Apprentissage automatique de cartes d'invariants d'objets combinatoires avec une application pour la synthèse d'algorithmes de filtrage

Jovial Cheukam Ngouonou

COLLEGES SCIENCES
BRETAGNE POUR L'INGENIEUR
LOIRE ET LE NUMERIQUE

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

UNIVERSITÉ LAVAL

THÈSE DE DOCTORAT DE

L'ÉCOLE NATIONALE SUPÉRIEURE
MINES-TÉLÉCOM ATLANTIQUE BRETAGNE
PAYS DE LA LOIRE – IMT ATLANTIQUE EN COTUTELLE AVEC
L'UNIVERSITÉ LAVAL

ÉCOLE DOCTORALE N° 648
Sciences pour l'Ingénieur et le Numérique
Spécialité : *Sciences et technologies de l'information et de la communication*

Par
Jovial CHEUKAM NGOUONOU

Apprentissage automatique de cartes d'invariants d'objets combinatoires avec une application pour la synthèse d'algorithmes de filtrage

Thèse présentée et soutenue à Université Laval, Québec, le 3 octobre 2024
Unité de recherche : Laboratoire LS2N
Thèse N° : 2024IMTA0418

Rapporteurs avant soutenance :

François Fages	Directeur de recherche INRIA
Christophe Lecoutre	Professeur CRIL Lens

Composition du Jury :

Présidente :	Nadia Tawbi	Professeure, Université Laval
Examinateurs :	Christine Solnon	Professeure INSA de Lyon
	François Fages	Directeur de recherche INRIA
	Christophe Lecoutre	Professeur CRIL Lens
Dir. de thèse :	Nicolas Beldiceanu	Professeur, IMT Atlantique, LS2N
Co-Dir. de thèse :	Claude-Guy Quimper	Professeur, Université Laval

Apprentissage automatique de cartes d'invariants d'objets combinatoires avec une application pour la synthèse d'algorithmes de filtrage

Thèse en cotutelle
Doctorat en informatique

Jovial Cheukam Ngouonou

Sous la direction de:

Nicolas Beldiceanu, Directeur de cotutelle
Claude-Guy Quimper, Directeur de recherche
Rémi Douence, Co-encadrant de la thèse

Résumé

Pour améliorer l'efficacité des méthodes de résolution de nombreux problèmes d'optimisation combinatoires de notre vie quotidienne, nous utilisons la programmation par contraintes pour générer automatiquement des conjectures. Ces conjectures caractérisent des objets combinatoires utilisés pour modéliser ces problèmes d'optimisation. Ce sont notamment les graphes, les arbres, les forêts, les partitions et les séquences. Contrairement à l'état de l'art, le système, dénommé Bound Seeker, que nous avons élaboré ne génère pas seulement de manière indépendante les conjectures, mais il explicite aussi des liens existant entre les conjectures. Ainsi, il regroupe les conjectures sous forme de bornes précises sur une même variable associée à un même objet combinatoire. Ce regroupement est appelé carte de bornes de l'objet combinatoire considéré. Enfin, une étude consistant à établir des liens entre les cartes générées est faite. Le but de cette étude est d'approfondir les connaissances sur les objets combinatoires et de développer des prémices de preuves automatiques des conjectures. Pour montrer la cohérence des cartes générées par le Bound Seeker, nous élaborons quelques preuves manuelles des conjectures découvertes par le Bound Seeker, ce qui permet de démontrer la pertinence de quelques nouveaux théorèmes de bornes. Pour illustrer l'une des utilités pratiques de ces bornes, nous introduisons une méthode de génération semi-automatique d'algorithmes de filtrage qui réduisent l'espace de recherche des solutions d'un problème d'optimisation combinatoire. Cette réduction est faite grâce aux nouveaux théorèmes de bornes que nous avons établis après les avoir sélectionnés automatiquement parmi les conjectures générées par le Bound Seeker. Pour montrer l'efficacité de cette technique, nous l'appliquons avec succès au problème d'élaboration des cursus académiques équilibrés d'étudiants.

Abstract

To improve the efficiency of solution methods for many combinatorial optimisation problems in our daily lives, we use constraints programming to automatically generate conjectures. These conjectures characterise combinatorial objects used to model these optimisation problems. These include graphs, trees, forests, partitions and Boolean sequences. Unlike the state of the art, the system, called Bound Seeker, that we have developed not only generates conjectures independently, but it also points to links between conjectures. Thus, it groups the conjectures in the form of bounds of the same variable characterising the same combinatorial object. This grouping is called a bounds map of the combinatorial object considered. Then, a study consisting of establishing links between generated maps is carried out. The goal of this study is to deepen knowledge on combinatorial objects and to develop the beginnings of automatic proofs of conjectures. Then, to show the consistency of the maps and the Bound Seeker, we develop some manual proofs of the conjectures discovered by the Bound Seeker. This allows us to demonstrate the usefulness of some new bound theorems that we have established. To illustrate one of its concrete applications, we introduce a method for semi-automatic generation of filtering algorithms that reduce the search space for solutions to a combinatorial optimisation problem. This reduction is made thanks to the new bound theorems that we established after having automatically selected them from the conjectures generated by the Bound Seeker. To show the effectiveness of this technique, we successfully apply it to the problem of developing balanced academic courses for students.

Table des matières

Liste des tableaux

Liste des figures

Listes des morceaux de cartes de conjectures

Cette section présente la liste des morceaux de cartes de conjectures illustrées dans toute la thèse :

— Figure 2.1 : Carte de deux bornes supérieures précises du nombre d'arcs d'un graphe orienté.

— Figure 2.2 : Carte de quatre bornes supérieures précises du nombre d'arcs d'un graphe orienté.

— Figure 2.3 : Carte de quatre bornes supérieures précises d'une partition d'ensembles.

— Figure 2.4 : Carte de bornes supérieures de la taille maximale d'une composante connexe d'un graphe orienté.

— Figure 2.7 : Carte de bornes inférieures précises du nombre de feuilles d'un arbre enraciné.

— Figure 2.8 : Carte de 13 bornes supérieures précises du nombre d'arcs d'un graph orienté.

— Figure 2.9 : Carte de conjectures de bornes supérieures précises du nombre de variables valant 1 dans une séquence de booléens.

— Figure 3.1 : Corespondance entre la carte inversée de bornes inférieures de la somme des carrés des tailles des partitions d'un ensemble $\mathcal{M}^{S\geq}_{\{n,\overline{M}\}}$ et la carte inverse de bornes supérieures du cardinal de l'ensemble partitionné $\mathcal{M}^{n\leq}_{\{S,\overline{M}\}}$ par inversion de conjectures.

— Figure 3.2 : Carte inverse $\mathcal{W}^{P\leq}_{\{n,\underline{M},S\}}$ de bornes supérieures du nombre de partitions P d'un ensemble partitionné, obtenue en inversant la carte $\mathcal{M}^{S\leq}_{\{n,P,\underline{M}\}}$ de bornes supérieures de la somme des carrés des tailles des partitions d'un ensemble et la carte $\mathcal{M}^{\underline{M}\leq}_{\{n,P\}}$ de bornes supérieures de la taille minimale d'une partition.

— Figure 3.5 : Correspondance entre la carte $\mathcal{M}^{f\ \geq}_{\{v,\underline{d},\overline{d}\}}$ de bornes inférieures du nombre de feuilles d'un arbre enraciné et la carte $\mathcal{M}^{P\ \leq}_{\{n,\underline{M},\overline{M}\}}$ de bornes supérieures du nombre de partitions d'un ensemble partitionné par encodage de l'objet combinatoire que constitue la partition d'ensembles par l'objet combinatoir que constitue l'arbre enraciné.

Listes des conjectures prouvées

Cette section présente la liste des 9 conjectures découvertes par le Bound Seeker et prouvées manuellement au chapitre 4 :

— Conjecture 4 : Cette conjecture exprime la borne inférieure précise du nombre de composantes connexes c d'un graphe orienté en fonction de son nombre de sommets v, de la taille de la plus grande composante connexe et de celle de la plus petite composante fortement connexe.

— Conjecture 5 : Cette conjecture exprime la borne inférieure précise du nombre de composantes connexes c d'un graphe orienté en fonction de son nombre de composantes fortement connexes s, de la taille $\overline{c}$ de la plus grande composante connexe et de la taille $\underline{s}$ de la plus petite composante fortement connexe.

— Conjecture 6 : Dans le contexte d'un graphe orienté, cette conjecture exprime la borne inférieure précise de la taille $\overline{s}$ de la plus grande composante fortement connexe en fonction du nombre v de sommets, du nombre s de composantes fortement connexes et de la taille $\underline{s}$ de la plus petite composante fortement connexe.

— Conjecture 7 : Dans le contexte d'un graphe orienté, cette conjecture exprime la borne inférieure précise de la taille $\overline{c}$ de la plus grande composante connexe en fonction de la taille $\underline{c}$ de la plus petite composante connexe, du nombre v de sommets, du nombre c de composantes connexes et de la taille $\overline{s}$ de la plus grande composante fortement connexe.

— Conjecture 8 : Cette conjecture exprime la borne inférieure précise et la borne supérieure précise du nombre f de feuilles d'un arbre enraciné en fonction du nombre v de sommets, du nombre minimal $\underline{d}$, et du nombre maximal $\overline{d}$ de fils d'un nœud n'étant pas une feuille de l'arbre.

— Conjecture 9 : Cette conjecture exprime la borne inférieure précise de la somme S des carrés des tailles partitions d'un ensemble en fonction de la cardinalité n de l'ensemble, du nombre P de partitions, et des tailles maximale $\overline{M}$ et minimale $\underline{M}$ d'une partition.

— Conjecture 11 : Cette conjecture exprime la borne supérieure précise de la somme S des carrés des tailles des partitions d'un ensemble en fonction de la cardinalité n de l'ensemble, du nombre P de partitions, et des tailles minimale $\underline{M}$ et maximale $\overline{M}$ d'une partition.

— Conjecture 12 : Cette conjecture exprime la borne supérieure précise de la différence $\overline{M}$ entre la taille de la plus grande partition et la taille de la plus petite partition en fonction du cardinal n de l'ensemble partitionné, du nombre P de partitions, et de la taille minimale $\underline{M}$ d'une partition.

— Conjecture 13 : Cette conjecture exprime la borne supérieure précise de la différence $\overline{M}$ entre la taille de la plus grande partition et celle de la plus petite en fonction du cardinal n de l'ensemble partitionné, du nombre P de partitions et de la taille maximale $\overline{M}$ d'une partition.

Je dédie cette thèse à mon défunt père Cheukam et ma défunte sœur Cheukam Tchouanche Camille. Je sais que vous auriez été fiers de moi si vous étiez encore là en ce jour tant attendu.

Remerciements

Je transmets mes sincères infinis remerciements à mes directeurs de thèse Nicolas Beldiceanu et Claude-Guy Quimper. Leurs mentorats, exigences, conseils, écoutes, soutiens et humanismes m'ont permis d'accéder à un grand savoir en optimisation combinatoire, en programmation par contraintes, en techniques de preuves et méthodologie de la recherche.

Je remercie aussi infiniment mon co-encadrant de thèse Rémi Douence. Ses exigences, conseils, écoutes, soutiens et son humanisme m'ont permis de parfaire mes connaissances en techniques de preuves et méthodologie de la recherche.

Je voudrais également remercier Christophe Lecoutre, François Fages, Nadia Tawbi, Christine Solnon, Catherine Dubois, Gilles Caporossi pour leurs examens constructifs de mes travaux de recherche. Leurs conseils et encouragements m'ont permis d'aller de l'avant.

Je remercie également mon cher collègue, collaborateur et ami Ramiz Guindullin qui m'a été d'une grande aide dans la quête d'excellents résultats de recherche.

J'adresse également mes remerciements à mes collègues et amis de bureaux. Je pense entre autre à Frédéric Berthiaume, Samuel Cloutier, Anthony Deschênes, Manuel Chastenay, Antoine Laviolette, Jean-Thomas Sexton, Charles Verneray, Sandrine Blais-Deschênes, Christopher Coulomb, Paula Metzker. Grâce à votre humour, votre bonne humeur, votre disponibilité et votre soutien, j'ai pu affronté l'adversité qui s'est présentée durant ces études de doctorat.

Je souligne également le support constant de ma famille. Merci à ma mère Dombou Suzanne, mon frère Cheukam Dawak Yannick Bernard, ma sœur Cheukam Sitta Pierrette Sandra pour leur soutien émotionnel qui m'a aidé à tenir bon du début jusqu'à la fin.

Je remercie enfin tous ceux qui ont contribués de près ou de loin à cet aboutissement. Je pense notamment à Hervé Grall et Marie-Hélène Beldiceanu.

Introduction

Depuis l'avènement des sociétés et en vue de l'amélioration des conditions de vie, l'humanité se confronte à des questions de prise de meilleures décisions face à des problèmes à grande échelle. Parmi ces problèmes figurent ceux qui généralement se dénomment problèmes d'optimisation combinatoire. Il s'agit de problèmes nécessitant la recherche de solutions optimales dans un ensemble discret et fini de solutions réalisables. Il s'agit par exemple de problèmes de planification optimale de gros projets tels que la programmation des cursus scolaires des étudiants dans les universités, la répartition de tâches dans toutes les grandes organisations telles que les usines, les industries, les hôpitaux et les gouvernements.

L'enjeu est que, généralement, l'ensemble des solutions réalisables est très grand et par là même, nécessite une description implicite à l'aide d'un modèle à contraintes satisfaites par les solutions réalisables. Ainsi, pour trouver une solution optimale au problème combinatoire, il faut en théorie comparer la qualité de toutes les solutions pour en tirer la meilleure. Mais en pratique, le nombre de solutions réalisables est si grand que le temps pour rechercher toutes les solutions réalisables fait défaut. C'est cette raison qui fait que les problèmes d'optimisation combinatoires sont réputés difficiles.

Malgré leurs difficultés de résolution, les problèmes d'optimisation combinatoires sont abordés par plusieurs approches dans la littérature. L'une d'elles est l'utilisation des solveurs de contraintes. Un solveur de contraintes est un programme qui construit progressivement des solutions réalisables en effectuant les choix compatibles avec les contraintes et en utilisant une méthode de recherche prédéfinie appelée heuristique. Cependant, pour être efficace, un solveur le plus souvent s'appuie sur des algorithmes de filtrage associés à chaque contrainte.

Un algorithme de filtrage est spécifique à une des contraintes du problème d'optimisation combinatoire. C'est un algorithme dont le but est de réduire l'espace de recherche des solutions réalisables d'un problème d'optimisation combinatoire. Plus précisément, l'algorithme de filtrage s'appuie sur une des contraintes descriptives du problème d'optimisation combinatoire afin de réduire le nombre de choix possibles qui mènent aux solutions réalisables. Et plus les contraintes caractérisent le mieux le problème, plus le filtrage est efficace, conduisant à une meilleure résolution par le solveur. Tout cela nécessite donc la connaissance des bonnes contraintes qui décrivent le mieux le problème. De telles contraintes échappent le plus sou-

vent à l'humain chargé de modéliser le problème. De ce fait, il existe dans la littérature des méthodes d'apprentissage automatique de contraintes des problèmes combinatoires. Certaines apprennent des contraintes spécifiques à un problème donné tandis que d'autres apprennent des contraintes plus générales sous la forme de conjectures mathématiques à démontrer formellement pour au final devenir applicables à une large gamme de problèmes. Rappelons qu'une conjecture en mathématiques est une affirmation qui, bien qu'on ne connaisse pas encore de preuve, semble apparemment vraie.

La nécessité de démontrer formellement les conjectures apprises repose sur le fait que l'apprentissage se fait sur un nombre limité de données. Afin d'augmenter fortement les chances d'apprendre des conjectures vraies, les données d'apprentissage doivent remplir un certain nombre de conditions sortant du cadre courant de l'apprentissage automatique. En l'occurrence :

— les données ne doivent pas être erronées.
— seules les données qui conduisent aux théorèmes les plus généraux doivent être sélectionnées.

Les contraintes bornant un paramètre par rapport à d'autres paramètres sont présentes dans nombre de problèmes combinatoires. Ces contraintes permettent de délimiter l'espace de recherche des solutions, ce qui améliore la recherche des solutions. Cette recherche est d'autant plus efficace que les bornes sont précises. Malheureusement, il est difficile d'acquérir des bornes précises prenant en compte l'intégralité des paramètres du problème à résoudre. De plus, l'obtention de telles bornes est dans l'immense majorité des cas un processus manuel fastidieux.

Cette thèse s'intéresse donc à la découverte automatique de conjectures mathématiques sur des données exactes caractérisant les objets combinatoires qui le plus souvent interviennent dans la modélisation des problèmes d'optimisation combinatoire.

La thèse se concentre sur la génération automatique de conjectures de bornes précises d'objets combinatoires. Lorsque ces conjectures de bornes précises sont formellement démontrées, elles sont appelées invariants de bornes. L'objectif principal de la thèse est l'amélioration de l'efficacité des solveurs en procédant à l'apprentissage automatique de cartes d'invariants de bornes dans le cadre d'objets combinatoires, pour la synthèse d'algorithmes de filtrage efficaces qui sont si chers aux solveurs. En effet, une carte d'invariants de bornes peut être vue comme un graphe dans lequel les sommets sont des invariants de bornes et les arcs correspondent à la relation entre deux invariants de bornes donnés. Cette relation montre comment un invariant de borne se dérive à partir d'un invariant de borne plus général en éliminant un paramètre.

Pour atteindre cet objectif, la thèse s'est défini les trois sous-objectifs suivants :

— l'apprentissage automatique de cartes de conjectures d'objets combinatoires sur des don-

nées exactes.

— l'élaboration manuelle des preuves formelles des conjectures découvertes et contenues dans les cartes.

— la génération semi-automatique d'algorithmes de filtrage à partir des cartes de conjectures.

Nous avons atteint ces trois sous-objectifs qui constituent donc les contributions générales de nos travaux. Ainsi, nous apportons des contributions scientifiques sur trois axes qui sont l'axe de la découverte automatique des conjectures, l'axe de l'automatisation des preuves de conjectures et l'axe de la résolution efficace des problèmes combinatoires en utilisant la programmation par contraintes.

Avant de présenter les contributions réalisées, nous présentons l'état de l'art et pointons les limites des méthodes existantes de génération et d'exploitation de conjectures dans le chapitre 1. Puis les quatre autres chapitres présentent de manière détaillée les contributions de ce travail. Notamment, le chapitre 2 présente l'apprentissage des conjectures sous forme de cartes en se basant sur la programmation par contraintes. Le chapitre 3 met en évidence des connexions exploitables existantes entre les cartes de conjectures et montre les prémices de techniques d'automatisation des preuves de conjectures. Le chapitre 4 présente des preuves manuelles de quelques conjectures découvertes du chapitre 2 dont certaines sont immédiatement utilisées au chapitre 5. Le chapitre 5 présente une méthode de génération semi-automatique d'algorithme de filtrage de contraintes globales.

Chapitre 1

Concepts préliminaires

De nombreux problèmes combinatoires apparaissent dans notre vie quotidienne au point où s'impose le besoin de les résoudre de manière systématique. Et pour les résoudre, on fait généralement appel à des concepts tels que l'optimisation combinatoire et la programmation par contraintes. Comme nous nous intéressons à la résolution efficace des problèmes combinatoires, voici une présentation des concepts que nous utilisons dans le cadre de notre travail de recherche. Ainsi, ce chapitre vise dans un premier temps à présenter les concepts de base et le cadre de la thèse, à définir les notions de base de l'optimisation combinatoire et de la génération automatique des conjectures. Puis dans un deuxième temps, il vise à exposer les limites de l'état de l'art sur lesquelles se justifient les objectifs et les contributions de la thèse.

1.1 L'optimisation combinatoire

Pour résoudre de manière systématique un problème combinatoire, on peut le formaliser en problème de satisfaction de contraintes. Un *problème de satisfaction de contraintes* est un problème dont une solution doit satisfaire un certain nombre de contraintes caractérisant le problème.

Définition 1 (Problème de satisfaction de contraintes)**.** Un *problème de satisfaction de contraintes* est défini par un triplet $\mathcal{P} = \langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$ où :

- $\mathcal{X}$ est un ensemble de n variables ;
- $\mathcal{D}$ est un ensemble de n domaines où un élément $\mathcal{D}(X)$ est le domaine d'une variable X de $\mathcal{X}$. Le domaine $\mathcal{D}(X)$ de X est l'ensemble fini des valeurs possibles de X ;
- $\mathcal{C}$ est un ensemble de contraintes. Chaque *contrainte* C est elle-même définie comme une paire $\langle \mathcal{X}(C), \mathcal{R}(C) \rangle$ où :
  1. $\mathcal{X}(C)$ constitue l'ensemble des variables de la contrainte C et est appelé *portée de la contrainte* C. C'est-à-dire : $\mathcal{X}(C) = \{X_1, \cdots, X_{n_C}\}$ où X_i est une variable de $\mathcal{X}$ utilisée pour exprimer la contrainte $C, \forall i \in \{1, \cdots, n_C\}$.

2. $\mathcal{R}(C)$ est une relation entre les variables de $\mathcal{X}(C)$. C'est-à-dire :

$$\mathcal{R}(C) \subseteq \prod_{X \in \mathcal{X}(C)} \mathcal{D}(X) \tag{1.1}$$

La relation $\mathcal{R}(C)$ définit l'ensemble des n_C-uplets qui satisfont la contrainte C. Les n_C-uplets sont de la forme $(x_1, \cdots, x_{n_C})$ telle que $x_i \in \mathcal{D}(X_i)\ \forall i \in \{1, \cdots, n_C\}$. Puis, lorsque $X_i = x_i\ \forall i \in \{1, \cdots, n_C\}$, la contrainte C est vérifiée. La relation $\mathcal{R}(C)$ peut s'exprimer sous la forme d'une égalité, d'une inégalité ou d'un prédicat dont les inconnues sont les variables de $\mathcal{X}(C)$. Des contraintes d'inégalités sont données par l'exemple 2. Une contrainte sous forme de prédicat est donnée par l'exemple 1.

Exemple 1. La contrainte AllDifferent(X_1, X_2, X_3) est un prédicat qui est satisfait lorsque les variables X_1, X_2, X_3 prennent des valeurs distinctes. Par exemple, si nous avons les domaines $\mathcal{D}(X_1) = \{1, 2\}, \mathcal{D}(X_2) = \{1, 3\}, \mathcal{D}(X_3) = \{4, 3\}$, alors AllDifferent$(X_1, X_2, X_3)$ est satisfait lorsque $X_1 = 1, X_2 = 3, X_3 = 4$.

L'ensemble des $|\mathcal{X}|$-uplets qui est défini par $\prod_{X \in \mathcal{X}} \mathcal{D}(X)$ est appelé *espace de recherche* des solutions du problème de satisfaction de contraintes.

Définition 2 (Solutions d'un problème de satisfaction de contraintes)**.** Soit $\mathcal{P} = \langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$ un problème de satisfaction de contraintes et soit $\mathcal{X}(C) = \{X_1, \cdots, X_{n_C}\}$ la portée d'une contrainte C de $\mathcal{P}$. On appelle *assignation des variables* de $\mathcal{X}$ une fonction $\sigma : \mathcal{X} \longrightarrow \bigcup_{X \in \mathcal{X}} \mathcal{D}(X)$ qui à toute variable X de $\mathcal{X}$ associe une valeur du domaine $\mathcal{D}(X)$. C'est-à-dire que $\forall X \in \mathcal{X}, \sigma(X) \in \mathcal{D}(X)$. On dit qu'une assignation *satisfait* une contrainte C si pour chacune des valeurs associées à chacune des variables de $\mathcal{X}(C)$ nous avons le fait que le n_C-uplet $(\sigma(X_1), \cdots, \sigma(X_{n_C}))$ appartienne à la relation $\mathcal{R}(C)$. On dit également que le *tuple* $[\sigma(X_1), \cdots, \sigma(X_{n_C})]$ *satisfait* la contrainte C. Une assignation est une *solution* de $\mathcal{P}$ si elle *satisfait toutes* les contraintes de $\mathcal{D}$. Nous notons par $Sol(\mathcal{P})$ l'ensemble des solutions de $\mathcal{P}$.

Un *problème d'optimisation combinatoire* est un problème qui peut se résoudre en parcourant un espace discret pour trouver la meilleure des solutions parmi les solutions réalisables qui s'y trouvent. Une *solution réalisable* est une solution satisfaisant toutes les contraintes définies par le problème, mais sans être nécessairement la meilleure. L'espace discret est fini et souvent très grand.

Pour prendre en compte la notion de qualité d'une solution, on utilise une fonction, appelée fonction objectif. Elle retourne pour chaque solution réalisable une valeur. Une solution est meilleure lorsqu'elle minimise ou maximise la fonction objectif.

Définition 3 (Problème d'optimisation combinatoire)**.** Un problème d'optimisation combinatoire est défini par le quadruplet $\mathcal{P}(Obj) = \langle \mathcal{X}, \mathcal{D}, \mathcal{C}, Obj \rangle$ où $Obj : \prod_{X \in \mathcal{X}} \mathcal{D}(X) \longrightarrow \mathbb{R}$ est

une fonction dont il faut optimiser la valeur. Cette fonction associe à une valeur du domaine de chacune des variables de $\mathcal{X}$ un nombre réel. Cette fonction est appelée *fonction objectif*. On appelle *solution d'un problème d'optimisation combinatoire* $\mathcal{P}(Obj) = \langle \mathcal{X}, \mathcal{D}, \mathcal{C}, Obj \rangle$ toute solution σ_{opt} du problème de satisfaction de contraintes $\mathcal{P} = \langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$ correspondante et pour laquelle la valeur de la fonction objectif Obj est *optimale*. Si l'optimisation est un problème de maximisation, alors :

$$\sigma_{opt} \in \underset{\sigma \in Sol(\mathcal{P})}{\operatorname{argmax}} \; Obj(\sigma(X_1), \cdots, \sigma(X_{|\mathcal{X}|})) \tag{1.2}$$

Si l'optimisation est un problème de minimisation, alors :

$$\sigma_{opt} \in \underset{\sigma \in Sol(\mathcal{P})}{\operatorname{argmin}} \; Obj(\sigma(X_1), \cdots, \sigma(X_{|\mathcal{X}|})) \tag{1.3}$$

Pour illustrer notre propos, nous donnons dans l'exemple 2 un problème d'optimisation combinatoire connu dans la littérature.

Exemple 2 (La répartition équilibrée des sessions)**.** Une répartition équilibrée de cours sur un ensemble donné de sessions est un problème d'optimisation combinatoire connu dans la littérature [40]. Dans ce problème, étant donné un ensemble de cours avec leurs crédits et prérequis associés, il s'agit de déterminer la répartition la plus équilibrée des cours, telle que les valeurs des totaux de crédits des cours pour chacune des sessions soient les plus proches. Un exemple d'instance du problème consiste à répartir sept cours avec leurs crédits associés entre les trois sessions d'hiver, d'été et d'automne. Les cours sont : les mathématiques, la physique, le français, l'anglais, la chimie, le sport, la biologie. Et les contraintes de prérequis sont telles que pour suivre le cours de mathématiques, il faut avoir suivi le cours de français. Pour suivre le cours de chimie, il faut avoir suivi les cours de mathématiques et d'anglais. De plus, un étudiant ne peut pas cumuler plus de 7 crédits de cours dans une session. La table 1.1 représente une solution à cette instance du problème. Comme les totaux de crédits par session sont égaux, nous disons que l'équilibrage est parfait. Pour cette instance, une solution réalisable est une répartition de sept cours en trois groupes distincts, c'est-à-dire trois parties d'une partition. Donc l'espace des solutions réalisables a $S(7,3) = \frac{1}{3!}\sum_{j=0}^{3}(-1)^{3-j}\begin{pmatrix} 3 \\ 1 \end{pmatrix} j^3 = 301$ éléments. Il s'agit de la formule du nombre de Stirling de second espèce [21]. La qualité d'une solution correspond à la qualité de l'équilibrage des crédits par session. Pour la mesurer, on utilise généralement l'une des fonctions objectifs suivantes :

— le plus grand cumul de crédits parmi les sessions ;
— la différence entre le plus grand cumul de crédits et le plus petit cumul de crédits parmi les sessions ;
— la somme des carrés des cumuls des crédits par session.

Une solution est meilleure lorsqu'elle minimise l'une de ces trois fonctions objectifs.

Session d'hiver	Crédits
Français	3
Physique	3
Cumul de crédits	6

Session d'été	Crédits
Mathématiques	4
Anglais	2
Cumul de crédits	6

Session d'automne	Crédits
Chimie	2
Sport	2
Biologie	2
Cumul de crédits	6

TABLE 1.1 – Équilibrage parfait d'une répartition de sept cours sur trois sessions.

Nous formalisons cette instance de la répartition équilibrée de cours sur un ensemble donné de sessions en problème d'optimisation combinatoire comme suit :

On attribue aux cours de français, physique, mathématiques, anglais, chimie, sport et biologie, respectivement les numéros 1 à 7. On attribue également les numéros 1 à 3 respectivement aux sessions d'hiver, été et automne.

Soit X_i la session à laquelle est dispensée le cours i et $c(i)$ son nombre de crédits sachant que $i \in \{1, \cdots, 7\}$. $\forall i \in \{1, \cdots, 7\}, \forall j \in \{1, 2, 3\}$, soit l'expression $[X_i = j]$ définie par :

$$[X_i = j] = \begin{cases} 1 \text{ si } X_i = j \text{ est vrai} \\ 0 \text{ sinon} \end{cases} \tag{1.4}$$

Alors l'instance de la répartition équilibrée de cours sur un ensemble donné de sessions peut être formalisée par le problème d'optimisation combinatoire $\mathcal{P} = \langle \mathcal{X}, \mathcal{D}, \mathcal{C}, Obj \rangle$ suivant :

$$\text{Minimiser } Obj(X_1, \cdots, X_7) \tag{1.5}$$

$$\mathcal{X} = \{X_1, \cdots, X_7\} \tag{1.6}$$

$$\forall i \in \{1, \cdots, 7\}, \mathcal{D}(X_i) = \{1, 2, 3\} \tag{1.7}$$

$$\mathcal{C} : \forall j \in \{1, 2, 3\}, \sum_{i=1}^{7} [X_i = j] \cdot c(i) \leq 7 \tag{1.8}$$

$$X_1 < X_3 \tag{1.9}$$

$$X_3 < X_5 \tag{1.10}$$

$$X_4 < X_5 \tag{1.11}$$

L'inégalité (1.8) exprime la contrainte que le cumul de crédits de chaque session ne peut dépasser sept crédits. L'inégalité (1.9) exprime la contrainte que le cours de français doit être

suivi avant le cours de mathématiques. Les inégalités (1.10) et (1.11) expriment les contraintes que les cours de Mathématiques et anglais doivent être suivis avant le cours de chimie.

La fonction objectif Obj qu'il faut minimiser est l'une des fonctions objectifs introduites dans l'exemple 2. Pour calculer ces fonctions objectifs, nous introduisons les variables suivantes :

$$\forall j \in \{1,2,3\}, O_j = \sum_{i=1}^{7} [X_i = j] \cdot c(i) \tag{1.12}$$

$$\underline{M} = \min(O_1, O_2, O_3) \tag{1.13}$$

$$\overline{M} = \max(O_1, O_2, O_3) \tag{1.14}$$

O_j représente le cumul de crédits dans la session j. Ainsi, nous avons les trois fonctions objectifs suivantes :

— le plus grand cumul de crédits entre les sessions $Obj(X_1, \cdots, X_7) = \overline{M}$;
— la différence entre le plus grand cumul de crédits et le plus petit cumul de crédits entre les sessions $Obj(X_1, \cdots, X_7) = \overline{M} - \underline{M}$;
— la somme des carrés des cumuls des crédits par session $Obj(X_1, \cdots, X_7) = O_1^2 + O_2^2 + O_3^2$.

L'une de ces trois fonctions peut donc être minimisée.

Une solution à ce problème de satisfaction de contraintes $\mathcal{P}$ est donnée par la Table 1.1. Elle est représentée par le 7-uplet $(X_1, \cdots, X_7) = (1,1,2,2,3,3,3)$. Pour trouver une telle solution, c'est-à-dire pour résoudre un problème d'optimisation combinatoire, on peut faire appel à la programmation par contraintes que nous introduisons maintenant.

1.1.1 La programmation par contraintes

La programmation par contraintes est un paradigme déclaratif utilisé pour résoudre des problèmes combinatoires de grandes tailles en évitant de programmer un algorithme spécifique pour chaque problème. Elle comporte les deux étapes suivantes :

— formaliser le problème combinatoire en problème de satisfaction de contraintes ou problème d'optimisation combinatoire en utilisant des contraintes caractérisant le problème ;
— utiliser un solveur pour faire une recherche systématique d'une solution. Généralement, le solveur utilise les algorithmes de filtrage associés à chaque contrainte prédéfinie. Ces algorithmes de filtrage de contraintes permettent la réduction de la taille de l'espace de recherche des solutions. Ainsi, pour trouver une solution, le solveur énumère seulement les assignations n'ayant pas été filtrées par les algorithmes de filtrage.

Remarque 1. Les *algorithmes de filtrage* de contraintes sont aussi appelés *propagateurs* de contraintes et constituent un point crucial de la programmation par contraintes.

Pour mieux expliquer l'objectif de notre recherche, nous présentons par la suite un outil essentiel en programmation par contraintes. Il s'agit du solveur de contrainte qui exploite les éléments fournis par les deux étapes de la programmation par contraintes, citées précédemment. Ces éléments fournis sont notamment : le modèle à contraintes du problème combinatoire, le propagateur de contraintes dédié au modèle et la méthode de recherche systématique d'une solution au problème.

1.1.2 Le solveur de contraintes

Définition 4. Le *solveur de contraintes* est un logiciel prenant en entrée un problème combinatoire sous la forme d'un problème de satisfaction de contraintes ou d'un problème d'optimisation combinatoire qui décrivent la solution devant être retournée. Pour ce faire, il parcourt systématiquement l'espace de recherche des solutions. Le parcours s'effectue en construisant :

— soit un arbre tel que chaque nœud représente une assignation de l'une des variables des contraintes du problème.
— soit un arbre binaire où pour une valeur v du domaine $\mathcal{D}(X)$ d'une variable X, les deux fils d'un nœud sont les contraintes $X = v$ et $X \neq v$.

Cette méthode de recherche d'une solution est appelée *l'exploration d'un arbre de recherche* que nous présentons maintenant.

L'exploration d'un arbre de recherche

La méthode de recherche d'une solution que constitue l'exploration d'un arbre de recherche s'effectue par une construction d'un arbre tel que chaque nœud représente une solution partielle, c'est-à-dire une solution où seules quelques variables sont affectées à une valeur. La racine de l'arbre est une solution partielle vide où aucune variable n'est affectée. Chaque nœud hérite de la solution partielle de son parent et y ajoute une affectation supplémentaire. De plus, à chaque nœud de l'arbre, les actions suivantes sont réalisées :

— le choix d'une variable auquel le solveur affecte une valeur de son domaine ;
— le test si aucune contrainte n'a été violée après l'affectation d'une valeur à la variable courante. Si une contrainte est violée, le solveur annule l'assignation faite et réaffecte une autre valeur à la même variable sur un nouveau nœud. On parle alors de *retour arrière*, dû à un échec d'assignation, fait par le solveur. Le nœud où se refait l'affectation de la même variable possède le même nœud parent que le nœud qui a eu l'échec de l'assignation. S'il y a retour arrière sur toutes les valeurs du domaine de la variable courante, le solveur remonte au nœud parent du nœud de la variable courante pour mettre à jour son assignation. Puis ce processus d'affectation recommence au niveau du nœud parent ;

— l'appel des propagateurs des contraintes du problème pour filtrer les domaines des variables qui n'ont pas encore été assignées. Si après le filtrage des domaines, l'un des domaines devient vide, il y a également un retour arrière du solveur. Nous décrirons en détail le processus de propagation à la suite de ces étapes ;
— lorsqu'il n'y a pas retour arrière, alors l'assignation est réussie et le solveur passe au prochain nœud pour une assignation d'une autre variable. Ce prochain nœud est le fils du nœud où l'assignation vient juste de réussir ;

Lorsqu'une assignation est réussie pour toutes les variables, le solveur la retourne comme solution au problème. Sinon il signale que le modèle de contraintes du problème est non satisfiable. Le moment où chacune de ces actions est réalisée dépend de l'implémentation du solveur.

Remarque 2. Le temps mis pour trouver une solution dépend principalement des deux points suivants :

— le premier point est l'ordre dans lequel s'effectue le choix de la variable à laquelle assigner une valeur et l'ordre dans lequel se fait le choix de la valeur dans le domaine de la variable sélectionnée. Cette politique de choix constitue ce que nous appelons heuristique de recherche. Il existe un grand nombre d'heuristiques dans la littérature [56] ;
— le deuxième point concerne le type des contraintes utilisées pour formaliser le problème. Ceci est dû aux algorithmes de filtrage dédiés qui les accompagnent. En effet, le choix des algorithmes est en fonction de leur temps d'exécution et de la quantité de valeurs qu'ils filtrent des domaines des variables.

C'est ce deuxième point qui nous intéresse. C'est-à-dire le domaine consacré à l'amélioration de l'efficacité des algorithmes de filtrage. Pour caractériser les algorithmes de filtrage, nous définissons des notions telles que le *support d'une valeur* d'une variable, la *cohérence* et la *propagation* des contraintes.

Définition 5. Support d'une valeur d'une variable

Soit C une contrainte et $\mathcal{X}(C) = \{X_1, \cdots, X_{n_C}\}$.

Nous appelons *support de domaine* d'une valeur $x_i \in \mathcal{D}(X_i)$ un n_C-uplet t tel que $\forall i \in \{1, \cdots, n_C\}$, $t[i] \in \mathcal{D}(X_i)$, t satisfait la contrainte C et $t[i] = x_i$.

Nous appelons *support d'intervalle* d'une valeur $x_i \in \mathcal{D}(X_i)$ un n_C-uplet t tel que $\forall i \in \{1, \cdots, n_C\}$, et $\min(\mathcal{D}(X_i)) \leq t[i] \leq \max(\mathcal{D}(X_i))$, t satisfait la contrainte C et $t[i] = x_i$.

Exemple 3. Soit la contrainte $X_1 + X_2 = X_3$ et les domaines $\mathcal{D}(X_1) = \{1, 3\}$, $\mathcal{D}(X_2) = \{2, 5\}$ et $\mathcal{D}(X_3) = \{3, 6\}$.

Alors le tuple $[1, 5, 6]$ est un support de domaine de la valeur 1 de $\mathcal{D}(X_1)$, car chaque valeur du tuple est dans le domaine de sa variable respective et le tuple satisfait la contrainte puisque nous avons $1 + 5 = 6$.

Le tuple $[3, 2, 5]$ est un support d'intervalle de la valeur 2 de X_2, car chaque valeur du tuple est contenue dans le plus petit intervalle qui contient le domaine respectif et le tuple satisfait la contrainte puisque nous avons $3 + 2 = 5$. Cependant, ce tuple n'est pas un support de domaine, car la valeur 5 n'est pas dans le domaine $\mathcal{D}(X_3) = \{3, 6\}$.

Le but d'un algorithme de filtrage est de retirer des valeurs incorrectes des domaines des variables des contraintes. Ces valeurs retirées sont celles pour lesquelles il n'existe pas de support. Il existe plusieurs niveaux de filtrage que nous définissons maintenant.

Définition 6. Cohérence d'une contrainte

Nous disons qu'une contrainte est *cohérente* lorsque l'algorithme de filtrage a terminé de filtrer les domaines de ses variables selon le type de support. C'est ainsi que nous distinguons les niveaux de cohérence suivants :

- une contrainte est *cohérente de domaine* si pour toute variable X de la contrainte et pour toute valeur x dans le domaine de X, il existe un support de domaine pour cette valeur.
- une contrainte est *cohérente d'intervalle* si pour toute variable X de la contrainte et pour toute valeur x dans le domaine de X, il existe un support d'intervalle pour cette valeur.
- une contrainte est *cohérente de borne* si pour toute variable X de la contrainte, il existe un support d'intervalle pour la plus petite valeur et la plus grande valeur du domaine de X.
- lorsque les contraintes d'un problème sont toutes cohérentes soit de domaine, soit d'intervalle ou soit de bornes, on dit que le problème de satisfaction de contraintes est *localement cohérent*.

Exemple 4.

1. Soit la contrainte AllDifferent(X_1, X_2, X_3) qui est satisfaite lorsque les variables X_1, X_2, X_3 prennent des valeurs distinctes. Nous avons les cas suivants :

   - pour les domaines $\mathcal{D}(X_1) = \{1, 3\}, \mathcal{D}(X_2) = \{1, 3\}$ et $\mathcal{D}(X_3) = \{1, 2, 3\}$, cette contrainte n'est pas cohérente de domaine, car la valeur 1 dans le domaine de X_3 n'a pas de support de domaine. Cependant, chaque valeur du domaine de chaque variable a un support d'intervalle, donc la contrainte est cohérente d'intervalles et cohérente de bornes. Par exemple, la valeur 1 de X_3 a le support d'intervalle $[3, 2, 1]$.

— pour les domaines $\mathcal{D}(X_1) = \{2, 3\}$, $\mathcal{D}(X_2) = \{2, 3\}$ et $\mathcal{D}(X_3) = \{1, 2, 3, 4\}$, cette contrainte n'est pas cohérente d'intervalle, car la valeur 2 dans le domaine de X_3 n'a pas de support d'intervalle. Cependant, les deux supports d'intervalle $[2, 3, 4]$ et $[3, 2, 1]$ montrent que la contrainte est cohérente de bornes.

2. les contraintes $X_1 + X_2 = 7$ et $X_1 \cdot X_2 = 12$, avec pour domaines $\mathcal{D}(X_1) = \{3, 4\}$ et $\mathcal{D}(X_2) = \{3, 4\}$, sont localement cohérents de domaines.

Ainsi, nous disons qu'un algorithme de filtrage d'une contrainte *applique* l'une des cohérences lorsque l'algorithme retire des domaines des variables de la contrainte, les valeurs qui n'ont pas un support associé à la cohérence souhaitée, afin d'obtenir ladite cohérence.

Remarque 3. La cohérence de domaine implique la cohérence d'intervalle qui implique la cohérence de borne.

Définition 7. Propagation de contraintes

La *propagation de contraintes* est un processus d'enchaînement des appels des algorithmes de filtrage par le solveur afin d'atteindre la cohérence locale d'un problème de satisfaction de contraintes selon un principe qui évite d'appeler inutilement certains algorithmes de filtrage. Le principe consiste à n'appeler un algorithme de filtrage d'une contrainte que si le premier appel d'un algorithme de filtrage a modifié le domaine d'une variable partagée avec cette contrainte.

Exemple 5. Soit les contraintes $X_1 \cdot X_2 = 12$ et $X_3 = 2 \cdot X_2$, avec pour domaines $\mathcal{D}(X_1) = \{2, 3, 4, 5, 6\}$, $\mathcal{D}(X_2) = \{2, 3, 4, 5, 6\}$ et $\mathcal{D}(X_3) = \{1, 2, 4, 6, 8, 10\}$.

L'appel en premier de l'algorithme de filtrage de la contrainte $X_3 = 2 \cdot X_2$ selon la cohérence de domaine réduit les domaines de X_2 et X_3 à $\mathcal{D}(X_2) = \{2, 3, 4, 5\}$ et $\mathcal{D}(X_3) = \{4, 6, 8, 10\}$. Comme le domaine de X_2 est modifié, cela entraîne l'appel de l'algorithme de filtrage de la contrainte $X_1 \cdot X_2 = 12$ selon la cohérence de domaine qui réduit les domaines à $\mathcal{D}(X_1) = \{3, 4, 6\}$ et $\mathcal{D}(X_2) = \{2, 3, 4\}$. Comme le domaine de X_2 est encore modifié, cela entraîne l'appel à nouveau de l'algorithme de filtrage de la contrainte $X_3 = 2 \cdot X_2$. Ce qui conduit à $\mathcal{D}(X_3) = \{4, 6, 8\}$.

Apprentissage des nogoods

Une autre façon d'améliorer le filtrage est de tenir compte des erreurs qui ont mené aux échecs lors des affectations précédentes des variables afin d'éviter de les répéter lors des prochaines affectations pendant l'exploration de l'arbre de recherche. Ces affectations qui ne peuvent mener à aucune solution sont communément appelées *nogoods*. Ce concept se désigne par l'*apprentissage des nogoods*. C'est-à-dire l'apprentissage des raisons qui ont causé un retour arrière lors de la recherche des solutions. Une fois que ces informations sont apprises, elles sont

utilisées dans la suite de la recherche afin d'éviter des branches de l'arbre de recherche qui conduisent aux mêmes échecs. Pour cela, les solveurs dédiés à ce concept expliquent un échec au moyen d'une nouvelle contrainte qui est ajoutée au modèle de contraintes. Cela permet au solveur d'utiliser l'algorithme de filtrage de cette contrainte pour filtrer les cas qui mènent au même type d'échecs dans la suite de la recherche. Les contraintes qui sont habituellement ajoutées sont des *clauses*. En effet, elles sont de la forme $p_1 \vee p_2 \vee \cdots \vee p_n$ sachant que p_i est la proposition logique $(X_i = v_i)$ ou $(X_i \leq v_i)$ ou $\neg(X_i = v_i)$ ou encore $\neg(X_i \leq v_i)$ où X_i est une variable du problème et v_i est une valeur de son domaine. L'exemple 6 illustre l'acquisition d'une clause.

Exemple 6. Nous reprenons le modèle de l'exemple 2 de Peter Stuckey dans l'article [30]. Le modèle a 5 variables X_1, X_2, X_3, X_4, X_5 de domaines $\mathcal{D}(X_i) = \{1, 2, 3, 4\}\ \forall i \in [1; 5]$ Les contraintes sont :

$$\sum_{i=1}^{5} X_i \leq 12 \tag{1.15}$$

$$\textsc{AllDifferent}(X_1, X_2, X_3, X_4, X_5) \tag{1.16}$$

Lorsque le solveur commence par affecter la valeur 1 à X_1, puis la valeur 2 à X_2, on a les implications ci-dessous menant à un échec.

Lorsque le solveur branche sur $X_1 = 1$ le filtrage de (1.16) donne l'implication

$$X_1 = 1 \Longrightarrow \quad \begin{cases} X_2 \geq 2 \\ X_3 \geq 2 \\ X_4 \geq 2 \\ X_5 \geq 2 \end{cases} \tag{1.17}$$

Lorsque le solveur branche sur $X_2 = 2$ le filtrage de (1.16) et (1.17) donne l'implication

$$X_2 = 2 \wedge \begin{cases} X_3 \geq 2 \\ X_4 \geq 2 \\ X_5 \geq 2 \end{cases} \Longrightarrow \quad \begin{cases} X_3 \geq 3 \\ X_4 \geq 3 \\ X_5 \geq 3 \end{cases} \tag{1.18}$$

Ensuite, le filtrage de (1.15) et (1.18) donne l'implication

$$\begin{cases} X_1 = 1 \\ X_2 = 2 \\ X_3 \geq 3 \\ X_4 \geq 3 \\ X_5 \geq 3 \end{cases} \Longrightarrow \quad \begin{cases} X_3 \leq 3 \\ X_4 \leq 3 \end{cases} \tag{1.19}$$

Ensuite, le filtrage de (1.19) donne l'implication

$$\begin{cases} X_3 \geq 3 \\ X_3 \leq 3 \end{cases} \implies X_3 = 3 \tag{1.20}$$

De même, le filtrage de (1.19) donne l'implication

$$\begin{cases} X_4 \geq 3 \\ X_4 \leq 3 \end{cases} \implies X_4 = 3 \tag{1.21}$$

Enfin, le filtrage de (1.16),(1.20) et (1.21) donne l'implication

$$\begin{cases} X_3 = 3 \\ X_4 = 3 \end{cases} \implies \text{Échec} \tag{1.22}$$

Ces implications peuvent être représentées par le graphe d'implication de la figure 1.1. La ligne en pointillés constitue une coupe qui sépare les branchements du nœud d'échec. Les nœuds qui sont les origines des arcs coupés donnent les conditions suffisantes pour produire un échec.

Nous concluons donc que pour éviter l'échec (1.22) ci-dessus, les variables X_1, X_2, X_3, X_4, X_5 doivent vérifier la contrainte (1.23) :

$$\neg(X_2 = 2) \vee \neg(X_3 \geq 2) \vee \neg(X_4 \geq 2) \vee \neg(X_5 \geq 2) \tag{1.23}$$

$$\iff \neg(X_2 = 2) \vee (X_3 \leq 1) \vee (X_4 \leq 1) \vee (X_5 \leq 1) \tag{1.24}$$

Cette contrainte (1.23) constitue donc une *clause apprise* pendant l'exploration de l'arbre de recherche et peut être ajoutée au modèle pour améliorer la recherche des solutions.

C'est donc ainsi que les solveurs utilisent les algorithmes de filtrage pour réduire l'espace de recherche des solutions d'un problème de satisfaction de contraintes ou d'un problème d'optimisation combinatoire. Nous avons expérimenté notre recherche sur le solveur `clp(FD)` de SICStus Prolog et le solveur `Chuffed` sous MiniZinc. Le solveur `Chuffed` utilise les algorithmes de filtrage et les nogoods tandis que le solveur `clp(FD)` de SICStus Prolog utilise seulement les algorithmes de filtrage. MiniZinc est un langage de modélisation interfacé à différents solveurs, dont SICStus Prolog et Chuffed. Nous présentons maintenant ces deux solveurs.

Le solveur `clp(FD)` de SICStus Prolog

C'est un système basé sur la programmation logique sous contraintes [43] qui est un paradigme efficace dans la résolution des problèmes combinatoires. La programmation logique sous contraintes intervient notamment dans la conception assistée par ordinateur, la recherche opérationnelle, les jeux et l'intelligence artificielle. Sur le plan pratique, ce paradigme est propre au langage Prolog qui possède les atouts suivants :

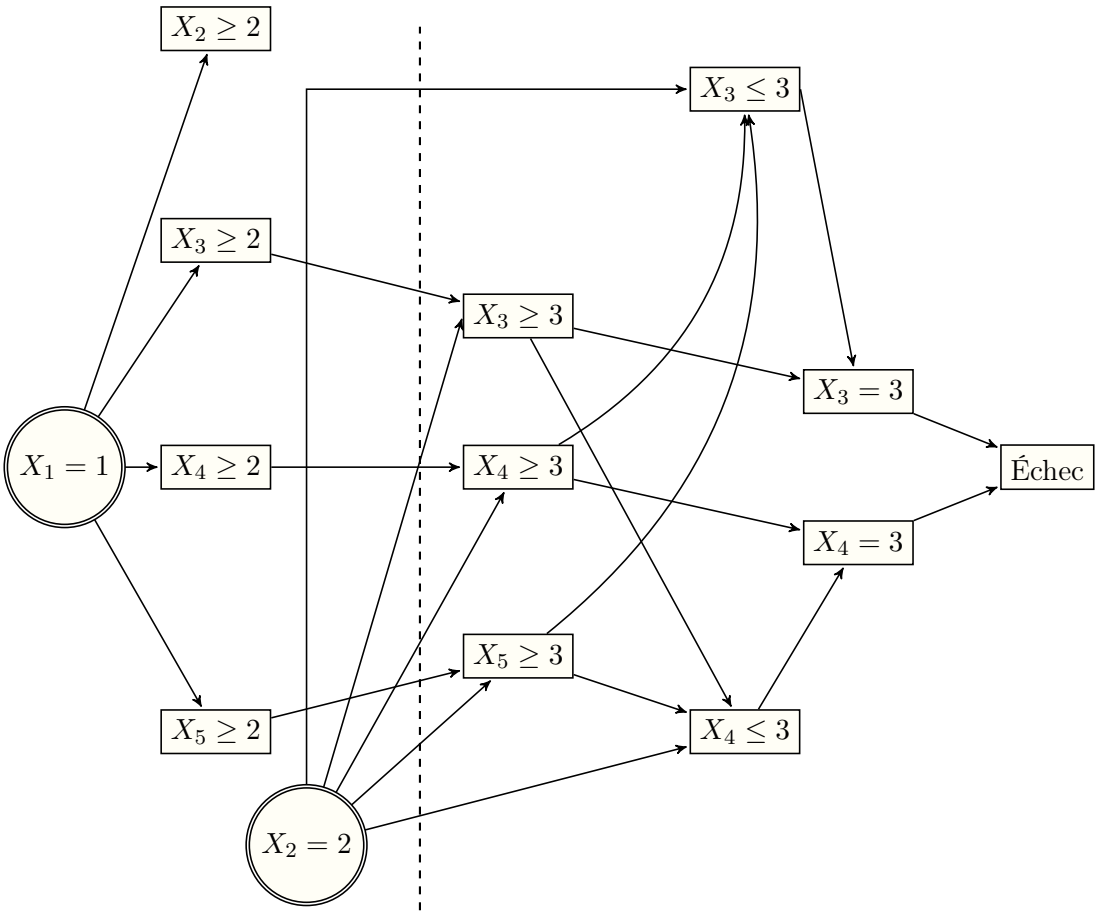


FIGURE 1.1 – Graphe d'implication pour acquisition des nogoods. Chaque sous-ensemble d'arcs entrants d'un nœud représente une des implications (1.17), (1.18),(1.19), (1.20),(1.21) et (1.22). Les branchements du solveur sont représentés par les nœuds circulaires. Les nœuds rectangulaires sont les nouveaux domaines résultant du filtrage des contraintes (1.16) et (1.15). La ligne en pointillés sépare les branchements $X_1 = 1$ et $X_2 = 2$ du nœud d'échec. Les nœuds à l'origine des arcs coupés constituent les conditions suffisantes qui conduisent à l'échec.

— il permet de programmer de manière concise, claire et avec une grande expressivité ;
— il s'interface simplement avec d'autres langages de programmation ;
— les propagateurs de contraintes rendent rapide l'exécution du programme ;
— le système de retour arrière de Prolog permet l'énumération directe de toutes les solutions d'une même instance du problème.

La figure 1.2 présente un exemple de code de SICStus Prolog pour résoudre le problème de l'exemple 2.

```
1  :- use_module(library(lists)).
2  :- use_module(library(clpfd)).

3  % Définition d'un prédicat à utiliser dans le prédicat principal "bacp":
4  vecteur_booleen_qui_associe_une_session_aux_cours(_,[],[]):-!.
5  vecteur_booleen_qui_associe_une_session_aux_cours(J,[X|R],[B|S]):-
6       B #<=> (X #= J), % B = 1 <=> Cours X dispensé à la session J
7       vecteur_booleen_qui_associe_une_session_aux_cours(J,R,S).

8  bacp:-
9       % Déclaration des variables:
10      length(Sessions,7), % Vecteur "Sessions" de 7 variables de sessions
11      domain(Sessions,1,3), % Les 7 variables prennent les valeurs 1,2 ou 3
12      Sessions = [X1,_,X3,X4,X5,_,_], % Variables des cours prérequis
13      vecteur_booleen_qui_associe_une_session_aux_cours(1,Sessions,Bools1),
14      vecteur_booleen_qui_associe_une_session_aux_cours(2,Sessions,Bools2),
15      vecteur_booleen_qui_associe_une_session_aux_cours(3,Sessions,Bools3),
16      Credits = [3,3,4,2,2,2,2], % Crédits par cours
17      scalar_product(Credits, Bools1, #=, O1), % Crédits O1 de la session 1
18      scalar_product(Credits, Bools2, #=, O2), % Crédits O2 de la session 2
19      scalar_product(Credits, Bools3, #=, O3), % Crédits O3 de la session 3

20      % Contraintes:
21      O1 #=< 7, O2 #=< 7, O3 #=< 7, % Crédits par session inférieure à 8
22      X1 #< X3, % Français suivi avant mathématiques
23      X3 #< X5, % Mathématiques suivies avant chimie
24      X4 #< X5, % Anglais suivi avant chimie
25      maximum(Obj,[O1,O2,O3]), % Calcul de la fonction Objectif Obj

26      % Recherche de la solution:
27      labeling([minimize(Obj)],Sessions), % Énumération de la solution
28      write(solution(Sessions,Obj,O1,O2,O3)). % Affichage de la solution
```

FIGURE 1.2 – Code SICStus Prolog encodant l'instance de la répartition équilibrée de cours sur un ensemble donné de sessions de l'exemple 2. Les lignes 1 et 2 permettent d'utiliser respectivement des listes et des contraintes. Les lignes 4 à 7 définissent un prédicat qui encode l'expression (1.4). Puis ce prédicat est ensuite utilisé aux lignes 13 à 15 après que les variables $X_i \ \forall i \in [1;7]$ du (1.4) soient déclarées aux lignes 10 à 12. Cela permet de calculer aux lignes 17 à 19 les crédits $O_j \ \forall j \in [1;3]$ de chacune des sessions j. Puis s'en suivent la pose des contraintes aux lignes 21 à 25 et la recherche de la solution à la ligne 27.

Le solveur `Chuffed` sous MiniZinc

C'est un système basé sur l'apprentissage des nogoods [57]. Ce système combine à la fois la technologie des solveurs à domaines finis comme le solveur `clp(FD)` et celle des solveurs dédiés à la résolution des problèmes de satisfiabilité booléenne. Cela lui donne de gros avantages dans la résolution des problèmes combinatoires. Pour l'utiliser, il peut être directement appelé

depuis le langage de modélisation MiniZinc. La figure 1.3 présente un exemple de code de MiniZinc pour résoudre le problème de l'exemple 2.

```
1  include "globals.mzn";

2  % Déclaration des variables
3  array [1..7] of var 1..3: Sessions;% Vecteur de 7 variables de sessions
4  array [1..3] of var 0..18: CredSession; % Vecteur des variables Oj
5  var 0..18: Obj; % Fonction objectif

6 % Contraintes :
7 array [1..7] of int: Credits = [3,3,4,2,2,2,2]; % Credits par cours
8 constraint forall(s in 1..3) (
    CredSession[s] =
    sum(c in 1..7) (bool2int(Sessions[c] =
    s) * Credits[c])); % Crédits Oj par session i
9 constraint forall(s in 1..3) (
    CredSession[s] <= 7); % Credits par session inférieure à 8
10 constraint Sessions[1] < Sessions[3]; % Français suivi avant Mathématiques
11 constraint Sessions[3] < Sessions[5]; % Mathématiques suivi avant chimie
12 constraint Sessions[4] < Sessions[5]; % Anglais suivi avant Chimie
13 constraint maximum(Obj, CredSession); % Calcul de la fonction objectif Obj

14 % Recherche de la solution:
15 solve minimize Obj;

16 % Affichage de la solution
17 output
    [show(Sessions)] ++ [","]++[show(Obj)]++[","]++[show(CredSession)];
```

FIGURE 1.3 – Code MiniZinc qui encode l'instance de la répartition équilibrée de cours sur un ensemble donné de sessions de l'exemple 2. La ligne 1 permet l'utilisation des contraintes. Les lignes 3 à 5 déclarent respectivement les domaines des variables $X_i, O_j \ \forall i \in [1;7], \ \forall j \in [1;3]$ et la fonction objectif Obj. La ligne 7 qui donne les crédits pour chaque cours est utilisée pour encoder l'expression (1.4) à la ligne 8. Puis s'en suit la pose des contraintes aux lignes 9 à 13, la recherche de la solution à la ligne 15 et son affichage à la ligne 17.

Nous montrons comment rendre plus efficaces de tels solveurs dans notre travail. Dans la littérature, il existe quelques méthodes qui, pour améliorer un solveur, font du prétraitement du modèle qui est fourni au solveur. Ces méthodes sont implémentées sous la forme de systèmes que l'on appelle présolveurs. Nous présentons par la suite les présolveurs.

Les présolveurs

Un présolveur est un programme exécuté avant le solveur qui simplifie ou enrichit un modèle pour permettre au solveur de trouver plus facilement la solution. Les présolveurs pour

les programmes d'entiers mixtes généralement fixent un sous ensemble de variables en les affectant à des valeurs, réduisent le nombre de contraintes, renforcent les bornes, suppriment les contraintes redondantes et resserrent les inégalités [2]. Les solveurs de problèmes de satisfaction booléenne utilisent souvent des techniques telles que la propagation unitaire, les littéraux échoués, les littéraux purs et la subsomption [17]. Le prétraitement d'un problème de satisfaction de contraintes est parfois plus difficile en raison de la richesse du langage et de l'hétérogénéité de ses contraintes. Bien qu'il soit plus difficile d'apporter des améliorations globales au modèle, des techniques telles que l'élimination de sous-expressions permettent d'augmenter le niveau de filtrage dans un modèle [66].

Le prétraitement ne signifie pas nécessairement l'amélioration du modèle pour une instance donnée du problème. Cela peut également signifier améliorer le modèle lui-même pour chaque instance, voire renforcer les méthodes de filtrage. Par exemple, Ekaterina *et al.* [5] proposent d'analyser les modèles de séries chronologiques pour déduire des bornes utilisées pour renforcer le filtrage. L'ajout de contraintes redondantes au modèle entraîne davantage de filtrage et réduit donc la taille de l'espace de recherche. Andrea *et al.* [34] prennent une contrainte en entrée et synthétisent un algorithme de filtrage. Leur méthode fonctionne pour de petites contraintes, car le prétraitement hors ligne prend un temps exponentiel, mais l'algorithme de filtrage s'exécute en temps polynomial pendant la recherche.

Notons enfin que dans le contexte de problèmes d'algèbre finie, Charnley *et al.* [29] proposent un système de génération de contraintes implicites simples, parmi lesquelles sont sélectionnées les plus intéressantes pour le contexte.

Pour améliorer davantage l'efficacité des solveurs, nous nous basons sur l'observation suivante : pour résoudre un problème combinatoire, l'informaticien peut généralement générer un modèle à contraintes et les algorithmes de filtrage dédiés qui caractérisent le problème. Puis, ces éléments sont fournis au solveur qui les exploite pour élaguer l'espace de recherche. Cependant, le modèle peut s'avérer incomplet et mauvais, car certaines contraintes, du moins les plus importantes ont échappé à l'attention humaine lors de la conception du modèle à contraintes. Pour répondre à cette insuffisance, nous faisons de l'apprentissage automatique d'un grand nombre de contraintes générales qui caractérisent les objets combinatoires intervenant généralement dans la modélisation de nombreux problèmes combinatoires. Ces contraintes sont apprises sous forme de conjectures à partir d'un nombre fini d'instances de l'objet combinatoire considéré. Puis, une fois les plus significatives de ces conjectures prouvées, elles sont ajoutées au modèle à contraintes de base afin de le rendre plus efficace. Avant d'illustrer en détail notre approche et les contributions de nos travaux, nous présentons quelques systèmes d'acquisition automatique de conjectures de l'état de l'art.

1.2 La génération automatique de conjectures

Dans le cadre de nos travaux, nous appelons *conjecture* toute formule mathématique qui décrit une propriété que l'on croit vérifiée par un objet combinatoire sans l'avoir encore prouvée. Nous entendons par *objet combinatoire* toute structure mathématique construite à partir d'un nombre fini d'éléments. Les partitions d'ensembles finis, les graphes orientés ou encore les forêts enracinées à sommets finis sont des exemples d'objets combinatoires que nous étudions. Il a été démontré que dans tout graphe orienté, le nombre d'arcs a du graphe est borné supérieurement par le carré du nombre de sommets v. C'est-à-dire que $a \leq v^2$. Cette formule est ce que nous appelons conjecture sur les graphes lorsqu'elle n'est pas encore prouvée. Une fois qu'elle est prouvée, elle devient ce que nous appelons *invariant* et prend le statut de théorème sur les graphes orientés. De même, la somme des carrés S des tailles des partitions d'un ensemble est bornée supérieurement par le carré du cardinal n de l'ensemble. C'est-à-dire $S \leq n^2$ est un invariant dans les partitions d'ensembles. Ces invariants s'expriment à l'aide de métriques que sont le nombre de sommets v et d'arcs a d'un graphe orienté, la somme des carrés S des tailles des partitions et le cardinal n d'un ensemble. De telles métriques à valeurs entières qui caractérisent une instance de l'objet combinatoire considéré constituent ce que nous appelons *caractéristiques* d'un objet combinatoire. Découvrir de nouvelles conjectures existantes entre caractéristiques d'objets combinatoires est l'une des préoccupations majeures de nos travaux. Ainsi, avant de présenter ce qui a été déjà fait dans le domaine de la découverte automatique de conjectures, nous présentons d'abord les objets combinatoires que nous étudions.

1.2.1 Objets combinatoires étudiés

Les objets combinatoires interviennent beaucoup dans la modélisation des problèmes combinatoires. Par exemple, le filtrage de la contrainte AllDifferent$(X_1, \ldots, X_n)$, le problème du commis voyageur, l'équilibrage des charges dans les systèmes de communication utilisent les graphes. Les structures de données telles que les arbres sont très utilisées pour améliorer l'efficacité des algorithmes. Les propriétés des partitions d'ensembles sont très sollicitées dans les problèmes de répartition. Les séquences de variables qui prennent des valeurs pour lesquelles le mot obtenu appartient à un langage régulier en théorie des automates sont des structures que l'on rencontre dans les problèmes d'emploi du temps. Pouvoir découvrir de nouvelles propriétés de ces objets combinatoires pourrait améliorer la résolution de ces problèmes combinatoires. De telles propriétés s'expriment le plus souvent à l'aide de relations entre les caractéristiques de l'objet combinatoire. C'est ainsi que nous présentons les caractéristiques des objets combinatoires étudiés dans nos travaux.

A– Graphes orientés n'ayant pas de sommet isolé

Généralement, un graphe peut avoir des sommets isolés, c'est-à-dire des sommets n'ayant aucun arc entrant ou sortant. Ici, nous nous intéressons uniquement aux graphes orientés pour

lesquels tout sommet possède toujours au moins un arc entrant ou sortant. En effet, cela nous permet de comparer nos résultats à ceux du catalogue de contraintes globales [10] qui impose cette restriction.

Définition 8 (Graphe orienté)**.** Un *graphe orienté* $\mathcal{G}(V, E)$ est un ensemble de *sommets* V et un ensemble E de *couples de sommets* tels que pour chaque sommet $i \in V$, il existe un sommet $j \in V$ tel que $(i, j) \in E$ ou $(j, i) \in E$. Tout couple de E est appelé *arc* du graphe orienté. En particulier, lorsque $i = j$ l'arc (i, i) est appelé *boucle* au sommet i. Et lorsque le sommet i n'a que cette boucle, il est appelé *sommet isolé*.

Définition 9 (Chemin dans graphe orienté)**.** Pour un graphe orienté, un *chemin* du sommet i vers le sommet j est une suite de n sommets $i_1, i_2, \cdots, i_n$ tels que $\forall k \in [1 : n-1], i_k \neq i_{k+1}, (i_k, i_{k+1}) \in E$ avec $i = i_1$ et $i_n = j$, où n est un entier supérieur à 1.

Définition 10 (Graphe non orienté)**.** Un *graphe non orienté* $\mathcal{G}(V, E)$ est un ensemble de *sommets* V et un ensemble E de *paires de sommets* de V. Toute paire $\{i, j\} \in E$ de sommets est appelée *arête* du graphe non orienté.

Définition 11 (Chaîne dans un graphe non orienté)**.** Pour un graphe non orienté, une *chaîne* qui relie un sommet i à un sommet j est une suite de n sommets $i_1, i_2, \cdots, i_n$ tels que $\forall k \in [1 : n-1], i_k \neq i_{k+1}, \{i_k, i_{k+1}\} \in E$ avec $i_1 = i$ et $i_n = j$, où n est un entier naturel supérieur à 1. En particulier lorsque $i_1 = i_n$, la chaîne est appelée *cycle*.

Dans un graphe orienté, nous distinguons deux structures qui sont les *composantes connexes* et les *composantes fortement connexes*.

Définition 12 (Composante connexe)**.** Une *composante connexe* d'un graphe non orienté est soit un *sommet isolé*, soit un *sous-ensemble maximal* $\overline{V}$ de sommets du graphe orienté tels que pour toute paire $\{i, j\}$ de sommets de $\overline{V}$, il existe une chaîne de i vers j.

Définition 13 (Composante fortement connexe)**.** Une *composante fortement connexe* d'un graphe orienté est soit un *sommet isolé*, soit un *sous-ensemble maximal* $\overline{V}$ de sommets du graphe orienté tels que pour toute paire $\{i, j\}$ de sommets de $\overline{V}$, il existe un chemin de i vers j et un chemin de j vers i.

Définition 14. Nous appelons *taille d'un graphe* ou *d'une composante connexe* ou encore *d'une composante fortement connexe*, le nombre de sommets du graphe, de la composante connexe ou de la composante fortement connexe.

Définition 15 (Composante connexe intermédiaire)**.** Une *composante connexe intermédiare* d'un graphe orienté est une composante connexe dont le nombre de sommets est strictement compris entre la taille de la plus grande composante connexe et la taille de la plus petite composante connexe.

Quelques caractéristiques d'un graphe orienté

Dans un graphe orienté, dont une instance est illustrée à la figure 1.4, nous nous intéressons aux caractéristiques suivantes :

1. v : le nombre de sommets ;
2. a : le nombre d'arcs du graphe orienté ;
3. c : le nombre de composantes connexes ;
4. s : le nombre de composantes fortement connexes ;
5. $\underline{c}$: la taille de la plus petite composante connexe ;
6. $\overline{c}$: la taille de la plus grande composante connexe ;
7. $\underline{s}$: la taille de la plus petite composante fortement connexe ;
8. $\overline{s}$: la taille de la plus grande composante fortement connexe ;
9. $\underline{oc}$: le nombre de composantes connexes de taille minimale ;
10. $\overline{oc}$: le nombre de composantes connexes de taille maximale ;
11. $\underline{os}$: le nombre de composantes fortement connexes de taille minimale ;
12. $\overline{os}$: le nombre de composantes fortement connexes de taille maximale ;
13. $c_{>1}$: le nombre de composantes connexes de taille supérieure à 1 sommet ;
14. $s_{>1}$: le nombre de composantes fortement connexes de taille supérieure à 1 sommet ;
15. r : le nombre de sommets qui reste à répartir entre les composantes connexes lorsque ces dernières ont déjà toutes le nombre minimal de sommets d'une composante connexe. C'est-à-dire que $r = v - c \cdot \underline{c}$.
16. $minmaxc$: la taille $\underline{c}$ de la plus petite composante connexe lorsqu'il existe une plus grande composante connexe de taille distincte de celle de la plus petite. Si au contraire toutes les composantes connexes sont de taille identique, alors $minmaxc = 0$.
17. $midc$: la taille minimale d'une composante connexe intermédiaire. Cette caractéristique vaut 0 lorsqu'il n'existe pas de composantes connexes intermédiaires ;
18. $omidc$: le nombre de composantes connexes intermédiaires entre celles de taille minimale et celles de taille maximale ;
19. $c_{\in\{2,3\}}$: le nombre de composantes connexes ayant deux ou trois sommets, tel que chacune de leurs composantes fortement connexes soit constituée d'un seul sommet.

B– Les arbres enracinés

Définition 16 (Graphe non orienté connexe)**.** Un graphe non orienté $\mathcal{G}(V, E)$ est dit *connexe* lorsque pour toute paire $\{i, j\}$ de sommets de V, il existe une chaîne $i_1, i_2, \cdots, i_n$, avec $i = i_1$ et $i_n = j$.

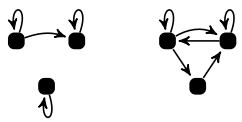


FIGURE 1.4 – Illustration d'une instance de graphe orienté avec les valeurs de ses caractéristiques $v = 6, a = 10, c = 3, s = 4, \underline{c} = 1, \overline{c} = 3, \underline{s} = 1, \overline{s} = 3, \underline{oc} = 1, \overline{oc} = 1, \underline{os} = 1, \overline{os} = 1, c_{>1} = 2, s_{>1} = 1, r = 3, minmaxc = 1, midc = 2, omidc = 1, c_{\in\{2,3\}} = 1$.

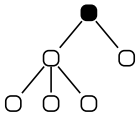


FIGURE 1.5 – Illustration d'une instance d'arbre enraciné avec les valeurs de ses caractéristiques sachant que la racine est le sommet coloré en noir : $v = 6, f = 4, \underline{d} = 2, \overline{d} = 3, \underline{p} = 1, \overline{p} = 2$.

Définition 17 (Arbre enraciné)**.** Un *arbre enraciné* est un graphe non orienté $\mathcal{G}(V, E)$ qui est connexe, sans cycle et est tel qu'un des sommets a été arbitrairement désigné comme sommet particulier et porte le nom de *racine*.

Définition 18 (Nœud fils d'un nœud père dans un arbre enraciné)**.** Soit i_r la racine d'un arbre enraciné. Un sommet i de l'arbre est appelé *nœud père* et un sommet j de l'arbre est appelé *nœud fils* du *nœud père* j s'il existe dans l'arbre une chaîne $i_1, i_2, \cdots, i_{n-1}, i_n$ qui relie la racine au nœud $j = i_n$ avec $i_{n-1} = i$ et $j \neq i_r$. Une *feuille* dans un arbre enraciné est un sommet qui n'est pas un nœud père.

Définition 19 (Degré et profondeur d'un sommet dans un arbre enraciné)**.** Le *degré d'un sommet* est le nombre de nœud fils qu'il possède. La *profondeur d'un sommet* est le nombre d'arêtes de la chaîne reliant la racine à ce sommet.

Quelques caractéristiques d'un arbre enraciné

Dans un arbre enraciné, dont une instance est illustrée à la figure 1.5, nous nous intéressons aux caractéristiques suivantes :

1. v : le nombre de sommets de l'arbre ;
2. f : le nombre de feuilles de l'arbre ;
3. $\underline{d}$: le degré minimal d'un sommet qui n'est pas une feuille de l'arbre ;
4. $\overline{d}$: le degré maximal d'un sommet qui n'est pas une feuille de l'arbre ;
5. $\underline{p}$: la profondeur minimale d'une feuille de l'arbre ;
6. $\overline{p}$: la profondeur maximale d'une feuille de l'arbre ;

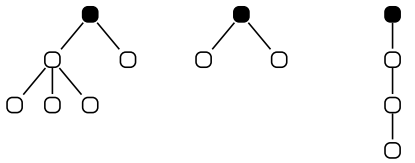


FIGURE 1.6 – Illustration d'une instance de forêt enracinée avec les valeurs de ses caractéristiques sachant que les racines sont les sommets colorés en noir : $v = 13, t = 3, f = 7, \underline{f} = 1, \overline{f} = 4, \underline{t} = 3, \overline{t} = 6, \underline{d} = 1, \overline{d} = 3, \underline{p} = 1, \overline{p} = 3$.

C– Les forêts enracinées

Définition 20 (Forêt enracinée)**.** Une *forêt enracinée* est une collection d'arbres enracinés.

Nous appelons *taille d'un arbre* le nombre de sommets de l'arbre.

Quelques caractéristiques d'une forêt enracinée

Dans une forêt enracinée, dont une instance est illustrée à la figure 1.6, nous nous intéressons aux caractéristiques suivantes :

1. v : le nombre de sommets de la forêt ;
2. t : le nombre d'arbres de la forêt ;
3. f : le nombre de feuilles de la forêt ;
4. $\underline{d}$: le degré minimal d'un sommet qui n'est pas une feuille de la forêt ;
5. $\overline{d}$: le degré maximal d'un sommet qui n'est pas une feuille de la forêt ;
6. $\underline{p}$: la profondeur minimale d'une feuille de la forêt ;
7. $\overline{p}$: la profondeur maximale d'une feuille de la forêt ;
8. $\underline{f}$: le nombre minimal de feuilles d'un arbre de la forêt ;
9. $\overline{f}$: le nombre maximal de feuilles d'un arbre de la forêt.
10. $\underline{t}$: la taille minimale d'un arbre de la forêt ;
11. $\overline{t}$: la taille maximale d'un arbre de la forêt.

D– Les forêts enracinées sans sommets isolés

Définition 21 (Forêt enracinée sans sommets isolés)**.** Une *forêt enracinée sans sommets isolés* est une collection d'arbres enracinés ayant chacun au moins 2 sommets.

Les caractéristiques auxquelles nous nous intéressons pour les forêts enracinées sans sommets isolés sont les mêmes que celles des forêts enracinées.

E– Les séquences de 0/1

Définition 22 (Séquence de 0/1)**.** Une séquence de 0/1 est une séquence de 0 et de 1. Dans une telle séquence, nous appelons *groupe* toute sous-séquence de 1 *maximale au sens de l'inclusion.*

Nous appelons *interdistance* dans une séquence de 0/1, une suite de 0 séparant deux groupes consécutifs.

Nous appelons *longueur d'un groupe* (resp. d'une *interdistance*), le nombre de valeurs 1 (resp. de valeurs 0) que contient le groupe (resp. l'interdistance).

Nous appelons *groupe intermédiaire* tout groupe dont la longueur est strictement comprise entre la longueur du groupe le plus court et du groupe le plus long.

Nous appelons *interdistance intermédiaire* toute interdistance dont la longueur est strictement comprise entre la longueur de l'interdistance la plus courte et de l'interdistance la plus longue.

Quelques caractéristiques d'une séquence de 0/1

Dans une séquence de 0/1, dont une instance est illustrée à l'exemple 7 nous nous intéressons aux caractéristiques suivantes :

1. n : le nombre total de 0 ou 1 de la séquence ;
2. n_g : le nombre de groupes de la séquence ;
3. n_v : le nombre de 1 dans la séquence ;
4. s_{min} : la longueur du groupe le plus court ;
5. s_{max} : la longueur du groupe le plus long ;
6. s_{range} : la différence de longueurs entre le groupe le plus long et le groupe le plus court ;
7. s_{sq} : la somme des carrés des longueurs des groupes de la séquence ;
8. d_{min} : la longueur minimale d'une interdistance ;
9. d_{max} : la longueur maximale d'une interdistance ;
10. d_{range} : la différence entre d_{max} et d_{min} ;
11. sd_{sq} : somme des carrés des longueurs des interdistances ;
12. s_{minmax} : la longueur minimale d'un groupe lorsque les groupes n'ont pas la même longueur, ou 0 lorsque les groupes ont tous la même longueur.
13. s_{mid} : la longueur du plus court groupe intermédiaire, ou 0 s'il n' existe pas de groupe intermédiaire ;
14. rs_{min} : le nombre de 1 restants lorsqu'on impose que tous les groupes de la séquence aient la longueur minimale. C'est-à-dire que $rs_{min} = n_v - n_g \cdot s_{min}$;
15. os_{min} : le nombre des groupes les plus courts ;
16. os_{max} : le nombre des groupes les plus longs ;

17. os_{mid} : le nombre de groupes intermédiaires ;
18. $os_{>1}$: le nombre de groupes qui ont plus d'un seul élément ;
19. d_{minmax} : la plus petite interdistance lorsque les interdistances n'ont pas la même longueur, ou 0 lorsque les interdistances ont la même longueur.
20. d_{mid} : la longueur de la plus courte interdistance intermédiaire, ou 0 lorsqu'il n'y a pas d'interdistances intermédiaires ;
21. rd_{min} : le nombre de 0 restants lorsqu'on impose que toutes les interdistances de la séquence aient la longueur minimale. C'est-à-dire que $rd_{min} = (n - n_v) - (n_g - 1) \cdot d_{min}$;
22. od_{min} le nombre d'interdistances de longueur d_{min} ;
23. od_{max} le nombre d'interdistances de longueur d_{max} ;
24. od_{mid} le nombre d'interdistances de longueur d_{mid} ;
25. $od_{>1}$ le nombre d'interdistances de longueur supérieure à 1.

Exemple 7. Soit la séquence de 0/1 (1.25) suivante :

$$0000111100110000111111011000000 11 \tag{1.25}$$

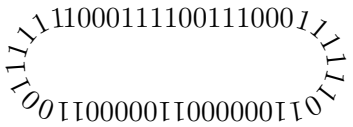
Cette séquence a les valeurs suivantes pour ces caractéristiques : $n = 33, n_g = 5, n_v = 16, s_{min} = 2, s_{max} = 6, s_{range} = 4, s_{sq} = 64, d_{min} = 1, d_{max} = 6, d_{range} = 5, sd_{sq} = 57, s_{minmax} = 2, s_{mid} = 4, rs_{min} = 6, os_{min} = 3, os_{max} = 1, os_{mid} = 1$.

F– Les séquences cycliques de 0/1

Définition 23 (Séquence cyclique de 0/1)**.** Une séquence cyclique de 0/1 est une séquence de 1 ou 0. Dans une telle séquence, une sous-séquence de 1 maximale au sens de l'inclusion est appelée *groupe*. Et si la séquence commence par 1 et se termine par 1, alors ces deux valeurs appartiennent à un seul et unique *groupe*.

Les caractéristiques auxquelles nous nous intéressons pour les séquences cycliques de 0/1 sont les mêmes que celles des séquences de 0/1. L'exemple 8 illustre une instance de séquence cyclique de 0/1.

Exemple 8. Soit la séquence cyclique de 0/1 (1.26) suivante :



(1.26)

Cette séquence a les valeurs suivantes pour ces caractéristiques : $n = 48, n_g = 7, n_v = 26, s_{min} = 2, s_{max} = 7, s_{range} = 5, s_{sq} = 69, d_{min} = 1, d_{max} = 6, d_{range} = 5, sd_{sq} = 66, s_{minmax} = 2, s_{mid} = 3, rs_{min} = 11, os_{min} = 3, os_{max} = 1, os_{mid} = 1$.

G– Les partitions d'ensembles

Définition 24 (Partition d'ensembles)**.** Une *partition d'un ensemble fini* $\mathcal{S}$ est une collection de *sous-ensembles* non vides et disjoints de $\mathcal{S}$ tels que l'union de ces sous-ensembles est égale à $\mathcal{S}$. Ces sous-ensembles sont aussi appelés *parties* de $\mathcal{S}$.

Nous appelons *taille d'une partie* le nombre d'éléments de la partie dans une partition.

Nous appelons *partie intermédiaire* toute partie dont la taille est strictement comprise entre la taille de la plus petite partie et la taille de la plus grande partie.

Quelques caractéristiques d'une partition d'un ensemble

Dans une partition d'un ensemble $\mathcal{S}$, dont une instance est illustrée à l'exemple 9, nous nous intéressons aux caractéristiques suivantes :

1. n : le cardinal de l'ensemble $\mathcal{S}$;
2. P : le nombre de parties de $\mathcal{S}$, c'est-à-dire le nombre de sous-ensembles en lequel est partitionné $\mathcal{S}$;
3. $\underline{M}$: la taille de la plus petite partie ;
4. $\overline{M}$: la taille de la plus grande partie ;
5. $\underline{\overline{M}}$: la différence opposée de tailles entre la plus grande et la plus petite partie ;
6. S : la somme des carrés des tailles des parties ;
7. M_{minmax} : la taille de la plus petite partition si les parties sont de tailles distinctes, 0 sinon ;
8. M_{mid} : la taille de la plus petite partition intermédiaire, 0 s'il n'existe pas de partie intermédiaire ;
9. O_{min} : le nombre de parties de taille $\underline{M}$;
10. O_{max} : le nombre de parties de taille $\overline{M}$;
11. O_{mid} : le nombre de parties de taille M_{mid} ;
12. $O_{>1}$: le nombre de parties de taille supérieure à 1 ;
13. $O_{=1}$: le nombre de parties de taille 1.

Exemple 9. La représentation (1.27) illustre une instance de partition d'un ensemble de 12 éléments (les 12 premiers entiers non nuls) en 4 sous-ensembles disjoints numérotés de 1 à 4. Cette partition a une représentation équivalente au vecteur $V = [1, 1, 1, 2, 2, 3, 3, 4, 4, 4, 4, 4]$. Dans ce vecteur, la valeur de la i-ème composante indique le numéro de la partie (sous-ensemble) à laquelle appartient l'élément i. Par exemple, $V[3] = 1$ signifie que l'élément 3 appartient à la première partie qui est $\{1, 2, 3\}$. Ainsi, les valeurs des caractéristiques sont les suivantes : $n = 12, P = 4, \underline{M} = 2, \overline{M} = 5, \underline{\overline{M}} = 3, S = 42, M_{minmax} = 2, M_{mid} = 3, O_{min} = 2, O_{max} = 1, O_{mid} = 1, O_{>1} = 4, O_{=1} = 0$.

$$\{1,2,3\},\{4,5\},\{6,7\},\{8,9,10,11,12\} \Longleftrightarrow [1,1,1,2,2,3,3,4,4,4,4,4] \tag{1.27}$$

H– Les partitions d'ensembles avec ensembles vides

Imaginons que nous voulions ranger n éléments dans P_{max} tiroirs et que $n < P_{max}$. Alors il y aura forcément un tiroir vide ! Ce tiroir constitue ce que nous appelons *partie vide*. Pour prendre en compte ce type de partie, nous introduisons les partitions d'ensembles avec ensembles vides.

Définition 25 (Partition d'ensembles avec ensembles vides)**.** Une partition d'un ensemble $\mathcal{S}$ avec ensembles vides est une collection de sous-ensembles disjoints de $\mathcal{S}$ qui peuvent être éventuellement vides et telle que l'union de ces sous-ensembles est égal à $\mathcal{S}$.

Les caractéristiques auxquelles nous nous intéressons pour les partitions d'ensembles avec ensembles vides sont les mêmes caractéristiques que celles des partitions d'ensembles en plus de la caractéristique P_{max} qui désigne le nombre de partitions ou sous-ensembles que l'on considère comme pouvant être éventuellement vides.

Les caractéristiques des objets combinatoires que nous venons de présenter sont utilisées dans nos travaux pour exprimer des conjectures sur les propriétés des objets combinatoires considérés. Du point de vue de la programmation par contraintes, ces conjectures constituent des contraintes probablement vérifiées par les objets combinatoires considérés. Bon nombre de systèmes ont été conçus pour la recherche automatique de conjectures. Nous présentons maintenant les plus marquants.

1.2.2 Les méthodes existantes de génération des invariants

Le premier système construit pour la recherche automatique de conjectures mathématiques est *Program II* fait par Hao Wang [71] dans les années 1950. Ce système génère des conjectures en logique des prédicats. Cependant, le programme est incapable de sélectionner les conjectures les plus significatives parmi des milliers de conjectures générées. Puis d'autres programmes ont fait leur apparition avec le souci de répondre à ce problème. Nous les classons en deux types de systèmes : ceux dont l'objectif est la découverte automatique de conjectures pouvant aboutir à des théorèmes et ceux qui font de l'apprentissage statistique de modèles approximatifs qui ne collent pas nécessairement à toutes les données d'apprentissage, mais peuvent cependant conduire à de bonnes prédictions. Les méthodes de ce deuxième type de système sont appelées régressions symboliques. Nous commençons par présenter ces dernières.

Définition 26. La régression symbolique

Soient n un entier non nul et $D = \{(x_i, y_i) : i \in [1 : n]\}$ un ensemble fini. Faire de la *régression symbolique*, c'est réaliser les étapes suivantes :

— séparer l'ensemble D en deux ensembles disjoints D_a et D_t : $D = D_a \cup D_t$ et $D_a \cap D_t = \emptyset$. D est appelé *données du problème* de régression symbolique. Les ensembles D_a et D_t sont respectivement appelés *données d'apprentissage* et *données de test*. Pour déterminer D_a et D_t, plusieurs méthodes d'échantillonnage proposées par Lohr Sharon [54] s'y prêtent ;

— trouver une fonction f minimisant l'écart entre $f(x_i)$ et y_i pour tous les (x_i, y_i) de D_a. C'est-à-dire que f est une solution au problème (1.28).

$$\underset{f}{\text{Minimiser}} \sum_{(x_i,y_i)\in D_a} \|f(x_i) - y_i\|^2 \tag{1.28}$$

— une fois que f est déterminée, vérifier que $\sum_{(x_i,y_i)\in D_t} \|f(x_i) - y_i\|^2$ est inférieur à un certain seuil fixé. Si c'est le cas, alors f est considéré comme *expliquant au mieux* D.

Comme l'explique l'article [50], la détermination de f se fait généralement par l'*analyse fonctionnelle* ou la *programmation génétique*.

Dans ce qui suit, nous présentons deux méthodes pour déterminer une solution f du problème de régression symbolique (1.28). L'une utilise l'analyse fonctionnelle tandis que l'autre utilise la programmation génétique.

Régression symbolique exploitant la modularité du graphe de représentation Cette méthode est présentée par Samaneh *et al.* [70], puis par Udrescu *et al.* [55]. Elle consiste à déterminer l'expression symbolique f qui explique le mieux les données en supposant que la représentation de f sous forme d'arbre est modulaire comme le montre la figure 1.7 dans le cas où $f(x, y) = \dfrac{1}{\sqrt{1 + (x - \frac{1}{y})^2}}$. C'est-à-dire qu'elle peut être décomposée en sous-expressions ayant moins de paramètres que l'expression symbolique originale. En effet, en posant $h(x, y) = (x - \frac{1}{y})$ et $g(w) = \dfrac{1}{\sqrt{1 + w^2}}$, on a $f(x, y) = \dfrac{1}{\sqrt{1 + h^2(x, y)}} = g(h(x, y))$. Dans le cas de l'article [55] ces sous-expressions sont obtenues en résolvant un problème d'optimisation tandis que dans le cas de l'article [70] ces sous-expressions sont obtenues par de l'analyse fonctionnelle axée sur le calcul des gradients. Par exemple, supposons que f, h et g sont les fonctions inconnues. Alors, l'égalité $f(x, y) = g(h(x, y))$, ne peut être vraie que si le gradient ∇h de h et le gradient ∇f de f sont proportionnels, car, d'après le calcul des gradients, nous avons $\nabla f = g'(h(x, y)) \cdot \nabla h$. Une fois que h est identifiée par une simple fonction dont le gradient est proportionnel à celui de f, il suffit de remplacer dans la table des données les valeurs de (x, y) par les valeurs de $h(x, y)$ et l'on obtient une table de données de $g(w)$ qui est une fonction à un seul paramètre ω, ce qui la rend plus facile à déterminer. Aussi, le calcul du gradient ∇f est fait à partir d'une approximation de f apprise par un réseau de neurones. D'autres méthodes de réduction du nombre de paramètres sont également appliquées. L'équation (1.29) représente une loi en sciences physiques, redécouverte sur des données. La formule mentionne les

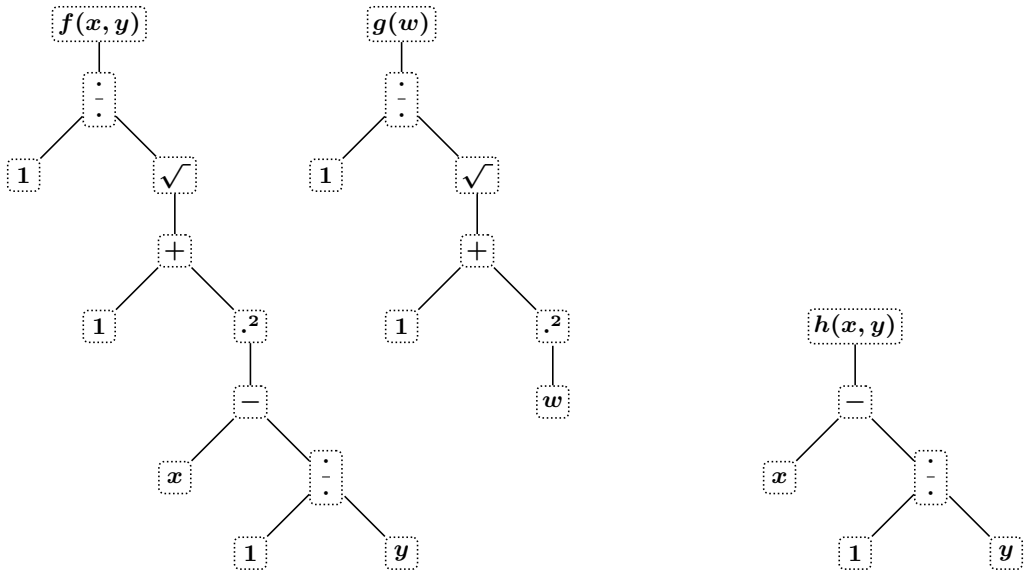


FIGURE 1.7 – Illustration d'une représentation d'une fonction sous la forme d'un arbre avec décomposition modulaire : la fonction f a une décomposition modulaire avec les fonctions g et h. La fonction g a un seul paramètre w alors que f les deux paramètres x et y. La fonction h est plus simple que f. En effet, on a $f(x, y) = g(h(x, y))$.

sept paramètres $I, V_0, \omega, t, R, L, C$ et les sept opérateurs $=, +, -, \sqrt{}, \div, cos, \times$ les connectant. Cet exemple montre le niveau de complexité des formules que la méthode de Silviu-Marian et *al.* [70] est capable de trouver.

$$I = \frac{V_0 \ cos(\omega t)}{\sqrt{R^2 + (\omega L - \frac{1}{\omega C})^2}} \tag{1.29}$$

L'identification des bons paramètres d'un modèle par la méthode des moindres carrés non linéaire en programmation génétique Cette méthode de Kommenda et *al.* [47] combine la programmation génétique avec l'optimisation non linéaire. En programmation génétique [48], la fonction f que l'on cherche est considérée comme un individu parmi un ensemble de fonctions. Cet ensemble est appelé population qui évolue selon des règles analogues à l'évolution biologique des êtres vivants. En effet, pour déterminer f, les algorithmes de programmation génétique aussi appelés algorithmes évolutionnaires exécutent des actions qui sont désignées par des termes de la biologie génétique. Notamment dans ces algorithmes, des individus de la population sont sélectionnés comme étant les plus prédisposés à faire évoluer la population vers la solution f selon un critère bien défini. Puis, les individus sélectionnés subissent des mutations ou des croisements. Pour effectuer une mutation, les individus sélectionnés sont représentés sous la forme d'arbres comme l'illustre la figure 1.7, puis le croisement consiste à s'échanger des branches tandis que la mutation consiste à remplacer une branche par une nouvelle. Puis, les nouveaux individus obtenus après ces opérations sont introduits

dans la population pour déclencher une opération de survie entre les individus qui conduit à une nouvelle population. Ces étapes sont répétées plusieurs fois.

La combinaison avec l'optimisation non linéaire consiste à calculer les valeurs optimales des constantes des feuilles des arbres de représentation des individus juste avant la phase de sélection. L'optimisation non linéaire des valeurs des constantes procède comme suit :

Étant donné un individu de la population, soit $T(\theta)$ son expression sous forme d'arbre avec $\theta \in \mathbb{R}^p$ un p-uplet où chaque composante est une des constantes de l'individu. Soit $T(\theta, X)$ l'évaluation de l'individu sur les données d'apprentissage (X, y) avec $y \in \mathbb{R}^n$. L'optimisation non linéaire des valeurs des constantes consiste à résoudre le problème suivant

$$\underset{\theta}{\text{Minimiser}} \ \frac{1}{2}\|T(\theta, X) - y\|^2. \tag{1.30}$$

Ce problème est résolu par l'algorithme de Levenberg–Marquardt qui emploie la méthode de descente de gradient et le jacobien de $T(\theta, X) - y$ calculé par la différenciation automatique [65]. L'équation (1.31) où $T(\theta, X) = T(\theta_1, \theta_2, \theta_3, \theta_4, x_1, x_2, x_3)$, est un exemple de formule que cette méthode est capable de redécouvrir dans les données.

$$T(\theta_1, \theta_2, \theta_3, \theta_4, x_1, x_2, x_3) = \frac{\theta_1(x_1 - \theta_2)(x_3 - \theta_3)}{x_2^2(x_1 - \theta_4)} \tag{1.31}$$

L'article [28] de La Cava et *al.*, fait une comparaison de 14 méthodes de régression symbolique où la méthode précédente qui combine la programmation génétique et l'optimisation non linéaire semble la meilleure.

Nous présentons maintenant les deux premiers systèmes dédiés essentiellement à la découverte automatique des conjectures et ayant été reconnus comme générant des conjectures assez significatives dans le domaine de l'automatisation de la recherche de conjectures [4]. Il s'agit du système *Graffiti* et du système *AutoGraphiX*.

L'heuristique Dalmatienne

Le système *Graffiti* [33] génère des bornes sur des caractéristiques d'un objet combinatoire en se basant sur un petit nombre d'instances de l'objet combinatoire et sélectionne les bornes les plus significatives. Pour cela le système *Graffiti* applique l'*heuristique Dalmatienne* décrite comme suit :

Soit u une borne supérieure potentielle d'une caractéristique α d'un objet combinatoire et un ensemble fini $O = \{O_1, ..., O_n\}$ d'instances de l'objet. Soit $u_1, ..., u_r$ des bornes supérieures déjà connues de α. On désigne par $\alpha(O_i)$ respectivement $u_j(O_i)$ la valeur de α, respectivement u_j pour l'objet O_i. Les deux tests suivants sont effectués :

— *test de vérité* : Vérification si la relation $\alpha(O_i) \leq u(O_i)$ est vraie pour tout élément O_i de O.

— *test de significativité* : Vérification s'il existe un objet O_s parmi ceux de O, pour lequel $u(O_s) < \min\{u_1(O_s), ..., u_r(O_s)\}$.

Lorsque les deux tests sont vrais alors u est validée comme conjecture sur la borne supérieure de α et ajoutée à l'ensemble $\{u_1, ..., u_r\}$ des bornes supérieures connues. Cependant, il est à signaler qu'avant l'application des deux tests, u est choisi arbitrairement.

Une remarque très importante dans cette heuristique est que le nombre de conjectures trouvées ne sera jamais supérieur au nombre d'objets utilisés pour faire le test de vérité. En effet, supposons que nous avons deux objets O_1, O_2 et trois bornes u_1, u_2, u_3. Alors, comme il y a plus de bornes que d'objets, il y a donc un objet qui vérifie le test de significativité pour au moins deux bornes distinctes. Supposons alors que cet objet est O_1 et que ces deux bornes sont u_1 et u_2. On a donc d'après le test de significativité :

$$u_1(O_1) < \min\{u_2(O_1), u_3(O_1)\} \text{ et } u_2(O_1) < \min\{u_1(O_1), u_3(O_1)\} \tag{1.32}$$

ce qui entraîne qu'on a la contradiction :

$$u_1(O_1) < u_2(O_1) \text{ et } u_2(O_1) < u_1(O_1). \tag{1.33}$$

qui est une contradiction à la supposition de départ. Par conséquent, lorsqu'une nouvelle borne est trouvée sans changement de la taille des objets, elle élimine une autre déjà connue qui dès lors est devenue moins significative. Ainsi, cette propriété qui permet de limiter le nombre de conjectures significatives est un moyen de répondre au problème rencontré par Hao Wang selon lequel son système de production de conjectures *Program II* était incapable par lui-même d'effectuer la sélection des conjectures significatives. C'est une heuristique qui, étant restée longtemps dans l'ombre, fait parler d'elle à partir de 2016 [52; 19; 24; 53], et est également utilisée en 2021 [22] pour la découverte de structures dans les données. Conjecture 1 est un exemple de conjecture découverte par *Graffiti*. Elle est présentée comme troisième conjecture dans l'article de Fajtlowicz [33].

Conjecture 1. Soit d_i le degré du i-ème sommet d'un graphe connexe ayant n sommets. Soit m le nombre de sommets du graphe qui ont le degré le plus fréquent parmi les degrés des sommets du graphe. Soit $E = \{(i, j) : i < j \in [1 : n]\}$ l'ensemble des arêtes du graphe. Alors nous avons la conjecture :

$$\sum_{i=1}^{n} \frac{1}{d_i} \leq m + \sum_{(i,j)\in E} \sqrt{d_i d_j} \tag{1.34}$$

Un autre système important de production de conjecture est *AutoGraphiX* élaboré par Hansen et Caporossi, que nous présentons maintenant.

La génération d'invariants de graphes extrêmes

Le système *AutoGraphiX* [26; 38; 3; 4; 25] part du principe que l'étude des cas limites permet d'apporter de l'information supplémentaire à la connaissance déjà acquise et qu'une fois ces cas limites maîtrisés, il devient moins ardu de passer à une généralisation à tous les cas possibles. Les cas limites générés par *AutoGraphiX* sont les graphes qui, pour certaines caractéristiques, atteignent leurs bornes. Ils sont qualifiés de *graphes extrêmes*. La génération de tels graphes est faite via la méthode de *variable neighbourhood search*. Une fois ces graphes générés, des conjectures sur les invariants de graphes en sont déduites en utilisant l'une des trois méthodes suivantes :

— la méthode numérique utilise l'analyse par composante principale, pas dans le but d'expliquer la variance c'est-à-dire la différence entre les invariants, mais cherche plutôt les points communs existants entre elles. En effet, soit la matrice X de taille m *lignes* $\times$ n *colonnes* telle que le coefficient $x_{i,j}$ de la i-ème ligne et la j-ème colonne représente une observation parmi n caractéristiques de m graphes extrêmes générés. Soit 1_m la matrice à une colonne avec tous ses m coefficients égaux à 1. Soit $\mu = \frac{1}{m}{}^t X \cdot 1_m$ le vecteur des moyennes pour chaque colonne de X. La méthode consiste à diagonaliser par élimination gaussienne la matrice centralisée des variances covariances ${}^t(X - {}^t 1_m \mu)(X - {}^t 1_m \mu)$. Cela fait, les lignes ne contenant que des termes nuls mettent en évidence les invariants qui sont dépendants linéairement.

— la méthode géométrique consiste à calculer l'enveloppe convexe dans un espace de p invariants $x_1, ..., x_p$, de tous les graphes qui sont extrêmes pour un certain invariant supposé dépendre des autres. Ainsi, chaque face de l'enveloppe convexe fournit une conjecture de la forme $\sum_{i=1}^{p} a_i x_i \geq b$ où les $a_1, ..., a_p$ et b sont des constantes.

— la méthode algébrique consiste à reconnaître la classe à laquelle appartient les graphes extrêmes générés pour une certaine formule et à utiliser les invariants connus de cette classe pour en déduire d'autres invariants par des procédés de substitution.

Conjecture 2 est un exemple de conjecture découverte par *AutoGraphiX*. Elle est présentée comme deuxième conjecture dans l'article de Caporossi et *al.* [26]. En plus de générer des conjectures, il est capable d'en réfuter. En effet, il a réfuté des conjectures générées par le système *Graffiti*.

Conjecture 2. Soient n_4 le nombre de sommets de degré 4 et α le nombre de couples des sommets non adjacents d'un arbre. Soit D le diamètre d'un arbre, c'est-à-dire la distance maximale entre deux sommets de l'arbre. Soit r le rayon d'un arbre, c'est-à-dire la plus petite distance à laquelle peut se trouver un sommet de tous les autres sommets. Alors nous avons la conjecture :

$$\alpha \leq \frac{1}{2}(m + n_4 + D - 2r) \tag{1.35}$$

1.3 Les limites des méthodes de l'état de l'art

Dans les domaines tels que la recherche opérationnelle, la programmation linéaire et la programmation par contraintes, identifier précisément les bornes des domaines des variables du problème combinatoire traité est crucial, car cela permet de réduire souvent l'espace de recherche des solutions. Or, remarquons que ces variables représentent le plus souvent les caractéristiques des objets combinatoires sous-jacents aux problèmes combinatoires. Ainsi, trouver des bornes de ces caractéristiques, c'est aussi trouver des bornes des variables des problèmes combinatoires. Cependant, la recherche des bornes se fait généralement manuellement par des mains expertes et peut se révéler rapidement fastidieuse. Les conjectures CONJECTURE 1 et CONJECTURE 2 sont des bornes sur des caractéristiques des graphes. Ces conjectures ont été trouvées automatiquement par les systèmes respectifs *Graffiti* et *AutoGraphiX*. Cela témoigne de la possibilité d'utiliser la machine pour assister l'humain dans la recherche des bornes. Ainsi, nous nous intéressons à la recherche automatique de non seulement des bornes, mais surtout de bornes précises. En effet, une borne précise délimite de manière exacte l'espace de recherche. Nous définissons maintenant ce que nous entendons par bornes précises.

Définition 27. Bornes précises

Soit $o, c_1, \cdots, c_n$ les $n+1$ caractéristiques d'un objet combinatoire. Soit f une fonction telle que la caractéristique o est bornée supérieurement (resp. inférieurement) selon l'expression

$$o \leq f(c_1, \ldots, c_n) (\text{resp. } o \geq f(c_1, \ldots, c_n)) \tag{1.36}$$

Nous disons que la borne $f(c_1, \cdots, c_n)$ de o est précise lorsque, *quelle que soit* la combinaison réalisable des valeurs $v_1, \cdots, v_n$ respectivement prises par les caractéristiques $c_1, \cdots, c_n$, il existe toujours une instance de l'objet combinatoire dont la valeur de la caractéristique o est v_o, pour laquelle on a l'égalité (1.37).

$$v_o = f(v_1, \ldots, v_n) \tag{1.37}$$

En d'autres termes, la borne (1.36) est précise lorsqu'elle est atteinte par une instance de l'objet combinatoire *pour toute* combinaison réalisable des valeurs des caractéristiques $c_1, \cdots, c_n$.

Nous appelons o la *caractéristique bornée* tandis que les caractéristiques $c_1, \cdots, c_n$ sont dénommées *caractéristiques bornantes*.

Exemple 10. Bornes précises

Comme exemples de bornes précises, nous avons :

— l'invariant $S \leq n^2$ des partitions d'un ensemble où S est la somme des carrés des tailles des partitions de l'ensemble et n est le cardinal de l'ensemble. En effet, l'instance qui

atteint la borne, quelle que soit la valeur de n est celle constituée d'une seule partition contenant tous les éléments de l'ensemble.

— l'invariant $a \leq v^2$ où a est le nombre d'arcs et v le nombre de sommets d'un graphe orienté ayant au moins un arc entrant ou sortant sur chaque sommet. En effet, l'instance atteignant la borne, quelle que soit la valeur de v est le graphe complet.

En nous basant sur les observations précédentes, nous énumérons les limites suivantes de l'état de l'art dans le cadre de la génération automatique d'invariants :

1. Le premier point faible des méthodes de l'état de l'art est le fait qu'elles ne garantissent pas que les conjectures générées puissent conduire à des bornes précises. Dans notre approche, non seulement nous garantissons que les bornes trouvées sont précises, mais également nous exploitons les techniques de la programmation par contraintes afin d'utiliser une riche grammaire de formules. Cette richesse permet de capturer un grand nombre d'informations précises existantes entre les caractéristiques des objets combinatoires. Notre réponse à ce premier point sera présentée au chapitre 2.
2. Le deuxième point faible est le fait qu'elles ne mettent pas en lumière des liens existant entre les bornes pouvant permettre de dériver une borne à partir d'une autre ou de mieux comprendre la structure reliant les bornes d'un objet combinatoire. Ce qui facilite l'élaboration des preuves des conjectures concernées. Dans l'approche que nous proposons, nous mettons en évidence des liens entre :

   — les bornes d'une caractéristique d'un même objet combinatoire ;
   — les bornes de plusieurs caractéristiques d'un même objet combinatoire ;
   — les bornes des caractéristiques de différents objets combinatoires.

   Notre réponse à ce deuxième point sera présentée aux chapitres 2, 3 et 4.
3. Dans le même sens, le troisième point faible est le fait que les approches existantes mettent très peu en avant la complexité et l'interprétabilité des formules générées. Dans l'approche que nous proposons, des caractéristiques bornantes sont exprimées à l'aide d'autres caractéristiques bornantes afin de les rendre plus interprétables. Ainsi, la formule de la borne est constituée de sous-termes interprétables, ce qui peut faciliter l'interprétabilité des bornes générées. De plus, l'approche utilise explicitement une heuristique de recherche des formules qui privilégie les expressions les moins complexes. Notre réponse à ce troisième point est présentée au chapitre 2.
4. Le quatrième point faible est le fait que, dans le cadre de la régression symbolique, bien qu'elle tienne compte parfois de la complexité des formules générées, ces modèles sont approximatifs. Il est donc difficile qu'ils conduisent à des théorèmes. Les méthodes de la régression symbolique sont conçues pour gérer des données erronées. De notre côté,

nous nous intéressons aux modèles qui collent parfaitement aux données. Ces données doivent être sans erreurs afin d'en déduire des théorèmes généraux pouvant s'appliquer à de nombreux problèmes combinatoires. Notre réponse à ce quatrième point est présentée au chapitre 4.

5. Le cinquième point faible est que, à part les applications faites dans le domaine de la chimie dans le cas des systèmes *Grafitti* et *AutoGraphiX* pour les conjectures de graphes, aucune autre exploitation des conjectures générées n'est faite, par exemple dans le cas de la programmation par contraintes. Dans notre approche, nous illustrons comment générer semi-automatiquement des propagateurs de contraintes en sélectionnant automatiquement les contraintes qui apportent plus de filtrage. Les travaux de Charnley *et al.* [29] se rapprochent de cette idée en générant des *contraintes implicites*. C'est-à-dire des contraintes induites par les contraintes explicites ou évidentes du problème combinatoire. Mais cette méthode se différencie de notre approche par les points suivants :

   — les contraintes implicites générées ayant une forme beaucoup plus simple que les nôtres, elles capturent donc moins d'informations exploitables ;
   — leur processus de sélection du sous-ensemble de contraintes implicites à conserver est coûteux, car il analyse l'effet de chaque sous-ensemble de contraintes implicites jusqu'à une limite du nombre de contraintes à conserver.

   Notre réponse à ce dernier point sera présentée au chapitre 5.

1.4 Discussion

Ainsi s'achève ce chapitre 1 sur les concepts préliminaires. Nous y avons présenté les notions de problème de satisfaction de contraintes et de problème d'optimisation combinatoire, ainsi que les outils utilisés en programmation par contraintes pour résoudre ces problèmes de manière automatique. Il s'agit notamment des solveurs et de leurs algorithmes de filtrage ayant des niveaux de filtrage distincts dénommés cohérences. Puis nous avons présenté quelques méthodes majeures de l'état de l'art dans le domaine de la génération automatique d'invariants. Ces invariants peuvent être utilisés pour améliorer l'efficacité des solveurs. Ensuite nous avons évoqué des limites de ces méthodes de l'état de l'art auxquelles nous répondrons dans les chapitres suivants. Puis pour terminer, nous avons présenté les objets combinatoires que nous étudions dans nos travaux. Cependant, en dehors des méthodes de génération de conjectures présentées, il existe d'autres méthodes dans le domaine de l'acquisition de contraintes [13; 16; 23; 49; 62; 64], mais elles sont trop spécifiques au problème donné et ne privilégient pas l'aspect preuves de conjectures. Par la suite, nous présentons au chapitre 2 comment nous générons automatiquement des liens entre conjectures de bornes précises d'une caractéristique d'un même objet combinatoire, ce qui permet de répondre au premier point ignoré dans l'état de l'art, à savoir

la recherche des bornes précises. Le chapitre 2 répondra aussi partiellement au deuxième point faible qui est la mise en évidence de liens entre conjectures d'un même objet combinatoire d'une même caractéristique. Enfin il répondra également au troisième point faible de l'état de l'art qui est la complexité et l'interprétabilité des formules découvertes.

Chapitre 2

Cartes de conjectures sur les bornes de caractéristiques d'objets combinatoires

Pour automatiser la découverte de conjectures sur les objets combinatoires, nous introduisons le concept d'une *carte de conjectures* sur les caractéristiques des objets combinatoires, qui fournit un ensemble de bornes précises interdépendantes pour ces objets combinatoires. Nous décrivons ensuite le *Bound Seeker*, un système basé sur la programmation par contraintes qui acquiert progressivement des cartes de conjectures. Le Bound Seeker a été testé pour la recherche de conjectures sur les bornes des caractéristiques des graphes orientés, des arbres et forêts enracinés, des partitions d'ensembles et des séquences de booléens. Pour chacun de ces objets combinatoires, il construit des dizaines de cartes regroupant des conjectures sur les bornes inférieures et supérieures précises de leurs caractéristiques associées. Chaque carte regroupe, soit des bornes supérieures précises d'une même caractéristique d'un même objet combinatoire, soit des bornes inférieures précises d'un même objet combinatoire. La formule donnant la borne s'exprime en fonction des autres caractéristiques du même objet combinatoire.

Quatre raisons motivent notre travail :

— souligner que la programmation par contraintes peut contribuer à la découverte automatique de conjectures,
— rechercher systématiquement des bornes précises sur les caractéristiques des objets qui apparaissent dans les problèmes combinatoires ;
— souligner la nécessité de développer des programmes de découverte de conjectures qui construisent un corps de connaissances fortement liées plutôt que des conjectures indépendantes les unes des autres comme c'est le cas jusqu'à présent ;
— par le fait que les bornes sont un composant essentiel des méthodes d'optimisation de type *branch-and-bound*, mais aussi un point faible de la programmation par contraintes.

En effet, la recherche de bornes précises prenant en compte plusieurs caractéristiques interdépendantes est encore un processus manuel géré au cas par cas. Notre approche est unique parmi tous les travaux de génération de conjectures, car le résultat n'est pas une collection de conjectures, mais plutôt un *graphe de conjectures*, reliées par des opérateurs de projection. Chaque opérateur de projection projette un ensemble de caractéristiques sur un de ses sous-ensembles en éliminant l'une des caractéristiques.

Nos contributions sont les suivantes :

1. Nous introduisons le concept de *carte de conjectures* comme un ensemble de conjectures interdépendantes fournissant des bornes inférieures et supérieures précises par rapport à la caractéristique d'un objet combinatoire.
2. Pour chaque conjecture sur une borne précise, la carte fournit quelques *caractéristiques extrêmes*, c'est-à-dire les valeurs des caractéristiques communes à toutes les instances de l'objet combinatoire atteignant cette borne précise.
3. En introduisant des caractéristiques secondaires et en autorisant l'utilisation de sous-expressions courantes d'un polynôme, ainsi que des conditionnelles simples, nous tendons à produire des *conjectures explicables*. Cela révèle également *des conjectures unifiées* à travers différents sous-ensembles de caractéristiques. Puis, nous comparons à plusieurs reprises nos résultats avec une méthode du Bound Seeker générant des formules booléennes. Notons que cette composante du Bound Seeker générant des formules booléennes ne fait pas partie des contributions des travaux présentés. Elle est publiée dans l'article [36] avec Ramiz Gindullin comme premier auteur.
4. Nous démontrons l'utilité de la programmation par contraintes pour l'acquisition de telles cartes : par exemple dans le cas des graphes orientés, le Bound Seeker produit 431 conjectures réparties en seize cartes. Chaque carte regroupe des bornes soit inférieures, soit supérieures d'une caractéristique parmi huit caractéristiques principales. Ainsi, le Bound Seeker retrouve un ensemble de résultats connus, améliore certaines bornes connues et produit de nouvelles conjectures, dont nous avons prouvé que certaines étaient vraies.

Les cartes ont une double interprétation :

— au-delà de proposer des bornes qui sont précises, une carte révèle et regroupe des relations entre ces bornes sous un même édifice. Elle précise également la structure des objets combinatoires atteignant chacune des bornes.
— une carte peut être utilisée pour tester la cohérence mutuelle de bornes acquises indépendamment les unes des autres en vérifiant qu'une borne peut être dérivée d'une autre borne en substituant la caractéristique que l'on veut éliminer par l'expression trouvée lorsqu'une caractéristique atteint sa valeur extrême.

Dans la section 2.1, nous introduisons le concept d'une *carte* qui présente un ensemble de

conjectures pour les bornes précises et leurs relations d'interdépendance. Dans la section 2.3.1, nous présentons l'ensemble des mécanismes utilisés par le système d'acquisition de cartes. Nous introduisons dans la section 2.3.8 un générateur de conjectures. Ce générateur est paramétré et il est basé sur la programmation par contraintes. Nous évaluons les conjectures produites dans la section 2.4 et concluons dans la section 2.5.

2.1 La carte de conjecture vue comme un ensemble structuré de connaissances révélant des propriétés d'un objet combinatoire

Après avoir donné un aperçu informel des cartes de conjectures et un premier exemple de carte, nous motivons, définissons et illustrons le concept de carte. Nous introduisons ensuite la notion de *caractéristiques secondaires* et montrons comment l'utilisation de telles caractéristiques permet à la fois d'acquérir des bornes partageant des sous-expressions communes, et parfois d'obtenir la même borne pour différents sous-ensembles de caractéristiques utilisées pour l'exprimer.

2.1.1 Aperçu informel d'une carte de conjectures

Considérons l'objet combinatoire que constitue un graphe orienté. Il est bien connu que tout graphe orienté $\mathcal{G}$ satisfait à l'invariant à savoir que le nombre d'arcs a de $\mathcal{G}$ est inférieur ou égal au carré v^2 du nombre de sommets v de $\mathcal{G}$, et que cette valeur maximale v^2 n'est atteinte que lorsque le graphe orienté $\mathcal{G}$ est une clique. Nous nous intéressons à la génération systématique de tels invariants candidats, c'est-à-dire la génération des conjectures, pour un ensemble plus riche de caractéristiques telles que le nombre de composantes connexes c de $\mathcal{G}$, le nombre $\underline{c}$ de sommets de la plus petite composante connexe de $\mathcal{G}$. Nos conjectures ont l'une des formes suivantes :

— des bornes précises des caractéristiques d'un graphe orienté exprimé en fonction d'autres caractéristiques du graphe orienté comme $a \leq v^2$. Cette expression donne la borne supérieure précise de a en fonction de v ;

— une implication montrant que, lorsqu'une borne précise est atteinte, certaines caractéristiques sont fixées ou déterminées fonctionnellement par d'autres caractéristiques, par exemple

$$(a = v^2) \Rightarrow (c = 1) \text{ et } \underline{c} = v). \tag{2.1}$$

Enfin, nos conjectures relient des bornes précises, révélant que le côté droit d'une implication comme celles mentionnées précédemment en (2.1), peut être utilisé pour éliminer une caractéristique d'une borne précise et retrouver une autre borne précise s'exprimant avec une caracté-

ristique de moins. Par exemple, en remplaçant $\underline{c}$ par v dans la borne précise $a \leq \underline{c}^2 + (v - \underline{c})^2$, nous retrouvons la borne précise $a \leq v^2$. Nous appelons carte l'ensemble de ces différentes conjectures et les liens reliant ces conjectures.

2.1.2 Motivation et définition du concept de carte

Nous introduisons le concept de *carte de conjectures* comme une façon de révéler les liens existant entre un ensemble de conjectures sur des bornes précises de caractéristiques d'objets combinatoires. Notre objectif est de décrire les conjectures sur les bornes précises de caractéristiques d'objets combinatoires et de les organiser en une unique structure qui :

— systématiquement interconnecte ces conjectures ;
— décrit la structure de l'objet combinatoire qui atteint la borne précise. Par exemple, dans la figure 2.1, nous considérons pour les graphes orientés trois caractéristiques a, v et c qui sont respectivement le nombre d'arcs, le nombre de sommets et le nombre de composantes connexes du graphe orienté.

Il s'agit précisément des objets combinatoires que sont les graphes orientés, les arbres enracinés, les forêts enracinées, les séquences de booléens ou encore les partitions d'ensembles.

Avant de définir le concept de carte, nous introduisons les notions de *conjecture maximum*, *conjecture minimum*, *conjecture de maximalité* et *conjecture de minimalité*.

Définition 28. Soit un ensemble de *caractéristiques bornantes* $\mathcal{P}$ d'un objet combinatoire $\mathcal{O}$, une *caractéristique bornée* $o \notin \mathcal{P}$ et un sous-ensemble $P_b \subseteq \mathcal{P}$ de caractéristiques bornantes. Soit également q une caractéristique du complémentaire de P_b dans $\mathcal{P}$. Pour une caractéristique $c \in P_b$, nous désignons par $\mathcal{D}(c)$ le domaine de ses valeurs possibles. Soient f et g_q deux fonctions de $\prod_{c \in P_b} \mathcal{D}(c)$ vers $\mathbb{N}$.

Nous appelons *conjecture maximum* (resp. *conjecture minimum*) une conjecture qui s'exprime sous la forme d'une borne précise $o \leq f(P_b)$ (resp. $o \geq f(P_b)$) selon la définition 27.

Nous appelons *conjecture de maximalité* (resp. *conjecture de minimalité*) une conjecture qui s'exprime sous la forme d'une égalité $q = g_q(P_b)$ qui est vérifiée lorsque o atteint son maximum (resp. minimum). C'est-à-dire que nous avons l'implication :

$$o = f(P_b) \Longrightarrow q = g_q(P_b) \tag{2.2}$$

L'exemple 11 suivant illustre la définition 28

Exemple 11. Pour une partition d'ensemble de cardinal n, partitionné en P partitions et telle la somme des carrés des tailles des partitions vaut S, nous identifions $o = S, P_b = \{n\}, q = P$.

Nous avons la conjecture maximum $S \leq n^2$ et la conjecture de maximalité $P = 1$. C'est-à-dire que nous avons l'implication :

$$S = n^2 \Longrightarrow P = 1 \tag{2.3}$$

En d'autres termes, pour que la somme des carrés des tailles des partitions soit égale au carré du cardinal de l'ensemble, il faut qu'il n'y ait qu'une seule partition.

Définition 29. Soient un ensemble de *caractéristiques bornantes* $\mathcal{P}$ d'un objet combinatoire $\mathcal{O}$, une *caractéristique bornée* $o \notin \mathcal{P}$ et deux sous-ensembles $P_{bi} \subseteq \mathcal{P}$ et $P_{bj} \subseteq \mathcal{P}$ de caractéristiques bornantes. Soit également $q_{i,j}$ une caractéristique du complémentaire de P_b dans $\mathcal{P}$. Soient f_i, f_j et g_q trois fonctions de $\prod_{c \in P_b} \mathcal{D}(c)$ vers $\mathbb{N}$.

Nous appelons *projection* de la conjecture maximum $o \leq f_i(P_{bi})$ sur la conjecture maximum $o \leq f_j(P_{bj})$ par élimination de la caractéristique $q_{i,j}$ à travers la conjecture de maximalité $q_{i,j} = g_{q_{i,j}}(P_{bj})$ sachant que $P_{bj} = P_{bi} \setminus \{q_{i,j}\}$, l'opération qui consiste à substituer la caractéristique $q_{i,j}$ par l'expression $g_{q_{i,j}}(P_{bj})$ dans la borne supérieure précise $f_i(P_{bi})$ pour obtenir la borne supérieure précise $f_j(P_{bj})$.

Nous disons alors que $f_j(P_{bj})$ est le *projeté* de $f_i(P_{bi})$ par élimination de la caractéristique $q_{i,j}$ à travers la conjecture de maximalité $q_{i,j} = g_{q_{i,j}}(P_{bj})$.

Nous appelons également cette opération une *projection* du sous-ensemble de caractéristiques bornantes P_{bi} sur le sous-ensemble de caractéristiques bornantes P_{bj} en éliminant la caractéristique $q_{i,j}$, c.-à-d. $P_{bj} = P_{bi} \setminus \{q_{i,j}\}$.

Nous définissons de manière analogue pour l'opération de projection entre deux conjectures minimum.

L'exemple 12 donne une projection entre deux conjectures du nombre d'arcs du graphe orienté.

Exemple 12. Dans un graphe orienté ayant v sommets, a arcs et tel que la taille de la plus petite composante connexe est $\underline{c}$, la conjecture maximum $a \leq v^2$ est le projeté de la conjecture maximum $a \leq \underline{c}^2 + (v - \underline{c})^2$ par élimination de la caractéristique $\underline{c}$ à travers la conjecture de maximalité $\underline{c} = v$.

En effet, en substituant $\underline{c}$ par v dans $\underline{c}^2 + (v - \underline{c})^2$, nous avons bien

$$\underline{c}^2 + (v - \underline{c})^2 = v^2 + (v - v)^2 = v^2 \tag{2.4}$$

Définition 30. Soit un ensemble de *caractéristiques bornantes* $\mathcal{P}$ d'un objet combinatoire $\mathcal{O}$, une *caractéristique bornée* $o \notin \mathcal{P}$ et trois sous-ensembles $P_b \subseteq \mathcal{P}$, $P_{bi} \subseteq \mathcal{P}$ et $P_{bj} \subseteq \mathcal{P}$ de caractéristiques bornantes. Soit également $q, q_{i,j}$ deux caractéristiques du complémentaire de P_b dans $\mathcal{P}$. Soit f, f_i, f_j, g_q et $g_{q_{i,j}}$ cinq fonctions de $\prod_{c \in P_b} \mathcal{D}(c)$ vers $\mathbb{N}$.

Une *carte de bornes supérieures précises* $\mathcal{M}_{\mathcal{P}}^{o\leq}$ est identifiée à un graphe orienté où :

— chaque nœud de la carte est associé à un sous-ensemble $P_b \subseteq \mathcal{P}$ de caractéristiques bornantes et correspond à une conjecture maximum $o \leq f(P_b)$. Et cette inégalité est une *borne précise*, c'est-à-dire qu'il existe une combinaison des valeurs des caractéristiques du paramètre P_b pour laquelle l'égalité est atteinte.

  De plus, un nœud contient ce que nous appelons des *conjectures de maximalité*. Ces conjectures de maximalité sont au nombre d'au plus un par caractéristique q du complémentaire de P_b dans $\mathcal{P}$. Ces conjectures de maximalité spécifient la propriété suivante : pour toute combinaison de valeurs des caractéristiques bornantes de P_b et pour toute instance d'objet combinatoire atteignant $f(P_b)$, on a la propriété que la caractéristique q est égale à $g_q(P_b)$.
— chaque arc d'une carte partant de la conjecture $o \leq f_i(P_{bi})$ et arrivant à la conjecture $o \leq f_j(P_{bj})$ correspond à une projection du sous-ensemble de caractéristiques bornantes P_{bi} sur le sous-ensemble de caractéristiques bornantes P_{bj} en éliminant la caractéristique $q_{i,j}$, c.-à-d. $P_{bj} = P_{bi} \setminus \{q_{i,j}\}$. L'arc est étiqueté par l'égalité $q_{i,j} = g_{q_{i,j}}(P_{bj})$ où $g_{q_{i,j}}(P_{bj})$ est la valeur donnée à $q_{i,j}$ pour atteindre l'égalité contenue dans le nœud de la conjecture $o \leq f_j(P_{bj})$. En d'autres termes, l'étiquette $q_{i,j} = g_{q_{i,j}}(P_{bj})$ est une *conjecture de maximalité* du nœud de la conjecture $o \leq f_j(P_{bj})$ dans la carte.

Dans une carte, il n'y a qu'une seule caractéristique que nous bornons par rapport à d'autres caractéristiques. En plus de la nommer caractéristique bornée, nous appelons aussi la caractéristique que nous bornons la *caractéristique de sortie*. Les caractéristiques bornantes utilisées pour borner la caractéristique de sortie sont aussi appelés *caractéristiques d'entrée*. Ainsi les conjectures maximums expriment des bornes précises de la caractéristique de sortie en fonction des caractéristiques d'entrée de P_b, tandis que les conjectures de maximalité indiquent les valeurs prises par les caractéristiques du complémentaire de P_b dans $\mathcal{P}$ lorsque la borne est atteinte par une instance de l'objet combinatoire. De manière similaire que pour la carte $\mathcal{M}_{\mathcal{P}}^{o\leq}$, une carte $\mathcal{M}_{\mathcal{P}}^{o\geq}$ constitue une collection de bornes inférieures et précises que nous appelons *conjectures minimum* de la forme $o \geq f_j(P_{bj})$ et un ensemble d'égalités qui sont vérifiées lorsqu'une instance de l'objet combinatoire atteint la borne inférieure et précise. Ces égalités sont appelées *conjectures de minimalité*.

Comme premier exemple, nous présentons dans l'exemple 13, une carte sur le graphe orienté avec les caractéristiques suivantes : le nombre v de sommets, le nombre a d'arcs, le nombre c (resp. s) de composantes connexes (resp. composantes fortement connexes), le nombre $\underline{c}$ (resp. $\overline{c}$) de sommets de la plus petite (resp. plus grande) composante connexe, le nombre $\underline{s}$ (resp. $\overline{s}$) de sommets de la plus petite (resp. plus grande) composante fortement connexe. Pour comparer les bornes obtenues par le Bound Seeker avec la base de données d'invariants du catalogue de contraintes globales, voir la section 4.3 de l'article [10], nous considérons que chaque sommet d'un graphe orienté possède au moins un arc entrant ou sortant.

Exemple 13. La figure. 2.1 illustre le concept de carte à l'aide d'une carte contenant trois conjectures du graphe orienté. Ces conjectures sont étiquetées par ❶, ❷ et ③ qui représentent :

— deux conjectures de bornes précises ❶ $a \leq (v-(c-1))^2+(c-1)$, et ❷ $a \leq v^2$ sur le nombre d'arcs a d'un graphe orienté $\mathcal{G}$ en fonction du nombre de sommets v et du nombre de composantes connexes c de $\mathcal{G}$.

— la conjecture ③ du nœud (B) indique que la borne v^2 est atteinte uniquement lorsque $c=1$.

— l'arc partant du nœud (A) et arrivant au nœud (B) est étiqueté par ③ car la borne v^2 est obtenue en remplaçant c par 1 dans la borne $(v-(c-1))^2+(c-1)$. Les boîtes de gauche et de droite de la figure. 2.1 illustrent, en bleu, deux graphes orientés atteignant chacun la borne supérieure correspondante du nombre d'arcs.

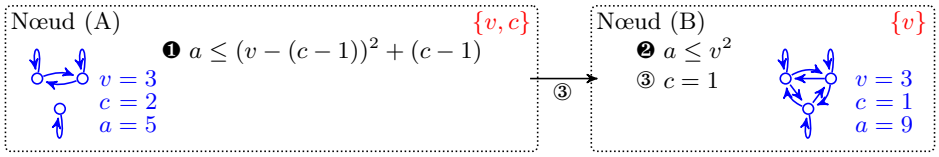


FIGURE 2.1 – Carte de deux bornes supérieures précises du nombre d'arcs d'un graphe orienté.

Dans la suite, toute carte est présentée de la même manière que la figure 2.1 :

1. le coin supérieur gauche d'une boîte donne le *nom de la boîte* en noire ;
2. le coin supérieur droit d'une boîte donne en rouge les *caractéristiques* utilisées pour exprimer la borne précise de la caractéristique à borner ;
3. un symbole noir de la forme ❶ permet d'étiqueter la *borne précise* ;
4. un symbole blanc de la forme ① indique l'*égalité* qui doit être vérifiée pour que la borne de la même boîte soit atteinte ;
5. une instance en bleue *illustre la précision de la borne* ;
6. un arc partant d'une boîte et arrivant à une autre indique quelle égalité contenue dans la boîte d'arrivée de l'arc devrait être utilisée pour éliminer certaines caractéristiques de la borne de la boîte de départ de l'arc, afin d'obtenir la borne de la boîte d'arrivée de l'arc.

Cependant pour des raisons de clarté, il peut arriver que les points 1. et 5. soient omis comme c'est le cas de la figure 2.4 lorsque qu'une carte comporte trop de bornes.

Exemple 14 (Extention de l'exemple 13 à une carte ayant quatre nœuds de conjectures)**.** La figure. 2.2 présente la carte $\mathcal{M}^{a\leq}_{\{v,c,\underline{c}\}}$, qui considère les caractéristiques suivantes du graphe orienté : les caractéristiques bornantes sont le nombre de sommets v, le nombre de composantes connexes c et la taille de la plus petite composante connexe $\underline{c}$; la caractéristique bornée est le nombre d'arcs a. Dans la carte $\mathcal{M}^{a\leq}_{\{v,c,\underline{c}\}}$, nous avons quatre nœuds correspondant chacun à un

sous-ensemble de caractéristiques bornantes $\{v, c, \underline{c}\}$, $\{v, \underline{c}\}$, $\{v, c\}$ et $\{v\}$ (en couleur rouge), tandis que l'ensemble puissance $\{v, c, \underline{c}\}$ contient huit sous-ensembles. Pour les quatre sous-ensembles restants, notamment $\{c, \underline{c}\}$, $\{\underline{c}\}$, $\{c\}$ et $\emptyset$, il n'existe pas de conjectures, car le nombre d'arcs d'un graphe orienté n'est pas borné supérieurement en fonction de ces caractéristiques. Dans les nœuds (A), (B), (C) et(D), les éléments étiquetés avec ❶, ❷, ❸ et ❹ indiquent une conjecture maximum du nombre d'arcs a, tandis que les éléments étiquetés avec ⑤, ⑥, ⑦ et ⑧ fournissent les conjectures de maximalité pour c et $\underline{c}$. Par exemple, dans le nœud (B), la conjecture maximum ❷ $a \leq \underline{c}^2 + (v - \underline{c})^2$ signifie concrètement que, parmi tous les graphes orientés de v sommets et dont leurs plus petites composantes connexes a $\underline{c}$ sommets, le graphe orienté qui a le plus d'arcs a exactement $\underline{c}^2+(v-\underline{c})^2$ arcs. Chaque arc de la carte est étiqueté par une conjecture de maximalité donnant la valeur de la caractéristique à éliminer. Par exemple, du nœud (A) au nœud (B), la caractéristique c qui est éliminée de ❶ vérifie la conjecture de maximalité ⑤ : lorsque le nombre d'arcs maximum est atteint, la valeur de c est 1 si $v = \underline{c}$, sinon c est égal à 2.

Dans le même ordre d'idée, la figure 2.3 présente la carte $\mathcal{M}^{S\leq}_{\{n,P,\underline{M}\}}$ de l'objet combinatoire que constitue la partition d'ensembles.

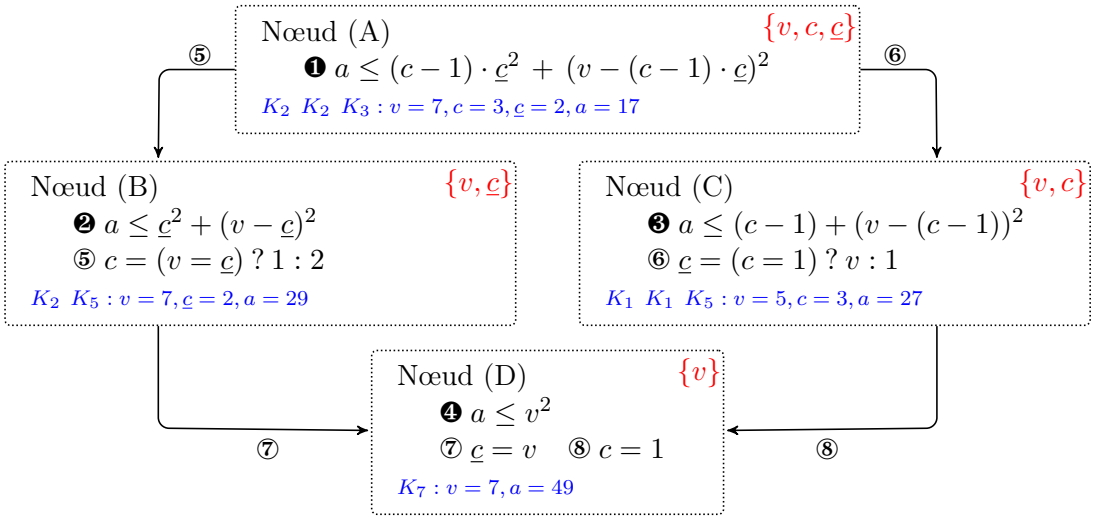


FIGURE 2.2 – Carte $\mathcal{M}^{a\leq}_{\{v,c,\underline{c}\}}$ de bornes supérieures précises ❶, ❷, ❸, ❹ du nombre d'arcs dans un graphe orienté. Chaque nœud de la carte présente une instance en bleu du graphe orienté : pour chaque combinaison de valeurs des caractéristiques bornantes en rouge de chaque nœud de la carte, l'instance atteignant le maximum d'arcs est décrite comme une union de *cliques* K_i avec i sommets. Une *clique* est un graphe orienté où tout couple de sommets est connecté par un arc et chaque sommet possède une boucle. Par exemple pour le nœud (B), avec la combinaison de valeurs $v = 7$ et $\underline{c} = 2$, le graphe orienté avec 2 cliques K_2, K_5 atteint le maximum de $a = 29$ arcs ; l'expression $cond\,?\,x : y$ s'évalue à x si la condition $cond$ est vérifiée et s'évalue à y dans le cas contraire.

2.2 Exploitation de caractéristiques secondaires pour capturer plus de bornes précises

Plus les caractéristiques d'entrée sont nombreuses, plus la borne précise peut s'avérer très complexe. Par conséquent nous introduisons un ensemble $\mathcal{A}$ de caractéristiques auxiliaires afin d'obtenir des formules moins complexes. De telles caractéristiques auxiliaires ne sont jamais considérées, au départ, comme des caractéristiques d'entrée, mais elles sont, en fin de compte, utilisées dans l'expression de la borne afin de la simplifier. Dans le cadre des graphes orientés, ce sont par exemple les caractéristiques notées par $(i)\, c_{>1}$, $(ii)\, s_{>1}$, et $(iii)\, c_{\in\{2,3\}}$ qui correspondent respectivement (i) au nombre de composantes connexes de taille supérieure à un sommet, (ii) au nombre de composantes fortement connexes de taille supérieure à un sommet, et (iii) au nombre de composantes connexes de deux ou trois sommets tels que leurs composantes fortement connexes sont toutes de taille un sommet. Ces caractéristiques qui initialement étaient introduites pour la recherche de certaines bornes inférieures précises se sont avérées très utiles pour de nombreuses autres bornes. Les exemples 15 et 16 illustrent com-

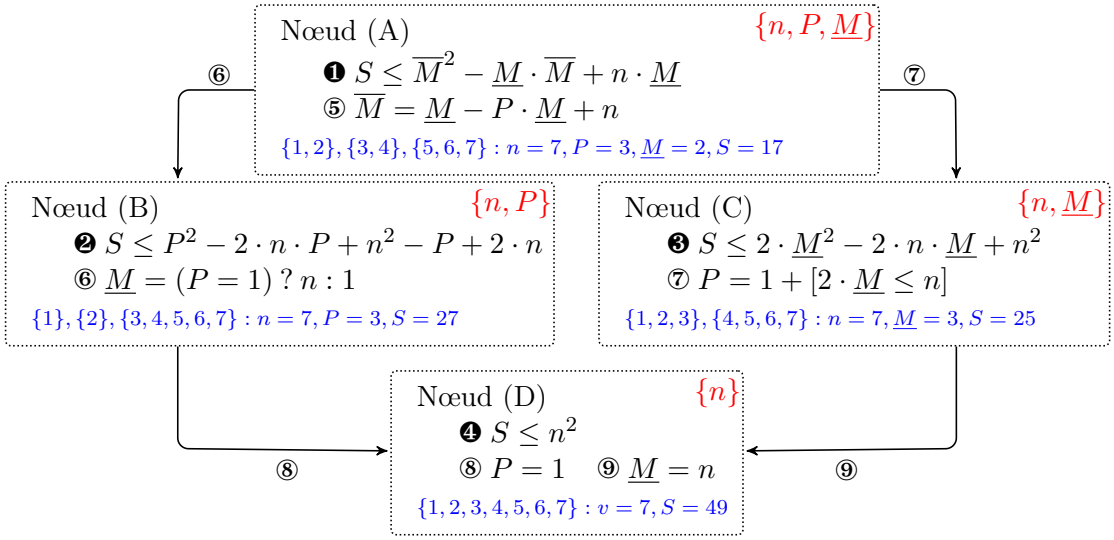


FIGURE 2.3 – Carte $\mathcal{M}^{S\leq}_{\{n,P,\underline{M}\}}$ des bornes supérieures précises ❶, ❷, ❸, ❹ de la somme S des carrés des tailles des partitions d'un ensemble. Chaque nœud de la carte présente une instance en bleu d'une partition d'un ensemble à 7 éléments. Par exemple pour le nœud (C), la combinaison de valeurs $n = 7$ et $\underline{M} = 3$ représente le partitionnement d'un ensemble à $n = 7$ éléments en deux sous-ensembles tels que le plus petit sous-ensemble a $\underline{M} = 3$ éléments. Ce partitionnement atteint le maximum de S qui est $2 \cdot \underline{M}^2 - 2 \cdot n \cdot \underline{M} + n^2 = 25$. De plus, on obtient la conjecture maximum ❸ du nœud (C) en éliminant la caractéristique P dans la conjecture maximum ❶ en la substituant par le terme de droite de la conjecture de maximalité ⑦ $P = 1 + [2 \cdot \underline{M} \leq n]$. Ce terme $1 + [2 \cdot \underline{M} \leq n]$ est la somme de l'entier 1 et d'une expression booléenne $[2 \cdot \underline{M} \leq n]$ qui vaut 1 si la condition $2 \cdot \underline{M} \leq n$ est vérifiée, et 0 autrement. L'expression $cond\,?\,x : y$ s'évalue à x si la condition $cond$ est vérifiée et s'évalue à y autrement.

ment ces caractéristiques auxiliaires sont utilisées. De même, les caractéristiques qui peuvent être des caractéristiques d'entrée jouent parfois le même rôle qu'une caractéristique auxiliaire lorsqu'elles ne font pas partie des caractéristiques d'entrée de la borne. Par exemple, dans le nœud (A) de la carte $\mathcal{M}^{S\leq}_{\{n,P,\underline{M}\}}$ de la figure 2.3, la caractéristique $\overline{M}$ est une caractéristique secondaire, car elle ne fait pas partie de l'ensemble $\{n, P, \underline{M}\}$ des caractéristiques bornantes. Cependant, elle est exploitée pour exprimer la borne $S \leq \overline{M}^2 - \underline{M} \cdot \overline{M} + n \cdot \underline{M}$. De plus, elle pourrait aussi être utilisée comme caractéristique d'entrée dans une autre carte, ce qui n'est pas le cas d'une caractéristique auxiliaire. C'est ainsi que nous regroupons les caractéristiques auxiliaires et les caractéristiques d'entrée sous la notion de caractéristique secondaire que nous introduisons par la suite.

Définition 31. Étant donné un nœud de la carte associée à un sous-ensemble de caractéristiques bornantes $P_b \subseteq \mathcal{P}$, une caractéristique bornée o, une conjecture maximum de la forme $o \leq f(P_b)$, et un ensemble de caractéristiques auxiliaires $\mathcal{A}$, l'*ensemble des caractéristiques secondaires* d'un nœud de la carte est défini comme les caractéristiques de l'ensemble $\mathcal{A} \cup (\mathcal{P} \setminus (P_b \cup \{o\}))$ qui sont *fonctionnellement déterminées* par P_b avec $o = f(P_b)$.

Pour vérifier qu'une caractéristique est fonctionnellement déterminée par P_b, nous vérifions que pour chaque combinaison de valeurs générées des caractéristiques de P_b, la valeur de la caractéristique secondaire est unique. Ce test est réalisé lors de la génération de notre jeu de données utilisé pour l'acquisition de conjectures. Nous disons aussi que la caractéristique est *fonctionnellement dépendante* de P_b.

Ces caractéristiques sont introduites manuellement. En effet, nous nous inspirons des caractéristiques des objets combinatoires déjà existantes dans la littérature, de l'analyse de certaines instances de l'objet combinatoire lors d'une recherche manuelle d'une borne peu complexe, pour définir des caractéristiques secondaires que nous jugeons utiles pour exprimer les bornes de caractéristiques d'un objet combinatoire. Par exemple, l'introduction de la caractéristique $c_{\in\{2,3\}}$ qui représente le nombre de composantes connexes ayant deux ou trois sommets tels que chacun de ces sommets est une composante fortement connexe vient du fait que nous avons constaté que les instances de graphes orientés (où chaque sommet a au moins un arc entrant ou un arc sortant) qui ont une valeur minimum du nombre d'arcs lorsque le nombre de sommets et le nombre de composantes fortement connexes sont fixés ont généralement des composantes connexes ayant non seulement deux ou trois sommets, mais aussi, chacun de ces sommets forme une composante fortement connexe à lui tout seul. Ce sont les composantes connexes de la forme •→•, •→•→•, •→•←• ou •←•→• . Donc parfois, pour introduire manuellement une nouvelle caractéristique secondaire, nous étudions la structure de l'objet combinatoire pour identifier des patrons récurrents.

Pour découvrir des bornes en exploitant ces caractéristiques secondaires, nous utilisons une approche à plusieurs phases :

— à la première étape, le Bound Seeker recherche une formule pour chaque caractéristique secondaire en fonction des caractéristiques d'entrée ou bornantes ;
— puis à la deuxième étape, le Bound Seeker recherche une formule pour chaque caractéristique secondaire en fonction des caractéristiques bornantes et des caractéristiques secondaires dont une formule a été trouvée à la première étape ;
— enfin, à la troisième étape, le Bound Seeker recherche une borne précise de la caractéristique bornée en considérant les caractéristiques secondaires comme des caractéristiques bornantes.

L'utilisation des caractéristiques secondaires facilite l'interprétation et la compréhension des formules découvertes dans un objectif d'établissement de preuves des conjectures et de vulgarisation scientifique. En effet, nous pouvons directement donner la signification des sous-termes qui s'identifient aux caractéristiques secondaires et qui composent la formule découverte. L'exemple 15 illustre ce point.

Exemple 15 (Borne exprimée en fonction de plusieurs caractéristiques secondaires)**.** Cet exemple présente une borne découverte par le Bound Seeker, du nombre d'arcs a d'un graphe orienté $\mathcal{G}$ en fonction de la taille $\overline{c}$ de sa plus grande composante connexe et de la taille $\underline{s}$ de la plus petite composante fortement connexe. Nous avons $\mathcal{P} = \{v, a, c, \underline{c}, \overline{c}, s, \underline{s}, \overline{s}\}$, les caractéristiques bornantes $P_b = \{\overline{c}, \underline{s}\}$, la caractéristique bornée $o = a$, et les caractéristiques auxiliaires $\mathcal{A} = \{c_{>1}, s_{>1}\}$. Les caractéristiques potentielles secondaires sont donc $\mathcal{A} \cup (\mathcal{P} \setminus (P_b \cup \{o\})) = \{v, c, \underline{c}, s, \overline{s}, c_{>1}, s_{>1}\}$. Ces caractéristiques sont fonctionnellement déterminées par $\overline{c}$ et $\underline{s}$. La borne inférieure précise découverte par le Bound Seeker est $a \geq s_{>1} - c_{>1} + v$ avec :
• $s_{>1} = \min(-\underline{s} + \overline{c} + 1, 2 \cdot [\underline{s} \geq 2])$,
• $c_{>1} = (\overline{c} = c \,?\, 0 : c)$ où $c = 1 + (((\overline{c} - 2 \cdot \underline{s}) \leq 0) \wedge ((\overline{c} \mod \underline{s}) \geq 1))$,
• $v = ((\overline{c} - \underline{c}) = 0 \,?\, \overline{c} : \overline{c} + \underline{c})$ où $\underline{c} = ((2 \cdot \underline{s} - \overline{c}) \leq 0 \,?\, \overline{c} : \underline{s})$.
L'expression $[\underline{s} \geq 2]$ utilisant le crochet d'Iverson vaut soit 1 si $\underline{s} \geq 2$ ou 0 sinon. Bien que la formule principale $s_{>1} - c_{>1} + v$ soit simple, elle utilise une caractéristique secondaire $s_{>1}$ qui s'exprime directement en fonction de $\overline{c}$ et $\underline{s}$. La formule de la borne utilise également deux autres caractéristiques $c_{>1}$ et v qui mentionnent dans leurs formules deux autres caractéristiques secondaires c et $\underline{c}$ pour lesquelles des formules impliquant seulement $\overline{c}$ et $\underline{s}$ sont découvertes. La table 2.1 illustre sur chaque ligne, une combinaison des valeurs des caractéristiques impliquées et leurs instances associées qui atteignent la borne inférieure du nombre d'arcs a.

Dans une même carte, exprimer les mêmes bornes en fonction des caractéristiques secondaires peut révéler une même formule de borne pour plusieurs sous-ensembles de caractéristiques bornantes, c'est-à-dire pour plusieurs nœuds de la même carte. Nous avons fait ce constat sur un grand nombre de cartes générées. L'exemple 16 illustre ce phénomène dans le cadre de la carte qui regroupe les bornes supérieures précises de la taille de la plus grande composante connexe d'un graphe orienté.

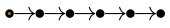
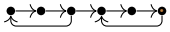
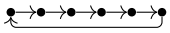
$\overline{c}$	$\underline{s}$	a	v	c	$\underline{c}$	$c_{>1}$	$s_{>1}$	graphe orienté témoin	$s_{>1} - c_{>1} + v$
6	1	5	6	1	6	1	0		$0 - 1 + 6$
6	3	7	6	1	6	1	2		$2 - 1 + 6$
6	4	10	10	2	4	2	2		$2 - 2 + 10$
6	6	6	6	1	6	1	1		$1 - 1 + 6$

TABLE 2.1 – Graphes orientés, avec au moins un arc entrant ou sortant pour chaque sommet, minimisant le nombre d'arcs pour quatre combinaisons de valeurs des caractéristiques bornantes $\overline{c}$ et $\underline{s}$.

Exemple 16 (Exemple de carte illustrant comment des bornes de plusieurs nœuds d'une même carte sont unifiées par l'utilisation de caractéristiques secondaires)**.** La figure. 2.4 présente les conjectures maximums et de maximalité de la carte $\mathcal{M}^{\overline{c}\leq}_{\{v,c,\underline{c},s,\underline{s},\overline{s}\}}$ découvertes par le Bound Seeker pour les bornes supérieures précises de la taille de la plus grande composante connexe $\overline{c}$ d'un graphe orienté. La carte révèle les différents liens de dérivation d'une borne à partir d'une autre. Notez que pour s'assurer qu'une borne existe bien, la caractéristique v fait toujours partie des caractéristiques bornantes. La partie (A) présente seize bornes découvertes en n'utilisant que les caractéristiques bornantes du nœud associé de la carte : ces bornes sont identifiées par cinq conjectures maximum ❶,...,❺ et quatre conjectures de maximalité ⑥,...,⑨. Chaque lien ou arc illustre comment une conjecture maximum est projetée sur une autre conjecture maximum via une conjecture de maximalité : par exemple, le lien ❺ $\xrightarrow{⑦}$ ❶ montre comment la borne ❺ $\overline{c} \leq \underline{s} - c \cdot \underline{s} + v$ est réécrite sous la forme ❶ $\overline{c} \leq v$ grâce à ⑦ $c = 1$. La partie (B) montre les bornes trouvées en fonction des caractéristiques secondaires r et $\underline{c}$, où r est une caractéristique secondaire correspondant à $v - c \cdot \underline{c}$. Nous avons seulement deux conjectures maximum ❶ $\overline{c} \leq v$ et ❷ $\overline{c} \leq r + \underline{c}$, avec r et $\underline{c}$ définis en fonction de cinq conjectures de maximalité ③,...,⑦ illustrées par la partie (B). La borne supérieure naturelle de $\overline{c}$ est le nombre de sommets du graphe orienté (voir ❶), sauf si c ou $\underline{c}$ font partie des caractéristiques bornantes (voir ❷), ce qui requiert de tenir compte des conditions de réalisabilité des instances induites par ces caractéristiques d'entrée.

Les arcs de projection manquants sont dus à l'absence de dépendance fonctionnelle de la caractéristique à éliminer. Par exemple, dans la partie (A), il n'y a pas d'arc allant du nœud $\{v, s\}$ au nœud $\{v\}$, car le nombre de composantes fortement connexes s n'est pas fonctionnellement dépendant de v lorsque la borne ❶ est atteinte, c'est-à-dire lorsque $\overline{c} = v$: en effet, pour $\overline{c} = v = 2$ nous avons à la fois $s = 2$ et $s = 1$ comme le l'illustrent et .

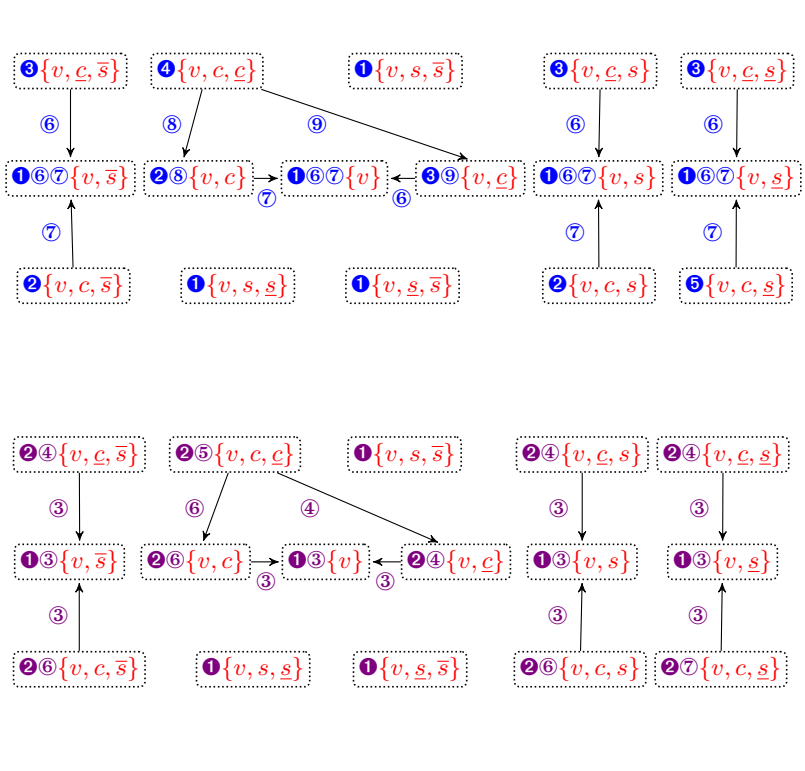


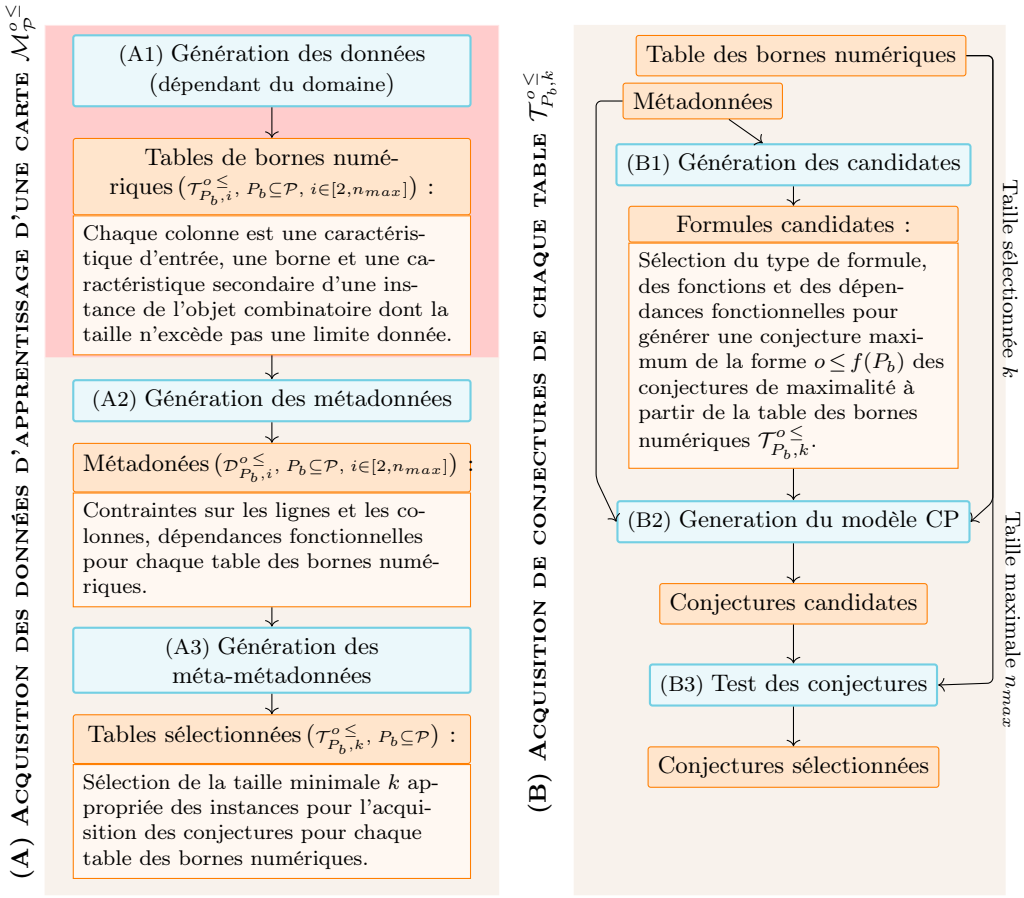
FIGURE 2.4 – Carte $\mathcal{M}^{\overline{c}\leq}_{\{v,c,\underline{c},s,\underline{s},\overline{s}\}}$ des bornes supérieures de la caractéristique de sortie $\overline{c}$, découvertes par le Bound Seeker, où chaque nœud de la carte contient, de la gauche vers la droite, une référence à la conjecture maximum ❶,...,❺, ❶,❷, puis possiblement un ensemble de conjectures de maximalité ⑥,...,⑨,③,...,⑦, et un ensemble de caractéristiques d'entrée en rouge ; la partie (A) correspond aux bornes découvertes en utilisant les caractéristiques d'entrée, tandis que la partie (B) fait référence aux mêmes bornes découvertes en utilisant en plus les caractéristiques secondaires r et $\underline{c}$.

2.3 Le Bound Seeker

2.3.1 Vue d'ensemble du système d'acquisition des cartes de conjectures

Les parties (A) et (B) de la figure 2.5 présentent les différentes phases intervenant dans la génération d'une carte : les composantes du système d'acquisition sont en couleur cyan. Ces composantes sont étiquetées avec des lettres majuscules, tandis que les types de données impliquées sont présentés en orange.

Nous détaillons dans la suite les différentes phases (A1), (A2), (A3), (B1), (B2), et (B3) impliquées dans la génération de conjectures. Pour expliquer chaque phase, nous utilisons la table $\mathcal{T}^{a\leq}_{\{v,c\},3}$ du jeu de données d'acquisition de bornes précises, fournie dans la partie (C) de la figure. 2.5.



(C) Exemple de table des bornes numériques

Combinaison des valeurs des caractéristiques		Bornes numériques de a	Caractéristique secondaire	Graphes orientés correspondants
v	c	a	$\overline{c}$	$\mathcal{G}$
1	1	1	1	K_1
2	1	4	2	K_2
2	2	2	1	K_1, K_1
3	1	9	3	K_3
3	2	5	2	K_2, K_1
3	3	3	1	K_1, K_1, K_1

Figure 2.5 – Organisation des composantes du Bound Seeker : la partie de gauche (A) illustre la génération des données d'acquisition de conjectures tandis que la partie de droite (B) présente les phases d'acquisition des conjectures ; la phase (A1) sur un fond rouge dépend du type d'objet combinatoire qui est considéré, tandis que les phases (A2), (A3), ..., (B3) en sont indépendantes ; la partie (C) présente un exemple de table de bornes avec les instances associées. C'est une table des bornes numériques supérieures de la *caractéristique d'entrée* qu'est le nombre d'arcs a d'un graphe orienté d'au plus trois sommets en fonction des *caractéristiques d'entrée* v, c et de la *caractéristique secondaire* $\overline{c}$ qui représente le nombre de sommets de la plus grande composante connexe. Pour chaque ligne de la table, il existe au moins une instance de graphe orienté possédant les valeurs des caractéristiques se trouvant sur ladite ligne.

2.3.2 Phase (A1) : Génération du jeu de données

Pour générer le jeu de données, nous considérons une caractéristique qui permet de quantifier la taille d'une instance de l'objet combinatoire. Nous appelons cette caractéristique *taille de l'instance de l'objet combinatoire*. C'est par exemple le nombre de sommets v d'un graphe orienté, d'un arbre enraciné ou encore d'une forêt enracinée. C'est également le cardinal n de l'ensemble à partitionner ou encore la longueur d'une séquence d'entiers 0/1. Pour apprendre des conjectures valides pour toute instance d'un objet combinatoire ayant une taille valant au plus k, nous produisons toutes les combinaisons de valeurs de toutes les caractéristiques d'intérêt de l'objet combinatoire pour les instances de taille d'au plus k. Puis pour chaque combinaison de valeurs des caractéristiques bornantes, nous ne retenons que la combinaison des valeurs de toutes les caractéristiques d'intérêt pour laquelle la caractéristique à borner a une valeur maximale ou une valeur minimale selon que l'on s'intéresse à une borne supérieure précise ou à une borne inférieure précise. Cela permet de s'assurer que la borne à découvrir est précise. Une génération exhaustive d'un tel jeu de données n'est pas un problème puisque c'est un programme qui est chargé de faire cela. Cependant, une difficulté est de déterminer la valeur appropriée de la taille k qui ne doit être ni trop petite afin d'éviter de générer des conjectures invalides pour des instances de grandes tailles, ni trop grande afin de limiter le nombre de données à générer pour apprendre des conjectures à la phase (B2). En effet, au moins une contrainte est générée pour chaque instance retenue dans le jeu des données. Pour donc déterminer la valeur appropriée de k, la phase (A1) produit une table $\mathcal{T}$ avec les valeurs des caractéristiques pour des instances de l'objet combinatoire d'une taille maximale n_{max} dans les limites de la mémoire du système d'acquisition. De cette table $\mathcal{T}$, la phase (A1) extrait pour chaque i compris entre 2 et n_{max}, pour chaque sous-ensemble P_b de caractéristiques bornantes de $\mathcal{P}$, et pour chaque caractéristique à borner o, une table dont chaque colonne correspond à une caractéristique et chaque ligne représente une instance de l'objet combinatoire pour lequel la valeur de la caractéristique à borner est un maximum ou un minimum. Nous appelons une telle table *table des bornes numériques* $\mathcal{T}^{o\leq}_{P_b,i}$ de la caractéristique de sortie. Cette table ne contient que les entrées de $\mathcal{T}$ qui correspondent aux instances de taille d'au plus i. Chaque ligne de la table des bornes numériques représente une *combinaison réalisable* des valeurs des caractéristiques de P_b avec la valeur optimale correspondante de o, et les valeurs correspondantes des caractéristiques secondaires.

Contrairement aux prochaines étapes, la phase (A1) dépend du type d'objet combinatoire pour lequel les conjectures sont générées. Les techniques de génération d'instances propres à chaque objet combinatoire étudié dans ce travail sont présentées par la suite.

Pour acquérir des propriétés des objets combinatoires, nous générons nos propres instances de ces objets combinatoires et calculons les caractéristiques associées définies aux sous-sections précédentes. Nous présentons maintenant comment nous procédons pour générer ces instances.

Comme nous sommes plus intéressés par les combinaisons valides des caractéristiques de l'objet combinatoire, que par les instances associées, nous générons plutôt des représentations de ces instances, qui occupent moins d'espace de stockage, lorsque c'est possible. C'est le cas par exemple de la génération des données du graphe orienté où, au lieu de générer des graphes avec des arcs, nous générons des partitions de partitions. En effet, lorsqu'on ne considère pas les arcs, un graphe orienté est une partition de composantes connexes qui sont elles-mêmes des partitions de composantes fortement connexes. Dans la même idée de sauvegarder de l'espace de stockage, nous exploitons le fait qu'une combinaison valide de valeurs de caractéristiques peut être associée à plusieurs instances. Nous disons alors que ces instances sont *symétriques* et il suffit de ne générer qu'une seule de ces instances. C'est ainsi que nous imposons des contraintes dans les méthodes de génération des données, afin de ne générer dans la mesure du possible qu'une seule instance pour un ensemble d'instances symétriques. Après la génération des instances, le calcul des valeurs des caractéristiques est effectué. Puis, les combinaisons des valeurs, telles que la valeur de la caractéristique bornée est minimale ou maximale, sont retenues dans une collection de tables. Chaque table présente un ensemble de combinaisons de valeurs de caractéristiques d'entrée avec les valeurs minimales ou maximales de la caractéristique bornée ou de sortie. Ces tables sont alors utilisées comme jeu de données pour l'acquisition de conjectures. Nous présentons maintenant la méthode de génération utilisée pour chaque objet combinatoire.

Génération du jeu de données pour les graphes orientés Dans le cadre du graphe orienté ou chaque sommet a au moins un arc entrant ou sortant, pour générer toutes les instances de taille v sommets, nous résolvons le problème de satisfaction de contraintes (2.9),...,(2.12), tout en sachant que la variable $X_{l,j}$ désigne le nombre de sommets de la j-ième composante fortement connexe de la l-ième composante connexe. Le symbole $\leq_{\text{LEX}}$ représente l'opérateur de comparaison lexicographique défini selon la définition 32.

Définition 32. Soient n et $a_i, b_i, \forall i \in [1; n]$ des entiers. Nous disons que le tuple $(a_1, \ldots, a_n)$ est lexicographiquement inférieur au tuple $(b_1, \ldots, b_n)$ si :

- soit nous avons $a_i = b_i$ pour tout rang i d'une composante d'un des tuples ;
- soit il existe un plus petit rang j d'une composante d'un des tuples tel que $a_j < b_j$ et pour $1 \leq i < j$, on a $a_i = b_i$.

Dans ce cas, on note alors que $(a_1, \ldots, a_n) \leq_{\text{LEX}} (b_1, \ldots, b_n)$. Formellement, l'opérateur d'ordre lexicographique $\leq_{\text{LEX}}$ (2.5) est défini par (2.6).

$$(a_1, \ldots, a_n) \leq_{\text{LEX}} (b_1, \ldots, b_n) \tag{2.5}$$

$$\Updownarrow$$

$$\big(\exists j \in [1; n+1], (\forall i \in [1; j[, a_i = b_i) \wedge (j = n+1 \vee a_j < b_j)\big) \tag{2.6}$$

Nous disons que le tuple $(a_1, \ldots, a_n)$ est lexicographiquement et strictement inférieur au tuple $(b_1, \ldots, b_n)$ s'il existe un plus petit rang j d'une composante d'un des tuples tel que $a_j < b_j$.

Dans ce cas, on note alors que $(a_1, \ldots, a_n) <_{\text{LEX}} (b_1, \ldots, b_n)$. Formellement, l'opérateur d'ordre lexicographique strict $<_{\text{LEX}}$ (2.7) est défini par (2.8).

$$(a_1, \ldots, a_n) <_{\text{LEX}} (b_1, \ldots, b_n) \tag{2.7}$$

$$\Updownarrow$$

$$\big(\exists j \in [1; n], (\forall i \in [1; j[, a_i = b_i) \wedge (a_j < b_j)\big) \tag{2.8}$$

$$\forall (l, j) \in [1; v] \times [1; v], X_{l,j} \in [0; v] \tag{2.9}$$

$$\sum_{l=1}^{v} \sum_{j=1}^{v} X_{l,j} = v \tag{2.10}$$

$$\forall l \in [1; v-1], (X_{l,1}, X_{l,2}, \ldots, X_{l,v}) \leq_{\text{LEX}} (X_{l+1,1}, X_{l+1,2}, \ldots, X_{l+1,v}) \tag{2.11}$$

$$\forall l \in [1; v], \forall j \in [1; v-1], X_{l,j} \leq X_{l,j+1} \tag{2.12}$$

Ce modèle considère un graphe orienté comme une partition de composantes connexes qui sont elles-mêmes partitionnées en composantes fortement connexes. Les contraintes (2.9) définissent le domaine des variables $X_{l,j}$. Les contraintes (2.10) s'assurent que la somme des tailles des partitions est le nombre de sommets i du graphe orienté. Les contraintes d'ordre lexicographique (2.11) cassent les symétries entre composantes connexes tandis que les contraintes (2.12) cassent les symétries entre composantes fortement connexes d'une même composante connexe. Donc chaque solution de ce modèle est une instance d'une partition d'un grahe orienté ayant i sommets, en composantes connexes qui sont à leurs tour partitionnées en composantes fortement connexes. Lorsque $X_{l,j}$ prend pour valeur zéro, la composante fortement connexe associée n'est pas considérée dans le calcul des valeurs des caractéristiques. Par exemple la taille de la plus petite composante connexe se calcule avec la formule

$$\underline{s} = \min\{X_{l,j} : X_{l,j} > 0, \forall (l, j) \in [1; i] \times [1; i]\}, \tag{2.13}$$

le nombre s de composantes fortement connexes se calcule avec la formule

$$s = \sum_{l=1}^{v} \left(\sum_{j=1}^{v} [X_{l,j} > 0] \right) \tag{2.14}$$

où $[X_{l,j} > 0]$ prend la valeur 0 ou 1 en fonction de la valeur de vérité de la condition $X_{l,j} > 0$ contenue dans les crochets de Iverson $[\cdot]$. Le nombre c de composantes connexes se calcule avec

la formule

$$c = \sum_{l=1}^{v} \left[\left(\sum_{j=1}^{v} X_{l,j} \right) > 0 \right]. \tag{2.15}$$

Ce modèle ne fournit pas le nombre d'arcs d'une instance quelconque du graphe orienté. Cependant, elle permet de calculer les valeurs minimum et maximum du nombre d'arcs dont nous avons besoin. En effet, le nombre maximum $\overline{a}$ d'arcs est

$$\overline{a} = \sum_{1 \leq l,j \leq v} X_{l,j}^2 + \sum_{l=1}^{v} \left(\sum_{1 \leq j < m \leq v} X_{l,j} \cdot X_{l,m} \right) \tag{2.16}$$

où le terme de droite représente la contribution de chaque composante fortement connexe et le terme de gauche est celui des arcs de connexion entre les composantes fortement connexes d'une même composante connexe. Le nombre minimum $\underline{a}$ d'arcs est

$$\underline{a} = \sum_{1 \leq l,j \leq v} \left((v = 1 \,?\, 1 : X_{l,j}) + \left(\sum_{j=1}^{v} [X_{l,j} > 0] \right) - 1 \right) \tag{2.17}$$

L'expression $(v = 1 \,?\, 1 : X_{l,j})$ est égale à 1 si $v = 1$ et $X_{l,j}$ sinon. La formule (2.17) vient du fait qu'un graphe orienté n'a qu'un seul arc s'il n'a qu'un seul sommet, sinon dans une composante connexe, le nombre minimum d'arcs d'une composante fortement connexe est le nombre de sommets de la composante connexe et il n'y a qu'un seul arc au minimum connectant deux composantes fortement connexes.

Comme l'illustre la partie (C) de la figure 2.5, la table des bornes numériques $\mathcal{T}^{a\leq}_{\{v,c\},3}$ fournie une borne supérieure précise de la caractéristique de sortie a en fonction des caractéristiques d'entrée v et c. Une table des bornes numériques peut aussi mentionner des caractéristiques secondaires comme, par exemple, $\overline{c}$ dans $\mathcal{T}^{a\leq}_{\{v,c\},3}$, qui est fonctionnellement déterminée par les caractéristiques d'entrée v et c. Chaque colonne de la table $\mathcal{T}^{a\leq}_{\{v,c\},3}$ fait référence à une caractéristique, c'est-à-dire v, c, a, $\overline{c}$, tandis que chaque ligne correspond à une combinaison des valeurs des caractéristiques v, c avec la valeur maximum associée du nombre d'arcs a et la valeur de la caractéristique secondaire $\overline{c}$.

Génération du jeu de données des partitions d'ensembles Dans le cadre des partitions d'ensembles, pour générer toutes les instances de taille n éléments, nous résolvons le problème de satisfaction de contraintes suivant, tout en sachant que la variable X_j est le nombre d'éléments du j-ème sous-ensemble du partitionnement :

$$\forall j \in [1; n], X_j \in [0; n] \tag{2.18}$$

$$\sum_{j=1}^{n} X_j = n \tag{2.19}$$

$$\forall j \in [1; n-1], X_j \leq X_{j+1} \tag{2.20}$$

Les contraintes (2.18) définissent les domaines des variables X_j. La contrainte (2.19) s'assure que la somme des tailles des partitions est le cardinal n de l'ensemble partitionné. Les contraintes (2.20) cassent les symétries, car l'ordre entre les tailles distinctes de deux sous-ensembles n'influence pas le calcul des caractéristiques. Ainsi, lorsque X_j prend pour valeur zéro, la j-ème partition n'est pas considérée et la caractéristique P, qui est le nombre de partitions ou de sous-ensembles, se calcule avec la formule

$$P = \sum_{j=1}^{n} [X_j > 0] \tag{2.21}$$

Les autres caractéristiques se déduisent directement de chaque solution de ce modèle. Par exemple, la taille minimale $\underline{M}$ d'une partition se calcule par

$$\underline{M} = \min\{X_j : X_j > 0, \forall j \in [1; n]\} \tag{2.22}$$

Dans le cas où l'on considère le partitionnement avec des sous-ensembles potentiellement vides, nous ajoutons au modèle précédent les contraintes

$$\sum_{j=1}^{n} [X_j > 0] \leq P_{max} \text{ et } P_{max} \leq n \tag{2.23}$$

où P_{max} est le nombre de parties ou sous-ensembles que l'on considère qu'ils soient vides ou pas parmi les n parties possibles.

Génération du jeu de données des séquences de 0/1 Dans le cadre des séquences de $0/1$, pour générer toutes les instances de taille n éléments, nous résolvons le problème de satisfaction de contraintes suivant, tout en sachant que la variable X_j de valeurs $0/1$:

$$\forall j \in [1; n], X_j \in [0; 1] \tag{2.24}$$

Pour éviter de générer des solutions symétriques, nous utilisons l'automate avec compteurs de la figure 2.6. La légende de la figure explique ce qu'est un automate à compteurs. En effet, deux solutions sont symétriques lorsqu'elles ont la même combinaison de valeurs des caractéristiques. Par exemple, les séquences 11100001100011 et 11100011000011 et 11000011000111 sont symétriques, car elles ont les mêmes valeurs des caractéristiques : la longueur de la séquence $n = 14$, la longueur du groupe le plus court $s_{min} = 2$, la longueur du groupe le plus long $s_{max} = 3$, l'interdistance la plus courte $d_{min} = 3$, l'interdistance la plus longue $d_{max} = 4$, le nombre de groupe de 1 $n_g = 3$. Ainsi, pour casser cette symétrie, l'automate n'accepte que les séquences ordonnées par taille décroissante des groupes de 1 et des groupes de 0 si ces derniers sont compris entre des groupes de 1. Ainsi, pour les trois séquences symétriques 11100001100011 et 11100011000011 et 11000011000111, l'automate n'accepte que la séquence 11100001100011. En effet, pour n'accepter que les séquences ordonnées par taille décroissante des groupes de 1, cet automate sauvegarde le nombre de 1 du premier groupe de la séquence

dans P_1, puis il lit les 1 du groupe courant tout en vérifiant que $P_1 > C_1$. Puis, à chaque fois que le test de $P_1 > C_1$ passe, il incrémente C_1 de 1 et accepte la séquence si c'était la dernière lecture. S'il reste encore des valeurs 0/1 à lire et qu'il lit un 0 à la prochaine lecture, alors il exécute $P_1 \leftarrow C_1$ pour sauvegarder dans P_1 la longueur C_1 du groupe courant de 1, afin de le comparer à la longueur du prochain groupe de 1. Si le test $P_1 > C_1$ ne passe pas, il rejette la séquence. Par le même procédé, l'automate n'accepte que des séquences ordonnées par taille décroissante des groupes de 0 compris entre des groupes de 1 et si ce n'est pas le cas, il accepte les séquences telles que 11001101000000 ou 110100 qui se terminent par un groupe de 0 de taille plus grande que les autres groupes de 0. Notons que l'automate n'accepte pas les séquences commençant par des 0 suivis des 1, car le groupe de 0 de début de la séquence n'est pas une interdistance. Par exemple la séquence 00110011000 n'est pas acceptée, car la sous-séquence 00 de début n'est pas comprise entre des groupes de 1. Donc la sous-séquence 00 de début n'est pas une interdistance et elle aurait dû être placée en fin de séquence.

Génération du jeu de données des séquences cycliques de 0/1 Pour générer les séquences cycliques de 0/1, nous utilisons le même automate de la figure 2.6 avec l'état c non acceptant pour casser les symétries sans perte de solutions. En effet, la séquence 11001100000000 que l'automate n'accepte pas dans le cas des séquences *non* cycliques de 0/1 précédentes, a les mêmes valeurs des caractéristiques que la séquence cyclique 11000000001100 ordonnée par taille décroissante de 0 puisque 11001100000000 est en fait sous la forme $\overset{1100}{0\,0}\overset{1}{0}\overset{0000}{0\,0}\overset{1}{0}$ Notons que l'état c n'est pas acceptant dans le cas des séquences cycliques de 0/1 afin de ne pas générer des séquences cycliques commençant par 1 et se terminant par 1 puisque ces deux 1 appartiennent au même groupe de 1. C'est-à-dire que la séquence cyclique 1001 est équivalente à la séquence cyclique 1100 acceptée par l'automate. Donc il est inutile que l'automate accepte la séquence cyclique 1001.

Génération du jeu de données des arbres et forêts enracinés Dans le cadre des arbres enracinés et forêts enracinées, pour générer toutes les instances de taille v sommets, nous utilisons l'algorithme dédié de Knuth [45] à la page 462. Pour exécuter cet algorithme, l'arbre de sommets numérotés est encodé sous forme d'une suite de $v-1$ numéros où le r-ème numéro est le nœud père du sommet r. Puis cet encodage est donné en entrée à algorithme qui effectue des transformations de l'arbre initial en tous les autres arbres possibles. Le même algorithme avec une légère modification est utilisé pour générer les forêts puisqu'un arbre enraciné de plus d'un sommet devient une forêt lorsqu'on lui supprime sa racine. De plus, cet encodage d'une instance sous la forme d'une séquence de pointeurs des nœuds pères nous permet de calculer toutes les caractéristiques d'intérêts en temps linéaire en fonction du nombre de sommets.

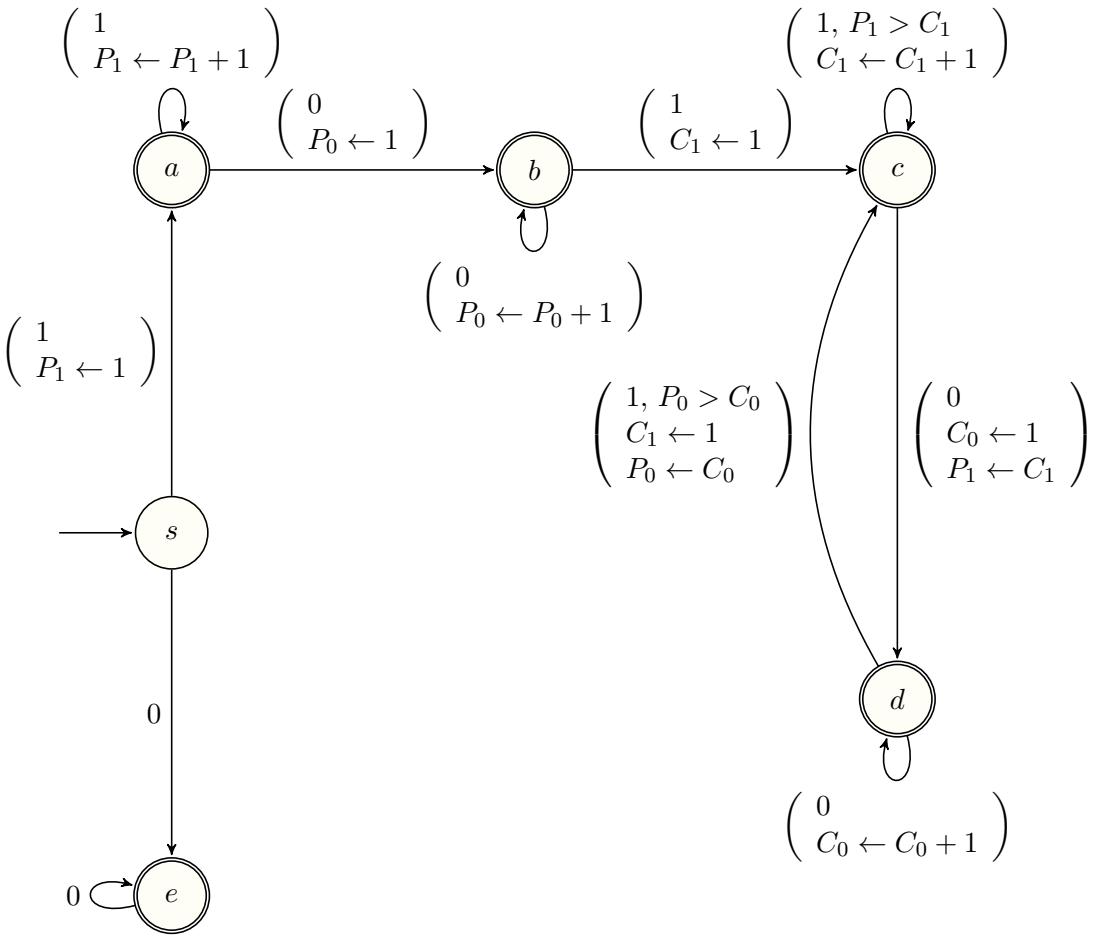


FIGURE 2.6 – Automate avec 4 compteurs P_0, P_1, C_0, C_1 utilisé pour générer des séquences de 0/1 par ordre décroissant des groupes de 1 et des groupes de 0. Un automate à compteurs est un automate disposant de variables appelées compteurs dont les valeurs sont modifiées en fonction des transitions. Dans un automate à compteurs, une transition peut engendrer le changement de la valeur d'un compteur. Par exemple, l'étiquette $\begin{pmatrix} 1 \\ P_1 \leftarrow P_1 + 1 \end{pmatrix}$ de la transition de l'état a vers lui-même signifie que la lecture de la valeur 1 d'une séquence commençant par 1 engendre l'incrémentation de 1 du compteur P_1. L'étiquette $\begin{pmatrix} 1 \\ C_1 \leftarrow 1 \end{pmatrix}$ de la transition de l'état b à l'état c signifie que le compteur C_1 est réinitialisé à 1 après lecture de la valeur 1 précédée d'une lecture de la valeur 0. Une transition peut être aussi conditionnée dans un automate à compteurs. Par exemple, l'étiquette $\begin{pmatrix} 1, P_1 > C_1 \\ C_1 \leftarrow C_1 + 1 \end{pmatrix}$ de la transition de l'état c vers lui-même après la lecture de 1, signifie que cette transition est effectuée si et seulement si $P_1 > C_1$ et dans ce cas, le compteur C_1 est incrémenté de 1.

2.3.3 Phase (A2) : Génération des métadonnées

Pour chaque table de bornes numériques $\mathcal{T}^{o\leq}_{P_b,i}$ (avec $P_b \subseteq \mathcal{P}$ et $i \in [2, n_{max}]$), avec *nrows* lignes, où $\mathcal{T}^{o\leq}_{P_b,i}[r, j]$ est la valeur de la r-ème ligne et la j-ème colonne, la phase (A2) calcule l'information agrégée $\mathcal{D}^{o\leq}_{P_b,i}$ (avec $P_b \subseteq \mathcal{P}$ et $i \in [2, n_{max}]$) utilisée pour sélectionner la taille

k appropriée pour l'acquisition de conjectures de la caractéristique de sortie o en fonction des caractéristiques d'entrée P_b. Ces informations sont exploitées pour cibler la recherche des conjectures : premièrement en sélectionnant les sous-ensembles de caractéristiques bornantes les plus susceptibles d'exprimer une formule de conjectures et deuxièmement en fournissant des informations qui permettent d'éviter de générer des formules simplifiables ou peu pertinentes. Par exemple, il est inutile d'utiliser dans une formule le terme $\min(v, c)$ du moment que l'on a la propriété $v \geq c$. Ces informations sont notamment :

— la valeur minimum ou maximum de chaque colonne et son nombre de valeurs distinctes ;

— les dépendances fonctionnelles minimales. Comme le stipule l'article [59], elles sont très sollicitées dans l'analyse des données afin d'en extraire des informations utiles. Dans notre cas, il s'agit des plus petits sous-ensembles de caractéristiques bornantes qui déterminent dans la table $\mathcal{T}^{o\leq}_{P_b,k}$ la caractéristique de sortie ou les caractéristiques secondaires. Par exemple, dans la table 2.2 des bornes numériques $\mathcal{T}^{a\leq}_{\{v,c\},3}$, les colonnes a et $\overline{c}$ sont fonctionnellement déterminées par les colonnes v et c. Mais la colonne a est aussi fonctionnellement déterminée par les colonnes v et $\overline{c}$.

— des contraintes binaires entre deux colonnes distinctes i et j de la table $\mathcal{T}^{o\leq}_{P_b,k}$, c'est-à-dire des contraintes de la forme $\forall r \in [1, nrows], \mathcal{T}^{o\leq}_{P_b,k}[r, i]$ *op* $\mathcal{T}^{o\leq}_{P_b,k}[r, j]$ (avec *op* $\in \{\leq, <, >, \geq\}$). Dans $\mathcal{T}^{a\leq}_{\{v,c\},3}$ nous avons pour chaque ligne la contrainte selon laquelle le nombre de sommets est supérieur ou égal au nombre de composantes connexes. C'est-à-dire $v \geq c$ et nous avons également les contraintes binaires $v \geq \overline{c}$, $a \geq v$, $a \geq c$ et $a \geq \overline{c}$.

Combinaison des valeurs des caractéristiques		Bornes numériques de a	Caractéristique secondaire	Graphes orientés correspondants
v	c	a	$\overline{c}$	$\mathcal{G}$
1	1	1	1	K_1
2	1	4	2	K_2
2	2	2	1	K_1, K_1
3	1	9	3	K_3
3	2	5	2	K_2, K_1
3	3	3	1	K_1, K_1, K_1

TABLE 2.2 – Table $\mathcal{T}^{a\leq}_{\{v,c\},3}$ des bornes numériques supérieures de la *caractéristique d'entrée* qu'est le nombre d'arcs a d'un graphe orienté d'au plus trois sommets en fonction des *caractéristiques d'entrée* v, c et de la *caractéristique secondaire* $\overline{c}$ qui représente le nombre de sommets de la plus grande composante connexe. Pour chaque ligne de la table, il y a une instance de graphe orienté qui possède les valeurs des caractéristiques se trouvant sur ladite ligne.

Ces informations agrégées constituent ce que nous appelons métadonnées. Elles servent à la phase suivante.

Formules candidates générées par la phase (B1)	Formules découvertes par la phase (B2)
polynôme de degré 1 de paramètres v et c déterminant $\overline{c}$	$\overline{c} = v - c + 1$
polynôme de degré 1 de paramètres v et c déterminant a	aucun
polynôme de degré 2 de paramètres v et c déterminant a	$a = c^2 - 2 \cdot v \cdot c + v^2 - c + 2 \cdot v$
polynôme de degré 1 de paramètres v et $\overline{c}$ déterminant a	aucun
polynôme de degré 2 de paramètres v et $\overline{c}$ déterminant a	$a = \overline{c}^2 - \overline{c} + v$

TABLE 2.3 – Exemples de formules candidates et de formules correspondantes générées pour la table $\mathcal{T}^{a\leq}_{\{v,c\},3}$ de la partie (C1) de la figure 2.5.

2.3.4 Phase (A3) : Génération des méta-métadonnées afin de déterminer la taille des données qui est pertinente pour l'acquisition de conjectures

En utilisant les métadonnées générées à la phase (A2), la phase (A3) détermine pour le sous-ensemble de caractéristiques bornantes P_b et la caractéristique de sortie o, la taille k appropriée pour apprendre des conjectures. Pour sélectionner la bonne taille k du jeu de données $\mathcal{T}^{o\leq}_{P_b,i}$ (avec $i \in [2, n_{max}]$) à partir duquel nous apprenons les conjectures, nous opérons comme suit. Sachant qu'une dépendance fonctionnelle ou une contrainte binaire d'une table $\mathcal{T}^{o\leq}_{P_b,i}$ peuvent devenir invalides pour la table $\mathcal{T}^{o\leq}_{P_b,j}$ avec $j > i$, nous identifions la plus petite taille k à partir de laquelle l'ensemble des dépendances fonctionnelles minimales et l'ensemble des contraintes binaires des tables $\mathcal{T}^{o\leq}_{P_b,k}, \ldots, \mathcal{T}^{o\leq}_{P_b,n_{max}}$ restent identiques. En pratique, pour des raisons d'espace mémoire, nous avons généré des instances d'une taille maximale de $n_{max} = 26$ sommets pour les graphes orientés, $n_{max} = 22$ sommets pour les arbres et forêts enracinés, $n_{max} = 30$ éléments pour les partitions d'ensembles et les séquences d'entiers 0/1. Pour éviter d'obtenir des conjectures invalides lorsque le nombre de lignes ou d'instances d'une table $\mathcal{T}^{o\leq}_{P_b,k}$ est trop bas, nous sélectionnons la plus petite taille dont la table a au moins 200 lignes ou instances. Les conjectures dans le cadre des graphes orientés étaient produites avec $k = 18$.

2.3.5 Phase (B1) : Génération des formules candidates

Cette phase génère pour une table des bornes numériques $\mathcal{T}^{o\leq}_{P_b,k}$, des formules candidates dont les coefficients ne sont pas encore fixés afin de produire les correspondantes conjectures maximum et de maximalité. Étant donné les caractéristiques d'entrée P_b, les caractéristiques de sortie o, l'ensemble des caractéristiques secondaires de la table sélectionnée $\mathcal{T}^{o\leq}_{P_b,k}$, la phase (B1) produit sous demande la prochaine formule candidate à utiliser pour découvrir une conjecture en considérant les formules candidates par complexité croissante. L'ensemble des caractéristiques potentielles qu'une formule peut mentionner et la formule elle-même sont restreints par les dépendances fonctionnelles et les contraintes binaires qui ont été identifiées par la phase de génération des métadonnées. La table 2.3 donne des formules candidates qui sont successivement produites pour la table des bornes numériques $\mathcal{T}^{a\leq}_{\{v,c\},3}$.

2.3.6 Phase (B2) : Génération d'un modèle de programmation par contraintes pour lier une formule candidate avec les données des tables des bornes numériques

Cette phase utilise une formule candidate générée à la phase (B1) pour poser une contrainte d'égalité pour chaque ligne de la table $\mathcal{T}^{o\leq}_{P_b,k}$ afin d'obtenir une formule où tous les paramètres d'entrée sont choisis et les coefficients sont fixés. L'exemple 17 présente une contrainte d'égalité posée pour la cinquième ligne de la table 2.2 dénommée $\mathcal{T}^{a\leq}_{\{v,c\},3}$. La phase (B2) fait appel à la phase (B1) pour la prochaine formule candidate paramétrée, tente de l'instancier et demande à nouveau la prochaine formule candidate. Pour trouver une valeur pour chaque coefficient d'une formule candidate, nous utilisons un modèle à contraintes afin de lier une formule candidate :

— aux dépendances fonctionnelles et aux contraintes binaires identifiées à la phase de génération des métadonnées ;

— à toutes les entrées de la table des bornes numériques ayant la taille k sélectionnée.

De nombreuses contraintes sont employées pour casser différents types de symétries et pour forcer l'utilisation de sous-termes interprétables dans une formule. La deuxième colonne de la table 2.3 montre, pour chaque formule candidate, la formule concrète correspondante trouvée par le modèle de programmation par contraintes.

Exemple 17. Soit la formule candidate générée par la phase (B1), à la dernière ligne de la table 2.3. C'est-à-dire la formule du polynôme de second degré de paramètres v et $\overline{c}$ qui déterminent a. Cela signifie que le système émet l'hypothèse que a est un polynôme qui s'écrit sous la forme (2.25).

$$a = Coef_1 \cdot v^2 + Coef_2 \cdot \overline{c}^2 + Coef_3 \cdot v \cdot \overline{c} + Coef_4 \cdot v + Coef_5 \cdot \overline{c} + Coef_6 \tag{2.25}$$

avec les variables $Coef_i, \forall i \in [1;6]$ qui représentent les coefficients du polynôme. Pour trouver les valeurs fixées de ces coefficients, nous posons pour la cinquième ligne de la table 2.2 des données, la contrainte (2.26). En effet, comme à la cinquième ligne de cette table $\mathcal{T}^{a\leq}_{\{v,c\},3}$, nous avons les valeurs des paramètres $v = 3, a = 5$ et $\overline{c} = 2$, nous substituons dans la formule (2.25) les paramètres v, a et $\overline{c}$ par ces valeurs pour obtenir la contrainte (2.26).

$$5 = 9 \cdot Coef_1 + 4 \cdot Coef_2 + 6 \cdot Coef_3 + 3 \cdot Coef_4 + 2 \cdot Coef_5 + Coef_6 \tag{2.26}$$

Le Bound Seeker va donc résoudre cette contrainte (2.26) en déterminant les valeurs fixées des coefficients $Coef_i, \forall i \in [1;6]$, pour obtenir la formule $a = \overline{c}^2 - \overline{c} + v$ de la table 2.2. Et comme cette table $\mathcal{T}^{a\leq}_{\{v,c\},3}$ est une table des bornes supérieures numériques du nombre d'arcs a d'un graphe orienté, la conjecture $a \leq \overline{c}^2 - \overline{c} + v$ est déduite.

2.3.7 Phase (B3) : Test des conjectures candidates

Cette dernière phase teste la validité des conjectures trouvées sur la table de bornes numériques $\mathcal{T}^{o\leq}_{P_b,n_{max}}$ ayant la plus grande taille que le jeu de données possède.

2.3.8 Une approche par la programmation par contraintes pour acquérir des équations symboliques

La recherche de bornes précises conduit à l'identification d'équations dans lesquelles le côté gauche de l'égalité correspond à une caractéristique de sortie ou une caractéristique secondaire et le côté droit correspond à une formule impliquant les caractéristiques de sorties et les caractéristiques secondaires. Tel que déjà mentionné dans l'introduction de Brence et *al.* [20] et dans la conclusion de Kolb et *al.* [46], l'espace des formules candidates constitue un défi majeur dans le domaine de la recherche d'équations régissant les phénomènes d'intérêts. Au lieu d'appliquer une approche de génération de formules par ordre croissant du nombre d'opérateurs et d'opérandes, nous adoptons la stratégie qui mesure la complexité d'une formule par son degré d'interprétabilité. Comme notre objectif est de trouver des formules simples, nous utilisons trois classes complémentaires de formules dont nous avons constaté leurs apparitions fréquentes et simultanées dans les cartes :

- les formules booléennes impliquant plusieurs conditions arithmétiques liées par des connecteurs logiques ou par une somme de leur valeur de vérité. Nous les considérons comme les formules les moins complexes ;
- les formules conditionnelles dont la condition est une simple comparaison. Nous les considérons comme plus complexes que les formules booléennes ;
- les formules impliquant des polynômes. Nous les considérons comme plus complexes que les formules conditionnelles. Ces polynômes ont des variables qui peuvent être des sous-expressions contenant des opérateurs. En effet, ne pas utiliser ces sous-expressions conduit le plus souvent, soit à des formules trop complexes, soit à ne trouver aucune formule. Par exemple, comme l'illustre la figure 2.4 de la page 49, la borne supérieure précise de taille $\overline{c}$ de la plus grande composante connexe en fonction du nombre de sommets v et du nombre de composantes connexes c trouvées par le Bound Seeker est $\overline{c} \leq r + \underline{c}$ avec $r = v \cdot [c \geq 2] - c \cdot [c \geq 2]$ et $\underline{c} = \max(2 \cdot v - v \cdot c, 1)$. Les sous-expressions dans cet exemple sont $\underline{c}$ et r qui lui-même contient la sous-expression $[c \geq 2]$. L'exemple 15 de la page 47 illustre aussi une formule composée de sous-expressions.

En se basant sur les métadonnées introduites à la section 2.3.3, nous décrivons une approche en programmation par contraintes pour restreindre l'espace des formules possibles. Nous présentons précisément l'espace des polynômes et celui des conditionnelles. Bien que nous l'utilisions pour comparer nos résultats, l'acquisition de formules booléennes réalisée par Gindullin et *al.* [36] ne fait pas partie des contributions de nos travaux.

Un générateur de formules candidates paramétrées de la phase (B1)

Syntaxe des formules générées Toutes les formules que nous générons ont la forme *caracteristique op formule*, où *op* est l'un des opérateurs de comparaison $\leq$, $=$, $\geq$, et *formule*

est une formule impliquant un ensemble de caractéristiques. Ainsi, les formules sont décrites par l'ensemble de règles d'une grammaire simplifiée où une "Petite Lettre Majuscule" indique un symbole non terminal, le style "roman" dénote une fonction ou une constante connue, le style "*italique*" souligne des caractéristiques, le style "**gras**" dénote une constante entière inconnue. À l'intérieur de ces règles, polynome(PARAMS, degré) dénote un polynôme dont le degré maximum est fixé (avec degré > 0) sur un sous-ensemble non vide de paramètres de ses paramètres potentiels PARAMS. Ce polynôme peut contenir les termes unaires comme $\text{geq0}(x) = (x \geq 0\ ?\ 1 : 0)$ et $\text{sum\_consec}(x) = \frac{x\cdot(x+1)}{2}$. Ce polynôme peut aussi contenir des termes binaires comme $\text{geq}(x, y) = (x \geq y\ ?\ 1 : 0), \text{cmod}(x, y) = x - (y \bmod x)$ ou encore $\text{dmod}(x, y) = x - (x \bmod y)$.

FORMULE ::= **cst** | BOOL | **cst** + BOOL | COND | POL | POLBINAIRE | POLUNAIRE

BOOL ::= BOOLOP(BOOLCONDS) BOOLOP ::= $\wedge | \vee | = | +$

BOOLCONDS ::= BOOLCOND, BOOLCONDS | BOOLCOND

BOOLCOND ::= PARAM CMP **cst** CMP ::= $\leq | = | \geq | \neq$

COND ::= (BOOLCOND ? PARAMCST : PARAMCST) PARAMCST ::= PARAM|**cst**

POL ::= polynome(PARAMS, degré)

POLBINAIRE ::= BF(POL, POL) BF ::= min | max | floor | mod | cmod | dmod | prod

POLUNAIRE ::= UF1(POL) | UF2(POL, **cst**)

UF1 ::= geq0 | sum_consec UF2 ::= min | max | floor | mod | power

PARAMS ::= PARAM* PARAM ::= CHAR|BTERME|UTERME CHAR ::= $v|c|\underline{c}|\overline{c}|s|\underline{s}|\overline{s}$

BTERME ::= BT(CHAR, CHAR)

UTERME ::= sum_consec(CHAR) | UT(CHAR, **cst**) | CHAR $\in$ [**cst**, **cst**]

BT ::= min | max | floor | ceil | mod | cmod | dmod | prod

UT ::= min | max | floor | ceil | mod | geq | power

Exemple 18 (Exemples de formules booléennes, conditionnelles, polynomiales générées)**.**

- $(\overline{s} = 1) \wedge (\overline{c} \in [2, 3])$ et $(v = \underline{c}) = (c = 1)$, où la seconde formule dénote une condition qui est satisfaite uniquement si les conditions $(v = \underline{c})$ et $(c = 1)$ sont à la fois les deux vraies, ou les deux fausses.
- $(\underline{s} = 1\ ?\ \lceil \frac{v}{2} \rceil : v)$ et $((\overline{c} - \underline{c}) = 0\ ?\ \underline{c} : \underline{c} + \overline{c})$, où $(cond\,?\,x : y)$ dénote x si la condition $cond$ est satisfaite, y sinon.
- $(v \bmod \overline{c})^2 - \overline{c}\cdot(v \bmod \overline{c}) + v \cdot \overline{c}$ et $\lfloor \frac{[\overline{s} \geq 2] + \overline{s} + v}{2} \rfloor$ où $v \bmod \overline{c}$ est un terme binaire BTERME partagé entre deux monômes du polynôme associé et $[\overline{s} \geq 2]$ est un terme unaire UTERME de la forme $\text{geq}(\overline{s}, 2)$.

Exemple 19 (Trouver des bornes plus interprétables par utilisation des formules booléennes et conditionnelles)**.** Nous montrons avec un exemple généré par le système pour la borne inférieure du nombre d'arcs a en fonction des tailles de la plus petite composante connexe $\underline{c}$, de la plus grande composante connexe $\overline{c}$, et de la taille $\overline{s}$ de la plus grande composante fortement connexe, comment l'utilisation des formules booléennes et conditionnelles conduit souvent

à des conjectures plus interprétables. Sans l'utilisation des booléens et conditionnelles, nous obtenons $a \geq s_{>1} - c_{>1} + v$ avec $s_{>1} = \min(\overline{s} - 1, 1)$, $c_{>1} = \min(\min(\underline{c}, 2), \min(\underline{c}, 2) + \overline{c} - \underline{c} - 1)$, et $v = \min(\overline{c} + \underline{c}, \underline{c} \cdot \overline{c} - \underline{c}^2 + \overline{c})$. Cette borne à première vue fournit peu d'informations sur la structure d'un graphe orienté dont le nombre d'arcs est minimal. Cependant, en utilisant les booléens et les conditionnelles, nous obtenons la borne suivante : $a \geq s_{>1} - c_{>1} + v$ avec $s_{>1} = (\overline{s} \geq 2)$, $c_{>1} = (\underline{c} \geq 2) + ((\overline{c} - \underline{c}) \geq 1)$, et $v = ((\overline{c} - \underline{c}) = 0\,?\,\underline{c} : \underline{c} + \overline{c})$. Cette borne est beaucoup plus interprétable que la borne précédente. En effet, elle nous indique clairement, par exemple, que $a \geq \underline{c}$ si $\overline{s} \geq 2, \underline{c} \geq 2$ et $\overline{c} = \underline{c} = v$. C'est-à-dire que, pour obtenir un nombre minimal $a = \underline{c}$ d'arcs, il suffit que le graphe n'ait qu'une seule composante connexe et que la plus grande composante fortement connexe ait au moins 2 sommets. Il s'agit d'un graphe de la forme  .

Le générateur de formules candidates Puisque nous voulons tester une grande variété de formules, nous avons élaboré un générateur de formules candidates qui, par retour arrière, propose une nouvelle formule candidate avec des coefficients non fixés. Ces coefficients sont les variables représentant les constantes et les caractéristiques d'entrées qui seront utilisées dans la formule candidate. Dans ce générateur, nous spécifions :

- la structure de la formule qui est utilisée, c'est-à-dire une formule booléenne, une formule conditionnelle simple ou une formule polynomiale. Dans le cas des polynômes nous spécifions aussi le nombre de sous-termes unaires ou binaires qui peuvent apparaître dans chaque polynôme ;
- le type de fonctions arithmétiques qui peut être utilisé dans les termes de la formule ;
- la complexité d'un polynôme, à savoir son degré maximum potentiel, son nombre maximum de coefficients non nuls, et le domaine des valeurs de ses coefficients ;
- la liste des possibles combinaisons des caractéristiques bornantes que la formule candidate peut utiliser dans ces paramètres. Une telle combinaison correspond aux dépendances fonctionnelles identifiées dans la phase de génération des métadonnées, c'est-à-dire la phase (A2).

Le générateur cible une catégorie de formules candidates à partir desquelles le Bound Seeker va essayer d'instancier une conjecture. C'est ainsi que nous distinguons un générateur de formules booléennes, un générateur de formules conditionnelles et plusieurs générateurs de formules polynomiales. Les générateurs de formules polynomiales se distinguent par le nombre maximal de monômes constituant la formule candidate. Chaque générateur de polynômes propose des polynômes dans un ordre croissant du degré. Puis, chacun de ces générateurs propose en dernier des polynômes contenant des termes unaires ou binaires. Donc le système, dans sa stratégie de recherche, commence par chercher des formules simples et interprétables avant d'aller vers des formules plus complexes s'il y a lieu. Donc, le Bound Seeker va essayer en premier avec le générateur des formules booléennes présenté dans l'article [36], puis celui

des formules conditionnelles et termine avec les générateurs des polynômes en commençant par ceux qui génèrent des polynômes ayant peu de monômes et un petit degré.

Modèle à contraintes pour l'acquisition d'une conjecture de formule polynomiale de la phase (B2)

Étant donnée une formule candidate $\mathcal{F}$, (correspondant soit à Pol, PolBinaire, Cond ou PolUnaire décrite dans l'ensemble des règles de la grammaire décrite plus haut), pour laquelle l'ensemble des paramètres sont partiellement déterminés, et pour laquelle les coefficients ne sont pas encore fixés, nous produisons un modèle de contraintes qui relie les différentes inconnues à chaque ligne d'une table des bornes numériques. Notre modèle utilise quatre types de contraintes :

- les contraintes sur la structure de la formule impliquant les caractéristiques d'entrée et les caractéristiques secondaires ;
- les contraintes pour casser les symétries au sein des termes de la formule ;
- les contraintes pour prévenir la production de formules simplifiables ;
- les contraintes d'égalité sur chaque ligne de la table des bornes numériques d'une caractéristique de sortie.

Par la suite, nous décrivons les variables du modèle, les contraintes sur les caractéristiques utilisées dans $\mathcal{F}$, les contraintes sur les termes unaires, binaires, puis sur les fonctions binaires de $\mathcal{F}$, ainsi que les contraintes d'égalités sur chaque ligne de la table des bornes numériques. Le nombre de variables et contraintes du modèle est linéaire par rapport aux nombres de lignes de la table des bornes numériques, car le modèle est largement dominé par les contraintes d'égalité associées à chaque ligne. Pour les contraintes de cassage de symétries et les contraintes d'égalités, nous détaillons uniquement les contraintes impliquant la fonction arithmétique $\min(\cdot,\cdot)$. Les contraintes générées pour les autres fonctions arithmétiques utilisées sont similaires.

Les variables du modèle La table 2.4 introduit les variables utilisées pour représenter une formule $\mathcal{F}$ qui peut être :

1. soit une fonction binaire ou unaire dont les arguments sont des polynômes non constants $\mathcal{P}_i, i \in \{1;2\}$. Chaque polynôme peut contenir n_c caractéristiques $\mathcal{C}_j, j \in [1;n_c]$, n_u termes unaires $\mathcal{U}_i$ ou n_b termes binaires $\mathcal{B}_i$. Ainsi, les variables booléennes $C_j, U_{i,j}, B_{i,j}$ et $P_{i,j}$ permettent de fixer dans le polynôme $\mathcal{P}_i$ respectivement le nombre de caractéristiques, de termes unaires et de termes binaires :

   - Pour que le terme unaire $\min(\mathcal{C}_j, U\_CST_i)$ soit utilisé dans le polynôme $\mathcal{P}_i$ avec U_CST_i comme variable représentant la valeur d'une constante, il faut qu'il ait dans la table des données au moins deux instances telles que pour l'une nous avons une valeur de $\mathcal{C}_j$ strictement inférieure à la valeur de la variable U_CST_i, et

Composants d'une formule	Variables	Description
Caractéristiques $\mathcal{C}_j$ ($j \in [1, n_c]$)	$C_j \in \{0,1\}$	$C_j = 1$ ssi $\mathcal{C}_j$ est utilisé par la formule $\mathcal{F}$
Terme unaire $\mathcal{U}_i$ ($i \in [1, n_u]$)	$U_{i,j} \in \{0,1\}$	$U_{i,j} = 1$ ssi $\mathcal{C}_j$ est utilisé par $\mathcal{U}_i$, $j \in [1, n_c]$
	$U\_IND_i \in [1, n_c]$	Indices des caractéristiques utilisées
	U_Min_i	Valeur minimum des caractéristiques utilisées
	U_Max_i	Valeur maximum des caractéristiques utilisées
	U_CST_i	Constante utilisée dans $\mathcal{U}_i$
($r \in [1, nrows]$)	$U\_VAL_{i,r}$	Valeur du terme $\mathcal{U}_i$ à la r-ème ligne et la j-ème colonne (avec $U_{i,j} = 1$) de la table $\mathcal{T}$
Terme binaire $\mathcal{B}_i$ ($i \in [1, n_b]$)	$B_{i,j} \in \{0,1\}$	$B_{i,j} = 1$ ssi $\mathcal{C}_j$ est utilisé par $\mathcal{B}_i$, $j \in [1, n_c]$
	$B\_IND1_i \in [1, n_c]$	Indice de la première caractéristique utilisée
	$B\_IND2_i \in [1, n_c]$	Indice de la seconde caractéristique utilisée
	$B\_O_i \in \{0,1\}$	Ordre des caractéristiques utilisées comme arguments
($r \in [1, nrows]$)	$B\_VAL_{i,r}$	Valeur du terme $\mathcal{B}_i$ en fonction de la r-ème ligne, de la B_IND1_i-ème, et de la B_IND2_i-ème colonne de la table $\mathcal{T}$
Polynôme $\mathcal{P}_i$ de degré d_i	$P_{i,j} \in \{0,1\}$ avec $j \in [1, n]$ et $n = n_c + n_u + n_b$	$\begin{cases} P_{i,j} = 1, j \in [1, n_c] & \Rightarrow \mathcal{C}_j \quad \text{utilisé par } \mathcal{P}_i \\ P_{i,j} = 1, j \in [n_c + 1, n_{cu}] & \Rightarrow \mathcal{U}_{j-n_c} \quad \text{utilisé par } \mathcal{P}_i \\ P_{i,j} = 1, j \in [n_{cu} + 1, n] & \Rightarrow \mathcal{B}_{j-n_c-n_u} \text{utilisé par } \mathcal{P}_i \end{cases}$
($i \in [1, n_p]$)	$M_{i,k}$ ($k \in [1, \binom{n+d_i}{d_i}]$)	$M_{i,k}$ est le k-ème coefficient de $\mathcal{P}_i$. Le coefficient ayant la plus grande valeur de k est la constante.
($r \in [1, nrows]$)	$P\_VAL_{i,r}$	Valeur du polynôme $\mathcal{P}_i$ en fonction de la r-ème ligne de la table $\mathcal{T}$
Conditionnelle ($\mathcal{C}ond\,?\,\mathcal{T}hen : \mathcal{E}lse$) où $\mathcal{C}ond$ est $\mathcal{C}_{j_1} = \mathcal{C}_{j_2}$ ($r \in [1, nrows]$)	$C\_IND1 \in [1, n_c]$	Indice j_1 de la première caractéristique utilisée
	$C\_IND2 \in [1, n_c]$	Indice j_2 de la seconde caractéristique utilisée
	C_VAL1_r	Valeur de $\mathcal{C}_{j_1}$ à la r-ème ligne de la table $\mathcal{T}$
	C_VAL2_r	Valeur de $\mathcal{C}_{j_2}$ à la r-ème ligne de la table $\mathcal{T}$
	$V_{j,r}$	Valeur de $\mathcal{C}_j$ à la r-ème ligne de la table $\mathcal{T}$
	$B\_VAL_r \in \{0,1\}$	Variable booléenne à la r-ème ligne de la table $\mathcal{T}$
	T_VAL_r	Valeur de $\mathcal{T}hen$ à la r-ème ligne de la table $\mathcal{T}$
	E_VAL_r	Valeur de $\mathcal{E}lse$ à la r-ème ligne de la table $\mathcal{T}$

TABLE 2.4 – Exemples de variables utilisées dans le modèle à contraintes du Bound Seeker, où $n_{cu} = n_c + n_u$.

pour l'autre instance une valeur de $\mathcal{C}_j$ strictement supérieure à la valeur de la même variable U_CST_i. Ceci évite que $\min(\mathcal{C}_j, U\_CST_i)$ se simplifie en $\mathcal{C}_j$ ou en U_CST_i. Pour s'en assurer le système sollicite les valeurs minimales U_Min_i et maximales U_Max_i de la caractéristique $\mathcal{C}_j$ avec $j = U\_IND_i$ à travers les contraintes (1b), (2b), (3b) et (4b) de la table 2.5. Ces valeurs sont fournies par les

métadonnées générées par la phase (A2).

— Pour utiliser un terme binaire $\mathcal{B}_i$ dans le polynôme $\mathcal{P}_i$, le Bound Seeker vérifie si $\mathcal{B}_i$ est commutatif et fixe la variable B_O_i à 0 si c'est le cas. Dans ce cas, l'ordre des arguments $\mathcal{C}_{j_1}$ et $\mathcal{C}_{j_2}$ n'est pas pertinent et le Bound Seeker peut alors ordonner ces arguments afin d'éviter de générer un terme binaire équivalent à une permutation des arguments près. Ceci est assuré par les variables d'indice $B\_IND1_i = j_1$ et $B\_IND2_i = j_2$ dans les contraintes (1c), (2c), (3c) de table 2.5.

2. soit une conditionnelle de la forme ($\mathcal{C}ond$? $\mathcal{T}hen$: $\mathcal{E}lse$). Pour exprimer cette conditionnelle dans le cas où la condition $\mathcal{C}ond$ correspond à l'égalité $\mathcal{C}_{j_1} = \mathcal{C}_{j_2}$, le Bound Seeker utilise les variables d'indice $C\_IND1 = j_1$ et $C\_IND2 = j_2$. Elles déterminent respectivement les caractéristiques $\mathcal{C}_{j_1}$ et $\mathcal{C}_{j_2}$ qui seront utilisées pour exprimer la condition $\mathcal{C}ond$. Cependant, pour éviter que la conditionnelle se simplifie, il faut s'assurer qu'il ait au moins deux entrées de la table des données $\mathcal{T}$ où la condition $\mathcal{C}ond$ est fausse pour l'une et vraie pour l'autre. C'est là qu'interviennent les contraintes de comparaisons générées par les métadonnées de la phase (A2). La table 2.6 présente les contraintes correspondantes à la description précédente de la structure de la formule $\mathcal{F}$ comme conditionnelle.

Contraintes sur la structure de la formule la partie supérieure de la table 2.5 liste les contraintes :

— spécifiant quelles caractéristiques la formule $\mathcal{F}$ utilise (voir (1a)) ;

— forçant un terme unaire, binaire et un polynôme à utiliser le nombre approprié de caractéristiques (voir (2a), (3a) et (4a)) ;

— connectant les caractéristiques utilisées dans les polynômes et dans la formule (voir (5a), (6a)) ;

— restreignant les coefficients non nuls des polynômes (voir (7a), (8a)).

Contraintes sur les termes unaires/binaires et sur les fonctions binaires À l'intérieur de la table 2.5, les contraintes (1b) (respectivement (1c), (2c)), lient les variables 0/1 $U_{i,j}$ (respectivement $B_{i,j}$) à l'indice de la caractéristique utilisée par le terme. Pour éviter de générer des termes unaires de la forme min(*Caractéristique*, *Cst*) qui peut se simplifier à *Caractéristique* ou à *Cst*, la contrainte (4b) restreint les valeurs minimum et maximum de la constante. Lorsque la fonction min est utilisée dans un terme binaire, la contrainte (4c) permet d'éviter de générer des termes binaires équivalents dont les arguments ont été permutés. La contrainte (5c) prévient la génération d'un terme binaire quand $\min(\cdot,\cdot)$ peut être simplifiée. Par exemple, elle évite de générer $\min(\underline{c}, \overline{c})$, car d'après les métadonnées générées à la phase (A2), $\underline{c}$ est toujours plus petit ou égal à $\overline{c}$. Finalement, quand la formule candidate $\mathcal{F}$ est une fonction binaire min, qui utilise comme arguments les polynômes $\mathcal{P}_1$ et $\mathcal{P}_2$ de degré

	Contraintes	Description
(1a)	$\text{TABLE}(\langle C_1, \ldots, C_{n_c} \rangle, \text{FD\_TABLE})$	Restreint la caractéristique utilisée dans $\mathcal{F}$
(2a)	$\forall i \in [1, n_u] : \sum_{j=1}^{j=n_c} U_{i,j} = 1$	$\mathcal{U}_i$ utilise une caractéristique
(3a)	$\forall i \in [1, n_b] : \sum_{j=1}^{j=n_c} B_{i,j} = 2$	$\mathcal{B}_i$ utilise deux caractéristiques
(4a)	$\forall i \in [1, n_p] : \sum_{j \in [1,n]} P_{i,j} \geq 1$	$\mathcal{P}_i$ utilise au moins une caractéristique ou au moins un terme binaire ou unaire
(5a)	$\forall j \in [1, n_c] : \; C_j \iff \bigvee_{i \in [1,n_u]} U_{i,j} \vee \bigvee_{i \in [1,n_b]} B_{i,j} \vee \bigvee_{i \in [1,n_p]} P_{i,j}$	Connecte $U_{i,j}$, $B_{i,j}$, et $P_{i,j}$ à C_j
(6a)	$\forall j \in [n_c + 1, n] : \sum_{i \in [1,n_p]} P_{i,j} > 0$	Force chaque terme unaire ou binaire à être utilisé par au moins un polynôme
(7a)	$\forall i \in [1, n_p] : \sum_{k=1}^{k<\binom{n+d_i}{d_i}} [M_{i,k} \neq 0] > 0$	Force les polynômes à ne pas être constant
(8a)	$\forall i \in [1, n_p] : \sum_{k=1}^{k\leq\binom{n+d_i}{d_i}} [M_{i,k} \neq 0] \leq maxz$	Force chaque polynômes à avoir un nombre limité de coefficients non nuls
(1b)	$\text{ELEMENT}(U\_IND_i, \langle U_{i,1}, \ldots, U_{i,n_c} \rangle, 1)$	Capture l'indice de la caractéristique utilisée
(2b)	$\text{ELEMENT}(U\_IND_i, \langle min_1, \ldots, min_{n_c} \rangle, U\_Min_i)$	Capture la valeur minimum de la caractéristique utilisée
(3b)	$\text{ELEMENT}(U\_IND_i, \langle max_1, \ldots, max_{n_c} \rangle, U\_Max_i)$	Capture la valeur maximum de la caractéristique utilisée
(4b)	$u_{f_i} \in \{\min\} \Rightarrow \left\{ \begin{array}{l} U\_CST_i > U\_Min_i \; \bigwedge \\ U\_CST_i < U\_Max_i \end{array} \right.$	Évite la simplification du terme unaire $\mathcal{U}_i$ s'il est utilisé sinon $\mathcal{U}_i$ est supprimé de $\mathcal{F}$
(1c)	$\text{ELEMENT}(B\_IND1_i, \langle B_{i,1}, \ldots, B_{i,n_c} \rangle, 1)$	Capture l'indice de la première caractéristique utilisée
(2c)	$\text{ELEMENT}(B\_IND2_i, \langle B_{i,1}, \ldots, B_{i,n_c} \rangle, 1)$	Capture l'indice de la deuxième caractéristique utilisée
(3c)	$(B\_O_i = 0) \Rightarrow \; B\_IND1_i < B\_IND2_i$	Ordonne les indices des caractéristiques
(4c)	$b_{f_i} \in \{\min\} \Rightarrow B\_O_i = 0$	Fixe l'ordre des deux arguments, car min est commutative
(5c)	$b_{f_i} \in \{\min\} \Rightarrow \text{TABLE} \left(\begin{array}{l} \langle B\_IND1_i, B\_IND2_i \rangle, \\ \text{TABLE\_UNORDERED} \end{array} \right)$	Assigne deux caractéristiques qui ne sont pas ordonnées

TABLE 2.5 – (**Partie supérieure**) Contraintes sur la structure de la formule $\mathcal{F}$; FD_TABLE est la liste des combinaisons de caractéristiques susceptibles d'être utilisées par $\mathcal{F}$, créée par le générateur de formules candidates tandis que *maxz* est le nombre maximum de coefficients non nuls d'un polynôme. (**Partie du milieu**) Contrainte sur un terme unaire $\mathcal{U}_i$ (avec $i \in [1, n_u]$), où u_{f_i} est la fonction assignée à $\mathcal{U}_i$ et min_j (avec $j \in [1, n_c]$) est la plus petite valeur de la j-ème caractéristique. (**Partie inférieure**) Contraintes sur un terme binaire $\mathcal{B}_i$ (avec $i \in [1, n_b]$), où b_{f_i} est la fonction assignée à $\mathcal{B}_i$, TABLE_UNORDERED est l'ensemble des paires d'indices des caractéristiques telles que la première caractéristique n'est pas toujours plus petite, ni plus grande que la deuxième caractéristique et FD_TABLE est l'ensemble des sous-ensembles minimaux de caractéristiques d'entrées qui déterminent chacun, la caractéristique de sortie.

d, nous posons une contrainte d'ordre lexicographique stricte.

$$\langle M_{1,1}, \ldots, M_{1,\binom{n+d}{d}} \rangle <_{\text{LEX}} \langle M_{2,1}, \ldots, M_{2,\binom{n+d}{d}} \rangle \tag{2.27}$$

entre les coefficients des monômes de $\mathcal{P}_1$ et $\mathcal{P}_2$. L'opérateur $<_{\text{LEX}}$ est l'ordre lexicographique strict défini selon la définition 32.

Pour les autres fonctions autres que $\min(\cdot,\cdot)$, il y a également des contraintes similaires de simplification et de cassage de symétries.

Contraintes pour $(\mathcal{C}ond\,?\,\mathcal{T}hen : \mathcal{E}lse)$	Description
(1d) $\text{ELEMENT}(C\_IND1, \langle V_{1,r}, \ldots, V_{n_c,r}\rangle, C\_VAL1_r)$	Capture la valeur de la première caractéristique à la r-ème ligne de $\mathcal{T}$
(2d) $\text{ELEMENT}(C\_IND2, \langle V_{1,r}, \ldots, V_{n_c,r}\rangle, C\_VAL2_r)$	Capture la valeur de la deuxième caractéristique à la r-ème ligne de $\mathcal{T}$
(3d) $C\_IND1 < C\_IND2$	Ordonne les indices des caractéristiques
(4d) $B\_VAL_r \iff C\_VAL1_r = C\_VAL2_r$	Teste l'égalité des deux caractéristiques à la r-ème ligne de $\mathcal{T}$
(5d) $\mathcal{C}ond \in \{\mathcal{C}_{j_1} = \mathcal{C}_{j_1}\} \Rightarrow \text{TABLE}\left(\begin{array}{c}\langle C\_IND1, C\_IND2\rangle,\\ \text{TABLE\_NOTALLEQUAL}\end{array}\right)$	Assigne deux caractéristiques $\mathcal{C}_{j_1}$ et $\mathcal{C}_{j_1}$ qui sont parfois égales et parfois pas

TABLE 2.6 – Contraintes sur la condition $\mathcal{C}ond$ lorsqu'elle correspond à l'égalité entre deux caractéristiques $\mathcal{C}_{j_1}$ et $\mathcal{C}_{j_1}$, où TABLE_NOTALLEQUAL est l'ensemble des paires d'indices des caractéristiques telles que les valeurs des deux caractéristiques sont parfois égales dans certaines entrées de la table des bornes numériques $\mathcal{T}$ et parfois différentes dans d'autres entrées de $\mathcal{T}$. Cet ensemble est fourni grâce aux contraintes de comparaisons générées dans les métadonnées de la phase (A2).

Contraintes d'égalité Pour chaque ligne r de la table des bornes numériques $\mathcal{T}$ nous posons des contraintes connectant les caractéristiques sélectionnées $\mathcal{C}_j$ avec :

- la valeur $U\_VAL_{i,r}$ de la variable évaluant le terme unaire $\mathcal{U}_i$;
- la valeur $B\_VAL_{i,r}$ de la variable évaluant le terme binaire $\mathcal{B}_i$;
- la valeur $P\_VAL_{i,r}$ de la variable évaluant le polynôme $\mathcal{P}_i$;
- les valeurs $C\_VAL1_r, C\_VAL2_r, B\_VAL_r, T\_VAL_r$ et E_VAL_r des variables évaluant la formule conditionnelle $(\mathcal{C}ond\,?\,\mathcal{T}hen : \mathcal{E}lse)$.

Pour chaque ligne r nous posons aussi une contrainte d'égalité connectant la valeur de la formule candidate $\mathcal{F}$ sur la ligne r avec la valeur correspondante $P\_VAL_{i,r}$ de la borne de la même ligne. Par exemple, dans le cas de la recherche d'un polynôme $\mathcal{P}_i$ de second degré avec deux caractéristiques bornantes et un terme unaire, la contrainte que l'on pose à la r-ème ligne est

$$P\_VAL_{i,r} = \sum_{0\leq \ell+m+u\leq 2} M_{\ell,m,u} \cdot (V_{j_1,r})^{\ell} \cdot (V_{j_2,r})^{m} \cdot (U\_VAL_{i,r})^{u} \tag{2.28}$$

où $V_{j,r}$ est la valeur de la j-ème caractéristique à la r-ème entrée de la table des bornes numériques et $M_{l,m,u}$ est le coefficient à déterminer du monôme dont les exposants de ses variables sont respectivement ℓ, m et u (il s'agit des coefficients $M_{i,k}$ de (7a) et (8a)).

Dans le cas de la conditionnelle ($\mathcal{C}ond\,?\,\mathcal{T}hen : \mathcal{E}lse$) où le $\mathcal{C}ond$ est une égalité entre deux caractéristiques $\mathcal{C}_{j_1}, \mathcal{C}_{j_2}$, le $\mathcal{T}hen$ et le $\mathcal{E}lse$ sont des polynômes, nous posons, en plus des contraintes (1d) à (5d) de la table 2.6, les contraintes suivantes à la r-ème entrée de $\mathcal{T}$:

$$B\_VAL_r = 1 \Rightarrow T\_VAL_r = P\_VAL_{i_1,r} \tag{2.29}$$

$$B\_VAL_r = 0 \Rightarrow E\_VAL_r = P\_VAL_{i_2,r} \tag{2.30}$$

$$T\_VAL_r \neq E\_VAL_r \tag{2.31}$$

Les conditions $\mathcal{C}ond$ autres que l'égalité entre deux caractéristiques ont des contraintes similaires. Notons que, pour une fonction binaire $\min(\cdot,\cdot)$ ayant pour arguments deux polynômes $\mathcal{P}_1$ et $\mathcal{P}_2$, nous imposons que pour au moins une des entrées de la table des bornes numériques la valeur de $\mathcal{P}_1$ soit strictement plus petite que la valeur de $\mathcal{P}_2$ sur la même entrée et le contraire soit également réalisé sur une autre entrée de la table. Finalement, pour limiter la génération des formules inutilement complexes, nous minimisons la somme des valeurs absolues des coefficients de la formule candidate $\mathcal{F}$.

2.4 Évaluation du Bound Seeker

Pour présenter l'évaluation du système d'acquisition de conjectures qu'est le Bound Seeker, nous mettons l'accent sur l'objet combinatoire qu'est le graphe orienté, car nous avons des résultats de la littérature avec lesquels nous pouvons faire des comparaisons. Néanmoins, la table 2.7 donne des résultats globaux du test du Bound Seeker sur tous les autres objets combinatoires introduits dans la section 1.2.1 du chapitre 1. Elle présente les résultats des expériences exécutées pendant sept jours au plus, fournissant un total de 16130 conjectures pour les 8 objets combinatoires. Pour réaliser les expériences qui ont fourni les résultats de la table 2.7, nous avons sollicité la grille de calculs parallèles de l'Alliance de recherche numérique du Canada (`https://alliancecan.ca/fr`). Nous y avons mobilisé 126 cœurs à raison d'un cœur pour la recherche des bornes d'une carte d'un objet combinatoire. Chaque cœur alloue 10 Go de mémoire. Comme caractéristiques des cœurs utilisés, nous avons les processeurs Intel Silver 4216 Cascade Lake de 2.1 GHz et E7-4809 v4 Broadwell de 2.1 GHz. Comme il est difficile de retracer les caractéristiques du cœur utilisé pour une carte, nous avons refait l'expérience pour le graphe orienté dans des conditions reproductibles en disposant d'une seule machine avec des caractéristiques bien déterminées. Nous décrirons dans ce qui suit le cadre expérimental et les résultats obtenus lors de cette deuxième expérimentation sur les graphes orientés.

Dans le cadre du graphe orienté, le Bound Seeker construit seize cartes sur les bornes précises inférieures et supérieures du nombre de sommets, du nombre d'arcs, du nombre de composantes connexes, du nombre de composantes fortement connexes et de leurs tailles maximum et minimum respectives. Les composantes du Bound Seeker sont écrites en SICStus Prolog et constituent près de 10000 lignes de code disponible à l'adresse `https://github.com/cquimper/Map`

Objet combinatoire		Poly. seule		Bool./Cond./Poly.			
	Nombre de cartes	#P1	Temps	#B2	#C2	#P2	Temps
Graphe orienté	16	1698	33339	502	535	1376	42373
Arbre enraciné	10	107	1159	61	65	63	1073
Forêt enracinée avec sommet isolé	20	1109	28238	436	702	724	27542
Forêt enraciné sans sommet isolée	20	1030	26916	415	445	919	30492
Partitions non vides	10	671	55109	113	182	484	79234
Partitions avec ensemble vide	10	275	20521	51	111	181	22048
Séquence d'entiers 0/1	20	3525	95847	1197	626	2780	110724
Séquence cyclique d'entiers 0/1	20	2887	114700	1092	657	2413	118043

TABLE 2.7 – Nombre de conjectures minimum/maximum et de conjectures de minimalité/maximalité découvertes pour toutes les cartes des 8 objets combinatoires avec le temps donné en minutes par le Bound Seeker pour trouver ces conjectures en utilisant seulement les polynômes (voir Poly. seule), puis les expressions booléennes et conditionnelles en plus (voir Bool./Cond./Poly.). Le terme #P1 représente la colonne du nombre de conjectures trouvées en utilisant uniquement les polynômes. Les termes #B2,#C2 et #P2 représentent les nombres de conjectures trouvées utilIsant respectivement des expressions booléennes, conditionnelles et polynômiales lorsque ces trois types de formules sont exploitées par le Bound Seeker.

`SeekerCP2022`. Ce sont notamment les composantes de la génération des données, des métadonnées, des formules candidates, du modèle de contraintes et de la phase des tests des conjectures. La phase de génération du jeu de données produit un total de 1944 tables de bornes numériques dans le cas du graphe orienté occupant une taille mémoire de 2 Go pour des instances de taille allant de 2 à 26 sommets ; chaque table fournit la borne supérieure ou inférieure précise d'une caractéristique en fonction de différents sous-ensembles d'autres caractéristiques. Pour les autres objets combinatoires, les conditions et résultats de l'expérience sont répertoriés dans la table 2.7. Nous évaluons le Bound Seeker sous les angles suivants :

— le pourcentage de conjectures qui, bien qu'apprises à partir des tables dont la taille maximale a été sélectionnée par la phase (A3), continuent d'être vraies sur les tables de grandes tailles, c'est-à-dire les tables dont les instances vont jusqu'à 26 sommets.

— le pourcentage des bornes de la base des données des invariants du document [10] sur le graphe orienté, qui ont été retrouvées par le Bound Seeker, ainsi que le pourcentage de celles qui n'ont pas été retrouvées.

— la véracité des bornes découvertes. En plus des conjectures retrouvées du catalogue des contraintes globales, nous avons prouvé manuellement quatre nouvelles conjectures de bornes sur le graphe orienté. Les preuves sont présentées au chapitre 4. En utilisant Wolfram|Alpha [42], nous avons aussi testé la cohérence de 105 liens de projections ou conjectures de maximalité et minimalité des cartes qui projettent une borne précise B_1 sur une autre borne précise B_0 s'exprimant avec une caractéristique en moins que B_1. Cela se fait en substituant dans B_1 la caractéristique à éliminer par l'expression définie par la conjecture de maximalité correspondante.

La complexité augmentant avec le nombre de caractéristiques d'entrée, nous limitons notre évaluation à au plus trois caractéristiques. Toutes les expériences d'acquisition des conjectures des seize cartes sont faites avec le même paramétrage du Bound Seeker, c'est-à-dire sans réglage manuel dépendant du type de carte. L'expérience est réalisée en calcul parallèle en sollicitant huit cœurs à raison d'un cœur pour deux cartes. L'expérience est faite en utilisant SICStus 4.6.0 sur un 2015 iMac avec 4 GHz Core i7 et 32Go de mémoire. La recherche des bornes d'une carte dure au plus quatre jours et quelques heures. En cherchant des bornes inférieures précises sur 350 tables de bornes inférieures numériques, le Bound Seeker trouve au moins une formule de borne inférieure précise pour 279 tables de bornes numériques et 1236 conjectures de minimalité lorsqu'il n'utilise que des polynômes. En cherchant des bornes supérieures précises sur 202 tables de bornes supérieures numériques, il trouve au moins une formule de borne inférieure précise sur 149 tables de bornes numériques et 975 conjectures de minimalité lorsqu'il n'utilise que des polynômes. Lorsqu'il utilise en plus les formules booléennes et conditionnelles, il trouve 3 nouvelles bornes inférieures et 93 nouvelles conjectures de maximalité et minimalité. La table 2.8 fournit les résultats des seize cartes : pour chaque carte, nous donnons le nombre de formules trouvées en utilisant uniquement les polynômes (voir la colonne #P1), puis le nombre de formules trouvées en utilisant en plus les expressions booléennes et les conditionnelles (voir les colonnes #B2, #C2 et #P2). En utilisant les expressions booléennes et conditionnelles, 3.8% de nouvelles formules sont générées par rapport au fait d'utiliser uniquement des polynômes. En plus de cela, 31.07% des formules qui utilisent les polynômes sont remplacées par des formules plus simples utilisant les expressions booléennes et conditionnelles. Le temps important que met le Bound Seeker pour trouver ces conjectures s'explique par le fait qu'un grand nombre de formules candidates est testé, ceci étant dû au grand nombre de dépendances fonctionnelles minimales qui sont des combinaisons des caractéristiques d'entrée et secondaires. Cela est aussi dû aux règles de la grammaire assez riche en formules. Cela est également dû aux occurrences multiples de variables et aux contraintes arithmétiques telles que div et mod qui sont peu propagées par les solveurs de contraintes.

2.4.1 Évaluation des conjectures acquises par rapport au jeu de données

Sur les 3625 conjectures acquises en utilisant uniquement les polynômes, 5 conjectures ont été invalidées en les testant sur toutes les instances de graphes orientés ayant au plus 26 sommets. Sur les 3624 conjectures acquises en utilisant en plus les expressions booléennes et conditionnelles, 16 conjectures ont été invalidées en les testant également sur toutes les instances du graphe orienté du jeu des données. Notons que le Bound Seeker arrête la recherche d'une formule d'une borne utilisant les polynômes lorsqu'il a trouvé pour la même borne une formule utilisant les expressions booléennes et conditionnelles.

Cartes	Poly. seule		Bool./Cond./Poly.				Cartes	Poly. seule		Bool./Cond./Poly.			
	#P1	Temps	#B2	#C2	#P2	Temps		#P1	Temps	#B2	#C2	#P2	Temps
$\mathcal{M}_{\mathcal{P}}^{c\geq}$	259	257	47	25	194	533	$\mathcal{M}_{\mathcal{P}}^{c\leq}$	100	218	9	17	77	439
$\mathcal{M}_{\mathcal{P}}^{\overline{c}\geq}$	129	230	0	13	120	476	$\mathcal{M}_{\mathcal{P}}^{\overline{c}\leq}$	153	193	32	31	94	542
$\mathcal{M}_{\mathcal{P}}^{\overline{s}\geq}$	97	130	0	8	90	306	$\mathcal{M}_{\mathcal{P}}^{\overline{s}\leq}$	102	392	15	31	60	999
$\mathcal{M}_{\mathcal{P}}^{a\geq}$	367	1248	38	102	255	3180	$\mathcal{M}_{\mathcal{P}}^{a\leq}$	384	2505	46	84	264	5939
$\mathcal{M}_{\mathcal{P}}^{\underline{c}\geq}$	63	167	10	27	27	388	$\mathcal{M}_{\mathcal{P}}^{\underline{c}\leq}$	130	223	16	14	102	457
$\mathcal{M}_{\mathcal{P}}^{\underline{s}\geq}$	43	54	0	18	25	226	$\mathcal{M}_{\mathcal{P}}^{\underline{s}\leq}$	48	171	1	8	40	365
$\mathcal{M}_{\mathcal{P}}^{s\geq}$	263	474	37	31	198	813	$\mathcal{M}_{\mathcal{P}}^{s\leq}$	93	100	5	17	73	267
$\mathcal{M}_{\mathcal{P}}^{v\geq}$	294	368	30	75	205	1570	$\mathcal{M}_{\mathcal{P}}^{v\leq}$	14	7	0	2	12	90

TABLE 2.8 – Nombre de conjectures minimum/maximum et de conjectures de minimalité/maximalité découvertes pour chacune des seize cartes avec le temps mis en minutes par le Bound Seeker pour trouver ces conjectures en utilisant seulement les polynômes (voir Poly. seule), puis les expressions booléennes et conditionnelles en plus (voir Bool./Cond./Poly.). Le terme #P1 représente la colonne du nombre de conjectures trouvées en utilisant uniquement les polynômes. Les termes #B2,#C2 et #P2 représentent les nombres de conjectures trouvées utilisant respectivement des expressions booléennes, conditionnelles et polynomiales lorsque ces trois types de formules sont exploitées par le Bound Seeker.

Nb. de caractéristiques d'entrée	1	2	3	Total	Pourcentage
Nb. de bornes précises équivalentes retrouvées par le BS	22	14	4	40	66,66%
Nb. de bornes plus précises trouvées par le BS	1	3	0	4	6,66%
Nb. de bornes précises plus générales trouvées par le BS	0	6	0	6	10%
Nb. de bornes erronées du GCC, corrigées par le BS	1	1	1	3	5%
Nb. de bornes du GCC qui ne sont pas retrouvées par le BS	0	0	7	7	11,66%
Nb. total de bornes du GCC par colonne	24	24	12	60	

TABLE 2.9 – Comparaison des conjectures des bornes précises découvertes par le Bound Seeker (BS) avec la base des données d'invariants du catalogue de contraintes globales (GCC). L'abréviation Nb. signifie nombre.

2.4.2 Comparaison des conjectures découvertes avec les bornes prouvées du catalogue de contraintes

Comme présenté dans la table 2.9, le Bound Seeker retrouve 66.66% des bornes du catalogue de contraintes même si les formules trouvées sont quelque peu différentes. Par exemple, la borne supérieure précise du nombre d'arcs a en fonction du nombre de sommets v, du nombre de composantes connexes c, et du nombre de composantes fortement connexes s dans le catalogue s'exprime par $a \leq c - 1 + (v - s + 1) \cdot (v - c + 1) + \lfloor \frac{(s-c+1)\cdot(s-c)}{2} \rfloor$, tandis que le Bound Seeker trouve une borne équivalente $a \leq \lfloor \frac{r^2+\overline{s}^2+v\cdot\underline{c}+r-\overline{s}+v}{2} \rfloor$, avec $\underline{c} = \max(2 \cdot v - v \cdot c, 1)$, $\overline{s} = v - s + 1$ et $r = v \cdot [c \geq 2] - c \cdot [c \geq 2]$; r est une caractéristique secondaire correspondante à $v - c \cdot \underline{c}$. Contrairement à la borne donnée par l'article [10], la borne trouvée par le Bound Seeker définit la taille $\underline{c}$ de la plus petite composante connexe et la taille $\overline{s}$ de la plus grande composante fortement connexe des instances extrêmes du graphe orienté pour lesquelles la borne supérieure

est atteinte.

Un exemple de borne généralisée trouvée par le Bound Seeker est la borne inférieure $a \geq ((v - \overline{c}) \leq 1\,?\,\max(v-1,1)\ :\ v-2)$, avec $v = (\underline{c} = \overline{c}\,?\,\underline{c}\ :\ \underline{c} + \overline{c})$ qui étend la borne du catalogue $\underline{c} \neq \overline{c} \Rightarrow a \geq \underline{c} + \overline{c} - 2 + (\underline{c} = 1)$. Un exemple de borne correcte découverte par le Bound Seeker qui corrige la borne erronée (i) $a \geq v - \lfloor \frac{s-1}{2} \rfloor$ du catalogue est (ii) $a \geq v - c_{\in\{2,3\}}$ avec $c_{\in\{2,3\}} = (v = s\,?\,\lfloor \frac{v}{2} \rfloor\ :\ \lfloor \frac{s-1}{2} \rfloor)$: lorsque la condition $v = s = 2$ est vérifiée, (i) retourne 2, au lieu de 1 comme le fait (ii). La borne (ii) $a \geq v - c_{\in\{2,3\}}$ peut s'interpréter de la manière suivante : pour minimiser le nombre d'arcs, on maximise le nombre de composantes connexes de la forme •→•, •→•→•, •→•←• ou •←•→• . Le fait de ne pouvoir retrouver certaines bornes du catalogue s'explique partiellement par la complexité limitée des sous-expressions (voir BTerme, UTerme dans la Section. 2.3.8) partagées par les formules polynomiales que génère le Bound Seeker et le manque de certaines caractéristiques secondaires.

2.4.3 Quelques cartes d'objets combinatoires

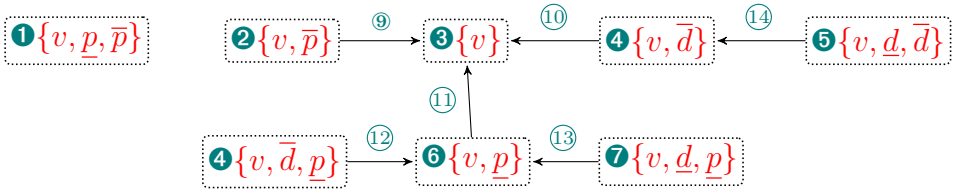
Pour montrer le côté général du concept de carte de conjectures et de sa construction, en plus des cartes de la figure 2.2 et la figure 2.3, nous présentons d'autres cartes de bornes supérieures ou inférieures de plusieurs caractéristiques de plusieurs objets combinatoires, qui ont été découvertes par le Bound Seeker. La figure 2.7 présente une carte de sept bornes inférieures précises du nombre de feuilles d'un arbre enraciné. La figure 2.8 présente une carte de treize bornes supérieures précises du nombre d'arcs d'un graphe orienté. La figure 2.9 présente une carte de sept bornes supérieures précises du nombre de variables valant 1 dans une séquence d'entiers 0/1.

2.5 Discussion

Nous avons introduit la notion de carte de conjectures qui est une structure qui connecte des ensembles de bornes précises. Pour obtenir une telle structure, nous proposons une approche constructive qui permet l'acquisition d'un ensemble des conjectures interconnectées sur des bornes précises des caractéristiques d'objets combinatoires. Nous avons montré qu'utiliser une grande variété de formules est pertinent dans le sens que cela permet de capturer un grand nombre de conjectures relativement simples sur des bornes précises. La variété des formules va des expressions booléennes aux polynômes partageant des sous-expressions interprétables, en passant par des formules conditionnelles. Ce travail ouvre un chemin vers de nouvelles applications du domaine de la programmation par contraintes dans les systèmes de découvertes automatiques de conjectures. Il crée une nouvelle ligne de recherche en apprentissage automatique dédiée à l'optimisation combinatoire telle que le mentionne l'article [14]. Cependant, il y a deux points que nous n'avons pas assez explorés dans ce chapitre. Le premier point est les conditions d'existence d'une instance d'un objet combinatoire. C'est-à-dire les combinaisons

valides, des valeurs des caractéristiques pour que l'instance soit constructible, car les bornes des cartes d'invariants reposent sur ces conditions. Par exemple, il est impossible de construire un graphe orienté dont le nombre de sommets est $v = 3$ et la taille de la plus petite composante connexe est $\underline{c} = 2$. Cela se traduit symboliquement d'une manière générale par l'expression $(v \geq 2 \cdot \underline{c}) \vee (v = \underline{c})$. En effet, lorsqu'un graphe orienté a une seule composante connexe, cette composante connexe est la plus petite et donc le nombre total de sommets est la taille de cette composante. Si le graphe orienté a plus d'une composante connexe, chacune de ces composantes connexes a au moins $\underline{c}$ sommets et par conséquent on a $v \geq 2 \cdot \underline{c}$. Nous appelons une telle propriété des graphes orientés une *condition de faisabilité* de l'objet combinatoire considéré. Nous aimerions acquérir automatiquement un grand nombre de telles conditions, car elles ont plusieurs utilités :

— elles définissent l'espace des instances possibles d'un objet combinatoires et donc elles



⑨ $\overline{p} = v - 1$ ⑩ $\overline{d} = [v \geq 2]$ ⑪ $\underline{p} = v - 1$

⑫ $\overline{d} = [v \geq 2] + [v - \underline{p} \geq 2]$ ⑬ $\underline{d} = [v \geq 2] + [v \cdot \underline{p} = 3]$

⑭ $\underline{d} = (v - \overline{d} = 1 \ ? \ \overline{d} : 1)$

❶ $f \geq \left(\underline{p} = 0 \ ? \ v : \left\lceil \frac{\overline{p} - \underline{p} + v - 1}{\overline{p}} \right\rceil\right)$

❷ $f \geq \left\lceil \frac{v - 1 + [v = 1]}{(\overline{p} = 0 \ ? \ 1 : \overline{p})} \right\rceil$

❸ $f \geq 1$

❹ $f \geq (\overline{d} = 0 \ ? \ 1 : \overline{d})$

❺ $f \geq \left(\underline{d} = 0 \ ? \ 1 : \left\lceil \frac{v \cdot \underline{d} + \overline{d} - \underline{d} - v + 1}{\underline{d}} \right\rceil\right)$

❻ $f \geq 1 + [(v - \underline{p}) \geq 2]$

❼ $f \geq \left(\underline{d} = 0 \ ? \ v : v + 1 - \left\lceil \frac{v}{\underline{d}} \right\rceil\right) + [(\underline{d} = 1) \wedge (v - \underline{p}) \geq 2]$

f : nombre de feuilles de l'arbre
$\boldsymbol{v}$: nombre de nœuds de l'arbre
$\underline{\boldsymbol{p}}$: profondeur minimale d'une feuille
$\overline{\boldsymbol{p}}$: profondeur maximale d'une feuille
$\underline{\boldsymbol{d}}$: nombre minimale de fils d'un nœud
$\overline{\boldsymbol{d}}$: nombre maximale de fils d'un nœud

FIGURE 2.7 – Carte $\mathcal{M}^{f\geq}_{\{v,\underline{p},\overline{p},\underline{d},\overline{d}\}}$ de bornes inférieures précises ❶ à ❼ du nombre de feuilles d'un arbre enraciné. Des exemples d'instances d'arbres enracinés atteignant les bornes sont omis. Les bornes ❶, ❷, ❺ et ❼ sont générées par le Bound Seeker mais en utilisant des techniques qui ne font pas partie des contributions de ce travail. La génération de telles bornes a été réalisée par Gindullin et *al.* et présentée dans l'article [36]. Remarquons qu'il n'existe pas de lien de projection ou encore de conjecture de minimalité $\underline{p} = g_{\underline{p}}(v, \overline{p})$ permettant de projeter la borne ❶ sur la borne ❷. Cela est dû au fait que $\underline{p}$ ne dépend pas fonctionnellement de v et $\overline{p}$ lorsque f est minimal. Par exemple, pour la combinaison $v = 8, \overline{p} = 3$, la valeur minimale de $f = 3$. Cependant, il y a deux valeurs possibles de $\underline{p}$ qui sont 1 et 2.

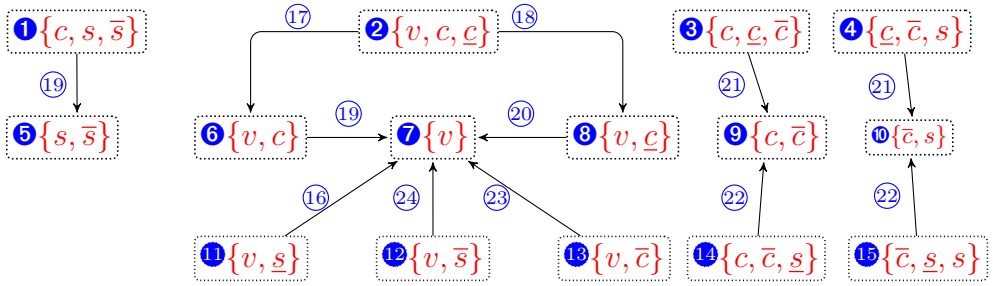
renforcent la cohérence des cartes de conjectures ;

— elles aident dans la construction automatique des cartes. En effet, les contraintes de comparaisons générées à la phase (A2) de génération des métadonnées sont des *conditions de faisabilités* simples impliquant juste deux caractéristiques ;

— elles seront d'un grand appui dans l'élaboration manuelle ou automatique des preuves des conjectures des cartes ;

— elles sont des contraintes à part entière que l'on peut utiliser pour améliorer le filtrage dans les problèmes d'optimisation combinatoire faisant intervenir des objets combinatoires.

Cependant, il y a plusieurs défis derrière l'acquisition automatique de conditions de faisabilités. Le premier est de générer un jeu minimal de données qui capture tous les cas d'invalidité d'une combinaison de valeurs des caractéristiques. Le deuxième défi est de générer des conditions de faisabilités qui ne s'impliquent pas entre elles. Nous aurons ce que nous appelons *carte des conditions des faisabilités* d'un objet combinatoire définie telle que deux conditions de faisabilités *sont connectées par un arc* lorsque l'une n'est pas impliquée par l'autre et utilise une des caractéristiques en moins que l'autre. Le dernier défi est de prouver les conditions de faisabilités acquises.



❶ $a \leq \overline{c}^2 - \overline{c} \cdot \overline{s} + 2 \cdot \overline{s} \cdot v$ avec $(v = s \cdot \overline{s})$ et $(\overline{c} = s \cdot \overline{s} - c \cdot \overline{s} + \overline{s})$
❷ $a \leq (c-1) \cdot \underline{c}^2 + (v - (c-1)) \cdot \underline{c})^2$
❸ $a \leq \underline{c}^2 + \overline{c}^2 \cdot (c-1)$
❹ $a \leq \underline{c}^2 + \overline{c}^2 \cdot (s-1)$
❺ $a \leq \lfloor (s^2+s)/2 \rfloor \cdot \overline{s}^2$
❻ $a \leq c^2 - 2 \cdot v \cdot c + v^2 - c + 2 \cdot v$
❼ $a \leq v^2$
❽ $a \leq v^2 - 2 \cdot v \cdot \underline{c} + 2 \cdot \underline{c}^2$
❾ $a \leq c \cdot \overline{c}^2$
❿ $a \leq s \cdot \overline{c}^2$
⓫ $a \leq v^2 - v \cdot \underline{s} + \underline{s}^2$
⓭ $a \leq (v \bmod \overline{c})^2 - c \cdot (v \bmod \overline{c}) + v \cdot \overline{c}$

⑯ $\underline{s} = v$ ⑰ $c = (v = \underline{c})?\ 1 : 2$ ⑱ $\underline{c} = (c = 1)?\ v : 1$

⑲ $c = 1$ ⑳ $\underline{c} = v$ ㉑ $\underline{c} = \overline{c}$ ㉒ $\underline{s} = \overline{c}$ ㉓ $\overline{c} = v$

a : nombre d'arcs
v : nombre de nœuds
c : nombre de composantes connexes
s : nombre de composantes fortement connexes
$\overline{c}$: taille de la plus grande composante connexe
$\underline{c}$: taille de la plus petite composante connexe
$\overline{s}$: taille de la plus grande composante fortement connexe
$\underline{s}$: taille de la plus petite composante fortement connexe

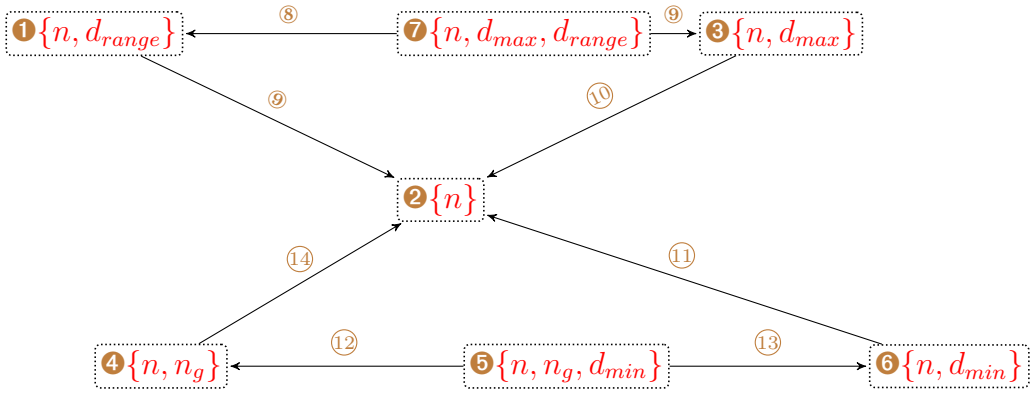
FIGURE 2.8 – Carte $\mathcal{M}^{a\leq}_{\{v,c,s,\overline{c},\underline{c},\overline{s},\underline{s}\}}$ de 13 bornes supérieures précises du nombre d'arcs d'un graphe orienté. Des exemples d'instances d'arbre enracinés atteignant les bornes ont été omis.

Le deuxième point que nous n'avons pas également assez exploré concerne la qualité et la disponibilité des données que nous utilisons pour l'apprentissage de cartes. En effet, la méthode décrite à la phase (A1) de génération du jeu des instances d'objet combinatoire ne garantit pas les qualités suivantes :

— la caractéristique à borner admet effectivement une borne supérieure ou une borne inférieure avec le sous-ensemble de caractéristiques bornantes considérées. Par exemple, le nombre d'arcs a d'un graphe orienté ne peut pas être borné supérieurement juste en connaissant les tailles $\overline{c}$ et $\underline{c}$ de sa plus grande et sa plus petite composante connexe ;

— la borne inférieure ou supérieure peut s'exprimer avec le biais des formules de la grammaire du Bound Seeker ou bien se calculer en temps $O(1)$;

— les valeurs optimales des caractéristiques à borner retenues dans le jeu de données sont correctes. En effet, supposons que nous voulons générer la table des bornes numériques



❶ $n_v \leq n - n_g - d_{range} + 1$ avec $n_g = (d_{range} = 0 \,?\, 1 : 3)$

❷ $n_v \leq n$
❸ $n_v \leq n - d_{max}$
❹ $n_v \leq (n \cdot n_g) \bmod (n+1)$
❺ $n_v \leq d_{min} + n \cdot \min(n_g, 1) - n_g \cdot d_{min}$
❻ $n_v \leq n - d_{min}$
❼ $n_v \leq n + d_{range} - d_{max} - d_{max} \cdot \min(d_{range}, 1)$

⑧ $d_{max} = (d_{range} = 0 \,?\, 0 : d_{range} + 1)$

⑩ $d_{max} = 0$ ⑪ $d_{min} = 0$ ⑨ $d_{range} = 0$

⑫ $d_{min} = (n_g \leq 1 \,?\, 0 : 1)$ ⑬ $n_g = (d_{min} = 0 \,?\, 1 : 2)$

⑭ $n_g = 1$

n_v : nombre de variables valant 1 dans la séquence
$\boldsymbol{n}$: longueur de la séquence
$\boldsymbol{n_g}$: nombre de groupes de la séquence
$\boldsymbol{d_{min}}$: longueur minimale d'une interdistance
$\boldsymbol{d_{max}}$: longueur maximale d'une interdistance
$\boldsymbol{d_{range}}$: différence entre d_{max} et d_{min}

FIGURE 2.9 – Carte $\mathcal{M}^{n_v \leq}_{\{n,n_g,d_{min},d_{max},d_{range}\}}$ de bornes supérieures précises ❶ à ❼ du nombre n_v de variables valant 1 dans une séquence de booléens. Des exemples d'instances de séquence de booléens atteignant les bornes ont été omis.

$\mathcal{T}^{v\leq}_{\{t,\bar{t}\},4}$ d'une forêt enracinée dont la caractéristique à borner est le nombre de sommets v et les caractéristiques bornantes sont le nombre d'arbres t de la forêt et la taille maximale $\bar{t}$ d'un arbre de la forêt. Alors cette table ne présente que les instances de forêt d'au plus quatre sommets. Par exemple la combinaison $t = 2, \bar{t} = 3, v = 4$ est contenue dans cette table. Lorsque nous appliquons à cette table la phase (A1) de génération des données sur l'ensemble de toutes les instances de forêt de taille au plus quatre sommets, la méthode retiendra $v = 4$ comme valeur maximale du nombre de sommets d'une forêt qui a $t = 2$ arbres lorsque la taille maximale d'un arbre de la forêt est $\bar{t} = 3$; ce qui est faux, car en réalité une telle forêt a un nombre maximum de 6 sommets répartis uniformément entre les deux arbres. Ainsi, la méthode génère des données erronées pour certaines combinaisons de caractéristiques bornantes. Un début de solution est d'identifier les combinaisons erronées de la table $\mathcal{T}^{o\leq}_{P_b,k}$ en les comparant aux combinaisons de la table $\mathcal{T}^{o\leq}_{P_b,n_{max}}$ où $n_{max} > k$ et utiliser une fois de plus la composante booléenne du Bound Seeker pour caractériser sous forme de propriété générale ces combinaisons. Une fois que ces propriétés sont apprises, elles peuvent être utilisées pour corriger des tables de taille quelconque, de sous-ensemble de caractéristiques quelconque et d'objet combinatoire distinct de celui à partir duquel la propriété a été apprise. En effet, il existe des relations entre objets combinatoires que nous présentons au chapitre suivant ;

— une autre qualité que doivent avoir les données c'est d'être assez représentatif des propriétés que l'on cherche à exprimer symboliquement. Cependant, la méthode de la phase (A1) demande au préalable la génération de toutes les instances d'une taille donnée. Cela peut être limité par la disponibilité de l'espace mémoire du système. Par conséquent, les données disponibles peuvent n'être pas assez représentatives de l'information que l'on veut apprendre des données, notamment pour des caractéristiques gloutonnes comme le nombre d'arcs a d'un graphe orienté. C'est la raison pour laquelle cette caractéristique ne figure pas comme caractéristique bornante ou d'entrée dans les tables de données. Pour répondre à cette faiblesse, nous faisons appel au calcul symbolique en nous basant sur les conjectures déjà générées par le Bound Seeker. Cette méthode est présentée au chapitre suivant.

Chapitre 3

Liens entre cartes d'invariants par inversion de conjectures et correspondance entre objets combinatoires

Comme nous l'avons vu au chapitre 2, générer des cartes demande beaucoup de temps. De plus, ces cartes sont incomplètes pour l'une des raisons suivantes :

— certaines bornes ne sont pas trouvées ou sont erronées, soit en raison du manque des données d'apprentissage complètes, soit parce que la grammaire des formules est incomplète.

— soit du fait que les cartes regroupant des bornes exactes et leurs relations devraient être complétées par des cartes regroupant de manière systématique les conditions de faisabilités entre les caractéristiques principales d'un objet combinatoire.

Pour répondre à ces faiblesses, nous proposons l'utilisation du calcul symbolique ou algébrique automatique pour trouver beaucoup plus de bornes et renforcer la cohérence des cartes. Cela consiste à calculer automatiquement l'inverse d'une borne générée par le Bound Seeker ou à identifier un objet combinatoire présent dans la structure d'un autre objet combinatoire.

Les motivations principales de ce chapitre sont :

— améliorer la compréhension et la cohérence des cartes de conjectures. En effet, nous voulons montrer que les cartes ne connectent pas seulement des bornes, mais aussi qu'elles se connectent entre elles ;

— compléter les cartes des bornes manquantes. Les bornes sont très sollicitées dans la résolution des problèmes combinatoire. Par conséquent nous souhaitons en acquérir davantage.

— gagner en temps d'acquisition des cartes de conjectures. En effet, le calcul symbolique automatique est plus rapide que l'apprentissage automatique.

Les contributions apportées par ce chapitre sont donc :

— la mise en exergue de liens entre cartes de conjectures d'un même objet combinatoire et entre cartes de conjecture d'objets combinatoires distincts ;
— la découverte assez rapide de nouvelles conjectures de bornes non accessibles par la grammaire des formules du Bound Seeker ;
— la mise en évidence de nouveaux éléments d'élaboration des preuves des conjectures des cartes autant sur le plan manuel qu'automatique.

Par la suite, à la section 3.1, nous montrons comment générer de nouvelles cartes en faisant une opération algébrique que nous appelons *inversion d'une conjecture*. La section 3.2 présente une technique de preuve de conjectures par encodage d'objets combinatoire. Puis la section 3.3 est consacrée à la prise de recul sur le travail effectué.

3.1 Inversion de conjectures

Partant d'une borne générée par le Bound Seeker, nous utilisons les méthodes de résolution d'équations et inéquations paramétrées [41] pour déduire des bornes de caractéristiques d'objets combinatoires et mettre en évidence des relations existant entre des cartes d'un même objet combinatoire. Dans ce contexte, nous appelons ce processus de déduction d'une borne non découverte par le Bound Seeker, à partir d'une borne initiale découverte par le Bound Seeker l'*inversion de conjectures*.

Définition 33 (borne inverse)**.** Soit f une fonction telle que $f(P_b)$ est une borne précise de la caractéristique bornée o en fonction des caractéristiques bornantes P_b. C'est-à-dire que l'on a la relation

$$o \ op_1 \ f(P_b) \text{ avec } op_1 \in \{\leq, \geq\} \tag{3.1}$$

et, de plus, pour chaque combinaison V_b réalisable de valeurs respectives des caractéristiques de P_b, il existe toujours une instance de l'objet combinatoire de valeur v_o de o telle que

$$v_o \ = \ f(V_b) \tag{3.2}$$

Nous appelons *borne inverse* de $f(P_b)$ pour la caractéristique $u \in P_b$, une fonction que nous notons $f_{inv}(P_b \cup \{o\} \setminus \{u\})$ telle que

$$u \ op_2 \ f_{inv}(P_b \cup \{o\} \setminus \{u\}) \text{ avec } op_2 \in \{\leq, \geq\} \text{ et } u \in P_b \tag{3.3}$$

et également

$$o \ op_1 \ f(P_b) \Longleftrightarrow u \ op_2 \ f_{inv}(P_b \cup \{o\} \setminus \{u\}) \tag{3.4}$$

obtenue en appliquant sur (3.1) les *règles d'équivalences du calcul algébrique de l'ensemble des nombres réels* $\mathbb{R}$. Dès lors que cette fonction $f_{inv}(P_b \cup \{o\} \setminus \{u\})$ est déterminée, elle constitue une *borne* de la caractéristique u qui s'exprime en fonction des autres caractéristiques restantes $P_b \cup \{o\} \setminus \{u\}$.

Exemple 20. Considérons les deux cartes $\mathcal{M}^{S\geq}_{\{n,\overline{M}\}}$ et $\mathcal{M}^{n\leq}_{\{S,\overline{M}\}}$ de l'objet combinatoire qu'est la partition d'ensembles de la figure 3.1. Chacune de ces deux cartes connecte respectivement 2 bornes inférieures et précises de la somme S des carrés des tailles des partitions et 2 bornes supérieures du cardinal n de l'ensemble à partitionner. La caractéristique $\overline{M}$ est la taille maximale d'une partition. Alors, l'une des cartes s'obtient par inversion des bornes de l'autre carte. En effet, pour faire le parallèle avec la définition 33, si l'on prend la borne ❶, la caractéristique à borner est $o = S$, les caractéristiques bornantes c_i sont n et $\overline{M}$ et la fonction f est $f(n, \overline{M}) = \overline{M}^2 - \overline{M} + n$. Ainsi, après inversion de la borne ❶ en faisant des additions de part et d'autre de son inégalité, nous obtenons la borne ❹ où $u = n$ et la fonction f_{inv} est $f_{inv}(S, \overline{M}) = S - \overline{M}^2 + \overline{M}$.

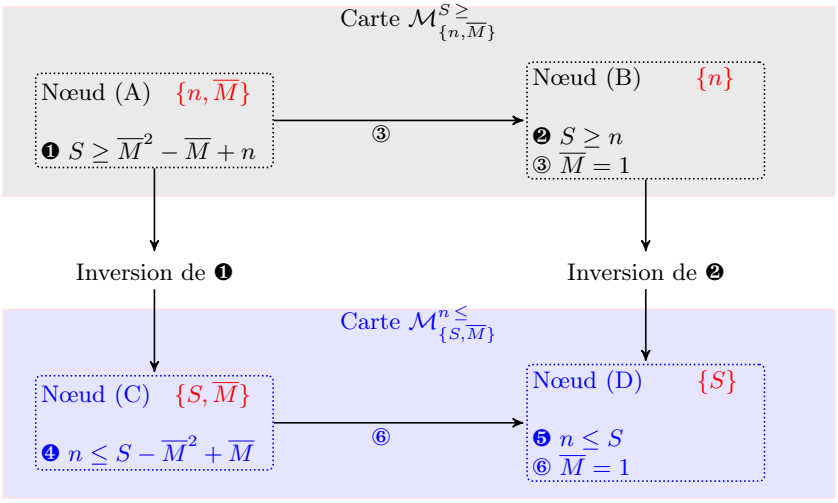


FIGURE 3.1 – Correspondance entre deux cartes $\mathcal{M}^{S\geq}_{\{n,\overline{M}\}}$ et $\mathcal{M}^{n\leq}_{\{S,\overline{M}\}}$ par inversion de conjectures. n est le cardinal de l'ensemble partitionné, $\overline{M}$ est la taille maximale d'une partition, S est la somme des carrés des tailles des partitions.

L'exemple 20 est un simple exemple introduisant la notion d'inversion. Par la suite, nous présentons un exemple plus complexe, l'exemple 21, en détaillant le processus d'inversion. En effet, la méthode d'inversion utilisée dans cet exemple nécessite des connaissances sur l'*étude des signes des polynômes de second degré* et sur des *conditions de faisabilités* de l'objet combinatoire considéré.

Exemple 21. Notons d'abord que le Bound Seeker ne trouve pas de borne précise pour la table $\mathcal{T}^{\overline{M}\leq}_{\{n,S\},30}$ de la carte $\mathcal{M}^{\overline{M}\leq}_{\{n,S\}}$. Partant à nouveau de la borne ❶ de la carte $\mathcal{M}^{S\geq}_{\{n,\overline{M}\}}$ de la figure 3.1, nous calculons l'inversion

$$\overline{M} \ op \ g(S,n) \text{ avec } op \in \{\leq, \geq\} \tag{3.5}$$

Pour déterminer l'inégalité *op* et la fonction $g(S,n)$, nous effectuons d'abord la simple transformation suivante :

$$❶ \ S \geq \overline{M}^2 - \overline{M} + n \Longleftrightarrow \overline{M}^2 - \overline{M} + n - S \leq 0 \tag{3.6}$$

Posons $P(\overline{M}) = \overline{M}^2 - \overline{M} + n - S$

$$(3.6) \Longleftrightarrow P(\overline{M}) \leq 0 \tag{3.7}$$

En considérant que S et n sont fixés, alors $P(\overline{M})$ est un polynôme de second degré dont la variable est $\overline{M}$. Le coefficient du monôme du second degré est $a = 1$, le coefficient du monôme de premier degré est $b = -1$ et le terme constant ou le monôme de degré 0 est $c = n - S$. Donc, en partant de la borne ❶, nous déduisons que l'ensemble des valeurs de $\overline{M}$ doit vérifier l'inéquation $P(\overline{M}) \leq 0$. Identifier toutes ces valeurs, c'est déterminer les bornes de $\overline{M}$. Pour trouver ces valeurs, il faut donc résoudre l'inéquation de second degré $P(\overline{M}) \leq 0$ dont l'inconnue est $\overline{M}$. Résoudre une inéquation de second degré se fait généralement par calcul des racines du polynôme après s'être assuré qu'elles existent. Cela se fait en vérifiant que le discriminant du polynôme est positif. Puis une fois les racines calculées, elles sont utilisées dans un tableau des signes qui permet d'identifier les valeurs de la variable, pour lesquelles le polynôme est positif ou négatif. Si les racines n'existent pas, c'est-à-dire si le discriminant est négatif alors le signe du polynôme est celui du coefficient a de son monôme de second degré. Et, dans ce cas, sa variable $\overline{M}$ n'est pas bornée. Ainsi, le discriminant de $P(\overline{M})$ est

$$\Delta = b^2 - 4 \cdot a \cdot c = 1 - 4 \cdot 1 \cdot (n - S) = 4 \cdot (S - n) + 1 \tag{3.8}$$

En utilisant la borne ❷ $S \geq n$ de la figure 3.1 qui est non seulement une borne précise, mais aussi une *condition de faisabilité* à savoir, nous avons

$$S \geq n \Longleftrightarrow S - n \geq 0 \tag{3.9}$$

$$\Longleftrightarrow 4 \cdot (S - n) \geq 0 \tag{3.10}$$

$$\Longleftrightarrow 4 \cdot (S - n) + 1 \geq 1 \tag{3.11}$$

$$\Rightarrow \Delta > 0 \tag{3.12}$$

Donc grâce à la condition de faisabilité $S \geq n$, nous déduisons que le discriminant Δ de $P(\overline{M})$ est toujours positif, quelles que soient les combinaisons *valides* des valeurs fixées de S et n.

Ainsi, nous pouvons calculer les racines $\overline{M}_1$ et $\overline{M}_2$ de $P(\overline{M})$ comme suit :

$$\overline{M}_1 = \frac{-b - \sqrt{\Delta}}{2 \cdot a} = \frac{1 - \sqrt{4 \cdot (S-n)+1}}{2} \tag{3.13}$$

$$\overline{M}_2 = \frac{-b + \sqrt{\Delta}}{2 \cdot a} = \frac{1 + \sqrt{4 \cdot (S-n)+1}}{2} \tag{3.14}$$

Nous pouvons donc les utiliser dans le tableau des signes comme suit :

$\overline{M}$	$-\infty$		$\overline{M}_1$		$\overline{M}_2$		$+\infty$
$P(\overline{M})$	signe de a		0	signe opposé de a	0	signe de a	

Comme $a = 1$, alors son signe est positif, et donc son signe opposé est le signe négatif. Par conséquent le tableau de signes nous indique que $P(\overline{M})$ est négatif, c'est-à-dire que $P(\overline{M}) \leq 0$, lorsque $\overline{M}_1 \leq \overline{M} \leq \overline{M}_2$. Donc $\overline{M}_1$ et $\overline{M}_2$ sont des bornes potentielles de $\overline{M}$. Il reste à vérifier la condition de faisabilité qui est la positivité des bornes de $\overline{M}$. D'après (3.11), on a

$$4 \cdot (S-n)+1 \geq 1 \iff \sqrt{4 \cdot (S-n)+1} \geq 1 \tag{3.15}$$

$$\iff 1 - \sqrt{4 \cdot (S-n)+1} \leq 0 \tag{3.16}$$

$$\iff \frac{1 - \sqrt{4 \cdot (S-n)+1}}{2} \leq 0 \tag{3.17}$$

$$\iff \overline{M}_1 \leq 0. \tag{3.18}$$

$\overline{M}_1$ est négatif. Seule $\overline{M}_2$ est positif. Ainsi, on peut ne pas considérer $\overline{M}_1$ comme une borne de $\overline{M}$. Finalement nous avons :

$$\overline{M} \leq \frac{1 + \sqrt{4 \cdot (S-n)+1}}{2} \tag{3.19}$$

Et comme $\overline{M}$ ne prend que des valeurs entières, alors nous obtenons la borne supérieure suivante de $\overline{M}$:

$$\overline{M} \leq \left\lfloor \frac{1 + \sqrt{4 \cdot (S-n)+1}}{2} \right\rfloor \tag{3.20}$$

Définition 34 (propriété de projection)**.** Soit un ensemble de *caractéristiques bornantes* $\mathcal{P}$ d'un objet combinatoire $\mathcal{O}$, une *caractéristique bornée* $o \notin \mathcal{P}$, $c_i \in \mathcal{P}$ une caractéristique bornante et deux sous-ensembles $P_b \subset \mathcal{P}$ et $P_{bi} \subset \mathcal{P}$ de caractéristiques bornantes. Pour une caractéristique c donnée, nous notons $\mathcal{D}(c)$ le domaine des valeurs de la caractéristique c. Soient f et f_i deux fonctions de $\prod_{c \in P_b} \mathcal{D}(c)$ vers $\mathbb{N}$. Soit une carte de bornes supérieures précises $\mathcal{M}^{o\leq}_{\mathcal{P}}$.

Nous disons que la borne $o \leq f(P_b)$ *respecte* la *propriété de projection* de la carte $\mathcal{M}^{o\leq}_{\mathcal{P}}$ s'il existe une borne $o \leq f_i(P_{bi})$ de la carte telle que :

— soit la borne $o \leq f(P_b)$ est le projeté de $o \leq f_i(P_{bi})$ avec $P_{bi} = P_b \cup \{c_i\}$ selon la définition 29 ;

— soit la borne $o \leq f_i(P_{bi})$ est le projeté de $o \leq f(P_b)$ avec $P_{bi} = P_b \setminus \{c_i\}$ selon la définition 29.

.

Exemple 22. La borne (3.20) appartient à la carte $\mathcal{M}^{\overline{M}\leq}_{\{n,S\}}$ si elle respecte la *propriété de projection* selon la définition 34 et si elle est précise pour la table $\mathcal{T}^{\overline{M}\leq}_{\{n,S\},30}$. Vérifions d'abord si elle respecte la propriété de projection. Pour la carte $\mathcal{M}^{\overline{M}\leq}_{\{n,S\}}$, le Bound Seeker fournit (3.21) comme conjecture de maximalité à utiliser pour éliminer la caractéristique S dans la conjecture de borne (3.20) afin d'en dériver la conjecture (3.22) également fournie par le Bound Seeker.

$$S = n^2 \tag{3.21}$$

$$\overline{M} \leq n \tag{3.22}$$

Lorsque nous effectuons l'élimination de S dans (3.20) en substituant S par n^2, nous obtenons

$$\left(\overline{M} \leq \left\lfloor \frac{1+\sqrt{4\cdot(S-n)+1}}{2} \right\rfloor \text{ et } S = n^2\right) \Rightarrow \overline{M} \leq \left\lfloor \frac{1+\sqrt{4\cdot(n^2-n)+1}}{2} \right\rfloor \tag{3.23}$$

$$\Rightarrow \overline{M} \leq \left\lfloor \frac{1+\sqrt{4\cdot n^2-4\cdot n+1}}{2} \right\rfloor \tag{3.24}$$

$$\Rightarrow \overline{M} \leq \left\lfloor \frac{1+\sqrt{(2\cdot n-1)^2}}{2} \right\rfloor \tag{3.25}$$

$$\Rightarrow \overline{M} \leq \left\lfloor \frac{1+2\cdot n-1}{2} \right\rfloor \tag{3.26}$$

$$\Rightarrow \overline{M} \leq n \tag{3.27}$$

Donc la conjecture de borne (3.20) vérifie bien la propriété de projection. Cependant, cette borne n'est pas précise. En effet, dans la table $\mathcal{T}^{\overline{M}\leq}_{\{n,S\},30}$, nous avons l'instance d'une partition d'un ensemble de $n = 8$ éléments, telle que la somme des carrés des tailles des partitions est $S = 32$. La taille maximale d'une partition d'une telle instance est $\overline{M} = 4$. Cependant la taille maximale fournie par (3.20) est $\left\lfloor \frac{1+\sqrt{4\cdot(32-8)+1}}{2} \right\rfloor = 5$ au lieu de 4. Ainsi la borne supérieure fournie par (3.20) est un peu plus grande que la borne exacte.

Cependant, cette imprécision n'est pas surprenante. En effet, l'instance de la partition d'un ensemble de $n = 8$ éléments ou la somme des carrés des tailles des partitions est $S = 32$, et ou la taille de la plus grande partition est $\overline{M} = 4$, n'est pas optimale pour la valeur minimale de la caractéristique S. La table originale $\mathcal{T}^{S\leq}_{\{n,\overline{M}\},30}$ à partir de laquelle le Bound Seeker a généré

la borne $S \geq \overline{M}^2 - \overline{M} + n$, fournit la valeur minimale de la somme des carrés des tailles des partitions S pour chaque combinaison de valeurs des caractéristiques n et $\overline{M}$. Or, lorsque $n = 8$ et $\overline{M} = 4$, la valeur minimale de la somme des carrés des tailles des partitions est $S = 20$ au lieu de $S = 32$. De plus, si nous utilisons la formule de la borne inverse $\left\lfloor \frac{1 + \sqrt{4 \cdot (S - n) + 1}}{2} \right\rfloor$ pour calculer la valeur maximale de $\overline{M}$ lorsque $n = 8$ et $S = 20$ au lieu de $S = 32$, on trouve bien $\left\lfloor \frac{1 + \sqrt{4 \cdot (20 - 8) + 1}}{2} \right\rfloor = 4$. On conclut donc que la borne inverse est précise pour le sous-ensemble d'instances telles que la valeur de S est minimale pour les couples valides de valeurs de n et $\overline{M}$. C'est-à-dire que pour les instances $(n = 8, S = 32, \overline{M} = 4)$ et $(n = 8, S = 20, \overline{M} = 4)$, la borne inverse est précise pour $(n = 8, S = 20, \overline{M} = 4)$, car cette dernière a une valeur minimale de $S = 20$. C'est d'ailleurs cette valeur minimale de S qui est fournie par la borne $S \geq \overline{M}^2 - \overline{M} + n$ que nous avons inversée pour obtenir $\overline{M} \leq \left\lfloor \frac{1 + \sqrt{4 \cdot (S - n) + 1}}{2} \right\rfloor$.

Mais bien que la conjecture de borne ne soit pas précise, elle est à considérer, car non seulement elle respecte la propriété de projection, mais de surcroît, le Bound Seeker ne trouve pas de borne précise pour la table $\mathcal{T}^{\overline{M}\leq}_{\{n,S\},30}$, exprimant la borne supérieure précise de $\overline{M}$ en fonction de n et S.

Pour illustrer un autre aspect bénéfique de la méthode d'inversion, nous présentons l'exemple 23

Exemple 23. Partant à nouveau de la borne ❷ $a \leq \underline{c}^2 + (v - \underline{c})^2$ de la carte $\mathcal{M}^{a\leq}_{\{v,\underline{c}\}}$ de la figure 2.2, nous calculons l'inversion

$$\underline{c} \ op \ g(a, v) \text{ avec } op \in \{\leq, \geq\} \tag{3.28}$$

En utilisant la même méthodologie que l'exemple 21, nous avons

$$❷ \ a \leq \underline{c}^2 + (v - \underline{c})^2 \Longleftrightarrow 0 \leq \underline{c}^2 + (v - \underline{c})^2 - a. \tag{3.29}$$

Ainsi, nous identifions le polynôme de second degré de variable $\underline{c}$ dont il faut étudier le signe.

$$P_1(\underline{c}) = 2 \cdot \underline{c}^2 - 2 \cdot v \cdot \underline{c} + v^2 - a \tag{3.30}$$

avec

$$P_1(\underline{c}) \geq 0 \tag{3.31}$$

Le discriminant de $P_1(\underline{c})$ est $\Delta = 8 \cdot a - 4 \cdot v^2$. C'est un discriminant dont le signe varie en fonction des combinaisons valides des valeurs fixées des caractéristiques a et v. Ce discriminant n'est pas négatif si et seulement si $2 \cdot a > v^2$. Donc, on ne peut calculer les racines distinctes

que lorsque la condition de faisabilité $2 \cdot a > v^2$ est vérifiée. Ainsi, sous la condition $2 \cdot a > v^2$, nous avons les racines suivantes :

$$2 \cdot a > v^2 \Longrightarrow \left(\underline{c}_1 = \frac{v - \sqrt{2 \cdot a - v^2}}{2} \quad \text{et} \quad \underline{c}_2 = \frac{v + \sqrt{2 \cdot a - v^2}}{2} \right) \tag{3.32}$$

Par conséquent, nous avons le tableau de signes suivant :

$\underline{c}$	$-\infty$		$\underline{c}_1$		$\underline{c}_2$		$+\infty$
$P_1(\underline{c})$	signe de 2		0	signe opposé de 2	0	signe de 2	

Le tableau de signes nous indique que $P_1(\underline{c})$ est non négatif, c'est-à-dire que $P_1(\underline{c}) \geq 0$ lorsque $\underline{c} \leq \underline{c}_1$ ou $\underline{c} \geq \underline{c}_2$ sous la condition que $2 \cdot a > v^2$. La racine $\underline{c}_2$ est positive, car elle est la somme des deux termes positifs $\frac{v}{2}$ et $\frac{\sqrt{2 \cdot a - v^2}}{2}$. La racine $\underline{c}_1$ est aussi positive, car nous avons

$$v^2 \geq a \Longleftrightarrow 2 \cdot v^2 \geq 2 \cdot a \tag{3.33}$$

$$\Longleftrightarrow v^2 \geq 2 \cdot a - v^2 \tag{3.34}$$

$$\text{Et comme } 2 \cdot a > v^2 \text{ nous avons donc } v \geq \sqrt{2 \cdot a - v^2} \tag{3.35}$$

$$\Rightarrow v - \sqrt{2 \cdot a - v^2} \geq 0 \tag{3.36}$$

$$\Rightarrow \frac{v - \sqrt{2 \cdot a - v^2}}{2} \geq 0 \tag{3.37}$$

$$\underline{c}_1 \geq 0 \tag{3.38}$$

Cela nous conduit à la relation de disjonction (3.39) qui est vérifiée lorsque la condition $2 \cdot a > v^2$ est vraie.

$$2 \cdot a > v^2 \Longrightarrow \left(\underline{c} \leq \frac{v - \sqrt{2 \cdot a - v^2}}{2} \quad \text{ou} \quad \underline{c} \geq \frac{v + \sqrt{2 \cdot a - v^2}}{2} \right) \tag{3.39}$$

Et comme $\underline{c}$ ne prend que des valeurs entières, alors nous obtenons la relation de disjonction (3.40) suivante :

$$2 \cdot a > v^2 \Longrightarrow \left(\underline{c} \leq \left\lfloor \frac{v - \sqrt{2 \cdot a - v^2}}{2} \right\rfloor \quad \text{ou} \quad \underline{c} \geq \left\lceil \frac{v + \sqrt{2 \cdot a - v^2}}{2} \right\rceil \right) \tag{3.40}$$

Cependant, cette relation de disjonction est incomplète, car elle ne fournit pas le cas où la condition $2 \cdot a > v^2$ est fausse. Par exemple, lorsque le nombre de sommets du graphe orienté est $v = 3$ et son nombre d'arcs est $a = 3$, la condition $2 \cdot a > v^2$ est fausse, car nous avons $2 \cdot a = 6 < 9 = v^2$. L'inversion ne nous fournit pas de borne de $\underline{c}$ dans ce cas.

Nous pouvons considérer la disjonction (3.40) comme une condition de faisabilité ; en effet, elle fournit des informations sur les tables $\mathcal{T}^{\underline{c} \leq}_{\{v,a\},26}$ et $\mathcal{T}^{\underline{c} \geq}_{\{v,a\},26}$ qui nous sont difficiles, voir

impossibles de générer à cause des problèmes de disponibilité d'espace mémoire évoqués à la discussion du chapitre 2. De plus, même si elle n'exprime pas une borne telle que définie selon la définition 27, elle délimite l'ensemble des valeurs de $\underline{c}$ en fonction de a et v.

Comme les bornes inverses dérivent des bornes liées dans une carte de bornes précises, il est naturel que les bornes inverses soient aussi liées. C'est ainsi que nous définissons par la suite la notion de *carte inverse*.

Définition 35 (carte inverse)**.** Étant donné $op_1, op_2 \in \{\leq, \geq\}$, un ensemble de caractéristiques bornantes $\mathcal{P}$, une caractéristique bornante $u \in \mathcal{P}$ et un ensemble $\mathcal{B}$ de caractéristiques bornées telles que $\mathcal{B} \cap \mathcal{P} = \emptyset$. Soit $\mathcal{M}_{\mathcal{P}}^{o\ op_1}$ avec $o \in \mathcal{B}$ des cartes ayant des bornes précises telles que u est une caractéristique bornante. Soit $\mathcal{P}_{inv} = \{o\} \cup (\mathcal{P} \setminus \{u\})$. Nous appelons *carte inverse*, notée $\mathcal{W}_{\mathcal{P}_{inv}}^{u\ op_2}$ pour la caractéristique u, à partir des cartes $\mathcal{M}_{\mathcal{P}}^{o\ op_1}$ avec $o \in \mathcal{B}$, un graphe tel que :

— chaque nœud constitue chaque borne inverse pour la caractéristique u de chaque borne précise des cartes $\mathcal{M}_{\mathcal{P}}^{o\ op_1}$ telle que u fait partie des caractéristiques bornantes de la borne précise ;
— un arc connecte une borne inverse f_1^{inv} vers une borne inverse f_2^{inv} dans la carte $\mathcal{W}_{\mathcal{P}_{inv}}^{u\ op_2}$ si un arc connecte la borne précise f_1 vers la borne précise f_2 dans une carte $\mathcal{M}_{\mathcal{P}}^{o\ op_1}$ sachant que f_1^{inv} et f_2^{inv} sont respectivement les bornes inverses de f_1 et f_2 pour la caractéristique u.

Dans ce cas, les cartes $\mathcal{M}_{\mathcal{P}}^{o\ op_1}$ avec $o \in \mathcal{B}$ sont appelées *cartes inversées*.

Comme exemple, nous avons $\mathcal{W}_{\{S,\overline{M}\}}^{n\leq} = \mathcal{M}_{\{S,\overline{M}\}}^{n\leq}$. C'est-à-dire que $\mathcal{M}_{\{S,\overline{M}\}}^{n\leq}$ est une carte inverse de la carte $\mathcal{M}_{\{n,\overline{M}\}}^{S\geq}$ comme cela est illustré à la figure 3.1. Nous avons également la figure 3.2 qui présente la carte inverse $\mathcal{W}_{\{n,\underline{M},S\}}^{P\leq}$ pour la caractéristique P obtenue en inversant les deux cartes $\mathcal{M}_{\{n,P,\underline{M}\}}^{S\leq}$ et $\mathcal{M}_{\{n,P\}}^{n\leq}$. Nous faisons la remarque selon laquelle il faut utiliser plusieurs cartes de bornes précises pour obtenir une carte inverse.

3.1.1 Liens de projection entre bornes inverses

Nous avons fait le constat de la conservation de la propriété de projection dans l'exemple 20 et l'exemple 21. Cela met en évidence l'existence de *liens directs de projection* entre deux bornes inverses des deux bornes initiales dont l'une est le projeté de l'autre. Nous insistons sur le fait que le lien de projection est *direct*, car dans la définition 35 d'une carte inverse nous utilisons les *liens de projection* de la carte inversée pour définir les liens dans la carte inverse. En effet, ce lien direct est confirmé par le théorème 3 de l'*inversion d'une projection* dans le cas des bornes polynomiales de second degré.

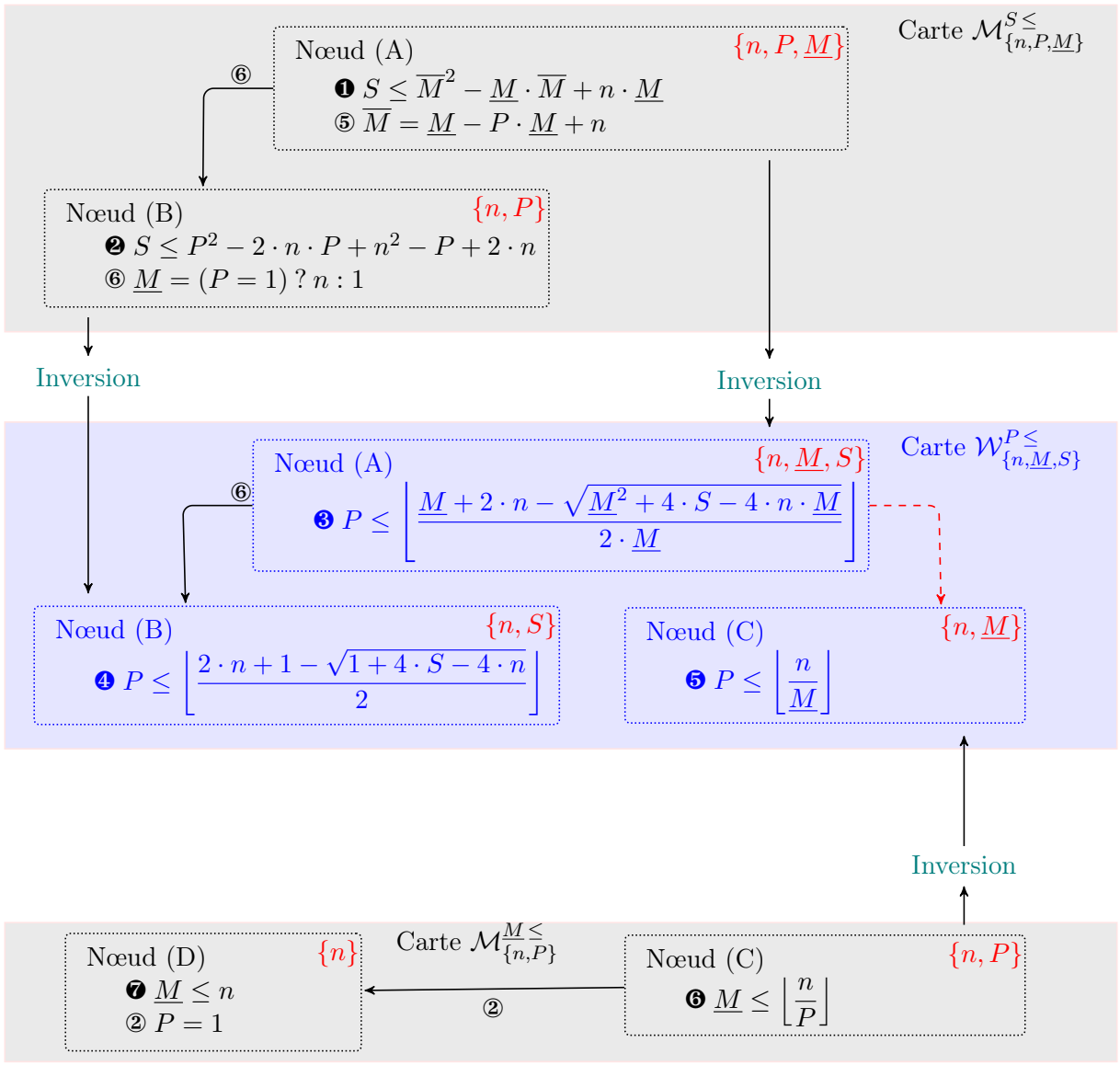


FIGURE 3.2 – Exemple de carte inverse. Les nœuds connectés de couleur bleue illustrent la carte inverse $\mathcal{W}^{P\leq}_{\{n,\underline{M},S\}}$ pour la caractéristique P obtenue en inversant les deux cartes $\mathcal{M}^{S\leq}_{\{n,P,\underline{M}\}}$ et $\mathcal{M}^{\underline{M}\leq}_{\{n,P\}}$. P est le nombre de partitions, n est le cardinal de l'ensemble partitionné, $\underline{M}$ est la taille minimale d'une partition, $\overline{M}$ est la taille maximale d'une partition, S est la somme des carrés des tailles des partitions. Remarquons l'arc en pointillés de la carte $\mathcal{W}^{P\leq}_{\{n,\underline{M},S\}}$. Elle exprime le fait qu'il n'existe pas de lien de projection ou encore de conjecture de maximalité $S = g_S(n, \underline{M})$ permettant de projeter la borne ❸ sur la borne ❺. Cela est dû au fait que S ne dépend pas fonctionnellement de n et $\underline{M}$ lorsque P est maximal. Par exemple, pour la combinaison $n = 11, \underline{M} = 3$, la valeur maximale de $P = 3$. Cependant, il y a deux valeurs possibles de S qui sont 43 et 41.

Définition 36. Soit o une caractéristique à borner en utilisant P_b comme sous-ensemble de caractéristiques bornantes d'un objet combinatoire. Soit x une caractéristique bornante contenue dans P_b. Nous notons par P_x l'ensemble des caractéristiques ne contenant pas x.

C'est-à-dire que $P_x = P_b \setminus \{x\}$. Soit $A(P_x), B(P_x), C(P_x)$ trois fonctions à valeurs entières (de $\prod_{c \in P_x} \mathcal{D}(c)$ vers $\mathbb{Z}$) qui s'expriment en fonction des caractéristiques de P_x.

Nous appelons polynôme $P(x)$ d'une variable x à coefficients variables $A(P_x), B(P_x), C(P_x)$, l'expression (3.41).

$$P(x) = A(P_x) \cdot x^2 + B(P_x) \cdot x + C(P_x) \tag{3.41}$$

Nous appelons l'expression (3.42) une *borne polynomiale de second degré* pour la caractéristique x.

$$o \ op_1 \ A(P_x) \cdot x^2 + B(P_x) \cdot x + C(P_x) \text{ avec } op_1 \in \{\leq, \geq\} \tag{3.42}$$

Théorème 3 (Inversion d'une projection de borne polynomiale)**.** *Soit o une caractéristique à borner et P_b un ensemble de caractéristiques bornantes d'un objet combinatoire. Soit p et x deux caractéristiques bornantes de P_b. Soit les sous-ensembles de caractéristiques bornantes P_p et P_x définis tels que $P_x = P_b \setminus \{x\}$ et $P_p = P_b \setminus \{p, x\}$. Soit $g_p : \prod_{c \in P_p} \mathcal{D}(c) \longrightarrow \mathcal{D}(p)$ une fonction telle que $p = g_p(P_p)$.*

Soit la borne polynomiale (3.43) de second degré en x

$$o \ op_1 \ A(P_x) \cdot x^2 + B(P_x) \cdot x + C(P_x) \ \textit{avec} \ op_1 \in \{\leq, \geq\} \tag{3.43}$$

qui admet la borne inverse (3.44) pour x

$$x \ op_2 \ \boldsymbol{f}\left(\frac{-B(P_x) \ op_3 \ \sqrt{\Delta}}{2 \cdot A(P_x)}\right) \ \textit{avec} \begin{cases} \Delta = B^2(P_x) - 4 \cdot A(P_x) \cdot (C(P_x) - o) \\ (op_2, \boldsymbol{f}) \in \{(\leq, \lfloor \cdot \rfloor), (\geq, \lceil \cdot \rceil)\} \\ op_3 \in \{+, -\} \end{cases} \tag{3.44}$$

Alors, l'inverse (pour x) du projeté de (3.43) par élimination de p à travers l'égalité $p = g_p(P_p)$, est le projeté de l'inverse (pour x) de (3.43), c'est-à-dire le projeté de (3.44) par élimination de p à travers l'égalité $p = g_p(P_p)$.

Démonstration. D'après leurs définitions, P_x et P_p ne contiennent pas la caractéristique x. Ainsi, seuls les coefficients $A(P_x), B(P_x)$ et $C(P_x)$ sont affectés par la projection de (3.43) en éliminant p à travers l'égalité $p = g_p(P_p)$ et indépendamment de la caractéristique x. Dans cette projection, les coefficients $A(P_x), B(P_x)$ et $C(P_x)$ sont respectivement transformés en $A_p(P_p), B_p(P_p)$ et $C_p(P_p)$ tels que nous avons :

$$\begin{cases} A_p(P_p) = A(P_p \cup \{p\}) \\ B_p(P_p) = B(P_p \cup \{p\}) \\ C_p(P_p) = C(P_p \cup \{p\}) \end{cases} \text{avec } p = g_p(P_p) \tag{3.45}$$

Ainsi, le projeté de la borne polynomiale de second degré (3.43) en éliminant p à travers l'égalité $p = g_p(P_p)$ est

$$o \ op_1 \ A_p(P_p) \cdot x^2 + B_p(P_p) \cdot x + C_p(P_p) \text{ avec } op_1 \in \{\leq, \geq\} \tag{3.46}$$

qui correspond encore à

$$o \ op_1 \ A(P_p \cup \{p\}) \cdot x^2 + B(P_p \cup \{p\}) \cdot x + C(P_p \cup \{p\}) \text{ avec } \begin{cases} op_1 \in \{\leq, \geq\} \\ p = g_p(P_p) \end{cases} \tag{3.47}$$

Puis, d'après la borne inverse (3.44) pour x de (3.43), la borne inverse de (3.47) pour X est

$$x \ op_2 \ \boldsymbol{f}\left(\frac{-B(P_p \cup \{p\}) \ op_3 \ \sqrt{\Delta_p}}{2 \cdot A(P_p \cup \{p\})}\right)$$

$$\text{avec } \begin{cases} \Delta_p = B^2(P_p \cup \{p\}) - 4 \cdot A(P_p \cup \{p\}) \cdot (C(P_p \cup \{p\}) - o) \\ (op_2, \boldsymbol{f}) \in \{(\leq, \lfloor \cdot \rfloor), (\geq, \lceil \cdot \rceil)\} \\ op_3 \in \{+, -\} \\ p = g_p(P_p) \end{cases} \tag{3.48}$$

Nous pouvons bien constater que (3.48) est le projeté de la borne inverse (3.44) en éliminant p à travers l'égalité $p = g_p(P_p)$. Ce qui achève la preuve.

□

Nous donnons l'exemple 24 pour appuyer le théorème 3.

Exemple 24. Nous faisons le constat dans l'exemple de carte inverse $\mathcal{W}^{P\leq}_{\{n,\underline{M},S\}}$ de la figure 3.2 où la borne ❹ est le projeté de la borne ❸ par élimination de la caractéristique $\underline{M}$ à travers les égalités $\underline{M} = 1$ ou $\underline{M} = n$. Pour faire le parallèle avec le théorème 3, nous avons $x = P, P_x = \{\underline{M}, n\}, p = \underline{M}, g_p(P_p) = 1$ ou $g_p(P_p) = n$. En effet, nous avons :

$$\left(❸ \ P \leq \left\lfloor \frac{\underline{M} + 2 \cdot n - \sqrt{\underline{M}^2 + 4 \cdot S - 4 \cdot n \cdot \underline{M}}}{2 \cdot \underline{M}} \right\rfloor \text{ et } \underline{M} = 1\right) \tag{3.49}$$

qui conduit après substitution de $\underline{M}$ à

$$❹ \ P \leq \left\lfloor \frac{1 + 2 \cdot n - \sqrt{1 + 4 \cdot S - 4 \cdot n}}{2} \right\rfloor \tag{3.50}$$

Nous avons également le même constat :

$$❸ \left(P \leq \left\lfloor \frac{\underline{M} + 2 \cdot n - \sqrt{\underline{M}^2 + 4 \cdot S - 4 \cdot n \cdot \underline{M}}}{2 \cdot \underline{M}} \right\rfloor \text{ et } \underline{M} = n\right) \tag{3.51}$$

qui, sachant que $\underline{M} = n \iff S = n^2$ conduit après substitution de $\underline{M}$ à

$$P \leq \left\lfloor \frac{n + 2 \cdot n - \sqrt{n^2 + 4 \cdot n^2 - 4 \cdot n^2}}{2 \cdot n} \right\rfloor = 1 \tag{3.52}$$

Cependant, lorsque $S = n^2$, nous avons

$$\text{❹ } P \leq \left\lfloor \frac{1 + 2 \cdot n - \sqrt{1 + 4 \cdot S - 4 \cdot n}}{2} \right\rfloor \Leftrightarrow P \leq \left\lfloor \frac{1 + 2 \cdot n - \sqrt{1 + 4 \cdot n^2 - 4 \cdot n}}{2} \right\rfloor \tag{3.53}$$

$$\Leftrightarrow P \leq \left\lfloor \frac{1 + 2 \cdot n - \sqrt{(2 \cdot n - 1)^2}}{2} \right\rfloor \tag{3.54}$$

$$\Leftrightarrow P \leq \left\lfloor \frac{1 + 2 \cdot n - 2 \cdot n + 1)^2}{2} \right\rfloor = 1 \tag{3.55}$$

Remarque 4. Dans la figure 3.2, le théorème 3 ne s'applique pas entre les bornes ❸ et ❺, car la borne inverse ❺ $P \leq \left\lfloor \frac{n}{\underline{M}} \right\rfloor$ provient de la borne inversée ❸ $\underline{M} \leq \left\lfloor \frac{n}{P} \right\rfloor$ qui n'est pas une borne polynomiale de la forme (3.43). Il n'est donc pas surprenant qu'il n'existe pas de lien de projection entre ❸ et ❺ comme cela est mentionné par l'arc en pointillés sur la figure 3.2 et dans la description de ladite figure.

Ce théorème 3 montre de manière forte les liens qui existent entre conjectures de bornes d'une même caractéristique d'un objet combinatoire dans une carte et les liens qui existent entre plusieurs cartes d'un même objet combinatoire. Par la suite, nous mettons en évidence les liens qui existent entre cartes d'objets combinatoires distincts.

3.1.2 Avantages de la méthode d'inversion des bornes

Nous récapitulons les avantages de la méthode d'inversion illustrés à l'exemple 20, l'exemple 21 et l'exemple 23 :

L'inversion apporte une réponse au problème de génération des données erronées que nous avons évoqué à la section Discussion du chapitre 3. En effet, la borne ❹ de la carte $\mathcal{M}^{n\leq}_{\{S,\overline{M}\}}$ de la figure 3.1 obtenue par inversion de la borne ❶ de la carte $\mathcal{M}^{S\geq}_{\{n,\overline{M}\}}$ est la borne de la table des bornes numériques $\mathcal{T}^{n\leq}_{\{S,\overline{M}\},30}$. Cette table entre dans la catégorie des tables de données potentiellement erronées en utilisant la méthode de génération des données de la phase (A1) de l'exécution du Bound Seeker. Cette table est du même type que la table erronée $\mathcal{T}^{v\leq}_{\{t,\overline{t}\},4}$ présentée à la section de discussion du chapitre 2. Le type des tables dont il s'agit est celui dont la caractéristique de la taille de l'instance ne fait pas partie des caractéristiques d'entrée. Et donc l'inversion propose la borne ❹ sans l'utilisation de la table $\mathcal{T}^{n\leq}_{\{S,\overline{M}\},30}$ potentiellement erronée.

L'inversion apporte une réponse partielle à la préoccupation d'obtenir des cartes aussi complètes que possible puisque pour les tables où les composantes polynomiales et conditionnelles

du Bound Seeker ne trouvent pas de bornes précises, l'inversion propose des bornes qui, bien qu'elles ne soient pas précises, peuvent conserver la propriété de projection d'une borne sur une autre. En effet, l'exemple 21 montre cet aspect bénéfique de l'inversion et le théorème 3 vient consolider cela.

L'inversion apporte une réponse au problème d'indisponibilité des jeux de données d'apprentissage dû au manque d'espace mémoire qui est évoqué à la section Discussion du chapitre 3. L'exemple 23 propose des bornes des tables $\mathcal{T}^{c\leq}_{\{v,a\},26}$ et $\mathcal{T}^{c\geq}_{\{v,a\},26}$ qui, à cause de la caractéristique du nombre d'arcs a d'un graphe orienté, sont difficiles à générer en utilisant la méthode de la phase (A1) du Bound Seeker.

L'inversion suggère d'ajouter à la grammaire des formules possibles du Bound Seeker de nouveaux types de formules. En effet, dans le cas de l'inversion des polynômes de second degré, l'exemple 21 et l'exemple 23 suggèrent d'ajouter les fonctions unaires de racine carrée entière $\lfloor\sqrt{\cdot}\rfloor$ et $\lceil\sqrt{\cdot}\rceil$.

L'inversion apporte sa pierre d'édifice dans l'élaboration des preuves des conjectures autant sur le plan manuel que sur le plan automatique. En effet, d'après la définition 33, une borne et son inverse sont équivalents logiquement. Ainsi, prouver l'une c'est également prouver l'autre. Dans la figure 3.1, si nous prouvons la borne ❶, il devient facile de prouver les trois autres bornes ❷, ❶ et ❷, car elles s'obtiennent juste en appliquant du calcul algébrique sans se soucier des notions de maximum et minimum. Comme de nos jours, le calcul algébrique automatisé a subi un développement considérable au travers des systèmes comme Wolfram|Alpha ou Mathematica [42], nous envisageons de les utiliser plus tard dans un objectif en phase avec l'automatisation des preuves des conjectures de bornes inverses.

L'inversion permet de consolider et étendre sur le plan théorique la notion de carte d'invariants par le fait qu'à partir d'une carte de bornes précises, elle génère une carte de bornes qui potentiellement ne sont pas précises, mais peuvent conserver la notion de projection d'une borne sur une autre. Ce constat est confirmé par le théorème 3. Du fait que la borne inverse n'est pas toujours précise, cela nous conduit à définir ce qu'est une *carte inversée* afin de distinguer les cartes de bornes précises de celles qui potentiellement ont des bornes pas précises qui conservent la propriété de projection.

Ainsi, la définition 35, la figure 3.1 ainsi que la figure 3.2 illustrent bien l'autre apport bénéfique de l'inversion qu'est la mise en évidence de liens entre cartes d'invariants d'un même objet combinatoire.

De plus, acquérir de telles cartes inverses est beaucoup plus rapide que l'apprentissage des cartes à bornes précises. En effet, ayant constaté que le Bound Seeker ne trouvait pas de bornes précises pour un grand nombre de tables inverses des tables des bornes numériques dont la formule de la borne est un polynôme de second degré, nous avons automatisé l'inversion

dans le cadre des polynômes de second degré. Notons bien que des algorithmes de résolution d'inéquations paramétriques existent déjà dans des logiciels comme Wolfram|Alpha [42]. Nous utilisons des algorithmes semblables que nous avons reprogrammés dans le langage SICStus Prolog. Dans la suite, nous présentons l'algorithme d'inversion des polynômes de second degré.

3.1.3 Algorithme d'inversion des polynômes de second degré à coefficients variables

Soit o une caractéristique à borner en utilisant P_b comme sous-ensemble de caractéristiques bornantes d'un objet combinatoire. Soit x une caractéristique bornante contenue dans P_b. Nous notons par P_x l'ensemble des caractéristiques ne contenant pas x. C'est-à-dire que $P_x = P_b \setminus \{x\}$. Soit $A(P_x), B(P_x), C(P_x)$ trois fonctions à valeurs entières (de $\prod_{c \in P_x} \mathcal{D}(c)$ vers $\mathbb{Z}$) qui s'expriment en fonction des caractéristiques de P_x. Le Bound Seeker trouve un grand nombre de bornes polynomiales de second degré de la forme (3.56) :

$$o \ op_1 \ A(P_x) \cdot x^2 + B(P_x) \cdot x + C(P_x) \text{ avec } op_1 \in \{\leq, \geq\} \tag{3.56}$$

comme le montre par exemple la carte $\mathcal{M}^{a\leq}_{\{v,c,s,\overline{c},\underline{c},\overline{s},\underline{s}\}}$ de la figure 2.8. Cependant, il ne trouve pas de borne inverse correspondante (3.57)

$$x \ op_2 \ P_{inv}(\{o\} \cup P_x) \text{ avec } op_2 \in \{\leq, \geq\} \tag{3.57}$$

Pour donc déterminer (3.57), nous passons (3.56) à l'algorithme 1. Nous définissons les variables suivantes utilisées dans l'algorithme 1 :

- Soit e une instance disponible de l'objet combinatoire. C'est-à-dire que nous avons e qui représente une ligne de la table de bornes numériques $\mathcal{T}^{o\ op_1}_{P_b,n_{max}}$ qui est la table à partir de laquelle le Bound Seeker génère la borne polynomiale à inverser (3.56).
- Soit $o(e), P_x(e)$ les valeurs respectives prises par les caractéristiques o et P_x pour chaque instance $e \in \mathcal{T}^{o\ op_1}_{P_b,n_{max}}$.
- Soit $\Delta(e) = B^2(P_x(e)) - 4 \cdot A(P_x(e)) \cdot (C(P_x(e)) - o(e))$ la valeur du discriminant Δ du polynôme $P(x) - o$ pour chaque instance $e \in \mathcal{T}^{o\ op_1}_{P_b,n_{max}}$.
- Soit $x(e)$ la valeur de la caractéristique x pour chaque instance $e \in \mathcal{T}^{o\ op_1}_{P_b,n_{max}}$.

L'algorithme 1 suit le schéma de résolution de l'exemple 21 et l'exemple 23. C'est-à-dire qu'aux lignes 1 et 2, il commence par vérifier qu'il existe une entrée e de la table $\mathcal{T}^{o\ op_1}_{P_b,n_{max}}$ telle que le discriminant du polynôme $P(x) - o$ est strictement positif. Ensuite, à la ligne 3, il vérifie que le coefficient $A(P_x)$ est soit seulement strictement positif, ou soit seulement strictement négatif pour toutes les entrées de la table $\mathcal{T}^{o\ op_1}_{P_b,n_{max}}$. À la ligne 4, il calcule la valeur $CoefA$ du coefficient $A(P_x)$ pour une entrée $e \in \mathcal{T}^{o\ op_1}_{P_b,n_{max}}$. Puis, aux lignes 5 et 6, il mémorise les formules de calcul des deux racines $racine_1(e)$ et $racine_2(e)$ pour chaque entrée $e \in \mathcal{T}^{o\ op_1}_{P_b,n_{max}}$. Puis, les

lignes 7 et 8 ou bien les lignes 17 et 18 testent le signe du $CoefA$ et du sens op_1 de la borne à inverser. Puis, il retourne (3.58) ou (3.59).

En effet, nous avons les cas suivants :

1. Si $CoefA < 0$, nous avons le tableau de signes suivant :

x	$-\infty$		$racine_1$		$racine_2$		$+\infty$
$P(x) - o$		-	0	+	0	-	

   Dans ce cas :

   — si $op_1 \in \{\geq\}$, alors on s'intéresse à la partie négative de $P(x) - o$ et donc l'algorithme 1 retourne

$$\Delta > 0 \Rightarrow (x \leq racine_1) \vee (x \geq racine_2) \tag{3.58}$$

   — si $op_1 \in \{\leq\}$, alors on s'intéresse à la partie positive de $P(x) - o$ et donc l'algorithme 1 retourne

$$\Delta > 0 \Rightarrow (x \geq racine_1) \wedge (x \leq racine_2) \tag{3.59}$$

2. Si $CoefA > 0$, nous avons le tableau de signes suivant :

x	$-\infty$		$racine_1$		$racine_2$		$+\infty$
$P(x) - o$		+	0	-	0	+	

   Dans ce cas, l'algorithme 1 permute la sortie précédente. C'est-à-dire qu'il retourne (3.59) si $op_1 \in \{\geq\}$ ou (3.58) si $op_1 \in \{\leq\}$.

Les lignes 9, 13, 18 et 22 de l'algorithme 1 assurent que ces conjectures sont valides sur toutes les entrées de la table $\mathcal{T}^{o\ op_1}_{P_b,n_{max}}$ pour lesquelles le discriminant du polynôme $P(x) - o$ est strictement positif. Par conséquent, si le discriminant est strictement positif sur toutes les entrées de la table $\mathcal{T}^{o\ op_1}_{P_b,n_{max}}$ alors la conjecture de borne s'écrit sans la condition $\Delta > 0$ comme dans l'exemple 21. Si dans le cas contraire, le discriminant n'est strictement positif que sur certaines entrées de la table $\mathcal{T}^{o\ op_1}_{P_b,n_{max}}$ comme dans l'exemple 23, alors la conjecture est *incomplète* lorsqu'elle est valide sur ces entrées. Ainsi, il va falloir déterminer ce que vaut la conjecture sur les entrées restantes. Notons aussi qu'il peut exister des entrées dans la table inverse $\mathcal{T}^{x\ op_2}_{P_b\cup\{o\}\setminus\{x\},n_{max}}$ qui ne sont pas dans la table à inverser $\mathcal{T}^{o\ op_1}_{P_b,n_{max}}$ et pour lesquelles la conjecture de borne inverse n'est pas précise comme l'illustre l'exemple 22. Pour ces entrées, il va aussi falloir déterminer la conjecture de la borne précise. Pour toutes ces entrées pour

lesquelles la borne n'est pas définie ou bien imprécise, nous envisageons d'appliquer directement le Bound Seeker.

Remarque 5. Lorsque le discriminant Δ est nul sur certaines entrées de la table $\mathcal{T}^{o\ op_1}_{P_b,n_{max}}$, alors le polynôme $P(x) - o$ n'a qu'une seule racine. Dans ce cas, ce polynôme n'a qu'un seul signe. Ainsi, il n'y a pas de changement de signe dans son tableau de signes et par conséquent l'algorithme ne peut pas fournir de conjecture de borne pour ces entrées. Pour cette raison, elles font partie des entrées où nous envisageons d'appliquer directement le Bound Seeker.

Algorithme 1 : Inversion de : $(o\ op_1\ A(P_x)\cdot x^2 + B(P_x)\cdot x + C(P_x))$

Entrée : $op_1, A(P_x), B(P_x), C(P_x), \mathcal{T}^{o\ op_1}_{P_b,n_{max}}$

Sortie : $(x\ op_2\ P_{inv}(\{o\} \cup P_x))$

if $\forall\, e \in \mathcal{T}^{o\ op_1}_{P_b,n_{max}},\ B^2(P_x(e)) - 4\cdot A(P_x(e))\cdot(C(P_x(e)) - o(e)) \leq 0$ **then**
  return $\emptyset$;
$CoefA \leftarrow (\exists\, e_1, e_2 \in \mathcal{T}^{o\ op_1}_{P_b,n_{max}}, [A(P_x(e_1)) = 0] \vee [A(P_x(e_1)) > 0 \wedge A(P_x(e_2)) < 0]\ ?\ 0 : 1)$;
Soit $e \in \mathcal{T}^{o\ op_1}_{P_b,n_{max}},\ CoefA \leftarrow CoefA \times A(P_x(e))$;
Soit $\Delta = B^2(P_x) - 4\cdot A(P_x)\cdot(C(P_x) - o)$;
$racine_1 \leftarrow \dfrac{-B(P_x) - \sqrt{\Delta}}{2\cdot A(P_x)}$ et $racine_2 \leftarrow \dfrac{-B(P_x) + \sqrt{\Delta}}{2\cdot A(P_x)}$.
$\forall e \in \mathcal{T}^{o\ op_1}_{P_b,n_{max}},\ racine_1(e) = \dfrac{-B(P_x(e)) - \sqrt{\Delta(e)}}{2\cdot A(P_x(e))}$ et $racine_2(e) = \dfrac{-B(P_x(e)) + \sqrt{\Delta(e)}}{2\cdot A(P_x(e))}$.
if $CoefA < 0$ **then**
  if $op_1 \in \{\geq\}$ **then**
    if $\exists\, e \in \mathcal{T}^{o\ op_1}_{P_b,n_{max}}$ *avec* $racine_1(e) < x(e) < racine_1(e)$ *et* $\Delta(e) > 0$ **then**
      return $\emptyset$;
    return $\Delta > 0 \Rightarrow (x \leq racine_1) \vee (x \geq racine_2)$;
  if $op_1 \in \{\leq\}$ **then**
    if $\exists\, e \in \mathcal{T}^{o\ op_1}_{P_b,n_{max}}$ *avec* $x(e) < racine_1(e) \vee x(e) > racine_1(e)$ *et* $\Delta(e) > 0$ **then**
      return $\emptyset$;
    return $\Delta > 0 \Rightarrow (x \geq racine_1) \wedge (x \leq racine_2)$;
if $CoefA > 0$ **then**
  if $op_1 \in \{\geq\}$ **then**
    if $\exists\, e \in \mathcal{T}^{o\ op_1}_{P_b,n_{max}}$ *avec* $x(e) < racine_1(e) \vee x(e) > racine_1(e)$ *et* $\Delta(e) > 0$ **then**
      return $\emptyset$;
    return $\Delta > 0 \Rightarrow (x \geq racine_1) \wedge (x \leq racine_2)$;
  if $op_1 \in \{\leq\}$ **then**
    if $\exists\, e \in \mathcal{T}^{o\ op_1}_{P_b,n_{max}}$ *avec* $racine_1(e) < x(e) < racine_1(e)$ *et* $\Delta(e) > 0$ **then**
      return $\emptyset$;
    return $\Delta > 0 \Rightarrow (x \leq racine_1) \vee (x \geq racine_2)$;
return $\emptyset$;

3.1.4 Expérimentation de l'inversion automatique sur huit objets combinatoires

Nous avons appliqué l'algorithme 1 sur les conjectures de bornes des huit objets combinatoires que nous étudions. Ces conjectures étant générées au préalable par le Bound Seeker. L'algorithme 1 a été implémenté en SICStus Prolog 4.6.0. Avant d'appliquer l'algorithme 1 aux conjectures du Bound Seeker, nous avons élargi son champ d'application aux bornes qui ont bien le format de la borne polynomiale de second degré (3.56), mais avec un terme paramétré qui remplace la caractéristique x. Le terme paramétré est tel que l'un des paramètres est la caractéristique x. Le terme paramétré est généralement binaire ou ternaire. L'extension du champ d'application de l'algorithme 1 se fait en substituant le terme paramétré par un majorant ou un minorant qui sied au calcul algébrique de l'algorithme 1. Le choix entre le minorant et le majorant se fait en fonction du sens de la borne et du signe du coefficient des monômes du format (3.56). L'exemple 25 illustre un tel format avec un terme paramétré binaire et l'extension correspondante.

Exemple 25. Soit le graphe orienté dont le nombre d'arcs est a, le nombre de sommets est v, le nombre de composantes connexes est c et la taille de la plus grande composante connexe est $\overline{c}$. Alors le Bound Seeker génère la conjecture (3.60). Cette conjecture respecte le format de la borne polynomiale de second degré (3.56) si l'on prend le terme binaire $(v \bmod \overline{c})$ comme la caractéristique x.

Pour inverser (3.60) afin de trouver une borne de $\overline{c}$ en fonction de a, c et v, nous commençons par remplacer $(v \bmod \overline{c})$ par son minorant 0 ou par son majorant $\overline{c} - 1$. Comme le sens de la borne est la borne supérieure, nous remplaçons $(v \bmod \overline{c})$ par son majorant lorsque le coefficient du monôme correspondant est positif et par son minorant dans le cas contraire. Ainsi, $(v \bmod \overline{c})^2$ est remplacé par $(\overline{c} - 1)^2$ alors que $-c \cdot (v \bmod \overline{c})$ est remplacé par 0. C'est ainsi que nous obtenons l'extension (3.61) à laquelle nous pouvons appliquer l'algorithme 1.

$$a \leq (v \bmod \overline{c})^2 - c \cdot (v \bmod \overline{c}) + v \cdot \overline{c} \tag{3.60}$$

$$a \leq (\overline{c} - 1)^2 + v \cdot \overline{c} \tag{3.61}$$

Remarquons que, bien que l'extension obtenue soit une borne valide, elle n'est plus précise puisque nous remplaçons le terme $(v \bmod \overline{c})$ par deux valeurs distinctes dans une même formule, alors que le terme $(v \bmod \overline{c})$ ne devait prendre qu'une seule valeur. Pour répondre à ce problème, nous envisageons d'utiliser plus tard la méthode d'Araya et *al.* [6] consistant à étudier la monotonie de la fonction $f(x) = x^2 - c \cdot x + v \cdot \overline{c}$ lorsque nous posons $x = v \bmod \overline{c}$. Cependant, pour étudier la monotonie, il faut calculer la dérivée de $f(x)$ par rapport à x afin de connaître les domaines de variations de $f(x)$. Or, notons que le terme $v \cdot \overline{c}$ de $f(x)$ dépend

de x. Donc, lorsque x varie, non seulement le terme $v \cdot \overline{c}$ varie, mais même la façon dont ce terme varie en fonction de x est indéterminée.

La table 3.1 récapitule les résultats de l'expérience. Elle illustre que, sur 282 conjectures polynomiales de second degré, 209 ont été inversées avec succès dans un temps de 5 secondes. Donc la méthode est très rapide pour trouver des conjectures de bornes. 73 conjectures n'ont pas pu être inversées parce que, soit la caractéristique correspondante x que nous voulons borner ne peut pas être bornée par les caractéristiques bornantes impliquées dans l'inversion, soit la structure de la formule à inverser n'est pas celle traitée par l'algorithme 1. L'exemple 26 illustre une conjecture de borne qui n'a pu être inversée.

Exemple 26. Soit la séquence de booléens dont les caractéristiques sont la longueur maximale d_{max} d'une interdistance, le nombre d'interdistances de longueur maximale od_{max}, la somme des carrés sd_{sq} des longueurs des interdistances de la séquence et la plus petite interdistance d_{minmax} lorsque les interdistances n'ont pas la même longueur. Notons qu'elle vaut 0 lorsque les interdistances ont toutes la même longueur. Le Bound Seeker fournit la borne (3.62) que l'algorithme 1 n'a pas pu inverser pour la caractéristique d_{max}.

$$sd_{sq} \leq od_{max} \cdot d_{max}^2 + d_{minmax}^2 \tag{3.62}$$

En effet, cette borne n'est pas inversible pour d_{max} car le coefficient od_{max} du monôme de second degré $od_{max} \cdot d_{max}^2$ est parfois nul, notamment lorsqu'il n' y a pas d'interdistance dans la séquence de booléen. Par exemple la séquence 111100 n'a pas d'interdistance, car cette séquence n'a qu'un seul groupe, c'est-à-dire le groupe 1111.

Les conjectures des bornes dans le cas de l'arbre enraciné, la forêt enracinée avec sommet isolé et celle sans sommet isolé n'ont pas été inversées, car ces bornes ne respectent pas le format des bornes polynomiales de second degré (3.56). L'exemple 27 illustre un exemple de telles conjectures de bornes, qui requiert plus d'opérations algébriques que celles implémentées par l'algorithme 1.

Exemple 27. Soit une forêt enracinée avec sommets isolés dont les caractéristiques sont la profondeur minimale $\underline{p}$ d'une feuille, la taille minimale $\underline{t}$ d'un arbre de la forêt, le nombre minimal $\underline{f}$ de feuilles d'un arbre de la forêt et le nombre de sommets v de la forêt. Alors $\underline{p}$ est borné inférieurement selon la conjecture (3.63) générée par le Bound Seeker. L'algorithme 1 a besoin de plus de règles de calcul algébrique pour inverser (3.63) pour la caractéristique $\underline{f}$.

$$\underline{p} \geq \left\lceil \frac{\underline{t} - 1}{\underline{f}^2 + v \cdot \underline{f} - \underline{t}} \right\rceil \tag{3.63}$$

Objet combinatoire	Nombre de conjectures polynomiales de second degré	Nombre de formules trouvées par inversion	Temps mis pour l'inversion
Graphe orienté	32	29	1s
Arbre enraciné	0	0	/
Forêt enracinée avec sommet isolé	0	0	/
Forêt enracinée sans sommet isolé	0	0	/
Partitions non vides	13	12	1s
Partitions avec ensemble vide	8	8	1s
Séquence de booléens	132	94	1s
Séquence cyclique de booléens	98	67	1s
Total	283	210	5s

TABLE 3.1 – Résultats d'acquisition automatique d'inverses des bornes polynomiales de second degré pour chacun des objets combinatoires.

Les bornes ❸ et ❹ de la carte inverse en bleu de la figure 3.2 sont des exemples d'inversions générées par l'algorithme 1 dans le cas des partitions d'ensembles. Elles ont également été retrouvées par Wolfram|Alpha [42] en lui donnant manuellement en entrée les bornes ❶ et ❷ de la même figure 3.2.

3.2 Correspondance entre objets combinatoires

En informatique, il est commun de réduire une instance d'un problème A en une instance d'un problème B comme le font par exemple Stephen Cook [31] et Karp [44]. Ainsi, les techniques connues pour résoudre B peuvent être appliquées au problème A. Dans ce travail, nous nous inspirons de la méthode de réduction pour montrer des liens existants entre objets combinatoires. Pour ce faire, la méthode que nous employons est similaire à la méthode de réduction dans le sens qu'elle transforme les instances d'un objet combinatoire $\mathcal{O}_1$ en un sous-ensemble d'instances d'un autre objet combinatoire $\mathcal{O}_2$. Dans la suite, nous présentons ladite transformation que nous appelons *encodage d'un objet combinatoire par un autre.*

Comme plusieurs instances d'un objet combinatoire peuvent avoir les mêmes combinaisons de valeurs des caractéristiques, nous commençons par présenter ce que nous appelons *instances symétriques.*

Définition 37 (Instances symétriques)**.** Soit P_b un ensemble de caractéristiques d'un objet combinatoire $\mathcal{O}$, et soit $i \in \mathcal{O}$ une instance de l'objet combinatoire.

Nous notons $P_b(i)$ le tuple de valeurs des caractéristiques de P_b pour l'instance i.

Nous spécifions que des instances d'un objet combinatoire sont *symétriques* si elles ont les

mêmes valeurs de caractéristiques. C'est-à-dire que deux instances i et i' de $\mathcal{O}$ sont symétriques si $P_b(i) = P_b(i')$.

Comme $P_b(i) = P_b(i)$, alors l'instance i est symétrique à elle-même.

Puisque nous nous intéressons beaucoup plus aux caractéristiques d'une instance qu'à l'instance elle-même, une seule instance parmi des instances symétriques est suffisante pour mener notre étude. C'est ainsi que nous définissons maintenant l'*ensemble des représentants d'instances symétriques*.

Définition 38 (Réprésentants d'instances d'un objet combinatoire)**.** Soit P_b un ensemble de caractéristiques d'un objet combinatoire $\mathcal{O}$, et soit $i \in \mathcal{O}$ une instance de l'objet combinatoire.

Nous notons par $I(\mathcal{O})$ un ensemble *maximal* d'instances qui *ne sont pas symétriques*. C'est-à-dire que $I(\mathcal{O})$ possède les deux propriétés suivantes :

— Deux instances quelconques de $I(\mathcal{O})$ ont des combinaisons de valeurs des caractéristiques distinctes. Formellement, cela signifie :

$$\forall i, i' \in I(\mathcal{O}), i \neq i \Longrightarrow P_b(i) \neq P_b(i') \tag{3.64}$$

— Toute instance qui n'appartient pas à $I(\mathcal{O})$ *admet un symétrique* qui appartient à $I(\mathcal{O})$. Formellement, cela signifie :

$$\exists i'' \in \mathcal{O} \setminus I(\mathcal{O}) \Longrightarrow \exists i \in I(\mathcal{O}), P_b(i) = P_b(i''), \tag{3.65}$$

Nous appelons alors $I(\mathcal{O})$ l'*ensemble des représentants d'instances* de l'objet combinatoire $\mathcal{O}$.

Le but de la définition 39 suivante est de montrer qu'il existe des transformations d'instances d'un objet combinatoire telles que deux instances qui *ne sont pas symétriques* ne se transforment pas en instances qui *sont symétriques* pour un autre objet combinatoire.

Définition 39 (Correspondance entre objets combinatoires)**.** Soit P_{b1} un ensemble de caractéristiques d'un objet combinatoire $\mathcal{O}_1$ et P_{b2} un ensemble de caractéristiques d'un objet combinatoire $\mathcal{O}_2$. Soit $I(\mathcal{O}_1)$ (resp. $I(\mathcal{O}_2)$) l'ensemble des représentants d'instances de $\mathcal{O}_1$ (resp. $\mathcal{O}_2$) pour les caractéristiques de P_{b1} (resp. P_{b2}).

Nous appelons *encodage d'un objet combinatoire* $\mathcal{O}_2$ par l'objet combinatoire $\mathcal{O}_1$, une fonction *injective* $\mathcal{H}$ dont l'ensemble de départ est $I(\mathcal{O}_1)$ et l'ensemble d'arrivée est $I(\mathcal{O}_2)$ telle qu'à chaque instance de $I(\mathcal{O}_1)$, elle fait correspondre une instance de $I(\mathcal{O}_2)$.

La fonction $\mathcal{H}$ est injective lorsque si une instance i_2 de $I(\mathcal{O}_2)$ est obtenue en appliquant la fonction $\mathcal{H}$ à une instance i_1 de $I(\mathcal{O}_1)$, alors l'instance i_1 est *unique*. Formellement, nous

avons :

$$\mathcal{H} \text{ est injective } \iff (\forall i_1, i'_1 \in I(\mathcal{O}_1), (\mathcal{H}(i_1) = \mathcal{H}(i'_1) = i_2) \Longrightarrow i_1 = i'_1). \tag{3.66}$$

Nous notons $\mathcal{D}(c)$ le domaine des valeurs d'une caractéristique c et nous notons $c(i)$ la valeur de la caractéristique c pour l'instance i.

Pour définir $\mathcal{H}$, nous énumérons les fonctions qui expriment les caractéristiques de $\mathcal{O}_2$ en fonction des caractéristiques de $\mathcal{O}_1$. C'est-à-dire les applications définies par

$$\forall c_2 \in P_{b2},\ \mathcal{H}_{c_2} : \prod_{c_1 \in P_{b1}} \mathcal{D}(c_1) \longrightarrow \mathcal{D}(c_2) \tag{3.67}$$

C'est-à-dire que pour des instances $i_1 \in I(\mathcal{O}_1), i_2 \in I(\mathcal{O}_2)$ telles que $i_2 = \mathcal{H}(i_1)$, nous avons

$$\forall c_2 \in P_{b2}, \quad c_2(i_2) = \mathcal{H}_{c_2}(P_{b1}(i_1)) \tag{3.68}$$

Ainsi, $\mathcal{H}$ est formellement définie par :

$$\begin{array}{rcl} \mathcal{H} : I(\mathcal{O}_1) & \longrightarrow & I(\mathcal{O}_2) \\ i_1 & \longmapsto & i_2 = \mathcal{H}(i_1) \end{array} \iff \forall c_2 \in P_{b2},\ c_2(i_2) = \mathcal{H}_{c_2}(P_{b1}(i_1)) \tag{3.69}$$

En d'autres termes, l'objet combinatoire $\mathcal{O}_2$ est encodé par l'objet combinatoire $\mathcal{O}_1$ lorsque nous exprimons les caractéristiques de l'objet $\mathcal{O}_2$ en fonction des caractéristiques de l'objet $\mathcal{O}_1$ de manière à ce qu'*une seule instance* de $I(\mathcal{O}_1)$ produise *une instance donnée* de $I(\mathcal{O}_2)$. Notons bien qu'une instance de $I(\mathcal{O}_1)$ représente toutes les autres instances qui lui sont symétriques. Et ces autres instances symétriques n'appartiennent pas à $I(\mathcal{O}_1)$, car $I(\mathcal{O}_1)$ est un ensemble maximal d'instances qui ne sont pas symétriques.

Remarquons qu'une instance d'un objet combinatoire est souvent la solution d'un problème combinatoire. Ainsi, si l'on identifie une instance d'un objet combinatoire à un problème combinatoire, alors la fonction $\mathcal{H}$ qui transforme une instance i_1 d'un objet combinatoire $\mathcal{O}_1$ en une instance i_2 d'un autre objet combinatoire $\mathcal{O}_2$ peut être vue comme transformant le problème combinatoire identifié à i_1 en problème combinatoire identifié à i_2.

Exemple 28 (Encodage entre partitions et arbre enraciné)**.** Soit l'objet combinatoire que constituent les partitions d'ensembles ayant les caractéristiques $n, P, \underline{M}, \overline{M}$ correspondant respectivement au cardinal de l'ensemble à partitionner, le nombre de partitions, la taille minimum d'une partition et la taille maximum d'une partition. Nous pouvons encoder les partitions d'ensembles par l'objet combinatoire qu'est un arbre enraciné ayant les caractéristiques $v, f, \underline{d}, \overline{d}$ correspondant respectivement au nombre de sommets, au nombre de feuilles, au degré minimum d'un sommet et au degré maximum d'un sommet de l'arbre. En effet, comme l'illustre la figure 3.3, nous pouvons partitionner les sommets en sous-ensembles tels qu'une partition est constituée des nœuds fils d'un même nœud père. Ainsi, nous obtenons l'encodage $\mathcal{H}$ suivant :

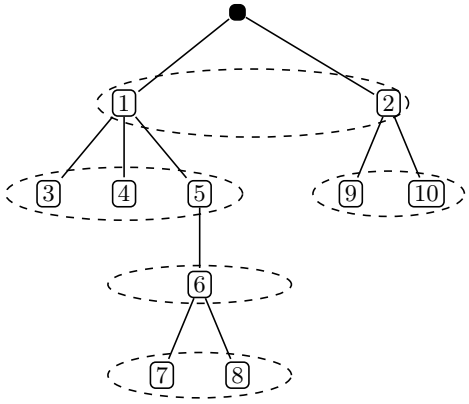


FIGURE 3.3 – Illustration d'un partitionnement d'un arbre enraciné de 11 sommets, sachant que la racine est le sommet noir. Les ellipses en pointillés partitionnent l'arbre. Une partition est constituée des nœuds fils d'un même nœud père. Par exemple, les nœuds 7 et 8 appartiennent à la même partition, car ils ont le même nœud père 6. Ainsi, le partitionnement de l'arbre donne la partition $\{7, 8\}, \{6\}, \{3, 4, 5\}, \{1, 2\}, \{9, 10\}$.

— le cardinal n de l'ensemble à partitionner est le nombre de nœuds fils. Tous les nœuds de l'arbre sont des nœuds fils sauf la racine de l'arbre. Donc nous avons

$$n = \mathcal{H}_n(v, f, \underline{d}, \overline{d}) = v - 1 \tag{3.70}$$

— le nombre P de partitions est le nombre de nœuds pères. Tous les sommets sont des nœuds pères, sauf les feuilles. Donc nous avons

$$P = \mathcal{H}_P(v, f, \underline{d}, \overline{d}) = v - f \tag{3.71}$$

— la taille minimum $\underline{M}$ d'une partition est le nombre minimum de nœuds fils d'un nœud père. Donc nous avons

$$\underline{M} = \mathcal{H}_{\underline{M}}(v, f, \underline{d}, \overline{d}) = \underline{d} \tag{3.72}$$

— la taille maximum $\overline{M}$ d'une partition est le nombre maximum de nœuds fils d'un nœud père. Donc nous avons

$$\overline{M} = \mathcal{H}_{\overline{M}}(v, f, \underline{d}, \overline{d}) = \overline{d} \tag{3.73}$$

Pour montrer que $\mathcal{H}$ est injective, soit $\mathcal{A}$ l'ensemble des arbres enracinés, $I(\mathcal{A})$ un ensemble maximal d'instances non symétriques de $\mathcal{A}$ et soit i et i' des instances d'arbres enracinés de

$I(\mathcal{A})$. Nous avons

$$\mathcal{H}(i) = \mathcal{H}(i') \iff \begin{cases} \mathcal{H}_n(v(i), f(i), \underline{d}(i), \overline{d}(i)) = \mathcal{H}_n(v(i'), f(i'), \underline{d}(i'), \overline{d}(i')) \\ \mathcal{H}_P(v(i), f(i), \underline{d}(i), \overline{d}(i)) = \mathcal{H}_P(v(i'), f(i'), \underline{d}(i'), \overline{d}(i')) \\ \mathcal{H}_{\underline{M}}(v(i), f(i), \underline{d}(i), \overline{d}(i)) = \mathcal{H}_{\underline{M}}(v(i'), f(i'), \underline{d}(i'), \overline{d}(i')) \\ \mathcal{H}_{\overline{M}}(v(i), f(i), \underline{d}(i), \overline{d}(i)) = \mathcal{H}_{\overline{M}}(v(i'), f(i'), \underline{d}(i'), \overline{d}(i')) \end{cases} \tag{3.74}$$

$$\mathcal{H}(i) = \mathcal{H}(i') \iff \begin{cases} v(i) - 1 = v(i') - 1 \\ v(i) - f(i) = v(i') - f(i') \\ \underline{d}(i) = \underline{d}(i') \\ \overline{d}(i) = \overline{d}(i') \end{cases} \tag{3.75}$$

$$\mathcal{H}(i) = \mathcal{H}(i') \iff \begin{cases} v(i) = v(i') \\ v(i) - f(i) = v(i') - f(i') \\ \underline{d}(i) = \underline{d}(i') \\ \overline{d}(i) = \overline{d}(i') \end{cases} \tag{3.76}$$

Comme $v(i) = v(i')$, nous avons en substituant $v(i')$ par $v(i)$

$$v(i) - f(i) = v(i') - f(i') \iff v(i) - f(i) = v(i) - f(i') \iff f(i) = f(i') \tag{3.77}$$

Donc finalement nous avons :

$$\mathcal{H}(i) = \mathcal{H}(i') \iff \begin{cases} v(i) = v(i') \\ f(i) = f(i') \\ \underline{d}(i) = \underline{d}(i') \\ \overline{d}(i) = \overline{d}(i') \end{cases} \tag{3.78}$$

D'après (3.78), i et i' sont symétriques. Comme i et i' appartiennent à $I(\mathcal{A})$, alors nous avons

$$\mathcal{H}(i) = \mathcal{H}(i') \iff \begin{cases} v(i) = v(i') \\ f(i) = f(i') \\ \underline{d}(i) = \underline{d}(i') \\ \overline{d}(i) = \overline{d}(i') \end{cases} \implies i = i' \tag{3.79}$$

car le seul symétrique à i contenu dans $I(\mathcal{A})$ est lui-même. Donc $\mathcal{H}$ est injective.

Exemple 29 (Encodage entre partitions et graphe orienté)**.** Soit l'objet combinatoire qu'est le graphe orienté ayant les caractéristiques $v, a_{max}, \underline{c}, \overline{c}$ correspondant respectivement au nombre de sommets, au nombre maximum d'arcs, à la taille de la plus petite composante connexe et à la taille de la plus grande composante connexe du graphe orienté. Nous pouvons encoder le graphe orienté par l'objet combinatoire que constituent les partitions d'ensembles ayant les caractéristiques $n, S, \underline{M}, \overline{M}$ correspondant respectivement au cardinal de l'ensemble à partitionner, la somme des carrés des tailles des partitions, la taille minimum d'une partition et

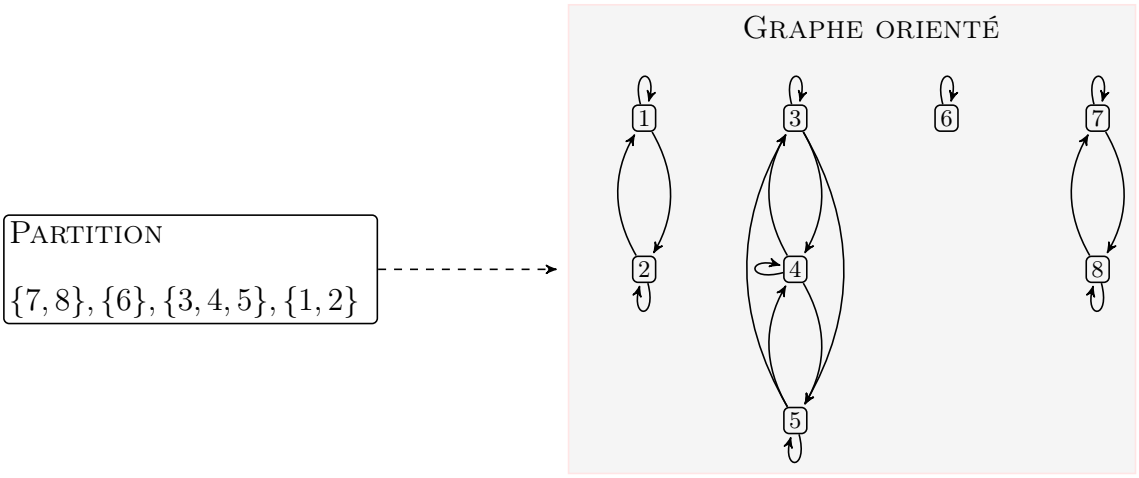


Figure 3.4 – Illustration d'un partitionnement d'un graphe orienté en composantes connexes. Les éléments d'une partition constituent une composante connexe du graphe orienté. Donc la partition $\{6\}$ ayant un seul élément constitue la composante connexe ayant un seul sommet et un arc en boucle. La partition la plus grande $\{3, 4, 5\}$ correspond à la composante connexe la plus grande de trois sommets et neuf arcs.

la taille maximum d'une partition. En effet, tel que l'illustre la figure 3.4, en partitionnant les sommets en sous-ensembles tels qu'une partition est constituée des sommets d'une même composante connexe, nous obtenons l'encodage $\mathcal{U}$ suivant :

— le cardinal n de l'ensemble à partitionner est le nombre de sommets du graphe. Donc nous avons

$$v = \mathcal{U}_v(n, S, \underline{M}, \overline{M}) = n \tag{3.80}$$

— la somme des carrés des tailles des partitions est le nombre maximum d'arcs du graphe orienté. Donc nous avons

$$a_{max} = \mathcal{U}_{a_{max}}(n, S, \underline{M}, \overline{M}) = S \tag{3.81}$$

— la taille minimum $\underline{M}$ d'une partition est la taille de la plus petite composante connexe. Donc nous avons

$$\underline{c} = \mathcal{U}_{\underline{c}}(n, S, \underline{M}, \overline{M}) = \underline{M} \tag{3.82}$$

— la taille maximum $\overline{M}$ d'une partition est la taille de la plus grande composante connexe. Donc nous avons

$$\overline{c} = \mathcal{U}_{\overline{c}}(n, S, \underline{M}, \overline{M}) = \overline{M} \tag{3.83}$$

Comme chaque caractéristique d'un des objets combinatoires est égale à une caractéristique de l'autre objet combinatoire, il est clair que l'encodage $\mathcal{U}$ est injectif.

Remarque 6. De par sa définition 39, l'encodage transforme toute *instance valide* i_1 de l'objet combinatoire d'origine $\mathcal{O}_1$ en une *instance valide* i_2 de l'objet combinatoire d'arrivée $\mathcal{O}_2$. Donc si par exemple dans l'exemple 29 lors de la définition de l'encodage $\mathcal{U}$, nous posons

$$v = \mathcal{U}_v(n, S, \underline{M}, \overline{M}) = S \text{ au lieu de } v = \mathcal{U}_v(n, S, \underline{M}, \overline{M}) = n \tag{3.84}$$

et

$$a_{max} = \mathcal{U}_{a_{max}}(n, S, \underline{M}, \overline{M}) = n \text{ au lieu de } a_{max} = \mathcal{U}_{a_{max}}(n, S, \underline{M}, \overline{M}) = S \tag{3.85}$$

alors, comme $S \geq n$, nous aurons

$$S > n \iff \mathcal{U}_v(n, S, \underline{M}, \overline{M}) > \mathcal{U}_{a_{max}}(n, S, \underline{M}, \overline{M}) \tag{3.86}$$

$$\iff v > a_{max} \tag{3.87}$$

Or, la condition $v > a_{max}$ est *irréalisable par toute instance* du graphe orienté puisque selon sa définition 8 au chapitre 1, les instances que nous considérons sont des graphes tels que chaque sommet a toujours un arc entrant ou sortant.

Par conséquent, si $S > n$, alors $\mathcal{U}$ va générer des *instances invalides* du graphe orienté à partir des *instances valides* de partitions d'ensemble ! Et dès lors $\mathcal{U}$ n'est pas un encodage selon la définition 39. Ainsi, lorsque nous construisons un encodage, il faut s'assurer qu'il génère des instances *valides* de l'objet combinatoire d'arrivée.

L'exemple 28 et l'exemple 29 mettent en évidence les liens entre caractéristiques d'objets combinatoires distincts. Et s'il y a un lien entre caractéristiques d'objets combinatoires distincts, alors il y a aussi un lien entre cartes d'objets combinatoires distincts. Dans la suite, la définition 40 met en évidence des liens qui peuvent exister entre cartes d'objets combinatoires distincts.

Définition 40 (Lien entre cartes d'objets combinatoires distincts)**.** Soit P_{b1} un ensemble de caractéristiques d'un objet combinatoire $\mathcal{O}_1$ et $\{o, c_1, \cdots, c_n\}$ un ensemble de caractéristiques d'un objet combinatoire $\mathcal{O}_2$ tels que $\mathcal{O}_2$ encode $\mathcal{O}_1$ en utilisant l'encodage défini par

$$\forall c \in \{o, c_1, \cdots, c_n\},\ \mathcal{H}_c : \prod_{u \in P_{b1}} \mathcal{D}(u) \longrightarrow \mathcal{D}(c) \tag{3.88}$$

Soit $op_1, op_2 \in \{\leq, \geq\}$. Soient $\mathcal{M}^{u\ op_1}_{P_{b1}}$ et $\mathcal{M}^{o\ op_2}_{\{c_1,\cdots,c_n\}}$ deux cartes de bornes précises des objets combinatoires respectifs $\mathcal{O}_1$ et $\mathcal{O}_2$.

Pour alléger la notation, nous notons

$$\forall i_1 \in I(\mathcal{O}_1), \exists i_2 \in I(\mathcal{O}_2), \forall c_2 \in P_{b2}, \quad c_2(i_2) = \mathcal{H}_{c_2}(P_{b1}(i_1)) \tag{3.89}$$

par

$$\forall c_2 \in P_{b2}, \quad c_2 = \mathcal{H}_{c_2}(P_{b1}) \tag{3.90}$$

Nous disons que $\mathcal{M}^{o\ op_1}_{P_{b1}}$ et $\mathcal{M}^{u\ op_2}_{P_{b2}}$ sont des *cartes liées par encodage* si l'on obtient une conjecture de borne $u\ op_1\ f_u(P_{b1})$ de $\mathcal{M}^{o\ op_1}_{P_{b1}}$ en effectuant les deux opérations suivantes :

1. Dans une conjecture de borne $o\ op_2\ f_o(c_1, \cdots, c_n)$ de $\mathcal{M}^{o\ op_2}_{\{c_1,\cdots,c_n\}}$ on substitue les caractéristiques $c_1, \cdots, c_n$ de $\mathcal{O}_2$ par leurs fonctions $\mathcal{H}_c, \forall c \in \{o, c_1, \cdots, c_n\}$ d'encodage des caractéristiques P_{b1} de $\mathcal{O}_1$ pour obtenir (3.91) :

$$\mathcal{H}_c(P_{b1})\ op_2\ f_o(\mathcal{H}_{c_1}(P_{b1}), \cdots, \mathcal{H}_{c_n}(P_{b1})) \tag{3.91}$$

2. Puis on *inverse* l'expression (3.91) si nécessaire tel que nous avons l'équivalence (3.92) :

$$\mathcal{H}_c(P_{b1})\ op_2\ f_o(\mathcal{H}_{c_1}(P_{b1}), \cdots, \mathcal{H}_{c_n}(P_{b1})) \Longleftrightarrow u\ op_1\ f_u(P_{b1}) \tag{3.92}$$

Exemple 30. Pour faire le parallèle avec la définition 40, nous prenons l'arbre enraciné comme $\mathcal{O}_1$ et les partitions d'ensembles comme $\mathcal{O}_2$. Puis nous prenons $P_{b1} = \{v, f, \underline{d}, \overline{d}\}$ comme ensemble des caractéristiques de $\mathcal{O}_1$ et $\{P, n, \underline{M}, \overline{M}\}$ comme l'ensemble des caractéristiques de $\mathcal{O}_2$.

Alors, la carte $\mathcal{M}^{P\ \leq}_{\{n,\underline{M},\overline{M}\}}$ contient la borne (3.93) générée par le Bound Seeker :

$$P \leq \left\lfloor \frac{n - (\overline{M} - \underline{M})}{\underline{M}} \right\rfloor \tag{3.93}$$

En appliquant l'encodage de l'exemple 28 selon la substitution (3.91), nous obtenons (3.94) :

$$v - f \geq \left\lfloor \frac{v - 1 - (\overline{d} - \underline{d})}{\underline{d}} \right\rfloor \tag{3.94}$$

Puis en procédant à une inversion appropriée, nous obtenons la borne (3.95) de la carte $\mathcal{M}^{f\ \geq}_{\{v,\underline{d},\overline{d}\}}$ qui a été également générée par le Bound Seeker.

$$f \geq \left\lceil \frac{v \cdot \underline{d} - v + 1 + (\overline{d} - \underline{d})}{\underline{d}} \right\rceil \tag{3.95}$$

Ainsi, l'exemple 30 et la figure 3.5, non seulement montrent un lien direct entre les cartes $\mathcal{M}^{f\ \geq}_{\{v,\underline{d},\overline{d}\}}$ et $\mathcal{M}^{P\ \leq}_{\{n,\underline{M},\overline{M}\}}$ de deux objets combinatoires distincts, mais aussi suggèrent une méthode de démonstration généralisée de la véracité des conjectures découvertes. En effet, pour prouver la conjecture (3.95) qui semble plus complexe, il suffit de prouver la conjecture (3.94) qui paraît plus simple. La preuve complète de ces deux conjectures en exploitant la notion d'encodage est faite au chapitre suivant.

Pour appuyer davantage la définition 40, la table 3.2 récapitule quelques correspondances entre caractéristiques d'objets combinatoires distincts.

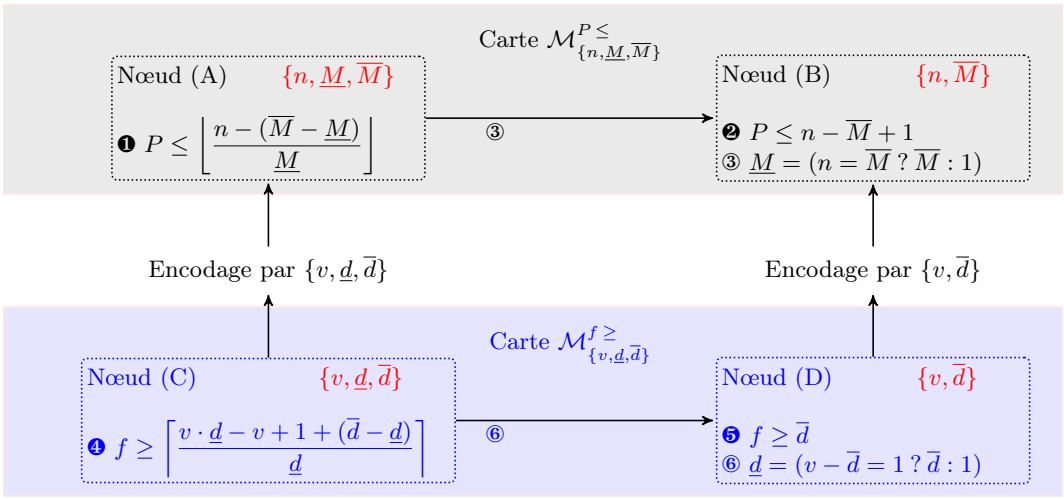


FIGURE 3.5 – Correspondance entre 2 cartes $\mathcal{M}^{f\geq}_{\{v,\underline{d},\overline{d}\}}$ et $\mathcal{M}^{P\leq}_{\{n,\underline{M},\overline{M}\}}$ par encodage de l'arbre enraciné (en bleu) suivant la partition d'ensembles (en noir). P est le nombre de partitions, n est le cardinal de l'ensemble partitionné, $\underline{M}$ est la taille minimale d'une partition, $\overline{M}$ est la taille maximale d'une partition. $v, f, \underline{d}$ et $\overline{d}$ sont respectivement le nombre de sommets, le nombre de feuilles, le nombre minimal de nœuds fils d'un nœud père et le nombre maximal de nœuds fils d'un nœud père dans un arbre enraciné. Les arcs étiquetés par ③ et ⑥ représentent les liens de projection qui existent entre deux conjectures de bornes d'une même carte, c'est-à-dire d'un même objet combinatoire. Notamment ③ est une conjecture de minimalité pour la carte $\mathcal{M}^{f\geq}_{\{v,\underline{d},\overline{d}\}}$, tandis que ⑥ est une conjecture de maximalité pour la carte $\mathcal{M}^{P\leq}_{\{n,\underline{M},\overline{M}\}}$. Notons que les caractéristiques $\{v,\underline{d},\overline{d}\}$ sont utilisées pour encoder les caractéristiques $\{n,\underline{M},\overline{M}\}$.

3.3 Discussion

Ce chapitre a abordé l'étude des liens entre conjectures de cartes d'objets combinatoires où les cartes concernent un même objet combinatoire ou plusieurs objets combinatoires distincts. Pour étudier ces liens, nous avons défini le concept de *carte inverse* et *d'encodage d'objets combinatoires*. Cela a permis de mettre en évidence ces liens via le théorème 3 d'*inversion d'une projection de borne polynomiale de second degré* et via le concept de *cartes liées par encodage* de leurs objets combinatoires. Nous avons testé notre approche d'inversion automatique des bornes polynomiales de second degré sur huit objets combinatoires et cela nous a donné les résultats de la table 3.1. Nous avons également observé des limites de la méthode d'inversion. Notamment, parfois cette méthode génère des bornes incomplètes comme l'illustre l'exemple 22, ou bien elle génère une condition de faisabilité à la place d'une borne comme l'illustre l'exemple 23. De plus, il y a deux aspects à creuser davantage que nous présentons :

— Comme nous recherchons des cartes aussi complètes que possible, il serait envisageable d'exploiter les informations que fournissent les notions de projection, d'inversion et d'encodage afin de découvrir d'autres bornes impliquant plus de caractéristiques bornantes. En effet, soit $op_1, op_2 \in \{\leq, \geq\}$ et $i \in [1:n]$. Supposons que nous voulions déterminer la formule f d'une borne $o\ op_2\ f(c_1, \cdots, c_n)$ d'un objet combinatoire et que nous ayons son

Objet combinatoire	Partition	Arbre enraciné	Graphe orienté	Forêt enracinée avec/sans sommet isolé
Cardinal de l'ensemble partitionné	n	$v-1$	v	v
Nombre de partitions	P	$v-f$	c	t
Taille maximale d'une partition	$\overline{M}$	$\overline{d}$	$\overline{c}$	$\overline{t}$
Taille minimale d'une partition	$\underline{M}$	$\underline{d}$	$\underline{c}$	$\underline{t}$
Somme des carrés des tailles des partitions	S	$\sum_{i=1}^{v-f} d_i^2$	a_{max}	$\sum_{i=1}^{t} t_i^2$

TABLE 3.2 – Correspondances entre caractéristiques d'objets combinatoires distincts. Soit $\mathcal{P}$ l'ensemble des nœuds pères d'un arbre enraciné. Pour tout $i \in [1; |\mathcal{P}|]$, soit d_i le nombre de nœuds fils du i-ème nœud père de $\mathcal{P}$. Pour tout $i \in [1; t]$, soit t_i la taille du i-ème arbre d'une forêt enracinée. v est le nombre de sommets d'un arbre enraciné ou d'un graphe orienté ou encore d'une forêt enracinée. $c, \overline{c}, \underline{c}$ et a_{max} sont respectivement le nombre de composantes connexes, la taille de la plus grande composante connexe, la taille de la plus petite composante connexe et la nombre d'arcs maximal d'un graphe orienté. $t, \underline{t}$ et $\overline{t}$ sont respectivement le nombre d'arbres, la taille maximale d'un arbre et la taille minimale d'un arbre d'une forêt enracinée. $f, \underline{d}$ et $\overline{d}$ sont respectivement le nombre de sommets, le nombre de feuilles, le nombre minimal de nœuds fils d'un nœud père et le nombre maximal de nœuds fils d'un nœud père dans un arbre enraciné. P est le nombre de partitions, n est le cardinal de l'ensemble partitionné, $\underline{M}$ est la taille minimale d'une partition, $\overline{M}$ est la taille maximale d'une partition, S est la somme des carrés des tailles des partitions.

projeté $o\ op_2\ f_1(c_1, \cdots, c_{n-1})$, ou ses inverses $c_i\ op_1\ f_i(o, c_1, \cdots, c_{i-1}, c_{i+1}, \cdots, c_{n-1})$, ou encore son encodage par un autre objet combinatoire. Ces formules étant fournies respectivement par une carte de bornes précises, une carte inverse ou une carte obtenue par encodage. Il est généralement plus difficile de déterminer une formule utilisant plus de caractéristiques qu'une formule qui en utilise moins, car l'ajout d'une caractéristique nouvelle apporte un ensemble de conditions de faisabilités qui doivent être prises en compte. Ces conditions de faisabilités sont définies par le domaine de valeurs de la nouvelle caractéristique et ceux des caractéristiques déjà présentes. Donc, ces conditions de faisabilités augmentent l'espace de recherche des formules, car elles rendent les formules plus complexes. Dès lors, nous pourrions exploiter les informations fournies par les liens de projection, d'inversion ou d'encodage pour mieux cibler efficacement la formule de la borne $o\ op_2\ f_2(c_1, \cdots, c_n)$. C'est-à-dire que le modèle à contraintes du Bound Seeker présenté au chapitre 2 pourrait être amélioré en lui ajoutant des contraintes de projec-

tion, d'inversion et d'encodage. Ces notions étant encore dans leurs états préliminaires, il y a matière à recherche dans cette direction.

— Le deuxième aspect est celui de l'élaboration des preuves avec une assistance automatisée ou semi-automatisée. Les propriétés de projection, d'inversion et d'encodage se réfèrent à la notion de *réécriture* dans le domaine de la *mécanisation ou automatisation des preuves*. Elles permettent donc de faire un grand pas vers l'élaboration des preuves automatiques afin de prouver un nombre maximum de conjectures générées par le Bound Seeker tout en testant ces liens existant entre conjectures et entre cartes. En effet, comme le stipule la section 2.4 du chapitre 2, nous avons des milliers de conjectures à démontrer. Donc une mécanisation d'une partie des preuves nous serait d'un grand apport. Il y a donc matière à réflexion dans cette direction de la recherche. Néanmoins, nous présentons au chapitre 4, suivant, des preuves de conjectures du Bound Seeker, élaborées manuellement.

Chapitre 4

Preuves de conjectures découvertes par le Bound Seeker

Une conjecture est une égalité ou une inégalité vérifiée par tous les exemples disponibles. Il faut ensuite prouver la validité de chaque conjecture pour n'importe quelle instance d'objet combinatoire. Bien que bon nombre de programmes à l'instar de Agda [1], Coq [18; 63] tentent de répondre à la problématique d'automatisation des preuves, ils sont généralement une mécanisation et une généralisation de preuves manuelles. Ces programmes se prêtent mieux pour des preuves de conjectures simples. Dans le cadre du Bound Seeker, il s'agit non seulement de prouver une borne, mais aussi de prouver que la borne est précise. C'est-à-dire qu'il s'agit de prouver non seulement la borne, mais de prouver aussi que pour une borne précise $o\ op\ f(P_b)$ avec $op \in \{\leq, \geq\}$, on peut toujours construire une instance de l'objet combinatoire, telle que $v_o = f(V_b)$ pour une combinaison valide V_b des valeurs des caractéristiques de P_b, qui conduit à la valeur v_o de o. Ainsi, dans ce chapitre, nous présentons des preuves manuelles de nouvelles conjectures découvertes par le Bound Seeker.

Les motivations du chapitre 4 sont doubles :

— d'abord, montrer que le Bound Seeker génère de nouvelles conjectures complexes valides dans le cadre d'invariants portant sur des objets combinatoires ;
— puis, mettre en lumière des pistes pour automatiser la preuve de conjectures.

Les contributions du chapitre 4 concernent les preuves de nouveaux théorèmes sur les invariants d'objets combinatoires qui deviennent donc directement exploitables dans la résolution des problèmes combinatoires.

Dans la section 4.1, nous présentons les techniques de preuves utilisées pour prouver les conjectures. Puis, la section 4.2 expose les preuves élaborées pour différents invariants des bornes d'objets combinatoires. Finalement, la section 4.3 est consacrée à une prise de recul sur le

travail effectué.

4.1 Techniques de preuves utilisées

Pour prouver les conjectures des bornes contenues dans les cartes générées par le Bound Seeker, nous utilisons principalement les techniques suivantes.

- **La déduction :** Elle consiste à partir des définitions des caractéristiques d'un objet combinatoire et des conditions de faisabilités reliant ces caractéristiques d'enchaîner des déductions menant à la borne.
- **La contradiction :** Elle consiste à supposer qu'une hypothèse est fausse et à montrer que cette supposition conduit à une contradiction. La contradiction provient souvent du fait qu'une condition de faisabilité sur les caractéristiques d'un objet combinatoire n'est pas respectée.
- **L'encodage d'objets combinatoires :** Elle consiste à manipuler algébriquement une borne déjà prouvée pour un objet combinatoire donné afin d'en déduire une autre borne pour un autre objet combinatoire. La déduction est faite (i) en substituant les caractéristiques de la borne prouvée par leurs expressions équivalentes telles que présentées par la définition 39, et (ii) en manipulant algébriquement si nécessaire la formule obtenue après substitution afin d'isoler la caractéristique que l'on veut borner.
- **L'élaboration d'une méthode de construction d'instances valides atteignant la borne précise de l'objet combinatoire :** Celle-ci s'inscrit dans le cadre de la preuve qu'une borne est précise. Elle consiste à s'assurer que les conditions de faisabilités sont toujours respectées lors de la construction d'instances atteignant la borne, quelles que soient les combinaisons valides des valeurs des caractéristiques bornantes.
- **La preuve par dépliage de la formule :** Elle consiste à décomposer la borne en plusieurs cas simples et disjoints qui dépendent des conditions de faisabilités des caractéristiques bornantes, puis de les prouver en s'appuyant sur ses conditions de faisabilités.

4.2 Preuves de quelques conjectures découvertes par le Bound Seeker

4.2.1 Preuves de conjectures sur les graphes orientés

Dans cette section, nous prouvons quatre conjectures sur les graphes orientés découvertes par le Bound Seeker et publiées dans l'article [35]. Nous rappelons que nous ne considérons que les graphes orientés où chaque sommet a au moins un arc entrant ou sortant et que l'expression $(cond\,?\,x\,:\,y)$ vaut x si la condition $cond$ est vraie et y sinon. La notation $[cond]$ est une abréviation de $(cond\,?\,1:0)$.

Pour prouver les quatre conjectures Conjecture 4–Conjecture 7 qui suivent, nous procédons de la façon suivante :

— Dans un premier temps, nous déplions la conjecture, c'est-à-dire que nous énumérons toutes ses conditions explicites ou implicites possibles conduisant à une expression simplifiée de la conjecture. Ces conditions sont mutuellement disjointes et couvrent tous les cas possibles.
— Dans un deuxième temps, nous prouvons chaque cas induit par une condition en utilisant une preuve par contradiction. Les preuves de chaque cas sont indépendantes les unes des autres. Nous prouvons aussi que certains cas sont impossibles, car ils ne correspondent à aucune instance réalisable d'un objet combinatoire.

Les tables 4.1, 4.2 et 4.3 montrent respectivement les différents cas induits par les conjectures qui sont Conjecture 4, Conjecture 6 et Conjecture 7. Pour chaque cas, la dernière colonne des tables 4.1, 4.2, et 4.3 donne un exemple de graphe orienté atteignant la borne correspondante. Finalement, comme la conjecture Conjecture 5 ne mentionne pas d'expression conditionnelle, elle ne nécessite pas de dépliage.

Conjecture 4. Cette conjecture exprime la borne inférieure précise du nombre de composantes connexes c d'un graphe orienté en fonction de son nombre de sommets v, de la taille de la plus grande composante connexe et de celle de la plus petite composante fortement connexe.

$$c \geq \left\lceil \frac{v}{\overline{c}} \right\rceil + [\neg((2 \cdot \underline{s} \leq \overline{c}) \vee (\underline{s} \geq (v \bmod \overline{c} = 0 \ ? \ \overline{c} \ : v \bmod \overline{c}))] \tag{4.1}$$

Preuve de Conjecture 4. Nous associons les trois conditions suivantes ① $v \bmod \overline{c} = 0$, ② $\underline{s} \geq (v \bmod \overline{c} = 0 \ ? \ \overline{c} \ : \ v \bmod \overline{c})$, ③ $2 \cdot \underline{s} \leq \overline{c}$ à Conjecture 4 et les combinons d'une manière systématique afin de produire un ensemble de sept cas mutuellement incompatibles tels que présentés dans la Table 4.1. Notons que le cas ① ∧ ② ∧ ③ n'est pas réalisable. Nous prouvons chacun des sept cas restant par contradiction.

Preuve par contradiction du cas ① ∧ ② ∧ ¬③ et du cas ① ∧ ¬② ∧ ③ : Pour tout graphe orienté, nous avons $c \cdot \overline{c} \geq v$. Mais la négation de Conjecture 4, c'est-à-dire $c < \lfloor v/\overline{c} \rfloor$, implique que $c \cdot \overline{c} < v$, ce qui est une contradiction, sachant que $x < \lfloor y/z \rfloor \Rightarrow x \cdot z < y$.

Preuve par contradiction des cas ¬① ∧ ② ∧ ③, ¬① ∧ ② ∧ ¬③, et **¬① ∧ ¬② ∧ ③ :** Pour tout graphe orienté, nous avons $c \cdot \overline{c} \geq v$, qui implique $c \geq \lfloor v/\overline{c} \rfloor$. De la négation de Conjecture 4, nous avons $c < \lfloor v/\overline{c} \rfloor + 1$, qui implique $c \leq \lfloor v/\overline{c} \rfloor$. Donc finalement, nous avons $c = \lfloor v/\overline{c} \rfloor$, et le nombre maximal correspondant de sommets est $\lfloor v/\overline{c} \rfloor \cdot \overline{c}$, qui est strictement plus petit que $v = \lfloor v/\overline{c} \rfloor \cdot \overline{c} + v \bmod \overline{c}$ puisque d'après ¬①, nous avons $v \bmod \overline{c} > 0$. C'est une contradiction.

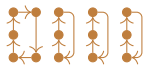
Cond. ①	Cond. ②	Cond. ③	borne simplifiée de c	instances témoins	
①	②	¬③	$\lfloor v/\overline{c} \rfloor$	$\begin{cases} v = 1 \\ \overline{c} = 1 \\ \underline{s} = 1 \end{cases}$ 	$c = 1$
①	¬②	③	$\lfloor v/\overline{c} \rfloor$	$\begin{cases} v = 2 \\ \overline{c} = 2 \\ \underline{s} = 1 \end{cases}$ 	$c = 1$
①	¬②	¬③	$\lfloor v/\overline{c} \rfloor + 1$	$\begin{cases} v = 9 \\ \overline{c} = 3 \\ \underline{s} = 2 \end{cases}$ 	$c = 4$
¬①	②	③	$\lfloor v/\overline{c} \rfloor + 1$	$\begin{cases} v = 3 \\ \overline{c} = 2 \\ \underline{s} = 1 \end{cases}$ 	$c = 2$
¬①	②	¬③	$\lfloor v/\overline{c} \rfloor + 1$	$\begin{cases} v = 5 \\ \overline{c} = 3 \\ \underline{s} = 2 \end{cases}$ 	$c = 2$
¬①	¬②	③	$\lfloor v/\overline{c} \rfloor + 1$	$\begin{cases} v = 5 \\ \overline{c} = 3 \\ \underline{s} = 1 \end{cases}$ 	$c = 2$
¬①	¬②	¬③	$\lfloor v/\overline{c} \rfloor + 2$	$\begin{cases} v = 14 \\ \overline{c} = 5 \\ \underline{s} = 3 \end{cases}$ 	$c = 4$

TABLE 4.1 – Versions simplifiées de CONJECTURE 4 en fonction des conditions ① $v \bmod \overline{c} = 0$, ② $\underline{s} \geq (v \bmod \overline{c} = 0 \ ? \ \overline{c} \ : \ v \bmod \overline{c})$, ③ $2 \cdot \underline{s} \leq \overline{c}$, où v est le nombre de sommets d'un graphe orienté, $\overline{c}$ est la taille de la plus grande composante connexe, $\underline{s}$ est la taille de la plus petite composante connexe et c est le nombre de composantes connexes dudit graphe orienté.

Preuve par contradiction du cas ①∧¬②∧¬③ : Nous dérivons de la négation de CONJECTURE 4 une borne supérieure et une borne inférieure du nombre v de sommets et montrons que la borne inférieure est plus grande que la borne supérieure.

- **Calcul de la borne inférieure de v :** En simplifiant la négation de CONJECTURE 4, c'est-à-dire $c < \lfloor v/\overline{c} \rfloor + 1$, nous obtenons $c \leq \lfloor v/\overline{c} \rfloor$, qui implique $c \cdot \overline{c} \leq v$.
- **Calcul de la borne supérieure de v :** D'après ¬③ nous avons une seule composante fortement connexe dans chaque composante connexe. Une de ces composantes connexes a une taille égale à $\underline{s}$. Soit $|c_i|$ la taille de chacune des autres composantes connexes avec $i \in [1, c-1]$. Ainsi, le nombre total de sommets est $\underline{s} + \sum_{i=1}^{c-1} |c_i|$ qui est plus petit ou égal à $\underline{s} + (c-1) \cdot \overline{c}$.
- **Déduction d'une contradiction :** Nous avons montré que $v \in [c \cdot \overline{c}, \underline{s} + (c-1) \cdot \overline{c}]$. Donc nous avons $c \cdot \overline{c} \leq \underline{s} + (c-1) \cdot \overline{c}$. Mais ① entraîne que ¬② se simplifie en $\underline{s} < \overline{c}$, qui conduit au fait que la borne supérieure $\underline{s} + (c-1) \cdot \overline{c}$ est strictement plus petite que $c \cdot \overline{c}$, ce qui est une contradiction.

Preuve par contradiction du cas ¬①∧¬②∧¬③ : Pour tout graphe orienté, nous avons $c \cdot \overline{c} \geq v$. De plus, v peut être décomposé en fonction de $\overline{c}$ de la façon suivante $v = \lfloor v/\overline{c} \rfloor \cdot \overline{c} + v \bmod \overline{c}$. Cela conduit à l'inégalité suivante $c \cdot \overline{c} \geq \lfloor v/\overline{c} \rfloor \cdot \overline{c} + v \bmod \overline{c}$, qui peut être réécrite sous la forme suivante $c \geq \lfloor v/\overline{c} \rfloor + (v \bmod \overline{c})/\overline{c}$. D'après ¬①, nous avons $v \bmod \overline{c} > 0$ qui implique $c > \lfloor v/\overline{c} \rfloor$, ce qui implique $c \geq \lfloor v/\overline{c} \rfloor + 1$. En considérant la négation de CONJECTURE 4 nous obtenons $c = \lfloor v/\overline{c} \rfloor + 1$, et le nombre maximal correspondant de sommets est $\lfloor v/\overline{c} \rfloor \cdot \overline{c} + \underline{s}$, car une des composantes connexes a une taille de $\underline{s}$ sommets puisque d'après ¬③ chaque composante connexe contient exactement une seule composante fortement connexe. D'après ¬②, $\lfloor v/\overline{c} \rfloor \cdot \overline{c} + \underline{s}$ est strictement inférieur à $v = \lfloor v/\overline{c} \rfloor \cdot \overline{c} + v \bmod \overline{c}$, ce qui est une contradiction. □

CONJECTURE 5. Cette conjecture exprime la borne inférieure précise du nombre de composantes connexes c d'un graphe orienté en fonction de son nombre de composantes fortement connexes s, de la taille $\overline{c}$ de la plus grande composante connexe et de la taille $\underline{s}$ de la plus petite composante fortement connexe.

$$c \geq \left\lceil \frac{s}{\left\lfloor \frac{\overline{c}}{\underline{s}} \right\rfloor} \right\rceil \tag{4.2}$$

Preuve de CONJECTURE 5. Notons en premier que le nombre maximum de composantes fortement connexes dans une composante connexe est $\lfloor \overline{c}/\underline{s} \rfloor$. Puis, pour avoir un nombre minimal de composantes connexes, nous devons partager aussi équitablement que possible les s composantes fortement connexes entre les composantes connexes. Et comme une composante connexe contient au plus $\lfloor \overline{c}/\underline{s} \rfloor$ composantes fortement connexes, nous obtenons la borne exprimée par CONJECTURE 5, notamment $\lceil s/\lfloor \overline{c}/\underline{s} \rfloor \rceil$.

Pour montrer la précision de la borne, nous fournissons une construction d'une instance de graphe orienté qui a un nombre de composantes connexes minimales $c = \lceil s/\lfloor \overline{c}/\underline{s} \rfloor \rceil$ quelle que soit la combinaison valide des valeurs de $s, \overline{c}$ et $\underline{s}$. En effet avec $\overline{c}$ et $\underline{s}$, on construit chaque composante connexe maximale qui contient $\left\lfloor \frac{\overline{c}}{\underline{s}} \right\rfloor - 1$ composantes fortement connexes toutes de taille $\underline{s}$ et une composante fortement connexe de taille $\underline{s} + \overline{c} \bmod \underline{s}$. Cela est possible, car nous avons :

$$\left(\left\lfloor \frac{\overline{c}}{\underline{s}} \right\rfloor - 1 \right) \cdot \underline{s} + (\underline{s} + \overline{c} \bmod \underline{s}) = \overline{c} \tag{4.3}$$

Ainsi chaque composante connexe maximale de taille $\overline{c}$ a bel et bien $\left\lfloor \frac{\overline{c}}{\underline{s}} \right\rfloor$ composantes fortement connexes. Ensuite on construit $\left\lfloor \frac{s}{\left\lfloor \frac{\overline{c}}{\underline{s}} \right\rfloor} \right\rfloor$ composantes connexes de taille $\overline{c}$ qui contiennent chacune $\left\lfloor \frac{\overline{c}}{\underline{s}} \right\rfloor$ composantes fortement connexes. Puis, si $s \bmod \left\lfloor \frac{\overline{c}}{\underline{s}} \right\rfloor > 0$, il restera $s \bmod \left\lfloor \frac{\overline{c}}{\underline{s}} \right\rfloor$

Cond. ④	Cond. ⑤	borne simplifiée de $\overline{s}$	instances témoins	
④	⑤	v	$v=1\ s=1\ \underline{s}=1$	$\overline{s}=1$
¬④	¬⑤	$\lceil (v-\underline{s})/(s-1) \rceil$	$v=2\ s=2\ \underline{s}=1$	$\overline{s}=1$

TABLE 4.2 – Versions simplifiées de CONJECTURE 6 en fonction des conditions ④ $v=\underline{s}$, ⑤ $s=1$, où v est le nombre de sommets d'un graphe orienté, $\underline{s}$ est la taille de la plus petite composante fortement connexe, s est le nombre de composantes fortement connexes et $\overline{s}$ est la taille de la plus grande composante fortement connexe dudit graphe orienté.

composantes fortement connexes qui vont constituer une dernière composante connexe de taille inférieure à $\overline{c}$. Ainsi, le nombre de composantes connexes dans cette construction sera $c = \left\lfloor \frac{s}{\left\lfloor \frac{\overline{c}}{\underline{s}} \right\rfloor} \right\rfloor$ ou $c = \left\lfloor \frac{s}{\left\lfloor \frac{\overline{c}}{\underline{s}} \right\rfloor} \right\rfloor + 1$, c'est-à-dire $c = \left\lceil \frac{s}{\left\lfloor \frac{\overline{c}}{\underline{s}} \right\rfloor} \right\rceil$. □

CONJECTURE 6. Dans le contexte d'un graphe orienté, cette conjecture exprime la borne inférieure précise de la taille $\overline{s}$ de la plus grande composante fortement connexe en fonction du nombre v de sommets, du nombre s de composantes fortement connexes et de la taille $\underline{s}$ de la plus petite composante fortement connexe.

$$\overline{s} \geq \left\lceil \frac{(v=\underline{s}\ ?\ v : v-\underline{s})}{s-1+[s=1]} \right\rceil \tag{4.4}$$

Preuve de CONJECTURE 6. La preuve est faite en partitionnant un graphe orienté en composantes fortement connexes. Dans la table 4.2, nous associons les deux conditions ④ $v=\underline{s}$, ⑤ $s=1$ à CONJECTURE 6 et les combinons d'une façon systématique pour produire un ensemble de cas mutuellement indépendants où la borne est simplifiée. Notons que les cas ④ ∧ ¬⑤ et ¬④ ∧ ⑤ sont absents du fait qu'ils ne sont pas réalisables.

Cas ④ ∧ ⑤ : Ce cas est évident, car la condition ⑤ indique que le graphe orienté n'a qu'une seule composante fortement connexe. Cette dernière est donc la plus large et la plus petite composante fortement connexe. Dans ce cas la borne est précise, car nous avons toujours $\overline{s}=v$ pour tout graphe orienté ayant une seule composante fortement connexe.

Cas ¬④∧¬⑤ : Comme $s>1$, soit s_i le nombre de sommets de la i-ème composante fortement connexe du graphe orienté avec $i \in [1,s]$. Nous avons $\sum_{i=1}^{s-1} \overline{s} \geq \sum_{i=1}^{s-1} s_i$, qui équivaut à $\overline{s}\cdot(s-1) \geq v-\underline{s}$. Par conséquent $\overline{s} \geq (v-\underline{s})/(s-1)$, et comme $\overline{s} \in \mathbb{N}$ nous avons $\overline{s} \geq \lceil (v-\underline{s})/(s-1) \rceil$.

Pour montrer la précision de la borne, soit une combinaison valide de valeurs de $v, \underline{s}$ et s : nous construisons une instance de graphe orienté tel que $\overline{s} = \lceil (v-\underline{s})/(s-1) \rceil$ de la manière suivante :

Cond. ⑥	Cond. ⑦	borne simplifiée de $\overline{c}$	instances témoins	
⑥	⑦∨¬⑦	$\underline{c}$	$\begin{cases} v=1 & \underline{c}=1 \\ c=1 & \overline{s}=1 \end{cases}$ 	$\overline{c}=1$
¬⑥	⑦	$\overline{s}$	$\begin{cases} v=5 & \underline{c}=1 \\ c=3 & \overline{s}=3 \end{cases}$ 	$\overline{c}=3$
¬⑥	¬⑦	$\lceil (v-\underline{c})/(c-1) \rceil$	$\begin{cases} v=3 & \underline{c}=1 \\ c=2 & \overline{s}=1 \end{cases}$ 	$\overline{c}=2$

TABLE 4.3 – Versions simplifiées de CONJECTURE 7 en fonction des conditions ⑥ $v = c \cdot \underline{c}$ et ⑦ $\overline{s} \geq \lceil (v-\underline{c})/(c-1) \rceil$, où $\overline{c}$ est la taille de la plus grande composante connexe d'un graphe orienté, $\underline{c}$ est la taille de la plus petite composante connexe, v est le nombre de sommets, c est le nombre de composantes connexes et $\overline{s}$ est la taille de la plus grande composante fortement connexe dudit graphe orienté.

Notons d'abord que dans un graphe orienté qui a v sommets et s composantes fortement connexes, si $\underline{s}$ est la taille de la plus petite composante fortement connexe alors toutes les $s-1$ autres composantes fortement connexes ont une taille au moins égale à $\underline{s}$. Donc nous avons la relation $v - \underline{s} \geq (s-1) \cdot \underline{s}$ qui est vérifiée. Cette relation conduit à :

$$v - \underline{s} \geq (s-1) \cdot \underline{s} \iff \frac{v-\underline{s}}{s-1} \geq \underline{s} \implies \left\lfloor \frac{v-\underline{s}}{s-1} \right\rfloor \geq \underline{s} \tag{4.5}$$

La relation (4.5) est une *condition de faisabilité* que doivent satisfaire les combinaisons de valeurs de $v, \underline{s}$ et s.

Maintenant, nous construisons d'abord une composante fortement connexe de taille $\underline{s}$. Il reste donc $s-1$ composantes fortement connexes à construire avec les $v - \underline{s}$ sommets restants. Comme il s'agit de déterminer une borne inférieure de la taille de la plus grande composante fortement connexe, nous cherchons à avoir des composantes fortement connexes de plus petites tailles possibles. Pour cela, nous partageons équitablement les sommets restant entre les $s-1$ composantes fortement connexes restantes. Ainsi, ces autres composantes fortement connexes auront au moins une taille égale à $\lfloor (v-\underline{s})/(s-1) \rfloor$. Cela est possible grâce à la condition de faisabilité (4.5) qui soutient que de telles composantes fortement connexes ont une taille au moins égale à $\underline{s}$. Puis, si $(v-\underline{s}) \bmod (s-1) > 0$, il va rester $(v-\underline{s}) \bmod (s-1)$ sommets qui seront distribués à $(v-\underline{s}) \bmod (s-1)$ composantes fortement connexes à raison d'un sommet par composante fortement connexe afin de minimiser le nombre de sommets d'une composante fortement connexe. Au final la taille de la plus grande composante fortement connexe dans cette construction est $\overline{s} = \lfloor (v-\underline{s})/(s-1) \rfloor$ ou $\overline{s} = \lfloor (v-\underline{s})/(s-1) \rfloor + 1$, c'est-à-dire $\overline{s} = \lceil (v-\underline{s})/(s-1) \rceil$. □

CONJECTURE 7. Dans le contexte d'un graphe orienté, cette conjecture exprime la borne inférieure précise de la taille $\overline{c}$ de la plus grande composante connexe en fonction de la taille $\underline{c}$ de la plus petite composante connexe, du nombre v de sommets, du nombre c de composantes

connexes et de la taille $\overline{s}$ de la plus grande composante fortement connexe.

$$\overline{c} \geq \left(v = c \cdot \underline{c} \ ? \ \underline{c} \ : \ \max\left(\overline{s}, \left\lceil \frac{v - \underline{c}}{c - 1} \right\rceil\right)\right) \tag{4.6}$$

Preuve de Conjecture 7. La preuve est faite en partitionnant un graphe orienté en composantes connexes. Nous associons la condition ⑥ $v = c \cdot \underline{c}$ et la condition ⑦ $\overline{s} \geq \lceil (v - \underline{c})/(c-1) \rceil$ à Conjecture 7 et les combinons d'une façon systématique pour produire un ensemble de cas mutuellement indépendants où la borne est simplifiée comme le montre la table 4.3.

Cas ⑥ $\wedge$ (⑦ $\vee \neg$⑦) Comme la condition ⑥ indique que toute composante connexe du graphe orienté a une taille égale à celle de la plus petite composante connexe, cela implique $\overline{c} = \underline{c}$. La borne est donc précise pour toute instance de graphe orienté qui vérifie la condition ⑥ $v = c \cdot \underline{c}$.

Cas $\neg$⑥ $\wedge$ ⑦ **et cas** $\neg$⑥ $\wedge \neg$⑦ Soit $|c_i|$ le nombre de sommets de la i-ème composante connexe d'un graphe orienté avec $i \in [1, c]$. Notons que pour tout graphe orienté, comme $\sum_{i=1}^{c-1} \overline{c} \geq \sum_{i=1}^{c-1} |c_i|$, nous avons $\overline{c} \cdot (c-1) \geq v - \underline{c}$. Puisque $\neg$⑥ implique $c > 1$, nous avons donc $\overline{c} \geq \lceil (v - \underline{c})/(c-1) \rceil$. Par conséquent, comme pour tous les graphes orientés nous avons aussi $\overline{c} \geq \overline{s}$, nous déduisons $\overline{c} \geq \max(\overline{s}, \lceil (v - \underline{c})/(c-1) \rceil)$, ce qui prouve les deux cas $\neg$⑥ $\wedge$ ⑦ et $\neg$⑥ $\wedge \neg$⑦.

Pour montrer la précision de la borne, soit une combinaison valide de valeurs de $v, \overline{s}, c$ et $\underline{c}$: nous construisons une instance de graphe orienté tel que $\overline{c} = \max(\overline{s}, \lceil (v - \underline{c})/(c-1) \rceil)$ de la manière suivante :

Notons d'abord que dans un graphe orienté qui a v sommets et c composantes connexes, si $\underline{c}$ est la taille de la plus petite composante connexe alors toutes les $c-1$ autres composantes connexes ont une taille au moins égale à $\underline{c}$. Donc nous avons la relation $v - \underline{c} \geq (c-1) \cdot \underline{c}$ qui est vérifiée. Cette relation conduit à :

$$v - \underline{c} \geq (c-1) \cdot \underline{c} \iff \frac{v - \underline{c}}{c-1} \geq \underline{c} \implies \left\lfloor \frac{v - \underline{c}}{c-1} \right\rfloor \geq \underline{c} \tag{4.7}$$

La relation (4.7) est une *condition de faisabilité* que doivent satisfaire les combinaisons de valeurs de $v, \underline{c}$ et c.

De même, si $\overline{c}_f$ est la taille de la composante connexe qui contient la plus grande composante fortement connexe, les $c-1$ autres composantes connexes ont une taille au moins égale à $\underline{c}$. Donc nous avons la relation $v - \overline{c}_f \geq (c-1) \cdot \underline{c}$ qui est vérifiée. Alors nous avons

$$\overline{s} \leq \overline{c}_f \iff -\overline{s} \geq -\overline{c}_f \iff v - \overline{s} \geq v - \overline{c}_f \tag{4.8}$$

$$\implies v - \overline{s} \geq v - \overline{c}_f \geq (c-1) \cdot \underline{c} \tag{4.9}$$

$$\implies v - \overline{s} \geq (c-1) \cdot \underline{c} \tag{4.10}$$

$$\implies \left\lfloor \frac{v - \overline{s}}{c-1} \right\rfloor \geq \underline{c} \tag{4.11}$$

La relation (4.11) est une *condition de faisabilité* que doivent satisfaire les combinaisons de valeurs de $v, \underline{c}, \overline{s}$ et c.

1. Si nous avons $\neg⑦\ \overline{s} < \lceil (v - \underline{c})/(c-1) \rceil$, nous construisons d'abord une composante connexe de taille $\underline{c}$. Il reste donc $c-1$ composantes connexes à construire avec les $v - \underline{c}$ sommets restants. Comme il s'agit de déterminer une borne inférieure de la taille de la plus grande composante connexe, nous cherchons à avoir des composantes connexes de plus petites tailles possibles. Pour cela, nous partageons équitablement les sommets restants entre les $c-1$ composantes connexes restantes. Ainsi, ces autres composantes connexes auront au moins une taille égale à $\lfloor (v - \underline{c})/(c-1) \rfloor$. Cela est possible grâce à la condition de faisabilité (4.7) qui soutient que de telles composantes connexes ont une taille au moins égale à $\underline{c}$. Puis, si $(v - \underline{c}) \bmod (c-1) > 0$, il va rester $(v - \underline{c}) \bmod (c-1)$ sommets qui seront distribués à $(v - \underline{c}) \bmod (c-1)$ composantes connexes à raison d'un sommet par composante connexe afin de minimiser le nombre de sommets d'une composante connexe. Au final la taille de la plus grande composante connexe dans cette construction est $\lfloor (v - \underline{c})/(c-1) \rfloor$ ou $\lfloor (v - \underline{c})/(c-1) \rfloor + 1$, c'est-à-dire $\lceil (v - \underline{c})/(c-1) \rceil$ et la plus grande composante fortement connexe est contenue dans une composante connexe de taille $\lceil (v - \underline{c})/(c-1) \rceil$. Dans ce cas nous avons

$$\overline{c} = \lceil (v - \underline{c})/(c-1) \rceil = \max(\overline{s}, \lceil (v - \underline{c})/(c-1) \rceil) \tag{4.12}$$

2. Si nous avons $⑦\ \overline{s} \geq \lceil (v - \underline{c})/(c-1) \rceil$, nous construisons d'abord une composante connexe de taille $\overline{s}$ qui a une seule composante fortement connexe. Il reste donc $c-1$ composantes connexes à construire avec les $v - \overline{s}$ sommets restants. Comme il s'agit de déterminer une borne inférieure de la taille de la plus grande composante connexe, nous cherchons à avoir des composantes connexes de plus petites tailles possibles. Pour cela, nous partageons équitablement les sommets restants entre les $c-1$ composantes connexes restantes. Ainsi, ces autres composantes connexes auront au moins une taille égale à $\lfloor (v - \overline{s})/(c-1) \rfloor$. Cela est possible grâce à la condition de faisabilité (4.11) qui soutient que de telles composantes connexes ont une taille au moins égale à $\underline{c}$. Puis, si $(v - \overline{s}) \bmod (c-1) > 0$, il va rester $(v - \overline{s}) \bmod (c-1)$ sommets qui seront distribués à $(v - \overline{s}) \bmod (c-1)$ composantes connexes à raison d'un sommet par composante connexe afin de minimiser le nombre de sommets d'une composante connexe. Ainsi, la taille d'une composante connexe dans cette construction est $\lfloor (v - \overline{s})/(c-1) \rfloor$

ou $\lfloor (v-\overline{s})/(c-1) \rfloor + 1$, c'est-à-dire $\lceil (v-\overline{s})/(c-1) \rceil$. De plus, nous avons

$$\overline{s} \geq \left\lceil \frac{v-\underline{c}}{c-1} \right\rceil \geq \underline{c} \Longrightarrow \overline{s} \geq \underline{c} \tag{4.13}$$

$$\overline{s} \geq \underline{c} \Longleftrightarrow v - \overline{s} \leq v - \underline{c} \tag{4.14}$$

$$\Longleftrightarrow \frac{v-\overline{s}}{c-1} \leq \frac{v-\underline{c}}{c-1} \tag{4.15}$$

$$\Longleftrightarrow \left\lceil \frac{v-\overline{s}}{c-1} \right\rceil \leq \left\lceil \frac{v-\underline{c}}{c-1} \right\rceil \tag{4.16}$$

$$\text{Comme } \overline{s} \geq \left\lceil \frac{v-\underline{c}}{c-1} \right\rceil, \text{ on a } \overline{s} \geq \left\lceil \frac{v-\underline{c}}{c-1} \right\rceil \geq \left\lceil \frac{v-\overline{s}}{c-1} \right\rceil \Longrightarrow \overline{s} \geq \left\lceil \frac{v-\overline{s}}{c-1} \right\rceil \tag{4.17}$$

Au final, la taille de la plus grande composante connexe dans cette construction est

$$\overline{c} = \overline{s} = \max\left(\overline{s}, \lceil (v-\underline{c})/(c-1) \rceil\right) \tag{4.18}$$

□

4.2.2 Preuves de conjectures sur l'arbre enraciné

Dans cette section, nous prouvons deux conjectures sur des bornes supérieure et inférieure du nombre f de feuilles d'un arbre enraciné, découvertes par le Bound Seeker, et publiées dans l'article [35].

Conjecture 8. Cette conjecture exprime les bornes inférieure précise et supérieure précise du nombre f de feuilles d'un arbre enraciné en fonction du nombre v de sommets, du nombre minimal $\underline{d}$, et du nombre maximal $\overline{d}$ de fils d'un nœud n'étant pas une feuille de l'arbre.

$$\begin{cases} \underline{d} = 0 \Rightarrow f = 1 \\ \underline{d} > 0 \Rightarrow f \in \left[\left\lceil \dfrac{v \cdot \underline{d} + \overline{d} - \underline{d} - v + 1}{\underline{d}} \right\rceil, \left\lfloor \dfrac{v \cdot \overline{d} + \underline{d} - \overline{d} - v + 1}{\overline{d}} \right\rfloor \right] \end{cases} \tag{4.19}$$

Preuve de Conjecture 8. Le premier cas est évident, car $\underline{d} = 0$ signifie que l'arbre n'a qu'un seul sommet.

Pour prouver le deuxième cas $\underline{d} > 0$, nous commençons par prouver deux conjectures concernant l'objet combinatoire que constituent les partitions d'ensembles. Puis nous utilisons le transfert entre les partitions et les arbres enracinés donné dans l'exemple 28 du chapitre 3 pour prouver nos bornes sur le nombre de feuilles à partir des deux conjectures que nous allons prouver sur les partitions d'ensembles.

La preuve des deux conjectures des partitions d'ensembles est la suivante : en considérant l'objet combinatoire que constituent les partitions d'ensembles, soit O_i la taille de la i-ème partition et n le cardinal de l'ensemble partitionné en P partitions. Nous avons $n = \overline{M} + \sum_{i=1}^{P-1} O_i = \overline{M} + \underline{M} + \sum_{i=1}^{P-1} O_i \geq \overline{M} + P \cdot \underline{M}$. Et comme $P \in \mathbb{N}$, nous obtenons l'inégalité $P \leq \lfloor (n - \overline{M})/\underline{M} \rfloor$,

qui constitue la première conjecture prouvée pour les partitions d'ensembles. Nous avons également $n = \underline{M} + \sum_{i=2}^{P} O_i = \overline{M} - \underline{M} + \sum_{i=2}^{P} O_i \leq P \cdot \overline{M} - \underline{M}$, qui conduit à la deuxième conjecture prouvée pour les partitions d'ensembles $P \geq \lceil (n + \underline{M})/\overline{M} \rceil$.

Pour continuer, nous rappelons l'encodage d'une partition d'ensembles par un arbre enraciné de l'exemple 28 du chapitre 3.

En partitionnant les sommets en sous-ensembles tels qu'une partition est constituée des sommets d'une même composante connexe, nous obtenons, selon la définition 39 du chapitre 3, l'encodage $\mathcal{U}$ suivant :

— le cardinal n de l'ensemble à partitionner est le nombre de nœuds fils. La racine de l'arbre mise à part, tous les nœuds de l'arbre sont des nœuds fils. Donc nous avons

$$n = \mathcal{H}_n(v, f, \underline{d}, \overline{d}) = v - 1 \tag{4.20}$$

— le nombre P de parties de la partition est le nombre de nœuds pères. Les feuilles mises à part, tous les sommets sont des nœuds pères. Donc nous avons

$$P = \mathcal{H}_P(v, f, \underline{d}, \overline{d}) = v - f \tag{4.21}$$

— la taille minimum $\underline{M}$ d'une partie est le nombre minimum de nœuds fils d'un nœud père. Donc nous avons

$$\underline{M} = \mathcal{H}_{\underline{M}}(v, f, \underline{d}, \overline{d}) = \underline{d} \tag{4.22}$$

— la taille maximum $\overline{M}$ d'une partie est le nombre maximum de nœuds fils d'un nœud père. Donc nous avons

$$\overline{M} = \mathcal{H}_{\overline{M}}(v, f, \underline{d}, \overline{d}) = \overline{d} \tag{4.23}$$

Cet encodage prouve que les nœuds de l'arbre forment une partition avec le respect des conditions de faisabilités d'une partition. Ainsi, en appliquant ledit encodage, nous obtenons

$$\lceil (n + \underline{M})/\overline{M} \rceil \leq P \leq \lfloor (n - \underline{M})/\underline{M} \rfloor \tag{4.24}$$

$$\Leftrightarrow \lceil (v - 1 + (\overline{d} - \underline{d}))/\overline{d} \rceil \leq v - f \leq \lfloor (v - 1 - (\overline{d} - \underline{d}))/\underline{d} \rfloor \tag{4.25}$$

$$\Leftrightarrow v - \lfloor (v - 1 - (\overline{d} - \underline{d}))/\underline{d} \rfloor \leq \quad f \quad \leq v - \lceil (v - 1 + (\overline{d} - \underline{d}))/\overline{d} \rceil \tag{4.26}$$

$$\Leftrightarrow \lceil v - (v - 1 - (\overline{d} - \underline{d}))/\underline{d} \rceil \leq \; f \; \leq \lfloor v - (v - 1 + (\overline{d} - \underline{d}))/\overline{d} \rfloor \tag{4.27}$$

qui conduit à Conjecture 8. □

4.2.3 Preuves de conjectures sur les partitions d'ensembles

Dans cette section, nous prouvons deux conjectures concernant les bornes inférieure et supérieure de la somme S des carrés des tailles des partitions d'un ensemble, ainsi que deux conjectures sur la différence $\overline{M}$ entre les tailles maximale et minimale d'une partition d'ensemble. Ces quatre conjectures ont été découvertes par le Bound Seeker.

Conjecture 9. Cette conjecture exprime la borne inférieure précise de la somme S des carrés des tailles des parties de la partition d'un ensemble en fonction de la cardinalité n de l'ensemble, du nombre P de partitions, et des tailles maximale $\overline{M}$ et minimale $\underline{M}$ d'une partition,

$$S \geq -A^2 \cdot VV - A \cdot VV + 2 \cdot A \cdot NN + SS + NN \tag{4.28}$$

où VV est le nombre de partitions restantes en excluant la plus petite partition et la plus grande partition, et où NN est le nombre d'éléments restants de l'ensemble en excluant la plus petite partition et la plus grande partition.

$$VV = \begin{cases} P-2 & \text{si } P > 1 \\ 0 & \text{sinon} \end{cases} \tag{4.29}$$

$$NN = \begin{cases} n - \underline{M} - \overline{M} & \text{si } P > 1 \\ 0 & \text{sinon} \end{cases} \tag{4.30}$$

et où A est la taille moyenne arrondie à l'ordre inférieur des partitions restantes en excluant la plus petite et la plus grande. Nous avons $A = 0$ s'il n'y a qu'une ou deux partitions. SS est la somme des carrés des tailles de la plus petite et la plus grande partition. S'il n'y a qu'une seule partition, SS est le carré de sa taille.

$$A = \begin{cases} \lfloor \frac{NN}{VV} \rfloor & \text{si } P > 2 \\ 0 & \text{sinon} \end{cases} \tag{4.31}$$

$$SS = \begin{cases} \underline{M}^2 + \overline{M}^2 & \text{si } P > 1 \\ \underline{M}^2 & \text{sinon} \end{cases} \tag{4.32}$$

Pour prouver Conjecture 9, nous prouvons au préalable le théorème 10.

Théorème 10 (minimisation de $S = \sum_i^P y_i^2$)**.** *Soit $y_1, y_2, \cdots, y_P$ des entiers positifs dont la somme est égale à n, et qui de plus minimisent la somme des carrés $S = \sum_{i=1}^P y_i^2$. Alors $n \bmod v$ de ces entiers sont fixés à $\lfloor n/P \rfloor + 1$, et les autres entiers sont tous fixés à $\lfloor n/P \rfloor$.*

Preuve du théorème 10. Le théorème est prouvé par le fait que ces entiers $y_1, y_2, \ldots, y_P$ minimisent $\sum_{i=1}^P y_i^2$ si et seulement si $\forall i, j \in [1, P], y_j \geq y_i, y_i - y_j \leq 1$. En effet, s'il existe i et j avec $y_i - y_j \geq 2$, cela signifie que $(y_i-1)^2 + (y_j+1)^2 + \sum_{k \notin \{i,j\}}^P y_k^2 = \sum_k^P y_k^2 - 2(y_i - y_j) + 2 < \sum_k^P y_k^2$. Comme $(y_i - 1) + (y_j + 1) + \sum_{k \notin \{i,j\}}^P y_k = n$, l'inégalité stricte précédente montre que cette

décomposition $y_1 - 1, y_2 + 1, y_3, \ldots, y_P$ fournit une valeur de $\sum_{i=1}^{P} y_i^2$ qui est strictement plus petite que le minimum de $\sum_{i=1}^{P} y_i^2$ fourni par la décomposition. $y_1, y_2, \ldots, y_P$: ce qui est une contradiction. Et la seule façon d'avoir $y_i - y_j \leq 1$ avec $\sum_i^P y_i = n$, est de donner aux entiers, les valeurs indiquées par le théorème 10, sinon nous aurons toujours $\sum_{k=1}^{P} y_k \neq n$. □

Preuve de Conjecture *9.*

— Si $P = 1$, nous avons $n = \overline{M}$. Ainsi, en substituant les caractéristiques A, NN, VV et SS, la conjecture (4.28) se simplifie en $S \geq n^2$; ce qui est cohérent avec la définition des partitions d'ensembles.

— Si $P = 2$, nous avons $n = \overline{M} + \underline{M}$. Ainsi, en substituant les caractéristiques A, NN, VV et SS, la conjecture (4.28) se simplifie en $S \geq \overline{M}^2 + \underline{M}^2$; $S \geq \overline{M}^2 + \underline{M}^2$; ce qui est cohérent avec la définition des partitions d'ensembles.

— Si $P > 2$, nous avons $A = \lfloor \frac{NN}{VV} \rfloor$. Cela signifie que $NN = A \cdot VV + NN \bmod VV$. Ainsi après substitution de NN par $A \cdot VV + NN \bmod VV$ dans la conjecture (4.28) nous avons

$$S \geq -A^2 \cdot VV + 2 \cdot A^2 \cdot VV + 2 \cdot A \cdot (NN \bmod VV) + SS + NN \bmod VV \tag{4.33}$$

qui se simplifie en

$$S \geq A^2 \cdot VV + (2 \cdot A + 1) \cdot (NN \bmod VV) + SS \tag{4.34}$$

Ainsi, après la substitution de $(2 \cdot A + 1)$ par $(A+1)^2 - A^2$, et de SS par $\overline{M}^2 + \underline{M}^2$ dans la dernière inégalité, nous obtenons

$$S \geq \overline{M}^2 + \underline{M}^2 + (A+1)^2 \cdot (NN \bmod VV) + A^2 \cdot (VV - (NN \bmod VV)) \tag{4.35}$$

Cette dernière inégalité est prouvée par le théorème 10.

□

Conjecture 11. Cette conjecture exprime la borne supérieure précise de la somme S des carrés des tailles des partitions d'un ensemble en fonction de la cardinalité n de l'ensemble, du nombre P de partitions, et des tailles minimale $\underline{M}$ et maximale $\overline{M}$ d'une partition.

$$S \leq MID^2 + SM \cdot RR + SMIN \tag{4.36}$$

où RR est le nombre de partitions ayant la plus grande taille. Lorsque les partitions ont la même taille, RR vaut 0. MID est la taille d'une partition intermédiaire entre la plus grande et la plus petite lorsque le nombre des partitions ayant la plus grande taille est maximal. Si cette partition intermédiaire n'existe pas, alors MID est la taille de la plus petite partition.

$$RR = \begin{cases} \left\lfloor \frac{R}{\underline{M}} \right\rfloor & \text{si } \overline{M} > 0 \\ 0 & \text{sinon} \end{cases} \quad (4.37) \qquad MID = \begin{cases} \underline{M} + (R \bmod \overline{M}) & \text{si } \overline{M} > 0 \\ \underline{M} & \text{sinon} \end{cases} \quad (4.38)$$

R est le nombre d'éléments restants, une fois que $\underline{M}$ éléments ont été enlevés de chaque partition. *SM* est la différence entre les carrés des tailles de la plus grande partition et la plus petite partition. Finalement, *SMIN* est la somme des carrés des tailles des partitions quand elles sont toutes de taille minimale et lorsqu'une partition est exclue.

$$R = n - P \cdot \underline{M} \quad (4.39) \qquad SM = \overline{M}^2 - \underline{M}^2 \quad (4.40) \qquad SMIN = \underline{M}^2 \cdot (P-1) \quad (4.41)$$

Pour prouver Conjecture 11, nous commençons par prouver les cinq lemmes suivants :

Lemme 1. *S'il existe au moins deux partitions de tailles strictement comprises entre $\underline{M}$ et $\overline{M}$, c'est-à-dire, si leurs tailles sont $\underline{M} + r_1$ et $\underline{M} + r_2$ telles que $1 \leq r_1 \leq r_2 < \overline{M}$, alors S n'est pas maximale.*

Preuve. Nous avons

$$(\underline{M} + r_1 - 1)^2 + (\underline{M} + r_2 + 1)^2 = (\underline{M} + r_1)^2 + (\underline{M} + r_2)^2 + 2(r_2 - r_1) + 2 \quad (4.42)$$
$$> (\underline{M} + r_1)^2 + (\underline{M} + r_2)^2 \quad (4.43)$$

Soit $O_i, \forall i \in [1; P]$, les tailles des P partitions de n éléments. Cela signifie que nous avons $n = \sum_i^P O_i$ et $S = \sum_i^P O_i^2$. Les deux termes de (4.43) font partie des termes au carré de S. En d'autres mots, il est possible de retirer un élément de la partition de taille $\underline{M} + r_1$ et l'ajouter à la partition de taille $\underline{M} + r_2$ pour obtenir une somme plus grande des carrés des tailles des partitions sans affecter les autres termes au carré dans le calcul de S de telle manière que $\sum_i^P O_i = n$ reste inchangé. Cela prouve que S n'est pas maximale. □

Lemme 2. *Soient $\underline{\alpha}$ et $\overline{\alpha}$ le nombre de partitions qui ont respectivement les tailles $\underline{M}$ et $\overline{M}$. Si $R \bmod \overline{M} > 0$, alors il existe au moins une partition dont la taille est strictement comprise entre $\underline{M}$ et $\overline{M}$.*

Preuve. Par contradiction, supposons que $R \bmod \overline{M} > 0$ et toutes les partitions sont soit de taille $\underline{M}$ ou soit de taille $\overline{M}$. Par définition de R nous avons

$$R = \sum_{i=1}^{P} (O_i - \underline{M}) = \sum_{i=1}^{\overline{\alpha}} (\overline{M} - \underline{M}) + \sum_{i=1}^{\underline{\alpha}} (\underline{M} - \underline{M}) = \overline{\alpha} \cdot \overline{M} \quad (4.44)$$

ce qui contredit le fait que $R \bmod \overline{M} > 0$. Ainsi $R \bmod \overline{M} > 0$ implique qu'il existe une partition de taille strictement comprise entre $\underline{M}$ et $\overline{M}$. □

Lemme 3. *Soient $\underline{\alpha}$ et $\overline{\alpha}$ le nombre de partitions qui ont respectivement la taille $\underline{M}$ et $\overline{M}$. Si S est maximale et $\overline{M} > 0 \wedge R \bmod \overline{M} > 0$, alors il existe une seule partition de taille strictement comprise entre $\underline{M}$ et $\overline{M}$; de plus la taille de cette partition est égale à $\underline{M} + R \bmod \overline{M}$.*

Preuve. D'après le lemme 1, et comme S est maximale, il y a au plus une partition de taille strictement comprise entre $\underline{M}$ et $\overline{M}$; d'après le lemme 2, comme $R \bmod \overline{M} > 0$ il y a au moins une partition de taille strictement comprise entre $\underline{M}$ et $\overline{M}$. Donc il y a une seule partition de taille strictement comprise entre $\underline{M}$ et $\overline{M}$. Ainsi, soit O_P la taille de cette partition et $O_i, \forall i \in [1 : P-1]$ les tailles des partitions restantes. Soit également $r^* = O_P - \underline{M}$.

Alors nous avons $0 < r^* < \overline{M}$, car $\underline{M} < O_P < \overline{M}$. Ainsi, par définition, nous avons

$$R = \sum_{i=1}^{P}(O_i - \underline{M}) = \sum_{i=1}^{\overline{\alpha}}(\overline{M} - \underline{M}) + \sum_{i=\overline{\alpha}+1}^{\overline{\alpha}+\underline{\alpha}}(\underline{M} - \underline{M}) + r^* \tag{4.45}$$

$$= \overline{\alpha} \cdot \overline{M} + r^* \text{ avec } r^* < \overline{M} \tag{4.46}$$

D'après la définition de la division euclidienne, la relation (4.46) est équivalente à $r^* = R \bmod \overline{M}$. Ainsi $O_P = \underline{M} + R \bmod \overline{M}$. □

Lemme 4. *Soit $\underline{\alpha}$ et $\overline{\alpha}$ le nombre de partitions qui ont respectivement la taille $\underline{M}$ et $\overline{M}$. Si S est maximale et $\overline{M} > 0 \wedge R \bmod \overline{M} = 0$, alors il n'y a aucune partition de taille strictement comprise entre $\underline{M}$ et $\overline{M}$.*

Preuve. Par contradiction, supposons que S soit maximale, $\overline{M} > 0 \wedge (R \bmod \overline{M} = 0)$ et qu'il y a k ($k \geq 1$) partitions de taille strictement comprise entre $\underline{M}$ et $\overline{M}$ notés par $I_i, \forall i \in [1 : k]$. Alors, par définition

$$R = \sum_{i=1}^{\overline{\alpha}}(\overline{M} - \underline{M}) + \sum_{i=1}^{k}(I_i - \underline{M}) = \overline{\alpha} \cdot \overline{M} + \sum_{i=1}^{k}(I_i - \underline{M}) \tag{4.47}$$

$$\text{avec } I_i - \underline{M} < \overline{M}, \forall i \in [1 : k] \tag{4.48}$$

Comme $R \bmod \overline{M} = 0$, nous avons $R = \left\lfloor \frac{R}{\overline{M}} \right\rfloor \cdot \overline{M}$. Ainsi d'après (4.47), $\forall i \in [1 : k]$ avec $I_i - \underline{M} < \overline{M}$, nous avons

$$\overline{\alpha} \cdot \overline{M} + \sum_{i=1}^{k}(I_i - \underline{M}) = \left\lfloor \frac{R}{\overline{M}} \right\rfloor \cdot \overline{M} \tag{4.49}$$

Ainsi d'après (4.49), $\overline{M}$ est un diviseur de $\sum_{i=1}^{k}(I_i - \underline{M})$. Ce qui implique $k \geq 2$, à cause de (4.48). Et d'après le lemme 1, $k \geq 2$ implique que S n'est pas maximale. Ce qui est une contradiction. □

Lemme 5. *Soit $\underline{\alpha}$ et $\overline{\alpha}$ le nombre de partitions qui ont respectivement la taille $\underline{M}$ et $\overline{M}$. Donc $\left\lfloor \frac{R}{\overline{\underline{M}}} \right\rfloor$ $\left(\text{resp. } P - \left\lceil \frac{R}{\overline{\underline{M}}} \right\rceil \right)$ est une borne supérieure précise de $\overline{\alpha}$ (resp. $\underline{\alpha}$).*

Preuve. Nous bornons supérieurement les valeurs de $\overline{\alpha}$ en utilisant la définition de n. Nous avons :

$$n = \overline{M} \cdot \overline{\alpha} + \sum_{i=1}^{P-\overline{\alpha}} O_i \geq \overline{M} \cdot \overline{\alpha} + (P - \overline{\alpha}) \cdot \underline{M} \geq (\overline{M} - \underline{M}) \cdot \overline{\alpha} + P \cdot \underline{M} \tag{4.50}$$

$$\overline{\alpha} \leq \frac{n - P \cdot \underline{M}}{\overline{M} - \underline{M}} \tag{4.51}$$

Et par définition, $R = n - P \cdot \underline{M}$ et $\overline{\underline{M}} = \overline{M} - \underline{M}$. Ainsi :

$$\overline{\alpha} \leq \frac{R}{\overline{\underline{M}}} \tag{4.52}$$

De la même façon, nous bornons supérieurement $\underline{\alpha}$:

$$n = \underline{M} \cdot \underline{\alpha} + \sum_{i=1}^{P-\underline{\alpha}} O_i \leq \underline{M} \cdot \underline{\alpha} + \overline{M} \cdot (P - \underline{\alpha}) = -(\overline{M} - \underline{M}) \cdot \underline{\alpha} + \overline{M} \cdot P$$

$$\underline{\alpha} \leq \frac{\overline{M} \cdot P - n}{\overline{M} - \underline{M}} = \frac{(\overline{M} + \underline{M}) \cdot P - n}{\overline{\underline{M}}} = P + \frac{-(n - \underline{M} \cdot P)}{\overline{\underline{M}}} = P - \frac{R}{\overline{\underline{M}}}$$

Comme $\overline{\alpha}$, $\underline{\alpha} \in \mathbb{N}$, nous avons

$$\overline{\alpha} \leq \left\lfloor \frac{R}{\overline{\underline{M}}} \right\rfloor \qquad \text{et} \qquad \underline{\alpha} \leq P - \left\lceil \frac{R}{\overline{\underline{M}}} \right\rceil \tag{4.53}$$

Finalement, d'après les lemmes 3 et 4, lorsque S est maximale, nous avons l'égalité $R = \overline{\alpha} \cdot \overline{\underline{M}} + R \bmod \overline{\underline{M}}$ et il y a au plus une partition de taille strictement comprise entre $\underline{M}$ et $\overline{M}$. Cela signifie que lorsque S est maximale, nous avons :

$$\overline{\alpha} = \left\lfloor \frac{R}{\overline{\underline{M}}} \right\rfloor \tag{4.54}$$

$$\implies \underline{\alpha} = P - \overline{\alpha} - \beta = P - \left\lfloor \frac{R}{\overline{\underline{M}}} \right\rfloor - \beta \tag{4.55}$$

$$\text{avec } \beta = 0 \text{ si } R \bmod \overline{\underline{M}} = 0 \text{ et } \beta = 1 \text{ sinon.} \tag{4.56}$$

$$\implies \underline{\alpha} = P - \left\lfloor \frac{R}{\overline{\underline{M}}} \right\rfloor - \beta = P - \left\lceil \frac{R}{\overline{\underline{M}}} \right\rceil \tag{4.57}$$

□

Preuve de Conjecture 11.

— Supposons que $\overline{M} = \overline{M} - \underline{M} = 0$. Dans ce cas, les tailles de toutes les partitions sont égales. Alors d'après les égalités allant de (4.39) à (4.41) nous avons $MID = \underline{M}, SM = RR = 0, s_{min} = \underline{M}^2 \cdot (P-1)$. En remplaçant MID, SM, RR et $SMIN$ par leurs expressions respectives dans (4.36), nous obtenons

$$S \leq \underline{M}^2 + 0 + \underline{M}^2 \cdot (P-1) = \underline{M}^2 \cdot P \tag{4.58}$$

ce qui est cohérent avec la définition de S, car, comme les tailles des P partitions sont toutes égales, elles sont égales à $\underline{M}$. Ce qui signifie que $S = \underline{M}^2 \cdot P$.

— Supposons que $\overline{M} > 0 \wedge (R \bmod \overline{M} > 0)$, soient $\underline{\alpha}$ et $\overline{\alpha}$ le nombre de partitions qui ont respectivement la taille $\underline{M}$ et $\overline{M}$.

D'après le lemme 3 et le lemme 5, la valeur maximale S^* de S est

$$\begin{gathered} S^* = \overline{M}^2 \cdot \overline{\alpha} + \underline{M}^2 \cdot \underline{\alpha} + (\underline{M} + R \bmod \overline{M})^2 \\ \text{avec } \overline{\alpha} = \left\lfloor \frac{R}{\overline{M}} \right\rfloor \text{ et } \underline{\alpha} = P - \left\lfloor \frac{R}{\overline{M}} \right\rfloor - 1 \end{gathered} \tag{4.59}$$

Ainsi, comme $R \bmod \overline{M} > 0$, nous avons d'après les équations (4.39)–(4.41), $MID = \underline{M} + R \bmod \overline{M}, SM = \overline{M}^2 - \underline{M}^2, RR = \left\lfloor \frac{R}{\overline{M}} \right\rfloor, SMIN = \underline{M}^2 \cdot (P-1)$. En substituant MID, SM, RR et $SMIN$ dans (4.36), nous obtenons $S \leq S^*$ d'après (4.59). Ce qui est cohérent.

— Supposons que $\overline{M} > 0 \wedge (R \bmod \overline{M} = 0)$ et soient $\underline{\alpha}$ et $\overline{\alpha}$ le nombre de partitions qui ont respectivement pour taille $\underline{M}$ et $\overline{M}$.

D'après le lemme 4 et le lemme 5, la valeur maximale S^* de S est

$$\begin{gathered} S^* = \overline{M}^2 \cdot \overline{\alpha} + \underline{M}^2 \cdot \underline{\alpha} \\ \text{avec } \overline{\alpha} = \left\lfloor \frac{R}{\overline{M}} \right\rfloor \text{ et } \underline{\alpha} = P - \left\lfloor \frac{R}{\overline{M}} \right\rfloor \end{gathered} \tag{4.60}$$

Ainsi, comme $R \bmod \overline{M} = 0$, nous avons d'après les égalités (4.39)–(4.41), $MID = \underline{M}, SM = \overline{M}^2 - \underline{M}^2, RR = \left\lfloor \frac{R}{\overline{M}} \right\rfloor, SMIN = \underline{M}^2 \cdot (P-1)$. En remplaçant MID, SM, RR et $SMIN$ dans (4.36), nous obtenons $S \leq S^*$ d'après (4.60). Ce qui est cohérent.

□

Conjecture 12. Cette conjecture exprime la borne supérieure précise de la différence $\overline{M}$ entre la taille de la plus grande partition et la taille de la plus petite partition en fonction du

cardinal n de l'ensemble partitionné, du nombre P de partitions, et de la taille minimale $\underline{M}$ d'une partition.

$$\overline{M} \leq n - P \cdot \underline{M} \tag{4.61}$$

Preuve de Conjecture 12.

- Si $P = 1$, alors $\underline{M} = \overline{M} = n$. Donc $\overline{M} = 0$ et $n - P \cdot \underline{M} = n - \underline{M} = 0$. Ce qui implique $\overline{M} \leq n - P \cdot \underline{M}$.
- Si $P \geq 2$, alors la cardinalité de l'ensemble partitionné est égale à la somme des trois tailles suivantes :
  - la taille de la plus grande partition,
  - la taille de la plus petite partition,
  - la somme des tailles des autres partitions.

  $$n = \overline{M} + \underline{M} + \sum_{i=1}^{P-2} O_i \tag{4.62}$$

  Comme $\underline{M}$ est la taille de la plus petite partition, nous avons $O_i \geq \underline{M}$.

  $$n \geq \overline{M} + \underline{M} + \sum_{i=1}^{P-2} \underline{M} = \overline{M} + (P-1) \cdot \underline{M} \tag{4.63}$$

  $$n - P \cdot \underline{M} \geq \overline{M} - \underline{M} \tag{4.64}$$

  En utilisant la définition $\overline{\underline{M}} = \overline{M} - \underline{M}$ nous obtenons $n - P \cdot \underline{M} \geq \overline{\underline{M}}$.
- Précision de la borne (4.61) : Nous pouvons construire pour chaque valeur possible de n, P et $\underline{M}$ l'ensemble des partitions avec $n - P \cdot \underline{M} = \overline{\underline{M}}$, en mettant une seule partition à la taille $\overline{M} = n - (P-1) \cdot \underline{M}$ et le reste à la taille $\underline{M}$. En effet, dans ce cas nous avons :

  $$n = \overline{M} + (P-1) \cdot \underline{M} \tag{4.65}$$

  $$n - P \cdot \underline{M} = \overline{M} + (P-1) \cdot \underline{M} - P \cdot \underline{M} = \overline{M} - \underline{M} = \overline{\underline{M}} \tag{4.66}$$

□

Conjecture 13. Cette conjecture exprime la borne supérieure précise de la différence $\overline{\underline{M}}$ entre la taille de la plus grande partition et celle de la plus petite en fonction du cardinal n de l'ensemble partitionné, du nombre P de partitions et de la taille maximale $\overline{M}$ d'une partition.

$$\overline{\underline{M}} \leq \min(P \cdot \overline{M} - n, \overline{M} - 1) \tag{4.67}$$

Preuve de Conjecture 13.

— Si $P = 1$, alors $\underline{M} = \overline{M} = n$. Donc $\underline{\overline{M}} = 0$ et $P \cdot \overline{M} - n = \overline{M} - n = 0$. Ce qui implique $\underline{\overline{M}} \leq P \cdot \overline{M} - n$. Et comme $\overline{M} \geq 1$, nous avons aussi $\underline{\overline{M}} = 0$ et $0 \leq \overline{M} - 1$. Ce qui implique $\underline{\overline{M}} \leq \overline{M} - 1$. Comme $\underline{\overline{M}}$ est borné par deux quantités, il est borné par la plus petite et donc $\underline{\overline{M}} \leq \min(P \cdot \overline{M} - n, \overline{M} - 1)$.

— Si $P \geq 2$, alors nous montrons en premier $\underline{\overline{M}} \leq P \cdot \overline{M} - n$.

$$n = \overline{M} + \underline{M} + \sum_{i=1}^{P-2} O_i \tag{4.68}$$

$$n \leq \overline{M} + \underline{M} + \sum_{i=1}^{P-2} \overline{M} = (P-1)\overline{M} + \underline{M} \tag{4.69}$$

$$\overline{M} - \underline{M} \leq P \cdot \overline{M} - n \tag{4.70}$$

$$\underline{\overline{M}} \leq P \cdot \overline{M} - n \tag{4.71}$$

La plus grande valeur de $\underline{\overline{M}}$ est obtenue lorsqu'une des partitions a un seul élément et les $n - 1$ éléments restants constituent une autre partition. Ainsi, nous avons donc $\underline{\overline{M}} \leq \overline{M} - 1$.

Comme $\underline{\overline{M}}$ est borné par deux quantités, il est borné par la plus petite et donc $\underline{\overline{M}} \leq \min(P \cdot \overline{M} - n, \overline{M} - 1)$.

— Précision de la borne (4.67) :

Pour le cas $P = 1$, nous avons $\underline{M} = \overline{M} = n$. Donc $\underline{\overline{M}} = 0$ et $P \cdot \overline{M} - n = \overline{M} - n = 0$. Ce qui implique $\underline{\overline{M}} = P \cdot \overline{M} - n = 0$. Ainsi nous avons $0 = \min(P \cdot \overline{M} - n, \overline{M} - 1)$. Ce qui implique $\underline{\overline{M}} = \min(P \cdot \overline{M} - n, \overline{M} - 1)$. Donc la borne est précise.

Pour le cas $P \geq 2$, nous pouvons construire pour chaque valeur possible de n, P et $\overline{M}$ l'ensemble des partitions avec $P \cdot \overline{M} - n = \underline{\overline{M}}$ et $\overline{M} - 1 = \underline{M}$, soit en mettant une seule partition à la taille $\underline{M} = n - (P-1) \cdot \overline{M}$ et le reste à la taille $\overline{M}$ si $n > (P-1) \cdot \overline{M}$ ou soit en mettant une des partitions à la taille 1 si $n \leq (P-1) \cdot \overline{M}$. Parce que nous avons :

— Si $n > (P-1) \cdot \overline{M}$, alors $n = \underline{M} + (P-1) \cdot \overline{M}$. Ce qui implique $P \cdot \overline{M} - n = P \cdot \overline{M} - \underline{M} - (P-1) \cdot \overline{M}$. Ainsi $P \cdot \overline{M} - n = \overline{M} - \underline{M} = \underline{\overline{M}}$.

— Si $n \leq (P-1) \cdot \overline{M}$, nous avons $P > 2$ puisque si $P = 2$, cela implique $n \leq \overline{M}$ qui implique à son tour $n = \overline{M}$. Ceci signifie que $P = 1$. Ce qui est contradictoire avec $P = 2$. Ainsi pour atteindre la borne précise, nous mettons une partition à la taille $\overline{M}$ et l'autre à la taille 1. Et comme $P > 2$ nous pouvons mettre les $P - 2$ partitions restantes à la taille $\left\lfloor \frac{n - \overline{M} - 1}{P-2} \right\rfloor$ et à la taille $\left\lceil \frac{n - \overline{M} - 1}{P-2} \right\rceil$. En effet, nous avons $1 \leq \left\lfloor \frac{n - \overline{M} - 1}{P-2} \right\rfloor < \overline{M}$. Parce que premièrement, $n - \overline{M} - 1$ est le nombre d'éléments restants à partitionner en $P - 2$ partitions non vides. Donc

$P - 2 \leq n - \overline{M} - 1$. Ce qui conduit à $1 \leq \left\lfloor \frac{n - \overline{M} - 1}{P - 2} \right\rfloor$. Et deuxièmement, nous avons également $n \leq (P-1) \cdot \overline{M}$ qui équivaut à

$$n - \overline{M} - 1 \leq (P-1) \cdot \overline{M} - \overline{M} - 1 \tag{4.72}$$

$$\frac{n - \overline{M} - 1}{P - 2} \leq \frac{(P-2) \cdot \overline{M} - 1}{P - 2} \leq \overline{M} - \frac{1}{P-2} < \overline{M} \tag{4.73}$$

$$\Rightarrow \left\lfloor \frac{n - \overline{M} - 1}{P - 2} \right\rfloor < \overline{M} \tag{4.74}$$

□

4.3 Discussion

Dans ce chapitre, nous avons présenté les preuves de neuf conjectures générées par le Bound Seeker portant sur les bornes précises des caractéristiques des objets combinatoires tels les graphes orientés, les arbres enracinés et les partitions d'un ensemble. Pour des raisons de forme, de clarté et de capacités limitées par le fait que ces preuves sont longues et élaborées manuellement, nous n'avons pas traité l'aspect de la précision des bornes démontrées pour Conjecture 4 dans le cadre des graphes orientés et pour Conjecture 8 dans le cadre des arbres enracinés. Pour ces mêmes raisons, nous n'avons également pas insisté sur l'élaboration de quelques preuves de conjectures dans le cas des autres objets combinatoires présentés au chapitre 1. Un point que nous aurions voulu explorer dans ce chapitre est la preuve par projection d'une borne sur une autre ayant une caractéristique bornante en moins. En effet pour prouver la borne f_2 sur laquelle nous projetons une borne f_1 ayant une caractéristique bornante de plus, il suffit de prouver f_1 et de substituer le lien de projection adéquat dans f_1 pour retrouver f_2. C'est dans ce cadre que se révèle l'une des utilités des cartes générées par le Bound Seeker, car elles fournissent les liens adéquats pour prouver les bornes de type f_2 une fois que les bornes de type f_1 ont été prouvées. Par exemple, chaque borne prouvée dans ce chapitre s'exprime en fonction d'au moins trois caractéristiques bornantes. Donc toute borne qui s'exprime en fonction d'un sous-ensemble de ces caractéristiques bornantes se prouve par projection si le lien de projection est fourni par la carte à laquelle appartient la borne. Et cette façon de procéder est automatisable, car faire la substitution ou la projection c'est faire de la réécriture dans le domaine des preuves automatiques. Un autre point qui a été effleuré est l'exploitation des encodages entre objets combinatoires puisque la partie consacrée à l'encodage dans la preuve est la substitution d'une caractéristique par sa fonction d'encodage. Donc cette partie pourrait également être automatisée.

Pour terminer, une fois qu'une conjecture est prouvée, elle peut être utilisée pour résoudre efficacement des problèmes combinatoires courants. Nous montrons dans le chapitre 5 suivant comment exploiter de manière générale les conjectures générées par le Bound Seeker et dont la véracité a été prouvée. Puis nous illustrons une application en utilisant les quatre bornes

Conjecture 9–Conjecture 13, prouvées dans ce chapitre 4. Cela justifie le fait que nous n'ayons pas omis les preuves de la précision de ces bornes lors des démonstrations de ces dernières.

Chapitre 5

Vers la génération assistée de propagateurs pour les contraintes globales

Les solveurs de contraintes tirent leur efficacité des algorithmes de filtrage implémentés pour chaque contrainte. Au cours des trois dernières décennies, de nombreux efforts ont été déployés pour développer des algorithmes de filtrage efficaces capables d'élaguer l'espace de recherche [69; 10; 8]. Des humains ont conçu et mis en œuvre ces algorithmes. Des efforts ont également été faits pour prétraiter les modèles afin de les rendre plus efficaces à résoudre [66]. Ce prétraitement est généralement automatique et nécessite peu ou pas d'intervention humaine. Cependant, il existe peu de recherche sur la génération d'algorithmes de filtrage de contraintes globales avec une assistance automatisée [9; 39; 37].

Nous proposons une approche assistée par une machine pour synthétiser des algorithmes de filtrage pour les contraintes globales basées sur un objet combinatoire. En réutilisant le Bound Seeker [36], nous générons des centaines de relations entre caractéristiques d'un objet combinatoire. Nous présentons un algorithme qui sélectionne automatiquement parmi ces relations celles qui contribuent le plus à filtrer une contrainte et nous prouvons manuellement les relations sélectionnées. Nous considérons la contrainte Partition qui modélise les différentes manières de partitionner une collection d'objets en clusters. Nous utilisons cette contrainte pour résoudre le *problème de la répartition équilibrée des charges de cours dans les universités*. Ce problème est connu dans la littérature sous le sigle de BACP. Plutôt que de minimiser la taille de la plus grande partition, nous minimisons la somme des carrés des nombres de crédits par trimestre pour obtenir une répartition équilibrée. Pour trois modèles au BACP, nous montrons comment le filtrage inféré par le Bound Seeker améliore à la fois le temps de résolution et le coût de la solution trouvée : ce nouveau filtrage prouve l'optimalité pour toutes les instances de ce problème disponibles sur la librairie CSPLib [40].

Dans ce travail, nous considérons des problèmes de satisfaction ou d'optimisation dont la solution est donnée par un objet combinatoire, par exemple un graphe, une permutation, ou une partition. Les contraintes du problème sont imposées sur les caractéristiques de cet objet. Par exemple, étant donné un graphe $G = \langle V, E \rangle$ constitué de l'ensemble des sommets V et de l'ensemble des arêtes E, une solution au problème de l'existence d'un cycle hamiltonien est un sous-ensemble d'arêtes $E' \subseteq E$ qui forme un cycle unique visitant chaque sommet de G exactement une fois. Dans ce cas, l'objet combinatoire est un graphe dont les caractéristiques sont le nombre de sommets n, le nombre d'arêtes m, le degré minimum et maximum d'un sommet $\underline{d}$ et $\overline{d}$, et le nombre de cycles c. Une solution doit satisfaire les contraintes $m = n$, $\underline{d} = \overline{d} = 2$ et $c = 1$. Nous utilisons le Bound Seeker pour trouver de telles contraintes. Cependant, les contraintes que nous recherchons sont des bornes précises liées à un objet combinatoire.

Dans ce chapitre, nous concevons un algorithme qui sélectionne parmi les bornes générées par le Bound Seeker, un sous-ensemble de bornes pertinentes pour filtrer les domaines des variables d'un problème de satisfaction de contraintes dont la solution est une instance de l'objet combinatoire. Enfin, nous ajoutons au modèle de contraintes les inégalités associées aux bornes pertinentes sélectionnées. Le filtrage effectué par ces inégalités agit comme un algorithme de filtrage d'une contrainte globale encodant l'objet combinatoire.

Pour illustrer l'efficacité de cette méthode, nous choisissons comme objet combinatoire une partition. Les partitions sont utilisées dans de nombreux problèmes d'affectation de tâches pour répartir la charge de travail entre des personnes ou des machines, ou dans le problème d'élaboration de cursus académiques [40] pour regrouper les cours offerts dans un même trimestre. Un objectif commun à tous ces problèmes est d'obtenir des partitions de taille similaire. Il est courant de se rapprocher de cet objectif en minimisant la taille de la plus grande partition, car les algorithmes de filtrage pour cet objectif sont simples et efficaces. Des solutions plus équilibrées peuvent être obtenues en minimisant la taille des carrés des partitions. Cependant, minimiser la somme des carrés est plus difficile pour un solveur de contraintes et il existe peu d'algorithmes de filtrage traitant cela efficacement. C'est ici que l'exploitation des inégalités trouvées par le Bound Seeker devient primordiale pour obtenir un filtrage plus fort. Les contributions du chapitre 5 sont les suivantes :

1. Nous montrons comment générer automatiquement des algorithmes de filtrage pour les contraintes globales en synthétisant des contraintes redondantes efficaces qui peuvent être facilement ajoutées directement dans différents solveurs de contraintes.
2. Nous introduisons une méthode automatique pour sélectionner des conjectures pertinentes parmi un large ensemble de conjectures candidates pour filtrer une contrainte globale.
3. Nous améliorons la recherche de solutions mieux équilibrées aux problèmes de partitionnement en introduisant la contrainte Partition avec un encodage simple de cette

contrainte.

La section 5.1 rappelle les caractéristiques du Bound Seeker. La section 5.2 présente le système d'acquisition de propagateurs développé pour la génération et la sélection des contraintes redondantes qui fournissent un filtrage plus fort pour un objet combinatoire donné. La section 5.3 définit et reformule la contrainte PARTITION. La section 5.4 utilise le système introduit dans la section. 5.2 pour générer des contraintes redondantes pour la contrainte PARTITION, et la section 5.5 évalue l'impact de ces contraintes redondantes sur plusieurs modèles du BACP. Rappelons que les contraintes retenues ont été prouvées au chapitre 4. Enfin, la section 5.6 conclut.

5.1 La découverte automatique de conjectures

Soit $\mathcal{O}$ un objet combinatoire qui a m caractéristiques $x_1, x_2, \ldots, x_m$ prenant des valeurs entières positives et telles que la valeur maximale de la première caractéristique x_1 restreint l'ensemble des valeurs de $x_2, \ldots, x_m$. Le Bound Seeker prend en entrée un nombre fini d'instances extrêmes pour $\mathcal{O}$ et apprend les inégalités de la forme $x_i \leq u_i(P)$ ou $x_i \geq l_i(P)$, où l_i et u_i sont des fonctions symboliques et $P \subseteq \{x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_m\}$. Ainsi, il peut y avoir jusqu'à $m2^m$ bornes générées. Ces inégalités bornent la caractéristique x_i, et le Bound Seeker garantit que ces bornes sont précises, c'est-à-dire qu'il existe une instance de $\mathcal{O}$ pour laquelle l'égalité est vraie. Ces bornes sont appelées *conjectures*, car elles ont été apprises à partir de certaines instances de $\mathcal{O}$ et peuvent ne pas s'appliquer à toutes les instances possibles de l'objet combinatoire. Transformer ces conjectures en théorèmes nécessite une preuve formelle écrite par un humain ou un assistant de preuve de théorèmes.

5.2 Le système d'acquisition des propagateurs

Le *système d'acquisition des propagateurs* décrit dans cette section prend un objet combinatoire en entrée et retourne un ensemble de contraintes. Pour un objet combinatoire donné, ces contraintes sont calculées une fois pour toutes et stockées dans une base de données afin de pouvoir ensuite être ajoutées à un modèle dont la solution est une instance de cet objet combinatoire. L'ajout de ces contraintes au modèle fournit un filtrage plus fort.

5.2.1 Génération de contraintes candidates

Nous commençons par un problème de satisfaction de contraintes dont la solution est exprimée sous la forme d'instance d'un objet combinatoire, comme un graphe, une permutation, une partition. Nous identifions les caractéristiques $x_1, x_2, \ldots, x_m$ de cet objet combinatoire qui sont importantes pour le modèle. Ces caractéristiques sont des valeurs numériques qui caractérisent la solution. Pour un graphe orienté, cela peut être par exemple le nombre de sommets, d'arcs, de composantes connexes, de composantes fortement connexes. Habituellement, la définition

du problème que nous voulons résoudre va naturellement restreindre les valeurs possibles de plusieurs de ces caractéristiques.

Le Bound Seeker [36] prend en entrée des instances d'un objet combinatoire et une liste de caractéristiques qui peuvent être calculées sur ces objets. Il calcule un ensemble de conjectures $\mathcal{C} = \{C_1, \ldots, C_{|\mathcal{C}|}\}$. Chaque conjecture C_i est de la forme $x_i \leq u_i(x_{j_1}, x_{j_2}, \ldots)$ ou $x_i \geq l_i(x_{j_1}, x_{j_2}, \ldots)$. Ces inégalités pourraient être ajoutées directement au modèle du problème de satisfaction de contraintes pour améliorer le filtrage. Même si, en pratique, le Bound Seeker est incapable de trouver toutes les $m2^m$ conjectures distinctes possibles dans un temps raisonnable, il est néanmoins capable d'en calculer plusieurs centaines. Le temps passé à filtrer ces contraintes pourrait dépasser le temps gagné en élaguant l'arbre de recherche. Enfin, de nombreuses conjectures pourraient filtrer les mêmes valeurs, et certaines conjectures pourraient voir leur filtrage dominé par d'autres.

L'algorithme 2 est un algorithme glouton qui supprime certaines conjectures de l'ensemble des conjectures $\mathcal{C}$ sans pour autant réduire le filtrage effectué. Il renvoie un sous-ensemble minimal de conjectures au sens où la suppression de toute conjecture de ce sous-ensemble réduirait le filtrage.

Pour sélectionner des conjectures, l'algorithme doit résoudre un problème de satisfaction de contraintes. Soit $M(\mathcal{C})$ un problème de satisfaction de contraintes capable d'énumérer toutes les combinaisons possibles de valeurs pour les caractéristiques $x_1, x_2, \ldots, x_m$ par rapport à une borne supérieure ℓ pour x_1. Soit $I(\mathcal{O})$ l'ensemble des instances de l'objet combinatoire. Le modèle $M(\mathcal{C})$ est défini comme suit :

$$M(\mathcal{C}) \iff x_1 \leq \ell \tag{5.1}$$

$$(x_1, x_2, \ldots, x_m) \in I(\mathcal{O}) \tag{5.2}$$

$$C_1 \wedge C_2 \wedge \ldots \wedge C_{|\mathcal{C}|} \tag{5.3}$$

$$x_i \in \mathcal{D}(x_i) \qquad \forall i \in \{1, \ldots, m\}. \tag{5.4}$$

La contrainte (5.1) borne une caractéristique (souvent choisie comme étant la taille de l'instance de l'objet combinatoire) pour rendre fini le nombre de solutions au problème. La contrainte (5.2) force les caractéristiques à être cohérentes avec la définition de l'objet combinatoire. En d'autres termes, les caractéristiques doivent prendre des valeurs pour lesquelles il existe une instance de l'objet combinatoire. Par exemple, un graphe orienté ne peut pas avoir dix arcs avec seulement deux sommets. La contrainte (5.2) peut être soit omise, soit codée selon les préférences du modélisateur. Cela peut être considéré comme une connaissance préalable de la contrainte par le modélisateur. Il peut s'agir d'une décomposition en contraintes qui définissent entièrement les valeurs possibles des caractéristiques. En cas d'omission, le système apprend un algorithme de filtrage à partir de zéro. La contrainte (5.3) est la conjonction des conjectures dans $\mathcal{C}$.

Lors de l'énumération de toutes les solutions de $M(\mathcal{C})$, l'algorithme 2 garde une trace du nombre de retours arrière rencontrés par le solveur et du nombre de solutions trouvées. Si le nombre de retours arrière et le nombre de solutions sont les mêmes pour $M(\mathcal{C})$ et $M(\mathcal{C} \setminus \{C_i\})$, alors la conjecture C_i ne filtre pas les domaines $\mathcal{D}(x_1), \mathcal{D}(x_2), \ldots, \mathcal{D}(x_m)$ plus que les autres conjectures. La conjecture C_i peut être rejetée. Le modèle énumère les tuples de valeurs des caractéristiques $(x_1, x_2, \ldots, x_m)$ au lieu d'énumérer les instances de l'objet combinatoire. Par conséquent, la recherche se confronte à davantage de retours arrière, car plusieurs instances de l'objet combinatoire peuvent avoir un même tuple de valeurs des caractéristiques $(x_1, x_2, \ldots, x_m)$. Il est donc plus facile de déterminer quelle contrainte fait réellement la différence d'un point de vue du filtrage.

Algorithme 2 : Sélection de conjectures

Entrée : Un ensemble de conjectures $\mathcal{C} = \{C_1, C_2, \ldots, C_{|\mathcal{C}|}\}$ provenant du Bound Seeker.
Sortie : Un ensemble minimal de conjectures avec un niveau de filtrage équivalent à $\mathcal{C}$.

```
1 Soit nback le nombre de retours arrière, et nsol le nombre de solutions obtenues en
    énumérant toutes les solutions de M(C);
2 for k = |C| down to 1 do
3     Soit nback' le nombre de retours arrière et nsol' le nombre de solutions obtenues en
        énumérant toutes les solutions de M(C \ {C_i});
4     if nback = nback' ∧ nsol = nsol' then
5         C ← C \ {C_i}
6 return C;
```

L'ordre dans lequel les conjectures sont traitées dans la boucle « for » peut affecter le résultat final. En pratique, nous traitons les conjectures par nombre croissant de caractéristiques bornantes. En d'autres termes, en éliminant d'abord les conjectures avec peu de caractéristiques bornantes, nous favorisons les conjectures avec de nombreuses caractéristiques bornantes dans la sélection finale. En effet, plus une conjecture possède des caractéristiques bornantes, plus la borne sera précise.

Une fois les conjectures sélectionnées, elles peuvent désormais être prouvées à l'aide d'un assistant de preuve de théorèmes ou par un humain. Ces conjectures deviennent des théorèmes et le filtrage des inégalités exprimées par ces théorèmes garantit un algorithme de filtrage correct, c'est-à-dire n'écartant aucune solution réalisable.

5.3 Application à la contrainte Partition

Les humains sont intéressés par des solutions équilibrées aux problèmes d'affectation. Il s'agit d'une bonne application de la programmation par contraintes. Des exemples typiques de tels problèmes d'affectation sont :

— des problèmes de *clustering* où la taille des clusters doit être strictement équilibrée ;

— des problèmes de *routage des véhicules* où le nombre de clients visités par les différents véhicules doit être presque le même ;

— des problèmes de *calendrier* où la quantité de travail, ou le temps affecté à chaque travailleur, ou la quantité de cours à valider par les étudiants pour chaque session d'un programme académique doivent être répartis le plus uniformément possible.

Dans ce contexte, plusieurs contraintes globales, telles que NVALUE [58] et BALANCE [10; 15], ont été introduites en programmation par contraintes. NVALUE contraint un ensemble de valeurs à ne contenir qu'un nombre fixé de valeurs distinctes, tandis que BALANCE contraint un ensemble de valeurs à ne contenir que les occurrences les plus équilibrées des valeurs distinctes ; ces contraintes sont unifiées par la contrainte PARTITION que nous définissons maintenant.

Définition 41 (Contrainte PARTITION)**.** Soit un vecteur $\mathcal{X} = [X_1, X_2 \ldots, X_n]$ d'entiers qui encode une partition de la façon suivante. La valeur $i \in U$ appartient au sous-ensemble S_j si et seulement si $X_i = j$. La partition a comme caractéristiques : n qui est le nombre de composantes du vecteur, P qui est le nombre de valeurs distinctes dans $\mathcal{X}$, $\underline{M}$ (resp. $\overline{M}$) qui est le nombre d'occurrences de l'entier le moins (resp. plus) fréquent dans $\mathcal{X}$, $\underline{\overline{M}}$ qui est la différence $\overline{M} - \underline{M}$, S qui est la somme des carrés du nombre d'occurrences pour chaque entier distinct dans $\mathcal{X}$.

Nous définissons la contrainte

$$\text{PARTITION}([X_1, X_2, \ldots, X_n], P, \underline{M}, \overline{M}, \underline{\overline{M}}, S) \tag{5.5}$$

comme satisfaite si et seulement si

$$P = |\{X_1, X_2, \ldots, X_n\}| \tag{5.6}$$

$$\underline{M} = \min_j |\{i \mid X_i = j\}| \tag{5.7}$$

$$\overline{M} = \max_j |\{i \mid X_i = j\}| \tag{5.8}$$

$$\underline{\overline{M}} = \underline{M} - \overline{M} \tag{5.9}$$

$$S = \sum_{j \in \mathcal{X}} |\{i \mid X_i = j\}|^2 \tag{5.10}$$

5.3.1 Exemple de vecteur de la contrainte PARTITION

Étant donné le vecteur $\mathcal{X} = [1, 5, 6, 1, 1, 1, 1, 1, 1, 1, 6]$, la contrainte PARTITION$(\mathcal{X}, 3, 1, 8, 7, 69)$ est satisfaite, car le vecteur $\mathcal{X}$ contient 3 valeurs distinctes dont la moins (resp. plus) fréquente est la valeur 5 (resp. 1) avec 1 (resp. 8) occurrences, la différence entre le nombre d'occurrences de la valeur la plus fréquente et le nombre d'occurrences de la valeur la moins fréquente étant $8 - 1 = 7$, et la somme des carrés du nombre d'occurrences des valeurs distinctes étant $8^2 + 2^2 + 1^2 = 69$.

5.3.2 Une décomposition de la contrainte Partition

La contrainte Partition peut être reformulée à l'aide d'une contrainte de cardinalité globale (GCC) [68; 10] et de la contrainte NValue [58] comme suit :

$$\text{NValue}([X_1, X_2, \ldots, X_n], P) \tag{5.11}$$

$$\text{GCC}([X_1, X_2, \ldots, X_n], [O_1, O_2, \ldots, O_v]) \tag{5.12}$$

$$\sum_{j=1}^{v} O_j = n \ \wedge \ \sum_{i=1}^{n} X_i = \sum_{j=1}^{v} j \cdot O_j \tag{5.13}$$

$$\text{MinimumExcept0}([O_1, O_2, \ldots, O_v], \underline{M}) \tag{5.14}$$

$$\overline{M} = \max(O_1, O_2, \ldots, O_v) \tag{5.15}$$

$$\overline{\underline{M}} = \overline{M} - \underline{M} \ \wedge \ S = \sum_{j=1}^{v} O_j^2 \tag{5.16}$$

La contrainte de cardinalité globale GCC [68; 10] contraint les valeurs distinctes d'un multi-ensemble à avoir chacune une occurrence donnée. Les contraintes (5.13) sont redondantes, mais améliorent le filtrage. Soit v une borne supérieure du nombre de partitions. Par exemple, on peut fixer $v = \max_i(\max(\mathcal{D}(X_i)))$. Étant données les variables $O_1, O_2, \ldots, O_v$ dont les domaines se situent dans l'intervalle $[0, n]$, la contrainte MinimumExcept0$([O_1, O_2, \ldots, O_v], \underline{M})$ est satisfaite si et seulement si (i) au moins une des variables de $O_1, O_2, \ldots, O_v$ est affectée à une valeur qui est strictement plus grande que 0, et (ii) $\underline{M}$ est la plus petite valeur de ces variables $O_1, O_2, \ldots, O_v$ qui sont affectées à une valeur strictement plus grande que 0. En introduisant les variables auxiliaires $A_0, A_1, \ldots, A_v$, la contrainte MinimumExcept0$([O_1, O_2, \ldots, O_v], \underline{M})$ peut être reformulée par $A_0 = n$, $A_n = \underline{M}$, et pour tout $i \in [1, v]$ $O_i = 0 \Rightarrow M_i = M_{i-1}$, $O_i > 0 \wedge O_i \geq M_{i-1} \Rightarrow M_i = M_{i-1}$, $O_i > 0 \wedge O_i < M_{i-1} \Rightarrow M_i = O_i$.

5.3.3 Défi associé à la contrainte Partition

La principale faiblesse de la décomposition précédente est que, si la plupart des caractéristiques de la contrainte Partition sont liées aux variables $X_1, X_2, \ldots, X_n$, il n'y a pratiquement aucun lien direct entre n, P, $\underline{M}$, $\overline{M}$, $\overline{\underline{M}}$ et S, en dehors de l'égalité $\overline{\underline{M}} = \overline{M} - \underline{M}$. Puisque ces caractéristiques ne varient pas indépendamment, cela pose un problème pour obtenir un algorithme de filtrage efficace. Pour atténuer ce problème, nous montrons dans la section suivante comment extraire ces liens manquants.

5.4 Un algorithme de filtrage généré avec une collaboration homme-machine pour la contrainte Partition

Lorsque l'informaticien modélise un problème combinatoire en problème de satisfaction de contraintes ou problème d'optimisation combinatoire, il arrive que certaines contraintes pertinentes au sens du filtrage lui échappent. Nous présentons maintenant une application de l'algorithme 2 qui assiste l'informaticien dans la conception d'un modèle de satisfaction de contraintes ou d'optimisation combinatoire adéquat pour modéliser parfaitement le problème combinatoire. Ledit algorithme sélectionne les contraintes de bornes les plus influentes au sens du filtrage dans le cadre de la résolution d'un problème faisant intervenir les partitions d'ensembles. Grâce à cette sélection, le solveur appelle les algorithmes de filtrage adéquats pour chaque contrainte du modèle de satisfaction de contraintes ou d'optimisation combinatoire pendant la résolution du problème de satisfaction de contraintes ou du problème d'optimisation combinatoire.

5.4.1 Génération des conjectures

La première étape consiste à générer les conjectures à l'aide du Bound Seeker. Nous avons fourni en entrée toutes les instances de partitions $I(\mathcal{O})$ de taille $n \leq 30$. Pour chaque objet, nous donnons la valeur des caractéristiques n, $\underline{M}$, $\overline{M}$, $\overline{\underline{M}}$ et S qui intéressent la contrainte. Pour aider le Bound Seeker à exprimer les relations entre les caractéristiques n, $\underline{M}$, $\overline{M}$, $\overline{\underline{M}}$ et S, nous avons également introduit manuellement un ensemble de *caractéristiques secondaires* au niveau de l'objet de partition, que le Bound Seeker utilise pour rechercher systématiquement toutes les bornes précises des caractéristiques de l'objet de partition. La majorité de ces caractéristiques secondaires correspondent à des expressions conditionnelles, où la valeur associée au cas de base est soit liée uniquement à la taille de la plus petite partition, soit la constante 0. Comme les caractéristiques n, $\underline{M}$, $\overline{M}$, $\overline{\underline{M}}$ et S se focalisent sur les partitions les plus grandes et les plus petites, les caractéristiques secondaires fournissent des statistiques sur les autres partitions. Ces caractéristiques secondaires ont été introduites au chapitre précédent lors de l'élaboration des preuves des conjectures allant de la Conjecture 9 à la Conjecture 13 sur les partitions d'ensembles. Ce sont notamment les caractéristiques VV, NN, A, SS, RR, MID, R, SM et $SMIN$ qui ont pour formules respectives (4.29), (4.30), (4.31), (4.32), (4.37), (4.38), (4.39) et (4.41). Bien que ces caractéristiques soient introduites manuellement, le Bound Seeker les sélectionne ensuite automatiquement parmi un ensemble plus large de caractéristiques secondaires sous condition qu'elles dépendent fonctionnellement des caractéristiques bornantes de la caractéristique à borner.

Nous générons des instances de partitions $I(\mathcal{O})$, suivant les combinaisons valides de toutes leurs caractéristiques. Ces instances ont été générées jusqu'à des partitions de 30 éléments en moins d'une journée. Pour les générer, le solveur résout un problème de satisfaction de contraintes

contenant des contraintes cassant les symétries. Les partitions sont générées telles que les différentes parties d'une partition sont triées par taille croissante. Ces instances sont envoyées au Bound Seeker, qui renvoie une centaine de conjectures. Pour les bornes qui s'expriment en fonction de quatre caractéristiques bornantes, nous augmentons le temps de recherche à sept jours. En effet, Gindullin et *al.* [36] exécutent une recherche limitée à des bornes qui s'expriment avec au plus trois caractéristiques bornantes, pendant quatre jours.

5.4.2 Application de l'algorithme de sélection à la contrainte Partition

Pour exécuter l'algorithme 2, nous définissons $M(\mathcal{C})$ comme suit. On choisit comme caractéristiques les variables $\underline{M}$, $\overline{M}$, $\overline{\underline{M}}$, et S dont les domaines respectifs sont $[1, n]$, $[1, n]$, $[0, n-1]$ et $[n, n \cdot v]$. La contrainte (5.2) est codée à l'aide de la décomposition (5.11) en (5.16). Pour implémenter la contrainte (5.1), nous imposons la limite $n \leq 10$.

Parmi une centaine de conjectures renvoyées par le Bound Seeker et données en entrée de l'algorithme 2, ce dernier a sélectionné quatre conjectures parmi 93 en 5 secondes.[1] La première (5.17) donne une borne inférieure précise sur S en utilisant les quatre caractéristiques secondaires introduites dans (4.29)–(4.32) au chapitre précédent.

$$S \geq -A^2 \cdot VV - A \cdot VV + 2 \cdot A \cdot NN + SS + NN \tag{5.17}$$

avec

$$A = \begin{cases} \lfloor \frac{NN}{VV} \rfloor & \text{si } P > 2 \\ 0 & \text{sinon} \end{cases} \tag{5.18}$$

et

$$SS = \begin{cases} \underline{M}^2 + \overline{M}^2 & \text{si } P > 1 \\ \underline{M}^2 & \text{sinon} \end{cases} \tag{5.19}$$

Remarquons que les définitions de la caractéristique A par (5.18) et de la caractéristique SS par (5.19) conduisent au fait suivant : si dans un partitionnement d'ensemble, il y a au moins trois partitions, c'est-à-dire que $P > 2$, alors nous avons d'après la définition (5.18) de la caractéristique A, l'expression $A = \lfloor \frac{NN}{VV} \rfloor$ qui implique $NN = A \cdot VV + NN \mod VV$. Dans ce cas, la borne peut donc être exprimée comme (5.20) :

$$S \geq \overline{M}^2 + \underline{M}^2 + (A+1)^2 \cdot (NN \bmod VV) + A^2(VV - (NN \bmod VV)). \tag{5.20}$$

L'intuition derrière cette expression (5.20) de la borne inférieure précise de S est : pour obtenir la plus petite somme de carrés possible, la plus grande partition a une taille $\overline{M}$, la plus petite

1. L'algorithme 2 est écrit dans SICStus Prolog à l'aide du solveur clp(FD) fonctionnant sur Intel Core i7 à 3,1 Ghz. La taille maximale d'une instance de PARTITION est $n = 10$ éléments. Pour une taille plus grande, l'algorithme converge vers les quatre mêmes contraintes.

partition a une taille $\underline{M}$ et les autres partitions ont une taille NN/VV. Mais comme la taille doit être un nombre entier, les partitions $NN \mod VV$ ont leur taille arrondie à $A+1$, tandis que le reste des partitions ont leur taille arrondie à A. Ainsi, la borne exprime comment répartir les éléments entre les partitions afin d'obtenir la valeur minimale de S, c'est-à-dire la valeur de S pour laquelle la répartition est la plus équilibrée.

Les trois conjectures restantes sélectionnées par l'algorithme 2 ont été aussi introduites au chapitre précédent et sont les suivantes : une borne supérieure (5.21) sur S et deux bornes supérieures distinctes (5.22) et (5.23) sur $\overline{M}$.

$$S \leq MID^2 + SM \cdot RR + SMIN \tag{5.21}$$

$$\overline{M} \leq n - P \cdot \underline{M} \tag{5.22}$$

$$\overline{M} \leq \min(P \cdot \overline{M} - n, \overline{M} - 1) \tag{5.23}$$

Les preuves des quatre bornes sélectionnées (5.17), (5.21), (5.22), (5.23) élaborées au chapitre précédent (Conjecture 9 à la Conjecture 13), nous assurent que le filtrage est correct et sans perte de solutions lorsqu'on ajoute ces bornes à tout modèle de problème de satisfaction de contraintes dont $X_1, X_2, \ldots, X_n$ font partie de l'ensemble des variables.

5.5 Expérimentation sur le problème de répartition équilibré des charges de cours d'étudiants dans l'élaboration des cursus académiques (BACP)

Nous évaluons le système d'acquisition de propagateurs en l'appliquant sur le BACP [40]. Le BACP est un problème combinatoire dans lequel un ensemble de n cours doit être affecté à v périodes de temps. Soit X_i la période du cours i. Une borne inférieure $\underline{m}$ et une borne supérieure $\overline{m}$ données du nombre de cours par période doivent être respectées. Soit $\mathcal{P}rec$ l'ensemble des paires de cours (i, k) tel que i soit un prérequis pour k, et soit $c(i)$ le nombre de crédits du cours i . La *charge totale par période*, c'est-à-dire la somme des crédits des cours attribués pour chaque période, doit respecter une borne inférieure $\underline{L}_0$ et une borne supérieure $\overline{L}_0$. La charge de cours doit être équilibrée sur les différentes périodes. Dans la littérature reliée à la programmation par contraintes, il existe de nombreuses façons d'équilibrer la charge avec différentes fonctions objectives comme l'illustrent les articles [15; 67; 51; 32; 27]. Il est possible de minimiser la charge maximale ou la plage de charge (différence entre la charge maximale et la charge minimale). Pour répartir la charge encore plus uniformément, nous choisissons de minimiser la somme des carrés des charges par périodes. L'exemple 31 illustre le fait que minimiser la somme des carrés des charges est un bon critère pour équilibrer les charges entre les périodes.

Exemple 31. Comme présenté dans la table 5.1, supposons que 14 crédits soient répartis sur 4 sessions comme suit : 6 crédits à la première session, 1 crédit à la deuxième session, 5 crédits à la troisième session et 2 crédits à la quatrième session. Considérons une autre solution où la charge de travail est répartie comme suit : 6 crédits à la première session, 1 crédit à la deuxième session, 4 crédits à la troisième session et 3 crédits à la quatrième session.

La deuxième solution est plus équilibrée que la première, puisqu'avoir une session chargée avec 5 crédits, suivie d'une session peu chargée avec 2 crédits, est moins équilibrée que d'avoir deux sessions moyennes de 4 et 3 crédits.

Si l'on veut choisir la solution qui a la plus petite des charges parmi la plus grande charge de la première solution qui vaut 6 et la plus grande charge de la deuxième solution qui également 6, alors les deux solutions se valent.

Si l'on veut choisir la solution qui a la plus petite des différences de charges parmi la différence entre la plus grande charge et la plus petite de la première solution qui vaut $6 - 1 = 5$, et la différence entre la plus grande charge et la plus petite de la deuxième solution qui également $6 - 1 = 5$, alors les deux solutions se valent.

Si l'on veut choisir la solution qui a la plus petite somme des carrés des charges par période parmi la somme des carrés des charges par période de la première solution qui vaut 66, et la somme des carrés des charges par période de la deuxième solution qui également 62, alors la deuxième solution est bien celle qui sera choisie, c'est-à-dire la plus équilibrée.

	Sessions :				Critères d'équilibrage :		
	1^e	2^e	3^e	4^e	Charge maximale	Plage de charge	Somme des carrés des charges
Répartition des crédits de la 1^e solution	6	1	5	2	6	6 - 1 = 5	$6^2 + 1^2 + 5^2 + 2^2 = 66$
Répartition des crédits de la 2^e solution	6	1	4	3	6	6 - 1 = 5	$6^2 + 1^2 + 4^2 + 3^2 = 62$

TABLE 5.1 – Cette table montre que seule la somme des carrés des charges permet de départager le niveau d'équilibrage des deux solutions de répartition des crédits dans quatre sessions.

Pour montrer l'utilisation du système d'acquisition de propagateurs, nous codons d'abord le BACP en utilisant la contrainte PARTITION pour mettre en évidence la relation entre le BACP et l'objet combinatoire PARTITION. Nous testons également un deuxième modèle développé par Mats Carlsson [40] utilisant la contrainte CUMULATIVE [7; 10], et un troisième modèle utilisant des variables 0/1. Enfin, nous comparons les performances de ces différents modèles avec et

sans l'ajout des quatre conjectures sélectionnées par notre système d'acquisition d'algorithmes de filtrage.

5.5.1 Un premier modèle utilisant la contrainte Partition

Le premier modèle contient les contraintes (5.24) à (5.33). Le tableau $[X_1, X_2, \ldots, X_n]$ partitionne les cours $1 \ldots n$ en périodes $1 \ldots v$. Les variables $X_1, X_2, \ldots, X_n$ prennent les valeurs des périodes $1 \ldots v$. Les contraintes (5.25), (5.27), (5.28) et (5.29) imposent une borne inférieure et supérieure au nombre de cours enseignés par période. La contrainte (5.30) impose des priorités entre les cours correspondants i et k. La contrainte Partition (5.31) impose que le paramètre S soit la somme des carrés pour la charge de chaque période de temps. Le tableau $[\overbrace{X_1, \ldots,\ X_1}^{c(1) \text{ fois}}, \ldots, \overbrace{X_n, \ldots,\ X_n}^{c(n) \text{ fois}}]$ de la contrainte Partition (5.31) contient $c(i)$ occurrences de la variable X_i pour chaque cours i. Une occurrence d'une valeur de X_i représente un crédit du cours i. Comme X_i prend une valeur d'une période de temps, le nombre d'occurrences d'une période de temps j dans le tableau est la somme des crédits de tous les cours de la période de temps concernée. En d'autres termes, la charge totale de cours pour la période de temps j est $\sum_{i=1}^{n} [X_i = j] \cdot c(i)$. La même contrainte Partition (5.31) assure que $\underline{L}$ est la plage de charge entre la charge minimale $\underline{L}$ et la charge maximale $\overline{L}$ par période de temps. La contrainte (5.32) assure le respect des bornes inférieures et supérieures de la charge par période de temps. Les inégalités (5.33) définissent les domaines initiaux des variables X_i et garantissent que les périodes de temps utilisées sont contiguës. Enfin, le modèle minimise la somme S des carrés des charges pour un meilleur équilibrage des charges entre les périodes, plutôt que de minimiser $\underline{L}$ ou $\overline{L}$, comme cela se fait habituellement dans la littérature.

$$\text{Minimiser } S \tag{5.24}$$

$$\text{GCC}([X_1, X_2, \ldots, X_n], [M_1, \ldots, M_v]) \tag{5.25}$$

$$\sum_{j=1}^{v} M_j = n \ \wedge \ \sum_{i=1}^{n} X_i = \sum_{j=1}^{v} j \cdot M_j \tag{5.26}$$

$$\underline{M} = \min(M_1, \ldots, M_v) \tag{5.27}$$

$$\overline{M} = \max(M_1, \ldots, M_v) \tag{5.28}$$

$$\underline{m} \leq \underline{M} \ \wedge \ \overline{M} \leq \overline{m} \tag{5.29}$$

$$\forall (i,k) \in \mathcal{P}rec, X_i < X_k \tag{5.30}$$

$$\textsc{Partition}([\underbrace{X_1, \ldots, \ X_1}_{c(1) \text{ fois}}, \ldots, \underbrace{X_n, \ldots, \ X_n}_{c(n) \text{ fois}}], P, \underline{L}, \overline{L}, \underline{\overline{L}}, S) \tag{5.31}$$

$$\underline{L}_0 \leq \underline{L} \ \wedge \ \overline{L} \leq \overline{L}_0 \tag{5.32}$$

$$\forall i \in [1,n], 1 \leq X_i \leq v \wedge X_i \leq P \tag{5.33}$$

5.5.2 Un second modèle utilisant la contrainte Cumulative

Un deuxième modèle, de Mats Carlsson [40], utilise les contraintes (5.24) à (5.30) et la contrainte (5.33). Il code le reste du problème en utilisant une contrainte Cumulative [7; 10]. La contrainte Cumulative assure qu'un ensemble de tâches *taches* planifiées sur le temps n'excède jamais un plafond *limite* de ressources à tout instant de leur exécution. On associe une tâche par cours i avec une heure de début X_i, une durée d'une unité, une consommation de ressources $c(i)$, et une tâche par période j avec une heure de début j, une durée d'une unité et une variable d'utilisation des ressources H_j.

$$\textsc{Cumulative}(taches, limite) \tag{5.34}$$

$$\sum_{j=1}^{v} H_i + \sum_{i=1}^{n} c(i) = v \cdot limite \tag{5.35}$$

$$0 \leq H_j \leq limite - \underline{L}_0 \tag{5.36}$$

$$\underline{L}_0 \leq limite \leq \overline{L}_0 \tag{5.37}$$

Pour ajouter les quatre conjectures sélectionnées à ce modèle, nous créons les variables P, $\underline{L}$,

$\overline{L}$, S et les connectons au reste du modèle via les contraintes (5.38) à (5.44).

$$\forall i \in [1,n], \forall j \in [1,v], X_i = j \Leftrightarrow Y_{i,j} = 1 \tag{5.38}$$

$$\forall j \in [1,v], O_j = \sum_{i=1}^{n} c(i) \cdot Y_{i,j} \tag{5.39}$$

$$\underline{L} = \min(O_1, O_2, \ldots, O_v) \tag{5.40}$$

$$\overline{L} = \max(O_1, O_2, \ldots, O_v) \tag{5.41}$$

$$\overline{\underline{L}} = \overline{L} - \underline{L} \ \wedge \ S = \sum_{j=1}^{v} O_j^2 \tag{5.42}$$

$$\text{NVALUE}([X_1, X_2, \ldots, X_n], P) \tag{5.43}$$

$$\forall i \in [1,n], \forall j \in [1,v], Y_{i,j} \in \{0,1\} \tag{5.44}$$

5.5.3 Un troisième modèle utilisant des variables 0/1

Inspirés de ce que Bessiere et *al.* [15] fait, nous obtenons un troisième modèle en sélectionnant les contraintes (5.24) à (5.30) du premier modèle et les contraintes (5.38) à (5.44). Nous incluons les contraintes (5.38) à (5.44) même lorsque nous voulons utiliser ce modèle sans les contraintes trouvées par le Bound Seeker.

5.5.4 Cadre expérimental

Les trois modèles ont été implémentés dans SICStus Prolog et dans MiniZinc, avec et sans les quatre bornes sélectionnées par l'algorithme 2, donnant 12 implémentations. Les modèles écrits dans SICStus Prolog 4.6.0 ont été résolus à l'aide du solveur clp(FD), tandis que les modèles écrits en MiniZinc ont été résolus à l'aide du solveur Chuffed 0.12.1 avec l'option de recherche libre activée. Les expériences ont été réalisées sur un processeur Intel(R) Xeon(R) Silver 4110 de 2,1 GHz et 1 Go de mémoire.

Nous évaluons les modèles sur 31 instances réelles avec un nombre maximum de périodes de 8 à 10 et un nombre de cours de 42 à 66, tous disponibles sur CSPLib.

Pour le solveur clp(FD), nous utilisons deux stratégies de recherche. La première stratégie est utilisée pour le modèle sans les bornes apprises tandis que la seconde stratégie est utilisée pour le modèle avec les bornes. La première stratégie de recherche consiste à brancher sur l'une des variables $S, X_1, \ldots, X_n$ qui est la plus contrainte, c'est-à-dire la variable de plus petit domaine apparaissant dans le plus grand nombre de contraintes. Elle choisit une moitié du domaine de la variable sélectionnée et énumère les valeurs contenues dans cette moitié, dans l'ordre décroissant. La deuxième stratégie consiste à brancher sur les variables $\overline{\underline{L}}, S, P, \underline{L}, \overline{L}$ dans cet ordre et choisit une moitié du domaine de la variable sélectionnée, puis énumère les valeurs de cette moitié dans l'ordre croissant. Puis elle branche sur les variables $X_1, \ldots, X_n$ en utilisant la première stratégie.

La deuxième stratégie est plus agressive, car elle branche d'abord sur les variables qui codent la fonction objectif. Elle force la valeur objective à être petite, puis essaie de trouver une solution ayant cette valeur objective. Une telle stratégie ne peut fonctionner correctement qu'avec un filtrage puissant. C'est pourquoi nous n'utilisons la deuxième stratégie qu'avec des modèles qui incluent les bornes trouvées par le Bound Seeker. Pour les autres modèles, seule la première stratégie permet de trouver des solutions.

Pour le solveur Chuffed, nous utilisons une seule stratégie de recherche qui branche sur les variables $\overline{\underline{L}}, S, P, \underline{L}, \overline{L}$ dans cet ordre et énumère leurs valeurs en ordre croissant. Ensuite, elle branche sur les variables $X_1, \ldots, X_n$ dans cet ordre et énumère leurs valeurs par ordre décroissant.

Pour tous les modèles, nous fixons un délai de 13 minutes de recherche d'une solution optimale.

5.5.5 Résultats expérimentaux

Les figures 5.1 et 5.2 reportent la valeur objective S, c'est-à-dire la somme des carrés des charges par période, pour chacune des instances résolues par chacun des modèles. Chaque marque représente une instance du BACP. Une marque à gauche de la fonction identité indique une instance où les contraintes ajoutées par le Bound Seeker ont permis de trouver une meilleure solution. Une marque sur la fonction identité signifie que les deux modèles ont trouvé une solution équivalente. La couleur et la forme de la marque donnent le statut du solveur à la fin de l'expérience pour cette instance, c'est-à-dire si le solveur a prouvé l'optimalité, atteint le délai d'exécution ou qu'aucune solution n'a été trouvée. La couleur donne le statut du modèle sans bornes tandis que la forme de la marque donne le statut du modèle avec bornes. La même forme et le même code couleur sont utilisés dans les figures 5.3, 5.4, 5.5 et 5.6.

Bien que le modèle utilisant la contrainte Partition implémentée dans SICStus avec les bornes sélectionnées n'ait trouvé aucune solution minimisant S pour 20 instances, utiliser les bornes sélectionnées pour les autres solveurs et autres modèles est très avantageux. En effet, chacun des autres modèles avec les bornes sélectionnées prouve l'optimalité pour au moins 29 instances, alors que sans les bornes sélectionnées, le solveur atteint le délai d'exécution sans prouver l'optimalité. Pour prouver l'optimalité, le modèle avec la contrainte Partition de MiniZinc (resp. SICStus) passe un temps total de 10 (resp. 28) minutes sur 31 (resp. 11) instances. Le modèle avec la contrainte Cumulative de MiniZinc (resp. SICStus) passe un temps total de 38 (resp. 35) minutes sur 29 (resp. 29) instances. Le modèle 0/1 de MiniZinc (resp. SICStus) passe un temps total de 10 (resp. 40) minutes sur 30 (resp. 30) instances. Il y a donc une amélioration du temps de calcul d'un facteur de 8 pour le solveur clp(FD) SICStus et de 20 pour le solveur Chuffed.

L'avantage d'utiliser les bornes sélectionnées est également mis en évidence dans les résultats des modèles qui minimisent $\overline{\underline{L}}$, c'est-à-dire la plage entre la charge maximale et la charge

minimale de cours par période. Les résultats des expériences pour minimiser $\overline{\underline{L}}$ sont reportés dans les figures 5.3 et 5.4. La figure 5.3 compare les valeurs objectives de $\overline{\underline{L}}$ lors de l'utilisation du solveur clp(FD). On constate sur le graphique du coin supérieur droit que les bornes améliorent le modèle 0/1. En effet, la plupart des points se situent au-dessus de la fonction identité, ce qui signifie que de meilleures solutions sont trouvées dans le délai imparti en utilisant les bornes. Cependant, pour le modèle avec la contrainte Partition, les résultats ne sont pas aussi concluants. Pour prouver l'optimalité, le modèle avec la contrainte Partition de SICStus passe un temps total de 22 minutes sur 12 instances. Le modèle avec la contrainte Cumulative de SICStus passe un temps total de 24 minutes sur 30 instances. Le modèle 0/1 de SICStus passe un temps total de 19 minutes sur 30 instances. Enfin, les nouvelles bornes améliorent également le modèle avec la contrainte Cumulative dans le graphique de droite. Le solveur Chuffed atteint l'optimalité pour la fonction objectif $\overline{\underline{L}}$ sur la plupart des instances en utilisant tous les modèles avec ou sans bornes. Par conséquent, la figure 5.4 compare le temps de calcul plutôt que les valeurs objectives. Dans la figure 5.4, le solveur Chuffed a atteint l'optimalité avec le modèle 0/1 pour 29 instances avec ou sans bornes. Il est plus rapide avec l'utilisation des bornes sur 28 instances. Le solveur Chuffed atteint également l'optimalité pour le modèle avec la contrainte Partition sur 31 instances avec ou sans bornes. Il est plus rapide avec l'utilisation des bornes sur 19 instances. Le solveur Chuffed atteint également l'optimalité pour le modèle avec la contrainte Cumulative sur 22 instances avec ou sans bornes. Il est plus rapide avec l'utilisation des bornes sur 14 instances. Nous constatons que, la plupart du temps, l'utilisation des bornes apprises améliore le temps de calcul d'un facteur de 7 pour le solveur clp(FD) SICStus et d'un facteur de 900 pour le solveur Chuffed.

Pour la fonction objectif $\overline{L}$ à minimiser, c'est-à-dire la charge maximale de cours par période, les deux solveurs atteignent l'optimum dans la plupart des cas, qu'ils utilisent ou pas les bornes. Les figures 5.5 et 5.6 comparent donc les temps de calcul. Dans la figure 5.5, on constate que le solveur Chuffed a atteint l'optimalité pour le modèle 0/1 sur 31 instances avec ou sans bornes. Il est plus rapide avec l'utilisation des bornes sur 16 instances. Le solveur Chuffed a également atteint l'optimalité pour le modèle avec la contrainte Partition sur 31 instances avec ou sans bornes. Il est plus rapide avec l'utilisation des bornes sur 15 instances. Enfin, le solveur Chuffed a également atteint l'optimalité pour le modèle avec la contrainte Cumulative sur 30 instances avec ou sans bornes. Il est plus rapide avec l'utilisation des bornes sur 16 instances. Dans la figure 5.6, on constate que le solveur clp(FD) a atteint l'optimalité pour le modèle 0/1 sur 26 instances avec ou sans bornes. Il est plus rapide avec l'utilisation des bornes sur 10 instances. Le solveur clp(FD) a également atteint l'optimalité pour le modèle avec la contrainte Partition sur 10 instances avec ou sans bornes. Il est plus rapide avec l'utilisation des bornes sur 5 instances. Enfin, le solveur clp(FD) a également atteint l'optimalité pour le modèle avec la contrainte Cumulative sur 25 instances avec ou sans bornes. Il est plus rapide avec l'utilisation des bornes sur 9 instances.

Nous constatons que l'ajout de bornes n'améliore pas le temps de calcul des modèles qui minimisent la fonction objectif $\overline{L}$ dans la plupart des cas. Cela n'est pas surprenant puisque l'ajout des bornes de la variable S n'aide pas à résoudre le problème puisque S n'est pas minimisé dans cette version du problème. La propagation de ces bornes ne peut que ralentir le solveur. De plus, la fonction objectif $\overline{L}$ est facile à minimiser, car le filtrage de la borne supérieure de la fonction objectif filtre directement la borne supérieure des variables encodant la taille des partitions. C'est pourquoi la fonction objectif $\overline{L}$ est largement utilisée dans la littérature plutôt que les fonctions objectifs $\underline{\overline{L}}$ et S qui sont plus difficiles à minimiser. Mais on sait aussi que la fonction objectif S donne des solutions bien plus équilibrées que $\underline{\overline{L}}$ qui elle-même donne des solutions plus équilibrées que $\overline{L}$. Les expériences montrent que les modèles améliorés avec les nouvelles bornes fournissent de meilleurs résultats de calcul avec des solutions plus équilibrées dans le cadre d'utilisation des deux fonctions objectifs S et $\underline{\overline{L}}$. De plus, nous remarquons que lorsque les trois modèles utilisent les bornes pour minimiser $\overline{L}$, ils sont meilleurs sur les instances pour lesquelles les deux solveurs atteignent le délai d'exécution sans prouver l'optimalité. En fait, pour 186 exécutions (3 modèles $\times$ 2 solveurs $\times$ 31 instances), tous les modèles sans bornes prouvent l'optimalité 126 fois alors que ces mêmes modèles prouvent l'optimalité 164 fois, c'est-à-dire 38 fois de plus, lorsqu'ils utilisent les bornes sélectionnées.

Ainsi, le filtrage supplémentaire inféré par le Bound Seeker aide à résoudre le BACP en trouvant de meilleures solutions équilibrées. Le filtrage supplémentaire permet d'utiliser une stratégie de recherche plus agressive. Si la première stratégie était utilisée pour tous les modèles, les solutions optimales ne seraient pas obtenues dans la plupart des cas. Si la deuxième stratégie était utilisée pour tous les modèles, seuls les modèles améliorés par les contraintes générées et sélectionnées du Bound Seeker renverraient une solution. Le Bound Seeker a donc fait la différence en trouvant les solutions optimales des instances.

5.6 Discussion

Nous avons présenté comment exploiter les conjectures générées par le Bound Seeker pour obtenir de meilleurs algorithmes de filtrage sur des problèmes où nous devons contraindre plusieurs caractéristiques d'un même objet combinatoire. Cela se fait en synthétisant des contraintes redondantes, le principal avantage étant que ces contraintes peuvent être facilement ajoutées dans différents solveurs ou dans un langage de modélisation tel que MiniZinc. La méthode répond à deux problèmes majeurs, à savoir l'impossibilité de prouver des centaines de conjectures produites par le Bound Seeker dans un laps de temps raisonnable, et la nécessité d'identifier les conjectures qui auraient le plus grand impact sur le filtrage. Pour résoudre ces deux problèmes, nous avons proposé un algorithme qui sélectionne quelques conjectures pertinentes qui fournissent le plus de filtrage. Nous prouvons les conjectures sélectionnées et les ajoutons au modèle pour obtenir un filtrage supplémentaire. Nous avons introduit la contrainte Partition et trouvé quatre conjectures non triviales qui complètent la décomposition de cette contrainte.

Les expériences montrent que les contraintes générées automatiquement améliorent le temps de résolution. En pratique, l'amélioration ne se limite pas à un seul modèle et fonctionne pour tous les solveurs testés. À l'avenir, davantage de contraintes basées sur des objets combinatoires pourraient bénéficier du Bound Seeker. En effet, la contrainte Cycle de Pesant et Soriano [61] et la contrainte Tree de Beldiceanu et *al.* [12] pourraient bénéficier du système d'acquisition de propagateurs que constituent le Bound Seeker et l'algorithme 2 en fournissant des bornes obtenues dans le cadre des partitions d'ensembles. De même, la contrainte Stretch de Pesant [60] bénéficierait de ce système d'acquisition de propagateurs. Cependant, un point que nous n'avons pas abordé dans ce chapitre est le niveau de cohérence de l'ensemble des contraintes de bornes sélectionnées. En effet, ce qui serait intéressant est de vérifier si l'ensemble des bornes sélectionnées permet d'obtenir la cohérence de borne juste pour le modèle de contraintes qui génère les instances de l'objet combinatoire. Ainsi, s'il n'y a pas cohérence de borne, l'étude des valeurs non filtrées des domaines des variables ou des caractéristiques impliquées pourrait donner des informations sur les bornes manquantes qui permettraient d'obtenir une cohérence de borne. Et dès lors, le Bound Seeker pourrait intervenir à nouveau pour la recherche de telles bornes.

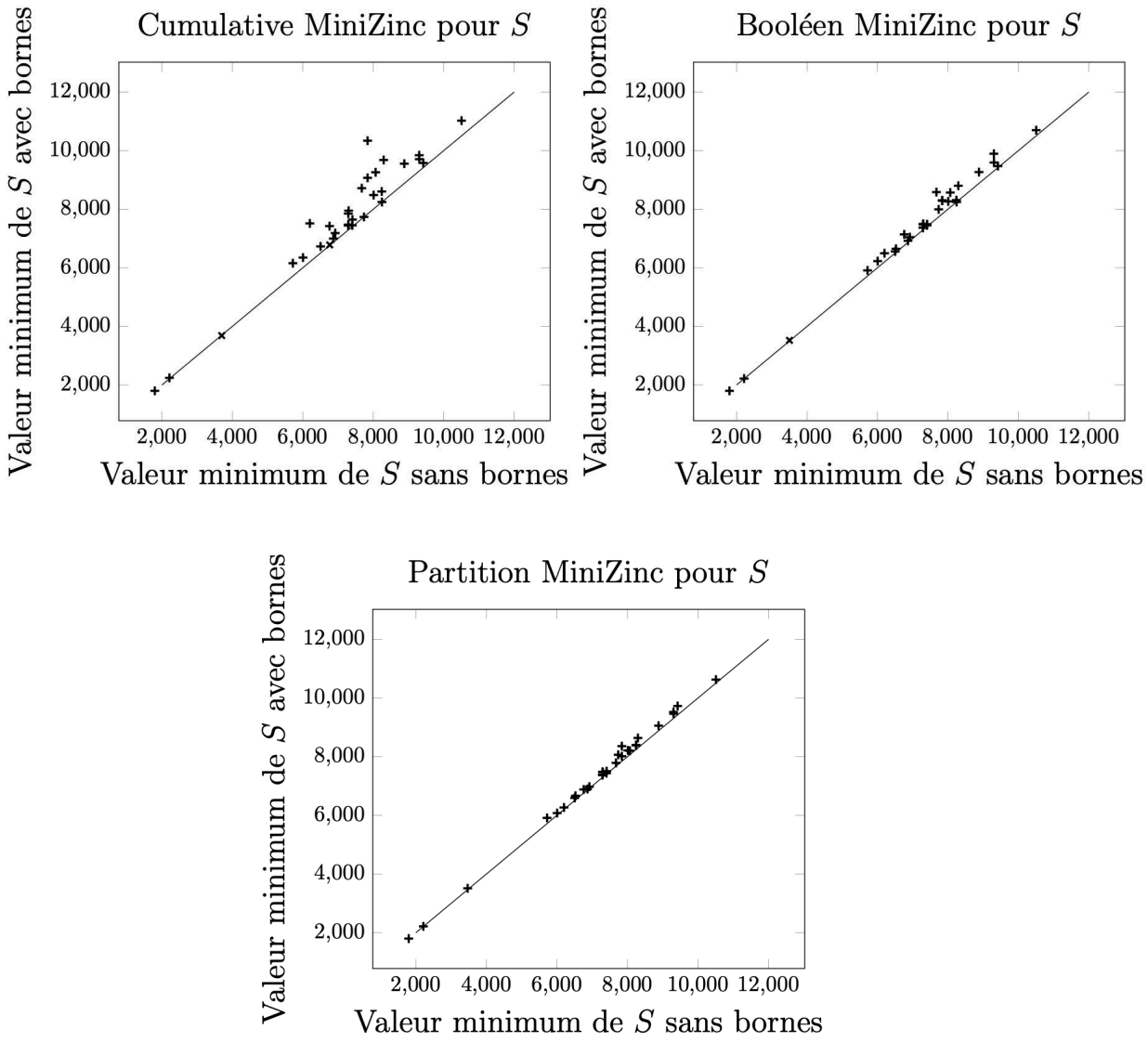


FIGURE 5.1 – Comparaison de la valeur objectif S lorsque le modèle 0/1, le modèle avec la contrainte PARTITION et le modèle avec la contrainte CUMULATIVE utilisent les bornes sélectionnées et lorsqu'elles ne les utilisent pas sur le solveur Chuffed MiniZinc. Chaque marque représente une instance du BACP. Une marque donne trois informations sur les résultats selon sa forme, sa couleur et sa position. Les couleurs bleu, noir et rouge indiquent que le modèle sans bornes a respectivement prouvé l'optimalité, n'a pas prouvé l'optimalité et n'a trouvé aucune solution pour l'instance correspondante. De même, les formes $+$, $\times$ et $\circ$ indiquent que le modèle avec bornes a respectivement prouvé l'optimalité, n'a pas prouvé l'optimalité et n'a trouvé aucune solution. L'ordonnée (resp. abscisse) est la valeur de S de la meilleure solution trouvée par le modèle sans (resp. avec) les conjectures sélectionnées. Ainsi, une marque qui ne figure pas sur la fonction identité signifie qu'un modèle est meilleur que l'autre. Le modèle qui est le meilleur est celui avec la plus petite valeur de S. Lorsqu'un modèle n'a pas trouvé de solution, la valeur correspondante de S est définie sur la limite de l'axe. Une forme ou une couleur n'est pas dans le graphique lorsque le modèle ne donne pas un tel résultat. Les graphiques montrent qu'aucun des modèles ne prouve l'optimalité sans utiliser les bornes.

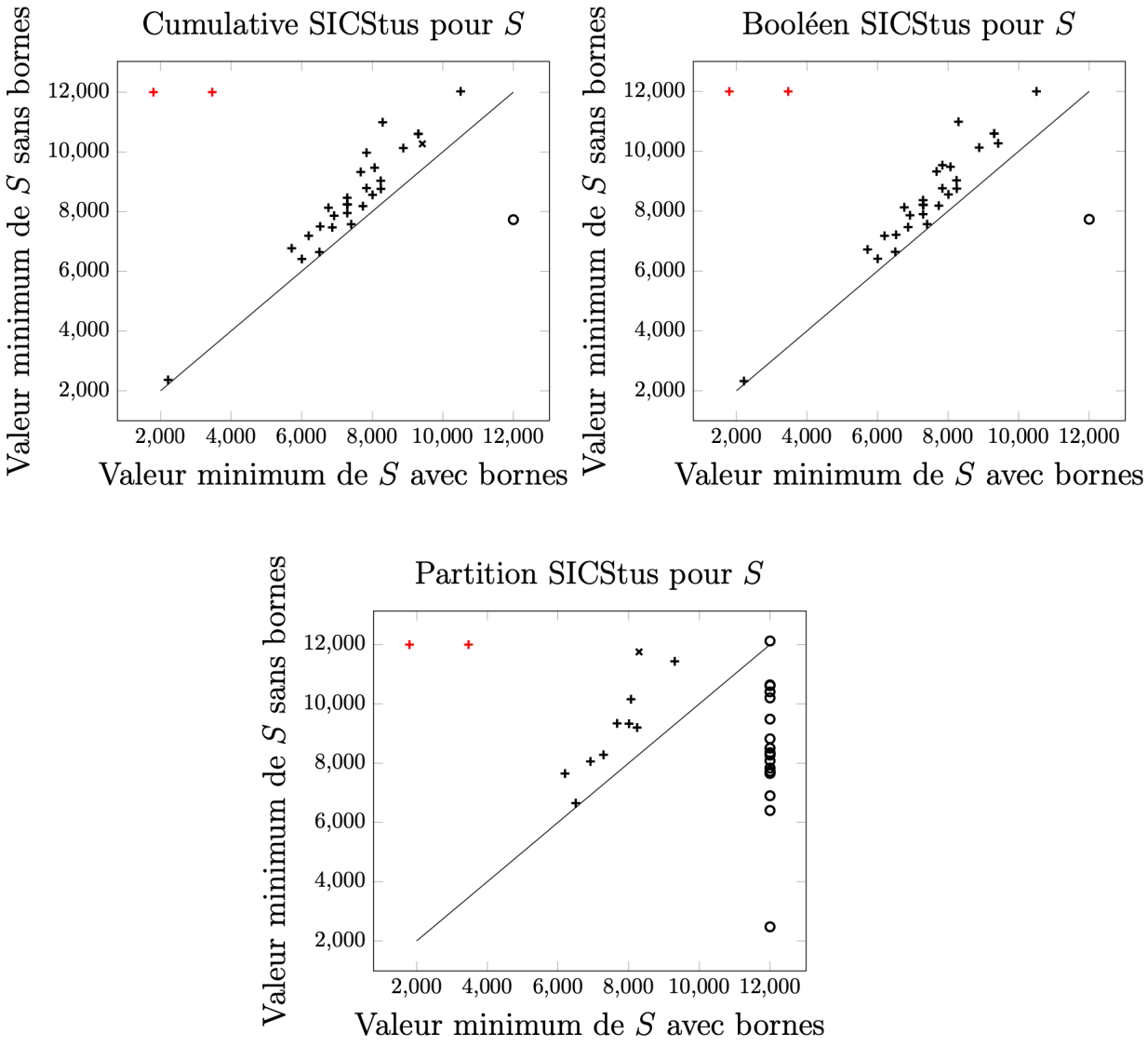


FIGURE 5.2 – Comparaison de la valeur objectif S lorsque le modèle 0/1, le modèle avec la contrainte PARTITION et le modèle avec la contrainte CUMULATIVE utilisent les bornes sélectionnées et lorsqu'elles ne les utilisent pas sur le solveur clp(FD) SICStus. Chaque marque représente une instance du BACP. Une marque donne trois informations sur les résultats selon sa forme, sa couleur et sa position. Les couleurs bleu, noir et rouge indiquent que le modèle sans bornes a respectivement prouvé l'optimalité, n'a pas prouvé l'optimalité et n'a trouvé aucune solution pour l'instance correspondante. De même, les formes $+$, $\times$ et $\circ$ indiquent que le modèle avec bornes a respectivement prouvé l'optimalité, n'a pas prouvé l'optimalité et n'a trouvé aucune solution. L'ordonnée (resp. abscisse) est la valeur de S de la meilleure solution trouvée par le modèle sans (resp. avec) les conjectures sélectionnées. Ainsi, une marque qui ne figure pas sur la fonction identité signifie qu'un modèle est meilleur que l'autre. Le modèle qui est le meilleur est celui avec la plus petite valeur de S. Lorsqu'un modèle n'a pas trouvé de solution, la valeur correspondante de S est définie sur la limite de l'axe. Une forme ou une couleur n'est pas dans le graphique lorsque le modèle ne donne pas un tel résultat. Les graphiques montrent qu'aucun des modèles ne prouve l'optimalité sans utiliser les bornes.

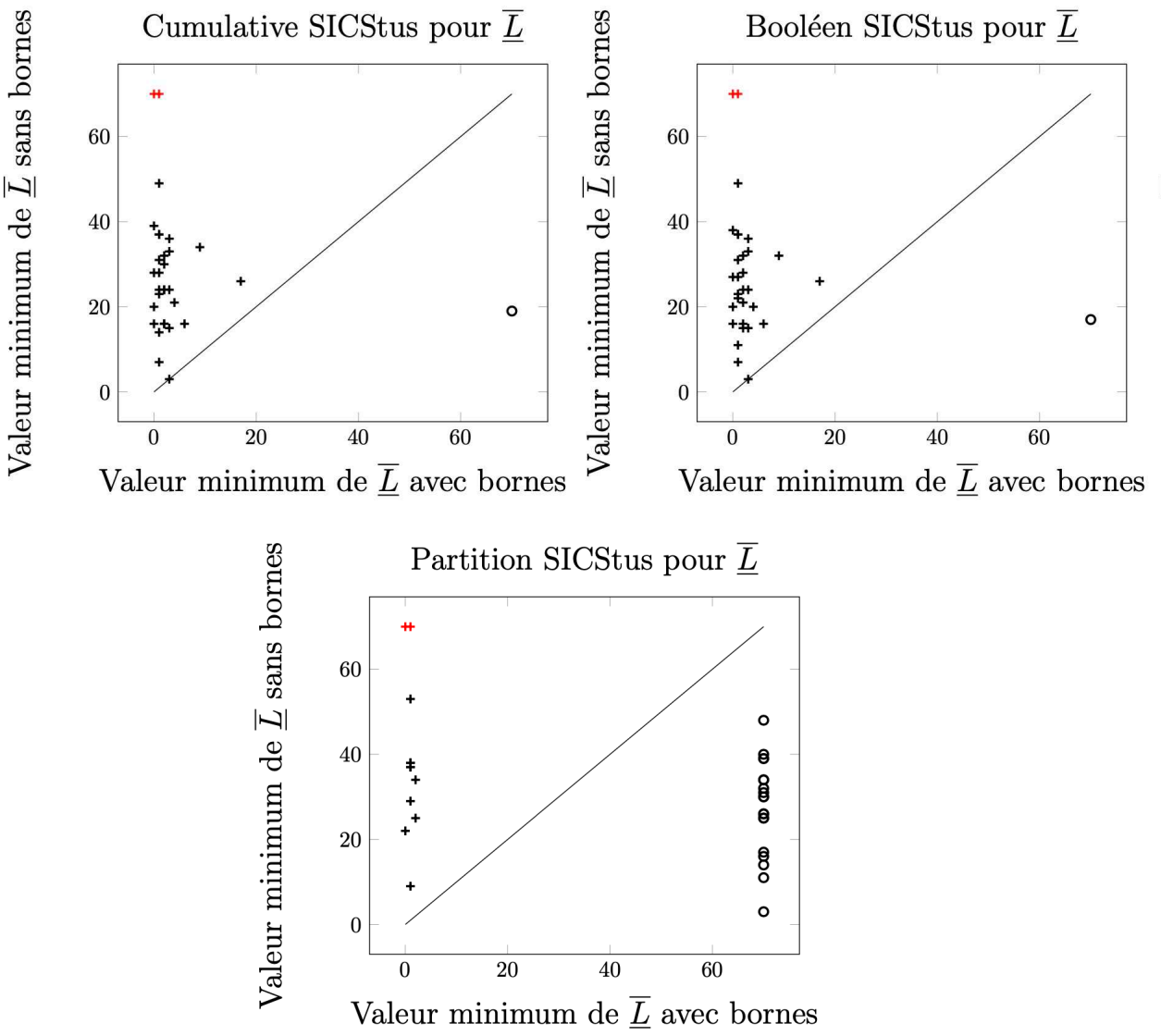


FIGURE 5.3 – Comparaison de la valeur objectif $\overline{\underline{L}}$ obtenue par le solveur clp(FD) SICStus lorsque le modèle 0/1, le modèle avec la contrainte PARTITION et le modèle avec la contrainte CUMULATIVE utilisent les bornes sélectionnées et lorsqu'elles ne les utilisent pas. Une marque donne trois informations sur les résultats selon sa forme, sa couleur et sa position. Les couleurs bleu, noir et rouge indiquent que le modèle sans bornes a respectivement prouvé l'optimalité, n'a pas prouvé l'optimalité et n'a pas trouvé de solution pour l'instance correspondante. Les formes $+$, $\times$ et $\circ$ indiquent que le modèle avec bornes a respectivement prouvé l'optimalité, n'a pas prouvé l'optimalité et n'a pas trouvé de solution. L'ordonnée (resp. l'abscisse) est la valeur de $\overline{\underline{L}}$ de la meilleure solution trouvée par le modèle sans (resp. avec) les conjectures sélectionnées. Ainsi, une marque qui ne figure pas sur la fonction identitaire signifie qu'un modèle est meilleur que l'autre. Le modèle qui est le meilleur est celui avec la plus petite valeur de $\overline{\underline{L}}$. Lorsqu'un modèle n'a pas trouvé de solution, la valeur correspondante de $\overline{\underline{L}}$ est définie sur la limite de l'axe. Lorsqu'une forme ou une couleur ne figure pas dans le graphique, cela signifie que le modèle n'a pas donné un tel résultat. Les graphiques montrent qu'aucun des modèles ne prouve l'optimalité sans utiliser les bornes.

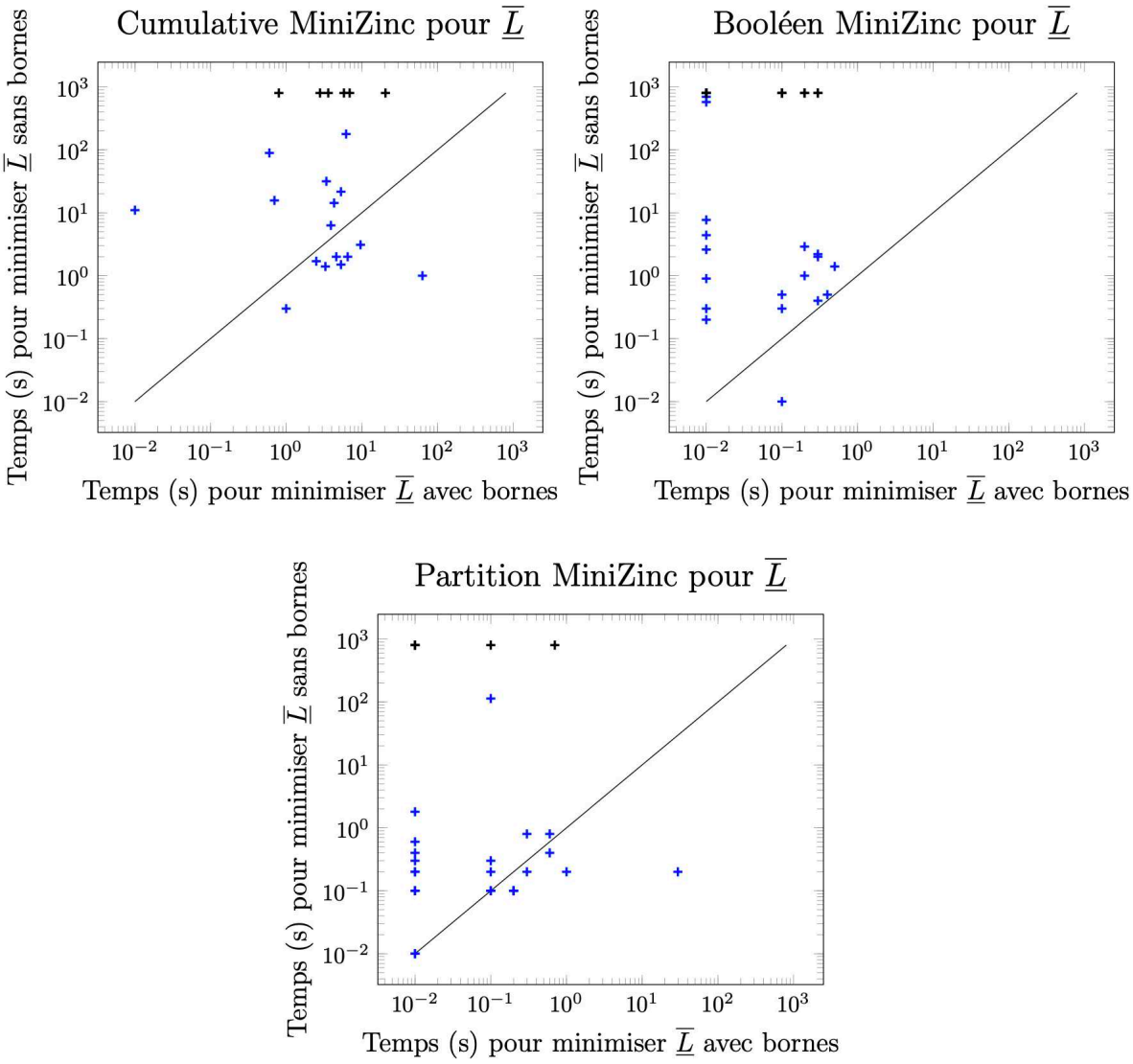


FIGURE 5.4 – Comparaison du temps de résolution requis par le solveur Chuffed MiniZinc pour minimiser la valeur objectif $\underline{\overline{L}}$ lorsque le modèle 0/1, le modèle avec la contrainte PARTITION et le modèle avec la contrainte CUMULATIVE utilisent les bornes sélectionnées et lorsqu'elles ne les utilisent pas. Une marque fournit trois informations sur les résultats selon sa forme, sa couleur et sa position. Les couleurs bleu, noir et rouge indiquent que le modèle sans bornes a respectivement prouvé l'optimalité, n'a pas prouvé l'optimalité et n'a pas trouvé de solution pour l'instance correspondante. Les formes $+$, $\times$ et $\circ$ indiquent que le modèle avec bornes a respectivement prouvé l'optimalité, n'a pas prouvé l'optimalité et n'a pas trouvé de solution. L'ordonnée (resp. l'abscisse) est la consommation de temps du modèle, le modèle sans (resp. avec) les conjectures sélectionnées pour trouver la plus petite valeur de $\underline{\overline{L}}$. Ainsi, une marque qui ne figure pas sur la fonction identité signifie qu'un modèle est meilleur que l'autre. Le modèle qui est le meilleur est le plus rapide. Lorsqu'une forme ou une couleur n'est pas présente dans le graphique, cela signifie que le modèle n'a pas donné ce résultat. Les graphiques montrent que l'utilisation des bornes améliore les modèles.

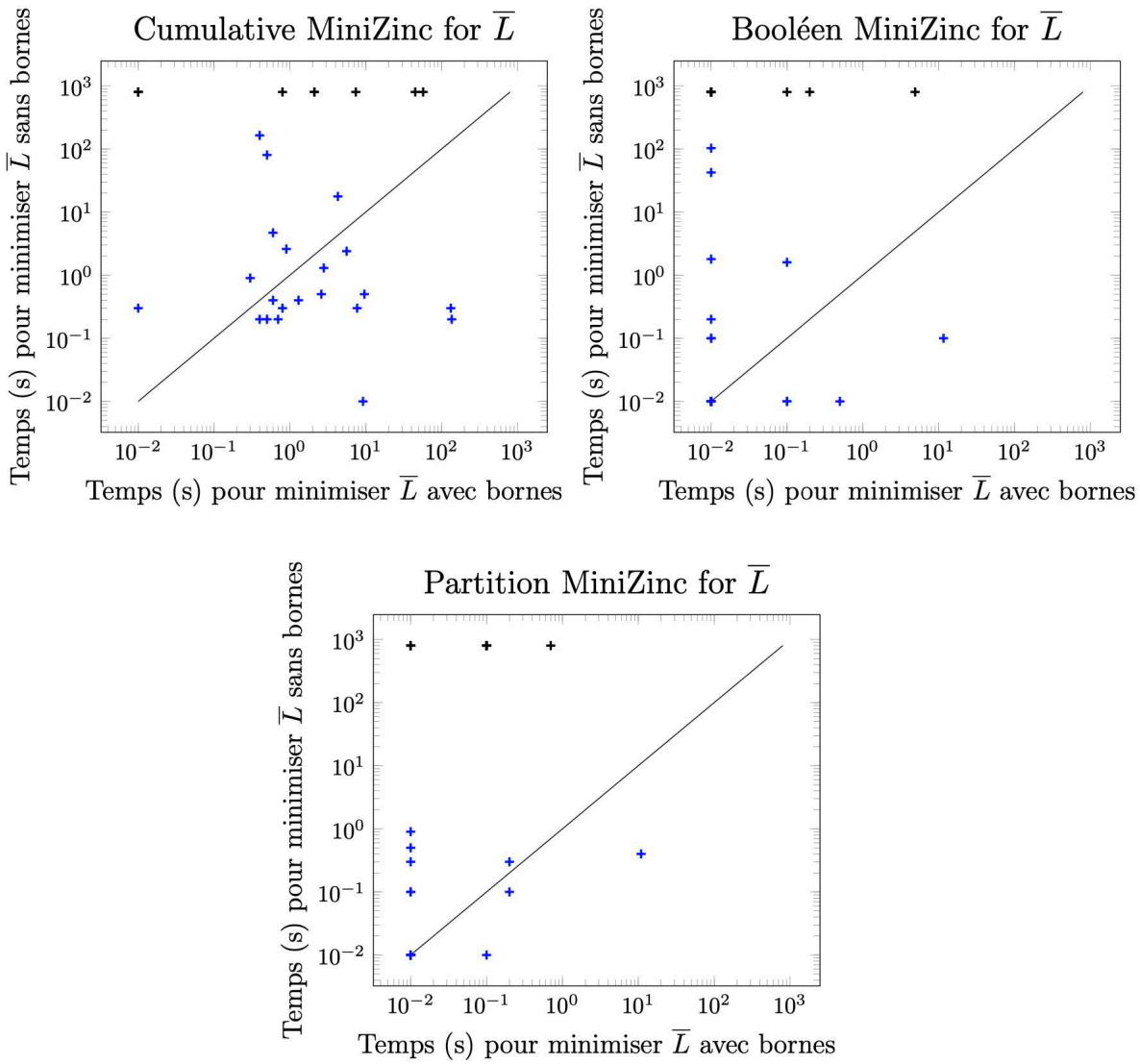


FIGURE 5.5 – Comparaison de la consommation de temps pour minimiser la valeur objectif $\overline{L}$ lorsque le modèle 0/1, le modèle avec la contrainte PARTITION et le modèle avec la contrainte CUMULATIVE utilisent les bornes sélectionnées et lorsqu'elles ne les utilisent pas avec le solveur Chuffed MiniZinc. Une marque donne trois informations sur les résultats selon sa forme, sa couleur et sa position. Les couleurs bleu, noir et rouge indiquent que le modèle sans bornes a respectivement prouvé l'optimalité, n'a pas prouvé l'optimalité et n'a pas trouvé de solution pour l'instance correspondante. Les formes $+$, $\times$ et $\circ$ indiquent que le modèle avec bornes a respectivement prouvé l'optimalité, n'a pas prouvé l'optimalité et n'a pas trouvé de solution. L'ordonnée (resp. abscisse) est la consommation de temps du modèle sans (resp. avec) les bornes sélectionnées pour trouver la plus petite valeur de $\overline{L}$. Ainsi, une marque qui ne figure pas sur la fonction identité signifie qu'un modèle est meilleur que l'autre. Le modèle qui est le meilleur est le plus rapide. Lorsqu'une forme ou une couleur ne figure pas dans le graphique, cela signifie que le modèle n'a pas donné un tel résultat. Les graphiques montrent que l'utilisation de bornes n'améliore pas les modèles.

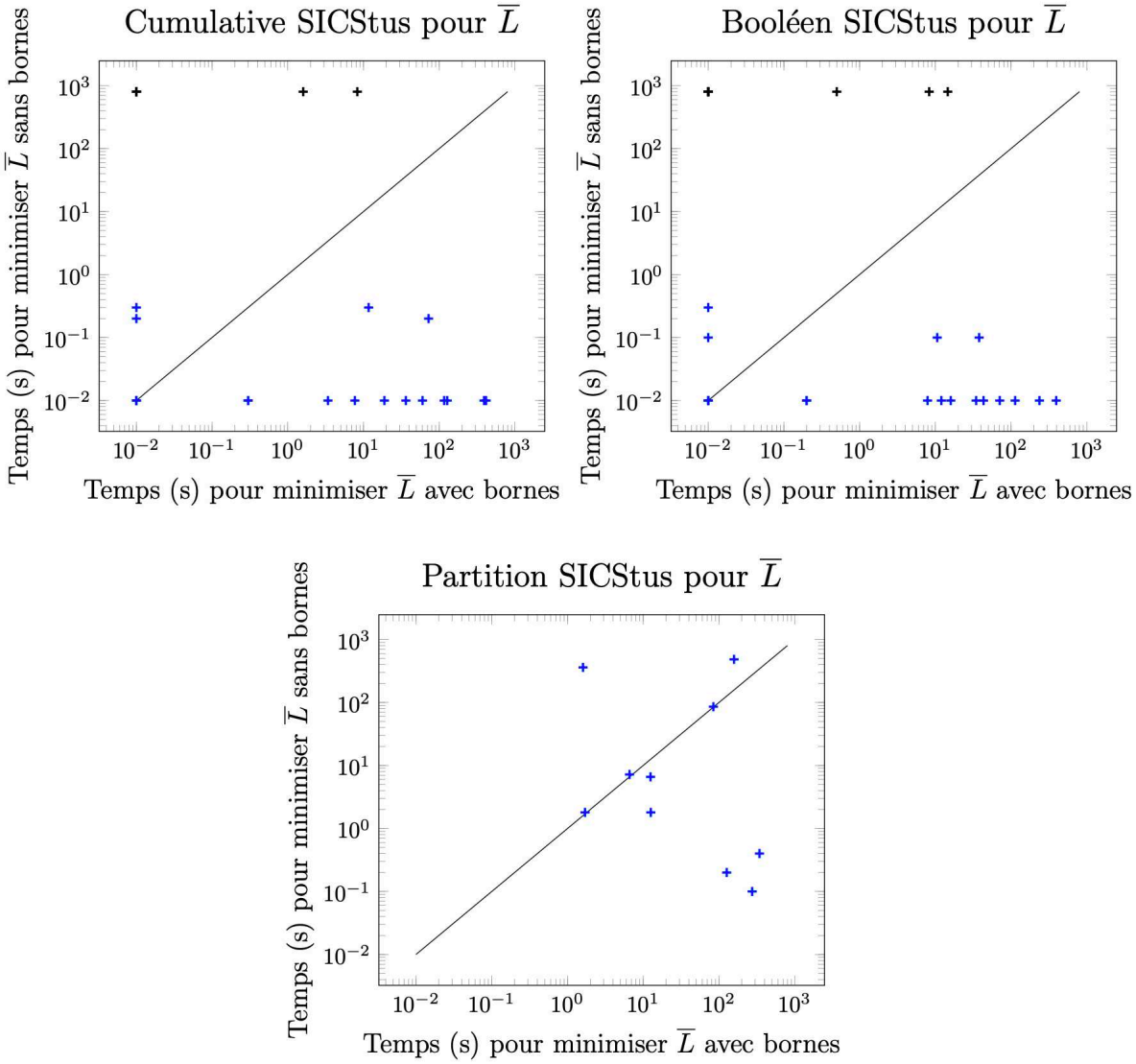


FIGURE 5.6 – Comparaison de la consommation de temps pour minimiser la valeur objectif $\overline{L}$ lorsque le modèle 0/1, le modèle avec la contrainte PARTITION et le modèle avec la contrainte CUMULATIVE utilisent les bornes sélectionnées et lorsqu'elles ne les utilisent pas avec le solveur clp(FD) SICStus. Une marque donne trois informations sur les résultats selon sa forme, sa couleur et sa position. Les couleurs bleu, noir et rouge indiquent que le modèle sans bornes a respectivement prouvé l'optimalité, n'a pas prouvé l'optimalité et n'a pas trouvé de solution pour l'instance correspondante. Les formes $+$, $\times$ et $\circ$ indiquent que le modèle avec bornes a respectivement prouvé l'optimalité, n'a pas prouvé l'optimalité et n'a pas trouvé de solution. L'ordonnée (resp. abscisse) est la consommation de temps du modèle sans (resp. avec) les bornes sélectionnées pour trouver la plus petite valeur de $\overline{L}$. Ainsi, une marque qui ne figure pas sur la fonction identité signifie qu'un modèle est meilleur que l'autre. Le modèle qui est le meilleur est le plus rapide. Lorsqu'une forme ou une couleur ne figure pas dans le graphique, cela signifie que le modèle n'a pas donné un tel résultat. Les graphiques montrent que l'utilisation de bornes n'améliore pas les modèles.

Conclusion

Dans ces travaux, nous avons commencé par présenter les enjeux de l'optimisation combinatoire dont l'une des spécificités est l'amélioration de l'efficacité des solveurs de contraintes pour des problèmes combinatoires. Puis, quelques limites des méthodes de résolution existantes dans la littérature ont été mises en évidence. Ensuite, pour répondre à ces limites, une approche effective que proposent nos travaux a été présentée. Elle consiste en l'apprentissage de cartes d'invariants d'objets combinatoires pour la synthèse automatique d'algorithmes de filtrage exploitables par les solveurs de contraintes. Cette approche a été réalisée lors du déroulement de ces travaux en trois étapes que sont : l'acquisition automatique de cartes de conjectures sur les bornes précises des objets combinatoires, l'élaboration des preuves des théorèmes découverts et la génération automatique des algorithmes de filtrage. Parmi les contributions qu'ont apportées ces travaux, certaines ont fait l'objet de publications dans les conférences internationales de programmation par contraintes (CP 2022) à travers l'article de Beldiceanu et *al.* [11], de recherche opérationnelle (CPAIOR 2023) à travers l'article de Gindullin et *al.* [36], d'intelligence artificielle (AAAI 2024) à travers l'article de Gindullin et *al.* [35]. Plus précisément, les contributions de ces travaux sont :

1. L'introduction du concept de *carte de conjectures* qui révèle et regroupe des relations entre bornes de caractéristiques d'un même objet combinatoire. Ainsi, la carte met en évidence une cohérence mutuelle entre des bornes acquises indépendamment tout en révélant des propriétés sur la structure de l'objet combinatoire.
2. Le Bound Seeker qui est un système d'acquisition automatique des cartes de conjectures utilisant la programmation par contraintes pour découvrir des conjectures et démontre ainsi l'utilité de la programmation par contraintes dans le domaine de recherche automatique des conjectures.
3. La mise en évidence de la cohérence des cartes en révélant des liens existants entre cartes d'un même objet combinatoire et entre objets combinatoires distincts.
4. La mise en évidence de nouvelles pistes d'automatisation de preuves de conjectures en utilisant la projection de bornes, l'inversion de bornes et l'encodage d'objets combinatoires.
5. La découverte de nouveaux théorèmes sur les graphes orientés, les arbres enracinés et

les partitions d'ensembles. Quatre de ces théorèmes se sont immédiatement avérés utiles en optimisation combinatoire.

6. L'aide à la conception d'algorithmes de filtrage. Cela se fait par la génération semi-automatique de propagateurs dont l'efficacité a été révélée dans le cas de la recherche des solutions les mieux équilibrées pour les problèmes de partitionnement.

Pour terminer cette thèse s'ouvre à d'autres voies de recherche, comme l'automatisation des preuves ou des réfutations des conjectures découvertes, l'amélioration des capacités du Bound Seeker dans la recherche des conjectures de bornes impliquant plus de caractéristiques bornantes, l'étude du niveau de cohérence des propagateurs générés semi-automatiquement en vue de leur amélioration, ou encore leur implémentation effective dans les solveurs de contraintes.

Bibliographie

[1] Andreas Abel, Thierry Coquand, and Ulf Norell. Connecting a logical framework to a first-order logic prover. In *Proceedings of the 5th International Conference on Frontiers of Combining Systems*, FroCoS'05, page 285–301, 2005.

[2] Tobias Achterberg, Robert E. Bixby, Zonghao Gu, Edward Rothberg, and Dieter Weninger. Presolve reductions in mixed integer programming. *INFORMS Journal on Computing*, 32(2) :473–506, 2020.

[3] Mustapha Aouchiche, Gilles Caporossi, and Pierre Hansen. Open problems on graph eigenvalues studied with autographix. *EURO J. Comput. Optim.*, 1(1-2) :181–199, 2013.

[4] Mustapha Aouchiche, Gilles Caporossi, Pierre Hansen, and M. Laffay. Autographix : a survey. *Electron. Notes Discret. Math.*, 22 :515–520, 2005.

[5] Ekaterina Arafailova, Nicolas Beldiceanu, and Helmut Simonis. Deriving generic bounds for time-series constraints based on regular expressions characteristics. *Constraints An Int. J.*, 23(1) :44–86, 2018.

[6] Ignacio Araya, Gilles Trombettoni, and Bertrand Neveu. Exploiting monotonicity in interval constraint propagation. *Proceedings of the AAAI Conference on Artificial Intelligence*, 24(1) :9–14, Jul. 2010.

[7] Philippe Baptiste and Claude Le Pape. Constraint propagation and decomposition techniques for highly disjunctive and highly cumulative project scheduling problems. In Gert Smolka, editor, *Principles and Practice of Constraint Programming*, volume 1330 of *Lecture Notes in Computer Science*, pages 375–389, 1997.

[8] Philippe Baptiste, Claude Le Pape, and Wim Nuijten. *Constraint-Based Scheduling : Applying Constraint Programming to Scheduling Problems*. 2012.

[9] Nicolas Beldiceanu, Mats Carlsson, and Thierry Petit. Deriving filtering algorithms from constraint checkers. In *Principles and Practice of Constraint Programming – CP 2004*, CP'04, 2004.

[10] Nicolas Beldiceanu, Mats Carlsson, and Jean-Xavier Rampon. Global Constraint Catalog, 2nd Edition (revision a). Technical Report T2012-03, Swedish Institute of Computer Science, 2012.

[11] Nicolas Beldiceanu, Jovial Cheukam-Ngouonou, Rémi Douence, Ramiz Gindullin, and Claude-Guy Quimper. Acquiring maps of interrelated conjectures on sharp bounds. In *International Conference on Principles and Practice of Constraint Programming*, volume 235 of *LIPIcs*, pages 6 :1–6 :18, 2022.

[12] Nicolas Beldiceanu, Pierre Flener, and Xavier Lorca. The tree constraint. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems : Second International Conference, CPAIOR 2005, Proceedings*, pages 64–78, 2005.

[13] Nicolas Beldiceanu and Helmut Simonis. A model seeker : Extracting global constraint models from positive examples. In *Principles and Practice of Constraint Programming - 18th International Conference, CP 2012*, volume 7514 of *Lecture Notes in Computer Science*, pages 141–157, 2012.

[14] Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial optimization : A methodological tour d'horizon. *Eur. J. Oper. Res.*, 290(2) :405–421, 2021.

[15] Christian Bessiere, Emmanuel Hebrard, George Katsirelos, Zeynep Kiziltan, Émilie Picard-Cantin, Claude-Guy Quimper, and Toby Walsh. The balance constraint family. In *Principles and Practice of Constraint Programming*, volume 8656 of *Lecture Notes in Computer Science*, pages 174–189, 2014.

[16] Christian Bessiere, Frédéric Koriche, Nadjib Lazaar, and Barry O'Sullivan. Constraint acquisition. *Artif. Intell.*, 244 :315–342, 2017.

[17] Armin Biere, Matti Järvisalo, and Benjamin Kiesl. *Handbook of Satisfiability*, chapter 9 : Preprocessing in SAT Solving, pages 391–435. 2021.

[18] Sandrine Blazy, Pierre Castéran, and Hugo Herbelin. L'Assistant de Preuve Coq Table des matières. *Techniques de l'Ingenieur*, August 2017.

[19] Alexander Bradford, J. Kain Day, Laura Hutchinson, Bryan Kaperick, Craig E. Larson, Matthew Mills, David Muncy, and Nico Van Cleemput. Automated conjecturing II : chomp and reasoned game play. *J. Artif. Intell. Res.*, 68 :447–461, 2020.

[20] Jure Brence, Ljupčo Todorovski, and Sašo Džeroski. Probabilistic grammars for equation discovery. *Knowledge-Based Systems*, 224 :107077, 2021.

[21] Andrei Z Broder. The r-stirling numbers. *Discrete Mathematics*, 49(3) :241–259, 1984.

[22] J. Paul Brooks, D. J. Edwards, Craig E. Larson, and Nicolas Van Cleemput. Conjecturing-based computational discovery of patterns in data. *CoRR*, abs/2011.11576, 2020.

[23] Céline Brouard, Simon de Givry, and Thomas Schiex. Pushing data into CP models using graphical model learning and solving. In *Principles and Practice of Constraint Programming*, pages 811–827, 2020.

[24] Neal Bushaw, Craig E. Larson, and Nicolas Van Cleemput. Automated conjecturing VII : the graph brain project & big mathematics. *CoRR*, abs/1801.01814, 2018.

[25] Gilles Caporossi. Variable neighborhood search for extremal vertices : The autographix-iii system. *Comput. Oper. Res.*, 78 :431–438, 2017.

[26] Gilles Caporossi and Pierre Hansen. Variable neighborhood search for extremal graphs : The autographix system. *Discret. Math.*, 212(1-2) :29–44, 2000.

[27] Carlos Castro, Broderick Crawford, and Eric Monfroy. A genetic local search algorithm for the multiple optimisation of the balanced academic curriculum problem. In *Cutting-Edge Research Topics on Multiple Criteria Decision Making*, pages 824–832, Berlin, Heidelberg, 2009.

[28] William G. La Cava, Patryk Orzechowski, Bogdan Burlacu, Fabrício Olivetti de França, Marco Virgolin, Ying Jin, Michael Kommenda, and Jason H. Moore. Contemporary symbolic regression methods and their relative performance. *CoRR*, abs/2107.14351, 2021.

[29] John William Charnley, Simon Colton, and Ian Miguel. Automatic generation of implied constraints. In *European Conference on Artificial Intelligence*, pages 73–77, 2006.

[30] Geoffrey Chu, Peter Stuckey, Maria Garcia de la Banda, and Christopher Mears. Symmetries and lazy clause generation. *IJCAI International Joint Conference on Artificial Intelligence*, pages 516–521, 01 2011.

[31] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, STOC '71, page 151–158, 1971.

[32] Luca Di Gaspero and Andrea Schaerf. Hybrid local search techniques for the generalized balanced academic curriculum problem. In *Hybrid Metaheuristics*, pages 146–157, 2008.

[33] Siemion Fajtlowicz. On conjectures of Graffiti. *Discret. Math.*, 72(1-3) :113–118, 1988.

[34] Ian P. Gent, Christopher Jefferson, Ian Miguel, and Peter Nightingale. Generating special-purpose stateless propagators for arbitrary constraints. In David Cohen, editor, *Principles and Practice of Constraint Programming - CP 2010*, volume 6308 of *Lecture Notes in Computer Science*, pages 206–220, 2010.

[35] Ramiz Gindullin, Nicolas Beldiceanu, Jovial Cheukam-Ngouonou, Rémi Douence, and Claude-Guy Quimper. Composing biases by using cp to decompose minimal functional dependencies for acquiring complex formulae. In *AAAI Conference on Artificial Intelligence*, 2024.

[36] Ramiz Gindullin, Nicolas Beldiceanu, Jovial Cheukam Ngouonou, Rémi Douence, and Claude-Guy Quimper. Boolean-arithmetic equations : Acquisition and uses. In *International Conference on the Integration of Constraint Programming (CPAIOR)*, 2023.

[37] Narendra Jussien Guillaume Rochart. Implémenter des contraintes globales expliquées. *Premières Journées Francophones de Programmation par Contraintes, CRIL - CNRS FRE 2499*, pages 393–402, Jun. 2005.

[38] Pierre Hansen and Gilles Caporossi. Autographix : An automated system for finding conjectures in graph theory. *Electron. Notes Discret. Math.*, 5 :158–161, 2000.

[39] Dávid Hanák. Implementing global constraints as graphs of elementary constraints. *Acta Cybernetica*, 16 :241–258, Jan. 2003.

[40] Brahim Hnich, Zeynep Kiziltan, and Toby Walsh. CSPLib problem 030 : Balanced academic curriculum problem (bacp), 1999.

[41] Bat-Sheva Ilany and Dina Hassidov. Solving equations with parameters. *Creative Education*, 05 :963–968, 01 2014.

[42] Wolfram Research, Inc. Mathematica, Version 14.0. Champaign, IL, 2024.

[43] J. Jaffar and J.-L. Lassez. Constraint logic programming. In *Proceedings of the 14th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, POPL '87, page 111–119, 1987.

[44] Richard M. Karp. *Reducibility among Combinatorial Problems*, pages 85–103. Plenum Press, 1972.

[45] Donald E. Knuth. *The Art of Computer Programming, Volume 4, Fascicle 4 : Generating All Trees–History of Combinatorial Generation (Art of Computer Programming)*. 2006.

[46] Samuel Kolb, Sergey Paramonov, Tias Guns, and Luc De Raedt. Learning constraints in spreadsheets and tabular data. *Mach. Learn.*, 106(9-10) :1441–1468, 2017.

[47] Michael Kommenda, Bogdan Burlacu, Gabriel Kronberger, and Michael Affenzeller. Parameter identification for symbolic regression using nonlinear least squares. *Genetic Programming and Evolvable Machines*, 21 :471–501, 2019.

[48] John R. Koza. Genetic programming - on the programming of computers by means of natural selection. In *Complex adaptive systems*, 1993.

[49] Mohit Kumar, Stefano Teso, and Luc De Raedt. Acquiring integer programs from data. In Sarit Kraus, editor, *International Joint Conference on Artificial Intelligence, IJCAI*, pages 1130–1136, 2019.

[50] William La Cava, Patryk Orzechowski, Bogdan Burlacu, Fabricio de Franca, Marco Virgolin, Ying Jin, Michael Kommenda, and Jason Moore. Contemporary symbolic regression methods and their relative performance. In *Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1, 2021.

[51] Tony Lambert, Carlos Castro, Eric Monfroy, and Frédéric Saubion. Solving the balanced academic curriculum problem with an hybridization of genetic algorithm and constraint propagation. pages 410–419, 01 2006.

[52] Craig E. Larson and Nicolas Van Cleemput. Automated conjecturing I : Fajtlowicz's Dalmatian heuristic revisited. *Artif. Intell.*, 231 :17–38, 2016.

[53] Craig E. Larson and Nicolas Van Cleemput. Automated conjecturing III - property-relations conjectures. *Ann. Math. Artif. Intell.*, 81(3-4) :315–327, 2017.

[54] Sharon L. Lohr. *Sampling : Design and Analysis.*, volume 3rd ed. 2021.

[55] Samaneh Sadat Mousavi Astarabadi and Mohammad Mehdi Ebadzadeh. A decomposition method for symbolic regression problems. *Applied Soft Computing*, 62 :514–523, 2018.

[56] Heiner Müller-Merbach. Heuristics and their design : a survey. *European Journal of Operational Research*, 8(1–2) :1–23, 1981.

[57] Olga Ohrimenko, Peter J. Stuckey, and Michael Codish. Propagation = lazy clause generation. In Christian Bessière, editor, *Principles and Practice of Constraint Programming – CP 2007*, pages 544–558, Berlin, Heidelberg, 2007.

[58] François Pachet and Pierre Roy. Automatic generation of music programs. In *Principles and Practice of Constraint Programming*, volume 1713 of *Lecture Notes in Computer Science*, pages 331–345, 1999.

[59] Thorsten Papenbrock, Jens Ehrlich, Jannik Marten, Tommy Neubert, Jan-Peer Rudolph, Martin Schönberg, Jakob Zwiener, and Felix Naumann. Functional dependency discovery : An experimental evaluation of seven algorithms. *Proc. VLDB Endow.*, 8(10) :1082–1093, 2015.

[60] Gilles Pesant. A filtering algorithm for the stretch constraint. In Toby Walsh, editor, *Principles and Practice of Constraint Programming — CP 2001*, pages 183–195, 2001.

[61] Gilles Pesant and P. Soriano. An optimal strategy for the constrained cycle cover problem. *Annals of Mathematics and Artificial Intelligence*, 34(4) :313–325, 2002.

[62] Émilie Picard-Cantin, Mathieu Bouchard, Claude-Guy Quimper, and Jason Sweeney. Learning the parameters of global constraints using branch-and-bound. In *Principles and Practice of Constraint Programming*, pages 512–528, 2017.

[63] Benjamin C. Pierce, Arthur Azevedo de Amorim, Chris Casinghino, Marco Gaboardi, Michael Greenberg, Cătălin Hriţcu, Vilhelm Sjöberg, and Brent Yorgey. *Logical Foundations*, volume 1, Version 6.1 of *Software Foundations*. 2021.

[64] Steve Prestwich. Robust constraint acquisition by sequential analysis. In *ECAI 2020 - 24th European Conference on Artificial Intelligence*, pages 355–362, 2020.

[65] L.B. Rall. Automatic differentiation : Techniques and applications. *Lecture Notes in Computer Science*, vol. 120, Springer, Berlin, 1981.

[66] Andrea Rendl, Ian Miguel, Ian P. Gent, and Peter Gregory. Common subexpressions in constraint models of planning problems. In Vadim Bulitko and J. Christopher Beck, editors, *Symposium on Abstraction, Reformulation, and Approximation*, 2009.

[67] Lorna V. Rosas-Tellez, Vittorio Zanella-Palacios, and Jose L. Martínez-Flores. Solution of a modified balanced academic curriculum problem using evolutionary strategies. In *Computational Intelligence*, pages 49–58, 2013.

[68] Jean-Charles Régin. Generalized arc consistency for global cardinality constraint. In *AAAI'96*, pages 209–215, 01 1996.

[69] Jean-Charles Régin. *Modélisation et Contraintes Globales en Programmation par Contraintes*. HDR dissertation, 2004.

[70] Silviu-Marian Udrescu, Andrew K. Tan, Jiahai Feng, Orisvaldo Neto, Tailin Wu, and Max Tegmark. AI feynman 2.0 : Pareto-optimal symbolic regression exploiting graph modularity. *CoRR*, abs/2006.10782, 2020.

[71] Hao Wang. Toward mechanical mathematics. In Jörg Siekmann and Graham Wrightson, editors, *Automation of Reasoning : Classical Papers on Computational Logic 1957–1966*, pages 244–264. 1983.

Index

COLLEGES SCIENCES
BRETAGNE POUR L'INGENIEUR
LOIRE ET LE NUMERIQUE

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

UNIVERSITÉ LAVAL

Titre : Apprentissage automatique de cartes d'invariants d'objets combinatoires avec une application pour la synthèse d'algorithmes de filtrage

Mot clés : Programmation par contraintes, Apprentissage automatique, Objet combinatoire, Conjectures mathématiques, Algorithme de filtrage

Résumé : Pour améliorer l'efficacité des méthodes de résolution de nombreux problèmes d'optimisation combinatoires de notre vie quotidienne, nous utilisons la programmation par contraintes pour générer automatiquement des conjectures. Ces conjectures caractérisent des objets combinatoires utilisés pour modéliser ces problèmes d'optimisation. Ce sont notamment les graphes, les arbres, les forêts, les partitions et les séquences. Contrairement à l'état de l'art, le système, dénommé Bound Seeker, que nous avons élaboré ne génère pas seulement de manière indépendante les conjectures, mais il explicite aussi des liens existant entre les conjectures. Ainsi, il regroupe les conjectures sous forme de bornes précises sur une même variable associée à un même objet combinatoire. Ce regroupement est appelé carte de bornes de l'objet combinatoire considéré. Enfin, une étude consistant à établir des liens entre les cartes générées est faite. Le but de cette étude est d'approfondir les connaissances sur les objets combinatoires et de développer des prémices de preuves automatiques des conjectures. Pour montrer la cohérence des cartes générées par le Bound Seeker, nous élaborons quelques preuves manuelles des conjectures découvertes par le Bound Seeker, ce qui permet de démontrer la pertinence de quelques nouveaux théorèmes de bornes. Pour illustrer l'une des utilités pratiques de ces bornes, nous introduisons une méthode de génération semi-automatique d'algorithmes de filtrage qui réduisent l'espace de recherche des solutions d'un problème d'optimisation combinatoire. Cette réduction est faite grâce aux nouveaux théorèmes de bornes que nous avons établis après les avoir sélectionnés automatiquement parmi les conjectures générées par le Bound Seeker. Pour montrer l'efficacité de cette technique, nous l'appliquons avec succès au problème d'élaboration des cursus académiques équilibrés d'étudiants.

Title: Acquiring Maps of Interrelated invariants for combinatorial objects with an application for the synthesis of filtering algorithms

Keywords: Constraint programming, Machine learning, Combinatorial object, Mathematical conjectures, Filtering algorithm

Abstract: To improve the efficiency of solution methods for many combinatorial optimisation problems in our daily lives, we use constraints programming to automatically generate conjectures. These conjectures characterise combinatorial objects used to model these optimisation problems. These include graphs, trees, forests, partitions and Boolean sequences. Unlike the state of the art, the system, called Bound Seeker, that we have developed not only generates conjectures independently, but it also points to links between conjectures. Thus, it groups the conjectures in the form of bounds of the same variable characterising the same combinatorial object. This grouping is called a bounds map of the combinatorial object considered. Then, a study consisting of establishing links between generated maps is carried out. The goal of this study is to deepen knowledge on combinatorial objects and to develop the beginnings of automatic proofs of conjectures. Then, to show the consistency of the maps and the Bound Seeker, we develop some manual proofs of the conjectures discovered by the Bound Seeker. This allows us to demonstrate the usefulness of some new bound theorems that we have established. To illustrate one of its concrete applications, we introduce a method for semi-automatic generation of filtering algorithms that reduce the search space for solutions to a combinatorial optimisation problem. This reduction is made thanks to the new bound theorems that we established after having automatically selected them from the conjectures generated by the Bound Seeker. To show the effectiveness of this technique, we successfully apply it to the problem of developing balanced academic courses for students.