Conception et réalisation d'algorithmes d'explication et de recherche d'optimum dans le contexte de produits hautement configurables

BOULAT Antoine

Responsable pédagogique UTT :
M. Alexandre Charles

Branche : Génie Mécanique

Semestre : Automne 2019

Résumé

Acuity Solutions développe un outil de Product Line Engineering, en accord avec la transformation de l'industrie qui tend vers une customisation afin d'être plus en adéquation avec les besoins réels du client. Cette solution logicielle permet donc de modéliser la variabilité de produits hautement configurables (~10^100 combinaisons possibles) et le sujet de ce stage porte sur le moteur combinatoire. Ce stage visait donc à concevoir et développer des algorithmes d'explications de propagation de contraintes, implémenter un algorithme de propagation de contraintes non parfait mais de complexité temporelle polynomiale, ou encore des algorithmes stochastiques visant à simuler des proportions de configurations.

Ce stage me permet de développer mes compétences algorithmiques, de recherche opérationnelle, de graphes et de PLM tout en restant proche des problématiques concrètes des différents clients d'Acuity Solutions, tels que PSA, Alstom ou encore Arquus.

Entreprise : Acuity Solutions

Lieu : Villeurbanne, 69

Responsable : François Greugny, directeur de l'entreprise

Mots clés (CF Thésaurus)

- Génie Logiciel
- Optimisation Mathématique
- Analyse des données -- Logiciels
- Mécanique

utt
UNIVERSITÉ DE TECHNOLOGIE
TROYES

Remerciements

En premier lieu, je souhaite vivement remercier François Greugny de m'avoir accueilli dans son entreprise. Ses qualités d'écoute, de pédagogie, son dynamisme et sa sympathie m'ont permis de progresser dans un sujet nouveau pour moi. Il m'a offert l'opportunité d'étendre mes compétences acquises à l'UTT et m'ouvrir à de nouveaux champs d'activités afin de mieux cibler mes aspirations professionnelles.

Je souhaite également remercier Felix Pinard, un développeur qui nous a rejoint au cours de mon deuxième mois de stage. Lui-même avait réalisé son stage de quatrième année dans l'entreprise et travaillé sur le moteur combinatoire de PLEIADE. Après une période d'adaptation j'ai beaucoup progressé dans la compréhension du système en bénéficiant de ses compétences. Ses qualités professionnelles, sa vivacité intellectuelle et sa sympathie ont été de réels atouts dans la réalisation de mon stage. Son arrivée m'a en outre appris à gérer un projet de développement en synergie.

Grace à l'aide apportée par l'ensemble de l'équipe d'Acuity, ma phase d'acquisition de connaissances a été accélérée. Je tiens à remercier tout particulièrement les développeurs qui m'ont facilité la prise en main des outils Java et Eclipse.

Je souhaite remercier Alexandre Charles qui m'a suivi et conseillé tout au long du stage, il s'est assuré de mon bien être dans l'entreprise et de la conformité de mon stage avec le sujet initial ainsi que Messieurs Guillaume Ducelier et Abel Cherouat qui ont été très présents lors de ma recherche de stage. Ils ont su répondre à mes interrogations et m'orienter afin de cerner au plus près mes aspirations en accord avec mes compétences.

Finalement, je tiens à remercier Isabelle Bourdier pour sa gentillesse et son aide lors de mon arrivée à Lyon. En effet, alors que j'avais des soucis pour trouver un logement, elle m'a accueilli chez elle. Cette marque de sympathie m'a beaucoup touché.

Table des matières

I. Introduction

1. Contexte industriel

Le Product Line Engineering (PLE) s'inscrit dans la quatrième révolution industrielle – l'industrie 4.0. Ce concept correspond à de nouvelles méthodes d'organisation des moyens de production et de conception à l'aide d'outils divers (Internet of Things, analyse de données, impression 3D par exemple), dans une optique de passer d'une production de masse à une production plus personnalisée, d'une maintenance plus précise/spécifique des machines, de diminuer les coûts de développement pour ne citer que ces avantages.

Le premier point va nous intéresser plus particulièrement : le PLE est un moyen de gérer la diversité en tirant parti des similitudes des produits tout en gérant leurs différences. Cette approche permet lors du développement produit une meilleure réutilisabilité des différents composants mécaniques et dans le même temps une meilleure customisation du produit pour répondre au mieux aux besoins du client.

2. Présentation de l'entreprise

Acuity Solutions s'inscrit dans ce contexte et développe un logiciel de PLE : PLEIADE (Product Line Engineering In Advanced Diversity Editor). C'est une PME qui compte 18 salariés répartis entre le bureau de développement composé de 4 développeurs et par ailleurs 14 consultants. Du fait de la taille de l'entreprise, certains de ses consultants fonctionnels ont un rôle technique. Ce dernier aspect est un point intéressant, à mettre en perspective avec mon stage réalisé dans un grand groupe, Safran Landing System, en 4ème année où les rôles sont plus cantonnés et moins polyvalents.

L'équipe de développement travail avec le langage de programmation Java et l'IDE (environnement de développement) Eclipse. Le logiciel PLEIADE intègre un grand nombre de fonctionnalités. On peut notamment citer la définition, la configuration et l'affichage 3D de produits pour la partie conception mais aussi par exemple la liste des opérations nécessaires à la réalisation du produit pour la partie production. De plus, il se couple avec Arras pour gérer la partie PLM.

Acuity Solutions est une entreprise reconnue dans le milieu de la gestion de la diversité. Parmi ses clients on peut citer Arquus, PSA, Alstom. Plusieurs autres projets de partenariat sont à l'étude. Son chiffre d'affaire reflète son dynamisme : il était de 1.436M € en septembre 2019, affichant une croissance de 22% par rapport à septembre 2018 (1.117M €).

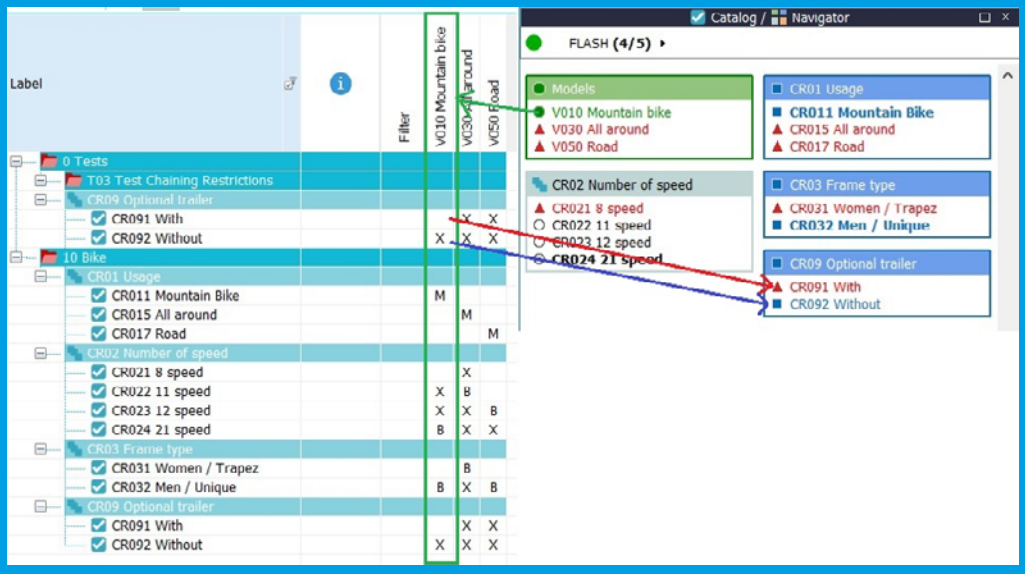
3. Problématique

Mon projet s'inscrit dans le moteur du logiciel appelé 3C (pour Cohérence Complétude et Concision).
Ce moteur est compilé sous forme de bibliothèque dans le logiciel, il permet de gérer l'aspect combinatoire (une combinaison représentant une configuration possible du produit).
Lorsqu'un utilisateur configure un produit sur PLEIADE, il doit choisir différentes options par-

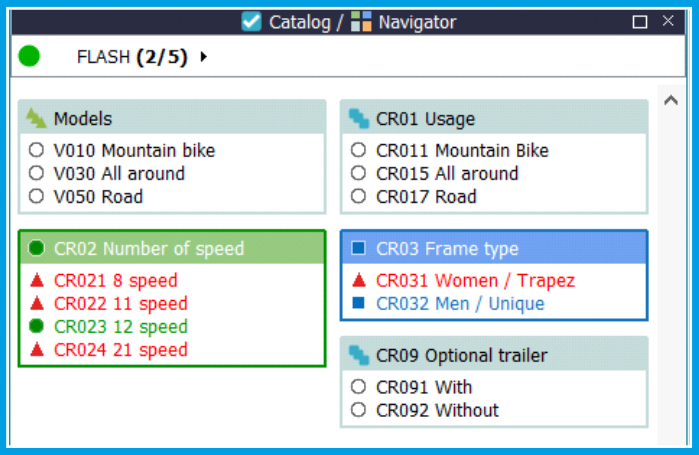
mi plusieurs critères (groupe de valeurs définies au préalable). Pour chaque critère, une seule option doit être sélectionnée. De plus entre ces valeurs, des restrictions ou des implications peuvent s'appliquer. Certaines valeurs ne peuvent être sélectionnées que dans certains modèles.

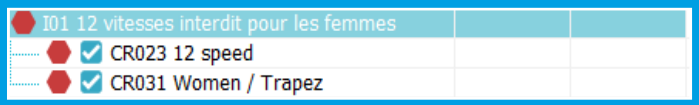
Exemple dans le logiciel

la sélection du model « V010 Mountain bike » restreint les choix comme nous pouvons le voir dans la capture d'écran de droite

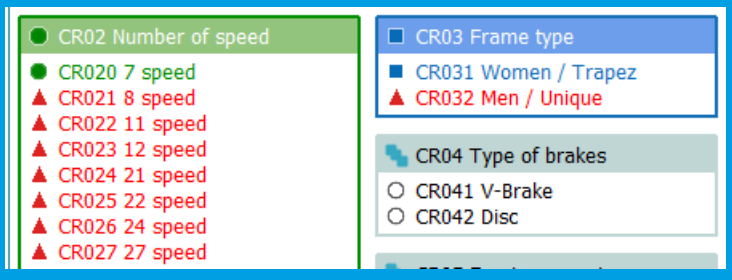


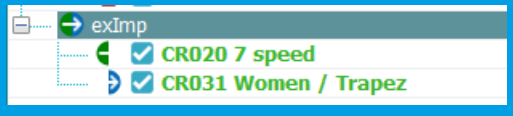
Exemple de restriction entre la valeur « CR023 12 speed » et « CR031 Women / Travez ». Lorsque la « CR023 12 speed » est sélectionnée, « CR031 Women / Travez » adopte le statut interdit et réciproquement.





Exemple d'implication entre « CR020 7 speed » et « CR031 Women / Trapez ». Lorsque « CR020 7 speed » est sélectionné, « CR031 Women / Trapez » est alors déduit.



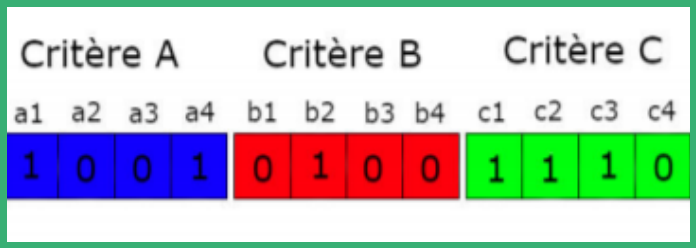


Des conditions supplémentaires s'appliquent : une seule valeur par critère peut être sélectionné. Cependant, il existe un mode permettant la sélection de plusieurs valeurs dans le même critère. Nous y reviendrons dans la description des taches effectués.

L'objectif de l'application est donc au fur et à mesure des sélections par l'utilisateur, de lui afficher quelles sont les options encore sélectionnables, celles déduites ou encore celles interdites. D'un point de vue théorique, la recherche de configurations respectant toutes les contraintes (problème de « satisfaction de contraintes ») est un problème de type SAT (satisfaisabilité booléenne) – en réalité, ce problème correspond à un ensemble de problème SAT. Plus précisément, la recherche des statuts pour chaque valeurs est de type SAT, soit un problème pour chaque statuts. Ce problème appartient à la classe NP, ce qui signifie qu'il n'existe pas d'heuristique générique et de complexité polynomiale.

Nous pouvons signaler la possibilité de formaliser de différentes manières ce type de problématique [1]. Celle retenue sur PLEIADE exploite la notion de Cas d'Emploi pour représenter l'ensemble des combinaisons. Un Cas d'Emploi valide possède au moins une option de valide par critère. Il représente l'ensemble des configurations possibles sous forme d'un tableau de bit. Ce choix a été fait afin d'optimiser les traitements dans le moteur. En effet, le moteur se doit d'être le plus efficace possible et manipuler directement des structures comme des tableaux de bit est plus efficace que des structures de type liste en Java. Lorsqu'une option est interdite, son bit est à 0, lorsqu'une option est sélectionnable, son bit est à 1. Néanmoins, lorsqu'il n'y a plus qu'une option à 1 dans un critère, elle est alors déduite (ou sélectionnée). Le critère est alors résolu.

Exemple



Ce Cas d'emploi représente la situation (a1 ∨ a4) ∧ (b2) ∧ (c1 ∨ c2 ∨ c3) = ((a1) ∧ (b2) ∧ (c1)) ∨ ((a4) ∧ (b2) ∧ (c1)) ∨ ((a1) ∧ (b2) ∧ (c2)) ∨ ((a4) ∧ (b2) ∧ (c2)) ∨ ((a1) ∧ (b2) ∧ (c3)) ∨ ((a4) ∧ (b2) ∧ (c3))

Dans le cas général, pour obtenir le nombre de configurations possibles, il faut multiplier le nombre d'options valides dans chaque critère (dans notre cas 2*1*3 = 6 configurations possibles).

Cet exemple extrêmement simple ne reflète pas la réalité industrielle. En effet, dans les cas extrêmes, comme certaines bases de données Volvo, les combinatoires sont proches du gogol (soit ~10^100). Lorsqu'une sélection est faite, la propagation de contraintes prend environ 10 secondes, ce qui rend l'expérience utilisateur beaucoup moins fluide et plus laborieuse.

Afin de diminuer les temps de traitement, il a été décidé d'abandonner le caractère parfait (tous

les statuts respectent les contraintes du produit) de la propagation et d'implémenter un algorithme dont la propagation ne calculerait pas correctement tous les statuts mais serait d'une complexité temporelle polynomiale. A l'instant où j'écris ces lignes, ce mode de propagation est assuré par un droit dans l'application. Il est réservé à des produits dont les combinatoires sont trop importantes pour utiliser le mode de propagation parfait.

Lors de la sélection faite par des utilisateurs, certaines valeurs deviennent interdites alors qu'aucune restriction ou implication ne concerne directement les valeurs sélectionnées et interdites. En effet, la propagation de l'information se fait entre les modèles, les implications et les restrictions. Ces ensembles font apparaître des comportements complexes qu'il est difficile d'expliquer. Dans cette optique, il a été décidé d'implémenter une fonctionnalité d'explication de la propagation. Malheureusement, les algorithmes utilisés pour la propagation ne permettent pas d'avoir accès à l'ordre ni à l'incidence des restrictions entre elles.

Il a donc fallu implémenter un algorithme d'explication de propagation de contraintes.
D'autres algorithmes répondant à des besoins ont été implémentés :

- Un algorithme permettant de traduire les implications en restriction,
- Un algorithme permettant de traduire les modèles en restriction,
- Un algorithme de fusion des Cas d'Emploi plus performant que celui déjà existant.

Dans une première partie, je ferai le point sur l'ensemble des éléments techniques nécessaires à la compréhension du traitement algorithmique des Cas d'Emploi. Dans une seconde partie, je décrirai l'ensemble des nouveaux algorithmes implémenter en 3C lors de mon stage. Finalement, je dresserai un bilan personnel de ma formation à l'UTT et je donnerai quelques pistes pour l'améliorer.

II. Déroulement du stage

1. Formation

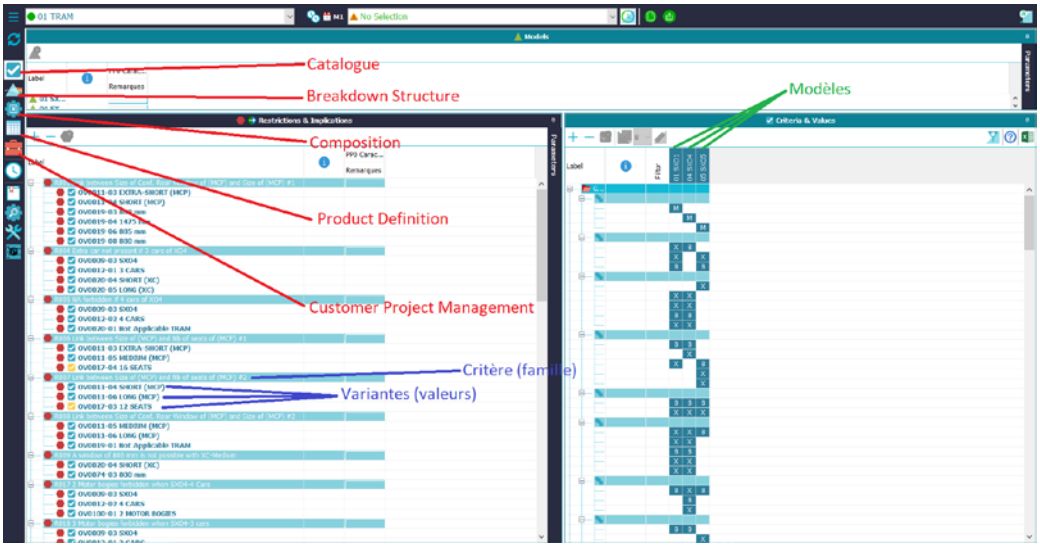
a. Les objectifs de PLEIADE

La première étape de mon stage s'est focalisée sur la compréhension des objectifs du Product Line Engineering (aussi connu comme : Product Family Engineering) à travers la lecture de différents articles (disponibles dans les sources). Les points clés que j'ai pu isoler sont les suivants :

- Nous ne considérons plus un système mécanique comme un ensemble de pièces mais comme un ensemble d'interactions entre elles,
- Le PLE facilite la gestion des produits similaires et la gestion des customisations,
- Le PLE permet une optimisation des coûts et des délais de production,
- Le PLE partage des instances des produits (avantage issu du Product Lifecycle Management) permettant une meilleure coordination en bureau d'étude.
- Le PLE permet aussi de poser un cadre à la customisation : en limitant l'explosion des combinatoires liée à l'ensemble des choix laissés au client, en posant des contraintes techniques (souvent des restrictions) ou des contraintes commerciales (souvent représentées en implication ou en modèle)

Il m'a fallu me former à l'outil de l'entreprise, nommé PLEIADE, à l'aide de Base de données et d'exercices décrits dans la partie formation du serveur de document partagé de l'entreprise. Cette étape m'a permis de me familiariser aux différentes sections de l'application telles que :

- Catalog : permet de définir la variabilité du produit – on y introduit les contraintes des modèles,
- Breakdown Structure : permet d'observer le produit (perspective du produit) suivant les différentes étapes du cycle de vie du produit (Perspective bureau d'étude, production, après-vente),
- Composition : se focalise sur les pièces mécaniques nécessaires à la réalisation,
- Product Definition : permet de composer un produit (c'est ici que se déroule la propagation). Les produits sont contraints par le catalogue,
- Customer Project Management : permet de suivre les commandes.



Capture d'écran depuis l'interface PLEIADE du catalogue.

b. Les outils de gestion de projets

Enfin, après avoir compris l'intérêt industriel de ces outils, je me suis familiarisé avec des outils permettant de mener à bien les projets de développement :

- L'utilisation de SVN afin de faciliter le partage de codes à l'aide de la fonctionnalité de mise à jour 'update' (téléchargement du code généré par l'équipe de développement) et la fonctionnalité de 'commit' (permettant de charger son propre code sur le serveur).
- L'utilisation de Jira afin de faire le lien entre les équipes fonctionnelles et les développeurs mais aussi pour les développeurs entre eux. Les tickets Jira sont utilisés pour lister les tâches à effectuer.

Quelques bases sur la manipulation SQL Workbench ont été nécessaires afin de récupérer les bases de données des clients pour pouvoir tester mes algorithmes en situation « réelle ». Ces bases de données sont partagées sur un serveur Amazon Web Services.

Je me suis également familiarisé avec la programmation orientée objet et plus particulièrement sur Java à l'aide d'un cours en ligne sur 'Open Classroom'. Lors de la rédaction d'algorithmes, il m'a fallu tester au fur et à mesure la cohérence des actions réalisées par ces derniers. Pour cela, j'ai utilisé des « tests unitaires » à l'aide de JUnit.

Finalement, j'ai complété ma formation technique par des révisions en mathématiques : en théorie des ensembles, théorie des graphes et en recherche opérationnelle afin d'avoir une compréhension la plus fine possible du traitement fait par le moteur combinatoire 3C.

c. Les concepts techniques du moteur 3C

Remarque :
Dans le glossaire est rassemblé l'ensemble des définitions permettant la compréhension des concepts abordés dans ce rapport de stage.

i. Formalisme des cas d'emplois

Décrivons quelques règles mathématiques propres aux Cas d'Emplois. D'un point de vu formalisme mathématique, les propriétés des CE sont proches des propriétés et des opérations utilisées en théorie des ensembles.

Propriétés mathématiques

Propriété 1 : l'intersection de deux cas d'emplois est un cas d'emploi. Cette intersection peut être vide.
$CE1 \cap CE2 = CE$

Propriété 2 : la soustraction de deux cas d'emploi est un (ou plusieurs) cas d'emplois.
$(CE1 - CE2) = CE1 - (CE1 \cap CE2)$

Propriété 3 : l'union de deux cas d'emploi est un (ou plusieurs) cas d'emploi.
CE1∪CE2= {CE} = CE1∪ (CE2-CE1) et CE1∪CE2= {CE1, CE2} ssi CE1∩CE2=∅

ii. Opération sur les cas d'emplois

Dans les exemples suivants, nous considérons le référentiel et les restrictions suivantes :

A(a1, a2), B(b1, b2), C(c1, c2, c3)
R1 = (a1, b1)
R2 = (a1, b2, c1, c2)
R3 = (a1, b2)

Deux fonctions permettent de traduire une liste de valeurs en un tableau de bit :

– D'une part l'encodage en 3C :
completeProjectAndCode(List<Valeur>)
Ex : completeProjectAndCode(R1) = 10 10 111

– D'autre part le décodage pour l'applicatif :
decode(List<Valeur>)
Ex : decode(01 11 010) = [a2, b1, b2, c2]

L'intersection de deux CE : en 3C, la méthode and permet d'intersecter deux Cas d'Emploi.
Exemple : encode(R1).and(encode(R2)) = (10 10 000).and(10 01 110) = (10 00 000)

L'union de deux CE : en 3C, la méthode or permet d'intersecter deux Cas d'Emploi. (Cette opération ne correspond pas exactement à la propriété 3).
Exemple : encode(R1).or(encode(R2)) = (10 10 000).or(10 01 110) = (10 11 110)

Ou exclusif de deux CE : en 3C, la méthode xor permet d'effectuer le ou exclusif logique de deux Cas d'Emploi.
Exemple : encode(R1).xor(encode(R2)) = (10 10 000).xor(10 01 110) = (00 11 110)

Le masque : le masque est un Cas d'Emploi contenant seulement un critère.
Exemple : Mask(A) = (11 00 000)

iii. La propagation

Ces opérations sont nécessaires pour traiter les CE au cours de la propagation dans le moteur. Néanmoins, il serait intéressant de comprendre la propagation à l'aide d'un exemple imagé.

Exemple

Comprendre la propagation parfaite à l'aide d'une table de Karnaugh :
Considérons le référentiel et les restrictions suivantes :
A(a1, a2)
B(b1, b2, b3)
C(c1, c2, c3)
R1 = (a2, c1)
R2 = (a2, b1, c2)
R3 = (b1, c3)

Cette situation peut être traduite dans la table de Karnaugh suivante :

	a1			a2		
	b1	b2	b3	b1	b2	b3
c1						
c2						
c3						

Toutes les cases non barrées sont les combinaisons possibles. Donc en sélectionnant a2, on peut conclure que c1 et b1 sont interdits car aucune combinaison n'autorise simultanément a2 et b1 d'une part et a2 et c1 d'autre part.

2. Réalisations techniques

Dans la suite du rapport, j'évoquerai les différents algorithmes développés pour enrichir le moteur du logiciel. J'y détaillerai l'utilité et le besoin de son implémentation, je présenterai une forme simplifiée en pseudo-code ainsi que les cas limites et les améliorations. J'appuierai mon propos à l'aide d'exemples afin de faciliter la compréhension des concepts abordé.

a. Transformation d'une implication en paquet de restrictions

La possibilité de créer des restrictions et des modèles n'est pas suffisante pour rendre compte de l'ensemble des situations lors de la configuration d'un produit.

La restriction s'applique généralement entre un ensemble de valeurs cohérentes qui sont incompatible d'un point de vue technique. Il n'est par exemple pas possible de combiner une jante 16 pouces avec un pneu 15 pouces.
Les modèles permettent de rassembler un ensemble de valeur cohérente entre elles. Par exemple, un modèle de voiture économique autoriserait seulement le choix entre des moteurs d'une faible puissance.
Les implications amènent une dimension plus commerciale et législative dans la définition d'un produit. Ainsi, lorsqu'un utilisateur choisi un ensemble de valeur, un autre ensemble de valeurs sont déduit. Par exemple, si l'utilisateur sélectionne les sièges enfants et les écrans à l'arrière, un ensemble d'options serait déduit faisant parti du pack « famille ».
Fondamentalement, une implication est équivalente à une interdiction : $A => B \Leftrightarrow \neg A \vee B$. Il est donc en théorie possible sur PLEIADE de créer un ensemble de restrictions pour simuler une implication. Cependant, cette solution n'est pas pérenne. Par exemple,

Soit le référentiel
A(a1, a2) B(b1, b2) et l'implication a1 => b1.
Cette implication est équivalente à une restriction R = a1, b2. Mais s'il y a création d'une valeur b3 au sein du critère B, la restriction ne sera plus équivalente à l'implication.
Néanmoins, cette propriété d'équivalence va être utilisée afin de facilité le traitement algorithmique de la propagation, les implications sont considérées comme un ensemble de restrictions. Il nous suffit alors de prendre l'opposé de cette affirmation afin d'obtenir les restrictions équivalentes à cette implication, soit $\neg B \wedge A$.

Soit l'implication a1, b1 => c2, d3, e4
Liste des antécédents : [a1, b1]
Liste des conséquents : [c2, d3, e4]
Le cardinal de la liste des conséquents est de trois. Cette implication génèrera donc trois restrictions.

Algorithme en pseudo de code de traitement des implications pour les traduires en restrictions dans le moteur 3C.

Algorithme en pseudo code

L'implémentation de cet algorithme en 3C est la suivante (pour une seule implication)
Données :
List<valeur> antecedentList
List<valeur> consequentList
List<critère> referentiel
Déclaration des variables :
Set<Critere> set
List<List<valeur>> restrictions
(1)
Pour valeur de consequentList
    set.ajouter(valeur.obtenirCritère)
Fin Pour
(2)
Implication = encode(consequentList)
(3)
Pour critère de set
    Implication = implication.and(critère.mask) (3.1)
    Implication = implication.xor(critère.mask) (3.2)
    List<valeur> listTraité = implication.decode (3.3)
    listTraité.ajouter(antecedentList) (3.4)
    restrictions.ajouter(listTraité) (3.5)
Fin Pour

Exemple d'application de l'algorithme

Exemple d'application de l'algorithme en 3C :
Soit le référentiel A(a1, a2), B(b1, b2), C(c1, c2, c3), D(d1, d2)
Considérons l'implication a1, b1 => c1, d1
On a :
antecedentList = [a1, b1]
consequentList = [c1, d1]
Etape (1) du calcul, nous aurons :
set = [C, D] (la méthode obtenirCritère permet de récupérer le critère de la valeur)
Etape (2)
Implication = (00 00 100 100)
Décrivons les étapes pour les deux passages dans la boucle (3)
Etape (3)
1er itération : critère C
(3.1) Implication = (00 00 100 100).and(00 00 111 000) = (00 00 100 000)
(3.2) Implication = (00 00 100 000).xor(00 00 111 000) = (00 00 011 000)
(3.3) listTraité = [c2, c3]
(3.4) listTraité.ajouter([a1, b1])
(3.5) restrictions.ajouter([a1, b1, c2, c3])

2nde itération :
(3.5) restrictions.ajouter([a1, b1, d2])
On obtient donc deux restrictions à partir de l'implication a1, b1 => c1, d1 :
R1 = a1, b1, c2, c3
R2 = a1, b1, d2

De manière générale, une implication générera une restriction par critère différent présent dans la liste des conséquents.

b. Transformation des modèles en un ensemble de restriction

Un modèle dans le contexte de PLEIADE définit un ensemble de valeur cohérente entre elles et autorise au moins une valeur par critère (un modèle sport sera par exemple équipé d'un moteur plus puissant et d'un système de freinage performant). Le modèle peut aussi être vu comme un objet interdisant un certain nombre de valeurs.

D'un point de vue technique, un modèle m1 est considéré comme une valeur au sein d'une famille M(m1, m2,..., mn). Un modèle peut donc être considéré comme un ensemble de restrictions. Plus particulièrement, un modèle génèrera donc un maximum d'une restriction par famille de valeurs.

Exemple :
Soit le référentiel : A(a1, a2, a3), B(b1, b2), C(c1, c2)
Soit le modèle M1 = [m1, a1, b1, c1, c2]
On peut remarquer que le modèle possède une valeur qui lui est propre : m1.
A partir de ce modèle on peut créer les restrictions suivantes :
R1 = m1, a2, a3
R2 = m1, b2
Il n'y a pas de restriction entre le modèle M1 et les valeurs du critères C car toutes les valeurs du critère C sont disponibles dans le modèle M1.

Cette propriété d'équivalence entre un critère et un ensemble de restriction est exploitée afin de facilité les traitements dans la propagation.

Néanmoins, nous verrons dans la partie sur l'implémentation de l'algorithme de propagation standard que ce choix sera remis en question.

c. Fusion (« gloups ») des Cas d'Emploi

Quand le résultat de l'union entre deux Cas d'Emploi n'est pas un ensemble de Cas d'Emploi mais un seul Cas d'Emploi, c'est un gloups. Cela arrive quand CE1 et CE2 ne diffère que d'un seul critère. Cette fonctionnalité est utilisée comme outil dans le moteur mais n'est pas accessible dans l'application PLEIADE.

Il pourrait être intéressant d'intégrer cette fonctionnalité afin de réduire le nombre de restric-

tions dans un produit. Mais il n'est parfois pas souhaitable de fusionner les restrictions (par exemple une restriction d'origine commerciale et l'autre d'origine technique). De plus, une restriction possède une description qui lui est propre. En fusionnant deux restrictions, il faudrait alors conserver seulement une de ces deux descriptions.
Ce serait donc une fonctionnalité à manipuler avec précaution, au risque de perdre des informations sur le produit.

Je ne détaillerai pas chacun des concepts techniques mis en place en 3C pour fusionner les cas d'emplois. Cependant, deux solutions sont déjà codées :

- Une première méthode traite des ensembles (listes) de Cas d'Emploi permet de réduire la quantité des cas d'emplois. Cette solution est la moins performante du point de vu complexité temporelle.

- Une classe permet de fusionner les Cas d'Emploi plus efficacement, en utilisant une structure d'arbre. Cependant, si les structures de type graphe permettent généralement une meilleure efficacité temporelle, elles ont une moins bonne complexité en espace. En pratique nous n'avons jamais eu de problèmes liés à la mémoire, même en fusionnant un grand nombre de restrictions (environ 60.000 au maximum)

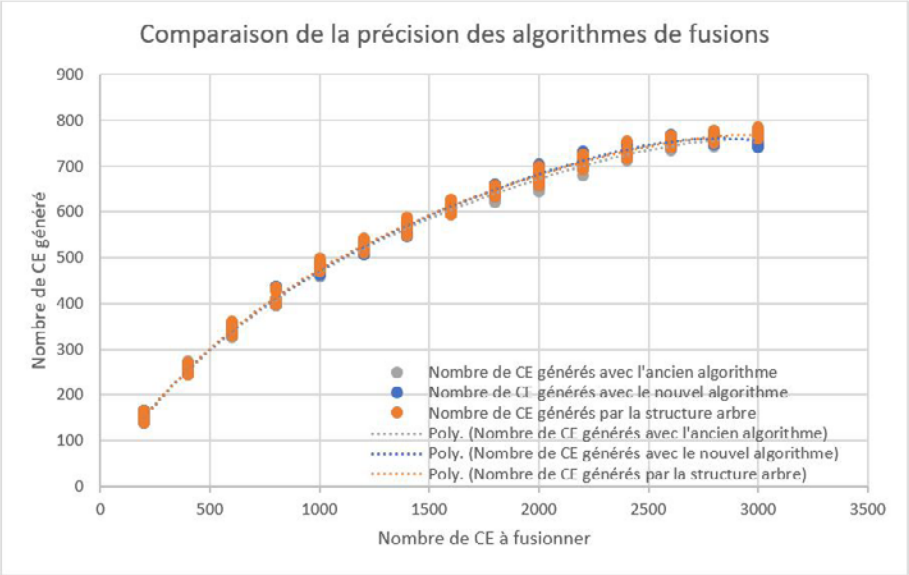
Exemple :
CE1 = (a1)(b1)(c1c2)
CE2 = (a1)(b2)(c1c2)
CE1 GLOUPS CE2 = a1(b1b2)(c1c2)

Exemple d'application de la méthode traitant des listes :
compact([(10 10 111], [10 01 111]) = (10 11 111)

J'ai développé une deuxième méthode plus performante en ajoutant un pré-traitement regroupant les Cas d'Emplois par critère afin de ne pas tenter de fusionner chacun des cas d'emplois avec tout les autres.

Exemple :
R1 = a1, b1, c1
R2 = a1, b1, c2
R3 = c3, e2
Après le pré-traitement, on obtiendrait deux listes :
([a1, b1, c1] [a1, b1, c2]) et ([c3, e2])
On ne tenterait alors pas de fusionner les restrictions suivantes : R1 et R3 ainsi que R2 et R3.

Nous avons cherché à quantifier le nombre de Cas d'Emploi générés à partir de ces trois options qui s'offre à nous. Plus un algorithme est efficace, plus le nombre de cas d'emploi qu'il génère sera faible. Le graphique suivant nous indique que les trois méthodes nous donne un nombre équivalent de fusions.

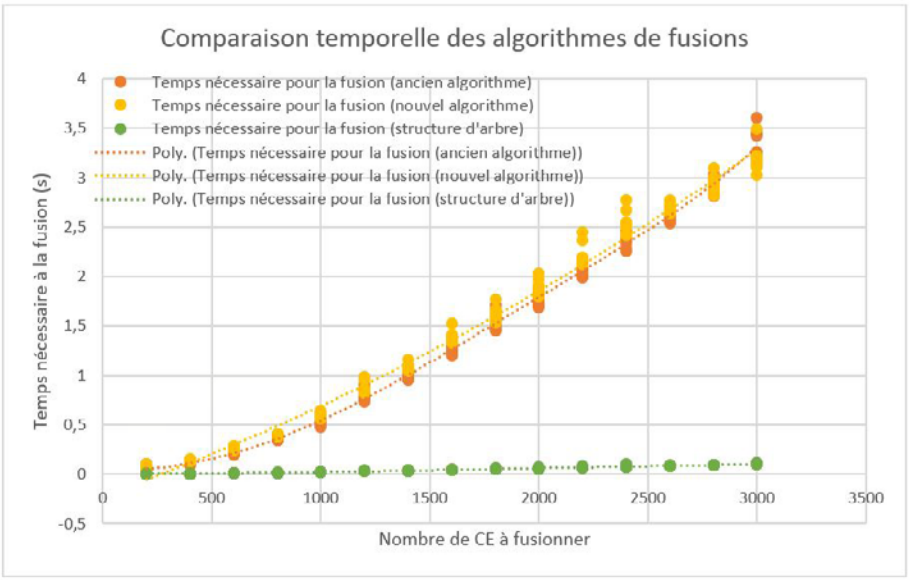


En abscisse, on peut lire le nombre de restrictions à fusionner et en ordonnée le nombre de restrictions restantes après fusions.

Nous ne pouvons pas juger de la performance des différentes méthodes de fusions sur le critère de la qualité.

Il semble néanmoins intéressant de remarquer que la quantité de Cas d'Emploi générée après fusion n'a pas une relation linéaire avec la quantité de Cas d'Emploi à fusionner.

Pour trancher sur l'utilisation préférentielle d'une méthode, nous avons comparer la rapidité de fusions des Cas d'Emploi.



Temps nécessaire (en seconde) pour la fusion en fonction du nombre de Cas d'Emploi

Il est clair sur ce graphique que l'option la plus intéressante pour la fusion de Cas d'Emploi est la méthode exploitant une structure d'arbre. Ce sera donc cette dernière qui sera conservée.

On peut aussi remarquer que la nouvelle solution de fusion en regroupant les Cas d'Emplois par critère dans un pré-traitement ne semble pas être plus performante que la précédente essayant

chacune des combinaisons. (Nous n'avons pas pousser les recherches mais il semble quand même que cette solution soit plus efficace à des nombres de Cas d'Emploi supérieur à 3000).

Remarque : aucune de ces trois méthodes ne permet d'obtenir la solution « idéale ». En effet, en fonction de l'ordre dans lequel nous fusionnons les Cas d'Emploi, nous n'obtenons pas les mêmes résultats. Et obtenir le résultat optimal est un problème NP-complet, donc très long à calculer.

Exemple :
Soit le référentiel A(a1, a2) B(b1, b2) C(c1, c2, c3)
R1 = a1, b1, c1
R2 = a2, b1, c1
R3 = b1, c2
R4 = a2, b1, c2
1ère possibilité :
On commence par fusionner R1 et R2, on obtient R5 = a1, a2, b1, c1 = b1, c1 (car vrai pour tout a)
On fusionne R5 avec R3, on obtient R6 = b1, c1, c2
A la fin de l'algorithme on a donc deux restrictions restantes :
R6 = b1, c1, c2
R4 = a2, b1, c2
Autre possibilité :
On commence par fusionner R2 et R4, on obtient R7 = a2, b1, c1, c2
On ne peut plus rien fusionner. A la fin de l'algorithme on a donc trois restrictions restantes :
R1 = a1, b1, c1
R3 = b1, c2
R7 = a2, b1, c1, c2

d. Propagation « standard »

Certaines bases de données Volvo ou Alstom possède une combinatoire trop importante pour pouvoir être traité dans des temps raisonnable (d'une à cinq secondes).
Le choix a donc été fait d'abandonner le caractère parfait de la propagation et d'implémenter un algorithme de complexité temporelle polynomiale mais dont le Cas d'Emploi renvoyé n'a aucune assurance d'être vrai. Plus précisément, les valeurs interdites sont réellement interdites mais celles encore sélectionnables pourrait ne pas être compatible avec l'ensemble des valeurs sélectionnées.
Partant de ce postulat, l'algorithme suivant a été pensé :

Algorithme en pseudo code

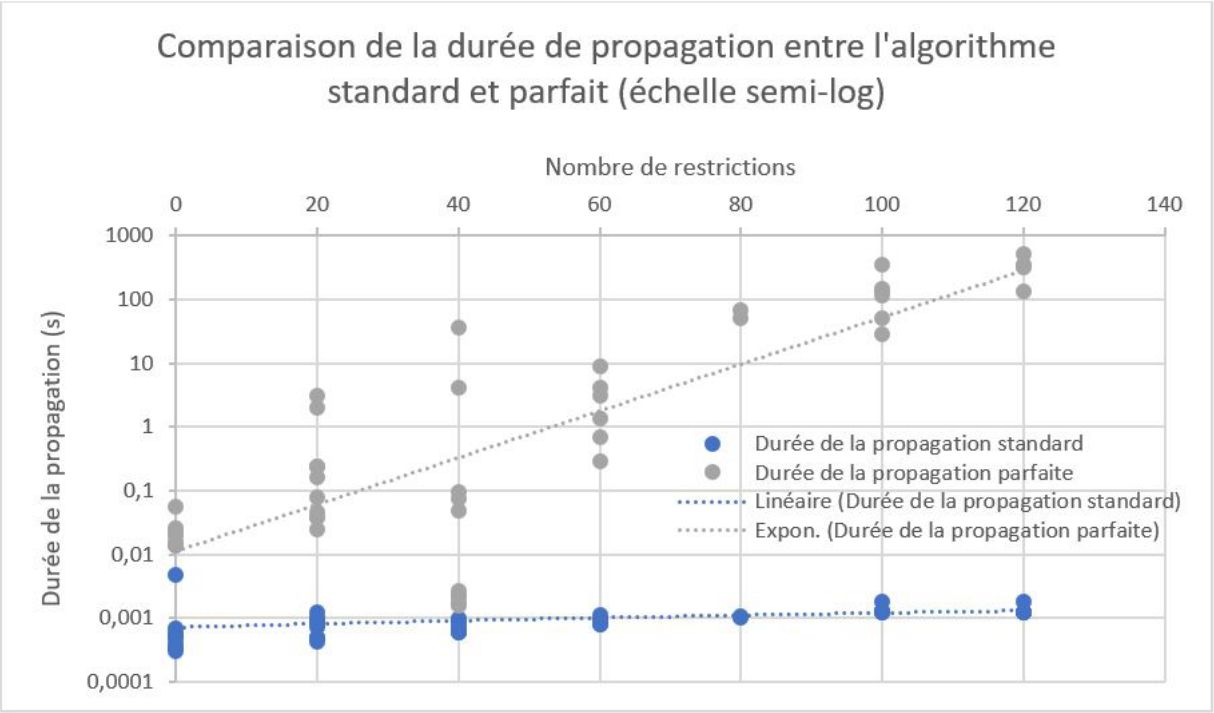
L'algorithme en pseudo-code :
Données :
List[valeurs] sélections
List[List[valeur]] restrictions
List[critère] référentiel
Initialisation des variables :
List[valeur] Interdictions

```
Pour sélection de Sélections :
        Pour restriction de Restrictions :
                Restriction.remove(selection) (1)
                Si toutes les variantes de la restriction sont dans le même critère :
                        Interdictions.ajouter(restriction.getElement()) (2)
                Fin Si
                Si toutes les valeurs d'un critère sont interdites sauf une cette valeur est
déduite
                        Sélections.ajout(valeur) (3)
                Fin Si
        Fin Pour
Fin Pour
```

Exemple d'application de l'algorithme

Exemple : Mettons-nous dans une situation d'application de l'algorithme
Soit le référentiel :
A(a1, a2, a3)
B(b1, b2)
C(c1, c2)
R1 = a1, b1, c1
R2 = a1, b2
On sélectionne a1.
1ère itération sur les restrictions :
(1)
R1 – {a1} = {a1, b1, c1} – {a1} = {b1, c1}
R2 – {a1} = {a1, b2} – {a1} = {b2}
(2)
R2 contient seulement une valeur, b1 est donc interdite.
Interdictions = {b1}
(3)
Le critère B contient les valeurs {b1, b2}. Or b1 est interdit. Donc le critère B n'a plus qu'une valeur possible : b2. Cette valeur est déduite.
2nd itération sur les restrictions :
(1)
R1 – {a1} = {b1, c1} – {b1} = {c1}
(2)
R1 contient seulement une valeur, c1 est donc interdite.
Interdictions = {b1, c1}
(3)
Le critère C contient les valeurs {c1, c2}. Or c1 est interdit. Donc le critère C n'a pu qu'une valeur possible : c2. Cette valeur est déduite.
On a donc pu déduire à partir de la sélection de a1 que le seul Cas d'Emploi respectant le problème est (a1, b2, c2).

Afin de vérifier la performance de ce nouvel algorithme, nous avons comparé la vitesse de propagation entre l'algorithme de propagation parfait et l'algorithme de propagation polynomial :



Nous pouvons voir d'une part que l'algorithme semble être polynomial car il n'y a pas une « explosion » soudaine du temps de calcul lié au nombre de restriction (contrairement à l'algorithme de propagation parfait, ou l'on peut clairement voir le coude de l'exponentielle se dessiner). Néanmoins, on ne peut pas assurer le caractère linéaire de la propagation standard lorsque le nombre de restrictions devient important (>500, voir la prochaine figure).
Il nous a néanmoins été nécessaire de comparer la « qualité » de la propagation entre l'algorithme de propagation standard et parfaite.
Au cours de nos premiers tests sur l'application, nous avons remarqué qu'une « mauvaise » propagation induit qu'après avoir sélectionné une variante qui apparait sélectionnable mais qui est en réalité interdite, l'ensemble des autres variantes du cas d'emplois deviennent interdite. En effet, le Cas d'Emploi devient faux dans le traitement 3C lorsqu'il n'y a plus au moins une valeur valide par critère.

<u>Quelques explications sur les limites de cet algorithme :</u>
De prime abord, il est difficile de comprendre pourquoi cet algorithme ne renvoie pas un résultat précis à chaque fois.
Considérons l'exemple suivant (déjà traité dans la partie Concepts techniques du moteur 3C -> La propagation).
Considérons le référentiel et les restrictions suivantes :
A(a1, a2, a3)
B(b1, b2)
C(c1, c2)
R1 = (a2, c1)
R2 = (a2, b1, c2)
R3 = (b1, c3)
Nous avons pu montrer que lorsqu'on sélectionne a1, les valeurs b1 et c1 sont interdites. En utilisant l'algorithme de propagation standard, on trouve seulement que c1 est interdit.
Phase d'amélioration de l'algorithme :

Une des premières améliorations que l'on peut mettre en place, est la fusion des restrictions réduites après chaque passage dans la boucle.

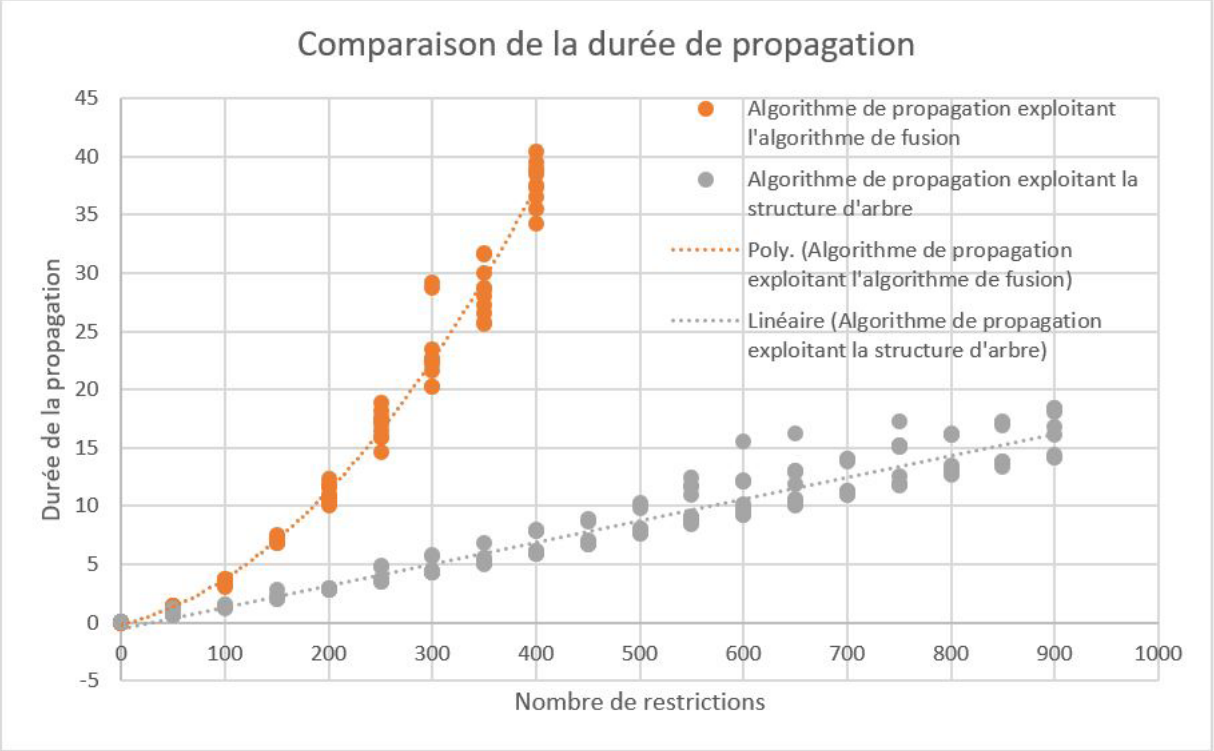
Par exemple, prenons cette situation :

Soit le référentiel :
A(a1, a2, a3)
B(b1, b2, b3)
C(c1, c2)
Soit les restrictions suivantes :
R1 = a1b1c1
R2 = b2b3c1
On sélectionne a1.
L'exécution de l'algorithme nous donnes :
Suppression de la sélection dans les restrictions :
R1 = b1c1
R2 = b2b3c1
A cette étape, nous ne serions pas capables normalement de conclure. Cependant, ces deux CE sont additionnables et peuvent être réduis :
R1 + R2 = b1c1 + b2b3c1 = b1b2b3c1
Donc c1 interdit pour tout b. Donc c1 interdit. Donc c2 déduit.
En situation réelle, c'est-à-dire sur le jeu de données test, pour quantifier l'amélioration en termes de précision de l'algorithme, on a lancé plusieurs sélections aléatoires.

Nous avons pris pour condition de test :
- Le nombre minimal de restrictions = 1000
- Le nombre maximal de restrictions = 2000
- Répété 10 fois
- Un pas de 200

Après analyse des données, on obtient les résultats suivants :
- Nous avons pu quantifier la différence entre les Cas d'Emploi issus de l'algorithme avant modification et l'algorithme après l'amélioration. Nous obtenons une moyenne de 0.2% de différence. Cet écart semble faible, mais l'ajout de la fusion à chaque étape ne diminue pas la performance de l'algorithme (en particulier en utilisant la structure d'arbre pour permettre une fusion très rapide des Cas d'Emploi).
- Nous avons obtenu des Cas d'Emploi nuls, symptomatique d'une valeur semblant être sélectionnable mais en réalité interdite. Et dans les tests que nous avons effectués, lorsqu'un CE est faux sur la première implémentation de l'algorithme, il est aussi faux sur l'amélioration que nous avons effectuée. On peut expliquer cela par la faible différence entre les deux cas d'emploi renvoyé (environ 0.2%).
- Finalement, il semble que le nombre d'erreur augmente avec le nombre de restrictions (les erreurs s'accumulent avec le nombre grandissant de restrictions). Alors qu'il est à moins de 1% pour des restrictions allant jusqu'à 1800, l'apport de l'amélioration semble être de plus en plus important avec le nombre de restrictions (différences d'environ 0.8% à partir de 2000 restrictions).

Nous avons comparé les performances de l'algorithme en fonction de l'utilisation d'arbre ou de la méthode de fusion exploitant des listes.



L'efficacité de l'algorithme en utilisant la structure d'arbre pour la fusion est, comme nous pouvions nous y attendre, beaucoup plus importante. Il est possible de lire la régression en fonction de la fusion. Il est étonnant de remarquer que l'utilisation de l'arbre semble demander un temps linéaire de calcul (tout du moins à ces échelles) alors que le temps de calcul à l'aide la fusion exploitant des listes semble être de complexité quadratique.

<u>Alternative en utilisant la propagation modèle par modèle :</u>
Comme nous avons pu le voir précédemment, les modèles vont être traités sous forme de restrictions. Nous avons aussi pu voir que le taux d'erreur semble augmenté avec le nombre de restrictions et qu'un modèle à lui seul peut créer jusqu'à n restrictions avec n égal au nombre de critères sur le produit.

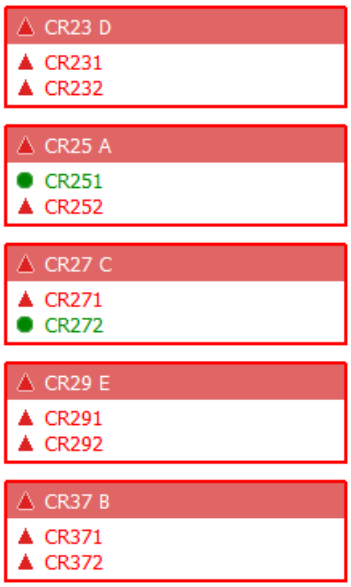
Pour éviter l'ajout de restrictions et donc l'augmentation des chances d'obtenir une erreur, nous avons ajouté un mode de propagation standard modèle par modèle. Pour cela, nous appliquons la propagation modèle par modèle et fusionnons ces résultats à l'aide d'un OU logique pour obtenir les valeurs interdites, sélectionnable ou déduite sur le produit. Un autre avantage de cette propagation est que le fait de considérer les restrictions au sein de chaque modèle permet de les simplifier.

e. Propagation standard en mode « multi-spécification »

Le mode multi-spécification permet à l'utilisateur de choisir plusieurs valeurs dans une même famille. Cette fonctionnalité permet une plus grande souplesse dans la définition d'un produit. Par exemple, un modèle de voiture propose trois types de phares : des phares à LED, des phares au Xénon ou des phares halogènes. Si l'utilisateur ne souhaite pas de phare xénon mais n'attache pas d'importance aux autres types, il peut sélectionner les deux à la fois, ce qui lui permet de ne pas restreindre ses choix.

L'algorithme de propagation standard a été adapté dans l'objectif de pouvoir faire plusieurs sélections dans le même critère. Pour cela, nous effectuons une première propagation. Puis nous appliquons une nouvelle propagation pour chaque critère contenant au moins une sélection pour obtenir les valeurs encore autorisées dans ce critère.

Exemple :
Soit le référentiel :
A(a1, a2, a3)
B(b1, b2)
C(c1, c2)
R1 = a1, b1, c1
R2 = a1, b2
On a vu précédemment que la sélection de a1 permettait d'obtenir le Cas d'Emploi seulement composé de (a1, a2, a3, b2, c2).
En multi-spécification, on autorise de sélectionner aussi a2. Or a2 n'intervient dans aucune restriction, donc lorsqu'on sélectionne a2, l'ensemble des valeurs restent disponible. Le Cas d'Emploi après la propagation à partir de de la sélection a2 est donc (a1, a2, a3, b1, b2, c1, c2).
(a1, b2,c2) ∨ (a1, a2, a3, b1, b2, c1, c2) = (a1, a2, a3, b1, b2, c1, c2)
Ce mode doit être manipuler avec précaution. En effet, il peut aboutir à un Cas d'Emploi interdit.
Reprenons notre exemple. Il est tout à fait possible de faire les sélections suivantes :
Sélection de a1 et a2. On obtient alors le CE : (a1, a2, a3, b1, b2, c1, c2)
Sélection de b1. Le CE reste alors (a1, a2, a3, b1, b2, c1, c2).
On désélectionne a2. Or a1 et b1 ne peuvent pas être sélectionné simultanément. Le Cas d'Emploi devient alors aberrant et toutes les familles du référentiel deviennent alors interdites.



Légende : Sur PLEIADE, cette situation entraine l'interdiction de l'ensemble des familles

f. Explication sur la base des statuts

Comme expliqué précédemment, dans des produits faisant appelle à un nombre important de critères, de valeurs, de restrictions, d'implications et de modèle, il est très difficile de répondre à la question « pourquoi cette valeur est-elle interdite ? » ou « pourquoi ce modèle est-il déduit ? ».
De plus, le traitement algorithmique de la propagation parfaite empêche de connaitre l'ordre d'activation des restrictions et donc il n'y a pas possibilité de connaitre leur influence.
Cependant, a posteriori, nous pouvons tout de même connaitre le statut de chacune des valeurs,

ainsi que les informations du produit : son référentiel, ses modèles et ses restrictions. En interrogeant PLEIADE sur le statut d'une valeur, l'algorithme cherche à comprendre ce qui a induit ce statut. Par exemple, une valeur est déduite lorsque toutes les autres valeurs de sa famille sont interdites. Ensuite, l'algorithme va tenter d'expliquer pourquoi ces valeurs sont interdites. Il s'agit donc d'un algorithme récursif permettant de « remonter » l'enchainement des restrictions/ implications/ modèles.
Pour permettre une compréhension aisée du mécanisme de propagation, nous avons fait le choix de rassembler les informations issues de l'algorithme dans un graphe. Chaque nœud représente une valeur et est associée à un nœud restriction / implication / modèle qui explique le statut de cette valeur.
D'un point de vue technique, nous avons choisi d'utiliser les bibliothèques Jgraph et Jgrapht pour la création et l'agencement graphique. Je décris les grandes lignes du traitement algorithmique dans l'encart suivant.

Algorithme en pseudo code

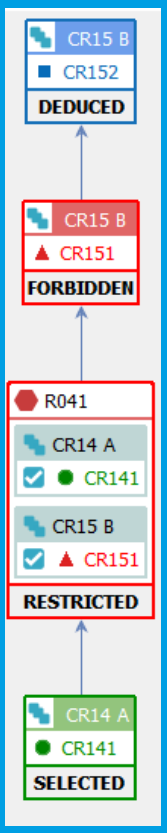
```
Données :
List<critère> Referentiel
List<List<valeurs>> Restrictions
List<List<valeurs>> Models
List<List<valeurs>> implicationAntecedent
List<List<valeurs>> implicationConsequent
List<status(valeur)> Status
Déclaration des variables :
Graph graphExplication
Explication(valeur, dernierNoeud) :
        dernierNoeud = graphExplication.ajouterNoeud(valeur, dernierNoeud)
        STATUS = status(valeur)
        Si (STATUS = déduit)
                Pour les autres valeurs contenues dans le même critère
                        Explication(autresValeurs, dernierNoeud)
                Fin Pour
        Fin Si
        Si (STATUS = interdit)
                Si une autre variante de la famille est sélectionnée
                        Explication(autreVariante, dernierNoeud)
                Else
                        Pour restriction dans Restrictions
                                Si la restriction est active
                                        dernierNoeud = graphExplication.ajouterNoeud(restriction, dernierNoeud)
                                        Pour les autres valeurs de la restriction
                                                Explication(autreValeur, dernierNoeud)
                                        Fin Pour
                                Fin Si
                        Fin Pour
                Fin Si
        Fin Si
Fin Fonction
```

Afin de mieux comprendre ce mécanisme, considérons l'exemple suivant.

Exemple d'explication de restriction

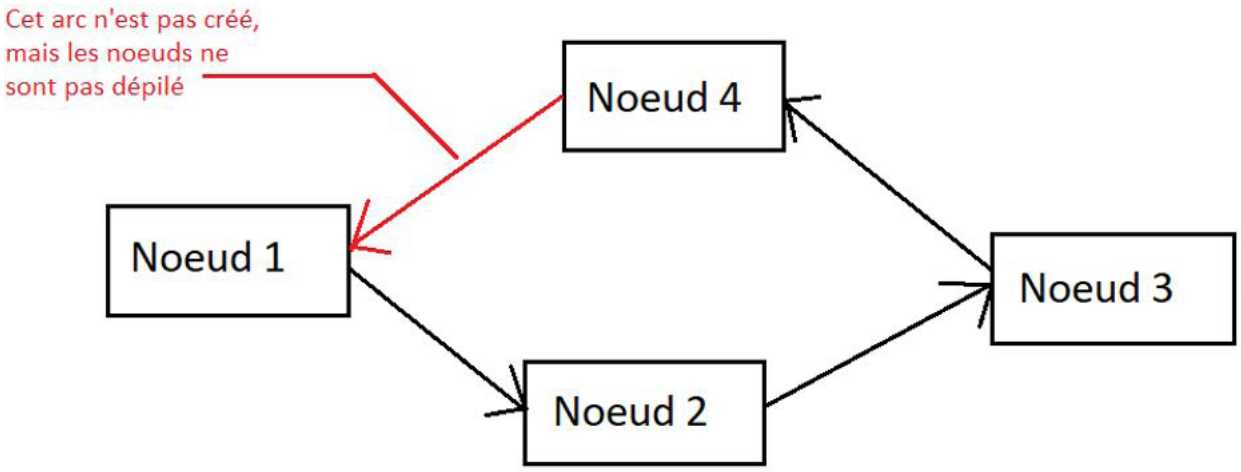
Plaçons-nous dans une situation simple. Soit le référentiel A(a1, a2, a3) B(b1, b2). Soit R = a1, b1.

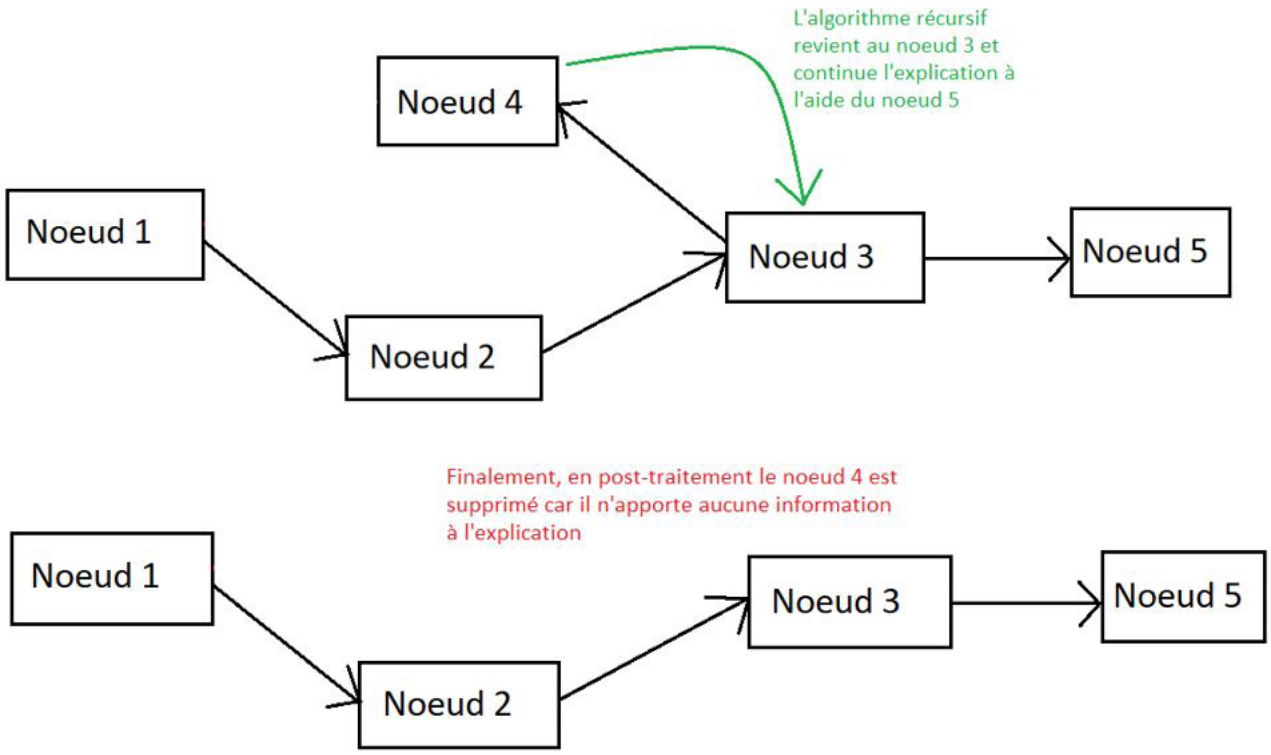
Lorsqu'on sélectionne a1, b1 est interdit et b2 est la dernière valeur non interdite de son critère. Cette dernière est donc déduite. Lorsqu'on lance l'explication sur la valeur b2, nous obtenons le graphe suivant :



Les limites de l'algorithme :

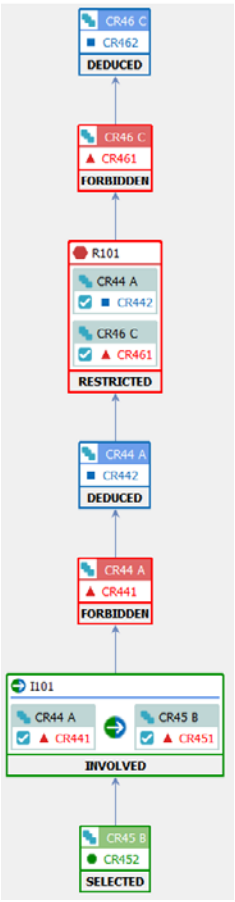
La première version de cet algorithme n'était pas exempte de problème. Une des premiers choses que nous avons dû corriger, est la création de cycles au cours du mécanisme d'explication.
La solution retenue pour résoudre ce problème est d'enregistrer les positions des nœuds lors de l'exécution de l'algorithme puis de dépiler les nœuds aberrant dans un post traitement. Il est en effet impossible de prévoir a priori quel nœud sera dépilé. (cf l'exemple suivant)

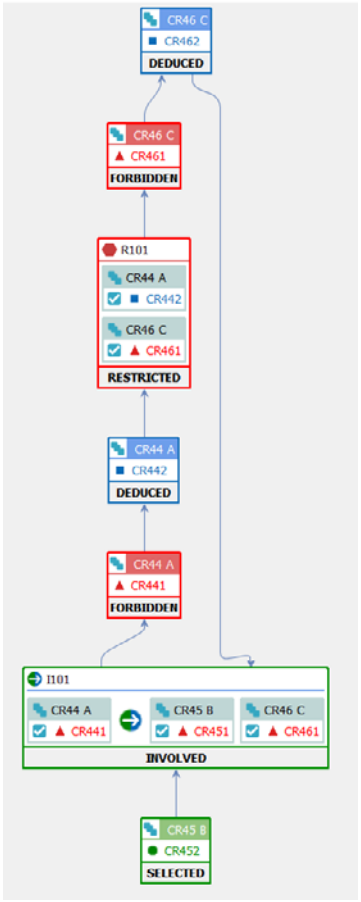




Dans la situation ci-dessus, lors de la création de l'arc entre le nœud 4 et le nœud 1 il était impossible de prédire jusqu'à quel point les nœuds devaient être dépilés. Si le nœud 5, utile à l'explication, était lié au nœud 4, il n'aurait fallu dépiler aucun nœud. S'il avait été sur le nœud 2, il aurait fallu déconstruire jusqu'au nœud 2.

Dans cet exemple, on cherche à expliquer pourquoi la valeur CR462 est déduite





Le graphe de gauche est avant le post-traitement. On s'arrête de dépiler lorsqu'on obtient l'explication de CR462. On se contente donc de supprimer l'arc entre CR462 et l'implication.

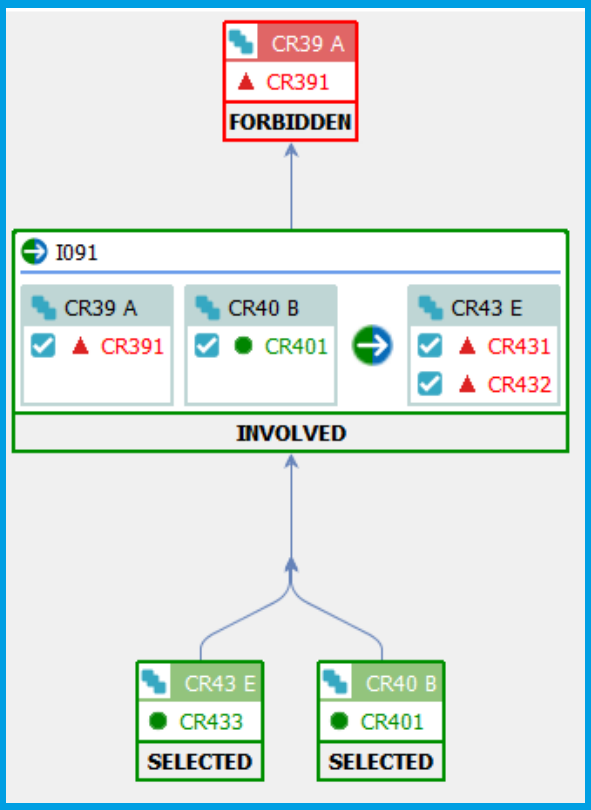
Différentiation des ET/OU :
Lors de la lecture du graphe, il est souvent difficile de savoir si deux arcs sont nécessaires à la conclusion ou si chacun se suffit pour expliquer le statut d'une valeur.
La solution actuelle est de créer un nœud intermédiaire vide dans l'algorithme (il est facile de détecter si chaque arc est nécessaire ou si chacun se suffit).

Exemple d'une contraposée

Soit le référentiel A(a1, a2) B(b1, b2) et E(e1, e2, e3) et l'implication a1, b1 => e1, e2
On sélectionne e3 donc e1 et e2 deviennent interdit.

Or nous avons sélectionné b1. Il est donc impossible de sélectionner a1 car cela violerait l'implication, a1 est donc interdit. Si nous avions seulement sélectionné e3 ou seulement b1, il n'aurait pas été possible de conclure que a1 serait interdit.

On joint donc les deux arcs afin de refléter la nécessité de la sélection de ces deux valeurs pour activer l'implication.



Il faut sélectionner CR401 ET CR433 pour que l'implication s'active (en l'occurrence, il s'agit de sa contraposée)

Finalement, bien souvent dans les produits, on se retrouve dans une situation où l'algorithme n'est pas capable d'expliquer des valeurs interdites car aucune restriction ne s'active directement. Si nous nous nous trouvons dans la situation où il n'y a aucune explication, nous explorons cette piste. Pour cela, on commence par isoler les restrictions qui couvrent partiellement le Cas d'Emploi associé à l'interdiction. Puis on trie au sein de ces restrictions les combinaisons de restrictions qui couvrent totalement le Cas d'Emploi. On explore aussi chaque modèle indépendamment les uns des autres afin de savoir s'ils ont un rôle dans l'explication.

III. Conclusion

Nous avons présenté dans ce rapport un ensemble de techniques basées sur des tableaux binaires pour résoudre des problèmes de propagation de contraintes sur des variables discrètes. Ces techniques permettent d'approcher de façon efficace un problème d'une grande complexité. Cependant et comme je l'ai relevé dans mon rapport, chacun de ces algorithmes a des limites :
La propagation standard : malgré une très grande efficacité temporelle, la propagation reste non parfaite et cela pourrait rendre l'expérience utilisateur moins agréable. Une solution est envisagée afin de coupler les avantages des deux méthodes : commencer la propagation à l'aide de la propagation standard puis envoyer la combinatoire réduite sur la propagation parfaite. Cette approche permettrait de traiter des Cas d'Emploi dont la combinatoire est plus importante tout en restant dans des temps de propagation acceptable (soit une à deux secondes).
Le mécanisme d'explication semble performant et abouti. Nous avons réfléchi à différentes améliorations pour que la compréhension soit plus facile. Finalement, nous avons encore quelques problèmes d'agencement graphique. Dans l'ensemble je suis assez confiant pour l'accueil de cette nouvelle fonctionnalité.

Il me reste 2 mois de stages au moment où j'écris ces lignes, il est difficile donc d'apporter une conclusion ferme et définitive à mon stage. De plus, il nous reste encore des fonctionnalités à implémenter (tel que la génération automatique de combinatoire, de recherche de configurations limites pour des grandeurs mécaniques).

IV. Mise en perspective de mes aspirations professionnelles par rapport à ma formation

Ce stage m'a, en premier lieu, permis de beaucoup progresser en programmation et en algorithmie et de me rendre compte que mes compétences actuelles sont en deçà des exigences dans le milieu professionnel. Ma formation initiale étant en Génie Mécanique, ces thèmes sont assez éloignés des cours que j'ai pu suivre à l'UTT. J'ai fait le choix de diversifier mes compétences car les perspectives professionnelles plus classiques de Génie Mécanique ne répondaient pas à mes attentes.

Néanmoins, des UV tel que MQ02 (« mécanique des milieux continus ») et MQ07 (« mécanique des fluides ») ainsi que l'approfondissement que j'ai suivi lors de mon semestre à l'étranger (« mécanique des fluides avancé ») n'ont pas été une perte de temps car elles m'ont permis de progresser en mathématique plus particulièrement en algèbre. D'autres UV tel que et MT13 m'ont aussi permis de progresser en algorithmie. L'UV LO15 (« approche fonctionnel du PLM ») m'a permis d'avoir les clés afin de comprendre le contexte industriel de l'utilisation d'outils tel que PLEIADE.

Cependant, la chance que nous avons à l'UTT est que nous sommes amenés à pouvoir choisir des UV en dehors de notre branche. J'ai pu prendre certaines UV des branches Génie Industriel (MT15, « valorisation des données pour l'ingénieur » et SY18, « Outils de modélisation et d'évaluations des performances ») et Informatique et Système d'Information (LO02, « Programmation orientée objet ») qui m'ont permis d'avoir les bases nécessaires au bon déroulé de mon stage. Ce système de choix des UV a un autre avantage : il m'a permis à chaque semestre de faire le point sur mes objectifs professionnels.

Finalement, je pense que les transformations de l'industrie vont permettre l'émergence de ce type d'outils permettant une optimisation et une rationalisation des connaissances dans les bureaux d'études. En effet, une quantité importante de données est générée par la recherche et le développement et pourraient être mieux exploité.
Une des pistes d'amélioration de notre formation en génie mécanique pourrait être d'intégrer des UV d'algorithmies, de maths et de programmation ou, à défaut, de guider les étudiants vers des UV de Génie Industriel ou d'Informatique et Système d'information. En effet, l'industrie cherche de plus en plus de profiles pluridisciplinaire par exemple en maintenance prédictive (utilisation de capteurs afin d'optimiser le temps d'usage des pièces dans une machine outils).

Sources utilisées pour ma formation et la rédaction de ce rapport

[1] T. Soininen, I. Niemelä, J. Tiihonen, and R. Sulonen, “Representing configuration knowledge with

weight constraint rules.,” Answer Set Programming, vol. 1, 2001.

[2] Wikipedia, Définition du Product Line Engineering, https://en.wikipedia.org/wiki/Product-family_engineering, 2019

[3] Aerospace Manufacturing And Design, Quel place pour le PLE dans l’industrie, https://www.aerospacemanufacturinganddesign.com/article/what-is-product-line-engineering-ple/, 2019

[4] Open Classroom, Débutez La programmation Avec Java, https://openclassrooms.com/fr/courses/6173501-debutez-la-programmation-avec-java, 2019

[5] Confluence, Wiki d’Entreprise, https://www.atlassian.com/software/confluence, 2020

[6] Site Web Volvo, Exemple de configurateur en ligne, https://www.volvocars.com/fr/configurateur, 2019

Glossaire

Valeur, option, variant, Option (objet 3C) : une valeur est un des choix possibles au sein d'un critère.

Critère, famille, OptionCategory (objet 3C) : groupement de valeurs dont seulement une peut-être sélectionnée.

Modèle : liste de valeurs. Contient au moins une valeur par critère.

3C : moteur de l'application. Traite les combinatoires à l'aide d'algorithme en exploitant la notion de Cas d'Emploi.

Restriction : ensemble de variantes dont la combinaison est interdite. Plus précisément, il s'agit de deux valeurs, ou plus, qui ne peuvent être choisies simultanément.
Ex : Soit R1(a1, b1, c1) appartenant respectivement aux critères A, B, C. Lorsqu'on sélectionne a1, la restriction ne s'active pas. Par contre si a1 et b1 sont sélectionnés (ou déduits), la restriction s'active et c1 est interdit.
Soit R2(a1, a2, b1, c1) appartenant respectivement aux critères A, B, C. Lorsque b1 et c1 sont sélectionné, a1 et a2 deviennent alors interdit

Implication : une implication est constituée de deux listes – une pour l'ensemble des variantes antécédentes et l'autre pour l'ensemble des variantes conséquentes.
Ex : $(a1 \vee a2) \wedge (b1) \Rightarrow (c1 \vee c2) \wedge d1$
On a1, b1 sélectionnés. D1 déduit.
On a a1 sélectionné et (c1 et c2) ou d1 sont interdits. b1 est alors interdit.
On a a1, b1 sélectionnés et c1 ou c2 interdits. d1 est alors déduit.

Cas d'Emploi (Use Case) : Représentation de la combinatoire dans le contexte de 3C
Ex : a1(b1b2) se lit « a1 » du critère A ET b1 OU b2 du critère B

Cas d'emploi élémentaire : il s'agit d'un cas d'emploi dont une seule valeur est possible par critère

Cardinalité : ensemble des cas d'emploi élémentaires dans le référentiel
Ex : Soit le référentiel : A (a1,a2,a3) et B (b1,b2,b3)
Cardinalité = 3*3= 9 cas d'emploi élémentaires

Multi-spécifications : mode qui permet la sélection de plusieurs variantes dans une même famille.

Les statuts :
Sélectionné : valeur sélectionnée par l'utilisateur
Déduit : après propagation, cette valeur est la seule possible au sein d'une famille (critère). Equivalent à une sélection.
Interdit : valeur ne pouvant être sélectionnée au sein d'une famille. Elle peut être interdite pour les raisons suivantes :

- Elle n'est présente dans aucun modèle

- Elle est interdite en raison des restrictions suite à la propagation
- Une autre valeur de sa famille a été déduite par une implication

Sélectionnable : dans un critère « non résolu », plusieurs variantes peuvent encore être sélectionnées.