UNIVERSITÉ
CAEN
NORMANDIE

Master Informatique DOP

Projet annuel
Compréhension du filtrage dans les CSP par un module d'explication

LEPETIT Lucie : 21203742
AUBRY Nicolas : 21603763

Enseignants :

Patrice BOIZUMAULT
Grégory BONNET

M1 DOP Informatique

Table des matières

1. *Introduction générale et situation du projet*

1.1 Introduction

1.2 Notre projet

Le sujet qui nous est proposé pour le projet annuel de Master 1 d'informatique est de réaliser un module d'explication pour la compréhension du filtrage dans les CSP.

Ce projet faisait partie de nos choix principaux concernant les projets annuels. En effet lors de la sélection des sujets, quand nous avons aperçu le sujet sur la création d'un module d'explication pour la compréhension du filtrage dans les CSP, nous n'avons pas hésité à le mettre dans notre liste.

Nous avions déjà pu travailler sur les CSP en L3 informatique grâce à l'option "Aide à la décision et intelligence artificielle".

Nous avions déjà quelques connaissances sur ce qu'était une variable, en effet l'année précédente une variable représentée un patient, son domaine était une liste de symptômes avec différentes hauteurs. Par exemple un domaine "fièvre" pouvait avoir trois valeurs dans son domaine "bas", "moyen" et "haut".

Une contrainte elle représentait une incompatibilité entre plusieurs symptômes et leurs domaines respectifs. Par exemple on ne pouvait pas avoir un domaine température "haute" avec un domaine fièvre "basse".

C'est alors naturellement que l'on a souhaité continuer ce que l'on avait pu apprendre en troisième année de licence sur les CSP.

Notre projet annuel lui se concentre sur la création d'un module d'explication pour la compréhension du filtrage dans les CSP.

Que cela signifie t'il ?

Dans les CSP lors de la résolution il n'est pas forcément évident de comprendre au premier regard le résultat obtenu. C'est pourquoi l'explication lors du filtrage du CSP est importante, elle permet de comprendre pourquoi certaines valeurs ont pu être supprimées et pourquoi un domaine est devenu vide.

2. *Présentation des CSP*

En notation complète le "Constraint Satisfaction Problem", symbolise un problème où des contraintes sont imposées sur des variables. Il faut satisfaire toutes les contraintes afin de résoudre un problème CSP.

2.1 Définitions

Un CSP est composé :

— d'un ensemble de variables,
noté X, où X= {x1,x2,....,xn} , x1, x2 ou encore xn sont des variables.

Les variables permettent de représenter des objets (humains, voitures ou encore animaux).

Un CSP est aussi composé d'un ensemble de domaines,
— noté D, où D = {D1,D2, ...,Dn}

Un domaine représente l'ensemble des valeurs possibles qu'une variable peut prendre.
Par exemple le domaine D1 pour la variable x1 s'écrit : D1={1,2,4}, x1 peut prendre ici la valeur 1, 2 ou 4 .

Et enfin, on y trouve un ensemble de contraintes,
— noté C, où C = {c1,c2,..,cn}.

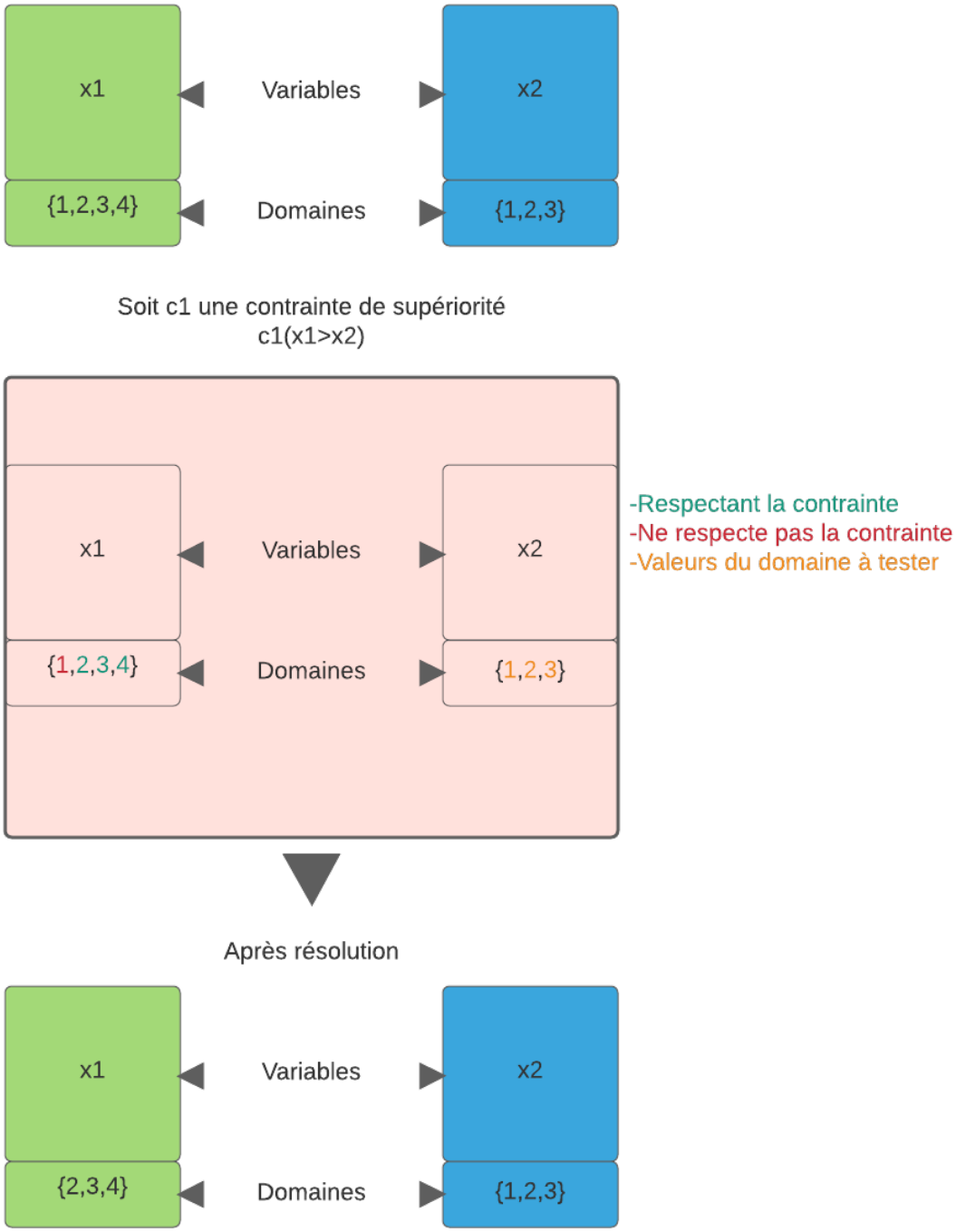
Chacune de ses contraintes porte sur certaines variables / ou toutes les variables, de l'ensemble de variables X.

2.2 Résolution d'un CSP

Pour résoudre un CSP, au fur et à mesure, on doit créer des couples de 'variable-valeur', lorsqu'une contrainte est appliquée. De ce fait, on affecte au fur et à mesure une valeur à une variable, ce qui permet de créer une instanciation, qui est un ensemble de couples 'variable-valeur'. Lorsque toutes les variables ont été affectées d'une valeur, on obtient une instanciation.

Si toutes les variables ont été affectées dans une instanciation ET, et que toutes les contraintes de C satisfont les valeurs affectées aux variables dans cette instanciation, alors l'instanciation est dite complète et on obtient une solution d'un CSP.

Voici un schéma représentant ce principe :



2.3 Le Cas binaire

2.3.1 C'est quoi ?

Dans notre projet, nous devions nous focaliser sur des contraintes de type binaire.

Une contrainte se base sur plusieurs variables, par exemple prenons une contrainte de différence notée c2(xi,xj). Que cela signifie-t-il ?

Elle signifie que pour qu'elle ne soit pas violée il faut qu'au moins l'une des valeurs du domaine de xj soit support pour une valeur du domaine de xi. Si c'est le cas alors la contrainte est respectée.

Nous savons que n'importe quelle contrainte de type n-aire peut se réduire à des contraintes de type binaire en effet prenons c1 une contrainte d'égalité sur trois variable x1, x2, x3.

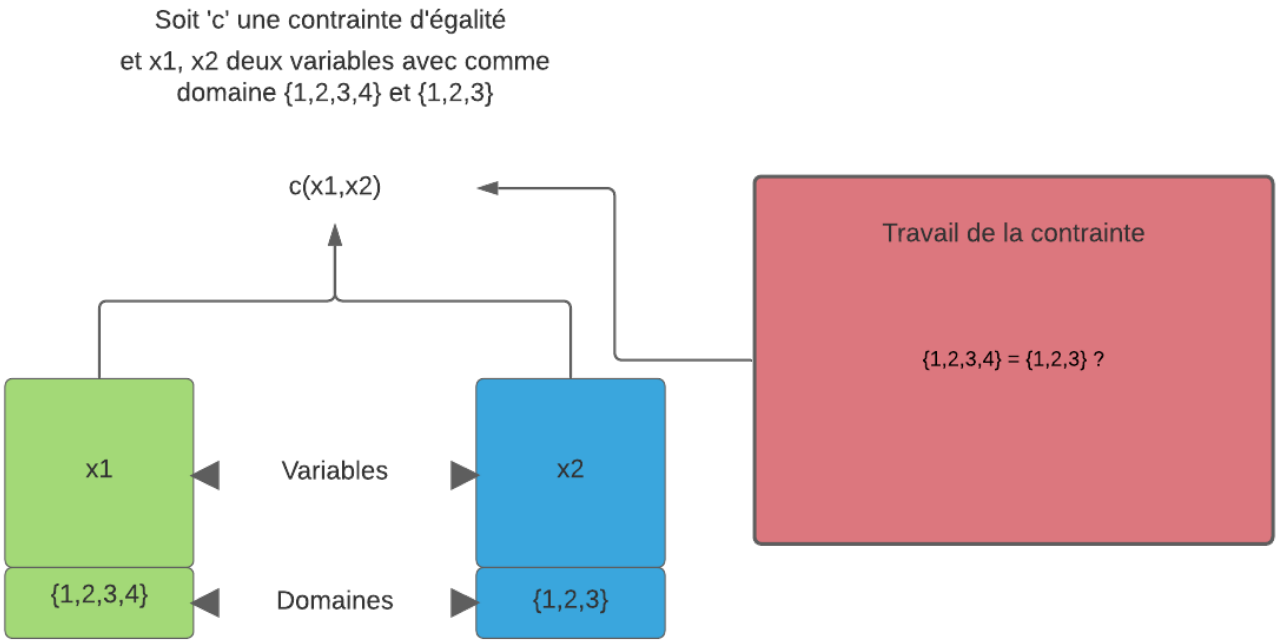
Si nous avons c1(x1 == x2 == x3). Nous pouvons le réécrire comme cela c2(x1 == x2) et c3(x2 == x3). Cette réécriture est équivalente à la contrainte c1.

Une contrainte binaire porte sur deux éléments, qui sont 2 variables. Par exemple, prenons une variable X et une variable Y.

Une contrainte binaire est caractérisée par X (opérateur de la contrainte) Y.

Aussi, dans notre projet nous utilisons une contrainte unaire, c'est une contrainte de type Xi = C où c'est est une constante. Par exemple, Xi = 4 signifie que pour satisfaire cette contrainte, il faut que la valeur du domaine de Xi soit égale à 4.

Vous trouverez le principe des contraintes binaires décrit ici dans le schéma ci-dessous :



3. *Les Explications*

3.1 Qu'est-ce qu'une explication ?

Une explication est un moyen de comprendre pourquoi une valeur d'un domaine a dû être retirée.

3.2 A quoi servent-elles ?

Les explications nous servent à comprendre quand et pourquoi certaines valeurs d'un domaine ne sont plus présentes.

Nous avons deux types d'explication :

— "Pourquoi une valeur a-t-elle disparu d'un domaine ?"
— "Pourquoi un domaine est-il devenu vide ?"

Pour montrer comment se développent ces explications, nous allons vous les montrer sur le problème suivant, qui n'est pas solvable, et qui comporte ces deux types d'explications.

3.3 Notre exemple : la conférence

Ce problème s'intitule le problème de "la Conférence", il s'agit d'une adaptation du problème de la conférence proposée par Cousin (1991), le voici :

Trois individus, Michel, Alain et Pierre doivent se rencontrer pour présenter leurs travaux. Ils ont 4 demi-journées libres et possibles pour faire leurs présentations, et une présentation prend une demi-journée à se faire.

Il y a 4 présentations à faire :

— Michel doit présenter ses travaux à Alain

— Michel doit présenter ses travaux à Pierre

— Pierre doit présenter ses travaux à Michel

— Alain doit présenter ses travaux à Michel

Michel a plusieurs demandes :

— a) Michel veut “avoir vu ce qu’on à dire Pierre et Alain avant de faire ses présentations”, c’est-à-dire d’avoir vu les exposés d’ Alain et de Pierre avant de faire le sien.

— b) Michel “ne veut pas venir la quatrième demi-journée”.

— c) Michel ne veut pas présenter ses travaux à Alain et Pierre en même temps.

Nous représenterons tout ceci par des variables et des noms de contraintes.

Les quatre présentations sont les variables de notre problème :

— Ma : Michel présente à Alain

— Mp : Michel présente à Pierre

— Am : Alain présente à Michel

— Pm : Pierre présente à Michel

Les domaines de ces quatre variables sont les quatre demi-journées possibles, que nous notons {1,2,3,4}. Par conséquent voici les valeurs possibles que peuvent au départ prendre les quatre variables du problème de la conférence :

— Ma : {1,2,3,4}

— Mp : {1,2,3,4}

— Am : {1,2,3,4}

— Pm : {1,2,3,4}

Ici, dix contraintes sont extraites suite à ce problème :

Contraintes d'intégrité :

Michel ne peut pas être à la fois à écouter Pierre et à écouter Alain : Am != Pm

Contraintes d'antériorité :

Michel veut avoir vu les exposés d' Alain et de Pierre avant de faire le sien (a) :

— Michel doit avoir vu la présentation d'Alain, avant de faire sa présentation à Alain : Ma > Am

— Michel doit avoir vu la présentation de Pierre, avant de faire sa présentation à Pierre : Mp > Pm

— Michel doit avoir vu la présentation de Pierre, avant de faire sa présentation à Alain : Ma > Pm

— Michel doit avoir vu la présentation de Alain, avant de faire sa présentation à Pierre : Mp > Am

Contrainte de non simultanéité :

Michel ne veut pas être à la fois à présenter à Pierre et à présenter à Alain (c) : Ma != Mp

Contrainte de quatrième demi-journée :

Michel "ne veut pas venir la quatrième demi journée" (b) :

— La présentation de Michel à Alain ne peut pas être la 4ème demi-journée : Ma !=4

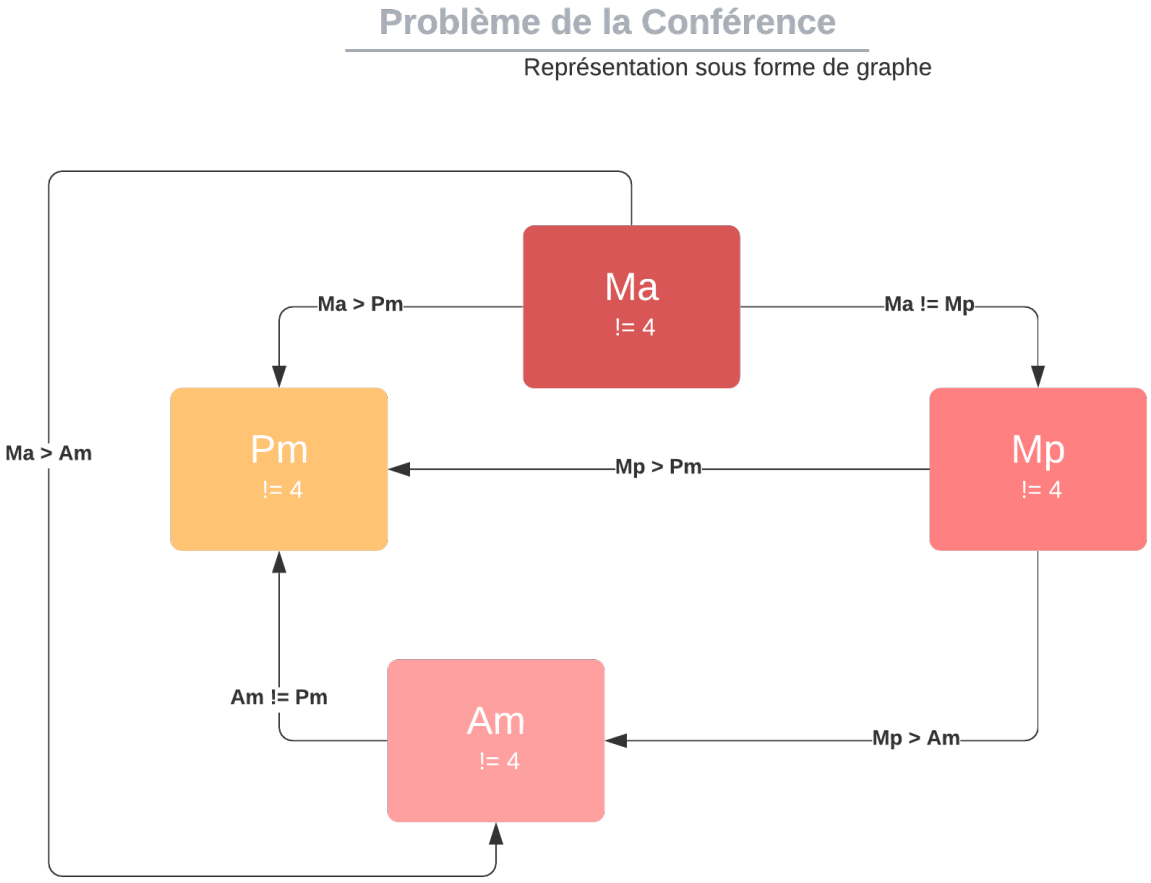
— La présentation de Michel à Pierre ne peut pas être la 4ème demi-journée : Mp !=4

— La présentation de Pierre à Michel ne peut pas être la 4ème demi-journée : Pm !=4

— La présentation de Alain à Michel ne peut pas être la 4ème demi-journée : Am !=4

Voici sous forme d'un graphe, avec pour nœuds chacunes des quatre variables du problème, et où chacuns des arcs représentent les différentes contraintes présentées ci-dessus.

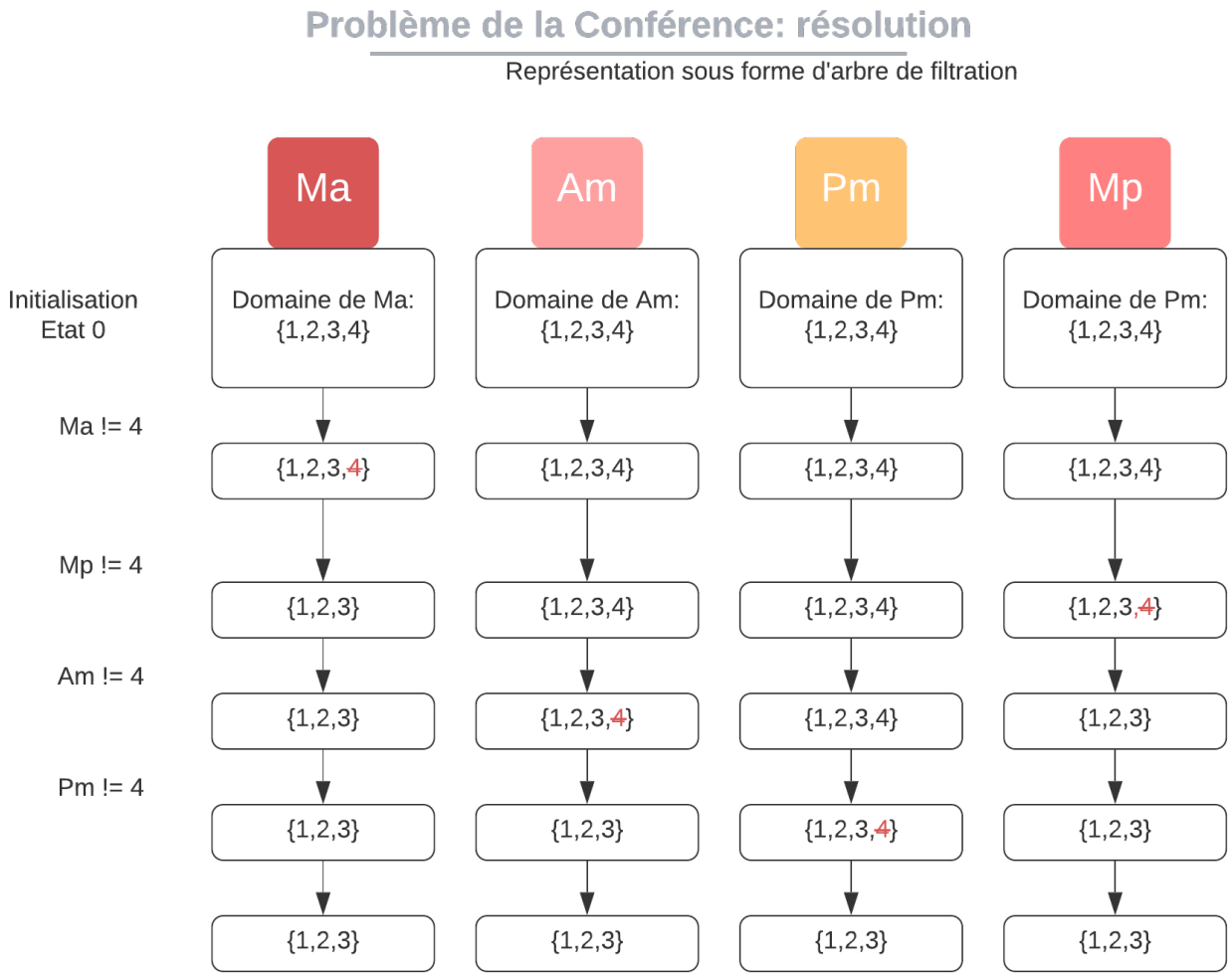
La contrainte de la quatrième demi-journée étant une contrainte unaire, nous plaçons directement pour ce schéma la contrainte à l'intérieur du nœud.



On peut voir qu'ici, un grand nombre de contraintes sont imposées. Ce problème est-il solvable ? (Nous allons vous montrer les étapes que réalise notre algorithme)

Aperçu de la résolution avec explication de cet exemple demandé :

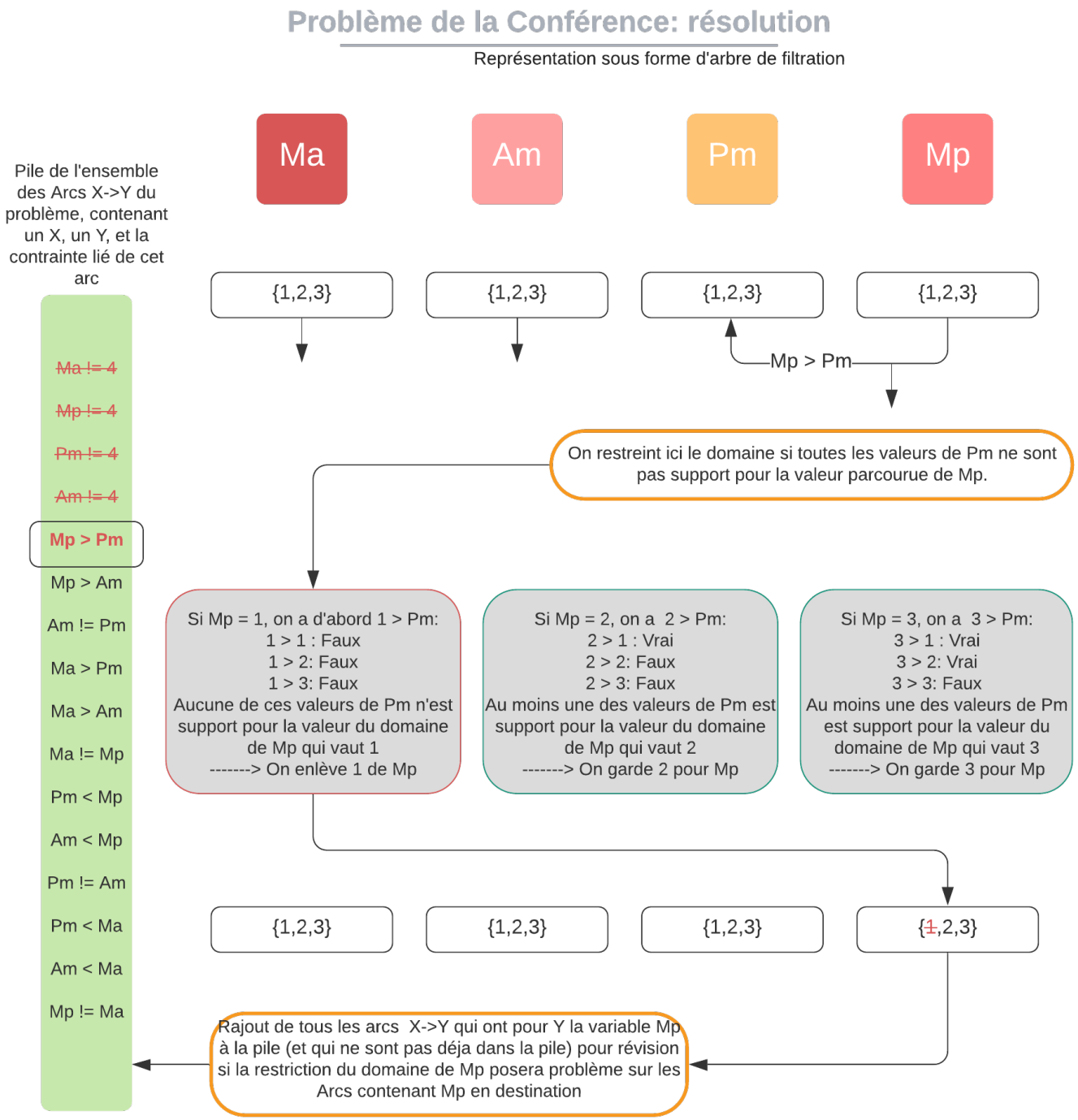
Etape 1 : Avec les contraintes unaires présentes pour la quatrième demi-journée, voici à quoi ressemble les domaines de chacunes des quatre variables :



Ici, l'explication est qu'après chacune des contraintes que l'on passe, lorsqu'une valeur du domaine de la variable est retirée, c'est qu'elle n'est pas support pour la valeur donnée en constante, ici 4, car les variables doivent avoir une valeur différente de 4.

Par conséquent, on retire 4 de chacun des domaines de chaque variable.

Passons ensuite aux contraintes binaires.

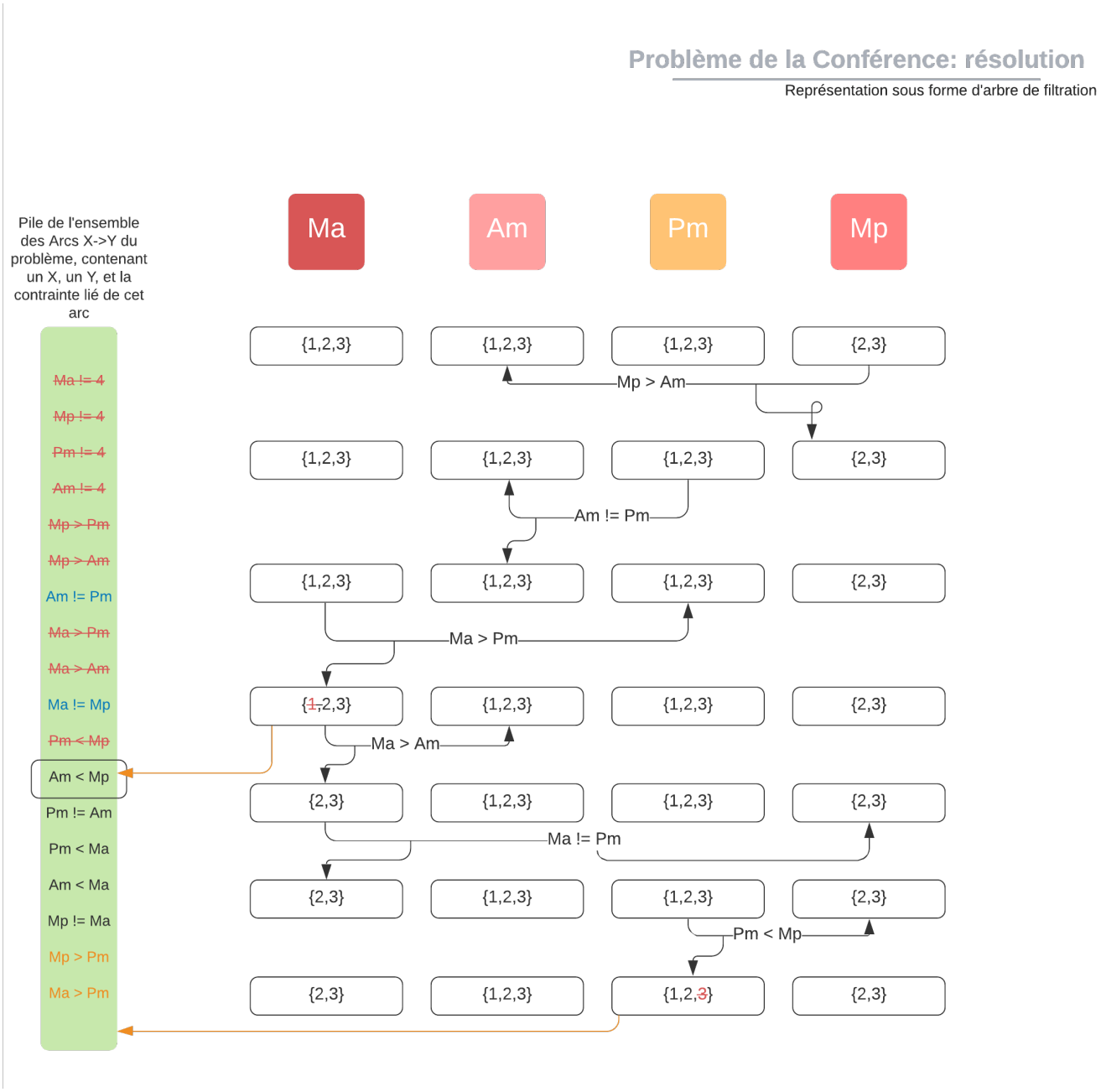


Dans le schéma ci dessus, on peut voir le premier type d'explication qui est :

— "Pourquoi une valeur a-t-elle disparu du domaine ? "
Ici on pose la contrainte binaire Mp > Pm. Comme c'est démontré dans ce schéma, au niveau des trois encadrés gris, seule la valeur 1 du domaine de Mp a disparu, car il ne reste aucune valeur du domaine de Pm pour laquelle la valeur 1 de la variable Mp serait supérieur à une des valeurs de Pm.

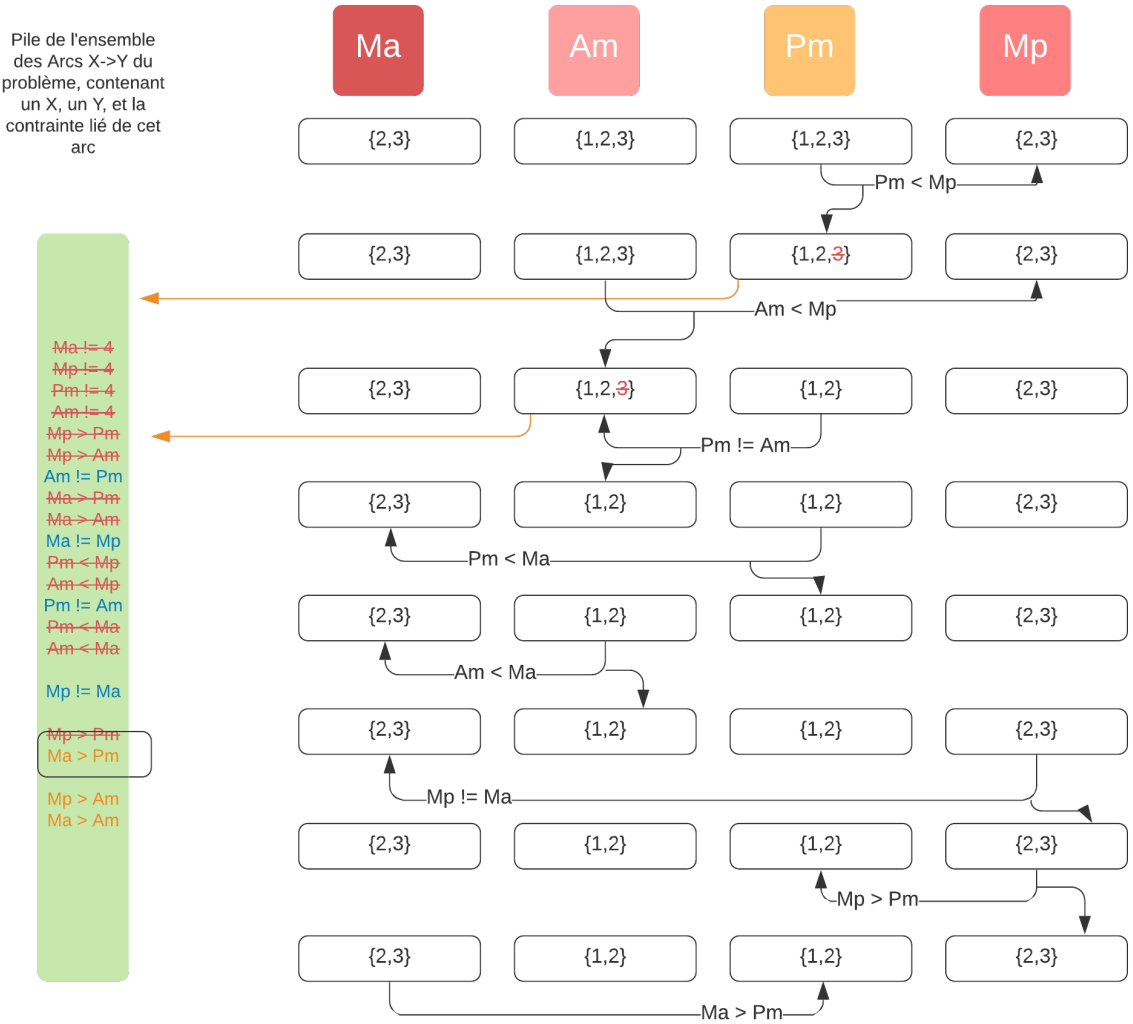
Dans ce schéma, on décrit aussi succinctement le fonctionnement de l'algorithme AC3 qui parcourt une pile d'arcs composé d'un nœud X, d'un nœud Y, et d'une contrainte. Nous verrons plus en détail cet algorithme par la suite.

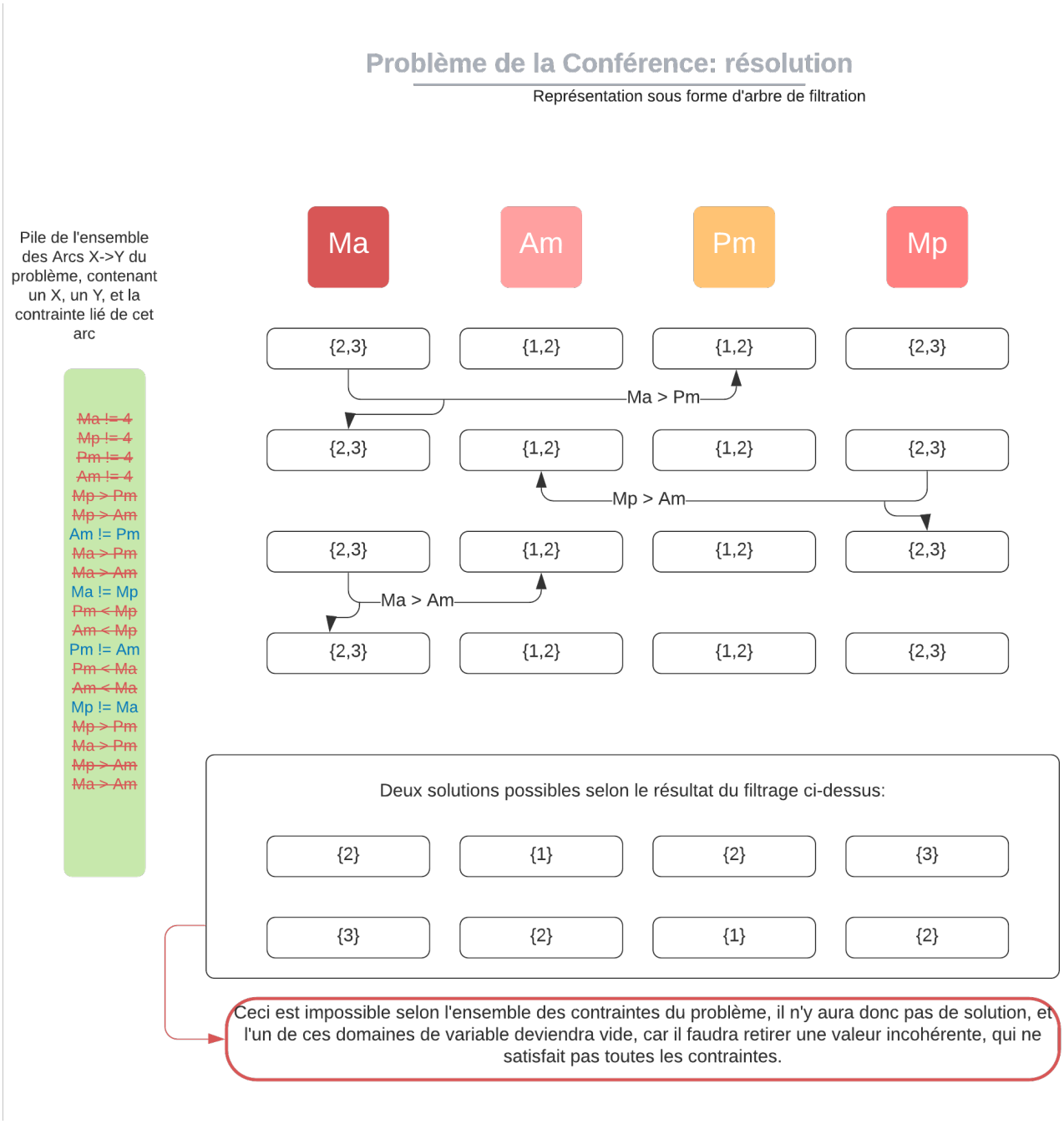
Mais voici un aperçu de la résolution du problème :



Problème de la Conférence: résolution

Représentation sous forme d'arbre de filtration





Comme on peut le voir ci-dessus, le problème ne pourra pas donner de solution valide.

D'une part, car il y a 4 exposés à faire et 4 demi-journées disponibles.

Or, dès le début, on aperçoit déjà un souci du fait qu'il y a la quatrième demi-journée qui est retirée du problème pour l'ensemble des 4 exposés, car Michel est présent à chacun d'entre eux.

Ici, pour le second type d'explication “"Pourquoi un domaine est-il devenu vide ?", on peut voir qu'un domaine est devenu vide suite à une multitude de restrictions du domaine d'une variable.

Si l'on exécute toutes les contraintes du problème une dernière fois pour vérifier si une des deux solutions détectées est valable, arc-cohérente, ce ne sera pas le cas. Par exemple, la contrainte Mp>Am ne sera pas satisfaite pour la deuxième solution, 2 n'est pas supérieur à 2, ou encore la contrainte Ma > Pm pour la première solution, 2 n'est pas supérieur à 2.

4. *Implémentation du projet*

4.1 Les algorithmes et méthodes utilisés

4.1.1 L'algorithme "AC-3" : arc cohérence

AC-3($\mathcal{R}$)
input: a network of constraints $\mathcal{R} = (X, D, C)$
output: $\mathcal{R}'$ which is the largest arc-consistent network equivalent to $\mathcal{R}$
1. **for** every pair $\{x_i, x_j\}$ that participates in a constraint $R_{ij} \in \mathcal{R}$
2. $\quad$ *queue* $\leftarrow$ *queue* $\cup \{(x_i, x_j), (x_j, x_i)\}$
3. **endfor**
4. **while** *queue* $\neq \{\}$
5. $\quad$ select and delete (x_i, x_j) from *queue*
6. $\quad$ *Revise*$((x_i), x_j)$
7. $\quad$ **if** *Revise*$((x_i), x_j)$ causes a change in D_i
8. $\quad\quad$ **then** *queue* $\leftarrow$ *queue* $\cup \{(x_k, x_i), i \neq k\}$
9. $\quad$ **endif**
10. **endwhile**

L'algorithme AC-3 est un algorithme qui permet la résolution de CSP. Son principe est très simple. Il prend en entrée une liste de contraintes avec les variables et leurs domaines.

Ce qu'il va faire ensuite c'est qu'il va construire un arc, c'est-à-dire qu'il va prendre une contrainte de type Xi > Xj par exemple. Et pour chacune de ses contraintes il va créer son "opposé" c'est-à-dire dans notre cas Xj < Xi, on appelle cela un arc. Il va ensuite ajouter les deux arcs dans une pile notée "Queue".

Ensuite il va boucler sur la taille de la pile , tant qu'elle n'est pas vide et va prendre un arc (Xi,Xj), présent au sommet de la pile avec la contrainte associée.

Dans cet arc il va regarder toutes les valeurs du domaine qui respecte la contrainte dans l'ordre Xi contraint Xj avec l'algorithme "revise".

Si des valeurs sont enlevées du domaine de Xi alors il faudra ajouter dans la pile un arc du type (Xk,Xi) avec sa contrainte associée, dans lequel Xi est contraint par Xk. Cela va permettre d'enlever toutes les valeurs qui ne remplissent pas les contraintes.

Et on fait ça jusqu'à ce que la pile d'arcs soit vide.

4.1.2 La méthode "révise"

REVISE$((x_i), x_j)$
input: a subnetwork defined by two variables $X = \{x_i, x_j\}$, a distinguished variable x_i, domains: D_i and D_j, and constraint R_{ij}
output: D_i, such that, x_i arc-consistent relative to x_j
1. **for** each $a_i \in D_i$
2. **if** there is no $a_j \in D_j$ such that $(a_i, a_j) \in R_{ij}$
3. **then** delete a_i from D_i
4. **endif**
5. **endfor**

L'algorithme revise permet à partir d'un couple de variables de renvoyer les variables avec leurs domaines respectant la contrainte.

Pour cela il va prendre en entrée un couple de variables (Xi,Xj) avec leurs domaines respectifs ainsi que la contrainte associée.

L'algorithme va ensuite boucler sur toutes les valeurs "vi" du domaine de Xi.

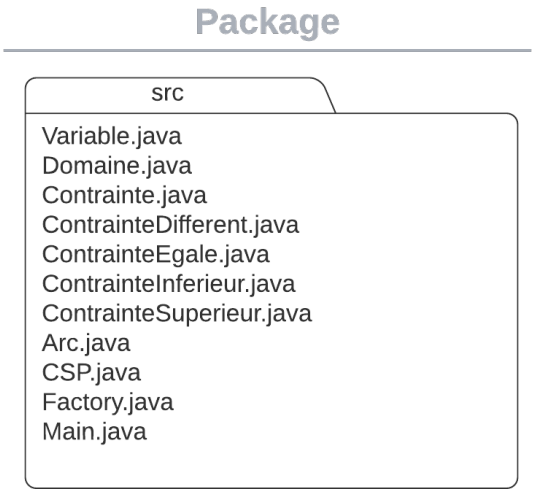
Il va vérifier que cette valeur "vi" du domaine de Xi ne viole pas la contrainte avec les valeurs "vj" du domaine de Xj.

S'il existe au moins une valeur "vj" qui est support (donc qui satisfait la contrainte "vi-contrainte-vj") pour la valeur "vi"de Xi, alors cette valeur "vi"de Xi n'est pas éliminée du domaine de Xi.

En revanche, si toutes les valeurs "vj" du domaine de Xj ne sont pas supports pour la valeur "vi"de Xi, alors cette valeur "vi" du domaine de Xi va être retirée du domaine de Xi.

4.2 Architecture de notre projet : l'implémentation

4.2.1 Le package



Voici à quoi ressemble notre package pour l'implémentation du module. Il est composé de 11 classes différentes.

Main est l'exécutable qui permet de lancer notre programme.

Le programme se résout à l'aide de la classe CSP, qui reprend toutes les données issues de la classe d'exemple Factory, où les contraintes, les variables et les domaines sont créés.

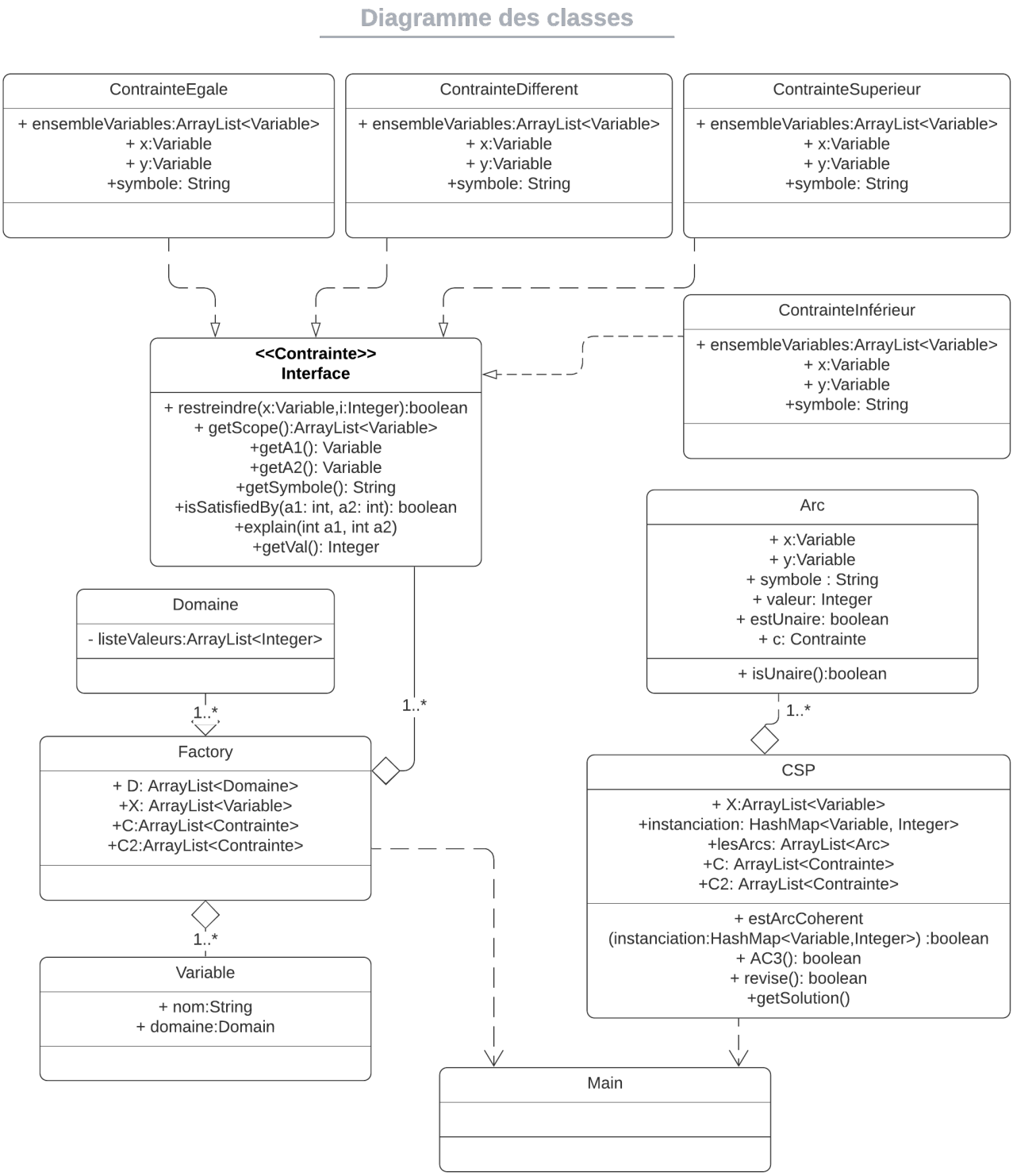
Les 4 types de contraintes sont issues de la classe abstraite Contrainte, qui permet d'implémenter les principales méthodes d'une contrainte.

La classe Arc est utilisée dans la classe CSP, qui crée des arc X->Y pour l'algorithme AC3, présent dans la classe CSP.

Nous avons plusieurs Factory disponibles, la Factory qui contient l'exemple de la conférence est la Factory2.

4.2.2 Diagramme des classes

Vous trouverez ci-après la construction de notre Diagramme de classe, pour avoir un meilleur aperçu de notre module.



4.3 Fonctionnement

Notre code actuel sert sur l'exemple de la conférence, mais il peut facilement être remplacé par n'importe quels autres problèmes de satisfaction de contrainte qui contient les contraintes de supériorité, infériorité, égalité ou différence et qui sont composées de domaines avec des valeurs numériques.

Il suffit pour cela de se rendre dans la classe Factory.java d'ajouter les variables, les domaines ainsi que les contraintes, et ensuite compiler le programme et nous aurons en sortie un résultat avec les explications liées aux retraits des différentes valeurs des domaines.

Pour mieux faire fonctionner notre programme, il faut de plus séparer les contraintes unaires et binaires : si elles sont de la forme binaire il faut les implémenter en les ajoutant dans l'ensemble de contraintes nommé C et sinon si elles sont binaires, dans l'ensemble de contraintes nommées C2.

4.3.1 Fonctionnement de nos classes représentées en java

Pour la réalisation de notre programme, nous nous sommes aidés des bases que nous avions appris dans le cours d'aide à la décision, où l'on devait réaliser un outil pour réaliser un diagnostic médical, à l'aide de règles, de variables et de domaines. Nous avons aussi essayé de mieux comprendre l'algorithme AC3 via la vidéo du professeur John Levine : Constraint Satisfaction : the AC-3 algorithm - YouTube .

Lien de la vidéo

Nous avons représenté les trois principaux éléments d'un CSP par 3 classes principales.

Contrainte :

C'est une classe abstraite "Contrainte", définissant une contrainte en général, avec plusieurs sous-classes pour définir chacun des types de contraintes nécessaires au problème que nous devons implémenter.

— pour la contrainte de différence, où x $\neq$ y , il y a la classe ContrainteDifferent

— pour la contrainte d'égalité, où x = y , il y a la classe ContrainteEgale

— pour la contrainte de supériorité, où x > y , il y a la classe ContrainteSuperieur

— pour la contrainte d'infériorité, où x < y , la classe ContrainteInferieur

Pour ces 4 types de contraintes, il est possible de les utiliser en mode contrainte binaire (x>y par exemple) ou bien en mode contrainte unaire (x!=5 par exemple).

Variable :

La classe Variable, est la classe qui nous permet de représenter une variable ainsi que son domaine. Les types de domaines des variables que nous avons choisi de traiter sont des entiers.

Notre classe variable est très simple, elle est composée d'attributs pour donner un nom, ainsi qu'un domaine.

Domaine :

Pour implémenter un domaine, nous avons défini une classe "Domaine". Un domaine d'une variable est représenté par une liste de valeurs possibles, qui sont des nombres .

5. *Conclusion*

5.1 Ce que nous avons appris

Ce que le projet nous a appris, c'est une connaissance plus fine sur les CSP et une compréhension approfondie sur le fonctionnement des différents algorithmes que nous avons utilisés.

Nous avons aimé ce sujet malgré les difficultés rencontrées durant l'élaboration de celui-ci.

Nous voyons à travers l'exemple du problème de la conférence que beaucoup de problèmes existants aujourd'hui peuvent être transformés en CSP et ainsi avoir une solution (un support pour toutes les variables ou alors un échec).

Nous pouvons ainsi traiter des problèmes du quotidien plus facilement, rapidement et efficacement grâce au CSP et aussi mieux les comprendre grâce aux explications générées par notre module.
Ce qui peut être utile pour comprendre pourquoi certains domaines sont vides ou alors pourquoi avons-nous dû réduire des domaines, cela devient très utile quand le CSP a de plus en plus de contraintes.

5.2 Les soucis rencontrés

Nous avons rencontré quelques soucis durant la création du module. En effet nous avions plusieurs erreurs concernant les contraintes d'infériorité et les contraintes de supériorité, mais elle ont été résolues, et maintenant le souci est dù aux contraintes de différence.

Actuellement, les contraintes de diffrence ne fonctionnent pas correctement, mais nous vous avons fournis d'autres exemples , comme la Factory numéro 3 qui fonctionne.

Seules les contraintes de différence de type binaires posent problème, lorsqu'il s'agit d'une contrainte de type unaire cela fonctionne bien.

Nous avons donc passé un temps important à essayer de trouver les problèmes et ainsi de pouvoir traiter toutes les contraintes correctement.

5.3 Les améliorations possibles

Bien sûr, comme tout projet, le nôtre peut bénéficier d'améliorations. Une des premières améliorations serait d'avoir encore plus d'explications afin de mieux comprendre le retrait de valeurs.

Une autre amélioration serait d'ajouter d'autres versions de l'algorithme d'arc consistance, en effet AC-3 que nous utilisons dans notre projet est un algorithme qui date de 1977. Depuis, de nouveaux algorithmes concernant l'arc consistance existent. Ils peuvent être plus rapides que AC-3 et ce serait intéressant de comparer les performances des algorithmes.

Une amélioration qui pourrait être bien serait de ne pas avoir seulement des contraintes binaires mais aussi des contraintes n-aires. On sait qu'une contrainte de taille N peut être réduite à des contraintes binaires donc l'implémentation pourrait se faire sur notre projet.

Aussi l'ajout de domaines non numériques pourrait ainsi permettre d'implémenter des nouveaux CSP.

L'une des dernières améliorations qui pourraient être ajoutées est d'avoir d'autres types de contraintes, des contraintes d'autres types que l'égalité ou la différence.

L'ajout d'un algorithme de Backtracking permettant de résoudre complètement un CSP, n'entrait pas dans le cadre de notre projet, mais il serait aussi une bonne voie d'amélioration pour la résolution d'un problème avec encore plus de valeurs.

5.4 Ce que ça nous a apporté

Ce projet nous a montré un côté plus professionnel, en effet nous avons eu plusieurs réunions avec notre client du projet qui est aussi notre professeur (Patrice Boizumault), nous avons essayé de jouer le jeu.

Nous avons dû aussi jongler entre le projet annuel ainsi que tous les autres projets que nous avons eus pour le semestre.

Nous avons essayé d'appliquer au maximum les conseils de notre professeur.

6. *Annexe*

6.1 Bibliographie

— *Thèse de M Narendra JUSSIEN sur la relaxation de contraintes pour les problèmes dynamiques*

— *Coins : a constraint-based interactive solving system de Samir Ouis, Narendra Jussien et Patrice Boizumault*

— *Introduction aux CSP de Patrice Boizumault*

6.2 Liens utiles

— *Lien sur l'explication de AC-3*

— *Compréhension des CSP*

— *Les algorithmes AC-3 et revise*

— *Pour comprendre comment fonctionne un CSP*