathcal{NP}$-hard and has many applications. We introduce a new global constraint to solve it efficiently with CP, by using a backtrackable datastructure introduced by Knuth in [Knu09]. We propose two extensions of this global constraint: an extension to constrain the number of selected subsets, and an extension to constrain minimal and maximal utility costs associated with selected subsets. We introduce different propagation algorithms for these global constraints, and we show that they are more efficient than existing declarative approaches for both mono and multi-criteria optimization problems.

## Outline of the thesis

The first part describes the applicative context of this work. In Chapter 2, we describe basic principles of ERP systems, with a specific focus on *Copilote*. In Chapter 3, we describe how experts configure *Copilote* when implementing it for a new company, and we describe the structure of the parameters that have to be configured. In Chapter 4, we introduce our applicative contribution, which is a new approach for guiding system integrators during the configuration step by using a business unit map.

The second part describes the technical context of this work. In Chapter 5, we introduce basic principles of CP. In Chapter 6, we describe the conceptual clustering problem and existing declarative approaches for solving this problem. In Chapter 7, we describe the exact cover problem and the *Dancing Links* data structured introduced by Knuth for this problem. We also describe and compare existing declarative approaches for solving this problem. In Chapter 8, we describe the two benchmarks that are used in our experimental evaluations: An existing benchmark often used to evaluate machine learning algorithms, and a new benchmark derived from our ERP application.

The third part describes our technical contributions. In Chapter 9, we introduce two new CP models for solving conceptual clustering problems, and we compare them with existing declarative approaches. In Chapter 10, we introduce the *exactCover* global constraint, and we describe and compare three propagation algorithms for this constraint. We also extend *exactCover* to the case where utility costs are associated with subsets and minimal or maximal utility costs of selected subsets are constrained. In Chapter 11, we introduce the *exactCoverK* global constraint, which is an extension of *exactCover* to the case where the number of selected subsets is constrained to be equal to a given integer variable. We describe and compare three propagation algorithms for this constraint. In Chapter 12, we evaluate the interest of our new global constraints for solving different conceptual clustering problems. We first consider mono-criterion optimization problems, where a single objective function must be optimized, and we consider four classical objective functions. Then, we consider multi-criteria optimization problems, that aim at computing the Pareto front of non-dominated solutions with respect to several objective functions. We introduce a new dynamic strategy to solve these multi-criteria optimization problems and compare it with existing static and dynamic strategies. In Chapter 13, we introduce our interactive mining tool that uses our CP models to mine relevant configuration parts and integrate expert feedbacks by means of constraints and optimization criteria. We list the different feedbacks we have collected when experimenting this tool with an expert on a specific business unit of *Copilote*, used as a proof of concept of our process.

In Chapter 14, we conclude this work with a discussion of some perspectives.

## Publications

The CP models described in Chapter 9 have been published in 2017, in the international conference on principles and practice of CP [Cha+17b], and in the *Journées Francophones de Programmation par Contraintes* [Cha+17a]. The global constraint *exactCover* described in Chapter 10 has been presented to the doctoral program of the international conference on principles and practice of CP in 2018 [Cha+18].

We plan to submit a journal paper where we describe the different propagation algorithms introduced in Chapters 10 and 11, in a few weeks.

# Part I

# Applicative Context and Proposed Approach

*Enterprise Resource Planning* (ERP) systems play a crucial role for industrial actors to structure and improve their processes in order to generate more profit. Therefore, any issue that involves ERP system efficiency becomes crucial and finding a solution to solve it represents an important economic challenge. This thesis has been done in collaboration with Infologic, a company that develops and implements its own ERP system called *Copilote*. We present in this part main issues of Infologic that led to launch this project.

We define in Chapter 2 what is an ERP system and we present the activity and the history of Infologic. Then, we describe the implementation process of ERP systems and we highlight some well-known issues. We focus in Chapter 3 on the configuration step of *Copilote* in order to understand the main issues encountered by Infologic. Finally, we propose in Chapter 4 a new approach to guide the implementation process of *Copilote* and assist the configuration step.

# Chapter 2

# Enterprise Resource Planning

## Contents



ERP systems have been widely studied since they have an essential role in industrial information systems. Reducing the risks when installing an ERP system in an industrial company is a major issue.

In this chapter, we first define what is an ERP system, in Section 2.1. Then, we present the history of Infologic and the ERP system developed and installed by the company in Section 2.2. We introduce some well-known issues encountered during the implementation process of ERP systems in Section 2.3. We focus on the implementation process of *Copilote* in Section 2.4, and we study the time spent on each step of this implementation process in Section 2.5

## 2.1 Definition of an ERP

**ERP history.** In the 1960s, first management software solutions emerged with the growth of the industry combined with technological progresses. *Material Requirements Planning* (MRP) systems helped manufacturers to translate their production schedule into time-phased net requirements for the sub-assemblies [Gum96]. Major industrial players such as Toyota used MRP systems to manage their materials.

In the mid 1970s, MRP systems were extended to a standard application of production resource planning called MRP II [Chu+99] that considers all the resources of a company such as workforce or machine capacities. However, the need to manage production facility's orders, production plans and inventories into the system led to the development of a more integrated solution called *Enterprise Resource Planning* (ERP) [Chu+99].

An ERP system may be defined as a customizable standard software application which includes integrated business solutions (also called modules) for the core processes such as production planning and control or warehouse management and the main administrative functions of a company such as accounting or billing [Ros+99]. An ERP system facilitates the storage, the retrieval and the analysis of the data by integrating functional modules all together. It may be seen as a tool that allows an industrial company to manage and monitor in real time its whole activity [Ros+99].

There exist many software specialized for one single business process such as sales forecast systems for instance. These systems may be more efficient and bring more functionalities than an ERP module on this specific process. However, the interface of these specialized software with the rest of the information system of a company can be very complex and expensive, particularly to make the data coherent with the whole system. Therefore, the fact that ERP systems propose a native integration of their functional modules is a significant advantage that provides efficiency, robustness and coherency of the data.

This logic has been widely followed by industrial companies: They find a real added value in having an information system that is robust and coherent between many functional modules. For a long time, especially during the 1990s, ERP systems were seen as expensive solutions, hard to install for Small and Medium Enterprises (SME) but nowadays, the strong competition between ERP vendors and the increase of the demand from industrial companies have allowed many companies to purchase ERP solutions. ERP systems have become essential for industrial companies. They are often considered as the backbone of a company [Gra+14] or as the nervous system of the organization in which data are nerve impulses [Ahm+13]. If data are not correct, responses of the functional modules will not be accurate.

It is important to note that nowadays, it would be extremely difficult, if not impossible, to develop from scratch a specific software for an industrial company considering the significant number of processes to manage.

## 2.2 Presentation of Infologic and *Copilote*

**History of Infologic.** Infologic was created by André Chabert in 1982 in Valence (France). Infologic develops and integrates its own ERP system specialized for agri-food industry. As shown in Figure 2.1, Infologic started to sell its first ERP solution, *Agro V1*, in 1984. Up to 20 companies used this system, mainly in poultry industry of Rhône-Alpes region. Infologic can be considered as a pioneer of ERP systems in France since its creation coincides with the apparition of first ERP systems.

During the 1990s, Infologic grew and developed a new version of its ERP system called *Agro V2*. To gain market shares, Infologic opened a new office in Nantes to reach many agri-food companies of the west of France. *Agro V2* had been sold to up to 200 customers and is still operating in a few companies.

In the 2000s, text-based systems such as *Agro V2* became obsolete and Infologic had to propose a graphic version of its ERP. Infologic chose to develop a new ERP system from scratch using Java technology. The huge gap between object oriented programming languages such as Java and old technologies used in previous versions of its ERP system forced Infologic to replace its development teams. However, the developer's shortage in France led Infologic to relocate part of the development team

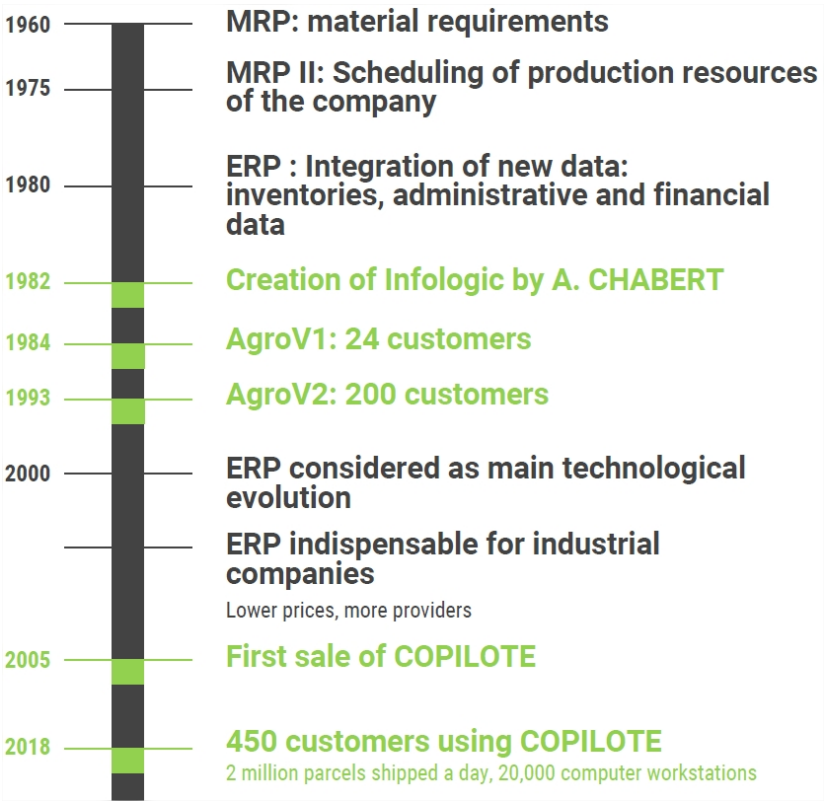



Figure 2.1 – Infologic evolution within ERP history.

in Romania, specially in Bucarest and Iasi. In 2005, Infologic sold for the first time its new ERP system called *Copilote*. It is important to note that most of the developers of *Agro V2* became system integrators of *Copilote*, *i.e.* experts that configure *Copilote* for a new customer of Infologic. Therefore, the experience accumulated for 20 years remained in the enterprise.

In the 2000s, Infologic kept on growing and opened a new office in Toulouse to reach market shares of the south west of France. Infologic has now up to 450 customers that represent up to 2 million parcels shipped every day or up to 20,000 computer workstations installed all over France. Infologic's growth has been particularly impressive for the last few years. The number of employees has grown from 120 in 2014 to up to 200 in 2018 and the revenue has increased from 12 million euros in 2014 to 21 million euros in 2018. The staff is mainly composed of system integrators (30%) and developers (24%). The rest of the staff corresponds to sales, maintenance, technical and administrative departments. Infologic has the particularity of selling a turnkey solution: technical engineers install the hardware such as servers or computer workstations and system integrators configure the ERP system according to customer requirements.

**Competitors.** The global leader in ERP solutions is SAP which is a German company that generates over $17 billion in sales revenue. In addition to SAP, other international software editors such as Oracle, Microsoft and Sage propose ERP solutions for French agri-food industry companies. Infologic also competes with French ERP editors such as Vif or Proginov which have a size comparable to Infologic. These smaller

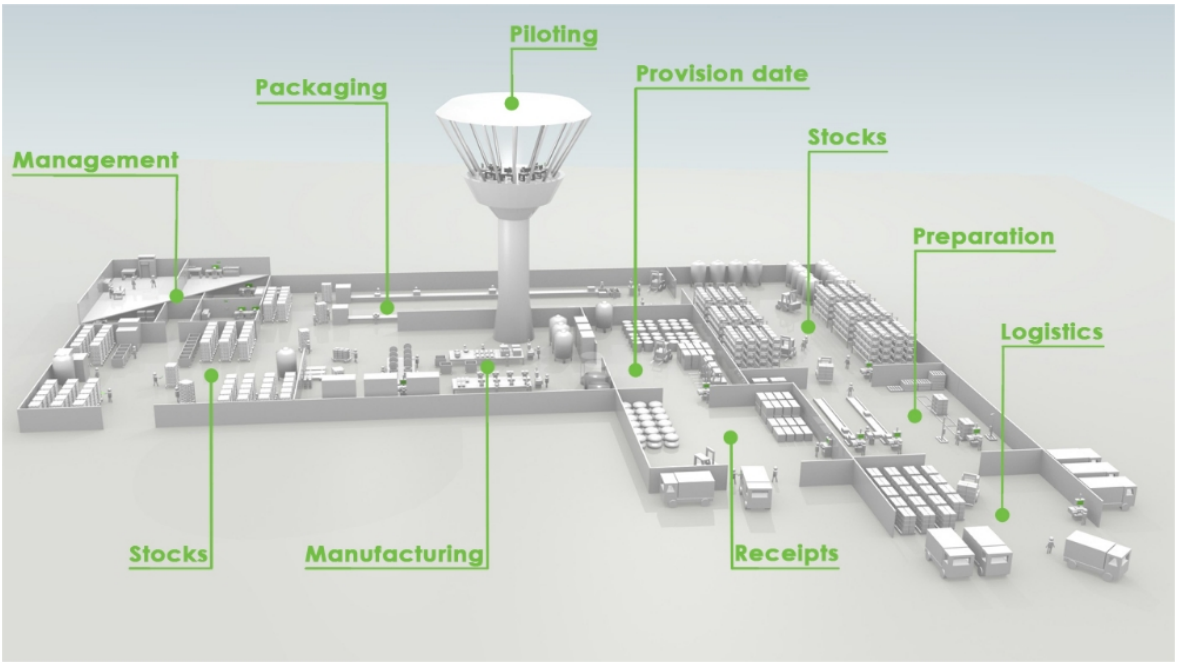



Figure 2.2 – Functional perimeter of the ERP Copilote.

editors take advantage of their size by being more reactive and attentive to their customer requirements, which is very appreciated. They are able to adapt or modify more easily their system compared to big editors which have much more inertia. However, most of international industrial companies still favor bigger editors such as SAP since they have many international references and a robust currency management, which is essential for this kind of companies.

**Presentation of *Copilote*.** *Copilote* is the ERP system currently developed and sold by Infologic. The system has been continually evolving in response to new customer requirements since 2002. As shown in Figure 2.2, the main strength of *Copilote* is its large functional perimeter. It integrates classical modules such as commercial management, warehouse management, manufacturing execution system or logistic management. However, Infologic differs from its competitors by integrating some modules that are rarely natively included in ERP systems. *Copilote* integrates its own financial and sales forecasts modules, a very powerful decision-making system that allows to monitor in real time activities of a company and an efficient Electronic Data Interchange tool.

The particularity of *Copilote* is that exactly the same software is installed for all the customers whereas they have different fields of activity. For instance, *Copilote* is the leader in poultry and egg industry, but it can be used in wine, ready-cooked or food trading companies while they have most of the time different processes and needs.

Therefore, *Copilote* has tens of thousands of parameters that are used to adapt it as precisely as possible to customers requirements. This flexibility of configuration is a real strength of *Copilote* that is recognized by its customers to be a very performing and adaptable ERP system. However, this ultra-customization makes the system very complex, especially for system integrators of Infologic. In particular, a same goal may be achieved by configuring the ERP system in many different ways. Some figures highlight the complexity and the dimension of *Copilote*:

- *Copilote* integrates 4,227 screens;
- *Copilote* code contains 67,000 Java classes and up to 10 million lines;
- *Copilote* has 5,351 general parameters that allow to configure modules;
- *Copilote* has 10,934 database tables.

## 2.3 ERP implementation issues

An ERP system is designed to serve a large variety of companies. That is why it has many configuration options to support various business processes used in different companies. The *system implementation* is the process that consists in assigning values to the system's parameters according to the company requirements. It determines the exact operations and processes supported by the system in the specific company. This implementation process has been widely studied [Bin+99; Ahm+13; Mot+05; Ahm+12; Käh14; AM+03; Rob+11; Som+01; Pan], and it appears to be very complex with many factors that can impact the success of ERP implementation projects.

The study of [Pan] focuses on 342 ERP implementation projects in 2017 and gives some figures about financial and organizational issues involved for companies acquiring a new ERP system. Companies that took part in the survey have an average annual revenue of 445M$ and most of them are part of distribution or manufacturing industry. Most of the time, these companies acquire a new ERP system to improve the business performance, to make employees jobs easier or to ensure compliance. The average duration of an ERP implementation project is 17 months whereas the average cost is around 1.3M$. However, 74% (resp. 59%) of the companies of the study have experienced costs (resp. duration) overruns and 25% characterize their project as a failure, since the outcome does not correspond to their expectations.

This study shows that an ERP system implementation is a complicated process, still not well understood. We present in this section main issues involved in ERP implementations.

### 2.3.1 Alignment of the ERP system with enterprise processes

One issue of the implementation of ERP system is the precise identification of the gap between business process requirements of a company and ERP system capabilities in order to avoid as much as possible misalignments [Mam13; BG+05]. A misalignment is defined as the fact that the processes placed under ERP system control will not be aligned with the real needs and the processes of the company [Mam13]. That is why the question of requirements elicitation becomes essential for the ERP implementation and many approaches have been proposed [Lui+; Sof+05; VIL09; Jan+15; Dar+93; Gar+17; Rol+01; Lac+14]. The main idea of these approaches is to model both company's requirements and ERP system capabilities with the same modeling framework to measure the gap between the system and the requirements in order to be able to react quickly and make the right decisions.

### 2.3.2 Best practices vs ERP customization

A critical issue in ERP implementation is how to bridge the gap between the ERP system and an organization's business processes by customizing either the system, or the business processes of the organization, or both [Rol+01; Luo+04].

**Best practices.** A best practice is defined as the way to transfer the past successful experience to new ERP projects in order to improve the chance of successful implementations [Sha+12]. ERP vendors would like to integrate as much as possible best practices into the system such as embedded standard process configurations, for instance. It has been proved that best practices have a positive impact on the coordination during the implementation of the system and boost project efficiency [Hua+04].

However, best practices are sometimes not in favor of the business of a company and can reduce there competitiveness since organizations have often unique manufacturing problems [Hua+04]. If unique processes enable the company to gain a competitive advantage in its industry or are better suited to its culture, the advantage can be lost by using a standard process of the system [Soh+00].

**ERP customization.** Since the same system may be used in companies that have different processes and needs, customization of the system becomes inevitable. ERP customization makes ERP systems more user friendly and increases their acceptance by users [Lig05]. The customization of the system can be done through either the configuration that consists in assigning values to ERP's parameters or the modification of the source code in order to implement a specific process [Bre+01]. The second way leads to some benefits such as adding functionalities to the system, automating a new process or improving competitiveness of the industrial company [Lig01].

However, modification of the source code of such a large system can be risky and critical, it requires a good expertise of the system to avoid side effects.

Customization plays an important role in the success of an ERP implementation because it requires expertise in software solution as well as business processes and it may be a time consuming procedure that can increase the expenditure [Par+14].

**Balance between good practices and customization.** ERP systems almost never fit all the requirements of an industrial company, especially for manufacturing processes [Wu+07; Luo+04; Dit+09; Lig05]. The study of [Pan] shows that only 23% of companies adopt an ERP system with no or few customizations whereas 34% of ERP implementations need an important customization of the system. Even if it is possible for huge industrial companies to use standard ERP system functionalities, the lack of flexibility and key components can have a negative impact on the competitiveness of smaller companies and adopting standard processes proposed by the ERP system should be made with caution [BG+05].

The balance is necessary between high customization that requires additional effort from the implementation team [Par+14] and a lack of acceptance of ERP standard processes that can contribute to the failure of the project [Sko+01]. Therefore, the solution seems to inevitably combine both best practices and customization. For instance, [Fer+06] ensures that deliberately and carefully deviating from the best practices may also be effective.

### 2.3.3 Critical success factors

The identification of key factors for the success of an ERP implementation is a hot research subject. These factors are called *critical success factors* (CSFs) [Bin+99; Ahm+13; Mot+05; Ahm+12; Käh14; AM+03; Rob+11; Som+01]. CSFs are all the domains for which satisfying results increases the chance of success of the implementation of the system, and, therefore, improve the competitiveness of the company after the launch of the ERP system [FR79].

Obviously, operational factors such as configuration of the system or functional scope expression are essential during the ERP implementation [AM+03; Ahm+13]. However, many organizational factors, that are not related to the quality of the system, may have a great impact on the implementation process [AM+03; Ahm+13; Som+01; Bin+99]. For instance, project management, readiness for change or training of the staff on the new system are essential to achieve a successful implementation. These factors are confirmed by Infologic experts who highlight the importance of involving key members of a company into the project and having the support and commitment of the company's management.

## 2.4 Implementation process of *Copilote*

To understand the difficulties encountered by Infologic during the implementation of *Copilote*, we detail how Infologic integrators implement *Copilote* for new customers, *i.e.*, industrial companies that purchased *Copilote*. Figure 2.3 describes the implementation process of *Copilote*. Each box corresponds to a step of the implementation. The time scale on the left gives an idea of the average time spent for each step for a 8 month project (which is a standard duration according to our observations). Obviously, the duration of the implementation process may be variable depending on the customer activity and the scope of *Copilote* to implement.

**Requirement analysis:** The first step of *Copilote* implementation is the collect of the functional requirements of the customer. Experimented integrators of Infologic visit customers facilities and interview main managers of the company about their business processes to understand how they work and their specificities. This task is achieved by skilled and experienced integrators because it requires expertise in business processes as well as in *Copilote* capabilities. There are always several system integrators involved in a new implementation because integrator expertise scope is often limited to one single module. The main challenge for Infologic is to get complete and stable requirements as soon as possible because the later a requirement changes, the more important is the cost and the impact on the implementation process.

The requirement elicitation can last a few weeks, depending on the size of the customer and on the functional scope of *Copilote* to install. After interviewing managers of the company, Infologic integrators formalize the current operations of every business process of the customer and how it will work under control of *Copilote* in a document called *requirement analysis*. This document has to be signed by both Infologic and the customer representative before continuing the implementation process. Theoretically, all misalignments between *Copilote* and customer requirements have to be identified

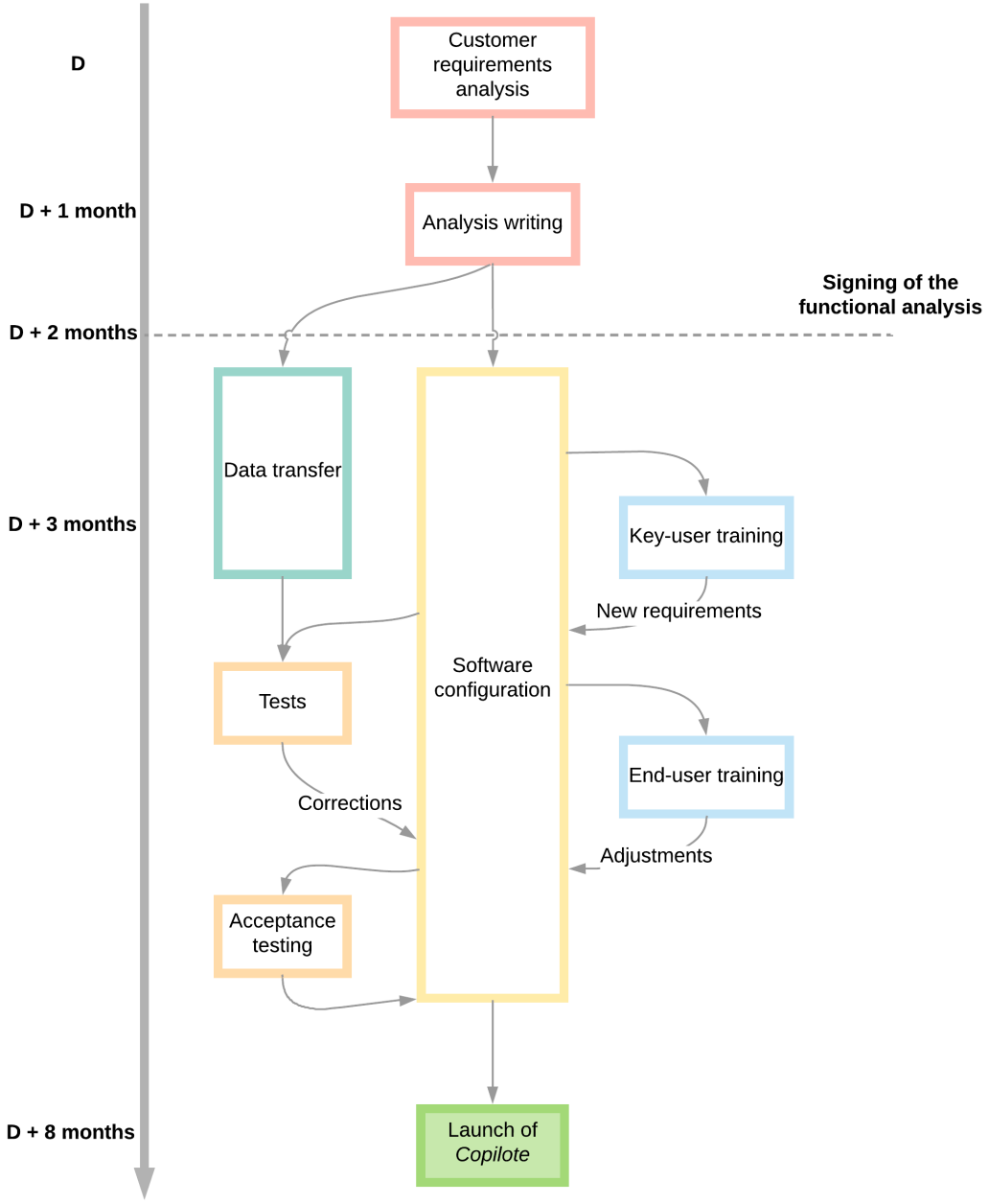



Figure 2.3 – Typical implementation process of *Copilote*.

before the signature and decisions of adapting customer processes to *Copilote* or modifying *Copilote* have to be taken before continuing the implementation. Furthermore, this document has a legal value and is the only reference for the rest of the implementation process which means Infologic is committed to meet the requirements as described in the document.

This step may be very hard to complete because it is complicated for the customers to explain clearly their current processes as well as the way they want to work since they do not know enough *Copilote* capabilities. The role of Infologic integrators is all the more important as they have to ask the good questions to make sure the managers give all the important details about the way they work and their specificities.

**Data transfer:** Data transfer is the step during which the customer has to transfer the data from his old system into *Copilote* using integration files. Infologic advices to integrate core data such as customers, suppliers, materials and items which are essential for *Copilote* since most of the operations done during business processes use these data. This step may become critical if it is too delayed because data are necessary to test the configuration of *Copilote* and validate that the implementation meets the requirements. Most of the time, customers underestimate the duration of this task that can be time-consuming. They do it at the last moment, most of the time with mistakes in the data they integrate.

**System configuration:** System configuration constitutes the longest step of the implementation of *Copilote*. This step basically involves assigning values to parameters in such a way *Copilote* fulfills the customer needs as described in the *requirement analysis* document. The goal of the configuration may be to customize the user interface, to create statistics or to configure critical business logic into process operations.

The complexity of this step comes from the fact that *Copilote* has a large number of parameters with strong interactions that are not always explicitly stated. When a system integrator has to configure a customer requirement, it may be hard to figure out which parameters have to be changed and what are the right values to assign to them, especially when this is the first time he encounters this kind of requirement. The complexity of this step motivated our work and we will detail the configuration process later in this thesis.

It is important to note that the configuration step does not only involve assigning values to parameters but also testing if *Copilote* is configured as expected. Therefore, data must have been transfered into *Copilote* to be able to run the processes and assess whether it works as expected. That is why the data transfer step is so important: It allows to do correct and relevant tests. This step can be time consuming, particularly when tests fail since it is hard to identify which parameters have wrong values.

**User training:** The user training step is essential during an ERP implementation. Obviously, a good training of all future users ensures a correct and efficient way to use *Copilote* when it is launched. It also allows Infologic to involve as many customer employees as possible into the project. It is important to get many employees to adhere to the new way of working since managing culture-change is pointed out as a critical success factor [Mot+05].

Infologic proposes a very early training for key-users. The goal is to make them understand the main concepts and specificities of *Copilote*. These trainings very often lead to new requirements or requirement modifications since key-users have a better understanding of *Copilote*.

The second step of training concerns end-users and aims to explain and practice all the operations they will have to do on the system. Therefore, these trainings are done almost at the end of the implementation process to prevent users from forgetting the training before launching the new system.

**Testing:** The testing step is one of the most important step to make sure *Copilote* is configured as expected. Tests are done all along the implementation process by both Infologic system integrators and future end-users. This way, Infologic integrators use

Table 2.1 – Evolution of time allocation for the last 5 years. For each year and each step, the column *days* gives the total time spent in man-days for all *Copilote* implementations, the column *%* gives the percentage of this step with respect to the whole process, and the column *avg* gives the average number of man-days spent per implementation.

| | 2013 | | | 2014 | | | 2015 | | | 2016 | | | 2017 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 34 implem. | | | 35 implem. | | | 34 implemen. | | | 38 implem. | | | 53 implem. | | |
| | days | % | avg | days | % | avg | days | % | avg | days | % | avg | days | % | avg |
| Config. and testing | 2,285 | 50 | 67.2 | 2,266 | 50 | 64.7 | 3,287 | 51 | 96.7 | 4,300 | 58 | 113.2 | 4,340 | 57 | 81.9 |
| User training | 1,046 | 23 | 30.8 | 930 | 21 | 26.6 | 1,322 | 21 | 38.9 | 1,263 | 17 | 33.2 | 1,251 | 17 | 23.6 |
| Requirement analysis | 647 | 14 | 19.0 | 670 | 15 | 19.2 | 1,110 | 17 | 32.7 | 841 | 11 | 22.1 | 951 | 13 | 18.0 |
| Project tracking | 306 | 7 | 9.0 | 412 | 9 | 11.8 | 498 | 8 | 14.6 | 890 | 12 | 23.4 | 868 | 11 | 16.4 |
| Data transfer | 261 | 6 | 7.7 | 213 | 5 | 6.1 | 215 | 3 | 6.3 | 181 | 2 | 4.8 | 150 | 2 | 2.8 |
| Total | 4,545 | 100 | 134 | 4,491 | 100 | 128 | 6,432 | 100 | 189 | 7,475 | 100 | 197 | 7,560 | 100 | 143 |

their expertise to adjust the *Copilote* configuration if necessary whereas future users can feel more comfortable with *Copilote* by practicing. Infologic experience shows that a lack of tests often leads later to the discovery of configuration problems or bad practices from the users.

Acceptance testing aims to contractually validate that Infologic has done the work as expected by the customer. For Infologic, this last battery of tests with the customer is essential to agree that the system is ready to be used live.

**Launch of *Copilote*:** The last step of the implementation process is the launch of the system live. Infologic sends teams of integrators to help end-users to use *Copilote* and to handle configuration problems that occur such as price incoherences or errors on sales item labels, for instance. The launch of a new system is always very intense because users do not feel comfortable with it and are slower than usual. It is always hard to measure immediately the expected gains. Most of the time, Infologic teams stay with the customer for a week.

Infologic remains in support to the customer after launching *Copilote* because the company may need assistance when a problem occurs with it. This situation happens particularly on operations that are done only a few times in a year such as inventory since the configuration is less used and tested.

## 2.5 Time allocation

Table 2.1 shows the evolution of the time spent by system integrators for each step of all implementations of *Copilote* from 2013 to 2017. These data have been collected from *Copilote* where all employees log their activity. For each year and each implementation step, we give the total number of man-days spent for all the implementations, the percentage it represents in the whole process and the average time in man-days for one implementation, *i.e.*, the number of man-days divided by the number of implementations of the year (which is reported in the second line of the table).

Time spent during the implementation process is essentially dedicated to configuration and testing. This step constitutes half of the total time from 2013 to 2015 and reaches up to 57% of the whole process in 2016 and 2017. In 2017, 4,340 man-days were spent for configuration and testing, which represents almost 20 full-time employees doing only configuration and testing.

The second longest step is the user training step. The time spent for this step has continually decreased from 2013 (23% of the time) to 2017 (17% of the time). The percentage of time allocated for requirement analysis has first increased, from 14% in 2013 to 17% in 2015, and then decreased to 13% in 2017. However, the requirement analysis step is divided into two activities: the analysis with the customer and the writing of the requirement analysis document. It is interesting to note that the time dedicated to requirement analysis writing increased whereas requirement analysis time decreased. It means that system integrators need more time than before to write the analysis document. In 2017, for 1 man-day (8 hours) of requirement analysis, a system integrator needed up to 6 hours to write the analysis whereas he needed less than 3 hours in 2013.

The only step whose percentage of time allocated increased, with configuration and testing step, are project tracking. Project tracking represents around 11% of the whole process in 2017. This trend can be explained by the increase of the requirements from the customers in terms of project management and by the efforts made by Infologic to structure and standardize the dialogue with the customer during the progress of the project.

We can note that the implementation activity has experienced a spectacular growth between 2014 and 2015. However, Infologic implemented *Copilote* in 2015 once less than in 2014 (35 implementations). To meet the workload, Infologic hired new system integrators with no experience on *Copilote* and trained them to be operational as soon as possible. Most of them were operational in 2016. This can explain the increase of the percentage of time dedicated to configuration since they were beginners.

When we relate these figures with the CSF described in part 2.3.3, the decrease of the time dedicated to requirement analysis is worrying since it is a critical step to identify all misalignments between *Copilote* and the customer requirements. Each time a misalignment is detected late during the implementation process, it implies some extra time to reconfigure *Copilote* according to the new requirements. Moreover, time allocated to the user training step has continually decreased since 2013 while this step may be essential to involve key-users into the project and manage the change of the processes.

## 2.6 Discussion

ERP systems are huge customizable standard software applications that are essential for industrial companies. However, the implementation process of an ERP system is very complex and involves both human and technical issues.

As we can see, most of the time spent during the implementation process of *Copilote* is dedicated to its configuration. After interviewing many system integrators, it turns out that they have difficulties to configure *Copilote*, particularly when this is the first time they are facing a new business logic requirement. Most of the time, to configure a new operation, integrators look for a similar case in their personal experience and try to adapt what they had done for the new case. This practice is based on personal experience and there is no capitalization between all integrators. It represents a critical issue for Infologic for several reasons. First, when system integrators leave Infologic, they leave with all their knowledge and no capitalization has been done on their experience. Moreover, it causes significant expertise gap between experienced integrators and new

ones who need to be trained for at least 8 months to be autonomous on the system. It is an important hindrance for the growth of Infologic who needs to integrate quickly and efficiently new system integrators. Solving this issue becomes vital for Infologic since the sales of *Copilote* are significantly increasing.

Furthermore, several studies have shown that this time-consuming configuration step is less critical than human factors such as user training or change readiness [Mot+05; Ahm+13]. Therefore, an important challenge is to reduce the time needed to configure an ERP system in order to spend more time on more critical tasks, such as requirements elicitation.

# Chapter 3

# Configuration of Copilote

## Contents



As explained in the previous chapter, the implementation process of an ERP system involves many issues and Infologic integrators spend most of their time configuring the system while other critical issues would need more resources. This chapter focuses on the configuration of *Copilote* to understand more precisely how system integrators configure it and identify the main issues.

We introduce existing configuration tools in Section 3.1 and then we focus on *Copilote*. We divide customer requirements into three categories in Section 3.2. Then, we present the architecture of the configuration of *Copilote* in Section 3.3 and we introduce the configuration methodology used by system integrators in Section 3.4. Finally, we focus on general parameters in Section 3.5 by analyzing the configuration screen, existing dependencies between these parameters and existing configurations.

## 3.1 Existing Configuration Tools

The configuration step requires deeply knowledgeable experts in specific modules who tend to be extremely expensive resources [Ari+03]. Their knowledge is rarely capitalized by companies that integrate ERP systems as Infologic who need a great number of integrators to meet the demand. Therefore, it may be hard, if not impossible, for very small industrial companies to purchase an ERP system since its implementation can be very expensive. Hence, accelerating the configuration step becomes an important economic issue for enterprises that integrate ERP systems as well as for small companies who need an ERP system. That is why several tools have been proposed to automate part of the configuration of an ERP system [Do+14; Ari+03; KW16; Buc+10].

**Tools based on decision trees.** Authors of [KW16] propose a configuration tool for small enterprises that cannot afford to pay consultants to configure an ERP system.

They present two approaches based on decision trees built on expert knowledge and classifiers to automate configuration options. The idea is to automatically configure the ERP system only with a questionnaire completed by the Chief Executive Officer (CEO) of a small enterprise. They propose to configure automatically business objects such as sites or regions of the enterprise. These data are called categories and are represented as abstract hierarchical entities. The first approach consists in building a decision tree from ERP expert interviews. The value of a parameter corresponds to a node of the tree which is associated to a question. The configuration is completed when the CEO reaches a leaf. The second approach consists in classifying the answers of the questionnaires. From each answer, they deduce which categories have to be added to the configuration. They use a Naive Bayes classifier trained with questionnaires completed by students.

Another automatic configuration tool is introduced in [Buc+10] for a material requirements planning software developed by SAP. Companies have generally a lot of difficulties to configure this kind of system because it requires many complex parameters and it is hard to benefit from the whole system when one is not an expert. The approach also uses decision trees built manually to find the good values of the parameter.

**Object oriented tool.** In [Ari+03], authors propose a tool to relate functional or organizational requirements of an enterprise with the configuration of an ERP. To do this, they model requirements and the configuration of the ERP system with an object oriented framework used to match requirements with configurations. The goal is to capitalize configurations of functionalities of different ERP systems with a user friendly interface. From the same interface, they can configure different ERP systems such as SAP R/3, PeopleSoft or Oracle Applications. The object oriented approach is used to model enterprise functions and enterprise schema which are directly linked with the user interface and ERP configurations. However, there is no automation to build the model and relate it with ERP configurations, which is clearly the most difficult part. Moreover, *Copilote* is much more adaptable and complex than the considered ERP systems.

## 3.2 Categories of requirements

When implementing *Copilote*, the configuration step consists in assigning values to parameters in such a way that the system fulfills the customer requirements described in the *requirement analysis* document. These requirements, for a given business process, may be divided into three main categories:

- Business logic requirements, *i.e.*, requirements that describe how the system must work during the business process;
- Operating requirements, *i.e.*, requirements that describe what information users need during the business process to do their job as well as possible;
- Reporting requirements, *i.e.*, requirements that describe how data collected during the business process must be aggregated and displayed for reporting and monitoring activities.

**Example 3.1.** We consider the purchase order process that consists, for a company, in purchasing materials needed for its activities. The main step of the purchase order process is to enter the order into the system.

A business logic requirement may be to specify how the system must compute the price of the material knowing that prices are often defined for a given period. Therefore, the reference date to compute the price is essential. A company may want to compute the price according to the date of order for 95% of the suppliers and according to the date of delivery for 5% of the suppliers.

When entering a new purchase order, an operating requirement may be to visualize the last ten purchase orders placed with the current supplier to know what were the last purchased materials, their prices and the last quantities the company ordered.

When analyzing purchase activities, a reporting requirement may be to visualize all the purchase orders done during the last 6 months, grouped by month of order and by supplier with the aggregated weight and price at each grouping level.

Operating requirements and reporting requirements are straightforward to configure in *Copilote*. There is no need of a deep knowledge of the system to configure this kind of data visualization even if it can be time consuming.

We focus on business logic requirements that are much more complex to configure since it is done with thousands of parameters that require a deep expertise of the system.

## 3.3 Architecture of the configuration of *Copilote*

We focus on the architecture of the parameters of *Copilote* that are used to configure the system to fulfill business logic requirements. Figure 3.1 shows the different levels of parameters embedded in *Copilote*.

*Copilote* is divided into 7 business modules: acquisition, sales, production, *Warehouse Management System* (WMS), financial, *Customer Relationship Management* (CRM) and *Electronic Data Interchange* (EDI). Each business module has the same configuration structure.

Functional module parameters are the highest level of parameters of a business module. They are used to activate high level functionalities such as export functionalities, for instance. Each functional module corresponds to a part of *Copilote* that customers can purchase when they choose the scope of *Copilote* they want. *Copilote* contains 232 functional module parameters.

Each of the seven business modules has hundreds of general parameters that are used to configure business logic into the processes handled in the module. Many general parameters depend on a functional module parameter: They can be configured only if their functional module parameter is activated. Table 3.1 details the number of parameters of these modules. For each module, we give the number of parameters according to their type:

- *Symb* gives the number of symbolic parameters, *i.e.*, parameters that take their value within a finite list of values;
- *Ref* gives the number of parameters that are a reference towards another database object;

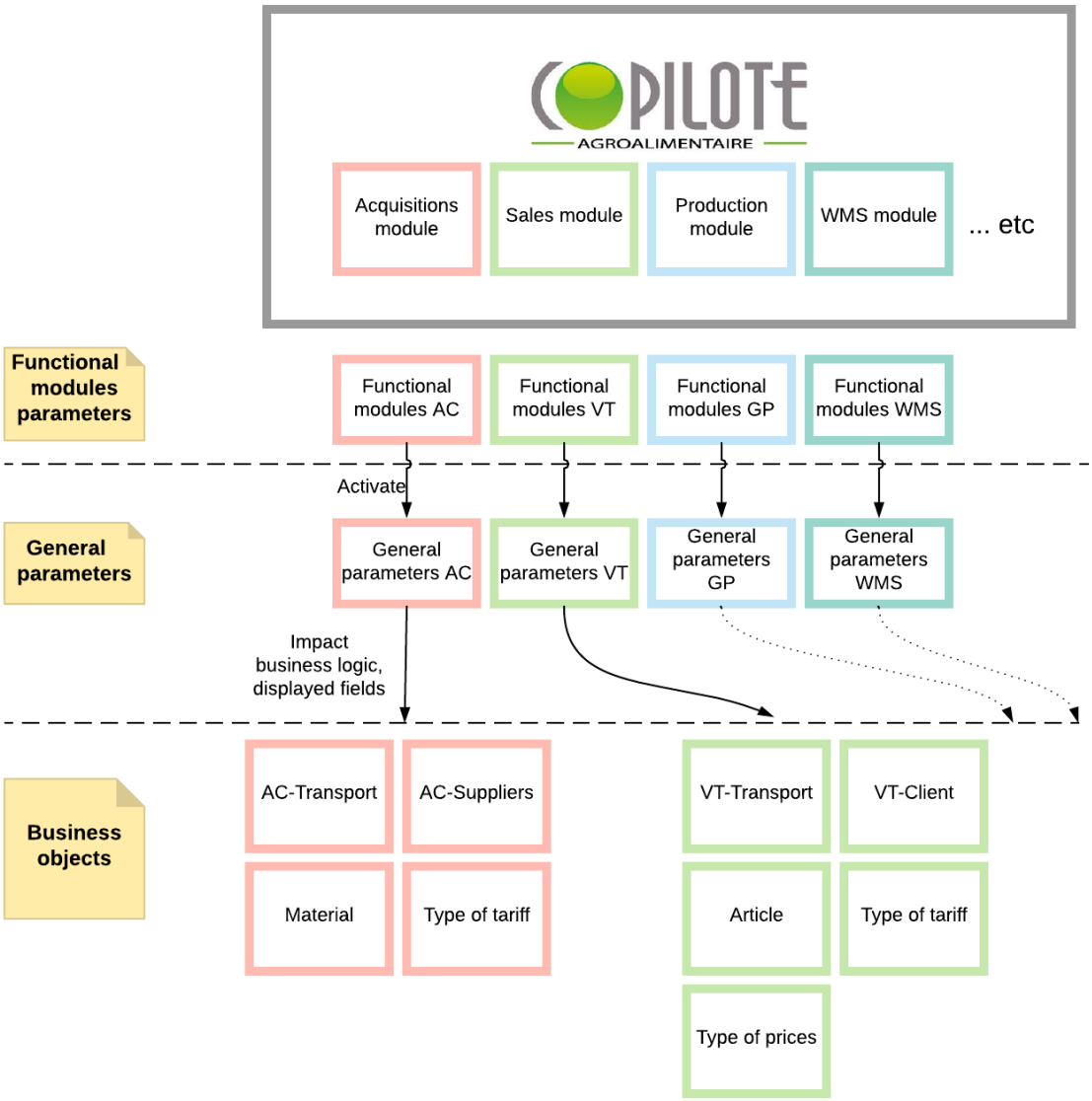



Figure 3.1 – Architecture of the different levels of parameters of *Copilote*.

- *Domain* gives the number of parameters that can take a value within a specific domain such as numeric, date or text parameters;
- *Multi* gives the number of multi-valued parameters, *i.e.*, parameters that are a list of objects.

Acquisition and sales modules have more general parameters than other modules because they cover much more business processes. 46% of the general parameters of *Copilote* are symbolic parameters. All these parameters are very important during the configuration step since they are used to specify the business logic of the system.

Each module has many different business objects that are manipulated, modified and created by *Copilote* during business processes. These business objects embed a lot of data that are essential for the business logic such as, for example, lead time, pricing details, billing details, or country of the supplier. This data is also considered as parameters.

Table 3.1 – Detail of the number of general parameters for each module.

| Module | Symb | Ref | Domain | Multi | Total |
|---|---|---|---|---|---|
| Acquisition | 604 | 282 | 232 | 91 | 1,209 |
| Sales | 1,167 | 525 | 303 | 165 | 2,160 |
| Production | 149 | 63 | 55 | 21 | 288 |
| Financial | 141 | 240 | 143 | 11 | 535 |
| Warehouse | 222 | 191 | 177 | 88 | 678 |
| CRM | 18 | 89 | 17 | 6 | 130 |
| EDI | 3 | 11 | 12 | 3 | 29 |
| Total | 2,304 | 1,401 | 939 | 385 | 5,029 |

Moreover, part of the business logic specified in the global parameters layer can be overwritten at the business objects level. This is especially useful to handle particular cases. During a business process, *Copilote* considers first the business logic of the business objects involved in the process and then the logic coming from the general parameters.

**Example 3.2.** If we consider the business logic requirement described in Example 3.1, a symbolic general parameter called *Price reference date* is used to specify what date must be taken into account to compute the price of the materials. The same field exists in the supplier business object that allows to consider another reference date for a particular supplier.

Therefore, we set the value "Date of order" in the general parameter and for the 5% of suppliers concerned, we set the value "Date of delivery" in the field of their business object. This way, the date of order is considered to compute material prices except for the suppliers for which we specified "Date of delivery" in their object.

**Example 3.3.** The trade of goods declaration is necessary for companies that export or import materials in the European Union. To configure the trade of goods declaration functionality, an integrator has first to activate the corresponding functional module.

Once this is done, general parameters that concern trade of goods declaration are visible in the general parameter screen of sales module and acquisition module. These parameters allow to specify how to do the declaration of the company. For instance, a parameter is used to specify whether the company declares sales item samples into the declaration or not.

Finally, a field for each supplier (resp. customer) business object specifies whether trades with this supplier (resp. customer) have to be taken into account in the declaration. Then, to test if the declaration is configured as expected, the integrator has to enter orders with these suppliers and generate the declaration to check if it is well done.

## 3.4 Methodology

We focus on how system integrators configure the system according to the *requirement analysis* document. To do this, we followed integrators during the implementation process of *Copilote*.

When system integrators start to configure the system for a customer, they focus on one single business process described in the *requirement analysis* document. Most of the time, they achieve the following tasks:

1. They activate the functional module needed for the process;
2. They configure operating requirements and reporting requirements;
3. They configure business logic requirements by setting values to general parameters to handle the general case;
4. They configure business objects needed for particular cases and to run the process for testing the configuration;
5. They execute as often as necessary the process to test all the cases that need to be handled by *Copilote*.

**Example 3.4.** If we consider our running example described in Example 3.1, as we do not need to activate any functional module, we start by configuring the view of the last 10 purchase orders for a given supplier and we add it to the purchase order screen.

Then, we configure the view that groups the purchase orders of the last 6 months by month and by supplier with the price and the quantity. Both views are done with the decisional tool of *Copilote* which is straightforward to use.

Then, we configure the general parameter *Price reference date* as explained in Example 3.2 and we enter new suppliers, some of which with the value "Date of delivery" in their *Price reference date* field.

Finally, we need to enter some materials, some pricing, some delivery details for each of the suppliers. Once this is done, we may enter a new order on a normal supplier, another one on a supplier having "Date of delivery" specified and we have to verify that *Copilote* works as expected. If it computes the good prices, we configure the next requirements, otherwise, we try to find out what parameters have to be modified to correct the problem.

Our running example is trivial since the requirement can be configured by instantiating only one parameter. However, most of the time it involves more parameters and is much harder to configure. Assigning the right values to the parameters in order to fulfill a requirement may be a very hard task, particularly when an integrator faces a requirement for the first time.

Our example still shows that the testing part may be time consuming since many business objects may need to be entered in the system to verify all the cases to handle. Moreover, each time the configuration is modified, a system integrator has to verify again all the cases to handle. Any mistake when configuring the system is time-consuming to both correct the mistake and verify that the system works as expected.

## 3.5 Focus on general parameters of *Copilote*

We focus on the general parameters of the acquisition module of *Copilote* to understand how they are configured by integrators. As shown in Table 3.1, the acquisition module contains more than 1,200 parameters and most of them are symbolic parameters.

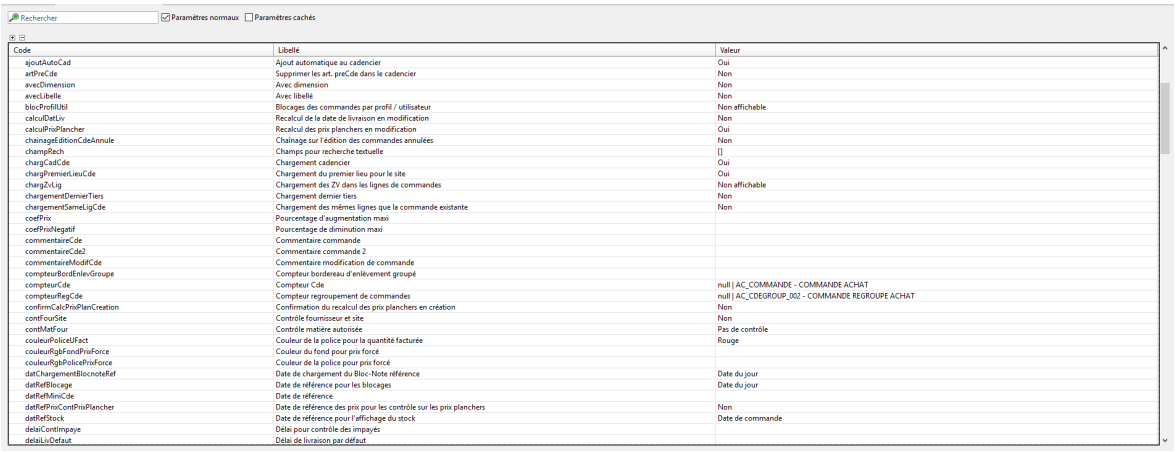



Figure 3.2 – Configuration screen of the general parameters of the acquisition module

**Configuration screen.** A screen is dedicated to the configuration of the general parameters of the acquisition module. As shown in Figure 3.2, the screen displays a list of all the parameters. There is only a text field to filter the parameters according to their label. Therefore, when configuring a requirement for the first time, a system integrator either finds a detailed and clear documentation that explains how to configure it, which rarely exists, or enter some keywords into the filter to hopefully find the right parameters to configure. System integrators may search for a long time into the huge amount of parameters what parameters need to be configured, and this may be frustrating.

Even when a system integrator has already configured the same requirement in a previous implementation, if he does not remember exactly how he configured it, he still has to remember for which customer he did it and then identify what parameters were concerned to find out how he configured it.

In this screen, the way parameters are displayed makes it hard to understand the business logic implied by the current configuration. To understand the meaning of a parameter value, system integrators need to visualize all the parameters that impact the same business logic. Displaying all the parameters in a list buries their business logic meaning into a huge amount of information.

Furthermore, parameters that are very important for the business logic are not distinguished from detail parameters. For instance, the parameter *Price reference date*, which is critical, is next to the parameter that specifies the font color of the purchase order screen.

**Dependencies between parameters.** Many dependencies exist between general parameters because some values may be incompatible from a business logic point of view. We detail two frequent dependencies.

- Master/slave activation: A slave parameter can be instantiated only if a master parameter is assigned to a given value. For instance, parameters concerning wine industry can be assigned only if the boolean parameter that activates wine functionalities is set to *true*. Otherwise, the slave parameter is not activated.

- Master/slave dependency: The value of the slave parameter is reset when the value of the master parameter is modified. For instance, when the parameter

that specifies the packing site is changed, the parameter that gives the packaging station is reset to null because only stations of the packing site can be chosen for this parameter.

Some of these dependencies are encoded directly in the general parameters screen to help the system integrator avoid mistakes on business logic coherency but still many mistakes are done which have, sometimes, bad consequences on customer activities. It would be a tremendous work to identify and encode manually all the dependencies that exist between this huge amount of parameters. However, identifying the most critical or the most frequent configuration mistakes in order to add, into the system, dependencies that prevent to make them again would probably improve the overall quality of the configuration.

Identifying automatically all the dependencies between parameters, which is a complex issue, would be an interesting problem to solve and many different techniques could be applied. These dependencies may allow one to know if a configuration is correct from a business logic point of view. However, this does not mean that the configuration is correct according to the requirements of the customer which is our main objective.

**Analysis of existing configurations.** We have collected 500 existing configurations of the general parameters of the acquisition module from existing implementations of *Copilote* in order to analyze how they are instantiated.

The analysis of existing configurations shows us that a significant part of these parameters are used by only few customers. Indeed, 110 parameters are used by only one customer (*i.e.*, instantiated in only one configuration), 200 by less than five customers and 240 by less than ten customers. When focusing on these parameters, we find out that they are dedicated to functionalities developed for a particular field of activity. For instance, 40 parameters are used by the only customer of Infologic that works in the wine industry. Therefore, part of the general parameters are very specific to fields of activity and are used by only few customers of Infologic. Most of the time, these parameters could be ignored by system integrators if they are not relevant for the current customer to configure.

When focusing on the values of the parameters, some parameters have a dominant value that is assigned in more than 90% of the configurations whereas some parameters have values almost never used which lead us to believe they correspond to very specific requirements. It would be very useful to know, when assigning a value to a parameter, the distribution of the assigned values in all the configurations.

Furthermore, around 15% of these parameters have a constant value, *i.e.*, they have the same value in all the configurations. These parameters are then useless and may be removed. When trying to understand the origin of these useless parameters, it turned out that it comes from the fact that this module is symmetric with the sales module since processes are very similar in both modules: when a functionality is developed for the sales module, it is, most of the time, developed in the acquisition module as well, and general parameters of the functionality are added to the module. However, some of these functionalities are never used.

We could think that two companies within the same activity field would have similar requirements and then similar configurations but this is not true. They may have very similar requirements for some parts of business processes but most of the requirements are independent from the business sector.

## 3.6 Discussion

Existing configuration tools are not suitable for Infologic because the model on which they are based are either built manually or based on questionnaires completed by students.

When we focus on the configuration of a business process in *Copilote*, it appears that the complexity remains in finding, from thousands of parameters, the subset of parameters that impact the considered business logic and then in choosing the right values to assign to these parameters in order to fulfill the requirement.

Obviously, simple improvements of the system could be done to help integrators during the configuration step:

- For a given parameter, by showing the values it takes in other configurations;
- By warning integrators when they are doing a configuration never done before;
- By displaying first the parameters used by more than $x\%$ of the customers.

Infologic started to develop a tool to provide these information which may help system integrators. However, this tool does not help them to understand the business logic induced by a configuration, which requires a deep expertise of *Copilote* and of business logic. Moreover, experts with a deep knowledge of *Copilote* tend to be extremely rare resources and we cannot base our approach only on them. In the next chapter, we present a new approach which exploits existing configurations of *Copilote*: These existing configurations contain the knowledge needed to configure *Copilote* according to the requirements of the customers of the corresponding implementations.

# Chapter 4

# Proposed approach: From Configurations to Requirements

## Contents



Infologic is convinced that it is possible to significantly reduce configuration time by developing a tool for reusing parts of previous configurations when implementing *Copilote* for a new customer. This intuition is confirmed by Daneva [Era+15] who measured requirements reuse, and found out that even if full reuse was not achieved, the rate of reuse could be remarkably high in some cases.

More precisely, the tool should guide integrators during the requirement analysis by proposing questions to ask to customer managers in order to collect efficiently the requirements. From each of these requirements, the tool should identify the corresponding configuration part used in previous implementations (if it exists) and automatically reuse it for the new case. In other words, the goal is to collect a catalog of business logic requirements with their corresponding configuration parts in order to reuse them during new implementations of *Copilote*.

The first way to build this catalog is to ask integrators to identify, one by one, all the requirements and their corresponding configuration parts they have implemented before. This solution is not an option for Infologic because this work would be amazingly complex and would be too time-consuming for system integrators.

The second way is to extract automatically, from existing configurations, parts of configurations that correspond to business logic requirements. More precisely, the main idea of this approach is to collect all existing configurations and apply data mining techniques to extract relevant parts of configurations to be interpreted by expert integrators and related to corresponding business logic requirements. This second proposition is more suitable for Infologic since it is much less time consuming for system integrators which are a critical resource.

In this chapter, we propose a new approach to assist the configuration of *Copilote*. In Section 4.1, we introduce the concept of business unit map, which is a set of functional goals which are structured thanks to a refinement relation. In Section 4.2, we show how to exploit this map during the implementation process of *Copilote*. Finally, in Section 4.3, we introduce our approach that consists in extracting, from previous configurations, parts of configurations that may correspond to business logic requirements in order to reuse them for new implementations.

## 4.1 Map of the business units of *Copilote*

To be able to extract relevant parts of configurations and then to reuse them to assist the configuration of *Copilote* from customer requirements, a new abstraction is necessary to relate parameters to requirements. As seen in Section 3.2, business logic requirements are customer requirements on the business logic applied by *Copilote* during a process, and this business logic is achieved by setting parameters. However, the connection between parameter settings and business requirements is not explicitly stated. Therefore, we propose to identify every part of a business process for which a business logic may be configured into *Copilote*. Such a part may be seen as the unit of reasoning of system integrators during the configuration process, and we call it a *business unit*. This concept of business unit is inspired by the definition of a *functionality* in [Rol+01] where authors propose a system to link ERP functionalities with customer requirements.

The goal is to obtain a complete map of the business units of *Copilote* and use this map all along the implementation process, particularly when collecting the customer requirements and configuring the system.

**Definition 4.1** (Business unit)**.** A *business unit* is defined as a goal to be achieved by *Copilote*. There may exist several ways for achieving this goal with *Copilote*, and each of them may be specified by a business logic requirement.

We denote $\mathcal{B}$ the set of all the business units of *Copilote*.

**Example 4.2.** For instance, *Enter a sales order* is a business unit. It can be achieved in many different ways. Two different business logic requirements for achieving this business unit are: (1) a telemarketer answers a customer call and uses *Copilote* to enter a new order; (2) a customer uses a web portal to directly enter a new order.

We may consider different levels of granularity when defining business units, and a high level business unit may be refined into a set of more detailed business units.

**Definition 4.3** (Refinement of a business unit)**.** The function $\mathit{refinement} : \mathcal{B} \to \mathcal{P}(\mathcal{B})$ associates with every business unit $bu \in \mathcal{B}$ a (possibly empty) set of business units $\{bu_1, \ldots, bu_n\} \subset \mathcal{B}$.

From a business logic point of view, the refinement function must not contain cycles. More formally, it defines a partial order on $\mathcal{B}$, and the corresponding graph is called the *map* of $\mathcal{B}$.

**Definition 4.4** (Business unit map)**.** The *business unit map* is a graph $G_{\mathit{refinements}} = (\mathcal{B}, E)$ such that $\mathcal{B}$ is a set of business units, and $E$ is the set of directed edges defined

by the refinement function, *i.e.*, $E = \{(bu_i, bu_j) \in \mathcal{B} \times \mathcal{B} | bu_j \in \mathit{refinement}(bu_i)\}$. $G_{\mathit{refinements}}$ must be a *Directed Acyclic Graph (DAG)*, *i.e.*, there must not exist a subset of $n \geq 2$ business units $\{bu_1, \ldots, bu_n\} \subseteq \mathcal{B}$ such that $\forall i \in [2, n], (bu_{i-1}, bu_i) \in E$ and $(bu_n, bu_1) \in E$.

Note that $G_{\mathit{refinements}}$ is a DAG and not a tree because a business unit may be used to refine several higher level business units.

We have designed and implemented a tool for assisting the construction of this map. The description of this tool is beyond the scope of this thesis, and we only use the resulting map. This tool has been used by an expert integrator to build a part of the business unit map of *Copilote*. This incomplete map only contains the business units related to three modules, *i.e.*, the acquisition, sales and production modules, which nearly correspond to half of the *Copilote* modules. The part of the map corresponding to the acquisition (resp. sales and production) module contains 48 (resp. 71 and 166) business units.

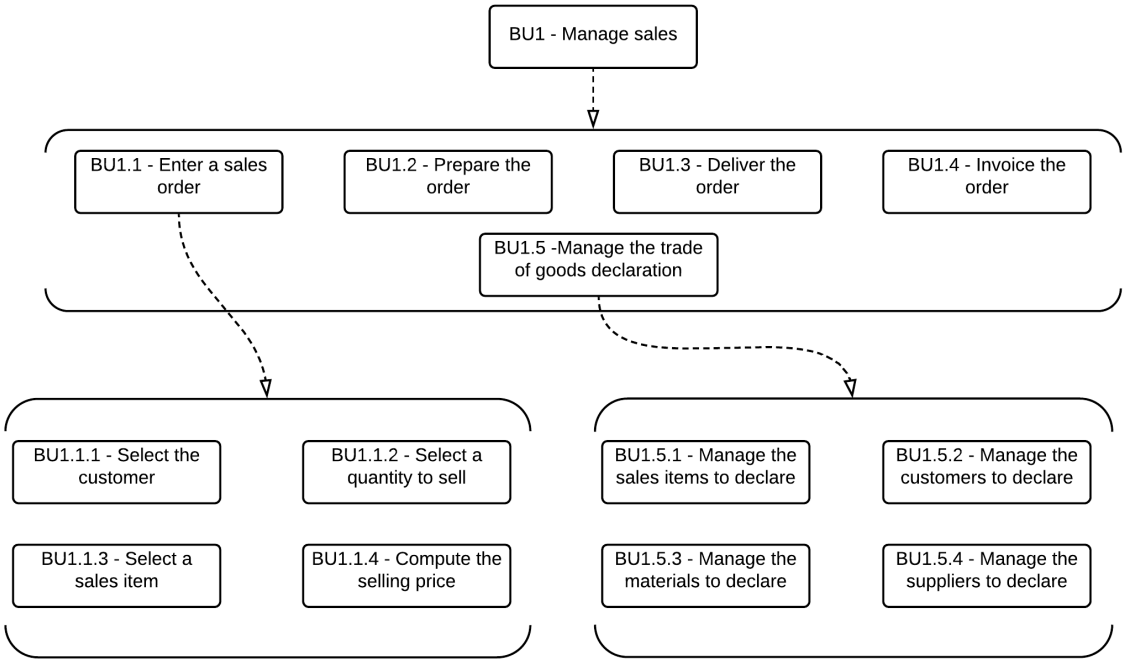



Figure 4.1 – Part of the business units map of the sales flow process in *Copilote*.

**Example 4.5.** Fig. 4.1 shows a small part of the business unit map of *Copilote*. The highest level unit is $BU1$ (*Manage sales*) which corresponds to the sales process. This process is refined into five more precise business units, *i.e.*, $\mathit{refinement}(BU1) = \{BU1.1, BU1.2, BU1.3, BU1.4, BU1.5\}$ where $BU1.1$ is *Enter a sales order*, $BU1.2$ is *Prepare the order*, $BU1.3$ is *Deliver the order*, $BU1.4$ is *Invoice the order*, and $BU1.5$ is *Manage the trade of good declaration*. Actually, $BU1$ is refined with much more *business units* but we display only a few of them in Fig. 4.1 to simplify it.

All these business units still correspond to high level units and are refined into more precise business units. For instance, $BU1.1$ is refined into four lower level business units, *i.e.*, $\mathit{refinement}(BU1.1) = \{BU1.1.1, BU1.1.2, BU1.1.3, BU1.1.4\}$ where

*BU*1.1.1 is *Select the customer*, *BU*1.1.2 is *Select a sales item*, *BU*1.1.3 is *Select a quantity to sell*, and *BU*1.1.4 is *Compute the price*.

For each of these lower level business units, there may exist different business logic requirements. For instance, for the business unit *BU*1.1.1, business logic requirements may be:

- Block customers that have not paid all their invoices;
- Display a pop-up with specific information that concern customers when a user selects them;
- Group customers by department in the selection list.

## 4.2 Use of the map

The business unit map is a model of all capabilities of *Copilote*. We propose to use this model to (i) structure the requirement analysis step (as explained in Section 4.2.1), and (ii) bridge the gap between business units and parameters (as explained in Section 4.2.2).

### 4.2.1 Collecting the requirements

The business unit map of *Copilote* may be used to guide the requirement analysis step. As explained in Section 2.4, the objective of the requirement analysis step is to collect all the requirements needed to configure the business logic of *Copilote* as expected by the customer. In other words, it consists in collecting, for all business units in $\mathcal{B}$, the requirements needed to configure the business logic of the units.

We propose to use the DAG structure of $G_{refinements}$ to guide system integrators when collecting customer requirements. A system integrator starts to collect the requirements by selecting the business unit $bu \in \mathcal{B}$ that represents the goal achieved by the process he wants to analyze. Then, the goal is to refine as much as possible the customer requirements.

More formally, for a given business unit $bu \in \mathcal{B}$, Algorithm 1 details how to collect recursively all the requirements needed to configure $bu$. When analyzing a business unit $bu$, the first point is to know if the customer is concerned by the goal achieved by $bu$ (line 3). If it is not the case, there is no need to collect requirements. Otherwise, if $bu$ is not refined, all the requirements needed to configure $bu$ need to be collected (line 5). If $bu$ is refined, all the business units that refine $bu$ must be analyzed in the same way (line 8).

**Example 4.6.** We consider the analysis of the requirements concerning the sales process. According to Figure 4.1, the business unit *Manage sales* represents the goal achieved by this process. The system integrator (denoted *SI*) interacts with the customer (denoted *C*) by following this predefined framework:

- *SI* asks if *C* is concerned by this process. Let us assume that this is the case.
- *SI* focuses on the refinement of *Manage sales*.

**Algorithm 1:** Analysis of a *business unit*

```
1 Procedure Analyze(G_refinements, bu)
    Input: A business unit map G_refinements = (B, E) and a business unit bu to analyze
2   begin
3     if The customer is concerned by bu then
4       if bu is a leaf in G_refinements then
5         Ask all the requirements needed to configure bu
6       else
7         for each (bu, bu_i) ∈ E do
8           Analyze(G_refinements, bu_i)
```

- *SI* asks if *C* has any specificities when *Entering a sales order*. Let us assume that this is the case.
- *SI* focuses on the refinement of *Enter a sales order*.
- *SI* considers the business unit *Select the customer* and collects all the requirements associated with this business unit. This is done by asking the following questions to *C*:
  - How do you select the customer on which you want to enter a new sales order?
  - Is there any control to do on the customer associated with a sales order?
  - Do you allow to enter a sales order for a customer that has unpaid invoices?

This way of collecting the requirements provides many advantages:

- The map gives a complete view of the business units that have to be analyzed with the customer. We can easily know the progress of the requirement analysis and what business units remain to be done.
- Each requirement is related to the business unit it impacts. We obtain as output a set of business units to be configured together with their requirements. This defines clearly the scope of the implementation process.

The method can easily be improved by providing, for each business unit, the questions a system integrator has to ask to collect efficiently and entirely the requirements.

### 4.2.2 Business logic scope of the parameters

In the current version of *Copilote*, there is no way to know or even to describe the scope of the business logic impacted by a parameter. This information is essential to be able to focus, for a given business unit, only on parameters that may impact it. Therefore, for each parameter $p$, we propose to identify all the business units whose business logic may be impacted by $p$. We call this set of business units the *business logic scope* of $p$.

More formally, let $P$ be the set of all parameters of *Copilote*. We define the function $scope : P \rightarrow \mathcal{P}(\mathcal{B})$ that gives for each parameter the set of *business units* it impacts.

We define the dual function *parameters* : $\mathcal{B} \rightarrow \mathcal{P}(P)$ that gives, for each business unit in $\mathcal{B}$, the set of the parameters that impact its business logic, *i.e.*, $bu \in scope(p) \Leftrightarrow p \in parameters(bu)$.

We have designed and implemented a tool for associating business units with parameters. We decided to focus on a subpart of *Copilote*: general parameters of the sales modules and parameters of the model of production planning business object. The task of associating a business logic scopes with the rest of the parameters is still in progress: It is time consuming since *Copilote* has thousands of parameters and only experts are able to do this critical job.

Our goal is to use this mapping between parameters and business units to restrict the list of parameters that are displayed to the system integrator: When configuring a business unit *bu*, *Copilote* will only display the parameters that may impact the business logic of *bu*, *i.e.*, $parameters(bu)$.

## 4.3 From configurations to requirements

A business unit defines a goal which may be achieved by different parameter settings depending on the customer needs (*i.e.*, the business logic requirements), but also on the system integrator who has configured *Copilote*. In this section, we introduce an approach for extracting from existing configurations of *Copilote* a catalog of configuration parts related to a description of the corresponding functional needs.

More precisely, let $D$ be the database of $n$ existing configurations of *Copilote*. Each configuration in $D$ corresponds to a different customer and gives the instantiation of the parameters for this customer. In other words, $D$ is a matrix which contains a row for each customer $c \in [1, n]$ and a column for each parameter $p \in P$ such that $D[c][p]$ is the value assigned to parameter $p \in P$ in the configuration of *Copilote* used by customer $c$.

Our goal is to extract relevant configuration parts from $D$. Let us first define what is a configuration part.

**Definition 4.7** (Configuration part)**.** A configuration part is defined by a couple $(P', I)$ such that $P' \subseteq P$ is a subset of parameters and $I$ is an instantiation of these parameters such that there exists at least one customer for which this instantiation has been used, *i.e.*, $\exists c \in [1, n], \forall p \in P', I(p) = D[c][p]$.

There exists an exponential number of configuration parts and our goal is to extract only those that are *relevant*. We may consider different criteria to evaluate the relevancy of a configuration part $(P', I)$. Some of these criteria may be simple measures such as, for example:

- The number of customers for which the part has been used, *i.e.*, $\#\{c \in [1, n] | \forall p \in P', D[c][p] = I(p)\}$;
- The number of parameters in $P'$, *i.e.*, $\#P'$.

An important criterion is the cohesion of the parameters in $P'$. In particular, we restrict our attention to configuration parts $(P', I)$ such that all parameters in $P'$ are related to a same business unit, *i.e.*, $\exists bu \in \mathcal{B}, P' \subseteq parameters(bu)$.

Other criteria may be more difficult to define in an *a priori* way, and we propose to interact with an expert of the configuration of *Copilote* to identify *relevant* configuration parts as follows:

1. Define a first set $C$ of constraints and criteria for identifying relevant configuration parts;
2. Apply data-mining techniques on $D$ to extract a set $S$ of the most relevant configuration parts according to $C$.
3. Ask the expert to select the configuration parts in $S$ that correspond to business logic requirements;
4. If the expert judges that $S$ contains irrelevant configuration parts, interact with the expert to update the set $C$ of constraints and criteria used to mine relevant configuration parts, and return to step 2.

Finally, for each relevant configuration part $(P', I)$ that has been selected by the expert, the expert has to describe the requirements that are fulfilled by the instantiation $I$ of the parameters $P'$. The resulting set of all selected configuration parts together with the description of the corresponding requirements is called a *catalog of configurations*.

This catalog of configurations capitalizes on the experience of all integrators that have implemented configurations in the database $D$. It will be used to assist system integrators for new implementations of *Copilote*. During the requirement analysis of a business unit, system integrators can choose directly, from the configuration catalog, the configuration part that corresponds to the business logic requirement of the customer if it exists. This configuration part $(P', I)$ can be used to automatically instantiate every parameter $p \in P'$ to $I(p)$.

## 4.4 Discussion

We have introduced in this chapter an approach for reusing previous configurations in order to assist *Copilote* integrators during the configuration step. The basic idea is to guide the implementation of *Copilote* by means of a business unit map. This map is used during the requirement analysis step to structure the interview of the customer for identifying all requirements associated with the business units that are targeted for this customer. This map is also used to identify the parameters which impact a business unit. Finally, a catalog of configurations is used to propose to the integrator some configuration parts that correspond to the business logic requirement of the customer.

A first proof of concept of the approach has been implemented. In particular, we have implemented a tool for assisting the construction of the business unit map, and we have used this tool to build this map for a subset of *Copilote*. We have also identified, for each business unit of this partial map, the set of parameters which impact it.

The most challenging part of our approach is to use data mining techniques to identify relevant configuration parts. As explained in the previous section, constraints and criteria used to define relevancy are not completely known and we need to interact with an expert to identify them. This data mining task constitutes the technical core of this thesis. To achieve it, we propose to use constraint programming because this

declarative framework allows us to easily integrate feedbacks of experts, by means of new constraints and criteria.

The database $D$ which is mined mainly contains symbolic data since most of the parameters described in Table 3.1 are symbolic. Therefore, we focus on symbolic data mining methods and we propose to use Formal Concept Analysis (FCA) which groups together objects sharing a same set of attribute values [Gan+97]. The aim is to group configurations that fulfill the same requirement together and identify the corresponding part of configuration.

# Part II

# Technical Context

We propose to use constraint programming (CP) to identify relevant configuration parts from a database of existing configurations of *Copilote*. This problem mainly involves solving a clustering problem. CP allows us to define our problem in a declarative way by means of variables and constraints. Since our approach is experimental, CP constitutes a flexible framework that allows us to easily and interactively modify and adapt our model according to *Copilote* expert feedbacks.

In this part, we describe the technical context of our work. In Chapter 5, we introduce the basic *Branch and Propagate* generic algorithm used in classical CP solvers to solve constraint satisfaction problems, and we describe different ingredients that are used in our CP models, *i.e.*, set variables, global constraints, and approaches for solving optimization problems with CP.

In Chapter 6, we describe Formal Concept Analysis, the idea of which is to group together objects that share a same set of attribute values. We more particularly focus on conceptual clustering that aims at partitioning a set of objects into homogeneous and well separated clusters such that each cluster is described by a set of attribute values shared by all its objects. We also describe existing declarative approaches to solve conceptual clustering problems.

In Chapter 7, we describe the Exact Cover problem that aims at selecting a subsets of objects that defines a partition of a set of objects and we show how conceptual clustering may be seen as an Exact Cover problem. We describe some dedicated and declarative approaches for solving this problem.

# Chapter 5

# Constraint Programming

## Contents



We propose to use Constraint Programming (CP) to mine relevant configurations parts from existing *Copilote* configurations. CP allows us to define our problem in a declarative way by means of variables and constraints. This kind of problems defined by means of constraints are called Constraint Satisfaction Problems (CSPs), and are described in Section 5.1. CSPs are solved by generic algorithms which are usually based on a *Branch and Propagate* principle: The propagation step, described in Section 5.2, exploits constraints to simplify the problem; The branching step, described in Section 5.3, decomposes the problem into subproblems which are recursively solved. In Section 5.4, we describe set variables, *i.e.*, variables that represent sets of values, and we show how their domains may be approximated by set intervals in order to efficiently propagate constraints on them. In Section 5.5, we describe global constraints, which are a key ingredient of the success of CP: They provide both a compact way for modeling complex relations and an efficient way for propagating them. In Section 5.6, we introduce Constrained Optimization Problems and Multi Objective Constrained Problems, and we show how these problems may be solved with CP approaches. Finally, in Section 5.7 we briefly describe how to implement a new global constraint in a CP library.

Many definitions and notations introduced in this chapter are taken from [Ros+06], and we refer the reader to this handbook for more details on CP.

Given a set $S$, we denote $\#S$ the cardinality of $S$ and $\mathcal{P}(S)$ the set of all subsets of $S$. Given two integer values $lb$ and $ub$, we denote $[lb, ub]$ the set of all integer values ranging between $lb$ and $ub$.

## 5.1 Constraint Satisfaction Problems

A *Constraint Satisfaction Problem* (CSP), also called *constraint network*, involves assigning values to variables so that constraints are satisfied. Each variable has a domain, which is the set of values that may be assigned to it. In this thesis, we only consider finite domains and, without loss of generality, we assume that all domains only contain integer values, *i.e.*, are finite subsets of $\mathbb{Z}$.

More formally, let us first define what is a constraint.

**Definition 5.1** (Constraint)**.** A constraint $c$ is a relation defined on a sequence of variables $X(c) = (x_{i_1}, \ldots, x_{i_{\#X(c)}})$, called the *scheme* of $c$. $c$ is the subset of $\mathbb{Z}^{\#X(c)}$ that contains the combinations of values $\tau \in \mathbb{Z}^{\#X(c)}$ that satisfy $c$, denoted $sol(c)$. $\#X(c)$ is called the *arity* of $c$.

A constraint may be defined in intention, by using mathematical operators, or in extension, by listing all the tuples in the relation.

**Example 5.2.** The constraint $c \equiv x_1 < x_2 \wedge x_2 < x_3$ is the relation defined in intention that contains every tuple $(a, b, c) \in \mathbb{Z}^3$ such that $a < b$ and $b < c$. The tuple $(4, 7, 8)$ satisfies $c$ whereas the tuple $(4, 1, 2)$ does not satisfy $c$. The arity of $c$ is 3.

The constraint $c' \equiv \{(1, 2, 3), (2, 1, 3)\}$ is the relation defined in extension which is satisfied by two tuples: (1,2,3) and (2,1,3).

**Definition 5.3** (CSP)**.** A CSP is defined by a triple $(X, D, C)$ such that:

- $X = (x_1, \ldots, x_n)$ is a finite sequence of integer variables;
- $D = D(x_1) \times \ldots \times D(x_n)$ is the domain for $X$, where $D(x_i) \subset \mathbb{Z}$ is the finite set of values that may be assigned to the variable $x_i$;
- $C = \{c_1, \ldots, c_m\}$ is a set of constraints such that, for each constraint $c_i \in C$, every variable in $X(c_i)$ belongs to $X$.

A CSP is *binary* if all its constraints involve two variables, *i.e.*, $\forall c_i \in C, \#X(c_i) = 2$

The variables of a CSP and the scheme of a constraint $c_i$ are sequences of variables and not sets because the order of values matters for tuples in $D$ or $c_i$. However, we may use set operators on sequences. In particular, given two constraints $c_i$ and $c_j$, we denote $X(c_i) \subseteq X(c_j)$ the fact that every variable in the scheme of $c_i$ also belongs to the scheme of $c_j$, whatever their order in the schemes. Also, given a constraint $c$ and a variable $x$, we denote $x \in X(c)$ the fact that $x$ belongs to the scheme of $c$.

Given a tuple $\tau$ on a sequence of variables $Y$, and another sequence of variables $W \subset Y$, we denote $\tau[W]$ the restriction of $\tau$ for the variables of $W$, ordered according to $W$. Given a variable $x_i \in Y$, $\tau[x_i]$ denotes the value of $x_i$ in $\tau$.

Solving a CSP involves assigning variables to values so that constraints are satisfied.

**Definition 5.4** (Instantiation)**.** Let $(X, D, C)$ be a CSP.

- An *instantiation* $I$ on $Y = (x_1, \ldots, x_k) \subseteq X$ is an assignment of values $v_1, \ldots, v_k$ to the variables $x_1, \ldots, x_k$. $I$ is a tuple and may either be denoted $((x_1, v_1), \ldots, (x_k, v_k))$ or $(I[x_1], \ldots, I[x_k])$.

- An instantiation $I$ on $Y$ is *valid* if for all $x_i \in Y, I[x_i] \in D(x_i)$.
- An instantiation $I$ on $Y$ is *partial* if $Y \subset X$ and *complete* if $Y = X$.
- An instantiation $I$ on $Y$ is *locally consistent* if it is valid and for every $c_i \in C$ such that $X(c_i) \subseteq Y$, $I[X(c_i)]$ satisfies $c_i$. If $I$ is not locally consistent, it is *locally inconsistent*.
- A *solution* is a complete instantiation $I$ on $X$ which is locally consistent. The set of solutions of $(X, D, C)$ is denoted $sol(X, D, C)$.
- An instantiation $I$ on $Y$ is *globally consistent* (or consistent) if it can be extended to a solution (*i.e.*, there exists $I' \in sol(X, D, C)$ with $I = I'[Y]$).

**Example 5.5.** As a running example, we use the 4-queens problem that aims at placing 4 queens on a $4 \times 4$ chess board in a way that no two queens can attack each other. Different CSPs may be used to model this problem. A classical CSP is $(X, D, C)$ with

- $X = (x_1, x_2, x_3, x_4)$;
- $D(x_i) = \{1, 2, 3, 4\}$ for all $i \in [1, 4]$;
- $C = \{c_{i,j} | \{x_i, x_j\} \subset X\}$ where

$$c_{i,j} \equiv (x_i \neq x_j) \wedge (|x_i + i| \neq |x_j + j|) \wedge (|x_i - i| \neq |x_j - j|).$$

In other words, this model associates a variable $x_i$ with every column $i \in [1, 4]$, as we know for sure that there is exactly one queen per column, and the value assigned to this variable corresponds to the row of this queen. There is a constraint $c(x_i, x_j)$ between every pair of queens. The first (resp. second, and third) part of this constraint ensures that all queens are placed on different rows (resp. different downward diagonals, and different upward diagonals). This CSP has two solutions, *i.e.*, $sol(X, D, C) = \{(2, 4, 1, 3), (3, 1, 4, 2)\}$.

For this CSP, the partial and valid instantiation $I = (1, 3)$ on $Y = (x_1, x_2)$ assigns $x_1$ to 1 and $x_2$ to 3. $I$ is locally consistent because it satisfies the constraint $c_{1,2}$. However, $I$ is not globally consistent because it cannot be extended to a solution.

## 5.2 Constraint Propagation

CSPs are usually solved by backtracking search algorithms which are described in the next section. These algorithms explore all possible instantiations in a systematic way. As the number of valid instantiations is exponential in the number of variables, this exhaustive exploration is combined with constraint propagation techniques which exploit constraints to reduce the search space.

More precisely, the propagation of a constraint $c$ aims at filtering its variable domains by removing values that cannot belong to solutions. After this propagation step, the filtered domains are said to be *locally consistent*. Different propagation algorithms may be proposed for a same constraint, and these algorithms may achieve different levels of local consistency. Given two propagation algorithms $P_1$ and $P_2$ for a same constraint $c$, we say that $P_1$ is *stronger* than $P_2$ if, for every variable $x_i \in X(c)$,

we have $D_1(x_i) \subseteq D_2(x_i)$ where $D_1$ and $D_2$ denote the filtered domains obtained by propagating $c$ with $P_1$ and $P_2$, respectively, given the same initial domains.

In this section, we describe arc consistency, which is the most famous local consistency, and bound consistency, which is a weaker consistency.

**Definition 5.6** (Arc Consistency (AC))**.** Let be $(X, D, C)$ a CSP, $c \in C$ a constraint, and $x_i \in X$ a variable.

- A value $v_i \in D(x_i)$ is consistent with $c$ if there exists a valid tuple $\tau$ satisfying $c$ such that $\tau[x_i] = v_i$. Such a tuple is called a *support* for $(x_i, v_i)$ on $c$.
- The domain $D$ is arc consistent on $c$ for $x_i$ if all values in $D(x_i)$ are consistent with $c$.
- The CSP $(X, D, C)$ is arc consistent if $D$ is arc consistent for all variables in $X$ on all constraints in $C$.
- The CSP $(X, D, C)$ is arc inconsistent if $\emptyset$ is the only domain tighter than $D$ which is arc consistent for all variables on all constraints.

Historically, arc consistency is associated with binary CSPs and generalized arc consistency with non-binary CSPs while both definitions are perfectly the same.

**Example 5.7.** Let us consider the 4-queens problem introduced in Example 5.5, and let us consider the following domains: $D(x_1) = \{1, 2\}$, $D(x_2) = \{1, 3\}$ and $D(x_3) = D(x_4) = \{2, 4\}$. This network is not arc consistent because the value 1 for $x_2$ has no support on the constraint $c_{1,2}$ (because $\{(x_1, 1), (x_2, 1)\}$ violates the row constraint and $\{(x_1, 2), (x_2, 1)\}$ violates the downward diagonal constraint).

When the domain of a variable $x_i$ is not arc-consistent on a constraint $c$, we may ensure arc-consistency by removing inconsistent values from $D(x_i)$ and detect arc-inconsistency if the domain becomes empty. This domain filtering step is called *constraint propagation* and designing efficient propagation algorithms is a key point for solving CSPs.

Many propagation algorithms have been proposed for ensuring AC, and one of the most famous of these algorithms is called AC3 and has been introduced by Macworth [Mac77]. AC3 is displayed in Algorithm 2. Its main component is the function *Revise* which updates the domain of a variable $x_i$ with respect to a constraint $c$: For each value $v_i$ in the domain of $x_i$, it searches for a support for $(x_i, v_i)$ on $c$ (line 5) and, if no support is found, $v_i$ is removed from the domain of $x_i$ (line 6). *Revise* returns *true* if a domain has been changed, and *false* otherwise.

The main AC3 algorithm is a loop that iteratively calls *Revise* for variable/constraint couples which are stored in a queue $Q$. This queue is initialized with all possible variable/constraint couples (line 11). The main loop removes a couple $(x_i, c)$ from $Q$ (line 13) and calls *Revise* (line 14). If the domain of $x_i$ has been emptied, an inconsistency has been detected and AC3 returns false (line 14). Otherwise, if the domain has been changed, $Q$ is updated (line 15) by adding to it every couple $(x_j, c')$ such that $\{x_i, x_j\} \subseteq X(c')$ because the values removed from $D(x_i)$ may belong to a support of a value in $D(x_j)$ and in this case we need to check if there is another support. When $Q$ is empty, the algorithm returns true.

**Algorithm 2:** AC3 algorithm

```
1  Function Revise(x_i, c)
       Input: A variable x_i, and a constraint c
       Output: A Boolean value
       Postcondition : Remove from D(x_i) every value that has no support on c.
                       Return true if a value has been removed, false otherwise
2      begin
3          CHANGE ← false
4          for each v_i ∈ D(x_i) do
5              if ∄τ ∈ c ∩ π_{X(c)} with τ[x_i] = v_i then
6                  remove v_i from D(x_i)
7                  CHANGE ← true
8          return CHANGE
9  Function AC3(X, D, C)
       Input: A CSP (X, D, C)
       Output: A Boolean value
       Postcondition : Filter D to ensure AC. Return true if (X, D, C) is AC, and
                       false otherwise
10     begin
11         Q ← {(x_i, c) | c ∈ C, x_i ∈ X(c)}
12         while Q ≠ ∅ do
13             select and remove (x_i, c) from Q
14             if Revise(x_i, c) then
15                 if D(x_i) = ∅ then return false ;
16                 Q ← Q ∪ {(x_j, c') | c' ∈ C ∧ c' ≠ c ∧ x_i, x_j ∈ X(c') ∧ i ≠ j}
17         return true
```

AC3 achieves AC in $\mathcal{O}(er^3d^{r+1})$ time and $\mathcal{O}(er)$ space, where $r$ is the maximum arity of a constraint, $e$ is the number of constraints, and $d$ is the maximum number of values in a domain.

**Example 5.8.** Let consider the 4-queens problem introduced in Example 5.5, and let us assume that $D(x_1) = \{1\}, D(x_2) = \{3, 4\}, D(x_3) = \{2, 4\}, D(x_4) = \{2, 3\}$ (*i.e.*, the first queen is on the first row, and we have already removed from the domains of the other variables the values that are directly conflicting with it).

Initially, $Q$ contains: $\langle (x_2, c_{2,3}), (x_3, c_{2,3}), (x_2, c_{2,4}), (x_4, c_{2,4}), (x_3, c_{3,4}), (x_4, c_{3,4}) \rangle$.

- *Revise*$(x_2, c_{2,3})$ removes 3 from $D(x_2)$ and returns *true*.
  We have $D(x_2) = \{4\}$.
  $(x_3, c_{2,3})$ and $(x_4, c_{2,4})$ are added to $Q$ (but they were already in $Q$).
  We have $Q = \langle (x_3, c_{2,3}), (x_2, c_{2,4}), (x_4, c_{2,4}), (x_3, c_{3,4}), (x_4, c_{3,4}) \rangle$.

- *Revise*$(x_3, c_{2,3})$ removes 4 from $D(x_3)$ and returns *true*.
  We have $D(x_3) = \{2\}$.
  $(x_2, c_{2,3})$ and $(x_4, c_{3,4})$ are added to $Q$.
  We have $Q = \langle (x_2, c_{2,4}), (x_4, c_{2,4}), (x_3, c_{3,4}), (x_4, c_{3,4}), (x_2, c_{2,3}) \rangle$.

- *Revise*$(x_2, c_{2,4})$ does not remove values and returns *false*.
- *Revise*$(x_4, c_{2,4})$ removes 2 from $D(x_4)$ and returns *true*.
  We have $D(x_4) = \{3\}$.
  $(x_2, c_{2,4})$ and $(x_3, c_{3,4})$ are added to $Q$.
  We have $Q = \langle (x_3, c_{3,4}), (x_4, c_{3,4}), (x_2, c_{2,3}), (x_2, c_{2,4}) \rangle$.
- *Revise*$(x_3, c_{3,4})$ removes 2 from $D(x_3)$ and returns *true*.
  We have $D(x_3) = \emptyset$: An inconsistency has been detected and AC3 returns *false*.

AC3 is rather straightforward to implement. However, its time complexity is not optimal because *Revise* searches for supports from scratch, without memorizing them. Many improvements have been proposed since AC3 such as, for example, AC4 [Moh+86; Moh+88], AC6 [Bes94; Bes+08] or AC2001 [Zha+01]. We refer the reader to [Ros+06] for more details.

When the domain of a variable $x_i$ contains all integer values ranging between a lower bound (denoted $x_i.lb$) and an upper bound (denoted $x_i.ub$), the domain of $x_i$ is represented by the interval $[x_i.lb, x_i.ub]$. When all variables have interval domains, we may consider a local consistency which only operates on bounds. This bound consistency is weaker than AC, but it is also quicker to achieve.

**Definition 5.9** (Bound consistency)**.** Let $c$ be a constraint on the variables $X(c) = (x_1, \ldots, x_k)$ with respective interval domains $D(x_1), \ldots, D(x_k)$. $c$ is bound consistent if for every variable $x_i \in X(c), \forall v_i \in \{x_i.lb, x_i.ub\}, \forall j \neq i, \exists v_j \in [x_j.lb, x_j.ub], (v_1, \ldots, v_k) \in c$.

**Example 5.10.** Let consider the 4-queens problem introduced in Example 5.5, and let us assume that $D(x_1) = [1, 2]$, and $D(x_2) = D(x_3) = D(x_4) = [1, 4]$. The constraint $c_{1,2}$ is not bound consistent because there exists no support for $(x_2, 1)$. To ensure bound consistency of this constraint, we have to change the lower bound of $x_2$ to 3.

Ensuring bound consistency amounts to shrinking the domain intervals as much as possible without losing any solutions. Sometimes, it allows to apply filtering algorithms that have a lower complexity than algorithms that achieve AC. Indeed, ignoring holes from the domain variables usually leads to graphs with a specific structure that can be exploited to derive more efficient graph algorithms. Moreover, achieving bound-consistency may be used in a pre-processing step before applying a more expensive filtering that achieves AC.

For instance, many algorithms have been introduced to ensure the bound consistency of the constraint *alldifferent* [Lec96; Pug98; Meh+00; LO+03]. Regin gives an $\mathcal{O}(n^{2.5})$ algorithm to ensure domain consistency [Rég94] where $n$ is the number of variables while Puget [Pug98], Mehlhorn and Thiel [Meh+00] and Lopez-Ortiz [LO+03] give $\mathcal{O}(n.log(n))$ algorithms to ensure bound consistency.

## 5.3 Backtracking search algorithms

Constraint propagation filters domains by removing values that cannot belong to solutions. However, it only ensures a local consistency, and constraint propagation must be combined with a systematic exploration of the remaining search space to actually find

**Algorithm 3:** BT algorithm

```
1 Function BT((X, D, C), I)
      Input: A CSP N = (X, D, C) and a valid instantiation I on Y ⊆ X
      Output: A Boolean value
      Postcondition : Return true if there exists S ∈ sol(X, D, C) such that I = Y[S]
2     begin
3         if I is not locally consistent then return false;
4         if Y = X then return true;
5         Choose a variable x_i ∈ X \ Y
6         for each value v_i ∈ D(x_i) do
7             if BT(N, I ∪ {(x_i, v_i)}) then
8                 return true
9         return false
```

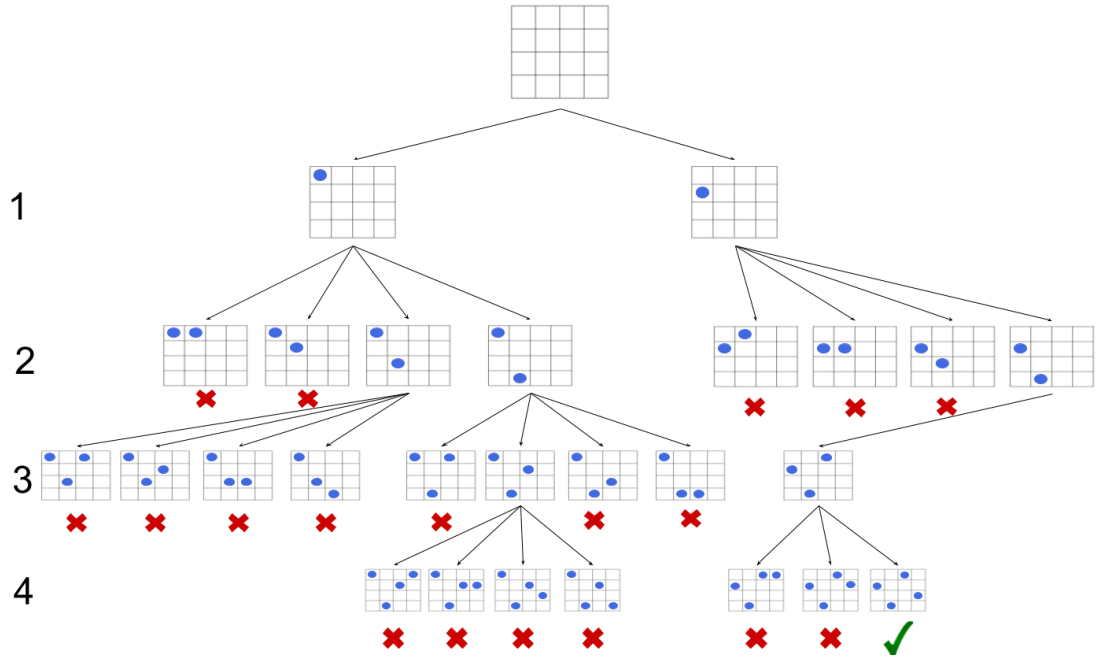
solutions (or prove inconsistency). This exploration is usually performed by a backtracking algorithm. In this section, we first introduce a naive backtracking algorithm, and then show how it may be combined with constraint propagation. Finally, we introduce ordering heuristics and branching strategies that may be used to customize the search.

**Naive backtracking algorithm (BT).** BT is described in Algorithm 3. It recursively extends a valid instantiation, starting from the empty instantiation, until either the current instantiation is not locally consistent (line 3) or the current instantiation is a solution (line 4). To extend a current instantiation $I$, an uninstantiated variable $x_i$ is chosen (line 5) and, for each value $v_i$ in its domain, BT is recursively called on $I \cup \{(x_i, v_i)\}$.

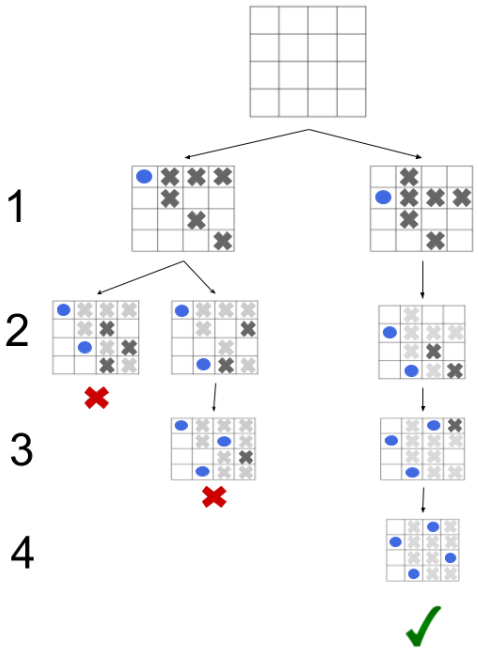
**Example 5.11.** The search tree explored by $BT$ for solving the 4-queens problem is displayed in Fig. 5.1a. This search tree has one node for each recursive call to $BT$, and an edge from node $(N, I_1)$ to node $(N, I_2)$ if $BT(N, I_1)$ has called $BT(N, I_2)$.

$BT$ is a naive algorithm because it does not propagate constraints to prune branches of the search tree: Only the constraints **with no uninstantiated variable** are checked. For instance, at level 2 (when $x_1$ and $x_2$ are assigned), only the constraints between variables $x_1$ and $x_2$ are checked, and $BT$ does not detect that the constraints between $x_3$ and $x_1$ and $x_2$ cannot be satisfied when $I = \{(x_1, 1), (x_2, 3)\}$.

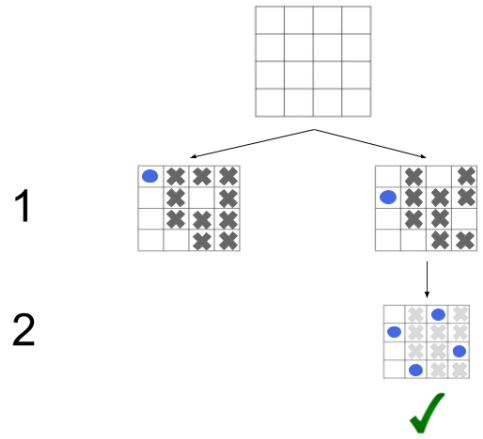
**Forward-Checking algorithm (FC).** FC refines BT by propagating constraints to filter domains at each recursive call. A trade-off has to be found between the cost of constraint propagation and its strength. FC reduces the cost of constraint propagation by maintaining AC only on constraints **with exactly one uninstantiated variable**. More precisely, line 3 of Algorithm 3 is replaced by a call to a procedure that filters domains as follows: For each uninstantiated variable $x_i \in X \setminus Y$ and each constraint $c \in C$ such that $X(c) \setminus Y = \{x_i\}$, it removes from $D(x_i)$ every value $v_i$ which is not AC on $c$ for $x_i$. If a domain becomes empty, *false* is returned and the search must backtrack.




(a) Search tree explored by BT.




(b) Search tree explored by FC.




(c) Search tree explored by MAC.

Figure 5.1 – Comparison of the search trees explored by BT, FC, and MAC for solving the 4-queens problem. Each search node is represented by the 4x4 chessboard corresponding to the current instantiation $I$.

**Example 5.12.** Fig. 5.1b displays the search tree explored by FC on the 4-queens problem. When $I = \{(x_1, 1)\}$, the filtering procedure propagates the constraint between $x_1$ and $x_2$ (resp. $x_3$, and $x_4$), and it removes 1 and 2 from $D(x_2)$ (resp. 1 and 3 from $D(x_3)$, and 1 and 4 from $D(x_4)$).

**Maintaining arc consistency algorithm (MAC).** Like FC, MAC propagates constraints to filter domains at each recursive call. However, it propagates more constraints than FC as it maintains AC on all constraints **with at least one uninstantiated variable**. More precisely, line 3 of Algorithm 3 is replaced by a call to a procedure (such as AC3, for example) that filters domains to ensure AC on all constraints with at least one uninstantiated variable. If a domain becomes empty, *false* is returned and the search must backtrack.

**Example 5.13.** Fig. 5.1c displays the search tree explored by MAC to solve the 4-queens problem. When $I = \{(x_1, 1)\}$, an algorithm such as AC3 detects an inconsistency (as illustrated in Section 5.2). When $I = \{(x_1, 2)\}$, all domains are reduced to singletons, and the corresponding instantiation is a solution. Hence, MAC only explores two branches.

**Ordering heuristics.** At each recursive call to Algorithm 3, an uninstantiated variable $x_i$ is chosen (line 5), and the rule used to choose this variable is called a *variable ordering heuristic*. Variable ordering heuristics have an impact on the size of the search tree. For example, the *minDom* heuristic [Gol+65] chooses a variable with the smallest domain, and this heuristic usually reduces the width of the search tree as the number of children of a node of the search tree is equal to the number of values in the domain of the chosen variable. Another classical heuristic is *maxDegree*, which chooses a variable involved in a maximum number of constraints, and this heuristic usually reduces the depth of the search tree as assigning a highly constrained variable usually allows to discover inconsistencies sooner.

*Value ordering heuristics* are used to define the order used to iterate on all values in the domain of the chosen variable (line 6). If the CSP is globally consistent, these ordering heuristics may allow to discover a solution quicker by first choosing the most promising values.

**Branching strategies.** In a search tree built by Algorithm 3, each node has one child per value in the domain of the selected variable $x_i$. Other branching strategies may be considered. For example, if the domain of $x_i$ is a closed interval of integer values $[lb, ub]$, we may create two branches: one where we reduce $D(x_i)$ to $[lb, v]$ and another one where we reduce $D(x_i)$ to $[v + 1, ub]$, where $v$ is a value that belongs to $[lb, ub]$ (typically, $v = \lfloor \frac{lb+ub}{2} \rfloor$). Another classical branching strategy is to create two branches: One where $x_i$ is assigned to a value $v \in D(x_i)$, and another one where $v$ is removed from $D(x_i)$.

## 5.4 Set Variables

Set variables are variables that are instantiated to sets of integer values. In most cases, it is not possible to define the domain of a set variable by enumerating all possible sets, because there exists an exponential number of subsets with respect to the number of elements: if a set variable may be assigned with any subset of $[1, k]$, it is not reasonable to enumerate the $2^k$ subsets when defining its domain. Hence, set domains are usually approximated by set intervals, as proposed by Gervet in [Ger95].

**Definition 5.14.** Let $glb$ and $lub$ be two sets. The set interval $[glb, lub]$ contains every set $S$ such that $glb \subseteq S \subseteq lub$.

In other words, $glb$ (also called the *kernel*) is the greatest lower bound and it contains all mandatory elements; $lub$ (also called the *envelope*) is the least upper bound and it contains all possible elements.

**Example 5.15.** The set interval $[\{1,4\}, \{1,2,4,5\}]$ contains the sets $\{1,4\}$, $\{1,2,4\}$, $\{1,4,5\}$, and $\{1,2,4,5\}$.

The domain of a set variable may be approximated by a set interval by computing its convex closure.

**Definition 5.16.** Let be $E$ a set of elements, and $S$ a set of subsets of $E$ (*i.e.*, $S \subseteq \mathcal{P}(E)$). The convex closure of $S$ is the set interval:

$$[\bigcap_{s_i \in S} s_i, \bigcup_{s_i \in S} s_i].$$

**Example 5.17.** The convex closure of $\{\{1,2,3\}, \{3\}, \{3,6\}\{3,4,5\}\}$ is the set interval $[\{3\}, \{1,2,3,4,5,6\}]$.

This convex representation of set domains allows to efficiently propagate set constraints by ensuring bound consistency (with a straightforward extension of Def. 5.9 to set variables).

**Example 5.18.** Enforcing bound consistency of the constraint $S = S_1 \cap S_2$ amounts to adding to $S.glb$ every value in $S_1.glb \cap S_2.glb$ and removing from $S.lub$ every value which does not belong to $S_1.lub \cap S_2.lub$.

**Branching Strategies for Set Variables.** When solving a CSP that has set variables, the branching strategy needs to be adapted: As set domains may contain a huge number of values, we usually do not create a branch for each possible value in set variable domains. A classical branching strategy is to choose a set variable $s$ such that $s.glb \subset s.lub$, to select an element $e \in s.lub \setminus s.glb$ and to create two branches: The first branch corresponds to the case where $e \in s$ and it is obtained by adding $e$ to $s.lub$; The second branch corresponds to the case where $e \notin s$ and it is obtained by removing $e$ from $s.glb$.

## 5.5 Global Constraints

Global constraints are a key point of the success of CP: They both ease the modeling step, by providing compact ways for declaring constraints, and speed-up the solution process, by providing dedicated propagators. In this section, we first recall some properties of global constraints, and then we describe the global constraints which are used in this thesis.

If global constraints are widely used in CP, defining what is a global constraint has been subject of many discussions. In the Global Constraint Catalog [Bel+05], a global constraint is defined as an *expressive and concise condition involving a non-fixed*

*number of variables*. In [Bes+03], Bessière and Van Hentenryck have introduced three properties that may be used to characterize constraint globality. These properties are defined with respect to constraint decompositions: the *decomposition* of a constraint $c$ is a CSP $P$ such that the constraints of $P$ have lower arities than $c$, and $sol(P) = sol(c)$.

Bessière and Van Hentenryck propose three levels of globality for a constraint.

**Definition 5.19.** A constraint is

- *semantically global* if it is not possible to decompose it;
- *operationally global* with respect to a filtering strength, if it is not possible to achieve the same strength of filtering when considering a decomposition of it;
- *algorithmically global* with respect to a filtering strength, if it is not possible to achieve the same strength of filtering with the same complexity when considering a decomposition of it.

Semantic globality implies both operational and algorithmic globality. Operational globality is important because it implies a stronger filtering, that removes more values, though it may be possible that this stronger filtering does not pay off if the complexity of achieving it is too high. Algorithmic globality does not necessarily implies a stronger filtering, but it implies a more efficient filtering (with respect to time or memory consumption).

**Example 5.20.** The global constraint $alldifferent(x_1, \ldots, x_n)$ is satisfied if all $x_i$ variables are assigned to different values. It has been introduced in [Rég94] and it is widely used as many problems involve finding injections.

Let $P = (X, D, C)$ be the CSP such that $X = (x_1, \ldots, x_n)$ and $C = \{c_{ij} : \{x_i, x_j\} \subseteq X\}$, where $c_{ij} \equiv x_i \neq x_j$. $P$ is a decomposition of the constraint $alldifferent(x_1, \ldots, x_n)$. Hence, *alldifferent* is not semantically global.

However, *alldifferent* is operationally global as there exists no decomposition of it that preserves AC [Bes+09a].

Let us now describe the global constraints that are used in this thesis. We assume that arrays of $n$ elements are indexed from 1 to $n$, and we denote $A[i]$ the $i^{th}$ element of an array $A$.

**NValue.** The constraint *NValue* has been proposed by Pachet and Roy to model a combinatorial problem involved in selecting musical playlists [Pac+99]. It ensures that the number of different values taken by a collection of integer variables is equal to a given integer variable.

More formally, given an array $X$ of $m$ integer variables, and an integer variable $N$, the constraint $NValue(X, N)$ ensures:

$$N = \#\{X[i] | i \in [1, m]\}.$$

Enforcing domain consistency on *NValue* is NP-hard [Bes+04b], even when the domain of $N$ is reduced to a singleton [Bes+07].

*NValue* can be decomposed with the constraints *atMostNValue* and *atLeastNValue*, that respectively constrain $X$ to take at most $N$ different values and at least $N$ different

values. Enforcing AC on *atLeastNValue* may be done in polynomial time whereas enforcing AC on *atMostNValue* is an NP-hard problem [Bes+05]. Therefore, several propagation algorithms that ensure lower consistencies have been proposed [Bel01; Bes+05]. Different decompositions of *NValue* are studied in [Bes+09b].

**Global Cardinality Constraint.** The Global Cardinality Constraint *gcc* is a generalization of *alldifferent*: Given a set of variables $X$, *allDifferent* ensures that each possible value $v$ is taken by at most one variable of $X$, whereas *gcc* ensures that the number of times a value $v$ is taken is equal to an integer variable.

More formally, given an array $X$ of $n$ integer variables, an array $val$ of $k$ integer values, and an array $Occ$ of $k$ integer variables, the constraint $gcc(X, val, Occ)$ ensures:

$$\forall i \in [1, k], Occ[i] = \#\{j \in [1, n] | X[j] = val[i]\}.$$

Enforcing AC on this *gcc* global constraint is $\mathcal{NP}$-complete [Qui+04] when domains are not closed intervals. In [Rég96], Régin introduces an algorithm for enforcing BC in $\mathcal{O}(n^2.d)$ (where $n$ is the number of variables and $d$ the number of values) while [Qui+04] improves this work with an $\mathcal{O}(n^{1.5}.d)$ algorithm.

**Element.** The global constraint *element* was introduced by Van Hentenryck and Carillon [Hen+88]. This constraint is very important to implement variable indices and to relate two sets of variables.

Given two integer variables $I$ and $V$ and an array of integer variables $T$, the constraint $element(I, T, V)$ ensures:

$$V = T[I].$$

CP solvers usually implement dedicated filtering algorithms for *element*.

**Precede.** The global constraint *precede* has been introduced in [Law+04]. It is often used to break symmetries of indistinguishable values.

More formally, given an array $X$ of $n$ integer variables and an array $v$ of $m$ integer values, the constraint $precede(X, v)$ ensures:

$$\forall k \in [1, m-1], \forall l \in [k+1, m], (\exists j \in [2, n], X[j] = v[l]) \Rightarrow (\exists i \in [1, j-1], X[i] = v[k]).$$

A filtering algorithm was proposed in [Law+04], it enforces GAC with a time complexity linear to the length of $X$.

**Partition.** The global constraint *partition* is described in [Bes+04a] and it is used to constrain a collection of set variables to be a partition of a given set variable.

More formally, given an array $S$ of $n$ set variables and a set variable $U$, the constraint $partition(S, P)$ ensures:

$$U = \bigcup_{i \in [1,n]} S[i] \wedge \forall \{i, j\} \subseteq [1, n], S[i] \cap S[j] = \emptyset$$

The *partition* constraint is decomposable into binary empty intersection constraints and ternary union constraints involving $n$ additional variables without hindering bound consistency [Wal03]. Therefore, bound consistency of *partition* may be enforced in polynomial time. In Choco solver [Pru+16], this constraint is decomposed into two global constraints:

- a *union* constraint that ensures $U = \bigcup_{i \in [1,n]} S[i]$.
- an *allDisjoint* constraint that ensures that $\forall \{i, j\} \subseteq [1, n], S[i] \cap S[j] = \emptyset$.

**AtLeast.** Given an array $X$ of $m$ integer variables, an integer variable $N$, and a value $v$, the constraint $atLeast(N, X, v)$ ensures:

$$N \leq \#\{i \in [1, m] | X[i] = v\}.$$

## 5.6 Solving Optimization Problems with CP

Many real world problems (such as scheduling, sequencing, or planning, for example) involve not only satisfying some constraints but also optimizing some objective functions. In this section, we first show how to solve these problems with CP when there is only one objective function to optimize. Then, we study the case where there are several objective functions to optimize.

### 5.6.1 Mono-criterion optimization

A Constrained Optimization Problem (COP) is defined by a quadruple $(X, D, C, f)$ such that $(X, D, C)$ is a CSP, and $f : X \to \mathbb{R}$ is a function which maps every instantiation of $X$ to a numerical value. Solving a COP amounts to finding a solution of $(X, D, C)$ that maximizes $f$. Without loss of generality, we only consider maximization problems.

Backtracking search algorithms introduced in Section 5.3 can be extended to solve COPs in a rather straightforward way, as proposed by Van Hentenryck in [VH89]. The idea is to solve a sequence of satisfaction problems. More precisely, before starting the search, we add to $X$ a new variable $obj$, and we add to $C$ a new constraint $obj = f(X)$. Then, we perform a backtracking search procedure (such as MAC, for example). However, when a solution is found (line 4 of Algorithm 3), we add the constraint $obj > v$ where $v$ is the value assigned to $obj$ in the solution, and we go on the search. This process is repeated until no more solution can be found in which case the last solution found has been proven optimal.

**Example 5.21.** Let us consider the COP $(X, D, C, f)$ such that $X = (x, y, z)$, $D(x) = D(y) = D(z) = \{1, 2, 3\}$, $C = (NValue([x, y, z], 2))$, and $f(x, y, z) = x + y + z$. We introduce the integer variable $obj$ with $D(obj) = [3, 9]$ and add the constraint $obj = x + y + z$ to $C$.

In Fig. 5.2, we show the search tree explored when using MAC with a lexicographical ordering heuristic for selecting variables and values. At each node, we display the domain of each variable after enforcing AC if it is not instantiated, otherwise, we give its value. All nodes are numbered according to the order they are explored. Solutions are successively found at nodes $n3$, $n4$, $n6$, and $n7$, and we successively add the constraints $obj > 4$, $obj > 5$, $obj > 7$, and $obj > 8$. Finally, at node $n8$, enforcing AC on constraint $c_2$ removes 2 from the domains of $y$ and $z$ while enforcing AC on the constraint $c_1$ removes 3 from the domain of $y$ that becomes empty. The search backtracks, and as there is no more branches to explore the search stops. The optimal solution is the last solution found, *i.e.*, $\{(x, 2), (y, 3), (z, 3)\}$.

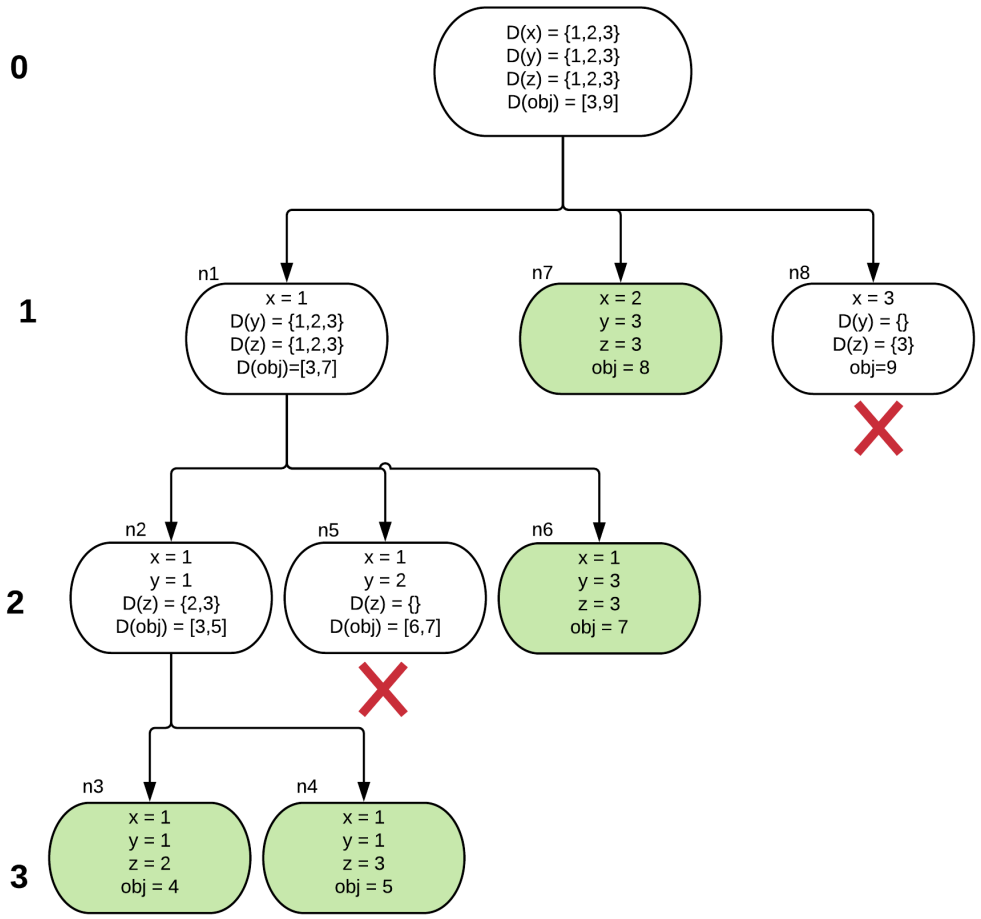



Figure 5.2 – Search tree for a single criterion optimization problem when using lexicographical order as value and variable ordering heuristics. For each node, we give the domain of the variable if it is not instantiated, otherwise, we give its value. Green nodes correspond to solutions.

Ordering heuristics usually have a great impact when solving COPs. In particular, the size of the search tree depends on the quality of the first solution found: The higher the value $v$ of the objective function, the more branches are pruned by the constraint $obj > v$. Hence, ordering heuristics usually aim at favoring solutions with high objective function values [Fag+17]. Note however that a large part of the search effort is spent for proving the optimality of the last solution found and heuristics that favor good solutions first no longer help in this case.

**Example 5.22.** In our previous example, if we use the *maxdom* value ordering heuristic, that first branches on the maximum value in the domain of the variable, the first solution found is $\{(x_1, 3), (x_2, 3), (x_3, 2)\}$, which is optimal, and the search tree is composed of a single branch (because the proof of optimality is achieved by AC which detects an inconsistency when adding the constraint $obj > 8$).

### 5.6.2 Multi-criteria Optimization

In many real-life problems, there are several criteria to optimize. When these different criteria can be aggregated into a single function (by considering a linear combination of them, for example), the problem can be reduced to a classical COP. However, in some cases it is not possible to define a suitable aggregation function. In this case, we may search for a set of non-dominated solutions that correspond to different compromises.

More formally, let us define multi-objective optimization problems and the dominance relation that is used to compare solutions of these problems.

**Definition 5.23** (Multi-Objective Combinatorial Optimization (MOCO))**.** A MOCO problem is a quadruple $(X, D, C, F)$ where $(X, D, C)$ is a CSP and $F = \{f_1, \ldots, f_m\}$ is a set of objective functions.

Without loss of generality, we assume that each objective function $f_i \in F$ is of the form $f_i(X) = obj_i$, where $obj_i \in X$ is a numeric variable: If this is not the case, we can easily add a new variable $obj_i$ to $X$ and constrain this variable to be equal to $f_i(X)$. Also, we assume that each objective function must be maximized.

**Definition 5.24** (Domination)**.** Let be $(X, D, C, F)$ a MOCO problem, and $I$ and $I'$ two solutions of the CSP $(X, D, C)$. $I$ dominates $I'$, denoted $I \succ I'$, if:

$$\forall obj_i \in F, I[obj_i] \geq I'[obj_i] \wedge \exists obj_i \in F, I[obj_i] > I'[obj_i]$$

Non-dominated solutions are said to be *Pareto optimal*, and the set of all Pareto-optimal solutions is called the *Pareto front* [Par96].

**Example 5.25.** Fig. 5.3 shows the set of solutions of a MOCO problem with two objective functions. Examples of dominance relations are: $e_9 \succ e_6$, $e_7 \succ e_2$, and $e_1 \succ e_0$. Solutions $e_0$, $e_3$ and $e_{10}$ do not dominate any solution whereas $e_{11}, e_{12}, e_{13}$ and $e_{14}$ are non-dominated solutions and constitute the Pareto front.

A first possibility to compute the Pareto front is to solve a sequence of COPs, as proposed by van Wassenhove and Gelders in [Was+80] and by Duong in [Duo14]. This approach is described by Algorithm 4 for the case where there are two objective functions $f_1(X) = obj_1$ and $f_2(X) = obj_2$. Algorithm 4 first searches for a solution

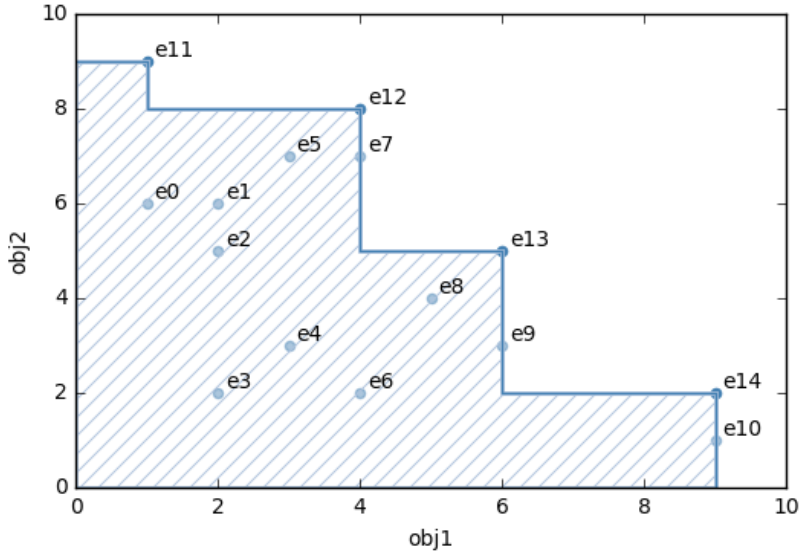



Figure 5.3 – Pareto front of a MOCO problem with two objectives $obj_1$ and $obj_2$: Each point $(x, y)$ corresponds to a solution $e_i$ such that $x = e_i[obj_1]$ and $y = e_i[obj_2]$.

**Algorithm 4:** Computation of Pareto fronts by solving a sequence of COPs

**Input:** A MOCO $(X, D, C, F)$ such that $F = \{obj_1, obj_2\}$
**Output:** The Pareto front of $(X, D, C, F)$

```
1 begin
2     P ← ∅
3     while true do
4         I1 ← solution of (X, D, C) that maximizes obj1
5         if I1 = null then return P;
6         I2 ← solution of (X, D, C ∪ {obj1 = I1[obj1]}) that maximizes obj2
7         Add I2 to P
8         Add the constraint obj2 > I2[obj2] to C
9         Add the constraint obj1 < I1[obj1] to C
```

$I_1$ that maximizes $obj_1$ (line 4) and then searches for a solution $I_2$ that maximizes $obj_2$ when constraining $obj_1$ to be equal to $I_1[obj_1]$ (line 6). $I_2$ is the solution of the Pareto front which has the largest possible value for $obj_1$. To search for other non-dominated solutions, we constrain $obj_2$ to be strictly greater than $I_2[obj_2]$ (line 8). We repeat these operations: At each iteration, we compute a new non-dominated solution by first optimizing $obj_1$ (line 4), and then optimizing $obj_2$ while fixing the value of $obj_1$ (line 6), and we add a constraint on the lower bound of $obj_2$ (line 8). Because of these successively added constraints, the solutions that are successively added to the Pareto front have increasing values for $obj_2$ and decreasing values for $obj_1$. We stop iterating when $obj_2$ cannot be improved anymore, *i.e.*, the CSP $(X, D, C)$ has no solution because the last solution added to the Pareto front has the largest possible value for $obj_2$.

**Example 5.26.** Let us consider the solutions displayed in Fig. 5.3.

**Algorithm 5:** Computation of Pareto fronts by enumerating all solutions and dynamically adding constraints

**Input:** A MOCO $(X, D, C, F)$
**Output:** The Pareto front of $(X, D, C, F)$
1 **begin**
2  $\mathcal{P} \leftarrow \emptyset$
3  **while** *true* **do**
4  Search for the next solution $I$ of $(X, D, C)$
5  **if** $I = null$ **then return** $\mathcal{P}$;
6  Add $I$ to $\mathcal{P}$, and remove from $\mathcal{P}$ every solution dominated by $I$
7  Add the constraint $\bigvee_{obj_i \in F} obj_i > I[obj_i]$ to $C$

- At iteration 1 of Algorithm 4, we first search for a solution that maximizes $obj_1$, *i.e.*, $I_1 \in \{e_{10}, e_{14}\}$. Then we search for a solution that maximizes $obj_2$ when $obj_1 = 9$, *i.e.*, $I_2 = e_{14}$. We add $e_{14}$ to the Pareto front, and we add the constraint $obj_2 > 2$.
- At iteration 2, we search for a solution that maximizes $obj_1$, *i.e.*, $I_1 \in \{e_9, e_{13}\}$. Then we search for a solution that maximizes $obj_2$ when $obj_1 = 6$, *i.e.*, $I_2 = e_{13}$. We add $e_{13}$ to the Pareto front, and we add the constraint $obj_2 > 5$.
- At iteration 3, we search for a solution that maximizes $obj_1$, *i.e.*, $I_1 \in \{e_7, e_{12}\}$. Then we search for a solution that maximizes $obj_2$ when $obj_1 = 4$, *i.e.*, $I_2 = e_{12}$. We add $e_{12}$ to the Pareto front, and we add the constraint $obj_2 > 8$.
- At iteration 4, we search for a solution that maximizes $obj_1$, *i.e.*, $I_1 = e_{11}$. Then we search for a solution that maximizes $obj_2$ when $obj_1 = 6$, *i.e.*, $I_2 = e_{11}$. We add $e_{11}$ to the Pareto front, and we add the constraint $obj_2 > 9$.
- At iteration 5, we search for a solution that maximizes $obj_1$, but there is no solution that satisfies the constraint $obj_2 > 9$, and we stop iterating.

Algorithm 4 solves a sequence of COPs and, for each of these COPs, a new tree search is performed. In [Gav02], Gavanelli proposes an alternative approach where a single tree search is performed for computing the whole Pareto front, as described in Algorithm 5: Each time a new solution $I$ is found (line 4), the Pareto front is updated by adding $I$ and removing all solutions dominated by $I$ (line 6), and a constraint is dynamically added in order to prevent the search from computing a solution which is dominated by $I$ (line 7). The search stops when no more solution can be found. A *Pareto* constraint based on this filtering rule has been introduced in [Sch+13] with an efficient filtering algorithm for bi-objective MOCOs.

**Example 5.27.** Let us consider the solutions displayed in Fig. 5.3, and let us suppose that the first solution found is $e_8$. In this case, the constraint $obj_1 > 5 \vee obj_2 > 4$ is dynamically added, and the points $e_3$, $e_4$, and $e_6$ are no longer solutions as they are dominated by $e_8$. Let us then suppose that the second solution found is $e_{13}$. As $e_{13}$ dominates $e_8$, $e_8$ is removed from the Pareto front. The constraint $obj_1 > 6 \vee obj_2 > 4$ is dynamically added, and the points $e_9$ and $e_2$ are no longer solutions.

The two approaches described in Algorithms 4 and 5 have complementary weaknesses and strengths. Algorithm 4 solves a sequence of COPs, and restarts a new tree search for each of these COPs. If the propagation of the constraint on the lower bound of $obj_2$ does not filter enough domains, it may be possible that some parts of the search tree are explored more than once. For example, once Algorithm 4 has found solution $e_{14}$, and added constraint $obj_2 > 2$, instantiations $e_3$, $e_6$, $e_{14}$ and $e_{10}$ are no longer consistent. However, it may be possible that the branches that lead to these instantiations are not pruned at the root of the search tree, leading to some redundancy.

Algorithm 5 performs a single search, so that this kind of redundancy is avoided. However, the search aims at enumerating all solutions (while dynamically adding constraints to prune branches that lead to dominated solutions). The efficiency of this process highly depends on the quality of the solutions that are found at the beginning of the search. On our example, if the first solution found is $e_3$, then the constraint added to prune dominated solutions ($obj_1 > 2 \vee obj_2 > 2$) does not remove any solution, whereas if the first solution found is $e_{13}$, then the constraint added to prune dominated solutions ($obj_1 > 6 \vee obj_2 > 5$) removes solutions $e_2$, $e_3$, $e_4$, $e_6$, $e_8$, and $e_9$.

## 5.7 Constraint Programming Libraries

Backtracking search algorithms described in the previous sections have been embedded in various CP libraries such as $ECL^iPS^e$ [Wal+97], GNU Prolog [Dia+00], Picat [Zho+13], Gecode [tea05], or Choco [Pru+16], for example. To solve a CSP with these libraries, the user mainly has to implement a model of the problem by using predefined procedures for declaring the variables and the constraints of the problem. Then, the model may be solved by using predefined search procedures. This approach is well summarized by Freuder in [Fre97]:

> *Constraint Programming represents one of the closest approaches computer science has yet made to the Holy Grail of programming: The user states the problem, the computer solves it.*

However, CSPs are $\mathcal{NP}$-complete problems in general, and generic search procedures may struggle to solve them. Hence, the user may customize them by specifying which ordering heuristics or which branching strategies should be used, for example.

Most of these libraries are open source software, and are designed to be easily extended. In particular, users can easily define new constraints. Actually, the "branch and propagate" approach makes it easy to add new constraints as each constraint is basically a set of propagators which filter the domains of its variables.

In this Section, we very briefly describe the main challenges for implementing a new constraint, with a specific focus on Choco 4 which has been used for our implementations. We refer the reader to [Ros+06] for more details on the implementation of CP libraries in general, and to [Pru+16] for more details on Choco 4.

Choco 4 is an Open Source library developed in Java 8, and it uses object oriented mechanisms and design patterns to ease the implementation of new features. In Choco 4, a constraint is an instance of the class `Constraint`, and it is a container which is composed of propagators: Each propagator is in charge of filtering the domains of some of its variables. Many predefined constraints are already implemented in Choco and to create one of these predefined constraints we simply call constraint factory

methods that are implemented in the class `Model`: Each constraint factory method returns an instance of `Constraint` corresponding to a different kind of constraint with its corresponding list of propagators.

To create a constraint that is not yet implemented in Choco, we use the constructor of the class `Constraint` which takes as arguments a list of propagators. Hence, implementing a new constraint simply amounts to implementing its propagators. Each propagator is an instance of a class that must extend the abstract class `Propagator`. To implement such a class, we mainly have to implement three methods:

- The constructor of the class, which usually creates the data structures needed for propagating the constraint;
- The method `isEntailed`, which returns `true` if the constraint is satisfied, `false` if it cannot be satisfied, and `undefined` if both cases are still possible;
- The method `propagate`, which filters variable domains.

Optionally, we may also specify the events that trigger a call to `propagate` by redefining the method `getPropagationConditions`.

A major issue when implementing a propagator is to restore the previous states of the data structures used by the propagator when backtracking. There exist three different ways to do this:

- Copying, *i.e.*, a copy of the complete state is created before changing it and, when backtracking, this copy is used to restore the state;
- Trailing, *i.e.*, operations that change the state are recorded and, when backtracking, inverse operations are performed (in reverse order) to restore the state;
- Recomputation, *i.e.*, the state is recomputed from scratch after each backtrack.

Trailing is the predominating approach used for state restoration in finite domain constraint programming system.

## 5.8 Discussion

CP allows a user to define a problem in a declarative way, by means of variables and constraints, and then to solve it by using generic search procedures. However, if it is usually very easy to model a CSP, a COP, or a MOCO with a CP language, it may happen that the generic search procedures are not able to solve these problems within a reasonable amount of time. In this case, the user may improve the model, by adding redundant constraints (*i.e.*, constraints that do not change the set of solutions, but the propagation of which reduces the search space), by introducing new constraints for breaking symmetries, by replacing some constraints with existing global constraints, or by defining new ordering heuristics or new branching strategies, for example. To improve the search process, the user may also introduce new global constraints, and dedicated propagation algorithms for this constraint, in order to filter more values and/or reduce the complexity of the propagation.

There exist two other well known declarative approaches for solving problems: Integer Linear Programming (ILP) and the satisfiability of Boolean formulas (SAT). Both

ILP and SAT provide a declarative way for modeling a problem by means of variables and constraints: numeric variables and linear (in)equalities for ILP and boolean variables and clauses for SAT. From a modeling point of view, CP may be viewed as a generalization of both SAT and ILP as the constraints used in ILP and SAT may be used in a CP model as well. However, CP, SAT and ILP use different generic procedures for searching for solutions.

- ILP solvers are usually based on *Branch and Bound* approaches: At each node of the search tree, a bound on the objective function is computed by solving a relaxation of the problem (typically, integrality constraints are relaxed to obtain a linear problem that may be solved in polynomial time). This approach may also be combined with *Branch and Cut* (where linear constraints are added to tighten relaxations and compute better bounds), or *Branch and Price* (where the number of active variables is reduced, and columns corresponding to omitted variables are generated during the solution process). The reader may refer to [Nem+88] for more details.

- SAT solvers are usually based on an exhaustive backtracking search called DPLL, which performs unit propagation at each node of the search tree, usually combined with *Conflict-driven clause learning* (CDCL) to allow the search to backjump to the first node involved in a conflict. The reader may refer to [Bie+09] for more details.

In the next two chapters, we define the conceptual clustering problem and the exact cover problem, and we describe existing CP, ILP, and SAT models for solving these problems. However, none of these models is able to solve all instances related to our ERP configuration problem within a reasonable amount of time. This motivated us for introducing new CP models in Chapter 9 and new global constraints in Chapters 10 and 11 dedicated to these problems.

# Chapter 6

# Conceptual clustering

## Contents



In this thesis, we want to extract relevant parts of configurations from a database of configurations imported from previous implementations of *Copilote*. Therefore, we focus on knowledge discovery in databases (KDD) approaches: These approaches aim at extracting reusable interpretable knowledge from large databases [NAP05]. A KDD process has three main steps: the selection and preparation of data, the data mining operation and finally the interpretation of the extracted knowledge. In this chapter, we focus on the data mining step that aims at extracting knowledge from the prepared data. Methods used during this step can be divided into two categories, *i.e.*, symbolic and numerical methods, depending on the type of data attributes. As most *Copilote* parameters have symbolic values, we focus on symbolic methods and, more particularly, on *Formal Concept Analysis* (FCA) which groups together objects sharing a same set of attribute values [Gan+97].

We present our motivations for using conceptual clustering in our applicative context in Section 6.1. In Section 6.2, we introduce definitions related to FCA and we present some measures used to characterize formal concepts. In Section 6.3, we define the problem of conceptual clustering, where each cluster corresponds to a formal concept, and we introduce classical criteria used to evaluate clusterings. In Section 6.4, we describe two existing CP approaches and one ILP-based approach to solve conceptual clustering problems.

## 6.1 Motivations

As explained in Chapter 4, we aim at extracting relevant parts of configurations that correspond to business logic requirements. To do this, we propose to use conceptual

| | Price reference date | Min order blocking | Order split | Stock control |
|---|---|---|---|---|
| $C_1$ | Delivery date | Yes | No | Blocking |
| $C_2$ | Delivery date | No | No | Alert |
| $C_3$ | Order date | Yes | No | Without |
| $C_4$ | Order date | Yes | Yes | Alert |

Table 6.1 – Example of configurations of the sale module of *Copilote*: Each line corresponds to a configuration and describes the setting of four parameters for this module.

Table 6.2 – Transactional database generated from the configurations displayed in Table 6.1: Lines correspond to transactions, columns to items, and there is a 1 (resp. 0) at line $t$/column $i$ if $(t, i) \in \mathcal{R}$ (resp. $(t, i) \notin \mathcal{R}$).

| | $i_1$ | $i_2$ | $i_3$ | $i_4$ | $i_5$ | $i_6$ | $i_7$ | $i_8$ | $i_9$ |
|---|---|---|---|---|---|---|---|---|---|
| $t_1$ | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| $t_2$ | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| $t_3$ | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 |
| $t_4$ | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 |

$i_1 \equiv$ (Price reference date, Delivery date)
$i_2 \equiv$ (Price reference date, Order date)
$i_3 \equiv$ (Order blocking, Yes)
$i_4 \equiv$ (Order blocking, No)
$i_5 \equiv$ (Order split, Yes)
$i_6 \equiv$ (Order split, No)
$i_7 \equiv$ (Stock control, Blocking)
$i_8 \equiv$ (Stock control, Alert)
$i_9 \equiv$ (Stock control, Without)

clustering. Indeed, we do not have any information about the business requirements previously configured in the system: We do not know how many business requirements exist, neither do we know how many times they were configured. We only assume that each configuration fulfills one business logic configuration. Conceptual clustering is well-suited to this context because it allows us, in an unsupervised way, to group together configurations that implement the same requirement and identify the part of configuration they share.

**From configurations to a transactional database.** In this thesis, we use the transactional database terminology: Objects are called *transactions*, and attribute values are called *items*. More formally, we assume that:

- $\mathcal{T}$ is a set of $m$ transactions (numbered from 1 to $m$);
- $\mathcal{I}$ is a set of $n$ items (numbered from 1 to $n$);
- $\mathcal{R} \subseteq \mathcal{T} \times \mathcal{I}$ is a binary relation that relates transactions to items, *i.e.*, $(t, i) \in \mathcal{R}$ denotes the fact that transaction $t$ has item $i$.

We denote $itemset(t)$ the set of items associated with a transaction $t$, *i.e.*, $\forall t \in \mathcal{T}, itemset(t) = \{i \in \mathcal{I} | (t, i) \in \mathcal{R}\}$.

To generate a transactional database $(\mathcal{T}, \mathcal{I}, \mathcal{R})$ from existing *Copilote* configurations, we proceed as follows: $\mathcal{T}$ contains a transaction for each existing configuration; $\mathcal{I}$ contains an item for each possible value of each parameter; $\mathcal{R}$ contains a couple

| | $i_1$ | $i_2$ | $i_3$ | $i_4$ |
|---|---|---|---|---|
| $t_1$ | 1 | 1 | 0 | 1 |
| $t_2$ | 1 | 0 | 1 | 1 |
| $t_3$ | 0 | 1 | 0 | 1 |
| $t_4$ | 0 | 1 | 1 | 0 |
| $t_5$ | 1 | 0 | 1 | 0 |

Table 6.3 – Example of transactional dataset with $m = 5$ transactions and $n = 4$ items.

$(t, i) \in \mathcal{T} \times \mathcal{I}$ if and only if the configuration associated with $t$ contains the parameter setting associated with $i$.

**Example 6.1.** Table 6.1 displays the setting of four parameters in four configurations of *Copilote*. Two of these parameters are symbolic ones: *Price reference date* may be assigned to *Delivery date* or *Order date*; and *Stock control* may be assigned to *Blocking*, *Alert*, or *Without*. The two other parameters are Boolean ones.

From these parameter settings, we create nine items: Two for *Price reference date*, *Min order blocking*, and *Order split*, and three for *Stock control*. The corresponding transactional database is displayed in Table 6.2.

## 6.2 Formal Concepts

Formal concepts are groups of transactions that share a same set of items. They are defined by means of intents and extents which are defined below.

**Definition 6.2** (intent)**.** The *intent* of a subset $T \subseteq \mathcal{T}$ of transactions is the intersection of their itemsets, *i.e.*, $intent(T) = \bigcap_{t \in T} itemset(t)$.

**Definition 6.3** (extent)**.** The *extent* of a set $I \subseteq \mathcal{I}$ of items is the set of transactions whose itemsets contain $I$, *i.e.*, $extent(I) = \{t \in \mathcal{T} : I \subseteq itemset(t)\}$.

**Example 6.4.** Table 6.3 gives an example of transactional dataset. We have:

- $intent(\{t_1, t_5\}) = \{i_1\}$ because $i_1$ is the only item contained in both $itemset(t_1)$ and $itemset(t_5)$
- $extent(\{i_2, i_3\}) = \{t_4\}$ because $t_4$ is the only transaction whose itemset contains both $i_2$ and $i_3$.

$\mathcal{P}(\mathcal{T})$ and $\mathcal{P}(\mathcal{I})$ are partially ordered sets (posets) when considering the set inclusion order relation $\subseteq$, and *intent* and *extent* operators induce a Galois connection between these two posets, *i.e.*, $\forall T \subseteq \mathcal{T}, \forall I \subseteq \mathcal{I}, T \subseteq extent(I) \Leftrightarrow I \subseteq intent(T)$.

**Definition 6.5.** A *formal concept* is a couple $(T, I) \in \mathcal{P}(\mathcal{T}) \times \mathcal{P}(\mathcal{I})$ such that $T = extent(I)$ and $I = intent(T)$. We denote $\mathcal{F}$ the set of all formal concepts.

**Example 6.6.** In Table 6.4, we give all the formal concepts we can extract from the dataset displayed in Table 6.3. $c_5 = (\{i_1, i_3\}, \{t_2, t_5\})$ is a formal concept because $t_2$, and $t_5$ are the only transactions that contain both $i_1$ and $i_3$, and, $i_1$ and $i_3$ are the only items contained in both $t_2$ and $t_5$.

| F | *intent* | *extent* | *frequency* | *size* | *diameter* | *split* |
|---|---|---|---|---|---|---|
| $c_0$ | $\emptyset$ | $\{t_1, t_2, t_3, t_4, t_5\}$ | 5 | 0 | 1 | 0 |
| $c_1$ | $\{i_1\}$ | $\{t_1, t_2, t_5\}$ | 3 | 1 | $\frac{3}{4}$ | $\frac{1}{3}$ |
| $c_2$ | $\{i_2\}$ | $\{t_1, t_3, t_4\}$ | 3 | 1 | $\frac{3}{4}$ | $\frac{1}{2}$ |
| $c_3$ | $\{i_3\}$ | $\{t_2, t_4, t_5\}$ | 3 | 1 | $\frac{3}{4}$ | $\frac{1}{2}$ |
| $c_4$ | $\{i_4\}$ | $\{t_1, t_2, t_3\}$ | 3 | 1 | $\frac{3}{4}$ | $\frac{2}{3}$ |
| $c_5$ | $\{i_1, i_3\}$ | $\{t_2, t_5\}$ | 2 | 2 | $\frac{2}{3}$ | $\frac{1}{2}$ |
| $c_6$ | $\{i_1, i_4\}$ | $\{t_1, t_2\}$ | 2 | 2 | $\frac{1}{2}$ | $\frac{1}{3}$ |
| $c_7$ | $\{i_2, i_3\}$ | $\{t_4\}$ | 1 | 2 | 0 | $\frac{2}{3}$ |
| $c_8$ | $\{i_2, i_4\}$ | $\{t_1, t_3\}$ | 2 | 2 | $\frac{1}{3}$ | $\frac{1}{2}$ |
| $c_9$ | $\{i_1, i_3, i_4\}$ | $\{t_2\}$ | 1 | 3 | 0 | $\frac{1}{2}$ |
| $c_{10}$ | $\{i_1, i_2, i_4\}$ | $\{t_1\}$ | 1 | 3 | 0 | $\frac{1}{3}$ |
| $c_{11}$ | $\{i_1, i_2, i_3, i_4\}$ | $\emptyset$ | 0 | 4 | 0 | 0 |

Table 6.4 – The set $\mathcal{F}$ of all formal concepts contained in the dataset described in Table 6.3. For each formal concept, we give its *intent*, *extent*, *frequency*, *size*, *diameter* and *split*.

Formal concepts are partially ordered with respect to the inclusion partial order defined on transactions or, equivalently, on items.

**Definition 6.7** (Subconcept)**.** Let $c_1 = (T_1, I_1)$ and $c_2 = (T_2, I_2)$ be two formal concepts. $c_1$ is a subconcept of $c_2$, denoted $c_1 \leq c_2$ iff $T_1 \subseteq T_2$ or, equivalently, $I_2 \subseteq I_1$.

**Example 6.8.** In Table 6.4, $c_9 \leq c_1$ because $\{t_2\} \subset \{t_1, t_2, t_5\}$ and $\{i_1\} \subset \{i_1, i_3, i_4\}$. $c_1$ and $c_3$ are not comparable because $\{t_1, t_2, t_5\}$ and $\{t_2, t_4, t_5\}$ are not comparable by means of set inclusion.

**Definition 6.9.** The set of all formal concepts $\mathcal{F}$ equipped with the subconcept ordering $\leq$ is called a concept lattice of $(\mathcal{T}, \mathcal{I}, \mathcal{R})$.

**Example 6.10.** We show in Fig. 6.1 the concept lattice of the formal concepts extracted from our example transactional dataset. Each node of the lattice corresponds to a formal concept and edges correspond to direct subconcept relations.

**Measures associated with formal concepts.** Two classical measures for characterizing formal concepts are the frequency and the size, which correspond to the number of transactions and items, respectively.

**Definition 6.11** (Frequency and size of a formal concept)**.** Let $c = (T, I)$ be a formal concept, $frequency(c) = \#T$ and $size(c) = \#I$.

Some measures that are often used to evaluate the quality of a clustering may be used to characterize formal concepts. In particular, the diameter and the split of a cluster evaluate the within-cluster homogeneity and the between-cluster separation, respectively. We may use them to evaluate the homogeneity of a formal concept and its separation with other transactions of the database.

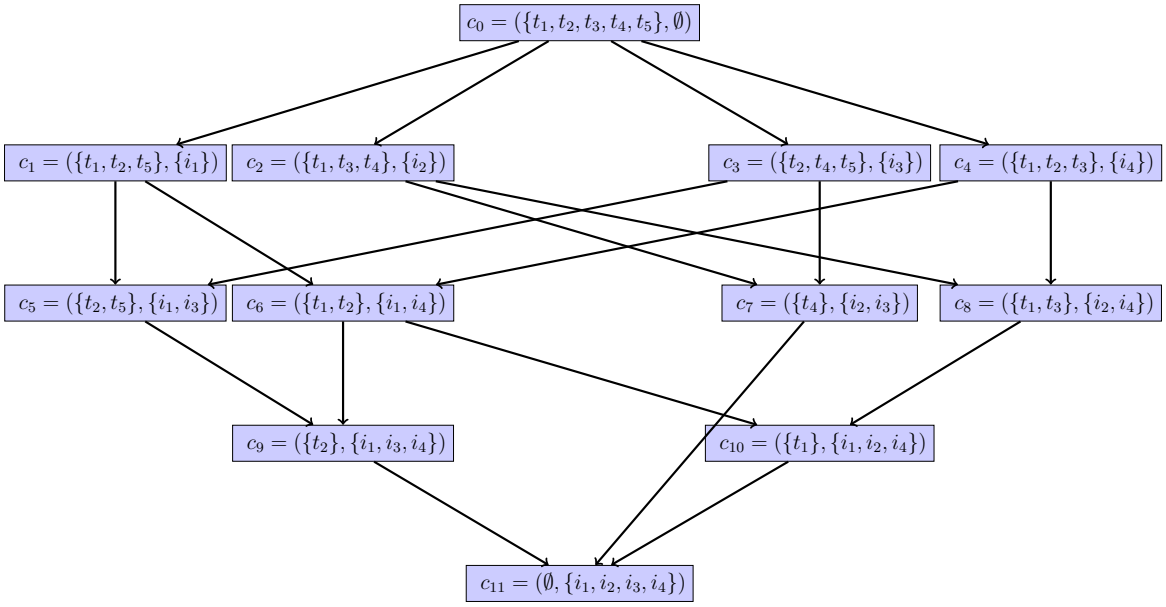



Figure 6.1 – Representation of the concept lattice of the formal concepts of the dataset represented in Table 6.3.

**Definition 6.12** (Diameter and split of a formal concept)**.** Let be $\mathcal{T}$ a set of transactions, $c = (T, I)$ a formal concept, and $d : \mathcal{T} \times \mathcal{T} \to \mathbb{R}$ a dissimilarity measure. The *diameter* of $c$ is the maximal dissimilarity between two transactions of $c$, *i.e.*,

$$diameter(c) = \max_{t,t' \in T} d(t, t')$$

and the *split* of $c$ is the minimal dissimilarity between a transaction of $c$ and a transaction not contained in $c$, *i.e.*,

$$split(c) = \min_{t \in T, t' \in \mathcal{T} \setminus T} d(t, t').$$

We may consider different dissimilarity measures for computing the diameter or the split of a formal concept. A classical measure is the Jaccard distance [Jac01] which measures the dissimilarity of two transactions by means of the ratio between the number of items they share and the total number of their items.

**Definition 6.13** (Jaccard distance)**.** The Jaccard distance between two transactions $t$ and $t'$ is:

$$d(t, t') = 1 - \frac{\#(itemset(t) \cap itemset(t'))}{\#(itemset(t) \cup itemset(t'))}$$

**Example 6.14.** In Table 6.4, we give for each formal concept of the dataset the value of each measure defined above. Concept $c_3$ has a frequency of 3 and a size of 1. Its diameter is $\frac{3}{4}$ because $d(t_2, t_4) = \frac{3}{4}$ and its split is $\frac{1}{2}$ because $d(t_1, t_2) = \frac{1}{2}$. As expected, formal concepts with only one transaction, *i.e.*, $c_7, c_9, c_{10}$, have a high size and a diameter equal to 0. Formal concepts with the highest frequency, *i.e.*, $c_1, c_2, c_3, c_4$, have the lowest size.

**Formal concepts and closed itemset mining.** Formal concepts correspond to *closed itemsets* as defined in the data mining community [Pas+99a]. The set $\mathcal{F}$ of all formal concepts may be computed by using algorithms dedicated to the enumeration of frequent closed itemsets, provided that the frequency threshold is set to 1.

Many algorithms with a variety of approaches have been proposed to compute frequent formal concepts [Aré+07; Kuz+02; Zak+05; Pas+99b; Pas+99a; Pei+00; Zak00; Zak+99]. Part of these algorithms perform additional tasks such as constructing the concept lattice that can slow down the performance and consume memory [Aré+07; Kuz+02; Zak+05], others enumerate frequent itemsets and output only those being closed [Pas+99b; Pas+99a; Pei+00; Zak00; Zak+99]. The two fastest algorithms to date are *LCM* (Linear time Closed pattern Miner) [Uno+04] and *In-Close* [And09]. Both approaches are conceptually based on the well known algorithm *Close-By-One* [Kuz99; And09].

In particular, *LCM* [Uno+04] is able to extract all formal concepts $\mathcal{F}$ in linear time with respect to $\#\mathcal{F}$. It uses a technique called *prefix preserving closure extension* which extends a closed pattern to another one by adding new items. Since any closed itemset is generated by the extension of exactly one of the other closed itemsets, frequent closed itemsets can be enumerated in a depth-first search manner with no need to store previously computed itemsets. Hence, the memory usage depends only on the size of the input dataset. Furthermore, it enables to completely prune unnecessary non-closed frequent patterns. This is a big advantage over approaches based on frequent itemset mining since the number of frequent itemsets can be exponentially larger than the number of closed itemsets.

As there is usually a huge number of closed itemsets, we may add constraints or optimization criteria to identify relevant concepts. For example, we may search for closed itemsets whose frequency is greater than some given threshold and whose size is maximal. Using CP to model and solve itemset search problems with additional user-constraints is a topic which has been widely explored during the last ten years [Rae+08; Khi+10; Gun+11; Gun15; Laz+16; Sch+17]. Indeed, CP allows one to easily model various constraints on the searched itemsets, corresponding to application-dependent constraints for example. These constraints are used to filter the search space during the mining process, and allow CP to be competitive with dedicated mining tools such as LCM.

## 6.3 Conceptual Clustering

Clustering is an unsupervised classification approach the goal of which is to partition a set of objects into homogeneous and well-separated groups called clusters: Objects within a same cluster are similar, whereas objects in different clusters are dissimilar. Clustering has a large field of applications such as economics, sociology, or natural sciences.

To partition objects into clusters, a first possibility is to rely on some dissimilarity measure between objects. A well-known algorithm to address this problem is *k-means* [Mac67], for example. This problem may also be seen as an optimization problem the goal of which is to find a partition of the objects that optimizes some

given criterion such as the split, the diameter for example. In [Duo14], Duong describes CP models for solving these problems, and shows that application-dependent constraints may easily be added to customize the model.

Conceptual clustering is an alternative approach which provides a description of each cluster in addition to clusters [Mic80]. The description or concept of a cluster is a set of properties shared by objects of the cluster. In the 1990s, many approaches have been proposed to learn from observations by using conceptual clustering [Che+85; Ste+86; Mic+83]. For example, `COBWEB` [Fis87] is an incremental conceptual clustering approach where the search is guided with a *category utility* measure based on object similarity, in order to maximize the information that can be predicted from observations, for each cluster. Each concept corresponds to a cluster and is described by a list of attributes associated with probabilities.

In many recent works [Dao+15a; Gun15; Oua+16], clusters are associated with formal concepts, and we consider this definition in this thesis.

**Definition 6.15** (Conceptual clustering)**.** Let $(\mathcal{T}, \mathcal{I}, \mathcal{R})$ be a transactional database. A conceptual clustering is a set of $k$ formal concepts $\mathcal{C} = \{(T_1, I_1), \ldots, (T_k, I_k)\}$ such that $\{T_1, \ldots, T_k\}$ is a partition of the set $\mathcal{T}$ of transactions, *i.e.*, $\forall t \in \mathcal{T}, \exists! i \in [1, k], t \in T_i$.

In the following, we talk indifferently of clusters or of formal concepts of a conceptual clustering since every cluster corresponds to a formal concept.

**Example 6.16.** Let us consider the set of formal concepts displayed in Table 6.4. Three examples of conceptual clusterings are: $P_1 = \{c_2, c_5\}, P_2 = \{c_3, c_8\}$, and $P_3 = \{c_5, c_7, c_8\}$.

**Optimization criteria.** There may exist different solutions to a conceptual clustering problem, and we may consider different optimization criteria to search for the best solution. In particular, the utility measures introduced in Section 6.2 (*i.e.*, size, frequency, split and diameter) may be used to define an objective function to optimize. These measures evaluate the quality of each cluster separately, and we may consider different ways for aggregating them to evaluate the quality of a clustering. More formally, let be $P = \{c_1, \ldots, c_k\}$ a conceptual clustering, and $u : P \to \mathbb{R}$ an utility measure such that $u(c_i)$ reflects the quality of the cluster $c_i$. Without loss of generality, we assume that the higher the value of $u(c_i)$, the better the quality of $c_i$: If this is not the case (as for the diameter, for example), the measure may be multiplied by $-1$ to define the utility (*e.g.*, $u(c_i) = -diameter(c_i)$). In [Ari+18], Aribi *et al.* consider four different aggregation functions:

- $sum_u$, which returns the sum of all utilities, *i.e.*, $sum_u(P) = \sum_{c_i \in P} u(c_i)$;
- $min_u$, which returns the smallest utility, *i.e.*, $min_u(P) = \min_{c_i \in P} u(c_i)$;
- $dev_u$, which returns the gap between the largest and the smallest utility, *i.e.*, $dev_u(P) = \max_{c_i \in P} u(c_i) - \min_{c_i \in P} u(c_i)$;
- $OWA_u$ (*Ordered Weighted Average*), which returns a weighted sum of utilities, *i.e.*, $OWA_u(P) = \sum_{i \in [1,k]} w_i x_i$ where $(x_1, \ldots, x_k)$ is a permutation of $(u(c_1), \ldots, u(c_k))$ such that $\forall i \in [1, k-1], x_i \leq x_{i+1}$, and $w_i = \sin \frac{(k+1-i)\pi}{2k+1}$ is the weight of the $i^{th}$ largest utility value.

| $u_1$ | $u_2$ | $u_3$ | $u_4$ | $u_5$ | *sum* | *min* | *dev* | *OWA* |
|---|---|---|---|---|---|---|---|---|
| 1 | 9 | 8 | 6 | 6 | 30 | 1 | 8 | $9w_1 + 8w_2 + 6w_3 + 6w_4 + w_5 = 24.25$ |
| 4 | 6 | 5 | 7 | 8 | 30 | 4 | 4 | $8w_1 + 7w_2 + 6w_3 + 5w_4 + 4w_5 = 22.65$ |
| 4 | 6 | 5 | 4 | 4 | 23 | 4 | 2 | $6w_1 + 5w_2 + 4w_3 + 4w_2 + 4w_1 = 16.80$ |
| 4 | 6 | 6 | 5 | 5 | 26 | 4 | 2 | $6w_1 + 6w_2 + 5w_3 + 5w_2 + 4w_1 = 19.01$ |

Table 6.5 – Comparison of four aggregation measures on four different clusterings with $k = 5$: Each line gives the utility value $u_i$ of each cluster $c_i$, and the aggregation computed with *sum*, *min*, *dev*, and *OWA*.

Table 6.5 compares these four aggregation functions on different utility measure values. Maximizing the *sum* ensures a good average quality, but the repartition of this quality among clusters may not be equitable: Some clusters may have very low utility values while some others may have very high values. This is the case, for example, of the first clustering of Table 6.5 which has a very low quality cluster (whose utility is 1) while its utility sum is maximal. Maximizing *min* ensures a minimal quality over all clusters, whereas minimizing *dev* favors clusterings with clusters of homogeneous quality. On our example, the second and third clusterings both have a minimum utility value of 4, but the third clustering has more homogeneous utility values (ranging between 4 and 6 instead of 4 and 8). *min* and *dev* are not sensible to intermediate values: Two clusterings with the same minimal and maximal utilities among all clusters, but different intermediate values, have the same aggregated value. On our example, this is the case for the third and fourth clustering, that both have values ranging between 4 and 6, though the fourth clustering has higher intermediate values. *OWA* has been introduced in [YAG93] to ensure a better equity by weighting utilities according to their rank. On our example, the fourth clustering is preferred to the third one when considering $OWA$.

In our applicative context, the *min* aggregation function is well suited because it ensures that all clusters have a minimal quality. In other words, it discards clusters of low utility. There is no need to ensure some kind of equity between the different clusters as it is most probable that all relevant configuration parts are not equally good according to utility measures.

**Impact of the number of clusters on quality measures.** The number $k$ of clusters is an important parameter which has a great influence on the *size*, the *frequency*, the *split* and the *diameter* of the clusters. In particular, when $k$ is small (typically, when $k = 2$), clusters contain more transactions as each transaction must belong to one cluster and, therefore, cluster frequencies tend to have higher values. In this case, cluster sizes tend to have low values because the number of shared items can only decrease when adding a transaction to a cluster. Also, when $k$ is small, cluster splits and diameters tend to have high values.

As a counterpart, when $k$ is large (and close to the number of transactions), clusters contain less transactions and, therefore, cluster frequencies, splits and diameters tend to have low values, whereas cluster sizes tend to have high values.

**Example 6.17.** In Table 6.4, $P_1 = \{c_2, c_5\}$ is the conceptual clustering with the smallest number of clusters, and it maximizes $Min_{frequency}$ and $Min_{split}$. $P_3 = \{c_5, c_8, c_7\}$

is the conceptual clustering with the largest number of clusters, and it maximizes $Min_{size}$ and $Min_{-diameter}$.

**Soft Clustering.** In some applications, constraining the clusters to be an exact partition of the set of transactions is not relevant, and we may soften this constraint. In this case, some transactions may belong to more than one cluster (*i.e.*, the non-overlapping constraint is relaxed) or to no cluster (*i.e.*, the covering constraint is relaxed) [Kog+06]. Many soft clustering techniques are characterized by a relaxation of the borders of the clusters [Bez81; Bez+78]. The soft borders address many typical real-life applications where overlapping clusters or uncertain cluster memberships can often be observed.

Soft conceptual clustering may be expressed with respect to the relation between transactions and formal concepts [Oua+16]. More formally, given two thresholds $1 \leq \delta_o \leq \#\mathcal{T}$ and $\delta_c \leq \#\mathcal{T}$, a set of formal concepts $P = \{c_1 = (I_1, T_1), \ldots, c_k = (I_k, T_k)\}$ is a soft conceptual clustering if:

- at most $\delta_c$ transactions are not covered by $P$, *i.e.*, $\#\mathcal{T} - \#(\bigcup_{i \in [1,k]} T_i) \leq \delta_c$.
- each transaction belongs to at most $\delta_o$ clusters, *i.e.*, $\forall t \in \mathcal{T}, \#\{c_i | t \in T_i\} \leq \delta_o$

**Example 6.18.** For the transactional dataset example of Table 6.3, if we set $\delta_c = 1$ and $\delta_o = 1$, $\{c_1, c_7\}$ or $\{c_3, c_{10}\}$ are soft conceptual clusterings. $\{c_6, c_7\}$ is not a solution because only 3 transactions are covered. If we set $\delta_c = 0$ and $\delta_o = 2$, $\{c_2, c_3\}$ is a soft conceptual clustering whereas $\{c_1, c_2, c_4\}$ is not because $t_1$ belongs to three clusters.

## 6.4 Declarative approaches for conceptual clustering

In this section, we describe existing declarative approaches for solving conceptual clustering problems. These declarative approaches allow the user to customize the problem by adding application-dependent constraints. In Section 6.4.1, we describe a first CP model which uses Boolean variables to model formal concepts. In Section 6.4.2, we describe another CP model which uses set variables. In Section 6.4.3, we describe an hybrid approach which combines a dedicated mining tool for extracting all formal concepts with an Integer Linear Programming model for selecting a subset of these formal concepts that defines a partition.

We present all models using the transactional database notations introduced in Section 6.2.

### 6.4.1 Boolean-based CP model

This CP model was introduced by Guns *et al.* in [Gun15]. It assumes that the number of clusters is defined by a constant value denoted $k$.

**Variables.** Boolean variables are used to represent both intents and extents of formal concepts that correspond to clusters:

- For each cluster $c \in [1, k]$ and each item $i \in \mathcal{I}$, the boolean variable $I_{c,i}$ is set to 1 iff $i$ belongs to the intent of the formal concept associated with cluster $c$;

- For each cluster $c \in [1, k]$ and each transaction $t \in \mathcal{T}$, the boolean variable $T_{c,t}$ is set to 1 iff $t$ belongs to the extent of the formal concept associated with $c$.

**Constraints.** For each cluster $c \in [1, k]$, we ensure that it corresponds to a formal concept by adding the following constraints:

- An *extent* constraint ensures that a transaction $t$ belongs to the extent of $c$ iff each item is either included in $itemset(t)$ or excluded from the intent of $c$, *i.e.*,

$$\forall t \in \mathcal{T}, T_{c,t} = 1 \Leftrightarrow \sum_{i \in \mathcal{I}} I_{c,i} \cdot (1 - \mathcal{R}_{t,i}) = 0$$

  where $\mathcal{R}_{t,i} = 1$ iff $(t, i) \in \mathcal{R}$;

- An *intent* constraint ensures that an item $i$ belongs to the intent of $c$ iff each transaction is either included in the extent of $c$ or its itemset does not contain $i$, *i.e.*,

$$\forall i \in \mathcal{I}, I_{c,i} = 1 \Leftrightarrow \sum_{t \in \mathcal{T}} T_{c,t} \cdot (1 - \mathcal{R}_{t,i}) = 0.$$

These constraints are implemented with $k \cdot (\#\mathcal{T} + \#\mathcal{I})$ reified constraints.

To ensure the partition constraint, we constrain each transaction to belong to exactly one cluster, *i.e.*,

$$\forall t \in \mathcal{T}, \sum_{c \in [1,k]} T_{c,t} = 1.$$

**Objective function.** The utility measures introduced in Section 6.3 are modeled by introducing an integer variable $u_c$ for each cluster $c \in [1, k]$, and constraining this variable to be equal to the utility measure of cluster $c$. If the utility measure is *frequency* (resp. *size*), this constraint is $u_c = \sum_{t \in \mathcal{T}} T_c[t]$ (resp. $u_c = \sum_{i \in \mathcal{I}} I_c[i]$).

Given these variables, we can search for a conceptual clustering that maximizes the minimum utility, among all clusters, by defining the objective function to maximize as $\min_{c \in [1,k]} u_c$.

### 6.4.2 Set-based CP model

In [Dao+15b], Dao *et al.* describe a CP model for clustering problems where a dissimilarity measure between objects is provided. In this case, the goal is to find a partition of the objects which satisfies some constraints and optimizes an objective function defined by means of this dissimilarity measure. This CP model is extended to solve conceptual clustering problems in [Dao+15a]. Experimental results reported in [Dao+15a] show that this model outperforms the boolean model of [Gun15] introduced in the previous section. This model also assumes that the number of clusters is defined by a constant value denoted $k$.

**Variables.** This model uses the following variables:

- For each transaction $t \in \mathcal{T}$, an integer variable $G_t$ represents the cluster of $t$ and its domain is $D(G_t) = [1, k]$.
- For each cluster $c \in [1, k]$, a set variable $Intent_c$ represents the intent of the set of transactions in $c$ with $D(Intent_c) = [\emptyset, \mathcal{I}]$.

**Constraints.** For each cluster $c \in [1, k]$, the *extent* constraint is defined by:

$$\forall t \in \mathcal{T}, G_t = c \Leftrightarrow Intent_c \subseteq itemset(t)$$

and it is implemented thanks to $k \times \#\mathcal{T}$ reified constraints.

For each cluster $c \in [1, k]$, the *intent* constraint is defined by:

$$Intent_c = \bigcap_{t \in \mathcal{T}, G_t = c} itemset(t)$$

and it is implemented thanks to $k$ constraints, such that each of these $k$ constraints needs $\#\mathcal{T}$ reified domain constraints to build the set of all transactions in cluster $c$, and a set *element* global constraint to select the corresponding itemsets and intersect them.

Symmetries (due to the fact that cluster numbers may be swapped) are broken by adding a $precede(G, [1, k])$ constraint [Law+04].

**Objective function.** We can search for a conceptual clustering that maximizes the minimum utility, among all clusters, as follows:

- To maximize $Min_{frequency}$, we introduce an integer variable $F$ to be maximized. For each cluster $c$, we add the constraint $atLeast(F, G, c)$ to ensure that $F$ is smaller than or equal to the number of transactions in $c$.
- To maximize $Min_{size}$, we introduce an integer variable $T$ to be maximized. For each cluster $c \in [1, k]$, we add the constraint $T \leq \#Intent_c$ to ensure that $T$ is smaller than or equal to the number of items in the intent of $c$.
- To maximize $Min_{split}$, we search for all solutions, and we dynamically add constraints each time a new solution is found. More precisely, when a new solution $I$ is found, we compute its associated minimum split *split*, *i.e.*,

  $$split = \min_{t,t' \in \mathcal{T}, I[G_t] \neq I[G_{t'}]} d(t, t')$$

  and for each pair of transactions $\{t, t'\} \subseteq \mathcal{T}$ such that $d(t, t') \leq split$, we add the constraint $G_t = G_{t'}$ to $C$. This constraint ensures that the next solution (if any) will have a larger split value.
- To maximize $Min_{-diameter}$, when a new solution $I$ is found, we compute its associated maximum diameter *diam*, *i.e.*,

  $$diam = \max_{t,t' \in \mathcal{T}, I[G_t] = I[G_{t'}]} d(t, t')$$

  and for each pair of transactions $\{t, t'\} \subseteq \mathcal{T}$ such that $d(t, t') \geq diam$, we add the constraint $G_t \neq G_{t'}$. This constraint ensures that the next solution (if any) will have a smaller diameter value.

| | |
|---|---|
| Maximize | $\sum_{f \in \mathcal{F}} v_f x_f$ |
| Subject to | (1) $\forall t \in \mathcal{T}, \sum_{f \in \mathcal{F}} a_{tf} x_f = 1$ |
| | (2) $k = \sum_{f \in \mathcal{F}} x_f$ |
| | (3) $k_{min} \leq k \leq k_{max}$ |
| | $k \in \mathbb{N}, x_f \in \{0,1\}, f \in \mathcal{F}$ |

Figure 6.2 – ILP model for conceptual clustering

**Extension of the model to the case where $k$ is not fixed.** The model proposed in [Dao+15a] assumes that the number of clusters is fixed to a constant value $k$.

It may be extended to the case where the number of clusters is bounded between $k_{min}$ and $k_{max}$ by introducing an integer variable $k$ with $D(k) = [k_{min}, k_{max}]$. To ensure that $k$ is equal to the number of non-empty clusters, we add the constraint $k = \max_{t \in \mathcal{T}} G_t$.

When maximizing $Min_{frequency}$, for each cluster $c \in [1, k_{max}]$, we post the constraint:

$$c \leq k \Rightarrow atLeast(F, G, c).$$

When maximizing $Min_{size}$, for each cluster $c \in [1, k_{max}]$, we post the constraint:

$$c \leq k \Rightarrow T \leq \#Intent_c.$$

### 6.4.3 Hybrid ILP model for conceptual clustering

In [Oua+16], Ouali *et al.* propose to compute conceptual clusterings by combining two exact techniques: In a first step, a dedicated closed itemset mining tool (*i.e.*, `LCM` [Uno+04]) is used to compute the set $\mathcal{F}$ of all formal concepts and, in a second step, Integer Linear Programming (ILP) is used to select a subset of $\mathcal{F}$ that is a partition of the set $\mathcal{T}$ of transactions and optimizes some given criterion.

The ILP model used to solve the second step is described in Fig. 6.2. This model associates a boolean variable $x_f$ with every formal concept $f \in \mathcal{F}$ such that $x_f = 1$ iff $f$ is selected in the solution. Selected formal concepts are constrained to define a partition of $\mathcal{T}$ by posting constraint (1), where $a_{tf} = 1$ if the transaction $t$ belongs to the extent of the concept $f$, and 0 otherwise.

Contrary to the CP approaches of [Gun15; Dao+15b], the number of clusters is not fixed. A variable $k$ is constrained to be equal to the number of selected concepts by posting constraint (2) and constraint (3) allows to control the bounds of $k$.

A cost value $v_f$ is associated with every formal concept $f \in \mathcal{F}$. This cost corresponds to the utility measure and may be the size, the frequency or any other utility measure associated with formal concepts, as introduced in Section 6.2.

The model introduced in [Oua+16] assumes that the aggregation function is *sum*, and the goal is to maximize the sum of utility measures associated with the selected concepts. If the case is not explicitly discussed in [Oua+16], we may easily modify this ILP model to the aggregation function *min*: We introduce a variable $v_{min}$ and enforce it to be smaller than or equal to utility values of selected concepts by adding the constraint $\forall t \in T, v_{min} \leq v_f x_f + M(1 - x_f)$, where $M$ is a positive constant greater than the largest value of $v$.

## 6.5 Discussion

We have introduced in this section the main definitions and principles related to formal concepts and conceptual clustering. In Chapter 13, we show how to use conceptual clustering to extract relevant parts of configurations from existing ERP configurations. We have chosen to use CP to achieve this task, because CP languages allow us to easily add new constraints or objective functions to customize the model and extract more relevant concepts according to ERP expert feedbacks.

We have described existing CP models for solving conceptual clustering problems with CP. These CP models consider that the number of clusters $k$ is a constant which is known in advance. This is not the case in our application. If existing models can easily be extended to the case where the number of clusters is not known, they do not scale well in this case and they are not able to solve all our instances within a reasonable amount of time. This motivated us to introduce new CP models for solving conceptual clustering problems, which are described in Chapter 9.

We have also described an hybrid approach that solves conceptual clustering problems by combining a dedicated tool that extracts formal concepts with an ILP model that selects a subset of formal concepts. The problem solved by the ILP model corresponds to the *exact cover* problem. We describe this problem in the next chapter, and we introduce a new global constraint dedicated to this problem in Chapter 10.

# Chapter 7

# Exact Cover problem

## Contents



As pointed out by Ian Davidson during a tutorial he gave in December 2017 on *Data Mining and Machine Learning using Constraint Programming Languages*, the problem that aims at selecting a subset of formal concepts that defines a partition of the set of transactions is a well known problem, called *Exact Cover*.

In Section 7.1, we formally define this problem and introduce some notations. In Section 7.2, we describe three applications of this problem. In Section 7.3, we describe an algorithm introduced by Knuth to solve this problem, together with a nice data structure, called *Dancing Links*, which is used to efficiently restore states when backtracking. In Sections 7.4 and 7.5, we introduce CP and SAT models to solve this problem. In Section 7.6, we experimental compare these declarative approaches with the dedicated algorithm of Knuth.

## 7.1 Definitions and notations

An instance of the *Exact Cover Problem* (EC) is defined by a couple $(S, P)$ such that $S$ is a set of elements and $P \subseteq \mathcal{P}(S)$ is a set of subsets of $S$. EC aims at deciding if there exists a subset $C \subseteq P$ which is a partition of $S$, *i.e.*, a subset $C \subseteq P$ such that:

$$\forall a \in S, \#\{u \in C | a \in u\} = 1.$$

This problem is $\mathcal{NP}$-complete [Kar72].

**Example 7.1.** For example, let us consider the instance $(S, P)$ displayed in Fig. 7.1. A solution is: $C = \{v, x, z\}$.

$$\begin{aligned} S &= \{a, b, c, d, e, f, g\} \\ P &= \{t, u, v, w, x, y, z\} \end{aligned}$$

where

$$t = \{a, g\}, u = \{a, d, g\}, v = \{a, d\}, w = \{d, e, g\}, x = \{c, e, f\}, y = \{b, c, f\}, z = \{b, g\}$$

Figure 7.1 – Example of instance of the exact cover problem.

Elements of $S$ are denoted $a$, $b$, $c$, etc, whereas elements of $P$ (*i.e.*, subsets) are denoted $t$, $u$, $v$, etc. For each element $a \in S$, we denote $cover(a)$ the set of subsets that contain $a$:

$$cover(a) = \{u \in P | a \in u\}.$$

**Example 7.2.** For example, in Fig. 7.1, we have $cover(a) = \{t, u, v\}$.

Two subsets $u, v \in P$ are *compatible* if $u \cap v = \emptyset$ and, for every subset $u \in P$, we denote $incompatible(u)$ the subsets of $P$ that are not compatible with $u$:

$$incompatible(u) = \{v \in P \setminus \{u\} : u \cap v \neq \emptyset\}$$

**Example 7.3.** For example, in Fig. 7.1, we have $incompatible(x) = \{w, y\}$.

The maximal cardinality of a subset in $P$ is denoted $n_p$, the maximal number of subsets that cover an element is denoted $n_c$, and the maximal number of subsets that are not compatible with another subset is denoted $n_i$:

$$\begin{aligned} n_p &= \max_{u \in P} \#u \\ n_c &= \max_{a \in S} \#cover(a) \\ n_i &= \max_{u \in P} \#incompatible(u) \end{aligned}$$

Given a set $C \subseteq P$ of selected subsets which are all pairwise compatible, the set of elements that are not covered by a subset in $C$ is denoted $S_C$:

$$S_C = \{a \in S : \forall u \in C, a \notin u\}$$

and for every non covered element $a \in S_C$, the set of subsets that cover $a$ and are compatible with every subset in $C$ is denoted $cover_C(a)$:

$$\forall a \in S_C, cover_C(a) = \{v \in cover(a) : \forall u \in C, v \cap u = \emptyset\}.$$

**Example 7.4.** For example, in Fig. 7.1, if $C = \{x\}$ then $S_C = \{a, b, d, g\}$ and $cover_C(g) = \{t, u, z\}$.

## 7.2 Applications of EC

Many real-world problems involve solving an exact cover problem. For example, Junttila and Kaski describe in [Jun+10] a benchmark of EC instances of combinatorial origin: Bell numbers, Perfect matchings in $K_{2n}$ graphs, Steiner triple systems, Latin squares, and Kirkman triple systems.

In this section, we describe three applications of EC, *i.e.*, tilling, instruction selection, and conceptual clustering.

**Tilling.** In [Knu09], Knuth illustrates EC on the problem that aims at tiling a rectangle figure composed of equal squares with a set of pentaminoes, as illustrated in Fig. 7.2. To model this problem as an EC, we define the sets $S$ and $P$ as follows:

- $S$ contains one element for each square of the rectangle to tile, and one element for each pentamino shape;
- $P$ contains one subset for each possible position of a pentamino on the rectangle, and this subset contains the shape of the pentamino and all squares that are covered by the pentamino.

For example, for the $6 \times 10$ rectangle displayed in Fig 7.2, we define:

$$S = \{s_{i,j} | i \in [1,6], j \in [1,10]\} \cup \{I, P, Y, V, T, X, L, F, Z, W, N, U\}.$$

The pentamino with a I shape has 20 different possible positions when placing it vertically and 36 different possible positions when placing it horizontally. Hence, there are 56 subsets in $P$ which correspond to the placement of this pentamino, *i.e.*,

$$\begin{aligned} & \{\{I, s_{i,j}, s_{i+1,j}, s_{i+2,j}, s_{i+3,j}, s_{i+4,j}\} | i \in [1,2], j \in [1,10]\} \\ \cup \quad & \{\{I, s_{i,j}, s_{i,j+1}, s_{i,j+2}, s_{i,j+3}, s_{i,j+4}\} | i \in [1,6], j \in [1,6]\} \end{aligned}$$

To find a tiling of the rectangle with non-overlapping polyominoes, we need to select a subset of $P$ that defines a partition of $S$.

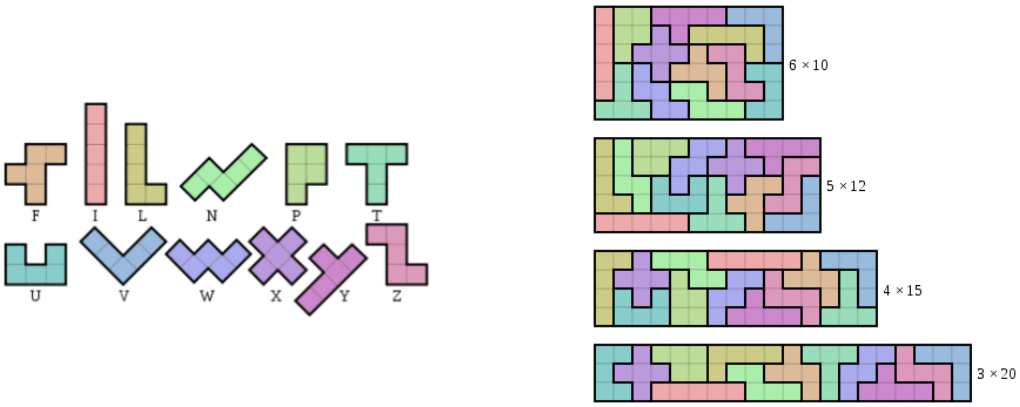



Figure 7.2 – Polyomino tiling problem. Images from https://en.wikipedia.org/wiki/Pentomino

**Instruction selection.** In [HB18], Hjort Blindell shows how to use CP to solve an instruction selection problem that occurs when compiling a source code to generate an executable code. This instruction selection problem may be decomposed into two subproblems:

1. A *matching problem*, that involves finding the instructions provided by the target processor that can implement one or more operations in the source code to compile;

2. A *selection problem*, that involves selecting a subset of these target instructions such that each source operation is covered exactly once.

The selection problem is an EC problem.

**Conceptual Clustering.** Our interest for EC comes from the conceptual clustering problem which is described in Section 6.3. As discussed in Section 6.4.3, this problem may be solved in two steps. The first step involves extracting the set $\mathcal{F}$ of all formal concepts from a set of transactions $\mathcal{T}$, and it may be solved by using a dedicated and efficient tool such as `LCM` [Uno+04]. The second step involves finding a subset of $\mathcal{F}$ that forms a partition $\mathcal{T}$ (and, optionally, that optimizes some given criteria). This second step is the instance of EC defined by the couple $(S = \mathcal{T}, P = \mathcal{F})$.

## 7.3 Dedicated Algorithm DLX

Knuth has introduced an algorithm called X to recursively enumerate all solutions of an instance $(S, P)$ of EC [Knu09]. This algorithm is displayed in Figure 6[1] and has three input parameters: the sets $S$ and $P$ that define the instance of EC to solve, and a partial cover $C \subseteq P$ that contains the subsets that have already been selected in the solution (for the first call to X, we have $C = \emptyset$). If the set $S_C$ of non covered elements is empty, then $C$ is a solution and the algorithm outputs it (line 3). If there is an element $a \in S_C$ such that $cover_C(a) = \emptyset$, then $a$ cannot be covered by any subset compatible with $C$ and the search must backtrack. Otherwise, we choose an element $a \in S_C$ (line 7) and, for each subset $u \in cover_C(a)$, we recursively try to add $u$ to the partial solution (line 9).

A first key point for an efficient enumeration process is to use an ordering heuristic to choose the next element $a$ (line 7). Knuth shows that this ordering heuristic has a great impact on performance, and that much better results are obtained by selecting an element $a \in S_C$ for which the number of subsets compatible with $C$ is minimal. Hence, the ordering heuristic used line 7 chooses an element $a \in S_C$ such that $\#cover_C(a)$ is minimal.

A second key point is to incrementally maintain $S_C$ and $cover_C(a)$ for each element $a \in S_C$. To this aim, Knuth introduces *Dancing Links* and the implementation of Algorithm X with Dancing Links is called DLX. As illustrated in Figure 7.3, the idea is to use doubly linked circular lists to represent a sparse matrix. Each cell $c$ in this matrix has five fields denoted $c.head$, $c.left$, $c.right$, $c.up$, and $c.down$, respectively.

[1] In [Knu09], the algorithm is introduced using a matrix representation (columns correspond to elements and lines to subsets). We describe it with the notations that are used in our propagation algorithms.

**Algorithm 6:** Algorithm X$(S, P, C)$

**Input:** An instance $(S, P)$ of EC and a set $C \subseteq P$ of selected subsets
**Postcondition:** Output every exact cover $C'$ of $(S, P)$ such that $C \subseteq C'$

```
1 begin
2     Let S_C = {a ∈ S : ∀u ∈ C, a ∉ u}
3     if S_C = ∅ then Output C;
4     else
5         ∀a ∈ S_C, let cover_C(a) = {v ∈ cover(a) : ∀u ∈ C, v ∩ u = ∅}
6         if ∀a ∈ S_C, cover_C(a) ≠ ∅ then
7             Choose an element a ∈ S_C
8             for each subset u ∈ cover_C(a) do
9                 X(S, P, C ∪ {u})
```

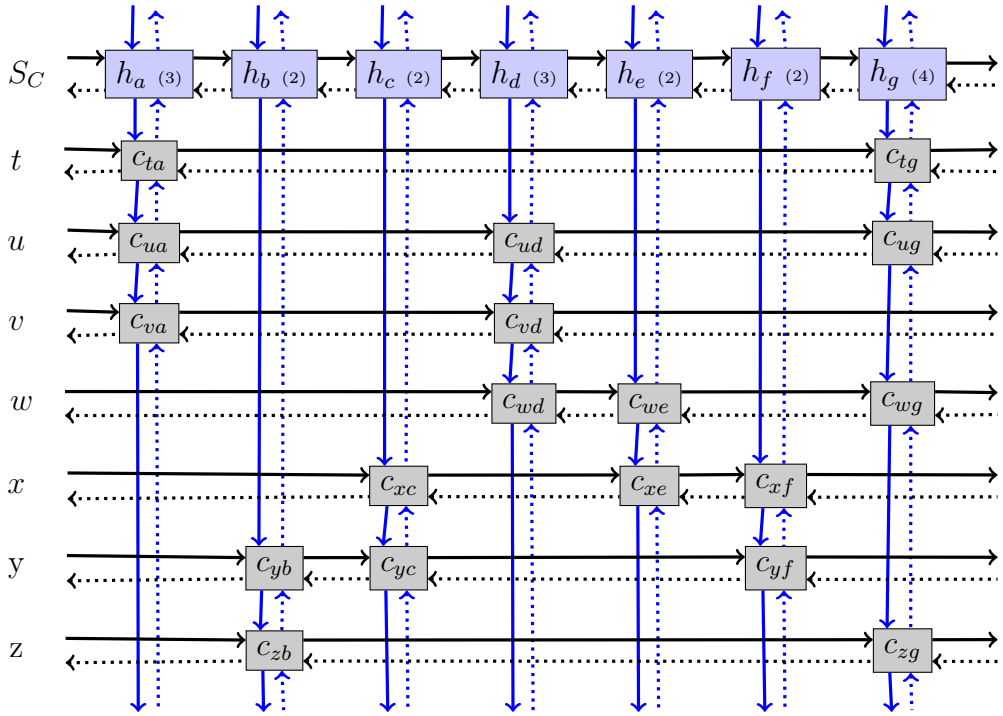



Figure 7.3 – Representation of the EC instance of Fig. 7.1 with Dancing Links when $C = \emptyset$. *right* (resp. *left*, *up*, and *down*) fields are represented by plain black (resp. dotted black, dotted blue, and plain blue) edges. Header cells are colored in blue, and their *size* fields are displayed in brackets. *head* fields are not displayed: the *head* field of each gray cell contains a pointer to the blue cell in the same column.

For each subset $u \in P$, the matrix has a row which contains a cell $c_{ua}$ for each element $a \in u$. This row is a doubly linked circular list, and we can iterate over all elements in $u$, starting from any cell in the row, by using *left* fields until returning back to the initial cell. If we use *right* fields instead of *left* fields, we also iterate over all elements in $u$, but we visit them in reverse order.

**Algorithm 7:** removeCells($u$)

```
1  for each a ∈ u do
2      h_a ← getHeader(a)
3      h_a.left.right ← h_a.right
4      h_a.right.left ← h_a.left
5      c_va ← h_a.down
6      while c_va ≠ h_a do
7          c_vb ← c_va.right
8          while c_vb ≠ c_va do
9              c_vb.down.up ← c_vb.up
10             c_vb.up.down ← c_vb.down
11             decrement c_vb.head.size
12             c_vb ← c_vb.right
13         c_va ← c_va.down
```

**Algorithm 8:** restoreCells($u$)

```
1  for each a ∈ u (in reverse order) do
2      h_a ← getHeader(a)
3      h_a.left.right ← h_a
4      h_a.right.left ← h_a
5      c_va ← h_a.up
6      while c_va ≠ h_a do
7          c_vb ← c_va.left
8          while c_vb ≠ c_va do
9              c_vb.down.up ← c_vb
10             c_vb.up.down ← c_vb
11             increment c_vb.head.size
12             c_vb ← c_vb.left
13         c_va ← c_va.up
```

Besides these $\#P$ rows, there is an extra row in the matrix, which is the first row and contains a cell $h_a$ for each non covered element $a \in S_C$. This cell is called the *header* and it has an extra field *size* which is equal to the cardinality of $cover_C(a)$. Like the other rows, the first row is a doubly linked circular list and we can iterate over all elements in $S_C$ by using *left* or *right* fields.

Each column of the matrix corresponds to an element $a \in S_C$ and is composed of $\#cover_C(a) + 1$ cells: the header $h_a$ plus one cell $c_{ua}$ for each subset $u \in cover_C(a)$. Each cell $c_{ua}$ in the column can access to its header thanks to the *head* field (*i.e.*, $c_{ua}.head = h_a$). This column is a doubly linked circular list, and we can iterate over all subsets in $cover_C(a)$, starting from the header $h_a$, by using *down* fields until returning to $h_a$. If we use *up* fields, we also iterate over all subsets in $cover_C(a)$, but we visit them in reverse order.

The advantage of using doubly linked circular lists is that a cell may be removed or restored (when backtracking) very easily. More precisely, to remove a cell $c$ from a column, we execute: $c.down.up \leftarrow c.up; c.up.down \leftarrow c.down$
To restore $c$ (when backtracking), we execute: $c.down.up \leftarrow c; c.up.down \leftarrow c$.
Similarly, to remove $c$ from a row, we execute: $c.right.left \leftarrow c.left; c.left.right \leftarrow c.right$
and to restore $c$ (when backtracking), we execute: $c.right.left \leftarrow c; c.left.right \leftarrow c$.
Also, doubly linked lists can be traversed in two directions: This way we can undo a sequence of cell removals by executing the inverse sequence of cell restorations.

The complete algorithms to update the matrix with Dancing Links are displayed in Algorithms 7 and 8: Algorithm 7 is called just before the recursive call (line 9 of Algorithm 6) to remove cells, and Algorithm 8 is called just after the recursive call (line 9 of Algorithm 6) to restore cells. The resulting algorithm is called DLX.

Algorithm 7 is called after the addition of a subset $u$ to $C$. For each element $a \in u$, we remove the header $h_a$ of the column associated with $a$ (lines 3-4). Then, we iterate over all subsets $v \in cover_C(a)$ by traversing the column list associated with $a$, starting from its header and using *down* fields. Each cell $c_{va}$ in this column corresponds to a subset $v \in cover_C(a)$ which is incompatible with $u$ (since $a$ is already covered by $u$). Hence, for each element $b \in v$, we must remove $v$ from $cover_C(b)$. To this aim, we iterate over all elements $b \in v$ by traversing the row list associated with $v$, starting

from $c_{va}$ and using *right* fields, and we remove every cell $c_{vb}$ from its column list (lines 9-10). Each time a cell $c_{vb}$ is removed, we decrement $c_{vb}.head.size$ to ensure that it is equal to $\#cover_C(b)$.

Algorithm 8 undoes all cell removals performed by Algorithm 7. It performs the same traversals but in reverse order and restores cells instead of removing them: The column list associated with $a$ is traversed using *up* fields instead of *down* fields and row lists are traversed using *left* fields instead of *right* fields.

The time complexity of Algorithms 2 and 3 is $\mathcal{O}(n_p^2 \cdot n_c)$ as the number of cells in a row is bounded by $n_p$ and the number of cells in a column is bounded by $n_c$.

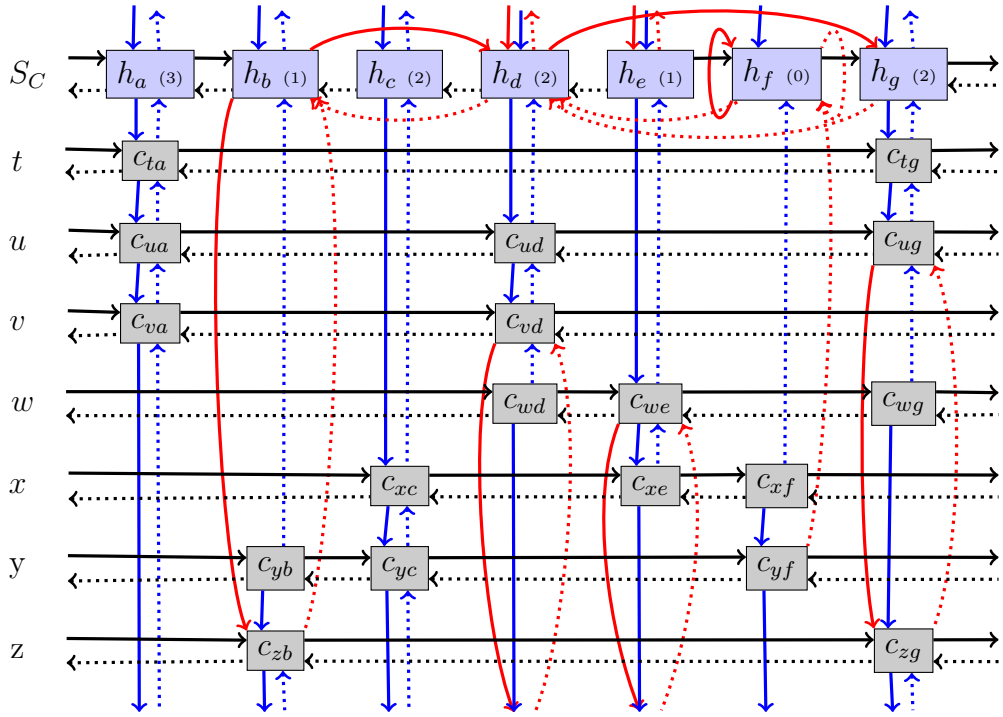



Figure 7.4 – Representation of the instance of Fig. 7.1 with Dancing Links when $C = \{x\}$. Links that have been modified are displayed in red.

**Example 7.5.** Let us consider the EC instance displayed in Figure 7.1, and let us assume that Algorithm 6 first chooses element $c$ (line 7) and recursively calls X with $C = \{x\}$. Before this recursive call, Algorithm 7 iterates on elements in $x$

- For element $c$, it removes cell $h_c$ from the first row and then successively removes from their columns cells $c_{xe}$, $c_{xf}$ (to remove subset $x$ ), and $c_{yf}$ and $c_{yb}$ (to remove subset $y$).
- For element $e$, it removes cell $h_e$ from the first row and then successively removes cells $c_{wg}$ and $c_{wd}$ from their columns (to remove subset $w$).
- For element $f$, it removes the cell $h_f$ from the headers.

The *size* fields of the headers of the columns in which cells have been removed are updated consequently. The resulting matrix is displayed in Figure 7.4.

After the recursive call to X, Algorithm 8 iterates on elements in $x$. For element $f$, it restores cell $h_f$ in the first row. For element $e$, it restores cell $h_e$ in the first row and then successively restores cells $c_{wd}$ and $c_{wg}$ in their columns. For element $c$, it restores cell $h_c$ in the first row and then successively restores cells $c_{yb}$, $c_{yf}$, and $c_{xf}$ and $c_{xe}$ in their columns.

We refer the reader to [Knu09] for more details on DLX. An open source implementation of DLX in C, called `libexact`, is described in [Kas+08].

## 7.4 Existing CP models to solve EC

Let us describe two different CP models that have been proposed to solve exact cover problems that occur when solving an instruction selection problem. In both models, for each element $a \in S$, an integer variable $coveredBy_a$ is used to decide which subset of $P$ covers $a$, and its domain is $D(coveredBy_a) = cover(a)$.

### 7.4.1 Boolean-based Model

A first model is described in [HB18], for solving an instruction selection problem. This model uses Boolean variables to model the selected subsets.

For each subset $u \in P$, a Boolean variable $isSelected_u$ indicates if $u$ is selected in the solution. *isSelected* variables are channeled with *coveredBy* variables by the following set of constraints:

$$\forall u \in P, \forall a \in u, coveredBy_a = u \Leftrightarrow isSelected_u.$$

When a variable $isSelected_u$ is assigned to *true*, enforcing AC on these constraints filters domains as follows:

- For every element $a \in u$, it removes from $D(coveredBy_a)$ every value different from $u$ (propagation of $isSelected_u = true \Rightarrow coveredBy_a = u$);
- For every subset $v \in P \setminus \{u\}$ such that $v \cap u \neq \emptyset$, it removes *true* from $D(isSelected_v)$ (because, for every element $b \in v \cap u$, $(isSelected_v, true)$ has no support on $coveredBy_b = u \Leftrightarrow isSelected_v = true$).
- For every element $a \in S \setminus \{u\}$, it removes from $D(coveredBy_a)$ every value $v$ such that $u \cap v \neq \emptyset$ (because $(coveredBy_a, v)$ has no support on $coveredBy_a = v \Leftrightarrow isSelected_v = true$).

**Example 7.6.** For example, let us assume that $D(isSelected_x) = \{true\}$ for the instance of Figure 7.1. In this case, ensuring AC filters domains as follows:

- All values different from $x$ are removed from $D(coveredBy[c])$, $D(coveredBy_e)$ and $D(coveredBy_f)$,
- $w$ and $y$ are removed from the domains of all *coveredBy* variables,
- *true* is removed from $D(isSelected_w)$ and $D(isSelected_y)$.

### 7.4.2 Gcc-based Model

A second model is described in [Flo+10], also for solving an instruction selection problem. This model uses a *gcc* constraint to model the problem.

In this model, an integer variable $nb_u$ is associated with every subset $u \in P$: It represents the number of times $u$ is assigned to a *coveredBy* variable and its domain is $D(nb_u) = \{0, \#u\}$. Indeed, the number of *coveredBy* variables assigned to $u$ must be either equal to 0 (if $u$ is not selected), or to $\#u$ (if $u$ is selected).

Given these variables, EC is modeled by a global cardinality constraint between *coveredBy* and *nb* variables: $gcc(coveredBy, P, nb)$.

Enforcing AC on this *gcc* global constraint is $\mathcal{NP}$-complete because domains of *nb* variables are not closed intervals (they only contain two values which are not successive integers). Therefore, CP solvers usually enforce weaker consistencies that may be different from a solver to another such as, for example, those ensured by the filtering algorithms described in [Qui+04; Nig11].

## 7.5 Existing SAT models to solve EC

In [Jun+10], Junttila and Kaski introduce SAT encodings for the exact cover problem. Given an instance $(S, P)$ of EC, they associate a Boolean variable $x_u$ with every subset $u \in P$, such that $x_u$ is assigned to true iff the subset $u$ is selected in the exact cover. The conjonctive normal form (CNF) formula associated with $(S, P)$ is

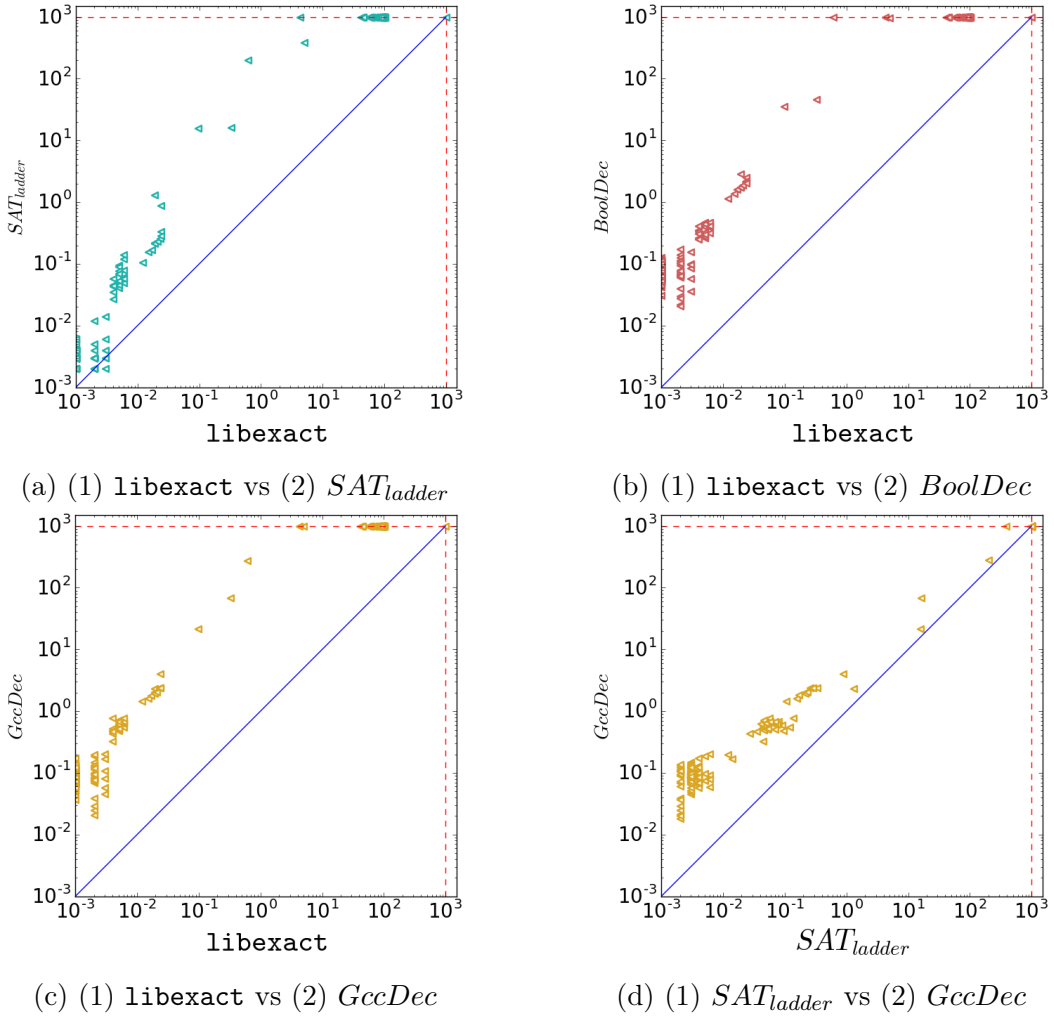
$$\bigwedge_{a \in S} \text{exactly-one}(\{x_u : u \in cover(a)\})$$

where exactly-one$(X)$ over a set $X$ of Boolean variables is a CNF formula which is satisfied by a complete truth assignment iff exactly one variable in $X$ is assigned to true. Junttila and Kaski describe three different encodings for exactly-one$(X)$. The first encoding is straightforward. For each element $a \in S$, it is composed of one $n$-ary clause and $(n^2 - n)/2$ binary clauses where $n = \#cover(a)$:

$$\text{exactly-one}(\{x_u : u \in cover(a)\}) = \bigvee_{u \in cover(a)} x_u \bigwedge_{u,v \in cover(a)} \neg x_u \vee \neg x_v$$

The two other encodings are less straightforward and use auxiliary variables to encode and exploit values of $x_u$ variables in order to require less clauses in the encoding. The `bitwise` encoding is composed of $n\lceil log_2 n \rceil$ binary clauses and one $n$-ary clause whereas the `ladder` encoding produces one unary clause, $3(n-1)$ binary clauses and one $n$-ary clause.

Several state-of-the-art SAT solvers, especially #SAT solvers, have been experimentally compared for enumerating all solutions of EC instances, for the three encodings. These experiments show that the `clasp` solver [Geb+12] has the best run time behavior among the DPLL-based approaches tested in [Jun+10], and is also very insensitive to the applied exactly-one encoding scheme. SAT solvers have also been compared with `libexact`, the C implementation of DLX [Kas+08], showing that SAT solvers explore smaller search space but do not perform that well in terms of running time: If SAT solvers are faster on some easy instances, they are often outperformed by `libexact` on harder instances.




(a) (1) `libexact` vs (2) $SAT_{ladder}$

(b) (1) `libexact` vs (2) *BoolDec*

(c) (1) `libexact` vs (2) *GccDec*

(d) (1) $SAT_{ladder}$ vs (2) *GccDec*

Figure 7.5 – Comparison of *GccDec*, *BoolDec* and *SAT* with `libexact` on benchmark instances of [Jun+10]: Each point (x,y) corresponds to an instance which is solved in $x$ seconds with (1) and $y$ seconds with (2). For SAT, we use `clasp` with ladder encoding.

## 7.6 Comparison of declarative approaches with DLX

In Figure 7.5a, we compare results obtained by `libexact` and `clasp` with the `ladder` encoding (denoted $SAT_{ladder}$) on the benchmark instances of [Jun+10][2]. We can see that $SAT_{ladder}$ is faster than `libexact` only for a few very easy instances. For harder instances, the gap between the two approaches increases in favor of `libexact` which is more than one order of magnitude faster for the hardest instances. These results are consistent with the results presented in [Jun+10].

We compare results obtained by `libexact` with the results of the boolean-based model and the Gcc-based model (denoted *BoolDec* and *GccDec*), in Figure 7.5b and Figure 7.5c, respectively. Both CP models are implemented with `Choco V4.3` [Pru+16]. They have similar results and cannot compete with `libexact` which is at least an order of magnitude faster for all instances. Figure 7.5d compares $SAT_{ladder}$ with *GccDec* and

[2]This benchmark is available at https://users.ics.aalto.fi/tjunttil/experiments/CP2010/

shows us $SAT_{ladder}$ is faster, but the gap between the two approaches decreases when instances become harder.

## 7.7 Discussion

We have described in this chapter the *Exact Cover* problem and some of its applications. Our interest for this problem comes from conceptual clustering, as selecting a subset of formal concepts that define a partition of the transactions is an instance of EC. We have described a simple backtracking algorithm introduced by Knuth to solve this problem, together with a backtrackable data structure, called *Dancing Links*, which is used to efficiently restore states when backtracking. We presented existing CP and SAT approaches to solve this problem, and experimentally showed that these declarative approaches are not competitive with a dedicated algorithm that uses Dancing Links. This motivates us for introducing a new global constraint dedicated to EC, together with an algorithm that uses Dancing Links to propagate this global constraint.

# Chapter 8

# Benchmark

## Contents



In Chapters 9 to 11, we introduce new CP models for solving conceptual clustering problems. These models are experimentally evaluated and compared with state-of-the-art approaches described in Chapters 6 and 7 on two sets of instances that we describe in this chapter: The first set of instances, described in Section 8.1, is coming from a classical benchmark for evaluating machine learning algorithms; The second set of instances, described in Section 8.2, is a new benchmark that we have extracted from our ERP configuration dataset.

## 8.1 Description of UCI instances

We describe in Table 8.1 six classical machine learning instances coming from the UCI database [Dhe+17] which have been used in other recent works on conceptual clustering [Gun15; Dao+15a; Oua+16; Ari+18]. These instances need to be preprocessed before searching for conceptual clusterings, to discretize continuous attributes, for example. We have considered instances preprocessed as in [Oua+16].

Some properties of these UCI instances are described in Table 8.1. The density is the percentage of ones in the database, *i.e.*, $100 * \frac{\#\mathcal{R}}{\#\mathcal{T}.\#\mathcal{I}}$. Typically, the higher the density, the larger the number of formal concepts. For example, UCI3 has less transactions and less items than UCI2, but it has a higher density and more than twice as more formal concepts.

The number of formal concepts varies from less than $5,000$ for UCI1 to more than $3,7$ millions for UCI6. The time spent by LCM to extract all formal concepts is closely related to this number, as expected for a linear-time complexity. It is smaller than one second for the five smallest instances, and close to 14 seconds for the largest instance, UCI6.

Let us recall that when solving conceptual clustering problems in two steps, by first computing all formal concepts and then solving an exact cover problem, the exact cover instance $(S, P)$ is defined by $S = \mathcal{T}$ and $P = \mathcal{F}$. Hence, instances with a very large number of formal concepts, such as UCI6, may be challenging.

| Name | $\#\mathcal{T}$ | $\#\mathcal{I}$ | d | $\#F$ | $t$ |
|---|---|---|---|---|---|
| UCI1 (zoo) | 101 | 36 | 44% | 4,567 | 0.01 |
| UCI2 (soybean) | 630 | 50 | 32% | 31,759 | 0.10 |
| UCI3 (primary-tumor) | 336 | 31 | 48% | 87,230 | 0.28 |
| UCI4 (lymph) | 148 | 68 | 40% | 154,220 | 0.52 |
| UCI5 (vote) | 435 | 48 | 48% | 227,031 | 0.68 |
| UCI6 (hepatitis) | 137 | 68 | 50% | 3,788,341 | 13.9 |

Table 8.1 – Description of UCI instances: For each instance, we display the number $\#\mathcal{T}$ of transactions, the number $\#\mathcal{I}$ of items, the density $d$, the number $\#\mathcal{F}$ of formal concepts, and the time $t$ (in seconds on an Intel(R) Core(TM) i7-6700 with 3.40GHz of CPU and 65GB of RAM) spent by LCM to extract all formal concepts.

| Name | $\#\mathcal{T}$ | $\#\mathcal{I}$ | d | $\#F$ | $t$ |
|---|---|---|---|---|---|
| ERP 1 | 50 | 27 | 48 | 1,580 | 0.01 |
| ERP 2 | 47 | 47 | 58 | 8,1337 | 0.03 |
| ERP 3 | 75 | 36 | 51 | 10,835 | 0.03 |
| ERP 4 | 84 | 42 | 45 | 14,305 | 0.05 |
| ERP 5 | 94 | 53 | 50 | 63,633 | 0.28 |
| ERP 6 | 95 | 61 | 47 | 71,918 | 0.45 |
| ERP 7 | 160 | 66 | 45 | 728,537 | 5.31 |

Table 8.2 – Description of ERP instances (see Table 8.1 for the meaning of the columns).

## 8.2 Description of ERP instances

We have used data coming from our ERP configuration database to generate a new benchmark for conceptual clustering. These instances have been randomly extracted from our database to evaluate scale-up properties of algorithms on a realistic dataset for our targeted application, *i.e.*, a dataset which has properties similar to real clustering problems we have to solve when searching for relevant configuration parts (which is not necessarily the case of UCI instances). However, these instances have no functional meaning, and clusterings extracted from them cannot be interpreted since parameters and configurations are randomly chosen. In Chapter 13, we consider more meaningful instances from an applicative point of view.

Each ERP instance has been built by randomly selecting a subset of parameters from the complete set of parameters, and a subset of configurations from the set of configurations that have a parameter instance for all the selected parameters. Then, each randomly selected subset of parameters/configurations is transformed into a transactional database as explained in Section 6.1. These instances are available at `http://liris.cnrs.fr/csolnon/ERP.html`, to allow other researchers to evaluate scale-up properties of their new algorithms on them.

We describe in Table 8.2 some properties of these instances. For these instances, the number of formal concepts varies from $1,580$ to $728,537$, and the time spent by LCM to extract them varies from 0.01 second to 5.31 seconds.

# Part III

# New CP Approaches for Conceptual Clustering

This part describes our main technical contributions, *i.e.*, new CP models and new global constraints for solving conceptual clustering problems that occur when searching for relevant configuration parts in existing *Copilote* configurations.

Existing CP models for solving conceptual clustering models do not scale well when the number of clusters $k$ is not fixed *a priori*. In our applicative context, the number of clusters correspond to the number of configuration parts, and we do not know this number. In Chapter 9, we introduce two new CP models that scale better when $k$ is not fixed. However, these CP models still struggle to solve all our instances when adding constraints on $k$ or when considering multi-criteria optimization problems, for example. This motivated us for introducing a new global constraint dedicated to exact cover problems. In Chapter 10, we introduce propagation algorithms for this constraint, and in Chapter 11, we extend our constraint to the case where the number of selected subsets is constrained. We evaluate our global constraint, and compare it with state-of-the-art declarative approaches, on different conceptual clustering problems in Chapter 12.

Finally, in Chapter 13, we describe the interactive mining tool that we have designed for extracting relevant configuration parts with CP and interactively integrating expert feedbacks after each mining process. We describe the different feedbacks we had to integrate after the first uses of this tool by a *Copilote* expert, and we show how we modified the CP model to integrate these feedbacks.

# Chapter 9

# New CP models

## Contents



We have described in Chapters 6 and 7 existing CP models for solving conceptual clustering and exact cover problems. In this chapter, we introduce two new CP models: The first CP model, described in Section 9.1, is dedicated to conceptual clustering problems; The second CP model, described in Section 9.2, is dedicated to the exact cover problem. These new CP models are compared with existing declarative approaches in Section 9.3.

## 9.1 New CP Model for Conceptual Clustering

In this section, we introduce a new CP model for computing conceptual clusterings. This model may be seen as an improvement of the CP model of [Dao+15b] (described in Section 6.4.2).

We do not assume that the number of clusters is fixed: We only assume that the number of clusters is bounded by two given bounds $k_{min}$ and $k_{max}$ such that $1 < k_{min} \leq k_{max} < \#\mathcal{T}$.

**Variables.** We use the following variables:

- An integer variable $k$ (with $D(k) = [k_{min}, k_{max}]$), which represents the number of clusters;
- For each transaction $t \in \mathcal{T}$:
  - An integer variable $G_t$ (with $D(G_t) = [1, k_{max}]$), which represents the cluster of $t$ like in the CP model of [Dao+15b];
  - A set variable $Intent_t$ (with $D(Intent_t) = [\emptyset, itemset(t)]$), which represents the set of items in the intent of the cluster of $t$;
- For each cluster $c \in [1, k_{max}]$, a set variable $Extent_c$ (with $D(Extent_c) = \mathcal{P}(\mathcal{T})$), which represents the set of transactions in $c$.

A first difference with the CP model of [Dao+15b] is that *Intent* set variables are associated with transactions instead of clusters. This simplifies the propagation of the intent constraint (as explained below). Another reason for associating *Intent* set variables with transactions instead of clusters is that $k$ is not fixed. In this case, it may happen that the variable $k$ has a value strictly lower than $k_{max}$ and, if we associate a variable $Intent_c$ with every possible cluster $c \in [1, k_{max}]$, then every $Intent_c$ variable associated with a cluster $c \in [k+1, k_{max}]$ is empty. In this case, the computation of the minimal intent size cannot be achieved by constraining a variable to be equal to the smallest cardinality of all $Intent_c$ variables, and we have to discard variables associated with empty clusters.

Also, we introduce new *Extent* set variables to explicitly model extents. These variables are associated with clusters because this allows us to easily channel them with $G_t$ variables. However, this implies that we have to discard every $Extent_c$ variable associated with an empty cluster $c \in [k+1, k_{max}]$ when computing the minimal extent frequency.

**Constraints.** We channel $Extent_c$ and $G_t$ variables by posting the constraint

$$\forall t \in \mathcal{T}, \forall c \in [1, k_{max}], t \in Extent_c \Leftrightarrow G_t = c$$

We reify $m(m-1)/2$ equality constraints between $G_t$ variables to ensure that two transactions are in a same cluster iff they have the same intent, and this intent is included in their itemsets: $\forall \{t_1, t_2\} \subseteq \mathcal{T}$

$$(G_{t_1} = G_{t_2}) \Leftrightarrow (Intent_{t_1} = Intent_{t_2}) \Leftrightarrow (Intent_{t_1} \subseteq itemSet(t_1) \cap itemSet(t_2))$$

This constraint ensures the extent property as any transaction $t_1$ such that $itemset(t1) \supseteq Intent_{t2}$ is constrained to be in the same cluster as $t_2$. However, this constraint only partially ensures the intent property: for each transaction $t$, it ensures $Intent_t \subseteq \cap_{t' \in \mathcal{T}, G_t = G_{t'}} itemset(t')$ whereas the intent property requires that $Intent_t$ is equal to the itemset intersection. However, given any solution that satisfies the constraint $Intent_t \subseteq \cap_{t' \in \mathcal{T}, G_t = G_{t'}} itemset(t')$, we can easily compute another solution that fully satisfies the intent property by adding to $Intent_t$ every item $i \in (\cap_{t' \in \mathcal{T}, G_t = G_{t'}} itemset(t')) \setminus Intent_t$. Hence, each time a solution is found, for each cluster $c$, we compute its actual intent by computing the intersection of all its transaction itemsets. This ensures that each cluster actually is a formal concept, and therefore this ensures correctness. Completeness is ensured by the fact that our constraint is a relaxation of the initial constraint.

As proposed in [Dao+15a; Dao+15b], we break symmetries (due to the fact that clusters may be swapped) by posting the global constraint:

$$precede(G, [1, k_{max}])$$

To constrain the number of clusters $k$, we add the constraint $k = \max_{t \in \mathcal{T}} G_t$ to ensure that $k$ is equal to the largest value assigned to $G$ variables.

**Objective function.** If the goal is to maximize $Min_{frequency}$, we introduce an integer variable $Min_{frequency}$ (with $D(Min_{frequency}) = [1, \#\mathcal{T} - 1]$) and post the constraints:

$$\forall c \in [1, k_{max}], Extent_c \neq \emptyset \Leftrightarrow Min_{frequency} \leq card(Extent_c)$$

If the goal is to maximize $Min_{size}$, we introduce an integer variable $Min_{size}$ (with $D(Min_{size}) = [1, \#\mathcal{I} - 1]$ and post the constraint:

$$Min_{size} = \min_{t \in \mathcal{T}} card(Intent_t)$$

When we optimize $Min_{split}$ or $Min_{-diameter}$, we dynamically add constraints each time a solution is found as proposed in [Dao+15a; Dao+15b]:

- If the goal is to maximize $Min_{split}$, we search for all solutions, and we dynamically add constraints each time a new solution is found. More precisely, when a new solution $I$ is found, we compute its associated minimum split value, *i.e.*,

  $$split = \min_{t,t' \in \mathcal{T}, I[G_t] \neq I[G_{t'}]} d(t, t')$$

  and for each pair of transactions $\{t, t'\} \subseteq \mathcal{T}$ such that $d(t, t') \leq split$, we add the constraint $G_t = G_{t'}$ to $C$. This constraint ensures that the next solution (if any) will have a larger split value.

- If the goal is to maximize $Min_{-diameter}$ (*i.e.*, minimize the maximal diameter value among all clusters), when a new solution $I$ is found, we compute its associated maximum diameter value, *i.e.*,

  $$diam = \max_{t,t' \in \mathcal{T}, I[G_t] = I[G_{t'}]} d(t, t')$$

  and for each pair of transactions $\{t, t'\} \subseteq \mathcal{T}$ such that $d(t, t') \geq diam$, we add the constraint $G_t \neq G_{t'}$. This constraint ensures that the next solution (if any) will have a smaller diameter value.

**Ordering heuristics.** The variable ordering heuristic first selects the variable $k$. The value ordering heuristic used for $k$ depends on the objective function: When the objective function tends to favor solutions with small values of $k$ (*i.e.*, $Min_{frequency}$ and $Min_{split}$), we first assign $k$ to its lower values; Otherwise, we first assign $k$ to its higher values.

Once $k$ has been assigned, the variable ordering heuristic selects $G$ variables, and it uses the *minDomain* heuristic for selecting first $G$ variables that have the smallest domains. We first assign $G$ variables to their lower values.

**Experimental evaluation.** All experiments have been done on an Intel(R) Core(TM) i7-6700 and 65GB of RAM. We consider the problem of finding a conceptual clustering that optimizes one single criterion (*i.e.*, maximizing $Min_{frequency}$, $Min_{size}$, $Min_{split}$ or $Min_{-diameter}$) when the number of clusters $k$ is fixed from 2 to 4, and then when $k$ is not fixed (in this case, we define $D(k) = [2, \#T - 1]$). Table 9.1 compares the approach of *Dao et al.* [Dao+15a] (denoted *FCP1*) implemented with `Gecode v4.3` [tea05] with our new CP model (denoted *FCP2*) implemented with `Choco v.4.0.3`[Pru+16].

When considering $Min_{size}$ and $Min_{frequency}$ criteria, *FCP1* is almost always faster for $k = 2$ and $k = 3$ whereas performances are degraded when $k = 4$: *FCP2* becomes more efficient for 10 instances and both approaches could not solve 10 instances. Increasing the number of clusters $k$ degrades performances for both approaches. When $k$ is not

| | *FCP1* | | | | *FCP2* | | | |
|---|---|---|---|---|---|---|---|---|
| | k=2 | k=3 | k=4 | N | k=2 | k=3 | k=4 | N |
| **Maximizing** $Min_{size}$ | | | | | | | | |
| ERP1 | **0.0** | **0.1** | **0.8** | **0.2** | 0.2 | 1.2 | 3.8 | 0.4 |
| ERP2 | **0.0** | **0.4** | 15.4 | 25.7 | 0.3 | 0.9 | **2.1** | **0.4** |
| ERP3 | **0.0** | **0.5** | 193.6 | 459.9 | 0.4 | 2.0 | **7.6** | **0.5** |
| ERP4 | **0.0** | **0.6** | 138.4 | 2.0 | 0.6 | 6.4 | **37.2** | **1.2** |
| ERP5 | **0.0** | **2.2** | - | - | 0.9 | 8.4 | **65.1** | **0.4** |
| ERP6 | **0.0** | **10.4** | - | 6.9 | 1.6 | 16.5 | **139.1** | **1.3** |
| ERP7 | **0.0** | 181.5 | - | 49.1 | 4.0 | **72.6** | - | **2.5** |
| UCI1 | **0.0** | **0.2** | **15.7** | 1.4 | 0.5 | 4.0 | 16.5 | **0.5** |
| UCI2 | **0.1** | **3.8** | **833.9** | - | 15.1 | 97.9 | - | **493.6** |
| UCI3 | **0.0** | **7.5** | - | 334.0 | 5.4 | 88.0 | - | **20.8** |
| UCI4 | **0.0** | **11.8** | - | 212.7 | 1.9 | 30.8 | **424.8** | **3.1** |
| UCI5 | **0.1** | 9.9 | - | - | - | **7.7** | **218.6** | **19.9** |
| UCI6 | **0.0** | **144.1** | - | 193.4 | 4.5 | 199.0 | - | **1.3** |
| **Maximizing** $Min_{split}$ | | | | | | | | |
| ERP1 | **0.0** | **0.0** | **0.0** | **0.0** | 0.1 | 0.1 | 0.1 | 0.2 |
| ERP2 | **0.0** | **0.0** | **0.0** | **0.0** | 0.1 | 0.1 | 0.2 | 0.2 |
| ERP3 | **0.0** | **0.0** | **0.0** | **0.0** | 0.2 | 0.2 | 0.2 | 0.3 |
| ERP4 | **0.0** | **0.0** | **0.0** | **0.0** | 0.1 | 0.1 | 0.2 | 0.3 |
| ERP5 | **0.0** | **0.0** | **0.0** | **0.1** | 0.2 | 0.3 | 0.3 | 0.4 |
| ERP6 | **0.0** | **0.0** | **0.0** | **0.1** | 0.3 | 0.3 | 0.3 | 0.5 |
| ERP7 | **0.0** | **0.0** | **0.0** | **0.2** | 0.7 | 0.9 | 0.8 | 1.5 |
| UCI1 | **0.0** | **0.0** | **0.0** | **0.1** | 0.3 | 0.3 | 0.3 | 0.7 |
| UCI2 | **0.1** | **0.1** | **0.1** | **11.0** | 4.7 | 6.6 | 6.2 | 15.7 |
| UCI3 | **0.0** | **0.0** | **0.0** | **1.7** | 1.3 | 1.6 | 1.7 | 4.2 |
| UCI4 | **0.0** | **0.0** | **0.0** | **0.2** | 0.4 | 0.5 | 0.6 | 0.5 |
| UCI5 | **0.0** | **0.1** | **0.1** | **4.6** | - | 3.7 | 18.7 | 9.6 |
| UCI6 | **0.0** | **0.0** | **0.0** | **0.2** | 0.6 | 0.6 | 0.8 | 1.0 |

| | *FCP1* | | | | *FCP2* | | | |
|---|---|---|---|---|---|---|---|---|
| | k=2 | k=3 | k=4 | N | k=2 | k=3 | k=4 | N |
| **Maximizing** $Min_{frequency}$ | | | | | | | | |
| ERP1 | 0.3 | **0.0** | **0.3** | **0.2** | **0.2** | 1.0 | 14.4 | 0.3 |
| ERP2 | **0.0** | **0.3** | **3.6** | **0.3** | 0.3 | 0.7 | 18.2 | 0.4 |
| ERP3 | **0.0** | **0.2** | 15.6 | 1.4 | 0.5 | 2.0 | **10.5** | **0.9** |
| ERP4 | **0.0** | **0.3** | **29.3** | 1.2 | 0.7 | 3.2 | 124.7 | **1.0** |
| ERP5 | **0.0** | **1.3** | 440.6 | 72.5 | 1.0 | 6.0 | **150.3** | **1.5** |
| ERP6 | **0.0** | 7.7 | 613.1 | 47.6 | 1.3 | **4.1** | **343.8** | **1.7** |
| ERP7 | **0.1** | 78.8 | - | 952.7 | 4.6 | **64.8** | - | **6.5** |
| UCI1 | **0.0** | **0.2** | **7.6** | **1.1** | 1.1 | 4.9 | 188.4 | 1.6 |
| UCI2 | **0.1** | **1.2** | **28.1** | 211.0 | 122.8 | 188.5 | - | **192.1** |
| UCI3 | **0.0** | **2.0** | **250.6** | 530.0 | 23.4 | 103.0 | - | **33.5** |
| UCI4 | **0.0** | **1.3** | **296.5** | 101.1 | 3.1 | 42.1 | - | **4.5** |
| UCI5 | **0.1** | **8.7** | - | - | - | 32.2 | - | - |
| UCI6 | **0.0** | **11.6** | - | 38.9 | 2.3 | 412.6 | - | **3.1** |
| **Maximizing** $Min_{-diameter}$ | | | | | | | | |
| ERP1 | **0.0** | **0.0** | **0.0** | **0.0** | 0.1 | 0.1 | 0.1 | 0.2 |
| ERP2 | **0.0** | **0.0** | **0.0** | **0.0** | 0.2 | 0.3 | 0.6 | 0.2 |
| ERP3 | **0.0** | **0.0** | **0.0** | **0.0** | 0.2 | 0.3 | 0.5 | 0.3 |
| ERP4 | **0.0** | **0.0** | **0.0** | **0.0** | 0.2 | 0.4 | 1.3 | 1.1 |
| ERP5 | **0.0** | **0.0** | **0.0** | **0.1** | 0.2 | 0.4 | 1.4 | 0.4 |
| ERP6 | **0.0** | **0.0** | **0.0** | **0.1** | 0.5 | 0.9 | 2.9 | 1.2 |
| ERP7 | **0.0** | **0.0** | **0.0** | **0.2** | 0.9 | 1.5 | 2.7 | 1.0 |
| UCI1 | **0.0** | **0.0** | **0.0** | **0.1** | 0.3 | 0.4 | 0.6 | 0.4 |
| UCI2 | **0.1** | **0.1** | **0.1** | **11.3** | 7.3 | 9.6 | 37.4 | 446.5 |
| UCI3 | **0.0** | **0.0** | **0.0** | **1.7** | 1.7 | 9.3 | 84.9 | 13.9 |
| UCI4 | **0.0** | **0.0** | **0.0** | **0.2** | 0.6 | 1.3 | 2.8 | 3.5 |
| UCI5 | **0.0** | **0.1** | **0.1** | **4.5** | - | 2.0 | 69.8 | 18.6 |
| UCI6 | **0.0** | **0.0** | **0.0** | **0.2** | 1.1 | 7.4 | 86.7 | 1.8 |

Table 9.1 – Comparison of *FCP1* and *FCP2* when the goal is to optimize $Min_{size}$, $Min_{frequency}$ (upper part) and $Min_{split}$, $Min_{-diameter}$ (lower part): For each approach, we display the time (in seconds) when $k$ is fixed to 2, 3, and 4, respectively, and when $k$ is not fixed (N). '-' is reported when time exceeds 1,000s.

fixed, *FCP2* is faster than *FCP1* except for the easiest instances ERP1, ERP2 and UCI1. *FCP2* is the only approach that can solve all the instances for $Min_{size}$ criterion.

Optimizing $Min_{split}$ and $Min_{-diameter}$ is much easier since both models are able to solve almost all instances. *FCP1* is always faster than *FCP2* whether $k$ is fixed or not: It is able to solve all instances in less than 0.2 seconds (resp. 12 seconds), when $k$ is fixed (resp. $k$ is not fixed). However, *FCP2* scales rather well and is able to solve all instances but UCI5 when $k = 2$. Increasing the number of clusters $k$ does not change the time of *FCP1* whereas *FCP2* needs more than one minute when $k = 4$ for $Min_{-diameter}$ on UCI3, UCI5 and UCI6.

Both approaches are rather complementary and only seven instances could not be solved when considering both approaches.

## 9.2 New CP Model for the Exact Cover Problem

We introduce a new CP model for the exact cover problem, and this CP model may be used to solve conceptual clustering problems when combining it with LCM [Uno+04] to extract the set of all formal concepts from a set of transactions before searching for a subset of formal concepts that is a partition of the transactions.

**Variables.** Let $(S, P)$ be an instance of EC. Like the Boolean-based and the *gcc*-based models introduced in Section 7.4, for each element $a \in \mathcal{S}$, an integer variable $coveredBy_a$

is used to represent the selected subset that covers $a$ (with $D(coveredBy_a) = cover(a)$).

Besides these variables, we use a set variable $C$ (with $D(C) = [\emptyset, P]$), which represents the set of selected subsets that define an exact cover of S.

**Constraints.** For each element $a \in S$, we ensure that the selected subset that covers $a$ belongs to the solution by adding a member constraint $coveredBy_a \in C$.

We ensure that each element $a \in S$ is covered by exactly one selected subset by adding the constraint $\#(C \cap cover(a)) = 1$. This way, each time a subset $u$ is added to $C$, all the subsets $v$ such that $v \cap u \neq \emptyset$ are removed from the upper bound of $C$, and cannot be chosen anymore.

**Constraining the number of subsets of the solution.** To constrain the number of selected subsets (corresponding to the number of clusters when solving a conceptual clustering problem) to be bounded between two given bounds $k_{min}$ and $k_{max}$, we add an integer variable $k$ (with $D(k) = [k_{min}, k_{max}]$), which represents the number of selected subsets.

The way we constrain $k$ depends on the utility measure: When the utility measure tends to favor solutions with small values of $k$ (*i.e.*, $u = frequency$ or $u = split$), we use $NValue(G, k)$, otherwise we use $\#C = k$. Indeed, we experimentally observed that *NValue* is efficient when $k$ is small, but does not scale well when $k$ is large.

**Objective function.** When the EC instance $(S, P)$ corresponds to the selection step of a conceptual clustering problem (*i.e.*, $S$ is a set of transactions and $P$ a set of formal concepts), the objective function to maximize is the minimum utility measure associated with a selected subset. Let $u : P \to \mathbb{R}$ be this utility measure (as defined in Section 6.3). The objective function to maximize is

$$\min_{c \in C} u(c).$$

**Ordering heuristics.** We use the *minDomain* heuristic for selecting first *coveredBy* variables that have the smallest domains. As the goal is to maximize the minimum utility of a selected subset, we use a value ordering heuristic that selects first elements of $S$ that have a high utility, for all utility measures but $Min_{frequency}$.

**Special case for $Min_{frequency}$.** The frequency utility measure has particular properties that may be used to speed up the solution process when searching for an exact cover that maximizes $Min_{frequency}$. Indeed, an exact cover forms a partition of the set of elements $S$ (corresponding to transactions), and each subset $u \in P$ corresponds to a formal concept whose frequency is equal to $\#u$. Therefore, for any exact cover, the sum of the cardinalities of its subsets is equal to $\#S$.

More formally, let $C = \{u_1, \ldots, u_k\}$ be an exact cover such that $k > 1$, and let $m = \min_{i \in [1,k]} \#u_i$ and $M = \max_{i \in [1,k]} \#u_i$ be the minimum and maximum size of the subsets in $C$, respectively. The first property is:

$$m + M \leq \#S.$$

Indeed, let $u_{min}$ and $u_{max}$ be two subsets of $C$ such that $m = \#u_{min}$, $M = \#u_{max}$, and $u_{min} \neq u_{max}$. We have:

$$\begin{aligned} \#\mathcal{S} &= \sum_{u_i \in C} \#u_i \text{ (because } C \text{ defines a partition of } S) \\ &= m + M + \sum_{u_i \in C \setminus \{u_{min}, u_{max}\}} \#u_i \end{aligned}$$

As $\sum_{u_i \in C \setminus \{u_{min}, u_{max}\}} \#u_i \geq 0$, we have $m + M \leq \#\mathcal{S}$.

The second property is:

$$m * k \leq \#S.$$

Indeed, for all $u_i \in S$, we have $m \leq \#u_i$. Therefore, $\#\mathcal{S} = \sum_{c_i \in S} \#u_i \geq k * m$.

When we maximize $Min_{frequency}$, we take advantage of these two properties by adding two integer variables $Min_{frequency}$ and $Max_{frequency}$ that represent the minimal and maximal frequency of the selected formal concepts. We constrain these variables to be equal to the minimal and maximal frequencies by posting the constraints:

$$\begin{aligned} Min_{frequency} &= \min_{c \in C} frequency(c) \\ Max_{frequency} &= \max_{c \in C} frequency(c) \end{aligned}$$

We add constraints corresponding to the two properties:

- (C1) $Min_{frequency} \leq \#S - Max_{frequency}$
- (C2) $Min_{frequency} * k \leq \#S$

Furthermore, we deduce from the property $m * k \leq \#\mathcal{S}$ that the upper bound of $Min_{frequency}$ is $\lceil \frac{\#\mathcal{T}}{K.lb} \rceil$. However, when we maximize $Min_{frequency}$, our ordering heuristic favors the choice of formal concepts with the highest frequency that inevitably leads to conceptual clusterings with a low minimal frequency. To prevent this, we use the *ObjectiveStrategy* proposed by Choco [Pru+16], which performs a dichotomous branching over the domain of $Min_{frequency}$. Each time a solution is found, the *ObjectiveStrategy* first searches for a next solution such that:
$Min_{frequency} \in [\frac{Min_{frequency}.ub - Min_{frequency}.lb}{2}, Min_{frequency}.ub]$.

If it fails, it searches for a solution such that:
$Min_{frequency} \in [Min_{frequency}.lb, \frac{Min_{frequency}.ub - Min_{frequency}.lb}{2} - 1]$.

## 9.3 Experimental evaluation

We consider the problem of finding a conceptual clustering that optimizes one single criterion when the number of clusters $k$ is fixed from 2 to 4 and when $k$ is bounded between 2 and $\#T - 1$. We solve this problem in two steps: First, we use LCM to compute all formal concepts; Second, we solve an exact cover problem. We consider three CP models for the second step, *i.e.*, our new model denoted *SetDec*, the boolean-based model denoted *BoolDec*, and the Gcc-based model denoted *GccDec*. These three CP models are implemented with `Choco v.4.0.3`. We also consider the ILP approach of *Ouali et al.* [Oua+16], implemented with `CPLEX v12.7` and denoted *ILP*.

| | *BoolDec* | | | | *GccDec* | | | | *SetDec* | | | | *ILP* | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | k=2 | k=3 | k=4 | N | k=2 | k=3 | k=4 | N | k=2 | k=3 | k=4 | N | k=2 | k=3 | k=4 | N |
| **Maximize** $Min_{size}$ | | | | | | | | | | | | | | | | |
| ERP1 | 23.1 | 252.7 | - | 0.2 | 4.9 | 64.3 | - | 0.4 | 2.2 | 27.7 | 230.0 | **0.2** | **0.1** | **0.3** | **0.6** | 0.3 |
| ERP2 | - | - | - | 4.1 | 73.3 | - | - | 0.4 | 26.4 | 105.3 | 523.6 | **0.3** | **1.2** | **1.2** | **1.4** | 0.8 |
| ERP3 | - | - | - | 3.3 | 173.2 | - | - | 0.7 | 66.7 | 814.1 | - | **0.3** | **1.5** | **1.5** | **1.5** | 2.2 |
| ERP4 | - | - | - | 25.0 | 382.3 | - | - | 2.0 | 198.5 | - | - | **0.8** | **6.3** | **15.8** | **12.6** | 20.7 |
| ERP5 | - | - | - | 327.3 | - | - | - | 4.4 | - | - | - | **1.7** | **11.9** | **34.7** | **25.9** | 27.4 |
| ERP6 | - | - | - | - | - | - | - | 12.3 | - | - | - | **4.5** | **63.6** | **225.1** | **387.5** | - |
| ERP7 | - | - | - | - | - | - | - | 411.9 | - | - | - | **129.5** | - | - | - | - |
| UCI1 | - | - | - | 1.1 | 204.8 | - | - | 0.6 | 81.4 | - | - | 0.2 | **0.0** | **0.0** | **0.0** | **0.1** |
| UCI2 | - | - | - | 18.2 | - | - | - | 154.4 | - | - | - | **5.0** | **39.1** | **104.2** | **302.0** | 23.1 |
| UCI3 | - | - | - | - | - | - | - | 81.7 | - | - | - | **4.6** | - | - | - | 39.6 |
| UCI4 | - | - | - | - | - | - | - | 70.2 | - | - | - | **32.9** | **216.9** | **297.5** | - | - |
| UCI5 | - | - | - | - | - | - | - | - | - | - | - | **15.1** | **106.7** | **269.4** | **174.6** | 129.2 |
| UCI6 | - | - | - | - | - | - | - | - | - | - | - | **111.3** | - | - | - | - |
| **Maximize** $Min_{frequency}$ | | | | | | | | | | | | | | | | |
| ERP1 | 0.2 | 0.6 | 1.3 | 0.3 | **0.1** | 0.5 | 1.5 | **0.2** | 0.2 | 0.7 | 1.9 | 0.2 | 0.2 | **0.5** | **0.6** | 0.2 |
| ERP2 | 1.4 | 11.5 | 32.1 | 2.0 | **0.6** | 5.0 | 11.7 | 0.9 | 0.8 | 4.3 | 10.0 | **0.6** | 1.2 | **1.5** | **1.4** | 1.3 |
| ERP3 | 5.7 | 40.9 | 31.1 | 5.4 | 2.2 | 20.7 | 12.2 | 2.8 | 2.5 | 17.6 | 10.4 | 2.5 | **1.5** | **1.4** | **2.0** | **2.2** |
| ERP4 | 1.7 | 203.7 | - | 2.0 | **0.6** | 51.0 | 223.9 | 0.9 | 0.9 | 42.4 | 187.8 | **0.9** | 6.0 | **8.8** | **14.4** | 27.5 |
| ERP5 | 250.1 | - | - | 255.9 | 24.9 | 430.3 | - | 27.8 | 24.2 | 819.9 | - | **26.0** | **16.0** | **18.4** | **33.5** | 26.1 |
| ERP6 | - | - | - | - | 365.2 | - | - | 246.8 | 223.9 | - | - | **241.4** | **111.8** | **160.2** | **129.5** | 302.1 |
| ERP7 | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |
| UCI1 | 1.3 | 8.8 | 46.0 | 1.8 | **0.7** | 5.8 | 32.3 | 1.3 | 1.5 | 5.8 | 34.0 | **1.0** | 0.9 | **2.4** | **3.0** | 3.7 |
| UCI2 | 34.7 | 271.0 | - | 36.0 | **18.9** | 261.6 | - | **24.6** | 41.5 | 453.5 | - | 29.8 | 27.5 | **155.0** | **315.6** | - |
| UCI3 | 225.0 | 459.8 | - | 112.1 | **33.1** | **235.6** | - | **55.5** | 59.1 | 351.0 | - | 63.7 | 83.8 | - | - | - |
| UCI4 | 261.2 | - | - | 24.1 | **13.9** | - | - | **14.7** | 20.4 | **667.5** | - | 19.1 | 339.3 | 737.4 | - | - |
| UCI5 | - | - | - | - | - | **58.5** | - | 80.2 | - | 127.8 | - | **76.6** | **75.2** | 350.2 | **952.1** | - |
| UCI6 | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |
| **Maximize** $Min_{-diameter}$ | | | | | | | | | | | | | | | | |
| ERP1 | 23.0 | 189.4 | - | **0.2** | 4.9 | 47.6 | 173.9 | 0.4 | 2.0 | 25.8 | 91.7 | 0.2 | **0.2** | **0.5** | **0.7** | 0.7 |
| ERP2 | - | - | - | 3.1 | 57.7 | - | - | 0.7 | 17.2 | 141.8 | - | **0.3** | **1.0** | **1.2** | **1.5** | 0.8 |
| ERP3 | - | - | - | 2.7 | 142.7 | - | - | **0.1** | 65.3 | 725.5 | - | 0.3 | **1.5** | **1.6** | **1.6** | 1.8 |
| ERP4 | - | - | - | 18.6 | 346.6 | - | - | 2.1 | 140.7 | - | - | **0.8** | **6.0** | **8.5** | **9.9** | 19.0 |
| ERP5 | - | - | - | 241.3 | - | - | - | 4.2 | - | - | - | **1.6** | **15.6** | **19.6** | **43.2** | 22.9 |
| ERP6 | - | - | - | - | - | - | - | 12.0 | - | - | - | **4.3** | **127.8** | **80.3** | **164.1** | - |
| ERP7 | - | - | - | - | - | - | - | 413.9 | - | - | - | **125.6** | - | - | - | - |
| UCI1 | - | - | - | 1.3 | 187.0 | - | - | 0.6 | 79.0 | - | - | **0.2** | **1.3** | **2.0** | **3.0** | 1.1 |
| UCI2 | - | - | - | 20.6 | - | - | - | 125.8 | - | - | - | **4.6** | **26.8** | **156.4** | - | 20.2 |
| UCI3 | - | - | - | 444.7 | - | - | - | 58.9 | - | - | - | **4.5** | - | - | - | 45.3 |
| UCI4 | - | - | - | - | - | - | - | 58.7 | - | - | - | **30.8** | **188.1** | - | - | - |
| UCI5 | - | - | - | - | - | - | - | 485.6 | - | - | - | **14.0** | **90.1** | **221.3** | - | 176.9 |
| UCI6 | - | - | - | - | - | - | - | - | - | - | - | **99.6** | - | - | - | - |
| **Maximize** $Min_{split}$ | | | | | | | | | | | | | | | | |
| ERP1 | 4.0 | 2.7 | 5.2 | **0.1** | 1.3 | 1.1 | 2.3 | **0.1** | 0.6 | 0.4 | 1.0 | 0.1 | **0.2** | **0.1** | **0.2** | 0.2 |
| ERP2 | 0.3 | 0.3 | 0.4 | 0.3 | **0.2** | **0.2** | 0.3 | 0.3 | 0.2 | 0.3 | **0.3** | **0.2** | 1.0 | 0.7 | 0.7 | 0.5 |
| ERP3 | 0.3 | 0.3 | 0.4 | 0.4 | **0.3** | **0.2** | **0.3** | **0.4** | 0.4 | 0.3 | 0.4 | **0.4** | 1.5 | 1.0 | 1.2 | 1.7 |
| ERP4 | 0.3 | 0.7 | 1.0 | 0.4 | **0.2** | **0.3** | **0.6** | **0.3** | 0.4 | 0.5 | 0.6 | 0.4 | 5.6 | 5.2 | 4.0 | 2.0 |
| ERP5 | - | - | - | 17.4 | - | - | - | **1.9** | 464.7 | **2.0** | **1.9** | 2.0 | **15.3** | 9.4 | 9.3 | 13.0 |
| ERP6 | - | - | - | 432.0 | - | - | - | **3.8** | 316.5 | - | - | 14.1 | **49.6** | **24.4** | **96.5** | 164.3 |
| ERP7 | 67.1 | 121.3 | 140.9 | 139.3 | 65.3 | 71.0 | 72.0 | 65.3 | **27.6** | **28.6** | **28.5** | **22.2** | - | - | - | - |
| UCI1 | 0.3 | 0.4 | 0.3 | 0.3 | **0.1** | **0.1** | 0.2 | **0.2** | 0.2 | 0.2 | **0.2** | 0.2 | 1.3 | 0.6 | 0.8 | 1.0 |
| UCI2 | - | - | - | 3.4 | 200.4 | **1.0** | **1.5** | **1.4** | 103.4 | 3.6 | 4.8 | 3.5 | **38.8** | 169.1 | 229.7 | 25.8 |
| UCI3 | - | - | - | 129.8 | - | - | - | **8.6** | - | - | - | 319.7 | - | - | - | 933.0 |
| UCI4 | 18.4 | 21.8 | 21.8 | 22.6 | 3.3 | **4.8** | **4.8** | 4.8 | **3.2** | 6.2 | 6.3 | **3.2** | 188.7 | 58.5 | 24.9 | 88.9 |
| UCI5 | - | - | - | - | - | - | - | - | - | - | - | - | **104.5** | **266.1** | **269.8** | - |
| UCI6 | - | - | - | - | - | - | - | - | - | - | - | **192.6** | - | - | - | - |

Table 9.2 – Comparison of *BoolDec*, *GccDec*, *SetDec* and *ILP* when the goal is to optimize $Min_{size}$, $Min_{frequency}$, $Min_{-diameter}$, and $Min_{split}$: Total time in seconds (including the time spent by LCM) when $k$ is fixed to 2, 3, and 4, respectively, and when $k$ is not fixed (N). '-' is reported when time exceeds 1,000s.

We compare in Table 9.1 results obtained with these four approaches (we report the total time, that includes the time spent by LCM to compute all formal concepts). When $k$ is fixed, *ILP* solves more instances than CP approaches and is most of the

| | ERP | | | | | | | UCI | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 1 | 2 | 3 | 4 | 5 | 6 |
| #S | 50 | 47 | 75 | 84 | 94 | 95 | 160 | 101 | 630 | 336 | 148 | 435 | 137 |
| $Min_{size}$ | 49 | 45 | 63 | 83 | 85 | 94 | 159 | 59 | 501 | 215 | 147 | 342 | 136 |
| $Min_{-diameter}$ | 49 | 45 | 63 | 83 | 85 | 94 | 159 | 59 | 501 | 215 | 147 | 342 | 136 |
| $Min_{frequency}$ | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 3 | 2 |
| $Min_{split}$ | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 3 | 2 |

Table 9.3 – Number of clusters of optimal solutions when maximizing $Min_{size}$, $Min_{-diameter}$, $Min_{frequency}$ and $Min_{split}$.

time the fastest approach. *BoolDec* is almost always the slowest approach: For $Min_{size}$ and $Min_{-diameter}$ criteria, it is able to solve only ERP1 for $k < 4$ . *SetDec* solves more instances than *GccDec* and it is always faster for $Min_{size}$ and $Min_{-diameter}$ criteria. For instance, *SetDec* solves $ERP2$ in 142 seconds for $Min_{-diameter}$ and $k = 3$ while *GccDec* is not able to solve it in less than 1000 seconds.

When $k$ is not fixed, *BoolDec* is still the slowest CP approach: The use of many boolean variables seems to make the propagation very slow when solving big instances such that ERP7 or UCI6. *SetDec* is the fastest approach for most instances, particularly when maximizing $Min_{frequency}$ and $Min_{-diameter}$. *SetDec* scales generally better than other approaches for the hardest instances ERP7 and UCI6. For instance, it is the only approach able to solve UCI6 and ERP7 for all criteria but $Min_{frequency}$. *ILP* approach does not scale well for large instances: It is able to solve ERP6 only for $Min_{frequency}$ and $Min_{split}$.

We report the number of clusters of the optimal solutions when optimizing each criterion in Table 9.3. When we optimize $Min_{size}$ or $Min_{-diameter}$, optimal solutions have a large number of clusters (very close to $\mathcal{T} - 1$), whereas when we optimize $Min_{frequency}$ or $Min_{split}$, optimal solutions have very few clusters (very close to 2). As explained in Section 6.2, these results confirm that $Min_{frequency}$ and $Min_{split}$ criteria are conflicting with $Min_{size}$ and $Min_{-diameter}$ criteria.

## 9.4 Discussion

We have proposed two new CP models to solve conceptual clustering problems: The model introduced in section 9.1 may be seen as an extension of the model of *Dao et al.* [Dao+15a] whereas the model introduced in Section 9.2 is dedicated to the exact cover problem and it may be used to solve conceptual clustering problems provided that we first precompute all possible formal concepts. When considering the problem of finding a conceptual clustering that optimizes one single criterion with a number of clusters fixed, *ILP* and *FCP1* are the best approaches: *FCP1* is most of the time the fastest approach when $k = 2$ or $k = 3$ while its performances are degraded when $k = 4$ and, in this case, $ILP$ is usually faster.

However, when $k$ is not fixed, *SetDec* and *FCP2* are the best approaches when maximizing $Min_{frequency}$ and $Min_{size}$ while *FCP1* is the fastest approach when maximizing $Min_{-diameter}$ and $Min_{split}$.

When we observe the number of clusters of the solutions, we find out that $Min_{frequency}$ and $Min_{split}$ criteria are conflicting with $Min_{size}$ and $Min_{-diameter}$ criteria. To find a compromise between these complementary criteria, we may compute the Pareto front of non-dominated solutions. We have tried to compute Pareto fronts with these different models. However, *FCP1* and *FCP2* are not able to solve any instance within a reasonable amount of time. *SetDec* and *ILP* are able to solve small instances, but they do not scale well and cannot be used to solve larger instances. This motivated us to introduce new global constraints to solve conceptual clusterings more efficiently in Chapters 10 and 11.

# Chapter 10

# ExactCover global constraint

## Contents



As proposed in the ILP approach of Ouali *et al.* [Oua+16] described in Section 6.4.3, we may solve conceptual clustering problems by first extracting all formal concepts from the set of transactions with a dedicated tool such as LCM [Uno+04] and then solving an exact cover problem. In this chapter, we focus on the resolution of the exact cover problem. CP provides a framework that allows to embed dedicated propagators into global constraints to speed-up the solution process. Therefore, we propose to embed into a new global constraint called *exactCover* an efficient propagation algorithm which uses the *Dancing Links* introduced by Knuth [Knu09] and described in Section 7.3.

We define the *exactCover* constraint in Section 10.1. Then, we propose three filtering algorithms that ensure different level of consistency, *i.e.*, a basic propagator in Section 10.2, a propagator based on the dancing links in Section 10.3, and an advanced propagator in Section 10.4. In Section 10.5, we experimentally compare our filtering algorithms with state-of-the-art declarative approaches. Finally, in Section 10.6, we extend our global constraint to handle the case where costs are associated with subsets.

## 10.1 Definition of *exactCover*

Let $(S, P)$ be an instance of EC and, for each subset $u \in P$, let $isSelected_u$ be a Boolean variable. The global constraint $exactCover_{S,P}(isSelected)$ is satisfied iff all *isSelected* variables assigned to *true* correspond to an exact cover of $S$, *i.e.*,

$$\forall a \in S, \#\{u \in cover(a) : isSelected_u\} = 1.$$

**Example 10.1.** Let us consider the instance $(S, P)$ displayed in Fig. 7.1. Let $I$ be the instantiation that assigns $isSelected_v$, $isSelected_x$, and $isSelected_z$ to *true*, and all other *isSelected* variables to *false*. The instantiation $I$ satisfies $exactCover_{S,P}(isSelected)$.

Enforcing AC on *exactCover* is $\mathcal{NP}$-complete since the exact cover problem is $\mathcal{NP}$-complete.

We have described in Section 7.4 two CP models for the exact cover problem, and we have also introduced in Section 9.2 a new CP model. Each of these models may be used to define a decomposition of *exactCover*, showing us that it is not semantically global. We have shown in Section 9.3 that the best performing model is the new model introduced in Section 9.2. This model will be referred to as the set decomposition (*SetDec*) of *exactCover*, and it will be used as a baseline to evaluate the interest of introducing a global constraint.

## 10.2 *Basic* propagator

Let us first introduce a basic propagator which ensures the same level of consistency as AC on the set decomposition of *exactCover* without using any specific data structure. This basic propagator is used as a baseline to evaluate the interest of using Dancing Links.

To simplify notations, we denote $C$ the set of subsets associated with *isSelected* variables which are assigned to *true*, *i.e.*,

$$C = \{u \in P | D(isSelected_u) = \{true\}\}$$

and we use notations introduced in Chapter 7. In particular, $S_C$ denotes the set of elements that are not covered yet, and $cover_C(a)$ the set of subsets that may be selected to cover an element $a \in S_C$.

To ensure the same level of filtering as AC on the set decomposition, we have to ensure the following property: for each subset $u \in P$ such that $D(isSelected_u) = \{true\}$, and for each subset $v \in incompatible(u)$, we have $true \notin D(isSelected_v)$. In other words, each time a variable $isSelected_u$ is assigned to *true*, we have to assign to *false* every variable $isSelected_v$ such that $v \in incompatible(u)$. Also, for each element $a \in S_C$, we have to check that there is at least one subset $u \in cover_C(a)$ such that $true \in D(isSelected_u)$.

To this aim, for each subset $u \in P$, we compute the set $incompatible(u)$ of all subsets of $P$ that are not compatible with $u$. These incompatibility sets are computed before starting the search process in $\mathcal{O}(\#P^2 \cdot n_p)$.

Then, during the search, when a variable $isSelected_u$ is assigned to *true*, for each subset $v \in incompatible(u)$, we assign $isSelected_v$ to false. This is done in linear time with respect to the size of the largest incompatibility set, *i.e.*, $\mathcal{O}(n_i)$.

Also, to ensure that each element $a \in S_C$ can be covered by at least one subset compatible with the selected subsets, we incrementally maintain the cardinality of $cover_C(a)$ (without explicitly maintaining $cover_C(a)$). Initially, this cardinality is set to the cardinality of $cover(a)$, *i.e.*, we initialize $\#cover_C(a)$ to $\#cover(a)$. Then, each time a variable $isSelected_v$ is assigned to *false*, we decrement $\#cover_C(a)$ for each element $a \in v$, and we trigger a failure if $\#cover_C(a) = 0$. As the number of *isSelected* variables that are assigned to *false* is bounded by $n_i$, the complexity of the basic propagation algorithm is $\mathcal{O}(n_p \cdot n_i)$.

When backtracking, we restore $\#cover_C$ counters by performing the inverse operations. This is done in $\mathcal{O}(n_p \cdot n_i)$.

**Example 10.2.** For example, let us consider the instance displayed in Fig. 7.1. $\#cover_C$ counters are initialized to 2 for $b$, $c$, $e$, and $f$, to 3 for $a$ and $d$, and to 4 for $g$. Now, let us assume that $isSelected_x$ is assigned to *true*. The incompatibility set of $x$ is $incompatible(x) = \{w, y\}$. Hence, we assign $isSelected_w$ and $isSelected_y$ to *false* and we decrement $\#cover_C$ counters associated with elements of $w$ and $y$, *i.e.*, $d$, $e$ and $g$ for $w$, and $b$, $e$, and $f$ for $y$. We obtain: $\#cover_C(b) = \#cover_C(c) = \#cover_C(e) = \#cover_C(f) = 1$, and $\#cover_C(g) = \#cover_C(e) = 3$.

This propagator is called *Basic*, and the Choco implementation of *exactCover* with this propagator is denoted $EC_{Basic}$.

## 10.3 $DL$ Propagator

In the basic propagation algorithm, incompatibility lists are not incrementally maintained during the search: When a variable $isSelected_v$ is assigned to *false*, $v$ is not removed from other incompatibility lists. On our previous example, $u$ and $w$ both belong to $incompatible(v)$ and $incompatible(z)$. As a consequence, if $v$ and $z$ are successively selected, when propagating the assignment of $isSelected_z$, we consider again subsets $u$ and $w$.

To improve this, we propose to incrementally maintain $cover_C(a)$ for each element $a$ by using *Dancing Links* [Knu09] as described in Section 7.3. More precisely, each time a variable $isSelected_u$ is assigned to true, we call Algorithm 7. This algorithm is modified as follows:

- After line 7, if $u \neq v$, we remove *true* from the domain of $isSelected_v$, where $v$ is the subset associated with the row of $c_{vb}$;
- After line 11, if $c_{vb}.head.size = 0$, we trigger a failure.

When backtracking from the assignment of $isSelected_u$ to true, we call Algorithm 8.

The propagation algorithm (resp. the algorithm that restores data structures when backtracking) has the same complexity as Algorithm 7 (resp. Algorithm 8), *i.e.*, $\mathcal{O}(n_p^2 \cdot n_c)$.

This propagator is called $DL$, and the Choco implementation of *exactCover* with this propagator is denoted $EC_{DL}$.

## 10.4 $DL+$ Propagator

Propagators introduced in Sections 10.2 and 10.3 ensure the same level of consistency as AC on the set decomposition. In this section, we introduce a stronger propagator, that filters more values by exploiting a property introduced in [Dav+11]: If there exist two elements $a, b \in S_C$ such that $cover_C(a) \subseteq cover_C(b)$ then, for every subset $u \in cover_C(b) \backslash cover_C(a)$, we cannot select $u$ together with the sets in $C$ because $u$ does not cover $a$ and every subset $v \in cover_C(a)$ is incompatible with $u$. When propagating our global constraint, this implies that we can assign $isSelected_u$ to *false*.

To efficiently detect $cover_C$ inclusions, we exploit the following property:

$$cover_C(a) \subseteq cover_C(b) \quad \Leftrightarrow \quad \#(cover_C(a) \cap cover_C(b)) = \#cover_C(a).$$

Hence, for each pair of uncovered elements $\{a, b\} \subseteq S_C$, we maintain a counter, denoted $\#cover_C(a, b)$, that gives the number of subsets that both belong to $cover_C(a)$ and $cover_C(b)$, *i.e.*,

$$\#cover_C(a, b) = \#(cover_C(a) \cap cover_C(b))$$

These counters are initialized in $\mathcal{O}(n_p^2 \cdot \#P)$. To incrementally maintain them during the search, we modify Algorithm 7 by calling a procedure before line 13: This procedure decrements $\#cover_C(b, c)$ for every pair of elements $\{b, c\} \subseteq v$, where $v$ is the subset associated with cell $c_{va}$. Indeed, as $v$ has been removed from both $cover_C(b)$ and $cover_C(c)$, it must also be removed from the intersection of these two sets. The complexity of this procedure is $\mathcal{O}(n_p^2)$, and the complexity of Algorithm 7 with this call becomes $\mathcal{O}(n_p^3 \cdot n_c)$.

Then, at the end of Algorithm 7, for every pair of elements $\{a, b\} \subseteq S_C$ such that $\#cover_C(a, b) = \#cover_C(a)$, and for every subset $v \in cover_C(b) \setminus cover_C(a)$, we assign $isSelected_v$ to *False*. This is done in $\mathcal{O}(\#S_C^2 \cdot n_c)$. Therefore, the time complexity of $DL+$ is $\mathcal{O}(n_p^3 \cdot n_c + \#S_C^2 \cdot n_c)$.

**Example 10.3.** Let us consider the example displayed in Fig. 7.4, when $C = \{x\}$. At the end of Algorithm 7, we have $\#cover_C(a, d) = h_d.size = 2$ and, therefore, $isSelected[t]$ is assigned to *false*.

We modify similarly Algorithm 8 to restore $\#cover_C(a, b)$ counters when backtracking, and the complexity of Algorithm 8 becomes $\mathcal{O}(n_p^3 \cdot n_c + \#S_C^2 \cdot n_c)$.

This propagator is called $DL+$, and the Choco implementation of *exactCover* with this propagator is denoted $EC_{DL+}$.

## 10.5 Experimental Evaluation

**Comparison of *SetDec*, $EC_{Basic}$, $EC_{DL}$, and $EC_{DL+}$.** We have considered the same search strategy for all implementations, which corresponds to the ordering heuristic introduced by Knuth in [Knu09]:

- For *SetDec*, this is done by branching on *coveredBy* variables and using the *minDom* heuristic to select the next *coveredBy* variable to assign (as maintaining AC ensures that $D(coveredBy[a]) = cover_C(a)$);
- For $EC_{Basic}$, $EC_{DL}$, and $EC_{DL+}$, at each node of the search tree, we search for the element $a \in S_C$ such that $\#cover_C(a)$ is minimal, and for each subset $u \in cover_C(a)$ we create a branch where $isSelected_u$ is assigned to *true*.

In all cases, we break ties by fixing an order on elements and subsets, and we consider the same order in all implementations.

We consider the problem of enumerating all solutions of EC instances built from the instance ERP1 described in Chapter 8. As there is a huge number of solutions, we consider instances obtained from ERP1 by selecting $p$% of its subsets in $P$, with $p \in \{20, 25, 30, 35, 40\}$. For each value of $p$, we have randomly generated ten instances and we report average results on these ten instances.

Table 10.1 displays the number of choice points performed by *SetDec* and $EC_{DL+}$ to enumerate all solutions. $EC_{Basic}$ and $EC_{DL}$ explore the same number of choice

| | | | | | Choice points | | CPU time (in seconds) | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| p | #sol | $n_p$ | $n_c$ | $n_i$ | *SetDec* | $EC_{DL+}$ | *SetDec* | $EC_{Basic}$ | $EC_{DL}$ | $EC_{DL+}$ |
| 20 | $7 \cdot 10^3$ | 34 | 122 | 305 | $54 \cdot 10^3$ | $16 \cdot 10^3$ | 6 | 1 | 1 | **0** |
| 25 | $3 \cdot 10^5$ | 34 | 160 | 359 | $14 \cdot 10^5$ | $6 \cdot 10^5$ | 112 | 9 | 7 | **4** |
| 30 | $5 \cdot 10^6$ | 37 | 184 | 440 | $20 \cdot 10^6$ | $11 \cdot 10^6$ | 1,469 | 122 | 82 | **51** |
| 35 | $5 \cdot 10^7$ | 37 | 210 | 510 | $19 \cdot 10^7$ | $11 \cdot 10^7$ | 19,226 | 1,178 | 732 | **461** |
| 40 | $5 \cdot 10^8$ | 50 | 264 | 630 | $16 \cdot 10^8$ | $10 \cdot 10^8$ | - | 10,168 | 5,501 | **4,036** |

Table 10.1 – Comparison of *SetDec*, $EC_{Basic}$, $EC_{DL}$, and $EC_{DL+}$ for enumerating all solutions. For each percentage $p$ of selected subsets in ERP1, we display: the number of solutions (#sol), the maximum size of a subset ($n_p$), the maximum number of subsets that cover an element ($n_c$), the maximum number of subsets that are incompatible with a subset ($n_i$), the number of choice points of *SetDec* and $EC_{DL+}$, and the CPU time of *SetDec*, $EC_{Basic}$, $EC_{DL}$, and $EC_{DL+}$ (average values on ten instances per line). We report '-' when time exceeds 50,000 seconds.

points as *SetDec* since they achieve the same filtering level and they consider the same ordering heuristic. $EC_{DL+}$ explores less choice points: The number of choice points explored by $EC_{DL+}$ is twice as large as the number of solutions, whereas the number of choice points explored by SetDec, $EC_{Basic}$ and $EC_{DL}$ is 7 times (resp. 4, 4, 3, and 3) as large as the number of solutions when $p = 20$ (resp. $p = 25$, 30, 35, and 40).

If *SetDec*, $EC_{Basic}$ and $EC_{DL}$ explore the same number of choice points, $EC_{Basic}$ and $EC_{DL}$ are an order faster than *SetDec*, showing the interest of a global propagation algorithm. $EC_{DL}$ is faster than $EC_{Basic}$, and when increasing $p$ (*i.e.*, the number of subsets in $P$), the difference between $EC_{DL}$ and $EC_{Basic}$ also increases. Actually, the time complexities of the propagation algorithms used in $EC_{Basic}$ and $EC_{DL}$ are $\mathcal{O}(n_p \cdot n_i)$ and $\mathcal{O}(n_p^2 \cdot n_c)$. Average values for $n_p$, $n_c$, and $n_i$ are reported in Table 10.1, and we can see that $n_i$ is larger than $n_c$ which is much larger than $n_p$. Furthermore, the $n_c$ factor in the time complexity of $EC_{DL}$ is a loose upper bound as we incrementally maintain $cover_C$ lists: In practice, we do not iterate over $n_c$ subsets, but only over the subsets in $cover_C$ lists.

As expected, $EC_{DL+}$ explores fewer choice points than $EC_{DL}$. However, the gap decreases when $p$ increases because inclusions of $cover_C$ sets become less frequent when increasing the number of subsets in $P$. Even if the time complexity of $EC_{DL+}$ is an order higher than the time complexity of $EC_{DL}$ ($\mathcal{O}(n_p^3 \cdot n_c)$ instead of $\mathcal{O}(n_p^2 \cdot n_c)$), the reduction of the search space achieved by $EC_{DL+}$ allows it to be faster than $EC_{DL}$. However, if it is twice as fast for small instances, the gain becomes smaller when increasing $p$.

**Experimental Comparison with SAT and `libexact`.** Let us now compare $EC_{DL}$ and $EC_{DL+}$ with the SAT model of [Jun+10], using the SAT solver `clasp` [Geb+12] with the `ladder` encoding which obtains the best results, and the `libexact` [Kas+08] implementation of the dedicated DLX algorithm [Knu09]. Results are reported in Table 10.2. As expected, `libexact` is always faster than $EC_{DL+}$: `libexact` is 3 times as fast as $EC_{DL+}$, and this ratio is rather constant when $p$ increases. The gap between these two approaches is explained (1) by the difference of support languages (Java for

| | CPU time (in seconds) | | | | Memory consumption (in MegaBytes) | | | |
|---|---|---|---|---|---|---|---|---|
| p | $EC_{DL}$ | $EC_{DL+}$ | `libexact` | SAT | $EC_{DL}$ | $EC_{DL+}$ | `libexact` | $SAT_{ladder}$ |
| 20 | 1 | 0 | **0** | 2 | 158 | 130 | **2** | 9 |
| 25 | 7 | 4 | **2** | 59 | 181 | 170 | **2** | 13 |
| 30 | 82 | 51 | **18** | 1,360 | 221 | 175 | **2** | 19 |
| 35 | 732 | 461 | **143** | 14,507 | 319 | 210 | **2** | 32 |
| 40 | 5,501 | 4,036 | **1,315** | - | 345 | 268 | **2** | - |

Table 10.2 – Comparison of $EC_{DL}$, $EC_{DL+}$, `libexact`, and SAT for enumerating all solutions. We report '-' when time exceeds 50,000s.

$EC_{DL+}$ and C for `libexact`), and (2) by the cost of using a generic CP solver instead of a dedicated algorithm.

$EC_{DL+}$ is faster than $SAT_{ladder}$, and the gap between the two approaches increases when increasing $p$, showing that $EC_{DL+}$ has better scale-up properties than $SAT_{ladder}$: $EC_{DL+}$ is 10 times as fast as $SAT_{ladder}$ when $p = 20$ and 31 times as fast when $p = 35$. When $p = 40$, $SAT_{ladder}$ is not able to enumerate all solutions within the CPU time limit of $50,000$ seconds whereas $EC_{DL+}$ needs $4,036$ seconds on average.

We also display memory consumption in Table 10.2. Clearly, $EC_{DL}$ and $EC_{DL+}$ need much more memory than $SAT_{ladder}$ and `libexact`. Note that we have not optimized our code with respect to memory consumption. In particular, each instance is memorized in two different data structures: a first data structure which is built when reading the instance, and the Dancing Link data structure which is built from the first data structure.

Finally, in Fig. 10.1, we compare $EC_{DL}$, $EC_{DL+}$, `libexact` and $SAT_{ladder}$ on the benchmark instances of [Jun+10][1]. This benchmark is composed of several families of exact cover instances of combinatorial origin such as Latin squares or Steiner triple systems (see [Jun+10] for more details).

Figure 10.1a compares $EC_{DL}$ with `libexact`. $EC_{DL}$ is always at least one order of magnitude slower than `libexact` but the gap decreases for the hardest instances.

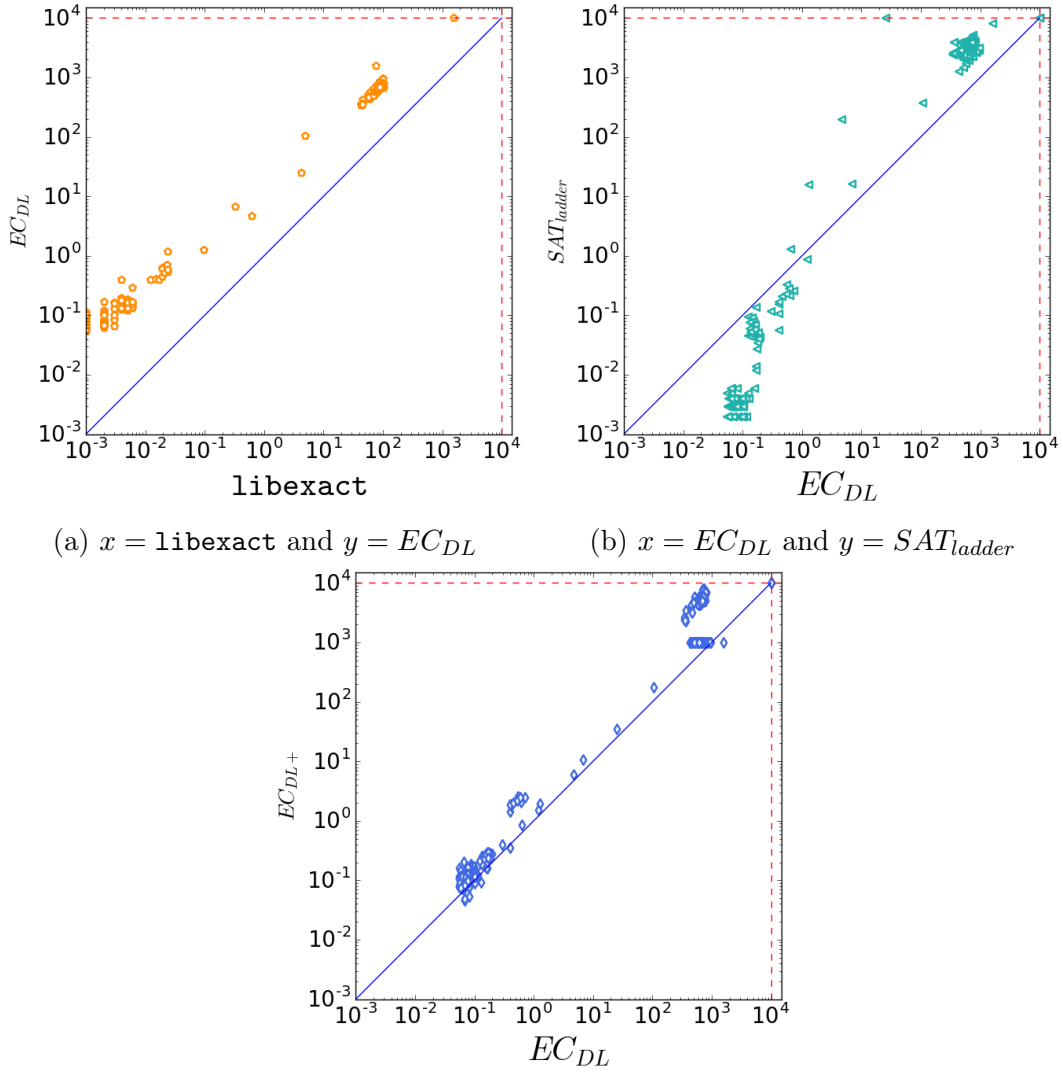
Figure 10.1b shows that $SAT_{ladder}$ is much more efficient than $EC_{DL}$ for the easiest instances. However, $EC_{DL}$ has better scale-up properties and becomes faster than $SAT_{ladder}$ for harder instances, which confirms the trends identified when analyzing Table 10.2.

Figure 10.1c highlights that the advanced filtering performed in $EC_{DL+}$ slightly pays off for only a few instances. $EC_{DL+}$ has most of the time similar performance than $EC_{DL}$ but $EC_{DL}$ becomes much better for Steiner triples system instances (on the top-right corner of the figure). Indeed, $EC_{DL}$ is 11 times faster than $EC_{DL+}$ on some of these instances.

## 10.6 Extension of *exactCover* to *exactCoverCost*

In some applications, costs are associated with subsets of $P$. This is the case in our conceptual clustering application, where different utility measures (such as the frequency, the size, the split, or the diameter) are associated with each subset. In this case, we

[1]This benchmark is available at https://users.ics.aalto.fi/tjunttil/experiments/CP2010/




(a) $x =$ `libexact` and $y = EC_{DL}$

(b) $x = EC_{DL}$ and $y = SAT_{ladder}$

(c) (1) $x = EC_{DL}$ and $y = EC_{DL+}$

Figure 10.1 – Comparison of $EC_{DL}$, $EC_{DL+}$, `libexact`, and SAT on benchmark instances of [Jun+10]: Each point $(x, y)$ corresponds to an instance which is solved in $x$ seconds with one approach and $y$ seconds with the other approach.

may add constraints on minimal and/or maximal costs associated with selected subsets, or we may search for solutions which maximize the minimal cost (or minimize the maximal cost) associated with a selected subset.

More formally, let $n$ be the number of different costs and, for each $i \in [1, n]$ and each subset $u \in P$, let $cost_i(u)$ denote the $i^{th}$ cost associated with $u$. For each $i \in [1, n]$, we introduce two integer variables $MinCost_i$ and $MaxCost_i$. We define the global constraint $exactCoverCost_{S,P,cost}(isSelected, MinCost, MaxCost)$ which is satisfied iff all *isSelected* variables assigned to *true* correspond to an exact cover of $(S, P)$ and *MinCost* and *MaxCost* variables are assigned to the minimum and maximum costs

associated with selected subsets, *i.e.*,

$$\begin{array}{rl} \forall a \in S, & \#\{u \in cover(a) : isSelected_u\} = 1 \\ \forall i \in [1,n], & MinCost_i = \min_{u \in P} isSelected[u] \cdot cost_i(u) \\ \forall i \in [1,n], & MaxCost_i = \max_{u \in P} isSelected[u] \cdot cost_i(u) \end{array}$$

This constraint is propagated like *exactCover*, but before starting the search we remove from $P$ every subset $u$ that does not satisfy the bound constraints, *i.e.*, such that there exists $i \in [1,n]$ for which $cost_i(u) < MinCost_i.lb$ or $cost_i(u) > MaxCost_i.ub$. Then, each time a variable $isSelected_u$ is assigned to *true*, for each $i \in [1,n]$, we propagate:

$$\begin{array}{rcl} MinCost_i.ub & = & \min\{MinCost_i.ub, cost_i(u)\}, \\ MaxCost_i.lb & = & \max\{MaxCost_i.lb, cost_i(u)\}. \end{array}$$

Also, each time $MinCost_i.lb$ (resp. $MaxCost_i.ub$) is updated, for each subset $u$ such that $cost_i(u) < MinCost_i.lb$ (resp. $cost_i(u) > MaxCost_i.ub$), we assign $isSelected_u$ to *false*.

## 10.7 Discussion

In this chapter, we have introduced a global constraint dedicated to the exact cover problem, and three algorithms for propagating it: A basic propagator, that does not use backtrackable data structures and ensures the same level of filtering as AC on the set decomposition, a propagator called DL that also ensures the same level of filtering but uses Dancing Links to efficiently maintain data, and a propagator called DL+ that ensures a stronger filtering than DL.

However, to solve conceptual clustering problems, we usually add constraints on the number of selected subsets, which corresponds to the number $k$ of clusters, at least to constrain $k$ to be strictly greater than one and strictly lower than the number of transactions. In the next chapter, we study how to extend *exactCover* for doing this efficiently.

# Chapter 11

# Constraining the number of selected subsets

## Contents



In some applications, we may need to add constraints on the number of selected subsets. For example, in our conceptual clustering application, the number of selected subsets corresponds to the number of clusters and we often add constraints in order to forbid solutions with too few or too many clusters. In this case, we declare an integer variable $k$ which is constrained to be equal to the number of selected subsets. This may be done either by explicitly adding constraints on $k$, as explained in Section 11.1, or by defining a new global constraint $exactCoverK_{S,P}(isSelected, k)$ and new propagators that ensure that the integer variable $k$ is equal to the number of *isSelected* variables assigned to *true*, as explained in Section 11.2.

## 11.1 Addition of Existing Constraints to *exactCover*

In this section, we study how to add constraints to $exactCover_{S,P}(isSelected)$ in order to ensure that the number of selected subsets is equal to an integer variable $k$.

A first possibility is to add the constraint:

$$\sum_{u \in P} isSelected[u] = k$$

We denote $EC_{DL,sum}$ and $EC_{DL+,sum}$ the Choco implementation of *exactCover* that combines this sum constraint with the propagation algorithms introduced in Sections 10.3 and 10.4, respectively.

Another possibility is to use the *NValue* global constraint introduced Section 5.5. To combine *NValue* with *exactCover*, we must introduce *coveredBy* variables: For each element $a \in S$, we define an integer variable $coveredBy_a$ whose domain is $D(coveredBy_a) =$

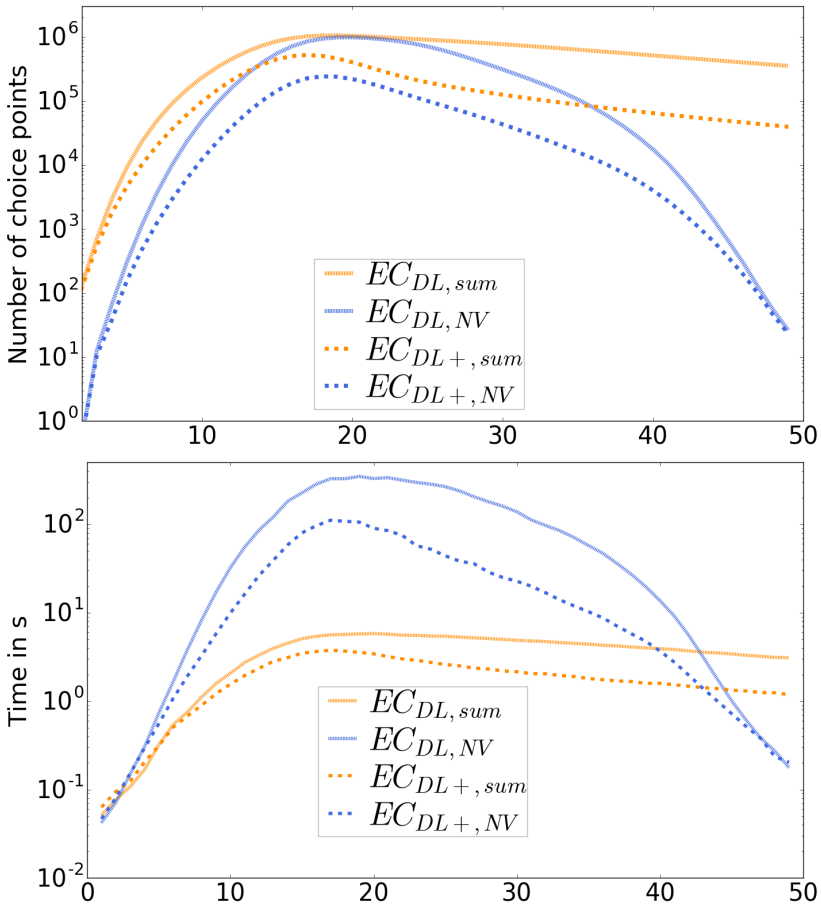



Figure 11.1 – Comparison of the number of choice points (top) and time (bottom) of $EC_{DL,sum}$, $EC_{DL,NV}$, $EC_{DL+,sum}$, and $EC_{DL+,NV}$ for enumerating all solutions of 10 instances obtained from $ERP1$ by selecting randomly 25% of its subsets, when $k$ is assigned to $x$, with $x \in [2, 49]$.

$cover(a)$, like in the boolean model introduced in Section 7.4.1. In this case, the complete set of constraints is:

$$\forall u \in P, \forall a \in cover(u), coveredBy_a = u \Leftrightarrow isSelected_u$$
$$NValue(coveredBy, k)$$
$$exactCover_{S,P}(isSelected)$$

We denote $EC_{DL,NV}$ and $EC_{DL+,NV}$ the Choco implementations that combine these constraints with the propagation algorithms introduced in Sections 10.3 and 10.4, respectively.

**Experimental Evaluation.** We consider the problem of enumerating all solutions of an exact cover problem when the number of selected subsets $k$ is constrained to be equal to an integer value. We solve this problem on 10 instances obtained from ERP1 by selecting randomly 25% of the subsets in $P$. These instances have $\#S = 50$ elements and the number of subsets is close to 400. We vary the value assigned to $k$ from 2 to $\#S - 1$. For each point $(x, y)$ in Figure 11.1, $y$ is the performance measure

(time or number of choice points) for enumerating all solutions when $k$ is assigned to $x$ (*i.e.*, to enumerate all exact covers with exactly $x$ selected subsets).

In Figure 11.1, we compare performances of $EC_{DL,sum}$, $EC_{DL,NV}$, $EC_{DL+,sum}$, and $EC_{DL,NV}$. Using *NValue* strongly reduces the number of choice points, especially for extremal values of $k$. However, the propagation of *NValue* is much more time consuming than the propagation of a *sum* constraint. As a consequence, using *NValue* does not pay-off, except for very large values of $k$ (*i.e.*, when $k > 40$) for which *NValue* reduces the number of choice points by several orders of magnitude. Using $DL+$ instead of $DL$ for propagating *exactCover* reduces the number of choice points, especially when $k$ is larger than 10, and this stronger filtering also reduces the search time, except for very low values of $k$: When $k$ is lower than 5, variants that use $DL$ are slightly faster than variants that use $DL+$.

Note that the advanced filtering $DL+$ seems to be complementary with the filtering performed by *NValue*: When $k \in [16, 23]$, the gain of $DL+$ is increased when it is combined with *NValue*. For instance, when $k = 19$, $EC_{DL+,NV}$ explores $743.10^3$ less choice points than $EC_{DL,NV}$ whereas $EC_{DL+,sum}$ explores $595.10^3$ less choice points than $EC_{DL,sum}$.

## 11.2 Definition of *exactCoverK* and *exactCoverCostK*

To better propagate constraints between $k$ and *isSelected* variables, we define the global constraint $exactCoverK_{S,P}(isSelected, k)$. This constraint is satisfied iff the number of *isSelected* variables assigned to *true* is equal to $k$ and the subsets associated with these variables define an exact cover of $S$, *i.e.*,

$$\forall a \in S, \#\{u \in cover(a) : isSelected_u\} = 1$$
$$\sum_{u \in P} isSelected_u = k$$

This constraint is extended to handle the case where costs are associated with elements, and bounds on the costs of the selected elements must be maintained, in a similar way as *exactCover*, as explained in Section 10.6. More precisely, we define the constraint $exactCoverCostK_{S,P,cost}(isSelected, MinCost, MaxCost, K)$ that combines the propagators of *exactCoverK* described in this section with the propagators associated with cost bounds described in Section 10.6.

In this section, we describe three algorithms that propagate the constraint that relates $k$ with *isSelected* variables, and that may be used both for *exactCoverK* and *exactCoverCostK*. These propagators are experimentally compared for the simple enumeration problem introduced in Section 11.1 which is solved with *exactCoverK* as it does not involve utility costs. Experimental results for *exactCoverCostK* are reported in the next chapter.

### 11.2.1 *Basic* Propagator

A first basic filtering simply ensures that $k$ is bounded by the number of subsets that are already selected on the lower side, and the number of subsets that can be selected on the upper side, *i.e.*, we ensure:

$$nbTrue \leq k.lb \leq k.ub \leq \#P - nbFalse$$

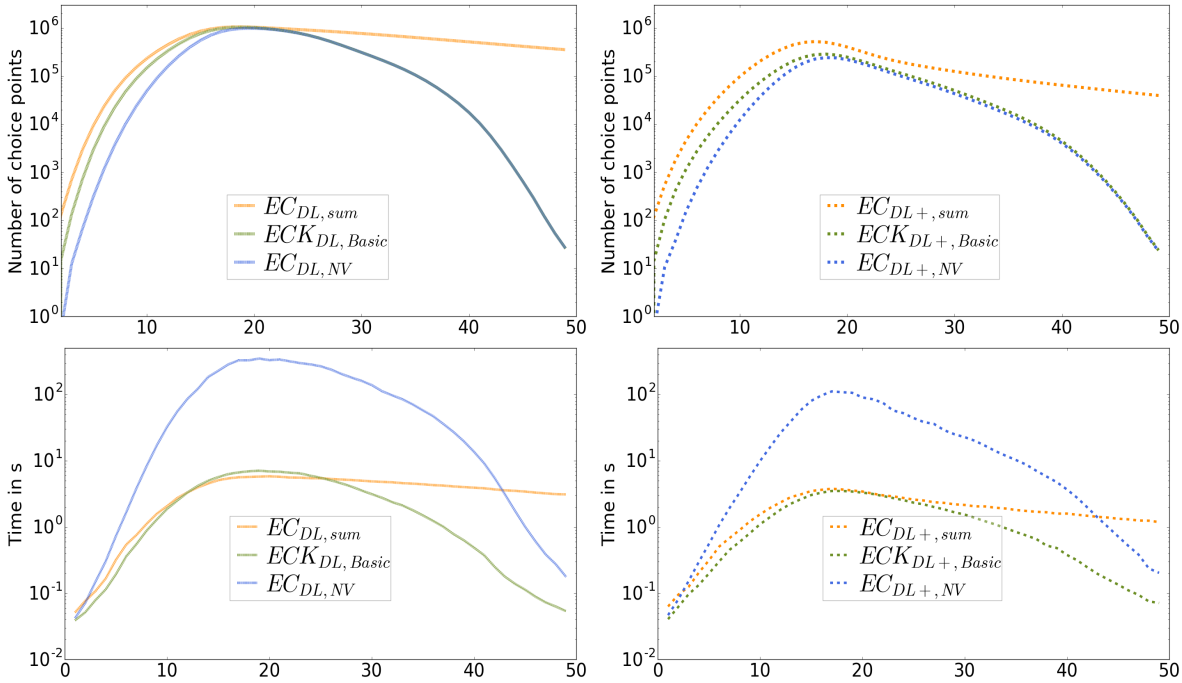



Figure 11.2 – Comparison of $ECK_{DL,Basic}$ with $EC_{DL,sum}$ and $EC_{DL,NV}$ (left), and $ECK_{DL+,Basic}$ with $EC_{DL+,sum}$ and $EC_{DL+,NV}$ (right) for enumerating all solutions of 10 instances obtained from $ERP1$ by selecting randomly 25% of its subsets, when $k$ is assigned to $x$, with $x \in [2, 49]$.

where *nbTrue* (resp. *nbFalse*) is the number of *isSelected* variables assigned to *true* (resp. *false*).

This filtering ensures the same consistency as maintaining AC on the *sum* constraint $\sum_{u \in P} isSelected_u = k$.

We tighten this filtering by taking into account the set $S_C$ of elements that are still not covered. More precisely, as the largest subset in $P$ contains $n_p$ elements, we need at least $\lceil \frac{\#S_C}{n_p} \rceil$ subsets to cover all elements in $S_C$ (where $\lceil x \rceil$ denotes the smallest integer greater than or equal to $x$). Similarly, we need at most $\lceil \frac{\#S_C}{\min_{u \in P} \#u} \rceil$ subsets to cover all elements in $S_C$. Hence, we tighten the filtering to ensure:

$$nbTrue + \lceil \frac{\#S_C}{n_p} \rceil \leq k.lb \leq k.ub \leq \min\{\#P - nbFalse, nbTrue + \lceil \frac{\#S_C}{\min_{u \in P} \#u} \rceil\}$$

We denote $ECK_{DL,Basic}$ (resp. $ECK_{DL+,Basic}$) the Choco implementation of *exactCoverK* that combines this basic filtering on $k$ with the propagation algorithms introduced in Section 10.3 (resp. 10.4).

**Experimental evaluation.** We compare $ECK_{*,Basic}$ with $EC_{*,sum}$ and $EC_{*,NV}$ (for $* \in \{DL, DL+\}$) in Fig. 11.2. When the number $x$ of selected subsets is smaller than 20, $EC_{*,sum}$ explores more choice points than $ECK_{*,Basic}$, and $ECK_{*,Basic}$ explores more choice points than $EC_{*,NV}$. However, as the propagation of *NValue* is very time consuming, $EC_{*,NV}$ needs much more time than $ECK_{*,Basic}$ and $EC_{*,sum}$ (except for the smallest values of $x$). $ECK_{*,Basic}$ is slightly faster than $EC_{*,sum}$ when $k$ is smaller than 12 for $DL$ and 20 for $DL+$.


Figure 11.3 – Intersection graphs associated with the instance displayed in Fig. 7.1 when $C = \emptyset$ (left) and when $C = \{x\}$ (right).

When the number $x$ of selected subsets is larger than 20, $ECK_{*,Basic}$ explores nearly the same number of choice points as $EC_{*,NV}$ whereas $EC_{*,sum}$ explore much more choice points. In this case, $ECK_{*,Basic}$ is faster than both $EC_{*,sum}$ and $EC_{*,NV}$.

### 11.2.2 *MD* Propagator

We propose to compute a better lower bound for $k$ by integrating a propagation algorithm introduced in [Bes+05] for *NValue*. The interest of integrating this algorithm within the propagation of *exactCoverK* instead of combining our global constraint with *NValue* is that the propagation algorithm of [Bes+05] exploits an intersection graph $G = (S_C, E_C)$ which can be derived in a straightforward way from the data structure we maintain when propagating *exactCover*. This graph associates a vertex with every non covered element in $S_C$ and an edge with every pair of non covered elements that may be covered by a same subset, *i.e.*,

$$E_C = \{\{a, b\} \subseteq S_C : cover_C(a) \cap cover_C(b) \neq \emptyset\}.$$

As we maintain in $\#cover_C(a, b)$ the size of $cover_C(a) \cap cover_C(b)$ when using $DL+$ filtering, edges of $E_C$ simply correspond to pairs $\{a, b\} \subseteq S_C$ such that $\#cover_C(a, b) > 0$.

**Example 11.1.** For example, we display in Fig. 11.3 the two intersection graphs associated with the instance displayed in Fig. 7.1 when $C = \emptyset$ and when $C = \{x\}$, respectively.

An independent set in the intersection graph is a set of vertices $I \subseteq S_C$ with no edge in common. In other words, for every pair of elements $\{a, b\} \subseteq I$, we have $cover_C(a) \cap cover_C(b) = \emptyset$. As a consequence, it is not possible to cover all the elements of $I$ with less than $\#I$ subsets.

In [Bes+05], Bessière *et al.* use this property to provide a lower bound on the number of different values in the *NValue* global constraint. In particular, they propose to use the greedy algorithm of [Hal+97], called Minimum Degree (MD), to compute a large independent set: Starting from an empty independent set, at each iteration they choose a vertex $v$ of minimum degree, add it to the independent set, and remove $v$ and all its adjacent vertices from the graph, until the graph is empty. The complexity of this algorithm is linear with respect to the number of edges in the intersection graph, provided that buckets are used to incrementally maintain the set of vertices of degree $d$ for every $d \in [0, \#S_c - 1]$.

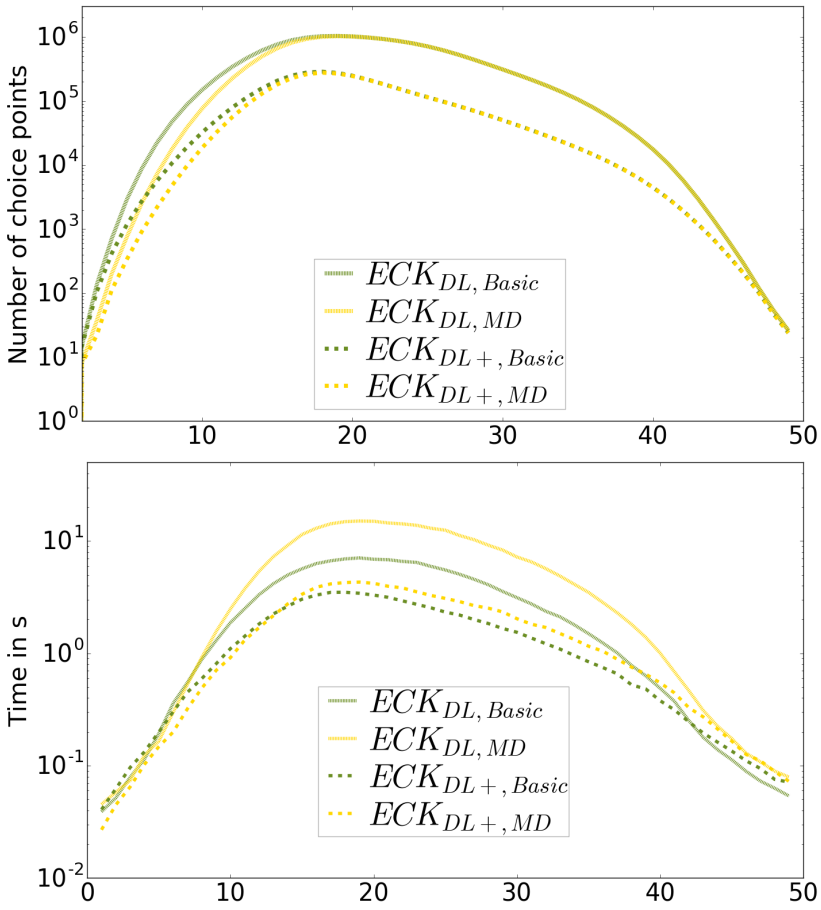



Figure 11.4 – Comparison of number of choice points (top) and CPU time (bottom) of $ECK_{DL,Basic}$, $ECK_{DL,MD}$, $ECK_{DL+,Basic}$ and $ECK_{DL+,MD}$ for enumerating all solutions of 10 instances obtained from $ERP1$ by selecting randomly 25% of its subsets, when $k$ is assigned to $x$, with $x \in [2, 49]$.

Given an independent set $I$ built with MD, we tighten the lower bound of $k$ by ensuring:

$$nbTrue + \#I \leq k.lb$$

**Example 11.2.** For example, when $C = \emptyset$, MD builds the independent set $\{a, e, b\}$. This allows us to infer that $k$ must be greater than or equal to 3.

For each propagation algorithm $* \in \{DL, DL+\}$ introduced in Sections 10.3 and 10.4, we denote $ECK_{*,MD}$ the Choco implementation of *exactCoverK* that combines this tightened lower bound on $k$ with the basic upper bound on $k$ introduced in the previous section and the propagation algorithm $*$.

**Experimental evaluation.** $ECK_{*,MD}$ and $ECK_{*,Basic}$ (with $* \in \{DL, DL+\}$) are experimentally compared in Figure 11.4. For low values of $k$ (*i.e.*, when $k \leq 20$), $ECK_{*,MD}$ explores less choice points than $ECK_{*,Basic}$, whereas for higher values of $k$ they explore the same number of choice points. However, computing independent sets is time consuming. Therefore, the $MD$ filtering reduces computation time only when $k < 8$ (resp. $k < 14$) when $* = DL$ (resp. $* = DL+$).

### 11.2.3 $MD+$ Propagator

In [Bes+05], Bessière *et al.* also show how to use independent sets to filter domains when the cardinality of the independent set is equal to the number of different values. In our context, this filtering allows us to assign to *false* some *isSelected* variables when $k.lb = k.ub = nbTrue + \#I$. More precisely, for every subset $u$ that does not cover an element of $I$ (*i.e.*, $u \notin \cup_{a \in I} cover_C(a)$), we can assign $isSelected_u$ to *false*.

This filtering may be done not only for $I$, but also for any other independent set that has the same cardinality as $I$. However, as this is too expensive to compute all independent sets that have the same cardinality as $I$, we only compute a subset of them using the algorithm described in [Bel01] (as proposed in [Bes+05]). This algorithm computes in linear time with respect to $\#E_C$ all independent sets that differ from $I$ by only one vertex: It iterates on every vertex $a \in I$, and for every edge $\{a, b\} \in E_C$; if $b$ is not adjacent to any vertex of $I \setminus \{a\}$, it adds the independent set $I \setminus \{a\} \cup \{b\}$.

Let $I_0$ be the initial independent set computed with MD, and $I_1, \ldots, I_n$ be the independent sets derived from $I_0$. We assign to *false* every variable $isSelected_u$ such that there exists an element $a$ in an independent set $I_j$ not covered by $u$, *i.e.*:

$$u \notin \bigcap_{j \in [0,n]} \bigcup_{a \in I_j} cover_C(a) \Rightarrow D(isSelected_u) = \{false\}$$

**Example 11.3.** On our running example, when $C = \{x\}$, MD builds a first independent set $\{b, a\}$. Therefore, the lower bound of $k$ is 3. The upper bound of $k$ is also equal to 3 because:

$$k.ub = \min\{\#P - nbFalse, nbTrue + \lceil \frac{\#S_C}{\min_{u \in P} \#u} \rceil\} = \min\{5, 1 + \frac{4}{2}\} = 3.$$

Since $k.lb = k.ub = nbTrue + \#I$, we can apply the filtering on *isSelected* domains. We derive from the first independent set $\{b, a\}$ a second independent set $\{b, d\}$. We have $cover_C(b) = \{z\}$, $cover_C(a) = \{t, u, v\}$ and $cover_C(d) = \{u, v\}$. We can assign to *false* every *isSelected* variable associated with a subset that does not belong to:

$$\{u, v, z\} \cap \{t, u, v, z\} \ = \{u, v, z\}.$$

Therefore, we assign $isSelected_t$ to *false*.

For each propagation algorithm $* \in \{DL, DL+\}$ introduced in Sections 10.3 and 10.4, we denote $ECK_{*,MD+}$ the Choco implementation of *exactCoverK* that combines this filtering of *isSelected* variables with the tightened lower bound on $k$ and the basic upper bound on $k$ introduced in the previous sections and the propagation algorithm $*$.

**Experimental evaluation.** $ECK_{*,MD+}$ and $ECK_{*,MD}$ (with $* \in \{DL, DL+\}$) are experimentally compared in Fig. 11.5. For low values of $k$ (*i.e.*, when $k \leq 27$), $ECK_{*,MD+}$ explores less choice points than $ECK_{*,MD}$, whereas for higher values of $k$ they explore the same number of choice points. For instance, when $k < 8$, $ECK_{*,MD+}$ explores twice less choice points compared with $ECK_{*,MD}$. $MD+$ filtering has the same complexity as $MD$ filtering. Therefore, $ECK_{*,MD+}$ is slightly faster than $ECK_{*,MD}$ when $k < 15$ whereas they have very similar performances for higher values of $k$.

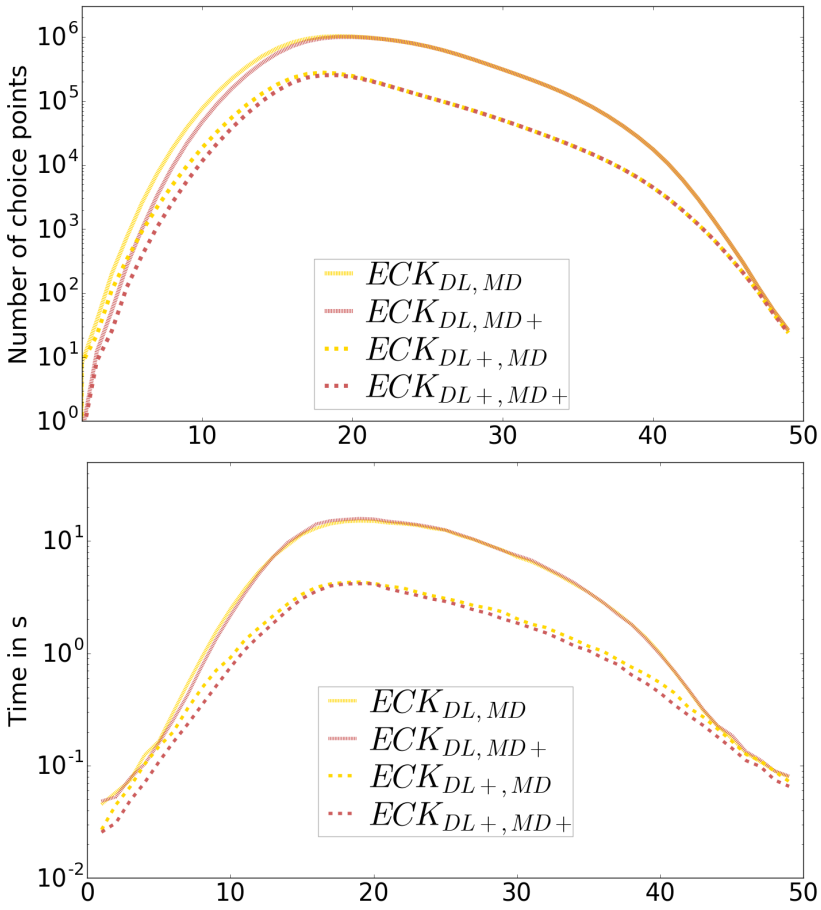



Figure 11.5 – Comparison of number of choice points (top) and CPU time (bottom) of $ECK_{DL,MD}$, $ECK_{DL,MD+}$, $ECK_{DL+,MD}$ and $ECK_{DL+,MD+}$ for enumerating all solutions of 10 instances obtained from $ERP1$ by selecting randomly 25% of its subsets, when $k$ is assigned to $x$, with $x \in [2, 49]$.

## 11.3 Discussion

We have studied two different ways for constraining the number of selected subsets to be equal to an integer variable $k$. A first possibility is to add existing constraints to *exactCover*, and we have shown that this may be done either with a *sum* or an *NValue* constraint. *NValue* has a stronger propagation than *sum*, but this propagation also has a higher time complexity, and we have experimentally shown that this pays off only for the largest values of $k$.

A second possibility is to define a new global constraint, called *exactCoverK*, together with new propagators. We have introduced three propagators for this new global constraint:

- The *Basic* propagator combines a filtering similar to the filtering achieved by a *sum* constraint with simple bounds on $k$;
- The *MD* propagator combines the *Basic* propagator with a propagator introduced for *NValue* in [Bes+05], which tightens the lower bound of $k$ by computing an independent set in the intersection graph of $cover_C$ sets. We show how to derive this intersection graph from our data structure without increasing the time

complexity. Hence, the overhead of $MD$ with respect to *Basic* is only due to the computation of an independent set, which is done in linear time with respect to the number of edges in the intersection graph.

- The $MD+$ propagator combines the $MD$ propagator with an advanced filtering, also introduced in [Bes+05] for *NValue*, which filters the domains of *isSelected* variables when the cardinality of the independent set is equal to $k$.

$MD+$ is stronger than $MD$ which is stronger than *Basic*. However, propagation times are also increased as $MD$ involves computing an independent set, and $MD+$ a set of independent sets. As a consequence, stronger filterings do not always pay off, and their interest depends on the value of $k$.

These different possibilities for propagating the fact that the number of selected subsets must be equal to an integer variable $k$ may be combined with the two different propagation algorithms that have been introduced in Chapter 10: $DL$, which uses Dancing Links to ensure the same consistency as AC on the set decomposition, and $DL+$, which is stronger than $DL$ and further filter *isSelected* variables when the set of available subsets for covering an element is included in another set of available subsets.

Hence, we have six different levels of filtering for *exactCoverK* (resp. *exactCoverCostK*), denoted $ECK_{p_1,p_2}$ (resp. $ECCK_{p_1,p_2}$) with $p_1 \in \{DL, DL+\}$ and $p_2 \in \{Basic, MD, MD+\}$. In the next chapter, we evaluate the interest of these propagators on conceptual clustering problems.

# Chapter 12

# Evaluation of *exactCover* on Conceptual Clustering Problems

## Contents



In this chapter, we experimentally evaluate our new global constraints *exactCoverCost* and *exactCoverCostK* for solving conceptual clustering problems. We consider two different kinds of problems: mono-criterion problems, where a single objective function is optimized, and bi-criteria problems, where we search for the Pareto set of all non-dominated solutions with respect to two objective functions.

The experimental evaluation is designed to address the following questions:

- What propagation algorithms of our global constraints are the most efficient according to the considered problem?
- Is our new global constraint competitive with state-of-the-art declarative approaches described in Chapter 6?

We describe the experimental protocol in Section 12.1. We report experimental results for mono-criterion problems in Section 12.2. We consider problems where the number $k$ of clusters is fixed, and problems where this number is not fixed but bounded within a given interval. In Section 12.3, we report experimental results for bi-criteria problems. This problem is more challenging, and we introduce a new dynamic strategy for solving it.

## 12.1 Experimental Protocol

**Considered implementations of our new global constraints.** All our CP models are implemented with `Choco v.4.0.3`.

For all problems addressed in this chapter, we consider the four optimization criteria introduced in Chapter 6, *i.e.*, $Min_{frequency}$, $Min_{size}$, $Min_{split}$ and $Min\__{diameter}$. To solve these problems with our new global constraints, we use the variants that allow us to add constraints on bounds of the costs of the selected subsets, *i.e.*, *exactCoverCost* and *exactCoverCostK*.

When the number $k$ of selected subsets is not tightly constrained (*i.e.*, when $k$ is not fixed but simply bounded by a given interval, and the optimization criterion is not $Min_{frequency}$), we combine *exactCoverCost* with a *sum* constraint to constrain the number of selected subset to be equal to $k$ (as explained in Section 11.1). This Choco implementation is denoted $ECC_{p_1,sum}$, where $p_1 \in \{DL, DL+\}$ is the algorithm used to propagate the exact cover constraint (as described in Chapter 10).

When the number $k$ of selected subsets is tightly constrained(*i.e.*, when $k$ is fixed, or the optimization criterion is $Min_{frequency}$), we use *exactCoverCostK*. Its Choco implementation is denoted $ECCK_{p_1,p_2}$, where $p_1 \in \{DL, DL+\}$ is the algorithm used to propagate the exact cover constraint (as described in Chapter 10), and $p_2 \in \{Basic, MD, MD+\}$ is the algorithm used to propagate constraints on the number of selected subsets (as described in Chapter 11). Furthermore, when the utility measure is $Min_{frequency}$, we add the constraints introduced in Section 9.2, *i.e.*, $Min_{freq} \leq \#S - Max_{freq}$ and $Min_{freq} * K \leq \#S$ and we use the *ObjectiveStrategy* proposed by Choco [Pru+16] which performs a dichotomous branching over the domain of $Min_{freq}$.

**Other considered approaches.** For each considered problem, we compare $ECC$ or $ECCK$ with the best performing approach among the following approaches:

- *FCP1*, the CP model introduced by Dao *et al.* in [Dao+15a] and described in Section 6.4.2. When $k$ is fixed, we use the Gecode V4.3 [tea05] implementation provided by the authors. When $k$ is not fixed, we consider the extension of this implementation which is described in Section 6.4.2.
- *ILP*, the hybrid approach introduced by Ouali *et al.* in [Oua+16] and described in Section 6.4.3. We used `CPLEX v12.7` for the implementation of the ILP model.
- *FCP2*, the CP model introduced in Section 9.1. This model has been implemented with `Choco v.4.0.3`.
- *SetDec*, the hybrid approach that first uses LCM to extract all formal concepts and then uses the CP model introduced in Section 9.2 for solving an exact cover problem. This model has been implemented with `Choco v.4.0.3`.

For each kind of problem, we only report the results of the best performing approaches:

- When the number of clusters $k$ is fixed, the best approaches are *FCP1* and *ILP*;
- When the number of clusters $k$ is not fixed, the best approaches are
  - *FCP2* and *SetDec* when optimizing $Min_{frequency}$ or $Min_{size}$;

– *FCP1* when optimizing $Min_{split}$ or $Min_{-diameter}$.

These results are consistent with the experimental results reported in Chapter 9.

**Performance measures.** We consider two different performance measures, *i.e.*, the number of choice points and the CPU time. All experiments were conducted on Intel(R) Core(TM) i7-6700 with 3.40GHz of CPU and 65GB of RAM, using a single thread.

For all hybrid approaches that use LCM to extract all formal concepts in a preprocessing step, and then solve an exact cover problem (*i.e.*, *SetDec*, *ILP*, $ECC_{p_1,sum}$, and $ECCK_{p1,p_2}$), CPU times that are reported always include the time spent by LCM to extract all formal concepts (see Tables 8.1 and 8.2 for information on this time).

**Benchmarks.** We consider instances coming from the two benchmarks described in Chapter 8. The UCI benchmark allows us to evaluate our approach on classical machine learning instances, which are usually used to evaluate clustering and classification algorithms. The ERP benchmark allows us to evaluate our approach on instances very similar to instances that we must solve in our applicative context.

## 12.2 Single criterion optimization

In this section, we consider the problem of finding a conceptual clustering that optimizes one of the criteria introduced in Section 6.3. We consider two different problems: In Section 12.2.1, we report results when the number of clusters is fixed, *i.e.*, when $k$ is set to a given value; In Section 12.2.2, we report results when the number of clusters is not a priori known and only bounded in $[2, \#\mathcal{T} - 1]$, *i.e.*, we want more than one cluster and at least one cluster must contain two transactions.

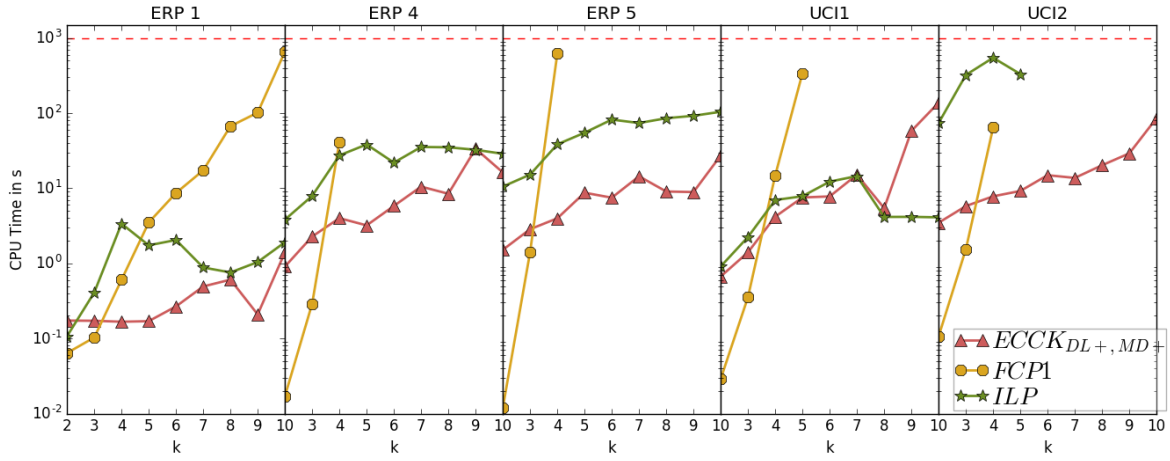
### 12.2.1 Single criterion optimization when $k$ is fixed

In our applicative context, we do not know *a priori* the number of clusters and, therefore, $k$ is not fixed. However, we made a few experiments to evaluate scale-up properties of our global constraint when $k$ is fixed to a given value.

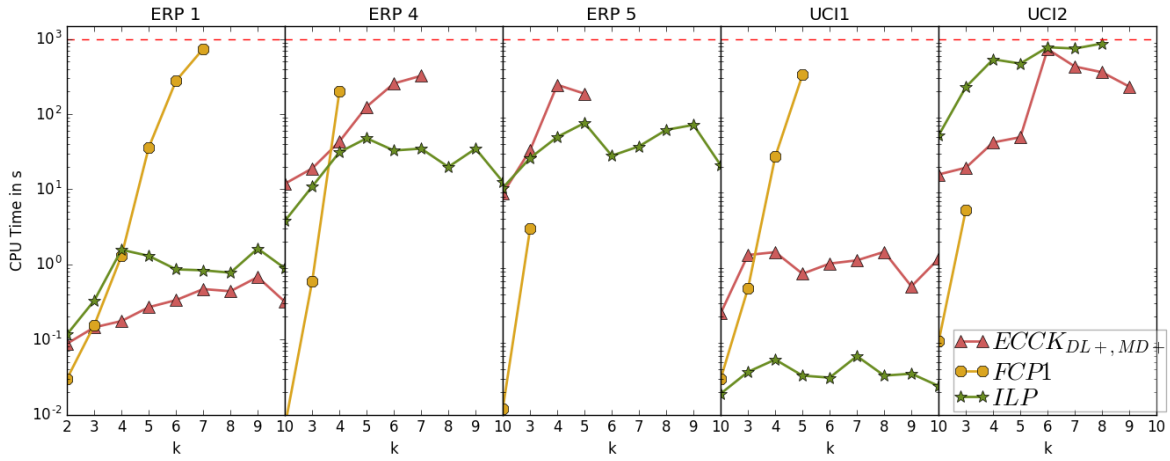
Fig. 12.1 reports results obtained with different values for $k$, from 2 to 10. We only consider two optimization criteria (*i.e.*, (a) $Min_{frequency}$ and (b) $Min_{size}$), and five representative instances (*i.e.*, ERP1, ERP4, ERP5, UCI1, and UCI2). We compare the variant of our global constraint which achieves the strongest filtering (*i.e.*, $ECCK_{DL+,MD+}$) with *FCP1* and *ILP*.

For $Min_{frequency}$, $ECCK_{DL+,MD+}$ is the only approach that is able to solve all instances for all values of $k$ within a CPU time limit of 1000 seconds. *FCP1* is always the fastest approach when $k < 4$ but it does not scale well when $k$ increases. *ILP* is also able to solve the first four instances for all values of $k$: It is slower than $ECCK_{DL+,MD+}$ for ERP instances, and faster for UCI1. However, ILP does not scale well for $UCI2$ and it is not able to complete its run when $k > 5$.

For $Min_{size}$, *ILP* is the only approach that is able to solve all instances except UCI2 when $k > 8$. Like for $Min_{frequency}$, *FCP1* is the fastest approach when $k = 2$ but it does not scale well when $k$ increases. $ECCK_{DL+,MD+}$ is the fastest approach for ERP1 and




(a) Average CPU time when maximizing $Min_{frequency}$




(b) Average CPU time when maximizing $Min_{size}$

Figure 12.1 – Comparison of $ECCK_{DL+,MD+}$, *FCP1* and *ILP* when $k$ is assigned to $x$, with $x \in [2, 10]$: (a) CPU time when maximizing $Min_{frequency}$; (b) CPU time when maximizing $Min_{size}$.

UCI2. However, it is not able to complete its run when $k > 7$ (resp. 5) for ERP4 and ERP5.

As a conclusion, for this problem, $ECCK_{DL+,MD+}$ and *ILP* are the best performing approaches and they are complementary: *ILP* tends to be more efficient for $Min_{size}$, and $ECK_{DL+,MD+}$ for $Min_{frequency}$.

### 12.2.2 Single criterion optimization when $k$ is bounded

Let us now consider the case where we do not know *a priori* the number of clusters, *i.e.*, $k$ is not fixed. In this case, we only constrain $k$ to be strictly greater than 2 and strictly smaller than the number of transactions, *i.e.*, $D(k) = [2, \#\mathcal{T} - 1]$.

| | $Min_{size}$ | | | | $Min_{split}$ | | | | $Min_{-diameter}$ | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $ECC_{DL,sum}$ | | $ECC_{DL+,sum}$ | | $ECC_{DL,sum}$ | | $ECC_{DL+,sum}$ | | $ECC_{DL,sum}$ | | $ECC_{DL+,sum}$ | |
| | time | nodes | time | nodes | time | nodes | time | nodes | time | nodes | time | nodes |
| ERP1 | **0.1** | **48** | 0.1 | **48** | **0.1** | **8** | 0.1 | **8** | **0.1** | **48** | 0.1 | **48** |
| ERP2 | **0.1** | **41** | 0.2 | 42 | **0.1** | **3** | 0.1 | **3** | **0.1** | **40** | 0.2 | 42 |
| ERP3 | **0.1** | **58** | 0.2 | 59 | **0.1** | **4** | 0.2 | **4** | **0.1** | **58** | 0.2 | 59 |
| ERP4 | **0.2** | **84** | 0.4 | **84** | **0.2** | **2** | 0.2 | **2** | **0.2** | **84** | 0.4 | **84** |
| ERP5 | **0.5** | **77** | 0.8 | 79 | **0.6** | 5 | 0.7 | **4** | **0.5** | **76** | 0.7 | 78 |
| ERP6 | **1.6** | **95** | 3.8 | **95** | **0.9** | 15 | 1.8 | **14** | **1.6** | **95** | 3.8 | **95** |
| ERP7 | **31.1** | **161** | 103.6 | **161** | **6.9** | **3** | 13.7 | **3** | **49.6** | **161** | 103.4 | **161** |
| UCI1 | **0.1** | **58** | 0.1 | **58** | **0.1** | **3** | 0.1 | **3** | **0.1** | **58** | 0.1 | **58** |
| UCI2 | **0.3** | **493** | 0.6 | **493** | **0.3** | **5** | 0.7 | **5** | **0.2** | **493** | 0.5 | **493** |
| UCI3 | **0.5** | **215** | 0.8 | **215** | **0.7** | **33** | 2.4 | **33** | **0.5** | **215** | 0.8 | **215** |
| UCI4 | **3.3** | **152** | 6.2 | **152** | **0.7** | **3** | 0.8 | **3** | **3.2** | **152** | 6.3 | **152** |
| UCI5 | **1.0** | **338** | 1.5 | **338** | **1.9** | **89** | 4.2 | **89** | **1.0** | **338** | 1.5 | **338** |
| UCI6 | **17.0** | **136** | 23.5 | **136** | **17.9** | **5** | 34.9 | **5** | **28.2** | **136** | 38.3 | **136** |

Table 12.1 – Time (in seconds) and number of choice points of $ECC_{DL,sum}$ and $ECC_{DL+,sum}$ to find a conceptual clustering that maximizes $Min_{size}$, $Min_{split}$ and $Min_{-diameter}$.

**Size, split and diameter criteria.** Table 12.1 displays the results of $ECC_{DL,sum}$ and $ECC_{DL+,sum}$ when maximizing $Min_{size}$, $Min_{split}$ and $Min_{-diameter}$. In this case, $k$ is not tightly constrained, and stronger propagation algorithms for constraining $k$ to be equal to the number of selected subsets (introduced in Section 11.2) are not useful.

$ECC_{DL,sum}$ and $ECC_{DL+,sum}$ often explore the same number of choice points, and when they do not explore the same number of choice points, the difference is lower than three. Hence, the stronger propagation of $DL+$ does not pay off and $ECC_{DL+,sum}$ is always slower than $ECC_{DL,sum}$. Times are similar for small instances but the gap increases for bigger instances. For instance, to maximize $Min_{size}$ for ERP7, $ECC_{DL,sum}$ needs 25 seconds while $ECC_{DL+,sum}$ needs almost 100 seconds.

**Frequency criterion.** We consider now the problem of finding a conceptual clustering that maximizes $Min_{frequency}$. In this case, $k$ is more tightly constrained as constraints on the frequency are strongly related to the number of clusters. Hence, we compare $ECC_{DL,sum}$ and $ECC_{DL+,sum}$ with $ECCK_{DL,p_2}$ (where $p_2 \in \{Basic, MD, MD+\}$) and $ECCK_{DL+,MD+}$ in Table 12.2.

$ECC_{DL,sum}$ explores much more choice points than $ECC_{DL+,sum}$. For instances, for ERP6, the number of choice points is reduced from 1212 to 7. Hence, for this problem, the advanced filtering $DL+$ pays off and $ECC_{DL+,sum}$ is able to solve all instances in less than 75 seconds while $ECC_{DL,sum}$ is not able to solve ERP7 within 1000 seconds.

Let us now compare $ECC_{DL,sum}$, which achieves a very simple filtering on $k$, with $ECCK_{DL,p_2}$ with $p_2 \in \{Basic, MD, MD+\}$ which achieve stronger filterings on $k$ (while using the same filtering $DL$ for propagating the exact cover constraint). $ECCK_{DL,Basic}$ and $ECC_{DL,sum}$ nearly always explore the same number of choice points and have very similar performances. $ECCK_{DL,MD}$ explores slightly less choice points than $ECCK_{DL,Basic}$ but, as the $MD$ propagation is expensive, this never pays off. $ECCK_{DL,MD+}$ explores much less choice points than $ECCK_{DL,MD}$ (for example, 9 instead of 1194 for ERP6), and this allows to strongly reduce time, especially for the

| | $ECC_{DL,sum}$ | | $ECC_{DL+,sum}$ | | $ECCK_{DL,Basic}$ | | $ECCK_{DL,MD}$ | | $ECCK_{DL,MD+}$ | | $ECCK_{DL+,MD+}$ | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | time | nodes | time | nodes | time | nodes | time | nodes | time | nodes | time | nodes |
| ERP1 | **0.1** | 7 | 0.1 | **5** | 0.1 | 7 | 0.1 | 7 | 0.1 | 6 | 0.1 | **5** |
| ERP2 | 0.5 | 159 | 0.2 | **6** | 0.6 | 159 | 0.9 | 159 | 0.2 | 9 | **0.1** | **6** |
| ERP3 | 0.6 | 106 | **0.2** | **6** | 0.6 | 106 | 0.7 | 83 | 0.2 | 12 | 0.2 | **6** |
| ERP4 | 1.5 | 138 | 0.4 | **9** | 1.5 | 138 | 2.5 | 138 | **0.3** | 12 | 0.4 | **9** |
| ERP5 | 16.8 | 346 | **1.0** | **7** | 17.2 | 346 | 28.5 | 277 | 4.2 | 23 | 1.0 | **7** |
| ERP6 | 90.7 | 1,212 | **1.9** | **7** | 87.7 | 1,212 | 161.6 | 1,194 | 2.1 | 9 | 2.3 | **7** |
| ERP7 | - | - | **23.9** | **7** | - | - | - | - | 421.1 | 109 | 34.7 | **7** |
| UCI1 | 0.2 | 34 | **0.2** | **9** | 0.2 | 34 | 0.2 | 30 | 0.2 | 15 | 0.2 | 13 |
| UCI2 | **0.6** | 16 | 0.8 | **7** | 0.6 | 14 | 1.5 | 14 | 1.5 | 11 | 1.5 | 11 |
| UCI3 | 2.4 | 27 | 2.3 | **6** | 2.5 | 27 | 5.1 | 23 | 2.8 | 9 | **2.3** | **6** |
| UCI4 | **1.6** | 18 | 1.9 | **10** | 1.6 | 17 | 2.4 | 17 | 2.4 | 14 | 2.2 | 11 |
| UCI5 | **1.3** | 6 | 2.1 | **4** | 1.3 | **4** | 2.5 | **4** | 2.2 | **4** | 2.2 | **4** |
| UCI6 | 382.7 | 175 | 74.4 | **17** | 371.7 | 175 | 461.2 | 175 | 299.7 | 121 | **67.6** | 18 |

Table 12.2 – Time (in seconds) and number of choice points of $ECC_{DL,sum}$, $ECC_{DL+,sum}$, $ECCK_{DL,Basic}$, $ECCK_{DL,MD+}$ and $ECCK_{DL+,MD+}$ to find a conceptual clustering that maximizes $Min_{frequency}$. '-' is reported when time exceeds 1,000s.

hardest instances. For instance, $ECCK_{DL,MD+}$ solves ERP7 in 421.1 seconds whereas this instance is not solved by $ECCK_{DL,Basic}$ nor $ECCK_{DL,MD}$ within 1000 seconds.

Finally, let us compare $ECC_{DL+,sum}$, $ECCK_{DL,MD+}$ with $ECCK_{DL+,MD+}$ to evaluate the interest of combining the strongest exact cover filtering ($DL+$) with the strongest filtering for $k$ ($MD+$). $ECCK_{DL+,MD+}$ explores slightly less choice points than $ECCK_{DL,MD+}$ for all instances but the two largest ones (ERP7 and UCI6) and, in this case the two approaches have very similar performance. For ERP7 and UCI6, the number of choice points is strongly decreased, and $ECCK_{DL+,MD+}$ is clearly faster than $ECCK_{DL,MD+}$. When comparing $ECC_{DL+,sum}$ with $ECCK_{DL+,MD+}$, we note that both approaches have very similar performance both with respect to the number of choice points and the time.

**Comparison with state-of-the-art declarative approaches.** In Table 12.3, we compare the best variant of our global constraint (*i.e.*, $ECC_{DL,sum}$ for all criteria but frequency, and $ECCK_{DL+,MD+}$ for frequency) with other declarative approaches. As pointed out in Section 9, the best performing approaches are different depending on the optimization criterion: For size and frequency, the best results are obtained by *FCP2* and *SetDec*, whereas for split and diameter, the best results are obtained by *FCP1*. Let us recall that *ILP* does not scale well for this problem and fails at solving many instances within a time limit of 1000 seconds (see Table 9.2).

For size and frequency, $ECC_{DL,sum}$ and $ECCK_{DL+,MD+}$ are the fastest approaches for all instances but the two largest ones, *i.e.*, $ERP7$ and $UCI6$ for which *FCP2* scales better. For these two instances, LCM needs 13.9 and 5.31 seconds, respectively, to enumerate all formal concepts, whereas *FCP2* is able to solve the complete problem in 4.4 and 4.3 seconds, respectively, thanks to ordering heuristics that allow it to quickly find the optimal solution without enumerating all formal concepts. However, on some smaller instances (such as UCI3, UCI4, and UCI5), FCP2 is much less efficient than $ECC_{DL,sum}$, and it is not able to solve UCI2 within the time limit.

| | Max($Min_{size}$) | | | Max($Min_{frequency}$) | | | Max($Min_{split}$) | | Min($Min_{-diameter}$) | |
|---|---|---|---|---|---|---|---|---|---|---|
| | $ECC_{DL,sum}$ | *FCP2* | *SetDec* | $ECCK_{DL+,MD+}$ | *FCP2* | *SetDec* | $EC_{DL,sum}$ | *FCP1* | $EC_{DL,sum}$ | *FCP1* |
| ERP1 | **0.1** | 0.4 | 0.2 | **0.1** | 1.0 | 0.2 | 0.1 | **0.0** | 0.1 | **0.0** |
| ERP2 | **0.1** | 0.4 | 0.3 | **0.1** | 1.0 | 0.6 | 0.1 | **0.0** | 0.1 | **0.0** |
| ERP3 | **0.1** | 0.6 | 0.3 | **0.2** | 3.2 | 2.5 | 0.1 | **0.0** | 0.1 | **0.0** |
| ERP4 | **0.2** | 1.6 | 0.8 | **0.4** | 4.5 | 0.9 | 0.2 | **0.0** | 0.2 | **0.0** |
| ERP5 | **0.5** | 0.8 | 1.7 | **1.0** | 4.4 | 26.0 | 0.6 | **0.1** | 0.5 | **0.1** |
| ERP6 | **1.6** | 2.5 | 4.5 | **2.3** | 5.7 | 241.4 | 0.9 | **0.1** | 1.6 | **0.1** |
| ERP7 | 31.1 | **4.4** | 129.5 | 34.7 | **23.0** | - | 6.9 | **0.2** | 49.6 | **0.2** |
| UCI1 | **0.1** | 0.8 | 0.2 | **0.2** | 5.1 | 1.0 | 0.1 | **0.1** | **0.1** | 0.1 |
| UCI2 | **0.3** | - | 5.0 | **1.5** | 536.8 | 29.8 | **0.3** | 11.0 | **0.2** | 11.3 |
| UCI3 | **0.5** | 55.7 | 4.6 | **2.3** | 70.1 | 63.7 | **0.7** | 1.7 | **0.5** | 1.7 |
| UCI4 | **3.3** | 16.3 | 32.9 | **2.2** | 8.9 | 19.1 | 0.7 | **0.2** | 3.2 | **0.2** |
| UCI5 | **1.0** | 33.2 | 15.1 | **2.2** | - | 76.6 | **1.9** | 4.6 | **1.0** | 4.5 |
| UCI6 | 17.0 | **4.3** | 111.3 | 67.6 | **8.7** | - | 17.9 | **0.2** | 28.2 | **0.2** |

Table 12.3 – Comparison of the time (in seconds) spent by $ECC_{DL,sum}$ (for all criteria but frequency) and $ECCK_{DL+,MD+}$ (for the frequency) with the best performing declarative approaches for each criterion, *i.e.*, *FCP2* and *SetDec* for size, and frequency, and *FCP1* for split and diameter. '-' is reported when time exceeds 1,000s.

For split and diameter, *FCP1* is faster than $ECC_{DL,sum}$ on many instances, especially for the two largest instances ERP7 ad UCI6.

## 12.3 Multi criteria optimization

When analyzing the number of clusters reported in Table 9.3, we note that we obtain solutions with a large number of clusters (close to $\#\mathcal{T} - 1$) when we optimize $Min_{size}$ or $Min_{-diameter}$ whereas we obtain solutions with very few clusters (close to 2) when we optimize $Min_{frequency}$ or $Min_{split}$. As explained in Section 6.2, this comes from the fact that $Min_{frequency}$ and $Min_{split}$ criteria are conflicting with $Min_{size}$ and $Min_{-diameter}$ criteria.

In this case, we may search for a set of solutions that represent different compromises between these conflicting criteria by computing the Pareto front of non-dominated solutions. In this section, we evaluate scale-up properties of our CP models for computing this Pareto front for two pairs of conflicting criteria, *i.e.*, $Min_{frequency}$ and $Min_{size}$ (denoted (frequency,size)) and $Min_{split}$ and $Min_{-diameter}$ (denoted (split,diameter)).

### 12.3.1 Comparison of propagation algorithms of *exactCoverCost*

Let us first compare the different propagation algorithms introduced for our global constraint. For (split,diameter), $k$ is not tightly constrained (it is only constrained to belong to $[2, \#\mathcal{T}]$). Therefore, in this case we only consider the two variants that achieve the simplest filtering on $k$, *i.e.*, $ECC_{DL,sum}$, and $ECC_{DL+,sum}$. For (frequency,size), $k$ is more constrained (because of the relation between the frequency and the number of clusters). Therefore, in this case we also consider variants that achieve stronger filterings on $k$, *i.e.*, $ECCK_{DL,MD+}$, and $ECCK_{DL+,MD+}$. We only report results with $MD+$ because $MD$ has rather similar results and *Basic* filters much less choice points than $MD+$.

| | (Split,Diameter) | | | | | (Frequency,Size) | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | $ECC_{DL,sum}$ | | $ECC_{DL+,sum}$ | | | $ECC_{DL,sum}$ | | $ECC_{DL+,sum}$ | | $ECCK_{DL,MD+}$ | | $ECCK_{DL+,MD+}$ | |
| | #s | time | nodes | time | nodes | #s | time | nodes | time | nodes | time | nodes | time | nodes |
| ERP1 | 1 | **0.2** | **48** | 0.3 | **48** | 7 | 0.5 | 1,077 | **0.3** | 448 | 0.4 | 1,055 | 0.3 | **428** |
| ERP2 | 5 | 0.6 | **89** | **0.6** | 92 | 9 | 10.2 | 6,284 | 0.9 | 571 | 12.3 | 5,994 | **0.9** | **486** |
| ERP3 | 2 | 0.5 | 63 | **0.4** | **62** | 10 | 13.9 | 6,619 | 1.2 | 692 | 19.8 | 5,556 | **1.1** | **608** |
| ERP4 | 2 | **0.9** | **88** | 1.0 | 89 | 13 | 109.7 | 68,710 | **3.4** | 1,230 | 157.0 | 65,726 | 3.5 | **1,068** |
| ERP5 | 3 | **2.3** | 106 | 2.3 | **100** | 13 | 842.3 | 27,280 | **13.7** | 1,144 | - | - | 16.1 | **1,099** |
| ERP6 | 3 | **4.8** | **108** | 8.0 | 111 | 15 | - | - | **25.8** | 1,423 | - | - | 30.5 | **1,345** |
| ERP7 | 2 | **135.9** | **164** | 198.0 | 173 | 17 | - | - | **987.7** | **4,559** | - | - | - | - |
| UCI1 | 3 | **0.4** | **78** | 0.4 | 81 | 13 | 3.8 | 6,498 | 1.1 | 1,291 | 4.8 | 7,713 | **1.1** | **1,140** |
| UCI2 | 3 | **1.4** | 560 | 2.2 | **549** | 12 | - | - | - | - | - | - | **280.0** | **232,871** |
| UCI3 | 1 | **1.0** | **215** | 2.0 | **215** | 11 | - | - | **335.7** | **17,541** | - | - | 499.2 | 50,879 |
| UCI4 | 5 | **4.4** | 319 | 15.9 | **297** | - | - | - | - | - | - | - | - | - |
| UCI5 | 2 | **4.7** | 428 | 5.4 | **400** | 8 | - | - | **130.2** | 103,361 | - | - | 136.5 | **103,290** |
| UCI6 | 4 | 536.4 | **433** | **316.8** | 437 | - | - | - | - | - | - | - | - | - |

Table 12.4 – Time (in seconds) and number of choice points needed by $ECC_{DL,sum}$ and $ECC_{DL+,sum}$ for (split,diameter) and $ECC_{DL,sum}$, $ECC_{DL+,sum}$, $ECCK_{DL,MD+}$ and $ECK_{DL+,MD+}$ for (frequency,size) to compute the set of non-dominated solutions using the static method of [Was+80]. $\#s$ gives the number of non-dominated solutions. '-' is reported when time exceeds 1,000s.

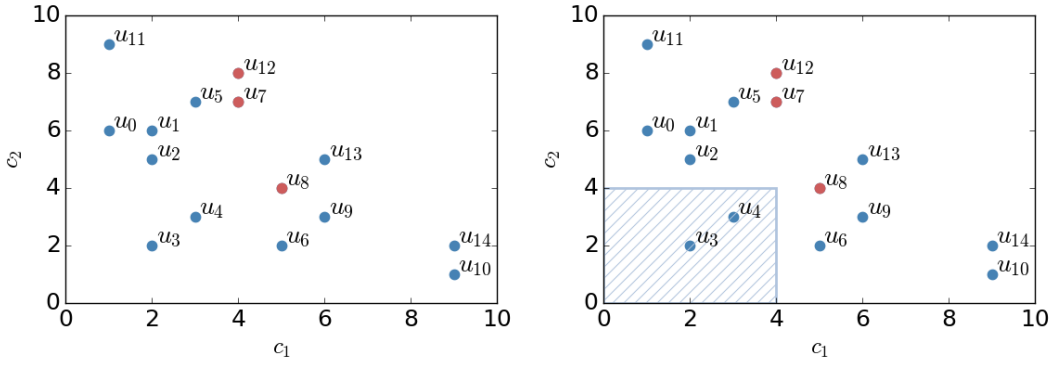
To compute the Pareto front, we use the static method of [Was+80] described in Section 5.6.2. More precisely, we consider as first criterion to maximize $Min_{split}$ (resp. $Min_{frequency}$), *i.e.*, it corresponds to $obj_1$ in Algorithm 4, when optimizing (split,diameter) (resp. (frequency, size)). If there is almost no difference between choosing $Min_{split}$ and $Min_{-diameter}$ as first criterion, considering $Min_{frequency}$ as first criterion to optimize has a significant impact on the time, and significantly reduces solving times compared when we consider $Min_{size}$ as first criterion. This may come from the side constraints we add on $Min_{frequency}$ and the objective strategy we use.

We display in Table 12.4 the CPU time in seconds and the number of choice points needed for each considered variant of *exactCoverCosts*.

For all approaches, the Pareto front for (split,diameter) is smaller and also easier to compute than for (frequency,size). This may come from the fact that frequency and size measures are very conflicting criteria (formal concepts with large frequencies usually have small sizes, and vice versa), whereas (split,diameter) are less conflicting criteria.

For (split,diameter), $ECC_{DL,sum}$ is often faster than $ECC_{DL+,sum}$ because $DL+$ never reduces significantly the number of choice points.

For (frequency,size), $DL+$ significantly reduces the number of choice points, compared to $DL$ for all instances, and $ECC_{DL+,sum}$ is able to solve four more instances than $EC_{DL,sum}$ within the time limit. $MD+$ also reduces the number of choice points, compared to $MD$, but the reduction is less drastic: $ECC_{DL,MD+}$ and $ECC_{DL+,MD+}$ explore less choice points than $ECC_{DL,sum}$ and $ECC_{DL+,sum}$, respectively, for all instances but UCI1 (for $ECC_{DL,*}$) and UCI3 (for $ECC_{DL+,*}$). $ECC_{DL+,MD+}$ solves $UCI2$ in 280 seconds whereas no other approach can solve it in less than 1,000 seconds. However, $MD+$ considerably degrades the performance of $ECC_{DL+,MD+}$ for $UCI3$. Most of the time, $EC_{DL+,MD+}$ is the approach that explores the smallest number of choice points while $EC_{DL+,sum}$ is the fastest approach.




(a) Exact Cover $C = \{u_7, u_8, u_{12}\}$.

(b) Subsets of $P$ dominated by $C$.

Figure 12.2 – Subsets dominated by an exact cover.

### 12.3.2 New dynamic approach

We have described two CP-based approaches to compute the Pareto front of non-dominated solutions in Section 5.6.2. In this section, we show how to improve the dynamic method of Gavanelli *et al.* [Gav02] when using it to solve a multi-criteria exact cover problem $(S, P)$.

We assume that there are $n > 1$ utility measures such that, for each $i \in [1, n]$, $c_i(u)$ is the utility of the subset $u \in P$, $Min_i$ is an integer variable to be maximized, and this variable is constrained to be equal to $\min_{u \in C} c_i(u)$ (where $C$ is the set of selected subsets).

During the search process, when an exact cover $C \subseteq P$ is found, we know that if an exact cover $C'$ contains a subset $u \in P$ which is dominated by $C$, then $C'$ is also dominated by $C$. In other words, once we have found an exact cover $C \subseteq P$, we can discard any subset $u$ such that $\forall i \in [1, n], c_i(u) \leq \min_{u \in C} c_i(u)$ before searching for other exact covers.

**Example 12.1.** We display in Figure 12.2a an exact cover instance with two utility measures: each point $(x, y)$ corresponds to a subset $u$ such that $x = c_1(u)$ and $y = c_2(u)$. Let us assume that $C = \{u_7, u_8, u_{12}\}$ is an exact cover. When $C$ is found, we have $Min_1 = 4$ and $Min_2 = 4$. Therefore, we know that any exact cover that contains a subset in the area $[0, 4] \times [0, 4]$ (displayed in blue) is dominated by $C$ and we can remove *true* from the domains of $isSelected_{u_3}$ and $isSelected_{u_4}$.

Hence, we propose to extend the dynamic approach of Gavanelli *et al.* [Gav02]. More precisely, each time a solution $I$ is found, we dynamically add two constraints. The first constraint is the constraint used in [Gav02] to prevent the search from computing a solution dominated by $I$, *i.e.*,

$$\bigvee_{i \in [1,\ldots,n]} Min_i > I[Min_i].$$

The second constraint is a new constraint which prevents the search from selecting a subset dominated by $I$, *i.e.*,

$$\forall u \in P, \bigwedge_{i \in [1,\ldots,n]} c_i(u) \leq I[Min_i] \Rightarrow isSelected_u = false$$

This second constraint immediately filters the domains of *isSelected* variables associated with subsets which are dominated by $I$, whereas the first constraint does not filter any domain when all upper bounds of $Min_i$ variables are greater than $I[Min_i]$.

**Example 12.2.** In our previous example, domains of *isSelected* variables are not immediately filtered when adding the constraint

$$Min_1 > 4 \vee Min_2 > 4$$

Indeed, when both upper bounds of $Min_1$ and $Min_2$ are greater than 4, this disjunctive constraint is not propagated. It is propagated only when the upper bound of one of these variables becomes lower than or equal to 4: When $Min_1$ (resp. $Min_2$) becomes lower than or equal to 4, the solver propagates the constraint $Min_2 > 4$ (resp. $Min_1 > 4$), and this propagation assigns to *false* every variable $isSelected_{u_i}$ such that $c_2(u_i) \leq 4$, *i.e.*, $u_3$, $u_4$, $u_5$, $u_8$, $u_9$, $u_{10}$, and $u_{14}$ (resp. $c_1(u_i) \leq 4$, *i.e.*, $u_0$, $u_1$, $u_2$, $u_3$, $u_4$, $u_5$, $u_7$, $u_{11}$, and $u_{12}$).

**Implementation.** We have implemented this extension for the CP model that uses our new global constraint *exactCoverCost*, and also for our CP model that uses the *SetDec* decomposition introduced in Section 9.2.

For the CP model that uses *exactCoverCost*, each time a solution is found, we dynamically add the clause $isSelected_u = False$ for each subset $u \in P$ which is dominated by the solution.

For the CP model that uses *SetDec*, we channel the set variable $C$ with *isSelected* boolean variables such that $\forall u \in P, isSelected_u = true \Leftrightarrow u \in C$ and we dynamically add the clause $isSelected_u = False$ for each subset $u \in P$ which is dominated by the solution. Another possibility is to directly add the constraint $u \notin C$, without introducing *isSelected*. However, this alternative is less efficient.

**Experimental evaluation.** Let us compare the three possibilities for computing the Pareto front of non dominated solutions:

- The static approach of [Was+80] described in Section 5.6.2 and denoted *Static*;
- The dynamic approach of [Gav02] described in Section 5.6.2 and denoted *Dynamic*;
- Our extension of *Dynamic* introduced in this section and denoted *Extended*.

For *Dynamic* and *Extended*, we adapt the ordering heuristic introduced by Knuth: We still search for the set of all elements $a \in S_C$ such that $\#cover_C(a)$ is minimal, but instead of selecting a subset that covers one of these elements and maximizes the utility measure, we select a subset that covers one of these elements and maximizes the number of dominated subsets in $P$.

Experimental results are reported in Table 12.5 when using the best performing propagation algorithms according to the experimental comparison reported in Section 12.3.1, *i.e.*, $ECC_{DL,sum}$ when the criteria to optimize are $Min_{split}$ and $Min_{-diameter}$, and $ECCK_{DL+,MD+}$ when the criteria to optimize are $Min_{size}$ and $Min_{frequency}$.

Our extended method explores less choice points and is clearly faster than *Dynamic*. In particular, it is able to solve four more instances than *Dynamic*.

| | #s | *Static* | | | *Dynamic* | | | *Extended* | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | time | nodes | nbSol | time | nodes | nbSol | time | nodes | nbSol |
| (split,diameter) with $ECC_{DL,sum}$ | | | | | | | | | | |
| ERP1 | 1 | 0.2 | **48** | 2 | 0.1 | 191 | 3 | **0.1** | 127 | 3 |
| ERP2 | 5 | 0.6 | **89** | 10 | 2.0 | 631 | 8 | **0.3** | 193 | 8 |
| ERP3 | 2 | 0.5 | 63 | 4 | 2.3 | 459 | 2 | **0.2** | **62** | 2 |
| ERP4 | 2 | 0.9 | 88 | 4 | 7.7 | 724 | 2 | **0.3** | **87** | 2 |
| ERP5 | 3 | 2.3 | 106 | 6 | 157.2 | 3,488 | 3 | **0.9** | **85** | 3 |
| ERP6 | 3 | 4.8 | **108** | 6 | 172.1 | 2,813 | 6 | **1.7** | 337 | 6 |
| ERP7 | 2 | 135.9 | **164** | 4 | - | - | - | **42.9** | 452 | 4 |
| UCI1 | 3 | 0.4 | **78** | 6 | 0.2 | 203 | 7 | **0.2** | 151 | 7 |
| UCI2 | 3 | 1.4 | **560** | 6 | **1.3** | 2,523 | 11 | 3.7 | 2,515 | 11 |
| UCI3 | 1 | **1.0** | **215** | 2 | 2.1 | 395 | 3 | 1.7 | 371 | 3 |
| UCI4 | 5 | **4.4** | **319** | 10 | 571.2 | 12,457 | 12 | 4.4 | 805 | 12 |
| UCI5 | 2 | **4.7** | **428** | 4 | 66.8 | 2,732 | 5 | 5.4 | 794 | 5 |
| UCI6 | 4 | 536.4 | **433** | 8 | - | - | - | **151.5** | 563 | 10 |
| (frequency,size) with $ECCK_{DL+,MD+}$ | | | | | | | | | | |
| ERP1 | 7 | 0.3 | 428 | 14 | **0.2** | 226 | 18 | 0.3 | **176** | 18 |
| ERP2 | 9 | **0.9** | **486** | 18 | 2.3 | 2,785 | 26 | 2.4 | 1,739 | 27 |
| ERP3 | 10 | **1.1** | **608** | 20 | 1.9 | 1,686 | 31 | 1.4 | 1,197 | 28 |
| ERP4 | 13 | **3.5** | 1,068 | 26 | 8.5 | 1,160 | 47 | 8.2 | **981** | 47 |
| ERP5 | 13 | **16.1** | **1,099** | 26 | 74.4 | 17,023 | 48 | 48.6 | 3,993 | 47 |
| ERP6 | 15 | **30.5** | **1,345** | 30 | 172.5 | 6,289 | 68 | 155.8 | 2,190 | 70 |
| ERP7 | 17 | **1,047.4** | **4,043** | 34 | - | - | - | - | - | - |
| UCI1 | 13 | **1.1** | 1,140 | 26 | 2.5 | 1,449 | 67 | 1.4 | **719** | 69 |
| UCI2 | 12 | **280.0** | **232,871** | 24 | - | - | - | - | - | - |
| UCI3 | 11 | **499.2** | **50,879** | 22 | - | - | - | 1,461.0 | 205,785 | 99 |
| UCI4 | 14 | **1,476.3** | **298,832** | 28 | - | - | - | 2,919.7 | 689,217 | 66 |
| UCI5 | 8 | **136.5** | **103,290** | 16 | - | - | - | - | - | - |
| UCI6 | - | - | - | - | - | - | - | - | - | - |

Table 12.5 – Comparison of *Static*, *Dynamic*, and *Extended* to compute the Pareto front for (split,diameter) with $ECC_{DL,sum}$, and for (frequency,size) with $ECCK_{DL+,MD+}$: #s gives the number of non-dominated clusterings, Time gives the CPU time in seconds (or '-' when time exceeds 3,600 seconds), nodes gives the number of choice points, and nbSol gives the number of solutions found.

For (split,diameter), *Static* is competitive with *Extended* for the small instances, but it is outperformed for larger instances such as ERP6, ERP7, or UCI6. This may come from the fact that the number of solutions computed by *Extended* is often close to the number of non dominated solutions: nbSol is equal to $\#s$ for three instances, and never greater than $4 * \#s$. This means that ordering heuristics are able to guide the search towards solutions that often belong to the Pareto front. All these solutions are computed by solving a single enumeration problem within a single search. As a comparison, *Static* always computes $2\#s$ solutions, and each of these solutions is obtained by solving a new optimization problem.

On (frequency,size), *Static* is the fastest approach for all instances but ERP1, and it scales much better: It is able to solve all instances but UCI6 in less than one hour, whereas *Extended* reaches the CPU time limit for ERP7, UCI2, UCI5, and UCI6.

| | (split,diameter) | | | (size,frequency) | | |
|---|---|---|---|---|---|---|
| Strategy | $ECC_{DL,sum}$ *Extended* | *SetDec Extended* | *ILP Static* | $ECCK_{DL+,MD+}$ *Static* | *SetDec Static* | *ILP Static* |
| ERP1 | **0.1** | 0.2 | 0.5 | **0.3** | 4.8 | 1.0 |
| ERP2 | **0.3** | 0.8 | 1.4 | **0.9** | 23.4 | 3.8 |
| ERP3 | **0.2** | 0.6 | 1.5 | **1.1** | 129.1 | 6.3 |
| ERP4 | **0.3** | 4.1 | 20.1 | **3.5** | 946.3 | 39.4 |
| ERP5 | **0.9** | 16.1 | 27.4 | **16.1** | - | 89.0 |
| ERP6 | **1.7** | 130.3 | 268.1 | **30.5** | - | 450.4 |
| ERP7 | **42.9** | - | - | **1,047.4** | - | - |
| UCI1 | **0.2** | 0.7 | 0.9 | **1.1** | 128.8 | 8.1 |
| UCI2 | **3.7** | 2,055.4 | 231.8 | **280.0** | - | - |
| UCI3 | **1.7** | - | 645.1 | **499.2** | - | - |
| UCI4 | **4.4** | 202.0 | - | **1,476.3** | - | - |
| UCI5 | **5.4** | - | - | **136.5** | - | - |
| UCI6 | **151.5** | - | - | - | - | - |

Table 12.6 – Time (in seconds) required by $ECC_{DL,sum}$ and *SetDec* and *ILP* to compute the set of non-dominated solutions for (split,diameter) and $ECCK_{DL+,MD+}$, *SetDec* and *ILP* for (frequency,size). We report '-' when time exceeds 3,600s. The third line precises the strategy (*Extended* or *Static*) used to compute the Pareto front.

This may come from the fact that the number of solutions computed by *Extended* is often much larger than the number of non dominated solutions. For example, for UCI3, *Extended* computes 99 solutions, whereas the Pareto front only contains 11 non dominated solutions. For this instance, *Static* solves 22 optimization problems, and it is three times as fast as *Extended*.

As a conclusion, *Dynamic* is outperformed by *Extended*, and *Extended* and *Static* are complementary: *Extended* is more efficient for (split,diameter), and *Static* for (frequency,size).

### 12.3.3 Comparison with state-of-the-art declarative approaches

Finally, let us compare our global constraint with other declarative approaches. In this study, we do not report results of the full CP approaches (*FCP1* and *FCP2*) because they hardly scale. For example, for the (size,frequency) criteria, *FCP2* is not able to solve ERP1 in less than one day using the *Static* strategy, whereas this instance is solved in less than one second with our global constraint.

Table. 12.6 compares the best propagation algorithms of our new global constraint (*i.e.*, $ECC_{DL,sum}$ for (split,diameter) and $ECCK_{DL+,MD+}$ for (frequency,size)) with *SetDec* and *ILP*. We used the following strategies for computing the Pareto front with CP-based approaches: For (split,diameter), we use *Extended* for $ECC_{DL,sum}$ and *SetDec*; For (frequency,size), we use *Static*. For *ILP*, we always use the *Static* strategy.

For (split,diameter), our global constraint $ECC_{DL,sum}$ is significantly faster than *ILP* and *SetDec*: It is able to solve all instances whereas *SetDec* and *ILP* fail at solving four instances. For some instances, $ECC_{DL,sum}$ is two orders of magnitude faster than *SetDec* and *ILP*. For example, it can solve $UCI3$ in less than two seconds whereas *ILP*

needs more than ten minutes and *SetDec* cannot solve it within the time limit of one hour.

For (frequency,size), $ECCK_{DL+,sum}$ is also significantly faster than *ILP* and *SetDec*: It is able to solve all instances but UCI6 whereas *SetDec* and *ILP* fail at solving eight and six instances, respectively. In particular, $ECCK_{DL+,MD+}$ is the only approach that is able to solve ERP7, UCI2, UCI3, UCI4, and UCI5. Finally, *ILP* scales better than *SetDec* and solves two instances that *SetDec* cannot.

## 12.4 Discussion

We have experimentally evaluated our new global constraints on various conceptual clustering problems. When the number of clusters $k$ is fixed, our approach does not always scale well, especially for the $Min_{size}$ criterion and, in some cases, it is outperformed by *ILP*. When $k$ is not fixed, our approach scales well and it is able to solve all mono-criterion optimization problems rather quickly: Most instances are solved in less than one second, and the hardest instance is solved in less than 100 seconds. For many instances, it is the best performing approach, though it is outperformed by full CP approaches (*FCP1* or *FCP2*) for some instances.

We extended the Gavanelli *et al.* approach to compute the Pareto front of non-dominated solution when considering two optimization criteria. This new approach always improves Gavanelli *et al.* approach, and it is faster than the static approach of [Was+80] for (split,diameter) criteria, whereas it is slower for the (frequency,size) criteria. Our new global constraint allows us to compute the complete Pareto front for all instances but one, and it scales much better than *ILP*.

Now we have an efficient method to solve conceptual clustering problems, we experiment it in the next chapter in our applicative context to extract relevant parts of configuration.

# Chapter 13

# Application to ERP customization

## Contents



Now we have an efficient declarative approach to solve conceptual clustering problems, we experiment our approach on a *business unit* of *Copilote*. As explained in Chapter 4, we aim at extracting relevant parts of configurations that correspond to business logic requirements. To do this, we use conceptual clustering to identify groups of configuration parts that implement a same business logic: Each cluster corresponds to a part of configuration, *i.e.*, the subset of parameter values shared by all the configurations of the cluster.

From an applicative point of view, the challenge is to identify relevant configuration parts, *i.e.*, configuration parts that have a business logic meaning and are reusable for new configurations. To improve the quality of the configuration parts we extract, we propose to interact with experts after each extraction step in order to integrate their feedback: At each iteration, we extract the most relevant configuration parts according to some given constraints and optimization criteria, and we ask an expert system integrator to discard parts with no functional meaning; This feedback is used to update the constraints and criteria before starting a new mining process.

In this chapter, we describe the business unit which is used for our proof of concept in Section 13.1. In Section 13.2, we focus on how to measure the relevancy of configuration parts and we define a new utility measure based on correlation. In Section 13.3, we present the main feedbacks expressed by the expert when he used our tool on the use case described in Section 13.1, and we describe how we adapted our CP model to integrate these feedbacks. Finally, we give an overview of the complete toolkit we developed to interactively mine configuration parts in Section 13.4.

## 13.1 Use case

For the proof of concept of our approach, we focus on one business unit of the production module called *Plan production*. As its name implies, this business unit corresponds to a functionality of *Copilote* that is used to plan the production of a company. To plan the production as well as possible, a customer may need to visualize many information such as the state of the storage, the incoming sales order, the sales forecasts and the current planning of production. All these information may be computed in many different ways according to the business rules of the customer. Moreover, the requirements may change when planning the production of a family of products to another. That is why a business object is dedicated to the configuration of the production planning and a single customer may have several configurations, *i.e.*, several instances of this object.

The main configuration options of the production planning tool are:

- the planning period, which may be a day, week, month or year, and which usually depends on the volume to produce and on the storage life of the product;
- the computation of the quantity to produce, which may depend on existing sale orders, or on sale forecasts, for example;
- the computation of the material needs, which depend on the production planning;
- the computation of the storage.

Only a part of the customers of Infologic need the production planning tool. We collected 1531 existing configurations and focused on twelve parameters of the production planning configuration business object. After translating these twelve parameters into binary items, we obtained 25 items. Therefore, in our proof of concept our transactional database has 1531 transactions and 25 items.

## 13.2 Relevancy measures

Before experimenting conceptual clustering on our use case, we have to determine how to characterize the relevancy of a configuration part, which is the main issue of our data mining process. From an applicative point of view, a configuration part is relevant if the parameter setting associated with it corresponds to a business logic requirement.

We have introduced four classical measures associated with formal concepts in Section 6.2, *i.e.*, frequency size, diameter, and split. We have used these utility measures to evaluate scale-up properties of our CP models. However, they are not really meaningful in our applicative context. In particular, the frequency corresponds to the number of customers that use it, and ERP experts have told us that there is no relation between the frequency and the relevancy of a configuration part: Frequent configuration parts may be as useful as rare ones, as long as they correspond to a business logic requirement. However, when interpreting the meaning of a configuration part, the frequency and the detail of the different customers that use a concept are very helpful information for system integrators.

The size of a configuration part corresponds to the number of parameters and it mainly affects the way an expert understands its functional meaning. Obviously, concepts with very small sizes may lack of information to be relevant whereas too long

concepts may be hard to interpret. Hence, we usually add constraints to bound the size of the configuration parts.

According to discussions we had with *Copilote* experts, the relevancy of a configuration part mainly depends on the correlation between the parameter values of a part. This correlation denotes a mutual relationship between parameter values. Typically, a configuration part is more relevant if its parameter values are usually grouped together, *i.e.*, they are highly correlated.

As explained in [Nov+09], different research communities have proposed many techniques to find comprehensible itemsets or models in data. Measuring the correlation between items has been widely studied in the data mining community, particularly in association rule mining. We introduce three well-known measures based on correlation which are used to assess the relevancy of association rules and we explain how we apply them on formal concepts.

**Association rules.** Association rules have been defined in [Agr+93] by Agrawal et al. as implications between two itemsets. We keep on using the transactional database terminology to have no ambiguities in the following definitions, *i.e.*, $\mathcal{T}$ is a set of transactions, $\mathcal{I}$ a set of items, and $\mathcal{R} \subseteq \mathcal{T} \times \mathcal{I}$ a binary relation between transactions and items. For more details, we refer the reader to [Nov+09] that presents a unified framework for descriptive rules.

**Definition 13.1** (Association rule)**.** Let be $X \subseteq \mathcal{I}$ and $Y \subseteq \mathcal{I}$ two itemsets. $r = X \rightarrow Y$ is an association rule. $X$ is the *antecedent* of $r$ and $Y$ its *consequent*.

**Definition 13.2** (Support)**.** The support of an itemset $Z \subseteq \mathcal{I}$ is the number of transactions that contain all items of $Z$, *i.e.*, $supp(Z) = \#\{t \in \mathcal{T} | Z \subseteq itemset(t)\}$ (in other words, if $Z$ is the itemset of a formal concept $c$, then the support of $Z$ corresponds to the frequency of $c$).

The support of an association rule $r = X \rightarrow Y$ is the support of $X \cup Y$, *i.e.*, $supp(r) = supp(X \cup Y)$.

**Example 13.3.** Let us consider the database displayed in Table 6.1. The rule $r_1 = \{i_2, i_3\} \rightarrow \{i_8\}$ has a support of 1 because only $t_4$ contains $i_2$, $i_3$ and $i_8$.

**Interestingness measures.** According to authors of [Héb+07], an interestingness measure is a function which assigns a numerical value to an association rule according to its quality. Many interestingness measures are based on the supports of the antecedent and the consequent of the rule. We consider two well-known measures called *RAcc* and *WRAcc*.

**Definition 13.4** (Relative accuracy (*RAcc*))**.** Let $r = X \rightarrow Y$ be an association rule. The *Relative Accuracy* of $r$ is

$$RAcc(r) = \frac{supp(X \cup Y)}{supp(X)} - \frac{supp(Y)}{\#\mathcal{T}}.$$

As explained in [Lav+99], *RAcc* gives the accuracy gain of $r$ compared to the rule $\emptyset \rightarrow Y$. A rule is interesting if it improves upon this default accuracy. However, the problem of the relative accuracy is that it is easy to obtain high relative accuracy with very specific rule, *i.e.*, rules with a low support since $supp(Y)$ is very low.

That is why a weighted variant, called *WRAcc*, was proposed in [Lav+99].

**Definition 13.5** (Weighted Relative Accuracy (*WRAcc*))**.** Let $r = X \rightarrow Y$ be an association rule. The *Weighted Relative Accuracy* of $r$ is:

$$WRAcc(r) = \frac{supp(X)}{\#\mathcal{T}} \cdot (\frac{supp(X \cup Y)}{supp(X)} - \frac{supp(Y)}{\#\mathcal{T}}) = \frac{supp(X)}{\#\mathcal{T}} * RAcc(r).$$

The *WRAcc* measure offers a trade off between the relative accuracy and the generality of the rule, *i.e.*, its support.

**Example 13.6.** For instance, let us consider a dataset with $\#\mathcal{T} = 100$ transactions such that $supp(Y) = supp(X) = 5$, $supp(W) = supp(Z) = 70$, $supp(X \cup Y) = 2$ and $supp(W \cup Z) = 65$. We have:

- $RAcc(X \rightarrow Y) = 0.35$ and $RAcc(W \rightarrow Z) = 0.32$
- $WRAcc(X \rightarrow Y) = 0.02$ and $WRAcc(W \rightarrow Z) = 0.22$

The *WRAcc* measure clearly favors $W \rightarrow Z$ which better matches with our intuition of interestingness of a rule.

Hence, we choose to use the *WRAcc* measure to measure the correlation between parameter instances of a configuration part.

**Application to formal concepts.** We propose to extend the *WRAcc* measure to formal concepts, in order to evaluate how correlated is each item of a concept with the other items of the concept.

**Definition 13.7** (*WRAcc* of a formal concept)**.** Let $c = (T, I)$ be a formal concept. If $\#I = 1$, then $WRAcc(c) = -\infty$. Otherwise:

$$WRAcc(c) = \frac{\sum_{i \in I} WRAcc(\{i\} \rightarrow I \setminus \{i\})}{\#I}.$$

In other words, the *WRAcc* of a formal concept that has only one item is defined as $-\infty$ because it does not bring any business logic. Otherwise, its *WRAcc* is the average correlation of every item $i \in I$ with the other items in $I$, *i.e.*, the average *WRAcc* of every association rule $\{i\} \rightarrow I \setminus \{i\}$.

Let us now define our new utility measure for evaluating the interest of a conceptual clustering.

**Definition 13.8** ($Min_{WRAcc}$ utility measure)**.** Let $P = \{c_1, \ldots, c_k\}$ be a conceptual clustering. We define: $Min_{WRAcc}(P) = \min_{c_i \in P} WRAcc(c_i)$.

When maximizing this utility measure, we favor clusterings with formal concepts such that every item is correlated with the other items in the intent of the concept.

**Example 13.9.** Let us consider the transactional database of Table 6.3.

$P = \{c_5, c_8, c_7, c_{10}\}$ is a conceptual clustering. We have:

$WRAcc(c_5) = \frac{1}{25}$ $\quad WRAcc(c_8) = \frac{1}{25}$

$WRAcc(c_7) = \frac{-4}{25}$ $\quad WRAcc(c_{10}) = \frac{-1}{25}$

Therefore, we have: $Min_{WRAcc}(P) = -0.03$.

## 13.3 Feedbacks and improvements

We considered the use case described in Section 13.1, and experimented our approach that iteratively extracts configuration parts and interacts with an expert integrator to integrate his feedback as described in Section 4.3.

More precisely, we developed a tool with an interface that allows the user (*i.e.* the expert) to apply conceptual clustering on the configurations of the business unit *plan production*. Initially, the user can choose an optimization criterion $Min_u$ with $u \in \{frequency, size, -diameter, split, WRAcc\}$. We display the resulting configuration parts and their associated properties:

- the size, the frequency, the split, the WRAcc and the diameter of the formal concept;
- the list of customers that use the configuration part.

The expert can qualify each configuration part by associating a description if it fulfills a requirement or by discarding the part with a commentary if it is not relevant. He may also give us some feedback to improve the mining process and find more relevant configuration parts. In this section, we present the main feedbacks we obtained and how we used them to improve the mining step.

### 13.3.1 Properties of the formal concepts.

**Feedback.** The first feedback of our expert concerns properties of extracted configuration parts. They want to avoid:

- too specific formal concepts which have a very low frequency and a high size because they are very hard to interpret;
- too short formal concepts with one or two parameters.

**Proposed solution.** To solve this problem, we allow the user to add thresholds on both minimal and maximal frequency or size of the configuration parts. When using our $exactCover_{S,P}$ global constraint to solve the mining problem, this amounts to removing from $P$ every formal concept that does not satisfy these threshold constraints.

### 13.3.2 Pivot items

**Feedback.** When configuring the business unit *Plan production*, *Copilote* integrators usually start by configuring the periodicity parameter, and the first question they ask to the customer aims at setting this parameter. This is a typical example of expert knowledge: An expert knows that this parameter is discriminant for the rest of the configuration. Therefore, the second feedback of our expert concerns this parameter: A configuration part that does not contain an item corresponding to a value for this parameter is not relevant because it is a key parameter to both configure the business unit and interpret configuration parts.

**Proposed solution.** To solve this problem, we allow the user to specify a list of *pivot parameters*, *i.e.*, parameters that must appear in configuration parts. Let $L_{pivot}$ be this list and, for each pivot parameter $p_i \in L_{pivot}$, let $items(p_i) \subseteq \mathcal{I}$ be the set of items associated with this parameter (each item in $items(p_i)$ corresponds to a different value that may be assigned to $p_i$). When searching for conceptual clusterings, we add the constraint that each selected formal concept must contain exactly one item in $items(p_i)$, for each pivot parameter $p_i \in L_{pivot}$.

When using our $exactCover_{S,P}$ global constraint to solve the mining problem, this amounts to removing from $P$ every formal concept that does not satisfy this constraint.

### 13.3.3 Soft clustering

**Feedback.** When analyzing configuration parts, our expert integrator discarded many configuration parts that are not relevant according to his knowledge because they do not correspond to any business logic.

When analyzing these parts, we found out that the quality of the solution is sometimes degraded because the set of selected formal concepts is constrained to define a partition of the configurations. This partition constraint is composed of two constraints: a *coverage* constraint, that ensures that every configuration is contained in a formal concept, and a *non-overlap* constraint, that ensures that a same configuration is not contained in more than one formal concept.

It may be interesting to allow the user to soften these constraints. When softening the coverage constraint, we allow a few configurations to belong to no formal concept. Indeed, some configurations may contain errors or correspond to bad practices (if they have been done by unexperienced *Copilote* integrators, for example). Ignoring these configurations will obviously improve the quality of the configuration parts we extract.

When softening the non-overlap constraint, we allow a few configurations to belong to more than one formal concept. In our applicative context, this may happen when a same configuration fulfills different business logic requirements.

**Proposed solution.** We propose to search for soft conceptual clusterings (as defined in Section 6.3). We introduce two thresholds $\delta_o \in [1, \#\mathcal{T}]$ and $\delta_c \in [1, \#\mathcal{T}]$ such that at most $\delta_c$ transactions are not covered by the clustering and each transaction can belong to at most $\delta_o$ clusters.

To solve this soft conceptual clustering problem, we have adapted the *SetDec* CP model introduced in Section 9.2 to solve the exact cover problem. The adaptation is rather straightforward.

- We modify the domains of *coveredBy* variables: In this initial model, for each element $a \in S$, we have an integer variable $coveredBy_a$ which is used to represent the selected subset that covers $a$. As an element may be covered by 0, 1, or several subsets, we use set variables instead of integer variables, and we define the domain of these variables by $D(coveredBy_a) = [\emptyset, cover(a)]$.

- We modify the coverage constraint: In the initial model, this constraint is $coveredBy_a \in C$ (where $C$ is the set variable that represents the selected subsets); We replace it by $cover(a) \cap C = coveredBy_a$.

- We control overlaps by constraining the cardinality of *coveredBy* variables: $\#coveredBy_a \leq \delta_o$.
- We control the number of uncovered transactions by posting the constraint $nbEmpty(coveredBy, N)$, which constrains $N$ to be equal to the number of empty *coveredBy* sets, and by posting the constraint $N \leq \delta_c$.

**Performances.** We give only a quick overview of the experiments we have done with this new model, denoted *SoftSetDec*. When we relax only the coverage constraint, *i.e.* $\delta_o = 1$ and $\delta_c > 0$, the soft conceptual clustering problem is solved faster by *SoftSetDec* than the initial problem with *SetDec* (when the solution is constrained to be a partition), and the larger the instance, the larger the difference of performance between the two models. This is due to the fact that soft conceptual clustering solutions are a superset of conceptual clustering solutions, which allows our heuristic to find a good solution faster.

However, when we relax only the non-overlap constraint, *i.e.* $\delta_o > 1$ and $\delta_c = 0$, the problem becomes much harder and *SoftSetDec* is always slower than *SetDec*. Probably, a better heuristic could be to first search for a conceptual clustering which is an exact partition with *SetDec* and then iteratively and greedily extend it by allowing overlaps that satisfy the threshold constraint and improve the considered utility measure.

### 13.3.4 Hierarchical clustering

**Feedback.** Our expert found that more meaningful results were obtained when using the *WRAcc* utility measure as optimization criterion, and using *pivot parameters*. However, conceptual clusterings computed with these criteria and constraints usually contain formal concepts that have very few items because *WRAcc* is weighted with the frequency, and formal concepts with high frequencies usually have small sizes (*i.e.*, small number of items).

In practice, customers almost never share exactly the same needs. Requirements may have a common business logic but they differ at some point. These configuration parts with very few items correspond to high level requirements, which are shared by several customers. However, each of these high-level requirements may be specialized into more specific variants. In other words, requirements can be described with a tree structure where most common requirements are at the top of the tree and children correspond to more precise requirements.

**Solution.** We propose to use hierarchical conceptual clustering. In many works, clusters are organized in hierarchies [Mic+83; Fis87]. As explained in [Fis96], a hierarchical-clustering creates a tree-structured clustering, where sibling clusters partition the transactions covered by their common parent.

We propose a top-down approach, described in Algorithm 9, that consists in recursively applying conceptual clustering on the clusters we find until getting clusters with only one transaction. The function *clusterise* applies conceptual clustering on the transactions of the cluster $c$ given in input (line 4). For each obtained cluster $c_i$, we add it to the children of $c$ (line 6) and we recursively call *clusterise* on $c_i$ (line 7). We stop when the input cluster contains only one transaction (line 3). To get the whole

**Algorithm 9:** Hierarchical conceptual clustering

```
1 Function clusterise(c)
      Input: A formal concept c = (T, I)
2     begin
3         if #T > 1 then
4             C ← conceptual_clustering(T, I)
5             for each c_i = (T_i, I_i) ∈ C do
6                 Add c_i to the list of children of c
7                 clusterise(c_i)
```

hierarchical structure, we start by calling *clusterise* with a cluster that contains all the transactions. We obtain a tree of formal concepts where each child represents an extension of its parent, *i.e*, every child is a sub-concept of its parent.

In our applicative context, sibling concepts correspond to variants of their parent configuration part: they may correspond to requirements slightly more precise than the parent requirement.

We apply the *WRAcc* measure in a more relevant way than defined in Section 13.2. We propose to optimize the *WRAcc* of sibling clusters with their parent which is much more intuitive: We maximize the correlation between a parent and its children. More formally, let $c = (T, I)$ be a formal concept, when recursively clustering $c$, the *WRAcc* measure used to evaluate the utility of $c_i = (T_i, I_i)$ when executing *conceptual_clustering*$(T, I)$ (line 4) is defined by $WRAcc(I \rightarrow I_i)$.

Note that the *WRAcc* measure seems very relevant to do top-down conceptual clustering since it favors formal concepts with both high frequency and high correlation, *i.e.*, configuration parts that correspond to high level requirements.

### 13.3.5 Default parameter values

**Feedback.** A recurrent feedback from system integrators is that some items appear in many configuration parts while they do not bring any business logic in the concept. These items make the interpretation much more complex, even to know whether a configuration part is interesting from a business logic point of view. We want to limit as much as possible these items that degrade the quality of the configuration parts and make it difficult to interpret them.

**Analysis of the problem.** When analyzing these items, we find out that they often correspond to the assignment of a default value to a parameter which is not used by the system. Indeed, when Infologic implements *Copilote* for a new customer, a default configuration is installed for this customer before system integrators can start to configure it. This default configuration embeds a lot of data in order to have a first version that already works and to reduce the work of configuration by setting the most standard values to parameters. When a parameter has not been changed from its default value, we cannot know if a system integrator validated the value assigned to the parameter, *i.e.*, whether the value is needed to fulfill a business requirement or

not. Many parameter settings are inherited from the default configuration while they are useless.

Therefore, when a parameter is almost never changed by system integrators, the item associated with its default value has a high frequency and may appear in many formal concepts since it is shared by most of the configurations, even if it is not used by the system.

It is very hard to automatically identify items corresponding to default parameter values that are not used by *Copilote*. We can access the log of the modifications of the parameters and therefore identify what parameters have been changed by a system integrator. However, selecting only these instances would not be suitable since a significant part of default values assigned to parameters is actually used by *Copilote*.

Hence, we have not found a solution to automatically identify parameter values that are not used by *Copilote* to ignore them when extracting configuration parts.

## 13.4 Complete toolkit for configuration part mining

In this section, we give an overview of the complete toolkit we have developed to allow a *Copilote* integrator expert to interactively mine configuration parts.

**Selection of the data.** Before mining configuration parts, the first step is to select a business unit *bu* from the business unit map (described in Section 4.1), together with a set of customers for which he wants to extract configuration parts. The toolkit displays the full list of parameters associated with *bu*, *i.e.*, *parameters*(*bu*), and the expert may add or remove some parameters.

**Configuration of the search.** The second step is to configure the search according to the constraints and criteria of the expert. To this aim, the expert may:

- Set a minimal and/or maximal frequency of the concepts;
- Set a minimal and/or maximal size of the concepts;
- Select some parameters and define them as pivot parameters in order to constrain every selected concept to contain exactly one parameter value for each of these pivot parameters.

As explained before, all these constraints are taken into account in a pre-processing step, by removing from the set $P$ of all candidate concepts those that do not satisfy them.

The expert may choose properties of the clustering, *i.e.*, choose between "hard", soft and hierarchical clustering. This is done by setting the following parameters:

- The percentage of configurations that must be covered by the clustering (the default value of this parameter is 100%);
- The maximal number of clusters a configuration part can belong to (the default value of this parameter is 1);

- A boolean parameter that indicates whether to apply hierarchical conceptual clustering or not (the default value is *false*).

All these options are compatible and one may apply soft hierarchical clustering.

The different utility measures that may be used to evaluate the quality of a clustering are: $Min_{frequency}$, $Min_{size}$, $Min_{split}$, $Min_{-diameter}$, and $Min_{WRAcc}$. If the expert has chosen to do hierarchical clustering, then he can select only one utility measure to optimize. Otherwise he can select one or two utility measures: If only one measure is selected, we search for a conceptual clustering that optimizes it; If two measures are selected, we search for all non-dominated clusterings.

Finally, the expert may set a timeout: If the search is not completed within the timeout, the best solution found is returned.

It is possible that new constraints and criteria may be further needed by experts. The interest of using CP to implement the search engine of our toolkit is that we should be able to quickly modify the CP model to integrate them.

**Visualization of the search results.** If hierarchical clustering has been selected, then we display a hierarchy of formal concepts. Otherwise, we display a set of formal concepts (corresponding to the optimal solution if only one utility measure is selected, or to a set of non-dominated solutions if two utility measures have been selected).

In both cases, for each formal concept, we visualize:

- The parameter and the value corresponding to each item of the concept;
- Measures associated with the concept, *i.e.*, frequency, size, split, diameter, and $WRAcc$;
- The set of customers that use the configuration part.

**Evaluation of the concepts.** The last functionality of the toolkit is the evaluation of the concepts. The expert can select concepts and, for each selected concept, he can either tag it as relevant or irrelevant. If the concept is tagged as relevant, the expert is invited to enter a textual description of the business logic it implements. If the concept is tagged as irrelevant, the expert is invited to enter a textual explanation of its irrelevancy.

**Expert feedback after using our toolkit.** Our toolkit has been used by one expert for mining configuration parts associated with one business unit described in Section 13.1. For this business unit, he found that the best results are obtained when applying hierarchical soft conceptual clustering using pivot parameters and selecting the $Min_{WRAcc}$ utility measure. This ensures that the configuration tree is split according to pivot parameters and that concepts within a same subtree are highly correlated. Most of the time, the expert chose to allow from 10 to 20% of configurations to be uncovered.

## 13.5 Discussion

We have described an experimental toolkit which may be used by an expert to mine configuration parts for a targeted business unit, and associate to every selected configuration part a description of the requirement it fulfills. The search engine used to mine concepts is implemented in CP, and this allowed us to easily integrate new constraints and criteria for automatically discarding irrelevant configuration parts. However, the interpretation of the extracted concepts is not straightforward and may be time consuming since some parts of concept, which are hard to identify, do not bring any business logic. It may be an important hindrance for the model and improving the relevancy of the configuration parts is still our main challenge.

# Chapter 14

# Conclusion

We introduced in this thesis a new approach to assist system integrators when implementing an ERP system for a new customer. First, we introduced the Business Unit Map, which is a structured model of the different parts of the ERP system that may be configured according to business logic requirements, and which allowed us to relate parameters with business logic scopes. Then, we proposed an approach for extracting a catalog of configuration parts from existing configurations of the ERP system: Each configuration part corresponds to a business logic requirement that may be reused for next implementations of the ERP system. Our approach has been illustrated and experimented with *Copilote*, the ERP system developed by Infologic. However, we believe it could be adapted to other ERP systems.

The main challenge was to design a mining tool for extracting relevant configuration parts (to be added to the catalog) from existing configurations. A first difficulty came from the fact that experts are not able to define an ideal measure for evaluating the relevancy of a configuration part. Hence, we proposed an interactive process for integrating expert feedbacks on the relevancy of the extracted configuration parts in order to improve the relevancy of the next extractions. Another difficulty came from the fact that each extraction process basically involves solving a constrained conceptual clustering problem which is $\mathcal{NP}$-hard. We proposed to use Constraint Programming to solve this problem because it is a flexible, declarative, and easy to use framework which is well suited in our interactive context: It allowed us to easily integrate expert feedbacks by means of constraints and optimization criteria. However, conceptual clustering is a challenging problem, especially for the largest instances that may have millions of formal concepts, or when the goal is to compute the whole set of non-dominated solutions with respect to several conflicting optimization criteria. Our main technical contributions aimed at improving scale-up properties of CP when solving these problems.

First, we proposed two new CP models, and we showed that they scale well when the number of clusters is not known *a priori* and when there is only one objective function to optimize. However, these models are not efficient enough when the number of clusters is more tightly constrained, or when the goal is to find the Pareto-front of all non-dominated solutions with respect to several conflicting objective functions, for example. CP is an extensible framework, where the user may integrate new constraints, together with their propagators, in order to improve the expressive power of the constraint language or the efficiency of the solving engine. We used this property to introduce a new global constraint dedicated to the exact cover problem, which is at the core of conceptual clustering problems. We proposed to use *Dancing Links* to efficiently propagate this constraint, and we showed how to strengthen this propagation when

the number of selected subsets is constrained. This new global constraint allowed us both to model more easily conceptual clustering problems, and to solve them more efficiently than existing declarative approaches. Exact cover problems occur in many other applications, and we plan to integrate our new propagators in the Choco library, for allowing further uses of them for other applications.

We integrated our CP models for solving conceptual clustering problems within a prototype that aims at demonstrating the interest of our interactive process for mining relevant configuration parts from existing *Copilote* configurations. An expert used this prototype for mining the configuration parts associated with the business unit related to *Plan production*. Most of his feedbacks were straightforward to integrate within the CP model and, in some cases, these new constraints improved the efficiency of the solving process because they reduced the search space. Other feedbacks are more challenging such as, for example, softening the non-overlapping constraint, or automatically identifying useless parameters that are assigned to default values.

# Perspectives

We have proposed a proof of concept of our approach which validates the potential gain of using our assistant of configuration. However, it is obviously not usable as it is and many perspectives of improvement remain from both CP and applicative points of view.

**Soft *exactCover* global constraint.** A first improvement is to extend our global constraint *exactCover* to allow the user to soften non-overlapping or coverage constraints. A convenient and flexible extension is to add $\#S$ integer variables to the input parameters: Each of these variables is associated with a different element and is constrained to be equal to the number of selected subsets that cover this element. This way, we allow the user to constrain in many different ways the coverage and the overlapping of the selected subsets. For instance, we may easily model the constraint of allowing at most $x\%$ of elements to overlap or allowing few elements not to be covered. Obviously, the extension of the propagators of *exactCover* are not straightforward but the gain of efficiency compared with our *softSetDec* model may be significant and may allow us to have a more reactive configuration mining tool.

**New strategies for computing Pareto fronts of multi-criteria optimization problems.** We have compared three different strategies for computing Pareto front: The static strategy of [Was+80], the dynamic strategy of [Gav02], and an improvement of this dynamic strategy that we have proposed for our specific problem where each objective function aims at maximizing a minimal utility cost. Experimental results have shown us that (i) these strategies are complementary, and (ii) they have a strong impact on the efficiency of the solution process. Hence, we are convinced that there is room for improvements.

First, when using the static approach, we solve a sequence of optimization problems, and we could exploit results of previous searches when solving the next optimization problem. For example, we could collect all solutions that have been found during all previous searches. When solving the next optimization problem, we could filter these

solutions to keep those that satisfy the current constraints, and update bounds on objective variables consequently. This may be seen as a merge of the static approach and the dynamic approach.

Also, we observed that ordering heuristics have a strong impact on the search process when using a dynamic strategy: Ideally, we would like to quickly find solutions that dominate the largest number of subsets. To this aim, we have proposed to favor the selection of subsets that dominate the largest number of subsets. However, selecting a good subset does not necessarily lead to a good solution. Hence, we believe there is room for improvement. For example, we could take into account the fact that selecting a subset degrades or not current minimum utility values.

Finally, parallelizing several searches that explore different parts of the search space may be relevant. We have introduced and compared different strategies for parallelizing a Pareto front search in [Cha+17b] but we did not explore all possibilities. Indeed, we have efficient models to solve mono-criterion optimization problems and computers have many cores. Therefore, we could take advantage of this by splitting the problem into subproblems that focus on different parts of the search space, corresponding to different compromises between the different criteria, and communicate the solutions found to reduce the search space.

**New relevancy measures.** The main hindrance of the use of our tool is the lack of relevancy of the extracted configuration parts that makes it hard and time consuming for the expert to interpret results. Improving the relevancy of the configuration parts we extract remains an important challenge for us. We have proposed to use the *WRAcc* utility measure, and our expert found that this improved the relevancy of the extracted configuration parts because they correspond to more correlated parameter values. However, our input data (*i.e.*, existing configurations) has specificities (such as default values assigned to parameters, or parameter values that are never used by *Copilote*) that degrade the quality of the solutions we obtain.

A promising improvement would be to take into account user experience data in our relevancy measure. Indeed, *Copilote* logs all modifications done on parameter values during the implementation process: Each time a value is assigned to a parameter, *Copilote* records the time, the user who modified the value and the new assigned value. Also, *Copilote* records every access to a screen with the user and the time of the access. We could exploit these data to improve our relevancy measure. For instance, we could favor configuration parts with parameters that are often modified together or parameters that are modified after accessing a given screen.

Furthermore, we could associate an importance rate with each parameter: *Copilote* has critical parameters that have a strong impact on the business logic while other parameters have only a small impact (such as layout of screen or color of fonts, for example). Favoring configuration parts with important parameters would ensure more interest and value added of the extracted configuration parts. However, finding the best way to exploit these different utility measures together may be a challenge.

**Using the business unit map as a common language.** This thesis is part of an ambitious project that aims at improving the whole implementation process of *Copilote*, which is a critical issue for Infologic. By discovering the implementation process of *Copilote* as any *Copilote* integrator beginner on one hand, and studying existing

researches about ERP implementation processes on the other hand, we had the chance to step back to think about improving the implementation process. We are convinced, together with Infologic experts, that the business unit map could be more widely exploited to significantly improve the quality of the services proposed by Infologic. A tremendous challenge is to relate the whole activity of Infologic to this business unit map. Obviously, most of the activities of Infologic employees concern *Copilote*: They either sell it (for sales representatives), or develop parts of it (for software developers), or configure it (for system integrators). At the moment, there exists no interface between these different categories of employees that all work on the same product: *Copilote*. However, system integrators need to know what scope of *Copilote* has been sold when implementing it, sales representatives and system integrators need to know what functionalities of *Copilote* have been newly developed, the management of Infologic needs to compare the cost of the development of new *Copilote* functionalities with prices they are sold, etc. The business unit map provides an obvious common language between all these activities: Every new development may be translated into business units of *Copilote* that can be sold and implemented; A sale of *Copilote* may be translated into a set of business units to install; etc. We can take advantage of this in many different ways.

For instance, we could use the map to manage skills of system integrators. Indeed, if every activity is related to a business unit, we can easily deduce from the past which employees may be competent to achieve a task that requires knowledge on a given business unit. For instance, we may record, for each system integrator, the list of business units he has already configured in previous implementations, and this may be used to define his scope of competences on *Copilote*. This may be useful to plan activities or to train integrators if there is a lack of competences to achieve incoming tasks, which are critical issues for Infologic.

# Bibliography


[Agr+93] R. Agrawal, T. Imielinski, and A. Swami. "Mining association rules between sets of items in large databases". In: *International Conference on Management of Data (SIGMOD)*. ACM. 1993, pp. 207–216 (cit. on p. 144).

[Ahm+12] N. Ahmad, A. Haleem, and A. A. Syed. "Compilation of Critical Success Factors in Implementation of Enterprise Systems: A Study on Indian Organisations". In: *Global Journal of Flexible Systems Management*, 13(4) (2012), pp. 217–232 (cit. on pp. 28, 30).

[Ahm+13] M. M. Ahmad and R. Pinedo Cuenca. "Critical Success Factors for ERP Implementation in SMEs". In: *Robot. Comput.-Integr. Manuf.* 29(3) (2013), pp. 104–111 (cit. on pp. 25, 28, 30, 35).

[AM+03] M. Al-Mashari, A. Al-Mudimigh, and M. Zairi. "Enterprise resource planning: A taxonomy of critical factors". In: *European Journal of Operational Research*, 146(2) (2003), pp. 352 –364 (cit. on pp. 28, 30).

[And09] S. Andrews. "In-Close, a fast algorithm for computing formal concepts". In: *International Conference on Conceptual Structures (ICCS)*. Final version of paper accepted (via peer review) for the International Conference on Conceptual Structures (ICCS) 2009, Moscow. 2009 (cit. on p. 80).

[Ari+03] B. Arinze and M. Anandarajan. "A Framework for Using OO Mapping Methods to Rapidly Configure ERP Systems". In: *Commun. ACM*, 46(2) (2003), pp. 61–65 (cit. on pp. 36, 37).

[Ari+18] N. Aribi, A. Ouali, Y. Lebbah, and S. Loudni. "Equitable Conceptual Clustering Using OWA Operator". In: *Advances in Knowledge Discovery and Data Mining - 22nd Pacific-Asia Conference, PAKDD 2018, Melbourne, VIC, Australia, June 3-6, 2018, Proceedings, Part III*. 2018, pp. 465–477 (cit. on pp. 81, 99).

[Aré+07] G. Arévalo, A. Berry, M. Huchard, G. Perrot, and A. Sigayret. "Performances of Galois Sub-hierarchy-building algorithms". In: *Fifth International Conference on Formal Concept Analysis*. Ed. by S. K. S. Schmidt. Vol. LNAI 4390. voir hal# lirmm-00163381 pour le rapport de recherche au format PDF. Clermont-Ferrand, France: LNAI, 2007, pp 166–180 (cit. on p. 80).

[Bel+05] N. Beldiceanu, M. Carlsson, and j.-x. Rampon. "Global Constraint Catalog". In: (2005) (cit. on p. 64).

[Bel01] N. Beldiceanu. "Pruning for the Minimum Constraint Family and for the Number of Distinct Values Constraint Family". In: *Principles and Practice of Constraint Programming — CP 2001*. Ed. by T. Walsh. Berlin,

Heidelberg: Springer Berlin Heidelberg, 2001, pp. 211–224 (cit. on pp. 66, 126).

[Bes+03] C. Bessière and P. Van Hentenryck. "To Be or Not to Be ... a Global Constraint". In: *Principles and Practice of Constraint Programming (CP)*. Springer, 2003, pp. 789–794 (cit. on p. 65).

[Bes+04a] C. Bessiere, E. Hebrard, B. Hnich, and T. Walsh. "Disjoint, Partition and Intersection Constraints for Set and Multiset Variables". In: *Principles and Practice of Constraint Programming – CP 2004*. Ed. by M. Wallace. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 138–152 (cit. on p. 66).

[Bes+04b] C. Bessiere, E. Hebrard, B. Hnich, and T. Walsh. "The Complexity of Global Constraints". In: *Proceedings of the 19th National Conference on Artifical Intelligence*. AAAI'04. San Jose, California: AAAI Press, 2004, pp. 112–117 (cit. on p. 65).

[Bes+05] C. Bessiere, E. Hebrard, B. Hnich, Z. Kiziltan, and T. Walsh. "Filtering Algorithms for the NValue Constraint". In: *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*. Ed. by R. Barták and M. Milano. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 79–93 (cit. on pp. 66, 124, 126–128).

[Bes+07] C. Bessière, E. Hébrard, B. Hnich, and T. Walsh. "The Complexity of Reasoning with Global Constraints". In: *Constraints*, 12(2) (2007), pp. 239–259 (cit. on p. 65).

[Bes+08] C. Bessiere and R. Debruyne. "Theoretical analysis of singleton arc consistency and its extensions". In: *Artificial Intelligence*, 172(1) (2008), pp. 29 –41 (cit. on p. 60).

[Bes+09a] C. Bessiere, G. Katsirelos, N. Narodytska, and T. Walsh. "Circuit Complexity and Decompositions of Global Constraints". In: *Proceedings of the 21st International Jont Conference on Artifical Intelligence*. IJCAI'09. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2009, pp. 412–418 (cit. on p. 65).

[Bes+09b] C. Bessiere, G. Katsirelos, N. Narodytska, C. Quimper, and T. Walsh. "Decomposition of the NVALUE constraint". In: *CoRR*, abs/0909.3273 (2009). arXiv: 0909.3273 (cit. on p. 66).

[Bes94] C. Bessiere. "Arc-consistency and arc-consistency again". In: *Artificial Intelligence*, 65(1) (1994), pp. 179 –190 (cit. on p. 60).

[Bez+78] J. C. Bezdek and J. D. Harris. "Fuzzy partitions and relations; an axiomatic basis for clustering". In: *Fuzzy Sets and Systems*, 1(2) (1978), pp. 111 –127 (cit. on p. 83).

[Bez81] J. C. Bezdek. *Pattern Recognition with Fuzzy Objective Function Algorithms*. Norwell, MA, USA: Kluwer Academic Publishers, 1981 (cit. on p. 83).

[BG+05] V. Botta-Genoulaz, P.-A. Millet, and B. Grabot. "A Survey on the Recent Literature on ERP Systems". In: 56 (2005), pp. 510–522 (cit. on pp. 28, 29).

[Bie+09] A. Biere, A. Biere, M. Heule, H. van Maaren, and T. Walsh. *Handbook of Satisfiability: Volume 185 Frontiers in Artificial Intelligence and Applications*. Amsterdam, The Netherlands, The Netherlands: IOS Press, 2009 (cit. on p. 74).

[Bin+99] P. Bingi, M. K. Sharma, and J. K. Godla. "Critical Issues Affecting an ERP Implementation". In: *Information Systems Management*, 16(3) (1999), pp. 7–14. eprint: http://dx.doi.org/10.1201/1078/43197.16.3.19990601/31310.2 (cit. on pp. 28, 30).

[Bre+01] L. Brehm, A. Heinzl, and M. Markus. "Tailoring ERP Systems: A Spectrum of Choices and Their Implications". In: *Proceedings of the 34th Annual Hawaii International Conference on System Sciences ( HICSS-34)-Volume 8 - Volume 8*. HICSS '01. Washington, DC, USA: IEEE Computer Society, 2001, pp. 8017– (cit. on p. 29).

[Buc+10] D. Bucher and J. Meissner. "Automatic Parameter Configuration for Inventory Management in SAP ERP/APO". In: (Jan. 2010) (cit. on pp. 36, 37).

[Cha+17a] M. Chabert, P.-A. Champin, A. Cordier, and C. Solnon. "Comparaison de différents modèles de programmation par contraintes pour le clustering conceptuel". In: *Actes JPFC 2017*. Montreuil-sur-Mer, France, 2017 (cit. on p. 20).

[Cha+17b] M. Chabert and C. Solnon. "Constraint Programming for Multi-criteria Conceptual Clustering". In: *Principles and Practice of Constraint Programming - 23rd International Conference, CP, Proceedings*. Vol. 10416. Lecture Notes in Computer Science. Springer, 2017, pp. 460–476 (cit. on pp. 20, 155).

[Cha+18] M. Chabert and C. Solnon. "A Global Constraint for the Exact Cover Problem: Application to Conceptual Clustering". In: *Doctoral Program of CP 2018*. Lille, France, 2018 (cit. on p. 20).

[Che+85] Y. Cheng and K. S. Fu. "Conceptual Clustering in Knowledge Organization". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-7(5) (1985), pp. 592–598 (cit. on p. 81).

[Chu+99] S. H. Chung and C. A. Snyder. "ERP Initiation - A Historical Perspective". In: *Proceedings of the Fifth Americas Conference on Information Systems August 13-15, 1999 Milwaukee, Wisconsin, USA*. 1999, pp. 213–215 (cit. on p. 24).

[Dao+15a] T.-B.-H. Dao, W. Lesaint, and C. Vrain. "Clustering conceptuel et relationnel en programmation par contraintes". In: *JFPC 2015*. Bordeaux, France, 2015 (cit. on pp. 13, 19, 81, 84, 86, 99, 104, 105, 110, 130).

[Dao+15b] T.-B.-H. Dao, K.-C. Duong, and C. Vrain. "Constrained Clustering by Constraint Programming". In: *Artificial Intelligence* (2015) (cit. on pp. 84, 86, 103–105).

[Dar+93] A. Dardenne, A. van Lamsweerde, and S. Fickas. "Goal-directed Requirements Acquisition". In: *Selected Papers of the Sixth International Workshop on Software Specification and Design*. 6IWSSD. Amsterdam, The Netherlands, The Netherlands: Elsevier Science Publishers B. V., 1993, pp. 3–50 (cit. on p. 28).

[Dav+11] J. Davies and F. Bacchus. "Solving MAXSAT by Solving a Sequence of Simpler SAT Instances". In: *Principles and Practice of Constraint Programming - 17th International Conference, CP. Proceedings*. Vol. 6876. Lecture Notes in Computer Science. Springer, 2011, pp. 225–239 (cit. on p. 114).

[Dhe+17] D. Dheeru and E. Karra Taniskidou. *UCI Machine Learning Repository*. 2017 (cit. on p. 99).

[Dia+00] D. Diaz and P. Codognet. "GNU Prolog: Beyond Compiling Prolog to C". In: *Practical Aspects of Declarative Languages*. Ed. by E. Pontelli and V. Santos Costa. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 81–92 (cit. on p. 72).

[Dit+09] Y. Dittrich, S. Vaucouleur, and S. Giff. "ERP Customization as Software Engineering: Knowledge Sharing and Cooperation". In: *IEEE Software*, 26(6) (2009), pp. 41–47 (cit. on p. 29).

[Do+14] T. N. Do, S. Moga, and P. Lenca. "Random forest of oblique decision trees for ERP semi-automatic configuration". In: *ACIIDS 2014 : the 6th Asian Conference on Intelligent Information and Database Systems*. Ed. by Springer. Vol. 551 - SCI (Studies in Computational Intelligence). Advanced approaches to intelligent information and database systems (Studies in Computational Intelligence, Volume 551, 2014). 2014, pp. 25–34 (cit. on p. 36).

[Duo14] K.-C. Duong. "Constrained clustering by constraint programming". Theses. Université d'Orléans, 2014 (cit. on pp. 69, 81).

[Era+15] I. Erasmus and M. Daneva. "ERP services effort estimation strategies based on early requirements". In: *2nd International Workshop on Requirements Engineering for the Precontract Phase (RE4P2) at REFSQ 2015*. Ed. by A. Kalenborg and M. Trapp. CEUR Workshop Proceedings. CEUR-WS.org, 2015, pp. 83–99 (cit. on pp. 12, 18, 45).

[Fag+17] J. Fages and C. Prud'Homme. "Making the First Solution Good!" In: *2017 IEEE 29th International Conference on Tools with Artificial Intelligence (ICTAI)*. 2017, pp. 1073–1077 (cit. on p. 69).

[Fer+06] T. W. Ferratt, S. Ahire, and P. De. "Achieving Success in Large Projects: Implications from a Study of ERP Implementations". In: *Interfaces*, 36(5) (Sept. 2006), pp. 458–469 (cit. on p. 29).

[Fis87] D. H. Fisher. “Knowledge Acquisition Via Incremental Conceptual Clustering”. In: *Mach. Learn.* 2(2) (1987), pp. 139–172 (cit. on pp. 81, 148).

[Fis96] D. Fisher. “Iterative Optimization and Simplification of Hierarchical Clusterings”. In: *J. Artif. Int. Res.* 4(1) (1996), pp. 147–179 (cit. on p. 148).

[Flo+10] A. Floch, C. Wolinski, and K. Kuchcinski. “Combined scheduling and instruction selection for processors with reconfigurable cell fabric”. In: *21st IEEE International Conference on Application-specific Systems Architectures and Processors, ASAP 2010*. 2010, pp. 167–174 (cit. on p. 96).

[FR79] J. F. Rockart. “Chief Executives Define Their Own Data Needs”. In: 57 (1979), pp. 81–93 (cit. on p. 30).

[Fre97] E. C. Freuder. “In Pursuit of the Holy Grail”. In: *Constraints*, 2(1) (1997), pp. 57–61 (cit. on p. 72).

[Gan+97] B. Ganter and R. Wille. *Formal Concept Analysis: Mathematical Foundations*. Springer, 1997 (cit. on pp. 52, 75).

[Gar+17] N. Garg, M. Sadiq, and P. Agarwal. “GOASREP: Goal Oriented Approach for Software Requirements Elicitation and Prioritization Using Analytic Hierarchy Process”. In: *Proceedings of the 5th International Conference on Frontiers in Intelligent Computing: Theory and Applications : FICTA 2016, Volume 2*. Ed. by S. C. Satapathy, V. Bhateja, S. K. Udgata, and P. K. Pattnaik. Singapore: Springer Singapore, 2017, pp. 281–287 (cit. on p. 28).

[Gav02] M. Gavanelli. “An Algorithm for Multi-criteria Optimization in CSPs”. In: *Proceedings of the 15th European Conference on Artificial Intelligence*. ECAI’02. Amsterdam, The Netherlands, The Netherlands: IOS Press, 2002, pp. 136–140 (cit. on pp. 71, 137, 138, 154).

[Geb+12] M. Gebser, B. Kaufmann, and T. Schaub. “Conflict-driven answer set solving: From theory to practice”. In: *Artif. Intell.* 187 (2012), pp. 52–89 (cit. on pp. 96, 116).

[Ger95] C. Gervet. “Set Intervals in Constraint Logic Programming: Definition and implementation of a language”. Theses. Université de Franche Comté Besançon, 1995 (cit. on p. 63).

[Gol+65] S. W. Golomb and L. D. Baumert. “Backtrack Programming”. In: *J. ACM*, 12(4) (1965), pp. 516–524 (cit. on p. 63).

[Gra+14] B. Grabot, A. Mayère, F. Lauroua, and R. Houe Ngouna. “ERP 2.0, what for and how?” In: 65 (2014) (cit. on p. 25).

[Gum96] R. Gumaer. “Beyond ERP and MRP II: optimized planning and synchronized manufacturing. (Enterprise Resource Planning; Manufacturing Resource Planning).” In: *IIE Solutions. 1996. HighBeam Research. (October 6, 2017).* (1996) (cit. on p. 24).

[Gun+11] T. Guns, S. Nijssen, and L. D. Raedt. “Itemset mining: A constraint programming perspective”. In: *Artif. Intell.* 175(12-13) (2011), pp. 1951–1983 (cit. on p. 80).

[Gun15] T. Guns. “Declarative pattern mining using constraint programming.” In: *Constraints*, 20(4) (2015), pp. 492–493 (cit. on pp. 13, 19, 80, 81, 83, 84, 86, 99).

[Hal+97] M. M. Halldórsson and J. Radhakrishnan. “Greed is good: Approximating independent sets in sparse and bounded-degree graphs”. In: *Algorithmica*, 18(1) (1997), pp. 145–163 (cit. on p. 124).

[HB18] G. Hjort Blindell. “Universal Instruction Selection”. PhD thesis. KTH, Software and Computer systems, SCS, 2018, p. 314 (cit. on pp. 91, 95).

[Hen+88] P. V. Hentenryck and J.-P. Carillon. “Generality versus Specificity: An Experience with AI and OR Techniques”. In: *AAAI*. 1988 (cit. on p. 66).

[Hua+04] S. Huang, Y.-C. Hung, H.-G Chen, and C.-Y Ku. “Transplanting the best practice for implementation of an ERP system: A structured inductive study of an international company”. In: 44 (June 2004), pp. 101–110 (cit. on p. 29).

[Héb+07] C. Hébert and B. Crémilleux. “A Unified View of Objective Interestingness Measures”. In: *MLDM*. 2007 (cit. on p. 144).

[Jac01] P. Jaccard. “Distribution de la flore alpine dans le bassin des Dranses et dans quelques régions voisines”. In: *Bulletin de la Société Vaudoise des Sciences Naturelles*, 37 (1901), pp. 241 –272 (cit. on p. 79).

[Jan+15] J. Jansson and F. Jonsson. *Development of an ERP Requirements Specification Method by Applying Rapid Contextual Design : A Case Study of a Medium-sized Enterprise*. 2015 (cit. on p. 28).

[Jun+10] T. Junttila and P. Kaski. “Exact Cover via Satisfiability: An Empirical Study”. In: *Principles and Practice of Constraint Programming – CP 2010*. Springer, 2010, pp. 297–304 (cit. on pp. 90, 96, 97, 116–118).

[Kar72] R. M. Karp. “Reducibility among Combinatorial Problems”. In: *Complexity of Computer Computations*. Springer, 1972, pp. 85–103 (cit. on p. 88).

[Kas+08] P. Kaski and O. Pottonen. “libexact User s Guide, Version 1.0”. In: *HIIT Technical Reports*, 187 (2008) (cit. on pp. 95, 96, 116).

[Khi+10] M. Khiari, P. Boizumault, and B. Crémilleux. “Constraint Programming for Mining n-ary Patterns”. In: *Principles and Practice of Constraint Programming - CP 2010 - 16th International Conference, CP 2010, St. Andrews, Scotland, UK, September 6-10, 2010. Proceedings*. 2010, pp. 552–567 (cit. on p. 80).

[Knu09] D. E. Knuth. “Dancing links”. In: *Millenial Perspectives in Computer Science*, 18 (2009), p. 4 (cit. on pp. 14, 19, 90, 91, 95, 112, 114–116).

[Kog+06] J. Kogan, C. Nicholas, and M. Teboulle. *Grouping Multidimensional Data: Recent Advances in Clustering*. Berlin , Heidelberg: Springer-Verlag, 2006 (cit. on p. 83).

[Kuz+02] S. O. Kuznetsov and S. Obiedkov. “Comparing Performance of Algorithms for Generating Concept Lattices”. In: *JOURNAL OF EXPERIMENTAL AND THEORETICAL ARTIFICIAL INTELLIGENCE*, 14 (2002), pp. 189–216 (cit. on p. 80).

[Kuz99] S. O. Kuznetsov. “Learning of Simple Conceptual Graphs from Positive and Negative Examples”. In: *Principles of Data Mining and Knowledge Discovery*. Ed. by J. M. Żytkow and J. Rauch. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 384–391 (cit. on p. 80).

[KW16] S. S. Klaus Wölfel Jean-Paul Smets. “Automating ERP Package Configuration for Small Businesses”. In: (2016) (cit. on p. 36).

[Käh14] A. A. andTommi Kähkönen. “Knowledge Transfer Challenges in ERP Development Networks: The Quest for a Shared Development Model”. In: *27th Bled eConference: eEcosystems, Bled, Slovenia, June 1-5, 2014*. 2014, p. 27 (cit. on pp. 28, 30).

[Lac+14] C. Lacombe, R. Pochelu, S. Tazi, and Y. Ducq. “Model-Driven Enterprise Resource Planning Specifications in SMEs”. In: *IFIP International Conference on Advances in Production Management Systems (APMS)*. Ed. by B. Grabot, B. Vallespir, S. Gomes, A. Bouras, and D. Kiritsis. Vol. AICT-440. Advances in Production Management Systems. Innovative and Knowledge-Based Production Management in a Global-Local World Part III. Part 2: Case Studies. Ajaccio, France: Springer, Sept. 2014, pp. 538–545 (cit. on p. 28).

[Lav+99] N. Lavrač, P. Flach, and B. Zupan. “Rule Evaluation Measures: A Unifying View”. In: *Inductive Logic Programming*. Ed. by S. Džeroski and P. Flach. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 174–185 (cit. on p. 144).

[Law+04] Y. C. Law and J. H. M. Lee. “Global Constraints for Integer and Set Value Precedence”. In: *10th International Conference on Principles and Practice of Constraint Programming (CP)*. Springer Berlin Heidelberg, 2004, pp. 362–376 (cit. on pp. 66, 85).

[Laz+16] N. Lazaar, Y. Lebbah, S. Loudni, M. Maamar, V. Lemière, C. Bessiere, and P. Boizumault. “A Global Constraint for Closed Frequent Pattern Mining”. In: *Principles and Practice of Constraint Programming - 22nd International Conference, CP 2016, Toulouse, France, September 5-9, 2016, Proceedings*. 2016, pp. 333–349 (cit. on p. 80).

[Lec96] *A Bounds-Based Reduction Scheme for Difference Constraints*. 1996 (cit. on p. 60).

[Lig01] B. Light. “The Maintenance Implications of the Customization of ERP Software”. In: 13 (2001), pp. 415 –429 (cit. on p. 29).

[Lig05] B. Light. “Going Beyond ’Misfit’ As a Reason for ERP Package Customisation”. In: *Comput. Ind.* 56(6) (Aug. 2005), pp. 606–619 (cit. on p. 29).

[LO+03] A. Lopez-Ortiz, C.-G. Quimper, J. Tromp, and P. Van Beek. "A Fast and Simple Algorithm for Bounds Consistency of the All Different Constraint". In: *Proceedings of the 18th International Joint Conference on Artificial Intelligence*. IJCAI'03. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2003, pp. 245–250 (cit. on p. 60).

[Lui+] J. Luis, D. V. González, and J. S. Díaz. *Business process-driven requirements engineering: a goal-based approach* (cit. on p. 28).

[Luo+04] W. Luo and D. M. Strong. "A framework for evaluating ERP implementation choices". In: *IEEE Transactions on Engineering Management*, 51(3) (2004), pp. 322–333 (cit. on p. 29).

[Mac67] J. MacQueen. "Some methods for classification and analysis of multivariate observations". In: *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics*. Berkeley, Calif.: University of California Press, 1967, pp. 281–297 (cit. on p. 80).

[Mac77] A. Mackworth. "Consistency in Networks of Relations". In: *Artificial Intelligence*, 8(1) (1977). Reprinted in Readings in Artificial Intelligence, B. L. Webber and N. J. Nilsson (eds.), Tioga Publ. Col., Palo Alto, CA, pp. 69-78, 1981. This paper was honored in Artificial Intelligence 59, 1-2, 1993 as one of the fifty most cited papers in the history of Artificial Intelligence. It also received the 2013 AIJ Classic Paper Award: http://www.journals.elsevier.com/artificial-intelligence/news/announcing-winners-of-the-2013-aij-classic-paper-award/, pp. 99–118 (cit. on p. 58).

[Mam13] S. Mamoghli. "Contribution to the alignment of off-the-shelf product based information systems : towards a model-driven engineering, based on risk identification". Theses. Université de Strasbourg, 2013 (cit. on p. 28).

[Meh+00] K. Mehlhorn and S. Thiel. "Faster Algorithms for Bound-Consistency of the Sortedness and the Alldifferent Constraint". In: *Principles and Practice of Constraint Programming – CP 2000*. Ed. by R. Dechter. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 306–319 (cit. on p. 60).

[Mic+83] R. S. Michalski and R. E. Stepp. "Learning from Observation: Conceptual Clustering". In: *Machine Learning: An Artificial Intelligence Approach*. Ed. by R. S. Michalski, J. G. Carbonell, and T. M. Mitchell. Berlin, Heidelberg: Springer Berlin Heidelberg, 1983, pp. 331–363 (cit. on pp. 81, 148).

[Mic80] R. Michalski. *Knowledge Acquisition Through Conceptual Clustering: A Theoretical Framework and an Algorithm for Partitioning Data Into Conjunctive Concepts*. Report (University of Illinois at Urbana-Champaign. Dept. of Computer Science). Department of Computer Science, University of Illinois at Urbana-Champaign, 1980 (cit. on p. 81).

[Moh+86] R. Mohr and T. Henderson. "Arc and Path Consistency Revisited". In: *Artificial Intelligence*, 28 (1986), pp. 225–233 (cit. on p. 60).

[Moh+88] R. Mohr and G. Masini. "Good Old Discrete Relaxation". In: *Proceedings of the 8th European Conference on Artificial Intelligence*. ECAI'88.

Marshfield, MA, USA: Pitman Publishing, Inc., 1988, pp. 651–656 (cit. on p. 60).

[Mot+05] J. Motwani, R. Subramanian, and P. Gopalakrishna. "Critical Factors for Successful ERP Implementation: Exploratory Findings from Four Case Studies". In: *Comput. Ind.* 56(6) (Aug. 2005), pp. 529–544 (cit. on pp. 28, 30, 32, 35).

[NAP05] A. NAPOLI. "Chapter 41 - A SMOOTH INTRODUCTION TO SYMBOLIC METHODS FOR KNOWLEDGE DISCOVERY". In: *Handbook of Categorization in Cognitive Science*. Ed. by H. Cohen and C. Lefebvre. Oxford: Elsevier Science Ltd, 2005, pp. 913 –933 (cit. on p. 75).

[Nem+88] G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization*. New York, NY, USA: Wiley-Interscience, 1988 (cit. on p. 74).

[Nig11] P. Nightingale. "The extended global cardinality constraint: An empirical survey". In: *Artificial Intelligence*, 175(2) (2011), pp. 586 –614 (cit. on p. 96).

[Nov+09] P. K. Novak, N. Lavrac, and G. I. Webb. "Supervised Descriptive Rule Discovery: A Unifying Survey of Contrast Set, Emerging Pattern and Subgroup Mining". In: *Journal of Machine Learning Research*, 10 (2009), pp. 377–403 (cit. on p. 144).

[Oua+16] A. Ouali, S. Loudni, Y. Lebbah, P. Boizumault, A. Zimmermann, and L. Loukil. "Efficiently Finding Conceptual Clustering Models with Integer Linear Programming". In: *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*. 2016, pp. 647–654 (cit. on pp. 13, 14, 19, 81, 83, 86, 99, 108, 112, 130).

[Pac+99] F. Pachet and P. Roy. "Automatic Generation of Music Programs". In: *Principles and Practice of Constraint Programming – CP'99*. Ed. by J. Jaffar. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 331–345 (cit. on p. 65).

[Pan] *Solutions, P.C.: Panorama consulting solutions research report - 2017 ERP Report*. Technical report, Panorama Consulting Solutions (2017), 2017 (cit. on pp. 28, 29).

[Par+14] S. Parthasarathy and S. Sharma. "Determining ERP customization choices using nominal group technique and analytical hierarchy process". In: *Computers in Industry*, 65(6) (2014), pp. 1009 –1017 (cit. on p. 29).

[Par96] V. Pareto. *Cours d'Economie Politique*. Genève: Droz, 1896 (cit. on p. 69).

[Pas+99a] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. "Discovering Frequent Closed Itemsets for Association Rules". In: *Database Theory - ICDT '99, 7th International Conference*. 1999, pp. 398–416 (cit. on p. 80).

[Pas+99b] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. "Efficient mining of association rules using closed itemset lattices". In: *Information Systems*, 24(1) (1999), pp. 25 –46 (cit. on p. 80).

[Pei+00] J. Pei, J. Han, and R. Mao. "CLOSET: An Efficient Algorithm for Mining Frequent Closed Itemsets". In: 2000, pp. 21–30 (cit. on p. 80).

[Pru+16] C. Prud'homme, J.-G. Fages, and X. Lorca. *Choco Documentation*. TASC, INRIA Rennes, LINA CNRS UMR 6241, COSLING S.A.S. 2016 (cit. on pp. 66, 72, 97, 105, 108, 130).

[Pug98] J.-F. Puget. "A Fast Algorithm for the Bound Consistency of Alldiff Constraints". In: *Proceedings of the Fifteenth National/Tenth Conference on Artificial Intelligence/Innovative Applications of Artificial Intelligence*. AAAI '98/IAAI '98. Menlo Park, CA, USA: American Association for Artificial Intelligence, 1998, pp. 359–366 (cit. on p. 60).

[Qui+04] C. Quimper, A. López-Ortiz, P. van Beek, and A. Golynski. "Improved Algorithms for the Global Cardinality Constraint". In: *Principles and Practice of Constraint Programming - CP 2004, 10th International Conference, Proceedings*. Vol. 3258. Lecture Notes in Computer Science. Springer, 2004, pp. 542–556 (cit. on pp. 66, 96).

[Rae+08] L. D. Raedt, T. Guns, and S. Nijssen. "Constraint programming for itemset mining". In: *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Las Vegas, Nevada, USA, August 24-27, 2008*. 2008, pp. 204–212 (cit. on p. 80).

[Rob+11] L. Robert, A. R. Davis, and A. McLeod. "ERP Configuration: Does Situation Awareness Impact Team Performance?" In: *2011 44th Hawaii International Conference on System Sciences (HICSS 2011)*, 00(undefined) (2011), pp. 1–8 (cit. on pp. 28, 30).

[Rol+01] C. Rolland and N. Prakash. "Matching ERP System Functionality to Customer Requirements". In: *International Symposium on Requirements Engineering*. Canada, 2001, p. 1 (cit. on pp. 28, 29, 46).

[Ros+06] F. Rossi, P. v. Beek, and T. Walsh. *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*. New York, NY, USA: Elsevier Science Inc., 2006 (cit. on pp. 55, 60, 72).

[Ros+99] M. Rosemann and J. Wiese. "Measuring the Performance of ERP Software: a Balanced Scorecard Approach". In: (1999) (cit. on p. 25).

[Rég94] J.-C. Régin. "A Filtering Algorithm for Constraints of Difference in CSPs". In: *Proceedings of the Twelfth National Conference on Artificial Intelligence (Vol. 1)*. AAAI '94. Menlo Park, CA, USA: American Association for Artificial Intelligence, 1994, pp. 362–367 (cit. on pp. 60, 65).

[Rég96] J.-C. Régin. "Generalized Arc Consistency for Global Cardinality Constraint". In: *Proceedings of the Thirteenth National Conference on Artificial Intelligence - Volume 1*. AAAI'96. Portland, Oregon: AAAI Press, 1996, pp. 209–215 (cit. on p. 66).

[Sch+13] P. Schaus and R. Hartert. "Multi-Objective Large Neighborhood Search". In: *Principles and Practice of Constraint Programming: 19th International*

*Conference, CP 2013, Uppsala, Sweden, September 16-20, 2013. Proceedings*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 611–627 (cit. on p. 71).

[Sch+17] P. Schaus, J. O. R. Aoga, and T. Guns. "CoverSize: A Global Constraint for Frequency-Based Itemset Mining". In: *Principles and Practice of Constraint Programming*. Ed. by J. C. Beck. Cham: Springer International Publishing, 2017, pp. 529–546 (cit. on p. 80).

[Sha+12] R Sharma, S M. Patil, and A. Tandon. "CUSTOMIZATION AND BEST PRACTICES MODEL FOR ADOPTING ERP SYSTEM: AN ANALYSIS". In: (June 2012) (cit. on p. 29).

[Sko+01] W. Skok and M. Legge. "Evaluating Enterprise Resource Planning (ERP) Systems Using an Interpretive Approach". In: *Proceedings of the 2001 ACM SIGCPR Conference on Computer Personnel Research*. SIGCPR '01. New York, NY, USA: ACM, 2001, pp. 189–197 (cit. on p. 29).

[Sof+05] P. Soffer, B. Golany, and D. Dori. "Aligning an ERP System with Enterprise Requirements: An Object-process Based Approach". In: *Comput. Ind.* 56(6) (2005), pp. 639–662 (cit. on p. 28).

[Soh+00] C. Soh, S. Sia, and J. Tay-Yap. "Enterprise Resource Planning: Cultural Fits and Misfits: Is ERP a Universal Solution?" In: 43 (2000), pp. 47–51 (cit. on p. 29).

[Som+01] T. M. Somers and K. Nelson. "The impact of critical success factors across the stages of enterprise resource planning implementations". In: *Proceedings of the 34th Annual Hawaii International Conference on System Sciences*. 2001, 10 pp.– (cit. on pp. 28, 30).

[Ste+86] R. E. Stepp and R. S. Michalski. "Conceptual clustering of structured objects: A goal-oriented approach". In: *Artificial Intelligence*, 28(1) (1986), pp. 43 –69 (cit. on p. 81).

[tea05] G. team. *Gecode (generic constraint development environment)*. 2005 (cit. on pp. 72, 105, 130).

[Uno+04] T. Uno, T. Asai, Y. Uchida, and H. Arimura. "An Efficient Algorithm for Enumerating Closed Patterns in Transaction Databases". In: *Discovery Science: 7th International Conference, DS 2004, Padova, Italy, October 2-5, 2004. Proceedings*. Ed. by E. Suzuki and S. Arikawa. Springer Berlin Heidelberg, 2004, pp. 16–31 (cit. on pp. 13, 19, 80, 86, 91, 106, 112).

[VH89] P. Van Hentenryck. *Constraint Satisfaction in Logic Programming*. Cambridge, MA, USA: MIT Press, 1989 (cit. on p. 67).

[VIL09] I. VILPOLA. "DEVELOPMENT AND EVALUATION OF A CUSTOMER-CENTERED ERP IMPLEMENTATION METHOD". In: Tampere University of Technology. 2009 (cit. on p. 28).

[Wal+97] M. Wallace, S. Novello, and J. Schimpf. *ECLiPSe: A Platform for Constraint Logic Programming*. 1997 (cit. on p. 72).

[Wal03] T. Walsh. "Consistency and Propagation with Multiset Constraints: A Formal Viewpoint". In: *Principles and Practice of Constraint Programming – CP 2003*. Ed. by F. Rossi. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 724–738 (cit. on p. 66).

[Was+80] L. N. V. Wassenhove and L. F. Gelders. "Solving a bicriterion scheduling problem". In: *European Journal of Operational Research*, 4(1) (1980), pp. 42 –48 (cit. on pp. 69, 136, 138, 141, 154).

[Wu+07] J.-H. Wu, S.-S. Shin, and M. S. Heng. "A methodology for ERP misfit analysis". In: *Information and Management*, 44(8) (2007), pp. 666 –680 (cit. on p. 29).

[YAG93] R. R. YAGER. "On Ordered Weighted Averaging Aggregation Operators in Multicriteria Decisionmaking". In: *Readings in Fuzzy Sets for Intelligent Systems*. Ed. by D. Dubois, H. Prade, and R. R. Yager. Morgan Kaufmann, 1993, pp. 80 –87 (cit. on p. 82).

[Zak+05] M. J. Zaki and C.-J. Hsiao. "Efficient Algorithms for Mining Closed Itemsets and Their Lattice Structure". In: *IEEE Trans. on Knowl. and Data Eng.* 17(4) (2005), pp. 462–478 (cit. on p. 80).

[Zak+99] M. J. Zaki and C.-J. Hsiao. *CHARM: An Efficient Algorithm for Closed Association Rule Mining*. Tech. rep. COMPUTER SCIENCE, RENSSELAER POLYTECHNIC INSTITUTE, 1999 (cit. on p. 80).

[Zak00] M. J. Zaki. "Scalable Algorithms for Association Mining". In: *IEEE Trans. on Knowl. and Data Eng.* 12(3) (2000), pp. 372–390 (cit. on p. 80).

[Zha+01] Y. Zhang and R. H. C. Yap. "Making AC-3 an Optimal Algorithm". In: *Proceedings of the 17th International Joint Conference on Artificial Intelligence - Volume 1*. IJCAI'01. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001, pp. 316–321 (cit. on p. 60).

[Zho+13] N.-F. Zhou and H. Kjellerstrand. *A User s Guide to Picat*. 2013 (cit. on p. 72).

FOLIO ADMINISTRATIF

THÈSE DE L'UNIVERSITÉ DE LYON OPÉRÉE AU SEIN DE L'INSA LYON

**Nom :** Chabert **Date de soutenance :** 18 Décembre 2018
**Prénom :** Maxime
**Titre :** Constraint Programming Models for Conceptual Clustering : Application to an ERP Configuration Problem
**Nature :** Doctorat **Numéro d'ordre :** 2018LYSEI118
**École doctorale :** InfoMaths
**Spécialité :** Informatique
**Résumé :**

Les ERP (Enterprise Resource Planning) sont incontournables dans les systèmes d'information des sociétés industrielles : ils jouent un rôle crucial pour automatiser et suivre leurs processus afin d'améliorer leur compétitivité. Un ERP est un logiciel générique qui est utilisé par plusieurs sociétés industrielles ayant des besoins et des processus différents. C'est pourquoi de nombreux paramètres permettent d'adapter le fonctionnement du système aux besoins d'une société. Cette thèse vise à simplifier le paramétrage d'un ERP. Nous proposons une approche visant à extraire, depuis l'ensemble des paramétrages existants, un catalogue de paramétrages correspondant à des besoins fonctionnels précédemment rencontrés afin de les réutiliser lors des prochains déploiements de Copilote. Nous proposons d'utiliser la programmation par contraintes pour cela, afin de pouvoir facilement personnaliser les solutions calculées en ajoutant des contraintes et des critères d'optimisation variés. Nous introduisons de nouveaux modèles à base de contraintes pour résoudre des problèmes de clustering conceptuel, ainsi qu'une contrainte globale pour le problème de couverture exacte avec plusieurs algorithmes de propagation.

**Mots-clefs :** ERP - Conceptual clustering - Constraint programming
**Laboratoire de recherche :** LIRIS
**Directrice de thèse :** Christine Solnon
**Président de jury :**
**Composition du jury :**
Thi-Bich-Hanh Dao
Christian Bessière
Elisa Fromont
Valérie Botta-Genoulaz
Christian Schulte
Christine Solnon
Pierre-Antoine Champin