N° d'ordre : 2233 2005

THÈSE

Présentée

en vue de l'obtention du grade de

docteur de

L'INSTITUT NATIONAL POLYTECHNIQUE DE TOULOUSE

École doctorale : Systèmes

Spécialité : Systèmes Industriels

par

Élise VAREILLES

Conception et approches par propagation de contraintes : contribution à la mise en œuvre d'un outil d'aide interactif

---

Soutenue le 24 juin 2005, devant le jury composé de :

MM.	Frédéric BENHAMOU	Rapporteur
	Alain BERNARD	Rapporteur
	Laurent GENESTE	Examinateur (Président du jury)
	Jean-Pierre NADEAU	Examinateur
	Michel ALDANONDO	Examinateur (Directeur de thèse)
	Khaled HADJ HAMOU	Examinateur (Encadrant de thèse)
	Paul GABORIT	Invité (Encadrant de thèse)
	Philippe DAVID	Invité (PDG de *S&CC*)
	Pascal LAMESLE	Invité

Thèse préparée au Centre de Génie Industriel de l'École des Mines d'Albi-Carmaux

Remerciements

C'est l'esprit qui mène le monde et non l'intelligence. (Antoine de Saint-Exupéry, Carnet)

Je tiens tout d'abord à remercier mes encadrants de thèse sans qui rien n'aurait été possible : M. Michel ALDANONDO, directeur de thèse et MM. Khaled HADJ-HAMOU et Paul GABORIT, encadrants de thèse. Merci à tous les trois pour vos qualités humaines, votre disponibilité, vos conseils avisés et de m'avoir fait confiance durant ces trois années.

J'adresse également mes remerciements aux personnes qui ont accepté de participer à mon jury de thèse :

- M. Frédéric BENHAMOU, professeur au LINA de Nantes et M. Alain BERNARD, professeur à l'IRCCyN de Nantes, qui m'ont fait l'honneur d'être rapporteurs de ma thèse,
- M. Laurent GENESTE, professeur au LGP de Tarbes, pour avoir accepté de présider ce jury et M. Jean-Pierre NADEAU, professeur à l'ENSAM de Bordeaux, pour l'intérêt qu'il a porté à ce travail,
- MM. Pascal Lamesle et Philippe David, respectivement maître-assistant du laboratoire CROMEP de l'EMAC et PDG de la société S&CC, pour avoir accepté de participer à ce jury en tant que membres invités.

Je tiens à remercier toutes les personnes de l'EMAC qui m'ont soutenu et supporté durant cette période. Je pense plus particulièrement aux permanents et aux doctorants du laboratoire Génie Industriel : Isabelle, Carmen, Jacqueline ($\times 2$), Lionel, Jacques, Hervé, Marc-André, Didier, Franck F., Frédéric, Matthieu L., David, Naly, Thomas, Jihed et Jahouer. Merci à tous pour les bons moments passés en votre compagnie.

Une pensée affectueuse à Franck D. et Matthieu D. qui m'ont accompagné tout au long de cette aventure et qui ont toujours répondu présent dans les moments difficiles.

Merci aux amis Castrais, Bordelais, Toulousains et Andorrans pour les week-ends détente en leur compagnie. Promis, à la prochaine rando, j'arrive jusqu'en haut ;).

Enfin, je tiens à remercier ma famille et ma belle-famille pour leur soutien et leur amour sans limite, qui font ce que je suis aujourd'hui. Merci à toi, Fabrice, d'être tout simplement là.

À Lili, Jules et Fabrice avec tout mon amour.

Table des matières

Introduction

Les travaux de recherche présentés dans cette thèse se situent dans une problématique d'aide à la conception interactive de produits et de procédés, exploitant des connaissances formalisées sous forme de contraintes. Ce besoin d'interactivité, nous conduit à exploiter les méthodes de filtrage des contraintes plutôt que les méthodes de résolution. Ces travaux sont sous-tendus par un projet européen[1], nommé *VHT* pour *Virtual Heat Treatment*, visant à mettre au point un outil d'aide à la conception interactif d'opérations de traitement thermique.

Les connaissances à exploiter et à modéliser étant de formes très diverses, nous utilisons différents types de *CSPs* : les *CSPs* discrets (variables symboliques et entières, contraintes discrètes), les *CSPs* continus (variables réelles, contraintes numériques) et les *CSPs* mixtes (variables discrètes et numériques, contraintes mixtes). La structure des problèmes étant d'autre part non figée, nous utilisons les *CSPs* dynamiques pour conditionner la prise en compte de certaines variables et contraintes. Nous avons, d'une part, analysé, sélectionné et adapté à nos besoins les méthodes de filtrage des différents types de *CSPs* utilisés, et, d'autre part, assemblé ces méthodes de filtrage dans un moteur de propagation de contraintes.

Ce mémoire se décompose en sept chapitres. Chacun d'entre eux est illustré par des exemples tirés de notre application :

- Le chapitre 1 présente le contexte de notre problématique et de manière succincte, les raisonnements à base de contraintes.
- Le chapitre 2 présente une typologie des variables et des contraintes utilisées pour construire nos modèles de raisonnement.
- Le chapitre 3 présente les contraintes de type table de compatibilité discrète, continue et mixte ainsi que leurs méthodes de filtrage. Notre apport se restreint aux méthodes de filtrage des contraintes de type table continue.
- Le chapitre 4 se concentre sur les contraintes de type fonction numérique dite « simple ». Plusieurs méthodes de filtrage existent sur ce type de contraintes. Nous les comparons, dans un premier temps, puis exposons la méthode de filtrage par 2B-cohérence retenue. Notre apport se limite à la prise en compte et au filtrage des domaines multi-intervalles.
- Le chapitre 5 présente une méthode de filtrage des contraintes de type fonction numérique dite « complexe ». Leur filtrage passe par leur discrétisation et par leur représentation sous forme de 2^k-arbre. Nous présentons la méthode de génération des 2^k-arbres à partir de la définition syntaxique des contraintes. Notre apport concerne d'une part, l'extension de cette méthode de génération aux contraintes de type fonction numérique dite

[1] projet n° G1RD-CT-2002-00835

« complexe » définies par morceaux et d'autre part, l'amélioration de la prise en compte simultanée de ce type de contraintes par une exploration opportuniste de l'espace de recherche.

– Le chapitre 6 présente les techniques d'adaptation des modèles de connaissances à un problème particulier. Nous présentons les différents types de *CSPs* permettant d'ajuster les modèles et de les structurer. Notre apport se restreint à l'extension des conditions d'activation des variables à l'activation de contraintes et à la mise en place d'une structure hiérarchique à base de groupes d'éléments.

– Le chapitre 7 présente l'application industrielle support de ces travaux. Nous exposons, tout d'abord, l'architecture générale des modèles de connaissances, puis comment sont intégrées les différentes méthodes de filtrage dans le moteur de propagation.

Ces travaux de recherche portent donc sur l'utilisation, l'adaptation et l'assemblage de divers types de *CSPs* pour aider à la conception de produits et de procédés.

Chapitre 1

Cadre général des travaux et problématique

1.1 Introduction

L'activité de conception est un processus complexe et décisif dans le cycle de vie des produits, des services et des procédés. C'est, en effet, à travers elle que les solutions techniques à mettre en œuvre sont déterminées afin de réaliser un objet[1] conforme aux exigences décrites dans un cahier des charges. De nombreuses décisions sont arrêtées durant cette activité. Prendre ces décisions nécessite une forte mobilisation de ressources humaines, techniques et informationnelles, dans le but de justifier que les choix effectués sont les plus adéquats. Remettre en cause l'un de ces choix génère un allongement de l'activité de conception. Il est donc primordial de réaliser des choix judicieux, pour éviter d'itérer et pour converger le plus rapidement vers une solution satisfaisante. Les erreurs de conception ne sont pas les seules responsables du délai important qui lui est consacré. En effet, il s'avère que lors de la conception d'objets nouveaux, le processus de conception est souvent reconduit dans sa totalité sans tenir compte de l'expérience acquise lors des conceptions antérieures. Or, l'exploitation de cette expérience permet de diminuer fortement les délais de conception, puisque certains choix ne sont plus ni à faire, ni à remettre en question. Dans ce contexte, beaucoup d'entreprises souhaitent disposer d'approches et d'outils d'aide à la conception exploitant connaissances et savoir-faire. La mise en place de ces outils passe par différentes étapes telles que la capitalisation des connaissances, leur formalisation et leur exploitation. Plusieurs techniques d'Intelligence Artificielle (*IA*) permettent de représenter les connaissances et de les exploiter dans le cadre d'une problématique d'aide à la conception.

Nous rappellerons, dans la première section, les différents types de conception au travers d'une revue des connaissances auxquelles ils font référence. Puis, nous présenterons, dans la seconde section, deux techniques issues de l'*IA*, les raisonnements à base de cas et les raisonnements à base de contraintes, permettant la formalisation et l'exploitation de connaissances et nous donnerons quelques éléments nous ayant conduit à retenir les approches à base de contraintes.

[1] Nous entendons par objet un produit, un service ou un procédé.

Nous présenterons dans la troisième section les approches à base de contraintes. La quatrième section exposera la problématique industrielle à l'origine de nos travaux, à savoir l'aide à la conception d'un procédé de traitement thermique de trempe. La dernière section situera globalement notre problématique.

1.2 La conception

La conception consiste à définir un objet (produit, service ou procédé) conforme à des exigences décrites dans un cahier des charges (CdC). Elle se décompose en trois étapes principales (Pahl et Beitz, 1996). La première consiste à dégager du cahier des charges les fonctionnalités attendues de l'objet à concevoir. Durant la deuxième étape, les choix de solutions de conception sont effectués afin de réaliser l'objet conforme à sa description fonctionnelle. La dernière étape consiste à évaluer les solutions fournies et à retenir celles qui semblent les plus adéquates. L'activité de conception est un processus itératif qui converge vers une solution satisfaisante. La moindre modification du cahier des charges, ou la moindre remise en question des choix de conception, nécessite la reconduite du processus de conception, cf. figure 1.1. Pour limiter le nombre d'itérations et se concentrer sur les points délicats, l'idée d'exploiter connaissance et savoir-faire au travers d'outils d'aide à la conception semble intéressante.

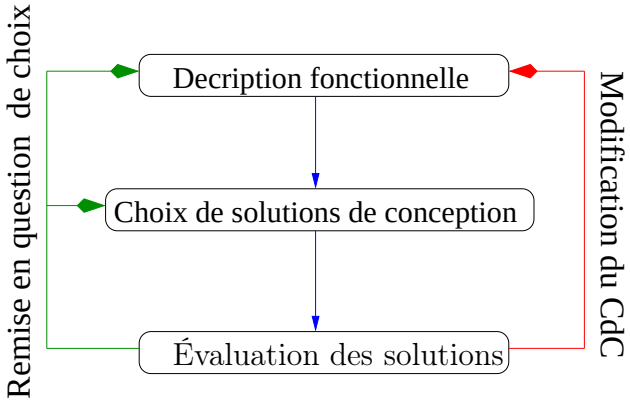


FIG. 1.1 – Processus de conception

L'aide à la conception nécessite la représentation des connaissances disponibles, au sein d'entreprise ou de corps de métier, afin de les exploiter, facilitant ainsi l'élaboration de nouveaux objets. Une typologie des problèmes de conception a été caractérisée par Chandrasekaran (1990). Cette classification fait apparaître trois grandes familles de problèmes de conception définies suivant les sources de connaissances disponibles :

- la première source de connaissances correspondant aux fonctionnalités de l'objet à concevoir est appelée **objectif**,
- la deuxième source de connaissances portant sur l'identification de l'environnement de l'objet et des technologies à déployer pour sa réalisation est désignée par le terme **domaine**,

– la dernière source de connaissances est fortement liée à la culture et au savoir-faire des entreprises. Elle caractérise l'habileté à concevoir une certaine gamme de produits, de services ou de procédés. Celle-ci se définit par le terme de **démarche** de conception.

Nous reprenons cette classification en insistant particulièrement sur le type des connaissances mises en jeu. Nous positionnerons par la suite nos travaux de recherche en fonction des connaissances mises à notre disposition.

Conception routinière

La conception routinière se définit par la disponibilité des trois sources de connaissances à savoir l'objectif, le domaine et la démarche. Nous pouvons parler de conception routinière dès lors que les concepteurs ont une certaine habitude et expérience sur la démarche de conception d'une même famille d'objets. Ce type de conception concerne généralement un objet de complément de gamme, de remplacement ou des améliorations incrémentales.

Conception innovante

Dans la conception innovante, seules deux sources de connaissances sont disponibles : l'objectif et le domaine. C'est-à-dire que l'expression du besoin et les technologies à employer sont le plus souvent identifiées mais les stratégies de conception restent à définir. Tel est le cas pour la mise sur le marché d'un objet correspondant à un besoin exprimé par les clients, mais non encore satisfait.

Conception créative

La conception créative est celle qui dispose du moins de connaissances. L'objet à concevoir est en général nouveau sur le marché, ce qui implique que seul l'objectif est défini. Les technologies à mettre en œuvre sont à identifier et à s'approprier. Les concepteurs n'ayant jamais eu à se confronter à ce type de problème n'ont aucune expérience sur la démarche de conception à déployer.

Synthèse

Nous présentons, dans le tableau 1.1, une synthèse des connaissances disponibles suivant le type de conception considérée.

Tab. 1.1 – Connaissances disponibles suivant le type de conception

Type de conception	Objectif	Domaine	Démarche
Routinière	oui	oui	oui
Innovante	oui	oui	non
Créative	oui	non	non

Les outils d'aide à la conception sont d'autant plus faciles à mettre en œuvre que les trois sources de connaissances sont disponibles. Dans le cadre de nos travaux, nous nous situons au niveau d'une conception routinière de procédés où les trois sources de connaissances sont mises à notre disposition.

1.3 Aide à la conception et contraintes

La mise au point d'outils d'aide à la conception exploitant connaissance et savoir-faire passe par trois phases distinctes qui sont l'extraction, la formalisation et l'exploitation de connaissances. Mais leurs développements ne requièrent pas les mêmes investissements par rapport à ces trois phases. Certains outils nécessitent un travail important en amont, lors des phases d'extraction et de formalisation, d'interprétation des connaissances. D'autres, au contraire, requièrent un travail en aval, lors de la phase d'exploitation, d'analyse des connaissances en termes d'impact sur la solution fournie. Nous présenterons deux techniques issues de l'*IA* permettant la modélisation des connaissances pour l'aide à la conception : les approches à base de cas et celles à base de contraintes. Nous en exposerons les avantages et les inconvénients en termes de mise en œuvre, de formalisation des connaissances et de résultats fournis. Nous présenterons plus en détail l'approche à base de contraintes que nous retenons.

1.3.1 Aide à la conception et connaissances

Plusieurs techniques d'Intelligence Artificielle permettent d'aider à la conception en exploitant de la connaissance. Nous concentrons notre discours sur deux approches particulières : les raisonnements à base de cas et à base de contraintes. Les raisonnements à base de cas s'appuient sur des connaissances implicites regroupées dans des cas, alors que les raisonnements à base de contraintes sur des connaissances explicites et formalisées. Comme ces deux techniques ne reposent pas sur le même type de connaissances, elles ne nécessitent pas les mêmes efforts pour la mise au point d'outils d'aide à la conception. Nous présentons dans ce qui suit ces deux approches, puis nous justifierons notre choix de méthode, porté sur les raisonnements à base de contraintes, en insistant sur les avantages qu'elle apporte dans une problématique d'aide à la conception.

1.3.1.1 Raisonnement à base de cas

Le concept de raisonnement à base de cas ne nécessite pas d'extraction à proprement parler de connaissances. Celles-ci sont regroupées dans les instances du problème appelées « cas ». Chaque cas est caractérisé par un ensemble de valeurs de paramètres qui va permettre de le différencier des autres cas présents dans la base et de le comparer ensuite à un nouveau problème. Une mesure de distance permet de classer les différents cas par rapport à leur similitude avec le problème soumis (Kolodner, 1993). Le résultat fourni correspond à l'ensemble des n cas mesurés les plus proches. Il faut alors sélectionner un ou plusieurs d'entre eux, les analyser et les adapter pour répondre au problème courant. La base est ensuite enrichie par le problème courant adapté et révisé, décrit au travers des paramètres.

Dans les systèmes à base de cas, les connaissances regroupées et noyées dans les cas n'ont pas besoin d'être formalisées pour être exploitées. Seule une description précise des cas est suffisante. Ceci a l'avantage de pouvoir mettre rapidement en place ce type d'outils d'aide à la conception et de les maintenir facilement. Mais ils présentent plusieurs inconvénients. Tout d'abord, l'ensemble des cas doit couvrir l'ensemble des solutions visées. Il faut ensuite fournir un nombre conséquent de cas pour obtenir des résultats significatifs. La mise au point de la

fonction de mesure d'écart est complexe et elle ne garantit pas pour autant l'optimalité[2] des solutions fournies. Enfin, l'analyse et l'adaptation des n cas solutions potentiels nécessitent un effort important de réflexion et un savoir-faire certain.

1.3.1.2 Raisonnement à base de contraintes

Le concept de raisonnement à base de contraintes nécessite un travail considérable d'extraction et d'interprétation des connaissances. La connaissance globale d'un problème est segmentée en fragments élémentaires de connaissance, puis modélisée sous forme de contraintes. Par exemple, nous entendons par fragment élémentaire de connaissance des relations logiques : $(A \wedge B) \vee C$, des expressions mathématiques : $y = x^4 + 3 \times x$, ou encore des domaines de validité : $A \in \{a, b, c\}$. C'est sur ces fragments élémentaires clairement identifiés et validés que va se baser la construction du modèle de connaissances et de raisonnement. Le modèle fondé sur ces briques de connaissances décrit un ensemble de solutions. Lorsqu'un nouveau problème apparaît, il est soumis au modèle de connaissances via un certain nombre de variables. Un raisonnement est alors conduit à travers les fragments de connaissances et réduit les domaines d'autres variables. L'ensemble des solutions au problème posé peut être alors déterminé.

Dans les systèmes à base de contraintes, un gros effort d'abstraction est indispensable pour identifier et formaliser les fragments de connaissances et pour bâtir un modèle cohérent et complet de raisonnement. Ce type d'outil d'aide à la conception est certes plus complexe et plus long à mettre en place, mais une fois le modèle de raisonnement construit et validé, il reste cohérent pour toutes les conceptions supportées par le modèle. D'autre part, les systèmes à base de contraintes ont l'avantage de pouvoir fournir des solutions inédites ou d'établir qu'un problème n'a pas de solution.

1.3.1.3 Approche retenue

L'activité de conception peut être vue comme un processus itératif qui débute, après la description fonctionnelle de l'objet à concevoir, par la réalisation de choix de solutions de conception. La faisabilité de ces choix est ensuite analysée en fonction de moyens techniques disponibles et de leurs répercussions. Il peut s'avérer que des choix de conception aboutissant au respect de fonctionnalités n'aient pas de supports techniques conciliables. Cette sélection de moyens incompatibles entraîne la reconduite du cycle de conception. Pour éviter de choisir des solutions irréalistes, il peut être intéressant, après l'identification des fonctionnalités, de recenser les solutions techniques afin d'éliminer les alternatives de conception ne menant à aucune possibilité de réalisation. Coupler ces aspects fonctionnels et techniques tend à limiter le nombre d'itérations du processus de conception et facilite donc l'atteinte d'une solution satisfaisante en un temps raisonnable.

Les deux approches décrites précédemment permettent de considérer les aspects fonctionnels et techniques simultanément. Contrairement aux approches arborescentes « primaires » qui imposent un ordre (*si entrée = vrai alors* $sortie_1$ *sinon* $sortie_2$), les raisonnements à base de

[2]Ici, l'optimalité représente les cas effectivement les plus proches du problème soumis.

cas et à base de contraintes ne séparent pas les paramètres et les variables d'entrée et de sortie. Les paramètres et les variables peuvent être renseignés dans un ordre indifférent.

Savoir *a priori* qu'une décision prise est cohérente avec les choix précédemment effectués à tout moment du processus de conception est essentiel pour réduire les délais de conception. La garantie de la cohérence des choix limite la remise en question des choix antérieurs. Pour autant, elle n'implique pas que ceux-ci conduisent à une solution satisfaisante ou existante. Les raisonnements à base de cas ne permettent pas de garantir la cohérence des choix. Même si des valeurs de paramètres incohérentes (correspondant éventuellement à des choix) sont fournies au système, celui-ci renverra toujours les n cas mesurés les moins éloignés. Les approches à base de contraintes garantissent quant à elles une certaine cohérence des choix à tout moment du raisonnement, grâce à la conduite d'un raisonnement cohérent reposant sur le respect des contraintes.

Les approches à base de contraintes offrent donc deux intérêts majeurs pour l'aide à la conception :

- la limitation du nombre d'itérations par la considération simultanée des fonctionnalités et des solutions techniques,
- certaines garanties de la cohérence des choix de conception.

1.4 Problèmes de satisfaction de contraintes

Nous présenterons dans cette section les raisonnements à base de contraintes ou problèmes de satisfaction de contraintes. Dans un premier temps, nous définirons formellement les problèmes de satisfaction de contraintes. Puis, nous exposerons succinctement les méthodes permettant la recherche de solutions de ces problèmes et quels sont leurs apports en aide à la conception. La première méthode fournit un ensemble de solutions, nous parlerons alors de résolution ; la seconde indique si des choix de conception conduisent effectivement à une solution, nous parlerons alors de filtrage ou propagation.

1.4.1 Définition d'un problème de satisfaction de contraintes

Les problèmes de satisfaction de contraintes ou *CSPs (Constraint Satisfaction Problems)* permettent de modéliser de la connaissance et de raisonner sur celle-ci afin de trouver l'ensemble des solutions compatibles avec un problème courant. Les premiers problèmes de satisfaction de contraintes ont été définis par Montanari (1974) il y a une trentaine d'année.

Définition 1 : *problèmes de satisfaction de contraintes*

Les problèmes de satisfaction de contraintes sont définis comme un triplet $(\mathbb{V}, \mathbb{D}, \mathbb{C})$ *où :*

- $\mathbb{V} = \{v_1, v_2, ..., v_k\}$ *est un ensemble fini de variables,*
- $\mathbb{D} = \{d_1, d_2, ..., d_k\}$ *est un ensemble fini de domaines de définition des variables,*
- $\mathbb{C} = \{c_1, c_2, ..., c_m\}$ *est un ensemble fini de contraintes portant sur les variables.*

Réaliser un outil d'aide à la conception à partir des concepts de *CSP* revient à traduire les fragments de connaissances élémentaires soit sous forme d'élément unique de $\mathbb{C}$, soit sous forme de plusieurs éléments répartis sur le triplet $(\mathbb{V}, \mathbb{D}, \mathbb{C})$ (Vernat, 2004). C'est le niveau d'abstraction du fragment qui va déterminer si celui-ci est directement utilisable sous la forme d'une contrainte, ou s'il doit être décomposé.

Le modèle de connaissances est alors confondu avec le problème de satisfaction de contraintes. Trouver les solutions d'un problème donné revient à résoudre le problème de satisfaction de contraintes.

Définition 2 : *solution d'un* CSP

Une solution d'un problème de satisfaction de contraintes est une instanciation de toutes les variables respectant toutes les contraintes.

1.4.2 Méthodes de résolution des problèmes de satisfaction de contraintes et aide à la conception

Les méthodes complètes de résolution de *CSP* explorent de manière systématique l'espace de recherche et sont capables de fournir toutes les solutions d'un problème. L'algorithme de recherche de base le plus souvent utilisé est l'algorithme de retour arrière ou *Backtrack* (Golumb et Baumbert, 1965). Cet algorithme met en place une stratégie de *profondeur d'abord* avec un mécanisme de retour arrière sur la situation précédente lorsqu'il détecte que l'affectation partielle courante[3] n'est pas cohérente. Cet algorithme est souvent amélioré par des heuristiques déterminant, par exemple, l'ordre des variables à instancier et l'ordre des valeurs à tester pour minimiser le nombre de branches à explorer.

Les méthodes de résolution dites incomplètes n'explorent pas de façon systématique l'espace de recherche. Elles sont basées sur une exploration opportuniste de l'ensemble des affectations complètes[4] et ne fournissent qu'un sous-ensemble de solutions. Elles nécessitent une fonction d'évaluation et de comparaison d'affectations. Nous pouvons citer la méthode de recherche tabou (Glover et Laguna, 1993) ou le recuit simulé (Kirkpatrick et al., 1983). Ces méthodes incomplètes sont généralement utilisées pour résoudre des problèmes de taille élevée.

Les méthodes de résolution utilisées pour l'aide à la conception fournissent un ensemble de solutions. Il faut alors choisir la solution la mieux adaptée au problème de départ. Comme pour les raisonnements à base de cas, ce choix final requiert un effort de réflexion et d'analyse supplémentaire. Ces méthodes sont donc habituellement combinées à des mécanismes d'optimisation.

1.4.3 Méthodes de filtrage des problèmes de satisfaction de contraintes et aide à la conception

Dans les techniques de résolution présentées précédemment, les contraintes sont utilisées de manière passive. Elles sont exploitées uniquement pour tester la cohérence des affectations

[3] Une affectation est partielle lorsque seul un sous-ensemble des variables est instancié.
[4] Une affectation est complète lorsque l'ensemble de toutes les variables est instancié.

partielles et complètes. Les techniques de filtrage utilisent les contraintes de manière active pour effectuer des déductions sur le problème. L'objectif principal des techniques de filtrage est la détection d'affectations partielles localement ou totalement incohérentes. Une des techniques les plus utilisées est la technique de renforcement de la cohérence locale ou cohérence d'arc (Montanari, 1974). La cohérence d'arc vérifie que toute valeur du domaine d'une variable est compatible avec chaque contrainte prise séparément.

Il existe plusieurs degrés de filtrage qui permettent de vérifier la cohérence de n-uplets de valeurs de variables. Le degré de filtrage correspond au nombre de variables participant à la vérification de la cohérence locale. Plus le degré est grand, meilleure sera la détection des combinaisons incohérentes, mais plus il faudra de temps pour les détecter.

Ces méthodes de filtrage utilisées pour l'aide à la conception permettent de répercuter des choix sur le problème courant en éliminant les valeurs devenues incohérentes. La recherche de solutions est alors interactive : c'est la séquence de choix cohérents qui conduit à une ou plusieurs solutions.

Ces techniques de filtrage ou de propagation permettent d'améliorer les algorithmes de résolution dont Lobjois et Lemaitre (1997) dressent un inventaire. Par exemple, le *Forward checking* (Haralick et Elliot, 1980) combine le filtrage par arc-cohérence avec un algorithme de retour arrière.

Dans le cadre de nos travaux, nous utilisons différentes techniques de filtrage pour aider à la conception de manière interactive. Cette interactivité oblige, d'un point de vue mise en œuvre, à utiliser des méthodes de filtrage dont les temps de propagation sont compatibles avec des délais d'interaction. Actuellement, seules les méthodes de faible degré, basées principalement sur l'arc-cohérence, possèdent des temps de réponse qui répondent à cette exigence.

1.5 Travaux effectués dans le domaine

De nombreux travaux d'aide à la conception se basent sur les approches par contraintes. Nous en rappelons certains que nous typons suivant leur contribution orientée soit démarche de conception, soit algorithme de résolution ou de filtrage.

Conception de pièces mécaniques

Un des premiers outils d'aide à la conception basés sur les concepts de *CSP* a été réalisé dans le cadre du projet européen DEKLARE[5] (Vargas (1995) et Saucier (1997)). Ces travaux se sont particulièrement intéressés à la conception de pièces mécaniques à travers la modélisation du produit et du processus de conception. Le modèle produit se décompose suivant trois points de vue : fonctionnel, physique et géométrique. Le processus de conception est représenté sous forme d'un arbre ET/OU de tâches et de méthodes. Des liens représentant des contraintes permettent de faire correspondre les trois modèles produits et le modèle processus. Les choix réalisés se propagent alors indifféremment d'un modèle à l'autre, garantissant ainsi leur cohérence.

[5]DEsign KnowLedge Acquisition and Redesign Environment.

Conception d'un mixer industriel

Les travaux de Gelle (1998) proposent quant à eux de nouveaux concepts de modélisation de connaissances à base de *CSP*. Ceux-ci sont soutenus par une problématique d'aide à la conception de mixers industriels où le besoin de lier des variables discrètes (géométrie du bol du mixer) à des variables continues (son volume) se fait sentir. Pour répondre à ce besoin, le cadre des *CSPs* Mixtes, permettant l'expression de combinaisons autorisées de valeurs par des contraintes liant à la fois des variables symboliques et des variables numériques continues, a été défini. Des mécanismes de propagation et de maintien de cohérence sur ce type de contraintes sont développés.

Conception d'appareils à pression

Les travaux de Fischer (2000) sont également basés sur une double modélisation : le modèle de produit et le modèle du processus de conception. Ces travaux préconisent l'utilisation au plus tôt des règles de calcul mécanique dans le cycle de conception : le calcul est intégré à la conception et participe aux choix de conception. Cette démarche de conception fait intervenir dès le début du processus de conception les objectifs pour générer les résultats. Cette démarche est appliquée à la conception d'appareils à pression parallélépipédiques dans le cadre du projet *RNTL CO2*[6]. Les concepts de *CSP* sont utilisés pour garantir la cohérence des choix des concepteurs.

Conception d'avions

Les travaux de Mulyanto (2002) présentent une modélisation permettant la représentation générique d'un problème de conception préliminaire d'avions. Pour cela, il combine une représentation à base de contraintes avec une représentation orientée objet du problème. L'auteur utilise un environnement de développement nommé *Eclipse*[7]. Ce logiciel est capable de réaliser un filtrage de haut degré sur des contraintes numériques variées. Enfin, une approche à base de réseaux neuronaux multi-couches est mise en place pour intégrer au modèle générique des données ne pouvant pas être mises sous forme de *CSP* tels que des résultats expérimentaux ou de simulation, mais aussi des codes exécutables de calcul.

Conception de produits à forte diversité

Les travaux de Hadj-Hamou (2002) portent sur l'intégration dans le processus de conception de produits à forte diversité d'une phase de dimensionnement de leur chaîne logistique. Cette prise en compte permet d'une part, de réduire les temps de conception par la création d'un modèle générique de produit et d'autre part, de minimiser le coût total de fonctionnement de la chaîne logistique associée. Les concepts liés aux *CSPs* discrets sont utilisés pour représenter la diversité de produits au travers du modèle générique et pour adapter ce modèle générique à une variante du produit. Ce concept d'ingénierie intégrée est appliqué au câblage électrique automobile.

Conception de processus d'usinage

Les travaux de Ruet (2002) combinent une modélisation objet, une approche à base de cas et une approche à base de contraintes pour aider à la conception d'opérations d'usinage. Pour

[6] Réseau National des Technologies Logicielles, COnception par COntraintes.
[7] http://www.icparc.ic.ac.uk/eclipse/

cela, des conceptions antérieures, décrites à l'aide d'une représentation sous forme de diagramme de classes, sont introduites dans un outil à base de cas. Les techniques de satisfaction de contraintes sont utilisées pour guider la phase d'adaptation de la solution retenue au problème soumis. L'auteur détermine un espace d'adaptation où les techniques de filtrage des *CSPs* flous (Codognet et Rossi, 2000) sont exploitées pour éliminer les valeurs incohérentes. La détermination de l'incohérence d'une valeur est réalisée à partir de contraintes métiers sélectionnées par l'utilisateur.

Synthèse

Nos travaux se positionnent dans une problématique d'aide à la conception routinière de procédés. L'outil d'aide à la conception associé repose sur un raisonnement à base de contraintes. La recherche de solutions se fait de manière interactive et exploite par conséquent les méthodes de filtrage des problèmes de satisfaction de contraintes. Différentes techniques de *CSP* sont assemblées dans un même outil d'aide à la conception pour prendre en compte la diversité des connaissances à exploiter. Nos travaux s'inscrivent donc dans la continuité de ceux de Ruet (2002) du fait de notre terrain d'application et de Gelle (1998) en ce qui concerne notre contribution en termes de méthodes de filtrage.

Nos travaux sont soutenus par une problématique industrielle d'aide à la conception d'opérations de traitement thermique. Pour cela, un environnement d'aide à la conception regroupant deux outils (le premier à base de cas, l'autre à base de contraintes) doit être développé. L'outil d'aide à la conception à base de cas ne sera pas abordé dans ce mémoire. La mise au point de l'outil d'aide à la conception à base de contraintes constitue notre problématique.

1.6 Terrain d'application

Nous présentons dans cette section la problématique industrielle à l'origine de nos travaux, à savoir l'aide à la conception d'opérations de traitement thermique de trempe avec ou sans apport de carbone (procédé de cémentation). Ces travaux s'inscrivent dans un projet européen[8] nommé *VHT* pour *Virtual Heat Treatment*. Dans un premier temps, nous expliquerons pourquoi les industriels utilisent des opérations de traitement thermique de trempe et pourquoi ce procédé pose problème aux métallurgistes. Puis, nous présenterons l'organisation du projet en lots de travail et nous positionnerons nos travaux par rapport à ceux-ci.

1.6.1 Problématique industrielle

Pour améliorer les caractéristiques mécaniques des aciers, et plus particulièrement leur dureté, les métallurgistes leur font subir un cycle de traitement thermique avec ou sans apport de carbone en surface, nommé trempe avec ou sans cémentation. La trempe consiste, dans une première phase de chauffe, à augmenter progressivement la température de la pièce jusqu'à une température dite d'austénitisation. À cette température, spécifique à chaque acier, la micro-structure de celui-ci commence à se modifier. La pièce est alors maintenue un certain

[8] projet n° G1RD-CT-2002-00835.

laps de temps à cette température d'austénitisation pour homogénéiser la micro-structure de l'acier : c'est la phase de maintien. Enfin survient la phase de trempe qui se caractérise par un refroidissement brutal de l'acier par immersion dans un fluide de trempe préalablement choisi pour ses propriétés. La figure 1.2 présente les différentes phases d'un cycle de traitement thermique.

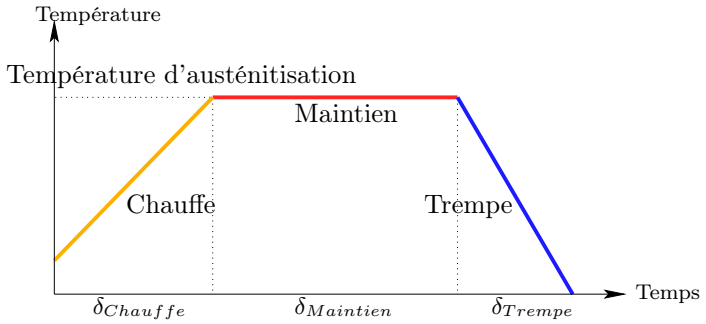


FIG. 1.2 – Exemple de cycle de traitement thermique

Lors de ce refroidissement brutal, la microstructure de l'acier trempé va se modifier, passant de la phase austénitique à la phase martensitique où elle se stabilise. La microstructure peut ainsi traverser différentes phases (perlitiques, ferritiques ou bainitiques) dépendant de l'acier utilisé et des conditions de traitement thermique. Ces transformations de phases couplées au gradient thermique, créé par le refroidissement brutal de l'acier, vont générer des contraintes résiduelles qui vont tendre à déformer la pièce. Ces déformations sont préjudiciables car elles peuvent conduire la pièce traitée soit à un nouvel usinage pour lui rendre sa forme originelle, soit à sa mise au rebut. C'est pourquoi les fabricants sont à la recherche de méthodes et d'outils leur permettant de définir le processus de traitement thermique et d'évaluer les déformations en résultant.

Aujourd'hui, la seule méthode disponible pour prédire les déformations consiste à simuler le traitement thermique par des approches de type éléments finis ou *FEM*. L'utilisation de ces logiciels nécessite un nombre important de données d'entrées : données thermophysiques, thermochimiques (température d'équilibre, taux de phases d'équilibre), cinétiques (transformation de phases), mécaniques (comportement des constituants) et chimiques (composition des aciers modélisés). Or, ces données sont complexes à obtenir et font toujours l'objet de recherches. Par exemple, l'obtention de données chimiques caractérisant un acier demande des années de travail. Nous pouvons citer les travaux de Veaux (2001) pour la caractérisation de l'acier *35MnV7* ainsi que ceux du *LSG2M*[9] portant, entre autres, sur la modélisation et la prise en compte dans les codes *FEM* des différents couplages thermique–transformations de phases et mécanique–transformations de phases. De plus, plusieurs données importantes, telles que l'historique des matériaux ou la géométrie de la charge, ne peuvent être prises en compte dans les approches par éléments finis.

Les approches *FEM* nécessitent un savoir-faire pour, par exemple, identifier et simuler les différents coefficients d'échanges thermiques à la surface des pièces. De plus, elles consomment énormément de temps, jusqu'à plusieurs jours de calcul pour une simulation. Leur utilisation est donc limitée industriellement aux entreprises spécialisées dans un procédé et un acier

[9]Laboratoire de Science et Génie des Matériaux et de Métallurgie de l'École des Mines de Nancy.

donnés. Pourtant, les traiteurs à façon arrivent à choisir le procédé conduisant à la dureté souhaitée tout en minimisant les déformations en exploitant des connaissances empiriques. D'où la problématique applicative supportant nos travaux.

1.6.2 Organisation du projet européen

Afin d'aider les industriels à choisir de manière plus rationnelle les conditions des opérations de traitement thermique sans recourir à la simulation, un consortium d'entreprises, dans le cadre du projet européen, souhaite mettre au point un système à base de connaissances interactif, basé sur des connaissances empiriques et sur leurs expériences, pour prédire les distorsions de manière qualitative et pour sélectionner les conditions de traitement thermique. Le projet se découpe en six lots de travail, nommés WP pour *Work Package*, comme illustré par la figure 1.3 :

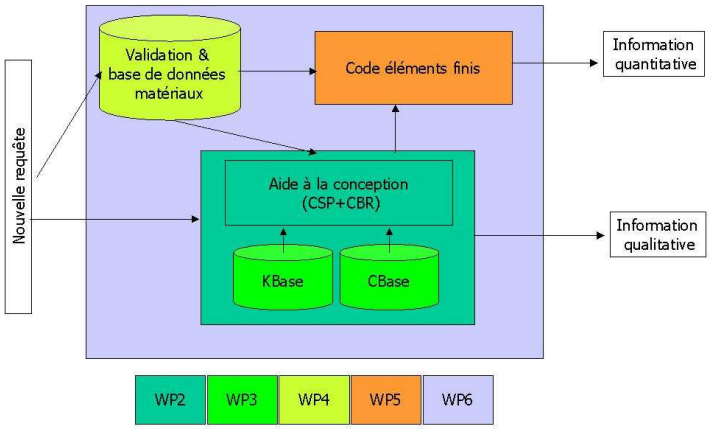


FIG. 1.3 – Plan général des lots de travail

WP1 Le WP1 correspond à l'organisation générale du projet.

WP2 Le WP2 se concentre essentiellement sur la mise au point des deux outils d'aide à la conception. Le premier repose sur une approche à base de cas appelée *CBase*, l'autre sur une approche à base de contraintes appelée *KBase*.

WP3 Le WP3 couvre les tâches de collecte des cas industriels pour le *CBase* et d'extraction des connaissances industrielles et empiriques pour le *KBase*.

WP4 Le WP4 valide, à partir d'essais industriels grandeur nature, différents fragments de connaissances afin de constituer un modèle de raisonnement cohérent et réaliste. Il doit aussi collecter dans une base de données les caractéristiques des matériaux les plus fréquemment utilisés par le consortium.

WP5 Le WP5 développe certains pré-traitements permettant de rendre les codes par éléments finis plus performants. Interfacés avec les bases de connaissances, ceux-ci doivent pouvoir sélectionner les modules de calcul indispensables à l'obtention de résultats proches de la réalité dans un temps plus raisonnable.

WP6 Le WP6 intègre les différents éléments dans un environnement global d'aide à la conception de traitement thermique et de prédiction de distorsions.

Nos travaux se concentrent sur deux lots de travail :

- tout d'abord, sur le WP2 par la réalisation de l'outil d'aide à la conception à base de contraintes et la mise au point des mécanismes de propagation et de maintien de cohérence,
- puis sur le WP3 pour aider les experts à extraire la connaissance, à la formaliser et à construire un modèle de raisonnement cohérent à base de contraintes.

1.7 Situation de la problématique

Nos travaux se positionnent donc :

- en conception routinière de procédé,
- dans une problématique d'aide à la conception,
- s'appuyant sur des raisonnements à base de contraintes,
- principalement dans le champ des outils d'aide plutôt qu'en démarche de conception,
- en assistance interactive et donc en méthodes de filtrage plutôt qu'en méthodes de résolution,
- sous-tendus par une problématique industrielle de procédé de traitement thermique qui regroupe une grande diversité de connaissances.

Les connaissances extraites et représentées sous forme de contraintes peuvent prendre des formes très diverses. Nous en dressons un inventaire, dans le chapitre 2, en termes de classes, de natures et de types de contraintes. Cette typologie de contraintes conduira au plan de lecture de notre mémoire.

Chapitre 2

Caractérisation des contraintes

Les contraintes expriment des relations entre les variables d'un problème de conception. Elles limitent l'espace de solutions de configuration en autorisant des combinaisons de valeurs et en en interdisant d'autres. C'est à travers elles que les choix de conception transitent pour supprimer les alternatives incompatibles avec les choix effectués. Les types de relations pouvant lier les variables d'un problème de conception sont généralement très divers. Ils nécessitent plusieurs modes de représentation et donc des méthodes de filtrage adaptées afin de garantir une certaine cohérence du problème. Les algorithmes de propagation de contraintes garantissent l'atteinte d'une solution cohérente si celle-ci existe et permettent, également, de prouver de manière indéniable qu'un problème n'a pas de solution.

Nous caractériserons, dans la première section, les variables en posant les notions de variables symboliques, numériques, discrètes et continues. Dans la deuxième section, nous affinerons la notion de contraintes en fonction de leurs natures, de leurs classes et de leurs types. Enfin, nous présenterons deux types de contraintes fréquentes en conception : les combinaisons de valeurs autorisées et les expressions mathématiques.

2.1 Variables

Les contraintes limitent les domaines de définition des variables en restreignant les combinaisons de valeurs autorisées. Les variables sur lesquelles s'appliquent les contraintes peuvent être de types différents : symboliques ou numériques, discrètes ou continues. Cette classification s'appuie sur le genre des éléments de leurs domaines et sur la cardinalité de ces derniers.

2.1.1 Variables symboliques

Les variables symboliques ont une sémantique bien définie. Elles représentent généralement des données de haute abstraction. Les variables symboliques font partie de l'ensemble des

variables discrètes puisque leur domaine, représenté par une liste de symboles, a un cardinal fini dénombrable[1].

Par exemple, dans notre application, nous pouvons considérer la variable représentant le choix du fluide de trempe comme symbolique puisque son domaine correspond à l'ensemble des noms des fluides de trempe disponibles :

```
variable symbolique FdT de domaine {"gaz", "huile", "eau"}
```

2.1.2 Variables numériques

Les variables numériques peuvent ne pas avoir de sémantique. Elles se décomposent en deux sous-ensembles disjoints :

- les variables numériques discrètes,
- les variables numériques continues.

Les domaines des variables numériques discrètes sont finis dénombrables ou infinis dénombrables. Ils se définissent à partir de listes d'entiers ou de réels, ou bien à partir d'intervalles de valeurs entières.

Par exemple, la catégorie d'un four de chauffe peut se représenter à l'aide d'une variable numérique discrète où chaque valeur a une sémantique bien définie :

```
variable entière Classe_de_four de domaine {[3, 5], [7, 10], 15}
```

Les domaines des variables numériques continues sont infinis non dénombrables[2]. Ils se limitent à des intervalles de réels.

Par exemple, la catégorie de four renseigne sur les plages de valeurs de températures que celui-ci peut atteindre :

```
variable réelle Température_four de domaine {[0, 1000]}
```

2.1.3 Synthèse

Il existe deux manières de cartographier l'ensemble des variables. Si nous considérons uniquement le genre des éléments des domaines de définition, nous obtenons deux sous-ensembles disjoints : les variables symboliques et les variables numériques. Si nous considérons la cardinalité des domaines de définition (dénombrable ou non), nous obtenons deux sous-ensembles : les variables discrètes et les variables continues. La figure 2.1 synthétise les notions de discret et de continu en fonction du type des éléments des domaines et de leur cardinal.

[1] Le cardinal d'un ensemble A est dénombrable s'il existe une bijection entre A et $\mathbb{N}$. Le cardinal des ensembles dénombrables est $\aleph_0$.

[2] Le cardinal des ensembles infinis non dénombrables est $\aleph_1$, ce qui leur confère la propriété de continuité.

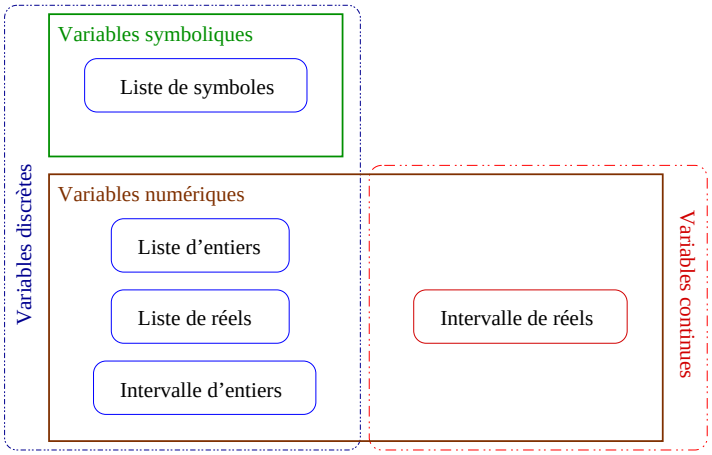


FIG. 2.1 – Classification des variables

2.2 Contraintes

Les contraintes permettent de restreindre l'espace de recherche en délimitant les combinaisons de valeurs que les variables peuvent prendre simultanément. Elles peuvent être exprimées en *extension* ou en *intension*. Les contraintes définies en extension présentent la liste exhaustive de tous les arrangements de valeurs autorisés. Elles portent le plus souvent sur des variables discrètes. Les contraintes décrites en intension concentrent les informations pour plusieurs raisons. Tout d'abord, il peut être plus facile d'écrire la contrainte en intension (pour éviter de lister toutes les possibilités autorisées) ou parce qu'il peut être tout à fait impossible de lister de manière complète les combinaisons de valeurs. Les contraintes en intension portent généralement sur des variables numériques.

Nous pouvons distinguer trois classes de contraintes à partir des types de variables sur lesquelles elles agissent (Gelle, 1998). La première classe est celle des contraintes discrètes qui portent uniquement sur des variables discrètes (*CSP* discret). La deuxième classe est celle des contraintes continues ($CCSP$[3]), appelée aussi contraintes numériques ($NCSP$[4]), qui ne concernent que des variables continues. La dernière classe est celle des contraintes mixtes ($MCSP$[5]) qui lient à la fois des variables discrètes et continues.

L'arité d'une contrainte représente le nombre de variables distinctes sur lesquelles elle agit. Une contrainte binaire portera sur deux variables, alors qu'une contrainte n-aire portera sur n variables distinctes. Par exemple, la contrainte $y = x + 2x^2$ est binaire, puisqu'elle ne lie que les variables x et y.

2.2.1 Nature de contraintes

Les contraintes permettent de représenter deux natures de relations. La première est la restriction des choix de valeurs autorisées par des contraintes dites de compatibilité. La seconde

[3] *Continuous Contraint Satisfaction Problem*
[4] *Numerical Constraint Satisfaction Problem*
[5] *Mixed Constraint Satisfaction Problem*

est liée à la structure du problème, c'est-à-dire aux ensembles de variables et de contraintes qui participent à la résolution du problème courant. Ces ensembles peuvent être modifiés au cours de la résolution par des contraintes dites d'activation. En effet, certains éléments du problème peuvent ne pas être pris en compte dès le début de la résolution. Des événements bien déterminés peuvent alors conduire à les inclure ou à les retirer du problème courant. Cette modification de la structure du problème nécessite la mise en place de méthodes assurant sa cohérence en fonction du type d'événements.

2.2.1.1 Contrainte de compatibilité

Les contraintes de compatibilité permettent de définir les combinaisons de valeurs autorisées pour un ensemble de variables. Elles peuvent être décrites aussi bien en extension qu'en intension. Les contraintes de compatibilité exprimées en extension sont représentées par des listes de n-uplets indiquant quelles sont les valeurs compatibles entre elles. Les contraintes de compatibilité exprimées en intension sont représentées soit par des listes de combinaisons factorisées de valeurs autorisées (intervalle d'entiers ou de réels), soit par des expressions mathématiques.

Par exemple, indiquer que la trempe d'un matériau, variable symbolique *Matériau*, n'est réalisable qu'avec certains types de fluides, variable symbolique FdT va être décrit, en extension, par une liste de couple de valeurs (*Matériau*, FdT) : {(42CrMo4, eau), (42CrMo4, air), (30CrNiMo8, huile), (30CrNiMo8, eau)}.

Tandis que la contrainte liant le type d'un four, variable symbolique *Four*, à sa classe, variable entière *Classe_ de_ four*, va être décrite en intension par une liste de couple de valeurs (*Four*, *Classe_ de_ four*) : {(Basse pression, $[3, 5]$), (Atmosphérique, $[7, 9]$, (Haute pression, $[9, 10]$)}.

2.2.1.2 Contrainte d'activation

Les contraintes d'activation permettent d'ajouter ou de supprimer des variables du problème courant (Mittal et Falkenhainer, 1990). C'est la détection d'événements singuliers qui déclenche l'ajout ou le retrait d'un ensemble d'éléments. Les contraintes d'activation sont définies par deux parties : une prémisse (disjonction de conjonctions) et un conséquent (ajout ou retrait d'éléments) liées par un opérateur d'implication $\rightarrow$. Si la prémisse est vraie, il y a application du conséquent. Sinon, le problème reste inchangé. La prémisse s'exprime comme une contrainte de compatibilité en extension ou en intension.

Par exemple, l'ajout de caractéristiques géométriques (trous, dents, etc) lors de la description de la forme de la pièce trempée peut être régie par des contraintes d'activation. L'ajout d'un trou dans la géométrie de la pièce, variable symbolique $Trou = oui$, va nécessiter la prise en compte de deux autres variables : le diamètre du trou, variable continue d, et sa longueur, variable continue l :

$$(Trou = oui) \rightarrow activation : \{d, l\}.$$

2.2.2 Type de contraintes

Lors de la phase de modélisation de connaissances, nous avons été confrontés à une grande diversité de contraintes de conception similaire au recensement présenté par Yannou (1998). Plusieurs d'entre elles évoquent des listes de combinaisons de valeurs, d'autres correspondent à des expressions mathématiques et certaines représentent des abaques tirés d'expérimentations. La modélisation de ces connaissances sous forme de contraintes nécessite l'utilisation de différentes techniques *CSP*. Nous présentons la recension des types de contraintes identifiées que nous illustrons par des exemples tirés de notre application.

2.2.2.1 Tables de compatibilité

La représentation de listes de combinaisons de valeurs autorisées peut se faire à l'aide des tables de compatibilité. Les tables de compatibilité permettent de représenter sous forme tabulaire des listes de n-uplets de valeurs autorisées. Chaque combinaison de n-uplet est mise sous forme de tuple autorisé. La combinatoire exprimée sous forme de table de compatibilité correspond à un sous-ensemble du produit cartésien des domaines des variables participant à la contrainte.

Par exemple, indiquer que la trempe d'un matériau n'est réalisable qu'avec certains types de fluides revient à créer la table de compatibilité binaire représentée par le tableau 2.1. Celle-ci indique entre autre que le matériau 42CrMo4 ne peut être trempé qu'avec un fluide de trempe correspondant à de l'eau ou de l'air.

Tab. 2.1 – Exemple de contrainte de compatibilité de type table de compatibilité

Matériau	FdT
42CrMo4	eau
42CrMo4	air
30CrNiMo8	huile
30CrNiMo8	eau

Certaines des contraintes d'activation utilisées dans notre application ont des prémisses de type table de compatibilité. Nous les aborderons dans le chapitre 6.

Par exemple, la présence d'un trou débouchant à l'intérieur de la pièce, variable symbolique *Trou*, et la direction perpendiculaire de la gravité par rapport à son axe, variable symbolique *Gravité*, vont entraîner l'apparition d'une déformation dite d'ovalisation. Cette contrainte peut se traduire par la contrainte d'activation de type table de compatibilité binaire, représentée par le tableau 2.2.

2.2.2.2 Expressions mathématiques

Il appert que la modélisation d'un grand nombre de problèmes industriels basée uniquement sur des contraintes de type table de compatibilité est pratiquement impossible. Les expressions mathématiques sont essentielles pour calculer des paramètres de conception ou pour

TAB. 2.2 – Exemple de contrainte d'activation de type table de compatibilité

Gravité	Trou
Perpendiculaire	oui

$\rightarrow activation : \{ovalisation\}$

représenter des lois physiques et des résultats d'expérimentations. Nous présentons dans cette sous-section deux types de contraintes numériques.

Fonctions numériques de n-variables Certaines relations peuvent se mettre sous la forme d'une fonction numérique de $\mathbb{R}^n$ dans $\mathbb{R}$ où n équivaut au moins à l'arité de la contrainte associée. Nous distinguons cependant deux cas : celui où $n = 1$ et celui où $n > 1$.

Le résultat d'une fonction numérique peut être soit affecté à une variable résultat, soit comparé à une valeur réelle implicite (précédemment affectée à une variable) ou explicite (une valeur réelle). Le sort du résultat et les propriétés des fonctions numériques (continuité, monotonie[6]) influent sur la modélisation de la contrainte.

Cas où n = 1. Un exemple de contrainte qui peut se représenter par une fonction numérique de $\mathbb{R}$ dans $\mathbb{R}$ est celle qui définit que le cylindre traité est semi-infini. Un cylindre semi-infini a sa hauteur, variable continue h, au moins trois fois supérieure à son diamètre, variable continue d. Ceci se traduit de la manière suivante :

$$\begin{array}{rccc} f: & \mathbb{R} & \rightarrow & \mathbb{R} \\ & d & \mapsto & 3 \times d \end{array}$$

Dans ce cas, le résultat de $f(d)$ est soit affecté ou comparé à la variable continue h par la contrainte binaire suivante $h \geq 3 \times d$, soit comparé à une valeur réelle par la contrainte unaire suivante : $80 \leq 3 \times d$.

Cas où n > 1. Pour évaluer, par exemple, la différence de température entre le cœur et la surface d'une pièce traitée, nous utilisons une fonction numérique simple de $\mathbb{R}^2$ dans $\mathbb{R}$. Soient t_c et t_s, les températures respectivement à cœur et en surface. La contrainte s'écrit de la manière suivante :

$$\begin{array}{rccc} g: & \mathbb{R}^2 & \rightarrow & \mathbb{R} \\ & (t_c, t_s) & \mapsto & t_c - t_s \end{array}$$

Le résultat de $g(t_c, t_s)$ est soit affecté ou comparé à une variable continue résultat T par la contrainte ternaire suivante : $t_c - t_s = T$, soit comparé à une valeur réelle par la contrainte binaire suivante : $t_c - t_s < 10$.

Certaines contraintes d'activation utilisées dans notre application ont des prémisses de type fonction numérique dont le résultat est comparé à une valeur réelle explicite ou non. Nous les aborderons dans le chapitre 6.

Par exemple, un épaulement, variable symbolique *Épaulement*, est pris en compte si sa hauteur maximale, variable continue H_{Max}_E, est supérieure ou égale à 1.5 fois le diamètre moyen de

[6]Uniquement dans le cas de fonctions numériques de $\mathbb{R}$ dans $\mathbb{R}$.

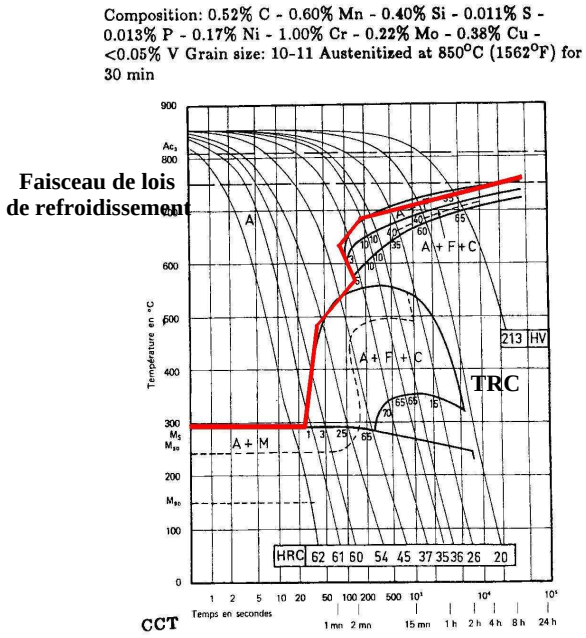


FIG. 2.2 – Exemple de diagramme *TRC* : acier 50CD4 et son réseau de loi de refroidissement

l'axe cylindrique, variable continue d. Ceci se traduit par la contrainte d'activation suivante :

$$\left(\frac{H_{Max} - E}{d} \geq 1.5\right) \quad \rightarrow \quad activation : \{\acute{E}paulement\}$$

Abaques Il est courant, en conception, de prendre en compte des résultats expérimentaux sous forme d'abaques (Mulyanto, 2002). L'intégration de ce type de connaissances dans un problème de satisfaction de contraintes peut se faire à l'aide d'approximation par des fonctions numériques continues et définies par morceaux. La figure 2.2 illustre ce type de connaissances.

Afin de caractériser la dureté après trempe et d'approximer les déformations potentielles, les métallurgistes exploitent deux abaques correspondant à un diagramme de phases spécifique à l'acier trempé (diagramme *TRC* pour *Transformation à Refroidissement Continu*) et à un faisceau de lois de refroidissement approximant la température de l'acier en fonction du temps, figure 2.2. Nous proposons d'approcher ce type de connaissances par des contraintes numériques continues pour les lois de refroidissement et des contraintes numériques continues définies par morceaux pour les différentes phases du diagramme *TRC*. La prise en compte de ces contraintes particulières dans le modèle de raisonnement est exposée dans le chapitre 5.

2.3 Synthèse et problématique

Dans ce chapitre, nous avons présenté une cartographie des variables à partir de deux points de vue. Le premier porte sur les éléments constitutifs des domaines de validité, le second, sur

le cardinal des domaines. À travers cette cartographie, les notions de discret et de continu sont positionnées.

Les contraintes sont les briques de construction des modèles de raisonnement. C'est à travers elles que les choix effectués sur les variables se propagent pour restreindre les choix possibles sur d'autres variables et aider ainsi à la conception. Deux natures de contraintes assurent cette fonction : les contraintes de compatibilité et les contraintes d'activation. Les contraintes de compatibilité et d'activation assurent la cohérence des choix. Les contraintes d'activation également jouent un rôle important dans la structuration du problème.

Les contraintes sont regroupées en trois classes selon le type de variables qu'elles restreignent : les contraintes discrètes, les contraintes continues ou numériques et les contraintes mixtes.

Deux types de contraintes, qui servent à la fois à définir des contraintes de compatibilité et à décrire les prémisses des contraintes d'activation, ont été identifiés. Le premier type de contraintes liste les combinaisons de valeurs autorisées sous forme de table de compatibilité. Le deuxième correspond aux expressions mathématiques.

La figure 2.3 synthétise notre classification des contraintes.

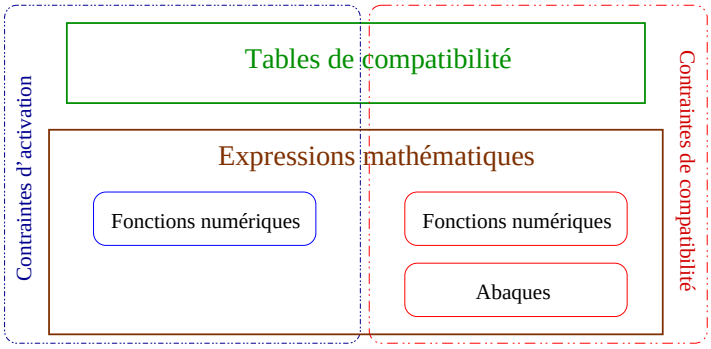


FIG. 2.3 – Classification des contraintes

La diversité des connaissances à modéliser et des contraintes à prendre en compte nous conduit à considérer différents types de problèmes de satisfaction de contraintes, mais aussi à en adapter ou à en étendre certains. Le système d'aide à la conception visé étant interactif, nous intégrons plusieurs méthodes de filtrage propres à chaque type de *CSPs* utilisé. Nous présentons ici le plan de lecture de notre mémoire.

Tables de compatibilité

Certaines connaissances peuvent se mettre sous forme de listes de combinaisons de valeurs autorisées. Pour formaliser ce type de connaissances à base de *CSP*, nous utilisons le concept de table de compatibilité que nous présentons dans le chapitre 3. Nous employons en l'état ces contraintes pour lier des variables discrètes. Nous proposons d'étendre le concept de table de compatibilité pour prendre en compte les variables continues.

Expressions mathématiques dites « simples »

Une autre forme de connaissance rencontrée correspond aux expressions mathématiques dites « simples » possédant des propriétés particulières telles que la continuité, la monotonie et

la projetabilité. La propagation de ce type de contrainte se fait à partir d'une technique de filtrage sur les *CSPs* numériques nommée 2B-cohérence que nous présenterons dans le chapitre 4. Nous utilisons en l'état cette technique pour filtrer les variables continues mono-intervalles et nous proposons, ensuite, de l'adapter pour filtrer les variables continues multi-intervalles.

Expressions mathématiques dites « complexes »

Les abaques expérimentaux sont intégrés dans les modèles de connaissances sous forme d'expressions mathématiques dites « complexes » continues et « complexes » définies par morceaux. Ces expressions mathématiques nécessitent une discrétisation complète de leur espace de solutions pour être filtrées. Cette discrétisation de l'espace est représentée sous forme d'une structure arborescente empruntée au traitement d'images, nommée 2^k-arbre où k correspond au nombre de variables participant à la contrainte. L'intégration de cette structure arborescente dans les *CSPs* ne pose pas de problème particulier pour les expressions mathématiques continues. Nous les utilisons en l'état pour les expressions mathématiques continues et nous proposons de les étendre pour prendre en compte les expressions mathématiques définies par morceaux.

La prise en compte simultanée d'abaques est réalisée par le mécanisme de fusion de leurs 2^k-arbres. Comme, dans notre application, les abaques ne sont discrétisés que lorsque certaines variables du traitement thermique sont renseignées, nous proposons d'améliorer la génération et la fusion des 2^k-arbres, afin de garantir l'interactivité du système, par une exploration opportuniste de l'espace de solutions.

Le chapitre 5 présentera la structure de 2^k-arbre permettant la prise en compte d'expressions mathématiques définies ou non par morceaux dans les *CSPs*, ainsi que les méthodes de génération et de fusion (conjonction de contraintes) de ce type particulier de contraintes.

Contraintes d'activation

Pour représenter la diversité des problèmes, certains éléments peuvent être ajoutés durant le processus de conception. Cette prise en compte passe par l'utilisation des *CSPs* dits dynamiques. Nous employons en l'état ces techniques introduites pour moduler l'existence de variables que nous complétons, par la suite, pour prendre en compte l'existence de contraintes. Le chapitre 6 présentera les concepts de *CSPs* dynamiques et les extensions apportées pour moduler l'existence des contraintes.

Application et modèle de raisonnement

Le chapitre 7 présentera un exemple de modèle de traitement thermique construit avec des experts, sa formalisation sous forme de contraintes et l'assemblage des différentes méthodes de filtrage utilisées. Ce modèle représente un mode de raisonnement permettant à la fois de concevoir une opération de traitement thermique de trempe et de prédire de manière qualitative les déformations en résultant. Notre proposition se limite à la famille des axes cylindriques qui peuvent se décliner sous plusieurs formes à partir, par exemple, de la présence de trous ou d'épaulements. Tous les concepts présentés précédemment sont exploités pour la représentation et l'utilisation de ce modèle.

Chapitre 3

Tables de compatibilité et extension

Les tables de compatibilité sont les premiers types de contraintes pris en compte dans les problèmes de satisfaction de contraintes (Montanari, 1974). Elles permettent de représenter de manière tabulaire des listes de combinaisons de valeurs autorisées. À l'origine, ces contraintes étaient exclusivement discrètes et binaires. Mais des besoins se sont vite faits sentir par rapport à ces restrictions : le premier sur l'arité des tables de compatibilité et le second sur la nature des variables contraintes.

Afin de remédier au problème d'arité limitée, la première idée fut de convertir les contraintes n-aires en plusieurs contraintes binaires par l'ajout de variables intermédiaires et d'obtenir ainsi un problème binaire équivalent au problème de départ (Rossi et al., 1990). Cette transformation n-aire $\rightarrow$ binaire se révéla peu naturelle à exploiter et difficile à mettre en place pour des problèmes réels de grandes tailles. Une seconde idée fut alors de conserver les contraintes n-aires et d'étendre les algorithmes de filtrage binaires aux contraintes n-aires (Bessière et al., 1999).

Pour pallier à la prise en compte exclusive des variables discrètes, le concept de table de compatibilité fut étendu aux variables continues par l'intermédiaire de contraintes mixtes liant des variables discrètes et continues (Gelle, 1998). Les méthodes de filtrage associées se ramènent à celles des tables de compatibilité discrètes par l'association de labels discrets aux intervalles des domaines continus. Pour lister des combinaisons d'intervalles autorisées, nous proposons d'élargir la notion de table de compatibilité aux variables continues en proposant des méthodes de filtrage, sans utilisation de labels, qui puisent les intervalles à tester directement dans les contraintes et non dans les domaines des variables. Ces méthodes sont de plus applicables à toutes les classes de tables de compatibilité.

Nous aborderons, dans ce chapitre, les différentes classes des contraintes de type table de compatibilité ainsi que les méthodes de filtrage associées et nous présenterons notre contribution pour le filtrage des tables de compatibilité continues.

3.1 Tables de compatibilité discrètes

Nous présentons dans cette section la définition des tables de compatibilité discrètes n-aires, ainsi que les méthodes de filtrage plus ou moins locales qui leur sont associées.

3.1.1 Définition

Les tables de compatibilité discrètes, par essence, ne lient que des variables discrètes. Elles peuvent être exprimées soit en extension, soit en intension.

Le tableau 3.1 présente une contrainte de compatibilité binaire décrite en extension pour la partie gauche et en intension pour la partie droite. Celle-ci lie une catégorie de four, variable symbolique *Four*, de domaine {Basse pression, Atmosphérique, Haute pression}, à sa classe, variable entière *Classe_de_four* de domaine $\{[3,5],[7,10],15\}$.

Four	*Classe_de_four*
Basse pression	3
Basse pression	4
Basse pression	5
Atmosphérique	7
Atmosphérique	8
Atmosphérique	9
Haute pression	9
Haute pression	10

Four	*Classe_de_four*
Basse pression	[3, 5]
Atmosphérique	[7, 9]
Haute pression	[9, 10]

TAB. 3.1 – Exemple de table de compatibilité en extension et en intension

3.1.2 Filtrage

L'objectif principal des techniques de filtrage est la détection des affectations partielles localement incohérentes. Pour cela, toutes les valeurs des variables sont testées afin de détecter et d'éliminer celles qui ne peuvent jamais mener à une solution. Il existe différents degrés de filtrage selon le nombre de variables participant à l'affectation partielle. Plus celui-ci est élevé, meilleur sera le filtrage, mais plus le temps consacré à l'élimination des valeurs de variables incompatibles avec une solution risque d'être long.

3.1.2.1 Arc-cohérence

La cohérence d'arc vérifie que toute valeur du domaine d'une variable est compatible avec une contrainte agissant sur la variable. Cette technique, développée par Waltz (1975), possède l'un des degrés de filtrage le moins élevé. La première définition formelle fut donnée par Mackworth (1977) pour des contraintes discrètes binaires.

Définition 3 : *arc-cohérence*

Un arc liant les variables (V_i, V_j) est arc-cohérent si pour toute valeur v_i du domaine D_{V_i} de la variable V_i, il existe une valeur v_j du domaine D_{V_j} de la variable V_j telle que le couple de valeurs (v_i, v_j) soit autorisé par la contrainte binaire liant V_i et V_j.

L'algorithme 1 nommé REVISE détecte et élimine les valeurs d'une variable qui ne peuvent mener à une solution. Cet algorithme est directionnel : il s'applique d'une variable V_i vers une autre variable V_j.

Alg. 1 REVISE(ARC : $V_i \rightarrow V_j$)

– $\therefore$ – ***Cet algorithme filtre l'arc $V_i \rightarrow V_j$.***
– $\therefore$ – *reduction indique si le domaine de V_i a été réduit.*
reduction $\leftarrow$ Faux
Pour toutes les valeurs $v_i \in D_{V_i}$ **Faire**
  Si ($\nexists v_j \in D_{V_j}$ / $(V_i = v_i, V_j = v_j)$ soit cohérent) **Alors**
    – $\therefore$ – *Il y a retrait de la valeur v_i du domaine de V_i*
    $D_{V_i} \leftarrow D_{V_i} \setminus \{v_i\}$
    reduction $\leftarrow$ Vrai
  Fin Si
Fin Pour
Retourner *reduction*

Définition 4 : CSP *arc-cohérent*

Un CSP *est arc-cohérent si pour tout couple de variables (V_i, V_j), de domaines respectifs D_{V_i} et D_{V_j} et pour toute valeur $v_i \in D_{V_i}$, il existe une valeur $v_j \in D_{V_j}$ telle que toutes les contraintes $C_{(i,j)}$ liant V_i et V_j, prises une à une, soient satisfaites.*

La procédure AC1 développée par Mackworth permet de filtrer par arc-cohérence un *CSP* discret binaire à partir de la fonction REVISE. Le problème majeur de AC1 est que la révision d'un seul domaine entraîne la reconduite du filtrage sur l'ensemble des variables du *CSP*. Pour éviter cela, Mackworth (1977) proposa une nouvelle version de cette procédure, nommée AC3 (algorithme 2 page suivante), où seules les variables dont le domaine est réduit subissent un nouveau filtrage. La procédure AC3 a une complexité temporelle polynomiale en $O(m.d^3)$ (Mackworth et Freuder, 1985) où m représente le nombre de contraintes et d la taille maximum des domaines. Cet algorithme a fait l'objet de nombreuses améliorations, mais reste l'un des plus utilisés pour le maintien de l'arc-cohérence (Barták, 1998). Nous citons, pour exemple, les algorithmes AC6 (Bessière et Cordier, 1993) et AC7 (Bessière et Régin, 1994) qui ont une complexité temporelle en $O(m.d^2)$.

Dans le cas de la contrainte présentée dans le tableau 3.1, le filtrage par arc-cohérence supprimera les valeurs numériques discrètes $\{7, 8, 15\}$ du domaine de définition de la variable *Classe_de_four* de domaine $\{[3, 5], [7, 10], 15\}$, si la variable *Four* est réduite à {*Haute pression*, *Basse Pression* }.

L'arc-cohérence filtre de manière locale, mais ne garantit pas qu'un problème possède bien une solution. Si nous considérons l'exemple présenté dans la figure 3.1, trois contraintes binaires

Alg. 2 AC3(CSP : G)

– ∴ – ***Cet algorithme filtre un CSP discret par arc-cohérence.***
– ∴ – Q = *liste de couples de variables liées par une contrainte*
$Q \leftarrow \{(V_i, V_j) \in arcs(G), i \neq j\}$
– ∴ – *Tant qu'il reste des couples de variables* (V_i, V_j) *à filtrer*
Tant Que $(Q \neq \emptyset)$ **Faire**
  Dépiler un arc (V_k, V_m) de Q
  – ∴ – *Si la variable* V_k *est réduite*
  Si (REVISE(V_k, V_m)(algorithme 1 page précédente)) **Alors**
    – ∴ – *Il faut filtrer les variables liées à* V_k
    – ∴ – *Empiler dans* Q *toutes les variables* V_i *liées à* V_k
    $Q \leftarrow (Q \cup \{(V_i, V_k)/\ (V_i, V_k) \in arcs(G), i \neq k, i \neq m\})$
  Fin Si
Fin Tant Que

lient trois variables discrètes. Ce problème est arc-cohérent sans pour autant qu'il ait de solution.

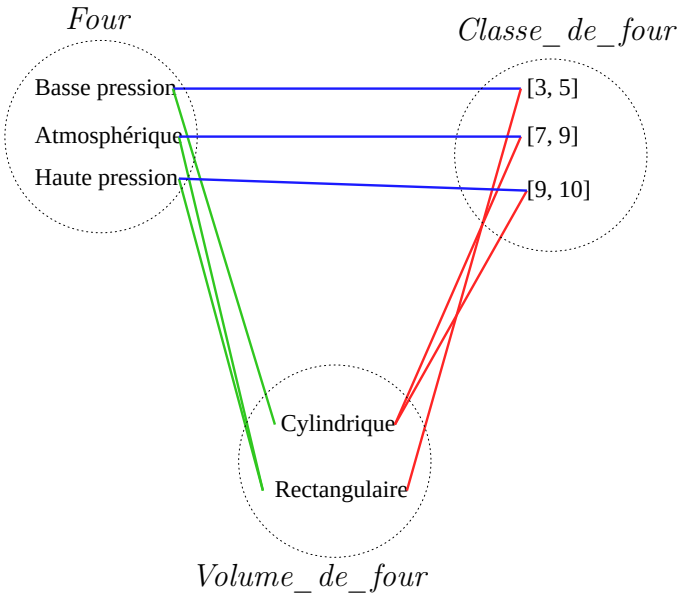


FIG. 3.1 – Problème arc-cohérent sans solution

3.1.2.2 K-cohérence

La notion d'arc-cohérence fut étendue par Freuder (1978) pour éliminer au plus tôt le plus grand nombre possible de combinaisons de valeurs incohérentes. Pour cela, la prise en compte simultanée d'un ensemble de k variables, avec $k \geq 3$, est nécessaire. Le degré de filtrage de cette technique dépend du nombre k de variables considérées. Plus k se rapproche du nombre de variables du *CSP*, plus le degré de filtrage est important.

Définition 5 : *K-cohérence*

Un CSP *est k-cohérent si pour tout k-uplet de variables* $(V_1, V_2, \ldots, V_k)$*, de domaines respectifs* D_{V_1}*,* D_{V_2}*,...,* D_{V_k} *et pour tout* $(k-1)$*-uplet de valeurs cohérentes* $(v_1 \in D_{V_1}, v_2 \in D_{V_2}, \ldots, v_{k-1} \in D_{V_{k-1}})$*, il existe une valeur* $v_k \in D_{V_k}$ *telle que toutes les contraintes* C_l *liant le k-uplet de variables* $(V_1, V_2, \ldots, V_k)$ *soient satisfaites.*

Pour détecter qu'un problème ne possède pas de solution, k doit être égal au nombre de variables du problème. Le problème représenté par la figure 3.1 sera détecté incohérent par un filtrage par 3-cohérence.

3.2 Tables de compatibilité mixtes

Nous présentons dans cette section la définition des tables de compatibilité mixtes n-aires, ainsi que les méthodes de filtrage plus ou moins locales qui leur sont associées.

3.2.1 Définition

La première définition des tables de compatibilité mixtes fut donnée par Gelle (1998). Elles permettent de lier des variables discrètes à des variables continues et ne peuvent, par conséquent, être exprimées qu'en intension.

Définition 6 : *table de compatibilité mixte*

Une table de compatibilité mixte $C_{(V_1,V_2,\ldots,V_m)}$ *est une relation définie par un couple de contraintes* (C^{Dis}, C^{Con}) *où la première contrainte est une contrainte discrète* $C^{Dis}_{(V_1,\ldots,V_j)}$ *définie sur des variables discrètes* $V_1, \ldots, V_j$ *et la seconde une contrainte continue* $C^{Con}_{(V_{j+1},\ldots,V_m)}$ *définie sur des variables continues* $V_{j+1}, \ldots, V_m$*.*

Les contraintes continues C^{Con} peuvent être de deux types, que nous illustrons par des exemples tirés de notre application :

- soit des intervalles de réels,
- soit des fonctions numériques.

La contrainte mixte, présentée par le tableau 3.2, liant la classe du four, variable discrète *Classe_de_four* de domaine $\{[3,5],[7,10],15\}$, à la température maximale atteinte par le four, variable continue *Température_four* de domaine $\{[0,1000]\}$, a sa composante C^{Con} décrite par une liste d'intervalles semi-ouverts et sa composante C^{Dis} décrite en intension par une liste d'intervalles fermés.

Par exemple, la contrainte mixte, tirée de Gelle (1998) et présentée dans le tableau 3.3 liant la géométrie du bac de trempe, variable symbolique G_BdT, à son volume, variable continue V_BdT a sa composante C^{Con} de type fonction numérique. La calcul du volume dépend en

TAB. 3.2 – Exemple de table de compatibilité mixte où C^{Con} est une liste d'intervalles semi-ouverts

Classe_de_four	*Température_four*
$[3, 5]$	< 1500
$[7, 9]$	< 1200
$[9, 10]$	< 900

effet de sa géométrie, variable symbolique G_BdT, de sa hauteur, variable continue h_BdT et de son diamètre, variable continue d_BdT où :

$$\begin{aligned} f(d\_BdT, h\_BdT) = & \ \frac{1}{4} \times \pi \times d\_BdT^2 \times h\_BdT \\ g(d\_BdT, h\_BdT) = & \ \frac{\pi \times d\_BdT^2}{4} \times (h\_BdT - \frac{d\_BdT}{6})) \end{aligned}$$

TAB. 3.3 – Exemple de table de compatibilité mixte où C^{Con} est une liste de fonctions numériques

G_BdT	V_BdT
Cylindrique	$f(d\_BdT,\ h\_BdT)$
Hémisphérique	$g(d\_BdT,\ h\_BdT)$

Nous ne considérons pas, dans nos travaux, les contraintes mixtes dont la composante continue C^{Con} est une liste de fonctions numériques. Celles-ci sont représentées par des contraintes d'activation de contraintes qui seront développées dans le chapitre 6.

3.2.2 Filtrage

Les méthodes de filtrage des tables de compatibilité mixtes dont la composante continue C^{Con} est décrite par des intervalles de réels utilisent les méthodes de filtrage des tables de compatibilité discrètes.

Les intervalles de valeurs sont associés à des labels discrets. Des fonctions permettent de passer de la variable continue à ses labels et inversement :

$$\begin{aligned} T_D : & \quad Intervalles \quad \rightarrow \quad Labels \\ T_D^{-1} : & \quad Labels \quad \rightarrow \quad Intervalles \end{aligned}$$

La figure 3.2 présente l'association de labels discrets aux intervalles du domaine de la variable continue *Température_four*.

C'est à partir de ces labels que le filtrage est réalisé (Faltings, 1994). Comme les labels sont discrets, les méthodes de filtrage associées aux tables de compatibilité discrètes peuvent être utilisées (arc-cohérence et k-cohérence). Une fois le filtrage discret réalisé, la fonction inverse T_D^{-1} permet de retrouver les intervalles encore cohérents avec le problème courant.

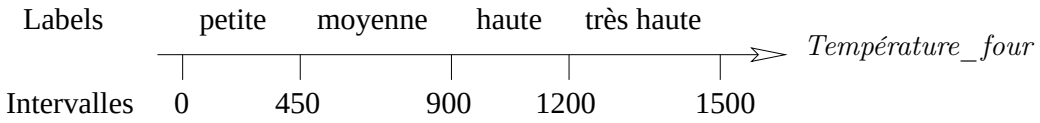


FIG. 3.2 – Exemple d'association de labels aux intervalles

3.3 Tables de compatibilité continues

Nous présentons dans cette section la définition des tables de compatibilité continues n-aires, ainsi que les méthodes de filtrage plus ou moins locales qui leur sont associées.

3.3.1 Définition

Les tables de compatibilité continues, par définition, ne lient que des variables continues et ne peuvent être exprimées qu'en intension. Elles permettent de répertorier les combinaisons d'intervalles autorisées pour des variables continues.

Le tableau 3.4 présente une contrainte de compatibilité continue binaire décrite en intension. Elle lie les plages de températures d'un four en degrés Celsius, variable numérique continue *Température_four* de domaine $\{[0, 1000]\}$, au laps de temps nécessaire pour que la microstructure de la pièce chauffée soit en phase austénitique en minutes, variable numérique continue *Temps_de_chauffe* de domaine $\{[70, 200]\}$. Plus la température de chauffe est élevée, moins il faudra de temps à l'acier pour atteindre sa phase austénitique.

TAB. 3.4 – Exemple de table de compatibilité continue

Température_four	*Temps_de_chauffe*
< 700	> 300
$[700, 850]$	$[180, 300]$
$[850, 1000]$	$[120, 180]$
> 1000	$[60, 120]$

3.3.2 Filtrage

L'objectif principal des techniques de filtrage est la détection des affectations partielles localement incohérentes. Lorsque les variables sont continues, il n'est pas possible de tester toutes les valeurs présentes dans leur domaine afin d'éliminer celles qui ne peuvent mener à une solution.

Pour déceler les intervalles continus incohérents et les éliminer des domaines de variables, sans passer par une discrétisation à base de labels, nous proposons de considérer directement les intervalles définis dans les tuples t des tables de compatibilité continues $C_{(i,j)}$, noté $t_{[.]}^{C_{(i,j)}}$. Il n'y a, en effet, aucun autre moyen de délimiter de manière raisonnée les intervalles à tester.

La phase de filtrage de ce type de contraintes continues consiste à supprimer les tuples des tables de compatibilité $C_{(i,j)}$ qui n'appartiennent pas à une solution. Un tuple t d'une contrainte $C_{(i,j)}$ contient des solutions si et seulement si pour toutes les variables V_j de $C_{(i,j)}$, l'intervalle du tuple t correspondant à la variable V_j, noté $t_{[V_j]}^{C_{(i,j)}}$ et le domaine de validité courant de la variable, noté D_{V_j} ne sont pas disjoints. Si l'intersection de $t_{[V_j]}^{C_{(i,j)}}$ et de D_{V_j} est vide, la variable ne peut prendre aucune valeur de l'intervalle $t_{[V_j]}^{C_{(i,j)}}$. La totalité du tuple t devient alors incohérente car il ne contient pas de solutions.

Après avoir testé la cohérence de tous les tuples et avoir éliminé les incohérents, nous devons reconstruire les domaines de validité des variables filtrées. Pour cela, nous réalisons pour chaque variable V_j de la contraintes $C_{(i,j)}$ l'union des tuples compatibles pour cette variable, $t_{[V_j]}^{C_{(i,j)}}$, intersectée avec le domaine courant D_{V_j} de la variable V_j. Les intervalles incohérents sont ainsi supprimés des domaines des variables.

3.3.2.1 Arc-cohérence

L'arc-cohérence permet de supprimer des domaines des variables les valeurs incompatibles avec l'atteinte d'une solution. Lorsque les variables sont continues, il n'y a aucun moyen de savoir comment partitionner leur domaine en intervalles de manière rationnelle pour les tester. Nous proposons pour les délimiter de considérer directement les intervalles présents dans les contraintes $C_{(i,j)}$, notés $t_{[variable]}^{C_{(i,j)}}$. Ce sont ces intervalles qui vont définir les intervalles à vérifier.

Nous proposons une définition de l'arc-cohérence continue sur intervalles.

Définition 7 : *arc-cohérence continue*

Un CSP *est arc-cohérent continu si, pour tout couple de variables (V_i, V_j), de domaines respectifs D_{V_i} et D_{V_j} et pour toutes les contraintes continues $C_{(i,j)}$ liant V_i et V_j prises une à une, et pour tout intervalle $t_{D_{V_i}} = t_{[V_i]}^{C_{(i,j)}}$ du domaine de la variable V_i, il existe au moins un intervalle $t_{D_{V_j}}$ du domaine de la variable V_j tel que $C_{(i,j)}$ soit cohérente.*

Nous proposons donc une variante de l'algorithme Revise, nommée Revise_C[1] présentée par l'algorithme 3 page suivante, permettant de détecter et de supprimer les intervalles qui ne sont pas arc-cohérents des domaines des variables continues.

Nous proposons une variante de l'algorithme AC3, nommée AC3_C et présentée par l'algorithme 4 page suivante, qui permet de filtrer par arc-cohérence des tables de compatibilité continues en utilisant la fonction Revise_C. Dans cette procédure, seules les variables dont le domaine est réduit subissent à nouveau un filtrage.

Dans le cas de la contrainte présentée par le tableau 3.4, le filtrage par arc-cohérence fournira les domaines :

[1] C pour continu

Alg. 3 REVISE_C(CONTRAINTE : $C_{(i,j)}$)

– $\therefore$ – ***Cet algorithme filtre une contrainte continue*** $C_{(i,j)}$ ***de type table de compatibilité par arc-cohérence.***
– $\therefore$ – *reduction indique si le domaine d'une variable a été réduit.*
reduction $\leftarrow$ Faux
Pour toutes les variables V_i de $C_{(i,j)}$ **Faire**
  Pour tous les tuples t de $C_{(i,j)}$ **Faire**
    – $\therefore$ – $t_{[V_i]}^{C_{(i,j)}}$ *correspond à l'intervalle du tuple de la contrainte* $C_{(i,j)}$ *concernant la variable* V_i
    – $\therefore$ – *Il faut tester si au moins une valeur de* $t_{[V_i]}^{C_{(i,j)}}$ *est inclue dans le domaine* D_{V_i} *de* V_i
    Si $(t_{[V_i]}^{C_{(i,j)}} \cap D_{V_i} = \emptyset)$ **Alors**
      – $\therefore$ – *Si l'intersection est vide,* V_i *ne peut prendre aucune valeur dans* $t_{[V_i]}^{C_{(i,j)}}$ *et le tuple est supprimé*
      $t^{C_{(i,j)}}$ est incohérent
      reduction $\leftarrow$ Vrai
    Sinon
      $t^{C_{(i,j)}}$ est cohérent
    Fin Si
  Fin Pour
Fin Pour
– $\therefore$ – *Il faut reconstruire le domaine de toutes les variables* V_i *si des tuples sont incohérents*
Si (*reduction* = Vrai) **Alors**
  Pour toutes les variables V_i de $C_{(i,j)}$ **Faire**
    – $\therefore$ – *Nous réalisons l'union des tuples cohérents*
    domaine_coherent $\leftarrow \bigcup\{t_{[V_i]}^{C_{(i,j)}} \text{cohérent}\}$
    – $\therefore$ – *Nous intersectons le domaine cohérent trouvé avec celui de la variable* V_i
    $D_{V_i} \leftarrow D_{V_i} \cap$ *domaine_coherent*
  Fin Pour
  Retourner l'ensemble des variables V_i de $C_{(i,j)}$
Sinon
  – $\therefore$ – *Si aucun domaine de variable n'est réduit, la fonction retourne l'ensemble vide.*
  Retourner $\emptyset$
Fin Si

Alg. 4 AC3_C(CSP : G)

– $\therefore$ – ***Cet algorithme filtre un CSP par arc-cohérence.***
– $\therefore$ – V = *liste des variables de* G
$V \leftarrow \{V_i\} \in variables(G)$
Tant Que $(V \neq \emptyset)$ **Faire**
  Dépiler une variable (V_i) de V
  Pour toutes les contraintes $C_{(i,j)} \in arcs(G)$ **Faire**
    $V \leftarrow$ (REVISE_C($C_{(i,j)}$) $\cup\, V$) (algorithme 3)
  Fin Pour
Fin Tant Que

– $\{[700, 1000]\}$ pour la variable *Température_four* de domaine initial $\{[0, 1000]\}$

$$\begin{aligned} D_{Température\_four} &= \bigcup\{t^C \text{cohérents}\} \cap D_{Température\_four} \\ &= [700, 1000] \cap [0, 1000] \end{aligned}$$

– $\{[120, 200]\}$ pour la variable *Temps_de_chauffe* de domaine initial $\{[70, 200]\}$

$$\begin{aligned} D_{Temps\_de\_chauffe} &= \bigcup\{t^C \text{cohérents}\} \cap D_{Temps\_de\_chauffe} \\ &= [120, 300] \cap [70, 200] \end{aligned}$$

ainsi que la table de compatibilité continue présentée dans le tableau 3.5.

TAB. 3.5 – Exemple de table de compatibilité continue filtrée par AC3_C

Cohérent	*Température_four*	*Temps_de_chauffe*	
Non	< 700	$> \mathbf{300}$	car $]300, +\infty[\cap [70, 200] = \emptyset$
Oui	$[700, 850]$	$[180, 300]$	
Oui	$[850, 1000]$	$[120, 180]$	
Non	$> \mathbf{1000}$	$[60, 120]$	car $]1000, +\infty[\cap [0, 1000] = \emptyset$
	{[700, 1000]}	*{[120, 300]}*	

L'ensemble des tuples cohérents diminue au fil des choix de conception. Il est donc intéressant de marquer les tuples devenus incohérents pour éviter de les tester à nouveau lors d'une nouvelle réduction de domaines. Les tuples possèdent donc un drapeau qui indique si ceux-ci sont cohérents (valeur du drapeau à 1) ou non (valeur du drapeau à 0) avec le problème courant. Seuls les tuples marqués cohérents (drapeau à 1) sont alors exploités dans les méthodes de filtrage présentées, puisqu'ils peuvent contenir des solutions. La suppression de tuples incohérents se fait par la mise à zéro de leur drapeau.

3.3.2.2 K-cohérence

La notion de k-cohérence ne varie pas de celle définie pour les tables de compatibilité discrètes. La k-cohérence teste toujours si l'affectation cohérente de $k-1$ variables peut être étendue à une $k^{\text{ième}}$ variable. Dans le cas de variables continues, les intervalles de valeurs affectés aux variables proviennent directement des contraintes et non des domaines de validité.

3.3.3 Application aux tables de compatibilité discrètes et mixtes

Les méthodes de filtrage associées aux tables de compatibilité continues peuvent être appliquées aux tables de compatibilité discrètes et mixtes dans le cas où C^{Con} est une liste d'intervalles. Les valeurs des tuples discrets (symboliques ou numériques) sont considérées comme des domaines singletons. Le filtrage se fait alors sur les domaines des tuples et non plus sur les valeurs des domaines de définition.

Il y a deux intérêts majeurs à utiliser la procédure AC3_C sur toutes les classes de tables de compatibilité :

– seules les valeurs présentes dans les contraintes sont testées et non toutes celles des domaines de définition, ce qui est intéressant pour les tables de compatibilité discrètes décrites en intension,

– et dans le cas de tables de compatibilité mixtes, le passage continu-discret via les labels devient inutile pour les variables continues,

Le filtrage par arc-cohérence AC3_C de la contrainte discrète présentée par le tableau 3.6 donne le même résultat qu'un filtrage par arc-cohérence AC3 lorsque la variable *Four* est réduite à {Basse pression, Haute pression}. La variable *Classe_de_four* est alors réduite à $\{[3,5],[9,10]\}$.

TAB. 3.6 – Exemple de table de compatibilité discrète filtrée par AC3_C

Cohérent	*Four*	*Classe_de_four*
Oui	Basse pression	[3, 5]
Non	**Atmosphérique**	[7, 9]
Oui	Haute pression	[9, 10]
	{ Basse pression, haute pression }	*{[3, 5], [9, 10] }*

Le filtrage par arc-cohérence AC3_C de la contrainte mixte, présentée dans la sous-section 3.2.1, donnera le même résultat qu'un filtrage par arc-cohérence AC3 après la double conversion continu-discret et discret-continu.

3.4 Synthèse

Les contraintes de type table de compatibilité permettent de lister les combinaisons de valeurs autorisées. Il existe trois classes de tables de compatibilité pouvant être filtrées deux méthodes de filtrage par arc-cohérence :

– les tables discrètes qui portent sur des variables symboliques ou numériques discrètes. Dans ce cas, la procédure AC3 filtre en prenant les valeurs à tester dans les domaines de variables. La procédure AC3_C, que nous proposons pour filtrer les contraintes continues, fournit les mêmes résultats en puisant directement les valeurs à tester dans les contraintes.

– les tables mixtes qui portent à la fois sur des variables discrètes et continues. Dans ce cas, la procédure AC3 filtre après avoir converti les intervalles en labels. Une fois le filtrage discret réalisé, une fonction permet de reconstruire les intervalles encore cohérents avec le problème courant. La procédure AC3_C fournit les mêmes résultats en testant directement les valeurs des contraintes, sans passer par la conversion continu-discret des variables continues.

– les tables de compatibilité continues qui portent uniquement sur des variables continues. Dans ce cas, la procédure AC3 filtre après l'association de labels aux variables continues. La procédure AC3_C fournit les mêmes résultats en puisant directement les intervalles à tester dans les contraintes.

Dans une problématique d'aide à la conception interactive, le choix des procédures de filtrage à utiliser est délicat : il faut arriver à trouver le bon compromis entre le degré de filtrage et le temps de filtrage. Dans notre application, nous privilégions l'interactivité du système d'aide à la conception et nous optons pour un filtrage par arc-cohérence par rapport à un filtrage de plus haut degré.

Le tableau 3.7 récapitule les procédures de filtrage par arc-cohérence, AC3 et AC3_C, utilisables en fonction des classes de table de compatibilité.

TAB. 3.7 – Méthodes de filtrage et classes de tables de compatibilité

	Discrète	Mixte	Continue
AC3	oui	oui avec labels	oui avec labels
AC3_C	oui avec singleton	oui avec singleton	oui

Une seule et unique méthode de filtrage peut donc être utilisée sur l'ensemble des tables de compatibilité : soit avec la procédure AC3 en convertissant les intervalles en labels et inversement, soit avec la procédure AC3_C sans devoir réaliser le moindre changement.

Dans le cadre de nos travaux, nous optons pour la méthode de filtrage par arc-cohérence AC3_C pour :

- éviter de tester toutes les valeurs des domaines de tables de compatibilité discrètes décrites en intension,
- éviter le passage continu-discret pour les variables continues.

Il serait intéressant de comparer les performances des deux méthodes de filtrage AC3 et AC3_C sur un ensemble de problèmes faisant intervenir les trois classes de tables de compatibilité. Il est probable que si les contraintes discrètes présentent un nombre de tuples supérieur au cardinal des domaines de définition des variables discrètes, la procédure AC3 devrait être plus rapide que la procédure AC3_C. Tandis que si le cardinal des domaines est supérieur au nombre de tuples des contraintes, c'est la procédure AC3_C qui devrait être la plus rapide.

Chapitre 4

Expressions mathématiques et filtrage par 2B-cohérence

Les contraintes numériques permettent de prendre en compte dans les problèmes de satisfaction de contraintes les expressions mathématiques qui sont nécessaires à la représentation de problèmes réels de conception. Plusieurs méthodes de filtrage sur ce type de contraintes ont été développées dans le but d'éliminer le plus grand nombre possible de valeurs incohérentes. La première idée fut d'appliquer les idées de l'arc-cohérence (Waltz, 1975) aux contraintes numériques, approche qui se révéla peu adaptée aux domaines continus (Davis, 1987). Il existe depuis lors deux grands courants : les méthodes de filtrage basées sur l'arithmétique des intervalles et celles basées sur une discrétisation de l'espace de recherche. Cette discrétisation est représentée sous forme d'un ensemble d'hyperboîtes organisées en arbres appelés 2^k arbres. Ce sont les propriétés et l'arité des expressions mathématiques qui vont influencer le choix d'une méthode. Ce chapitre se concentre sur les méthodes de filtrage basées sur l'arithmétique des intervalles adaptées aux fonctions numériques possédant des propriétés de *projetabilité*, de monotonie et d'arité quelconque. Nous exposerons les méthodes à base de 2^k arbres dans le chapitre 5 qui sont, elles, plus adaptées aux fonctions numériques non monotones et définies par morceaux mais d'arité limitée.

Hyvönen (1989) fut le premier à proposer l'utilisation de l'arithmétique des intervalles pour raisonner et filtrer les contraintes numériques. L'arithmétique des intervalles introduite par Moore (1966) étend l'arithmétique des réels aux intervalles pour gérer les incertitudes et pour contrôler les erreurs mathématiques, physiques (imprécisions des mesures) et, par conséquent, informatiques (approximations des calculs dues à l'utilisation de flottants). Plusieurs méthodes de filtrage, telles que la 2B-cohérence, la Box-cohérence, la 3B-cohérence, la Bound-cohérence ou la kB-cohérence, utilisent cette extension de l'arithmétique des réels vers l'arithmétique des intervalles. Certaines méthodes nécessitent la propriété de *projetabilité*, c'est-à-dire que toutes les variables d'une contrainte sont exprimables les unes en fonction des autres.

Nous allons, dans une première section, présenter les notations admises par la communauté *intervalle* et exposer le principe de l'arithmétique des intervalles, ainsi que les répercussions sur le filtrage du passage des réels vers les intervalles. Puis, nous comparerons, dans la deuxième section, les différentes méthodes de filtrage des contraintes numériques et justifierons notre

choix par rapport à leurs degrés et puissances de filtrage. Dans la troisième section, nous présenterons plus en détail la méthode retenue, le filtrage par 2B-cohérence dont nous préciserons les réquisits et les limites d'utilisation. Nous proposerons, dans la dernière section, une extension de l'arithmétique des intervalles permettant le filtrage de variables de domaines multi-intervalles.

4.1 Arithmétique des intervalles

Nous présenterons, dans un premier temps, les notations et les concepts utilisés pour définir l'arithmétique des intervalles (Moore, 1966). Puis, nous exposerons le principe de l'arithmétique des intervalles, à savoir l'extension de fonctions réelles aux fonctions portant sur des intervalles par la surdéfinition des opérateurs réels aux bornes des intervalles. Enfin, nous présenterons les conséquences de l'adaptation de l'arithmétique des réels à celle des intervalles en termes d'impact sur l'intervalle résultat fourni.

4.1.1 Notations et définitions

La communauté *intervalle* a opté pour une notation, issue de Kearfott (1996), que nous allons utiliser. En particulier :

- les intervalles sont spécifiés en $\boldsymbol{gras}$,
- les minuscules représentent des scalaires ou des intervalles,
- les majuscules représentent des vecteurs de scalaires ou d'intervalles,
- la borne supérieure (resp. inférieure) d'un intervalle $\boldsymbol{x}$ est notée $\overline{x}$ (resp. $\underline{x}$),
- $\tilde{x}$ est une valeur quelconque de l'intervalle $\boldsymbol{x}$.

Nous utiliserons aussi les notations suivantes :

- $\mathbb{R}^\infty = \mathbb{R} \cup \{-\infty, +\infty\}$: ensemble des réels augmenté des deux symboles d'infinité,
- $\mathbb{I}$ représente l'ensemble des intervalles de réels. $\mathbb{I}$ est ordonné partiellement par l'inclusion ensembliste,
- f, g sont des fonctions sur les réels,
- $\boldsymbol{f}, \boldsymbol{g}$ sont des fonctions sur les intervalles,
- $c : \mathbb{R}^n \rightarrow \mathit{Bool}$ est une contrainte sur les réels,
- $\boldsymbol{c} : \mathbb{I}^n \rightarrow \mathit{Bool}$ est une contrainte sur les intervalles.

Définition 8 : *intervalle*

Un intervalle de réels $\boldsymbol{x} = [\underline{x}, \overline{x}]$ *avec* $\underline{x}$ *et* $\overline{x} \in \mathbb{R}^\infty$ *est l'ensemble des nombres réels* r *tels que* $\{r \in \mathbb{R}^\infty \mid \underline{x} \leq r \leq \overline{x}\}$ *; si* $\underline{x}$ *ou* $\overline{x}$ *est un des symboles* $-\infty$ *ou* $+\infty$*, alors* $\boldsymbol{x}$ *est un intervalle ouvert.*

Définition 9 : *enveloppe*

Soit S *un sous-ensemble de* $\mathbb{R}$*. L'enveloppe* (Hull) *de* S*, dénotée* $\square S$*, est le plus petit intervalle* $\boldsymbol{i}$ *tel que* $S \subseteq \boldsymbol{i}$*.*

4.1.2 Bases du calcul d'intervalles

L'arithmétique des intervalles est basée sur l'extension de fonctions réelles aux bornes d'intervalles. Alors qu'une fonction numérique sur les réels f fait correspondre à un ou plusieurs réels un autre réel, la fonction numérique sur les intervalles $\boldsymbol{f}$ fait correspondre à un ou plusieurs intervalles un autre intervalle.

$$\begin{array}{rccc} f: & \mathbb{R}^n & \rightarrow & \mathbb{R} \\ \boldsymbol{f} & \mathbb{I}^n & \rightarrow & \mathbb{I} \end{array}$$

Si un réel $\tilde{x}$ appartient à un intervalle $\boldsymbol{x}$, alors l'image de $\tilde{x}$ par la fonction f doit appartenir à l'image de $\boldsymbol{x}$ par la fonction $\boldsymbol{f}$ qui est l'extension aux intervalles de f.

Définition 10 : ***extension d'une fonction***

$\boldsymbol{f} : \mathbb{I}^n \rightarrow \mathbb{I}$ est une extension aux intervalles de $f : \mathbb{R}^n \rightarrow \mathbb{R}$ si et seulement si $\forall(\boldsymbol{x_1}, \ldots, \boldsymbol{x_n}) \in \mathbb{I}^n : f(\tilde{x}_1, \ldots, \tilde{x}_n) \in \boldsymbol{f(x_1, \ldots, x_n)}$.

4.1.2.1 Extension optimale de fonctions

Pour obtenir le plus petit intervalle englobant toutes les images de $\tilde{x}$ par une fonction réelle f (ou enveloppe), Moore (1966) définit l'extension optimale de fonctions aux intervalles.

Définition 11 : ***extension optimale d'une fonction***

L'extension $\boldsymbol{f}$ d'une fonction f aux intervalles est optimale si et seulement si $\forall \boldsymbol{X} \in \mathbb{I}^n, \boldsymbol{f(X)} = \square(f(X) \mid X \in \boldsymbol{X})$

Des extensions optimales existent pour la plupart des fonctions élémentaires $(+, -, /, \times, e, etc)$. Nous présentons ici celles des opérateurs arithmétiques et d'une fonction élémentaire, la fonction exponentielle, aux bornes des intervalles :

Addition

$$\begin{array}{rccl} f: & (x, y) & \mapsto & x + y \\ \boldsymbol{f}: & \boldsymbol{(x, y)} & \mapsto & \boldsymbol{x} \oplus \boldsymbol{y} = [\underline{x} + \underline{y}, \overline{x} + \overline{y}] \end{array}$$

Soustraction

$$\begin{array}{rccl} f: & (x, y) & \mapsto & x - y \\ \boldsymbol{f}: & \boldsymbol{(x, y)} & \mapsto & \boldsymbol{x} \ominus \boldsymbol{y} = [\underline{x} - \overline{y}, \overline{x} - \underline{y}] \end{array}$$

Multiplication

$$\begin{array}{rccl} f: & (x, y) & \mapsto & x \times y \\ \boldsymbol{f}: & \boldsymbol{(x, y)} & \mapsto & \boldsymbol{x} \otimes \boldsymbol{y} = [min(\overline{x} \times \overline{y}, \underline{x} \times \overline{y}, \overline{x} \times \underline{y}, \underline{x} \times \underline{y}), max(\overline{x} \times \overline{y}, \underline{x} \times \overline{y}, \overline{x} \times \underline{y}, \underline{x} \times \underline{y})] \end{array}$$

Division

$$\begin{array}{lrcl} f: & (x,y) & \mapsto & x/y \\ \boldsymbol{f}: & \boldsymbol{(x,y)} & \mapsto & \boldsymbol{x} \oslash \boldsymbol{y} = [min(\overline{x}/\overline{y}, \underline{x}/\overline{y}, \overline{x}/\underline{y}, \underline{x}/\underline{y}), max(\overline{x}/\overline{y}, \underline{x}/\overline{y}, \overline{x}/\underline{y}, \underline{x}/\underline{y})] \end{array}$$

Exponentielle

$$\begin{array}{lrcl} f: & x & \mapsto & e^x \\ \boldsymbol{f}: & \boldsymbol{x} & \mapsto & \boldsymbol{e^x} = [e^{\underline{x}}, e^{\overline{x}}] \end{array}$$

Nous noterons $\oplus$, $\ominus$, $\otimes$, $\oslash$ et $\boldsymbol{e}$ les extensions optimales aux intervalles de $+$, $-$, $\times$, $/$ et *exponentielle*. Le cas de la division par un intervalle contenant 0 peut être résolu en utilisant les *intervalles étendus*. Ceux-ci définissent des intervalles infinis ou semi-infinis contenant $-\infty$ ou $+\infty$. Toutes les fonctions élémentaires doivent être surdéfinies pour prendre en compte les *intervalles étendus*.

4.1.2.2 Extension naturelle de fonctions

Les extensions naturelles aux intervalles de fonctions quelconques sont obtenues à partir des extensions optimales des fonctions usuelles. L'idée intuitive concernant l'extension d'une fonction consiste à remplacer chaque constante par le plus petit intervalle la contenant, chaque opérateur par son extension optimale aux intervalles et chaque variable x par une variable aux intervalles $\boldsymbol{x}$.

Définition 12 : ***extension naturelle d'une fonction***

$\boldsymbol{f}$ est une extension naturelle aux intervalles de f si $\boldsymbol{f}$ est obtenue en remplaçant dans f chaque constante k par le plus petit intervalle $\boldsymbol{k}$ contenant k, chaque variable x par une variable aux intervalles $\boldsymbol{x}$ et chaque opération arithmétique par son extension optimale aux intervalles.

Par exemple, soit $f(x) = e^x + x \times y + 3$, une fonction sur les réels. Son extension naturelle aux intervalles est définie par $\boldsymbol{f(x)} = \boldsymbol{e^x} \oplus \boldsymbol{x} \otimes \boldsymbol{y} \oplus \boldsymbol{3}$.

Les extensions naturelles des fonctions ne correspondent pas forcément à leur extensions optimales, car elles peuvent surestimer l'intervalle résultat.

Par exemple, soit $f(x) = x^2 - x$ une fonction sur les réels. Son extension naturelle aux intervalles est définie par $\boldsymbol{f} = \boldsymbol{x^2} \ominus \boldsymbol{x}$. Le calcul de $\boldsymbol{f}$ sur $\boldsymbol{x} = [0,2]$ fournit l'intervalle résultat $[-2,4]$:

$$\begin{array}{lccl} f: & \mathbb{R} & \rightarrow & \mathbb{R} \\ f: & x & \mapsto & x^2 - x \\ \boldsymbol{f}: & \mathbb{I} & \rightarrow & \mathbb{I} \\ \boldsymbol{f}: & \boldsymbol{x} & \mapsto & \boldsymbol{x^2} \ominus \boldsymbol{x} \\ \boldsymbol{f}: & D_x & \mapsto & D_x^2 \ominus D_x \\ \boldsymbol{f}: & [0,2] & \mapsto & [0,2]^2 \ominus [0,2] \\ \boldsymbol{f}: & [0,2] & \mapsto & [0,4] \ominus [0,2] = [-2,4] \end{array}$$

Or, l'intervalle $[-2,4]$ ne correspond pas à l'enveloppe englobant toutes les images de x par la fonction réelle f.

4.1.2.3 Extension de contraintes

L'extension de contraintes a été définie de manière similaire à celle des fonctions par Hansen (1992). L'extension d'une contrainte consiste à remplacer chaque constante par le plus petit intervalle la contenant, chaque opérateur par son extension optimale aux intervalles et chaque variable x par son intervalle de validité $\boldsymbol{x}$.

Définition 13 : ***extension de contraintes***

$\boldsymbol{c} : \mathbb{I}^n \rightarrow$ *Bool est une extension aux intervalles de* $c : \mathbb{R}^n \rightarrow$ *Bool si et seulement si* $\forall(\boldsymbol{x_1}, \ldots, \boldsymbol{x_n}) \in \mathbb{I}^n : c(\tilde{x_1}, \ldots, \tilde{x_n}) \Rightarrow \boldsymbol{c}(\boldsymbol{x_1}, \ldots, \boldsymbol{x_n})$

4.1.3 Conséquences du passage de l'arithmétique des réels vers l'arithmétique des intervalles

L'extension de fonctions numériques réelles aux intervalles a deux conséquences majeures. D'une part, les opérateurs $+$ et $\times$ gardent leurs propriétés de commutativité et d'associativité, mais perdent celle de distributivité[1]. La forme syntaxique de l'expression mathématique influe donc sur l'intervalle résultat. D'autre part, la présence d'occurrences multiples de variables entraîne une décorrelation de celles-ci.

4.1.3.1 Forme syntaxique de la fonction

La forme syntaxique d'une fonction a une très forte influence sur l'intervalle résultat fourni car la distributivité des opérateurs arithmétiques réels n'est pas conservée par leur extension aux intervalles.

L'exemple suivant illustre le fait que la forme syntaxique de la fonction réelle influe sur son extension aux intervalles et sur le plus petit intervalle résultat trouvé. Soient x et y deux variables réelles de domaine $D_x = D_y = \{[0, 5]\}$. Soient f une fonction numérique réelle qui associe à $(x, y) \mapsto x \times y - x$ et $\boldsymbol{f}$ son extension aux intervalles qui associe à $(D_x, D_y) \mapsto D_x \otimes D_y \ominus D_x$ et g une fonction numérique réelle qui associe à $(x, y) \mapsto x \times (y - 1)$ et $\boldsymbol{g}$ son extension aux intervalles qui associe à $(D_x, D_y) \mapsto D_x \otimes (D_y \ominus \mathbf{1})$.

$$
\begin{array}{lccl}
f: & \mathbb{R}^2 & \rightarrow & \mathbb{R} \\
f: & (x, y) & \mapsto & x \times y - x \\
\boldsymbol{f}: & \mathbb{I}^2 & \rightarrow & \mathbb{I} \\
\boldsymbol{f}: & \boldsymbol{(x, y)} & \mapsto & \boldsymbol{x \otimes y \ominus x} \\
\boldsymbol{f}: & (D_x, D_y) & \mapsto & D_x \otimes D_y \ominus D_x \\
\boldsymbol{f}: & ([0, 5], [0, 5]) & \mapsto & [0, 5] \otimes [0, 5] \ominus [0, 5] = [-5, 25]
\end{array}
$$

[1] Un opérateur $\odot$ est distributif (à gauche et à droite) sur un opérateur $\star$ si pour tout x, y, z, la propriété suivante est vérifiée : $x \odot (y \star z) = (x \odot y) \star (x \odot z)$ pour la distributivité à gauche et $(y \star z) \odot x = (y \odot x) \star (z \odot x)$ pour la distributivité à droite.

$$
\begin{array}{lccl}
g: & \mathbb{R}^2 & \rightarrow & \mathbb{R} \\
g: & (x,y) & \mapsto & x \times (y-1) \\
\boldsymbol{g}: & \mathbb{I}^2 & \rightarrow & \mathbb{I} \\
\boldsymbol{g}: & \boldsymbol{(x,y)} & \mapsto & \boldsymbol{x \otimes (y \ominus 1)} \\
\boldsymbol{g}: & (D_x, D_y) & \mapsto & D_x \otimes (D_y \ominus 1) \\
\boldsymbol{g}: & ([0,5],[0,5]) & \mapsto & [0,5] \otimes ([0,5] \ominus [1,1]) = [-5,20]
\end{array}
$$

Les fonctions f et g sont équivalentes sur $\mathbb{R}^2$, mais leur extension aux intervalles $\boldsymbol{f}$ et $\boldsymbol{g}$ ne fournit pas les mêmes résultats sur $\mathbb{I}^2$: $\boldsymbol{f}([0,5],[0,5]) = [-5,25]$ et $\boldsymbol{g}([0,5],[0,5]) = [-5,20]$.

4.1.3.2 Multi-occurrences de variables

La présence d'occurrences multiples de variables dans une expression mathématique réelle a un impact sur son extension aux intervalles. En effet, le calcul par intervalles gère toutes les occurrences comme des variables différentes ayant le même domaine. C'est le problème de la décorrélation des occurrences d'une même variable (Beaumont, 1999). Dès lors, le choix d'une valeur dans le domaine d'une variable pour la première occurrence est indépendant du choix pour la seconde.

L'exemple suivant illustre le fait que la présence d'occurrences multiples de variables est gérée comme un problème où les variables sont indépendantes. Soit x une variable réelle de domaine $D_x = \{[-1,1]\}$. Soient f une fonction réelle qui associe à la variable $x \mapsto x - x$ et $\boldsymbol{f}$ son extension aux intervalles qui associe à l'intervalle $D_x \mapsto D_x \ominus D_x$. La fonction f ne peut être égale qu'à 0. Or, le résultat fourni par l'arithmétique des intervalles est tout autre :

$$
\begin{array}{lccl}
f: & \mathbb{R} & \rightarrow & \mathbb{R} \\
f: & x & \mapsto & x - x \\
\boldsymbol{f}: & \mathbb{I} & \rightarrow & \mathbb{I} \\
\boldsymbol{f}: & \boldsymbol{x} & \mapsto & \boldsymbol{x \ominus x} \\
\boldsymbol{f}: & D_x & \mapsto & D_x \ominus D_x \\
\boldsymbol{f}: & [-1,1] & \mapsto & [-1,1] \ominus [-1,1] = [-2,2]
\end{array}
$$

4.1.4 Synthèse

L'arithmétique des intervalles étend les notions de fonctions numériques sur réels aux fonctions numériques sur intervalles. Elle permet de réduire les variables numériques continues contraintes par des expressions mathématiques. Mais ce passage réel vers intervalle n'est pas sans conséquence :

- la forme syntaxique de la fonction numérique f sur intervalles influe sur l'intervalle résultat fourni, ce dernier contenant toujours l'enveloppe de la fonction $\Box f$,
- la présence d'occurrences multiples de variable est gérée comme des variables différentes ayant le même domaine.

4.2 Méthodes de filtrage

Nous comparons dans cette section plusieurs méthodes de filtrage basées sur l'arithmétique des intervalles en termes de puissance et de degré de filtrage. Nous définirons dans un premier temps la propriété de *projetabilité* nécessaire à certaines méthodes puis, nous présenterons succinctement les différentes méthodes de propagation et nous indiquerons la méthode retenue.

4.2.1 Projetabilité et décomposition

Il existe plusieurs méthodes de filtrage sur les fonctions numériques basées sur l'arithmétique des intervalles. Certaines nécessitent la projection des contraintes, notée $\pi(c)$, sur chacune de leurs variables, notée $\pi_{var}(c)$.

Considérons la contrainte ternaire suivante $y = x + z$. Cette contrainte est projetable sur chacune de ses variables de la manière suivante :

$$\begin{array}{ll} \pi_y(c): & y = x + z \\ \pi_x(c): & x = y - z \\ \pi_z(c): & z = y - x \end{array}$$

Il n'est pas toujours possible de projeter les contraintes sur chacune des variables, en particulier si l'une des variables a des occurrences multiples ou si la contrainte n'est pas monotone[2].

Une solution consiste alors à décomposer les contraintes en fonctions primitives binaires ou ternaires monotones par l'introduction de nouvelles variables (Lhomme et Rueher, 1997). La décomposition d'un système de contraintes $\mathbb{SC}$ en un système décomposé contenant des contraintes primitives $\mathbb{SC}_{decomp}$ ne change pas la sémantique du problème. La présence d'occurrences multiples de variables engendre une prise en compte indépendante de celles-ci (ce qui revient à définir deux variables portant sur le même domaine).

Par exemple, considérons la contrainte binaire suivante : $y = x \times \ln(x)$ où $D_y = \{[0, 10]\}$ et $D_x = \{[2, 5]\}$. Celle-ci présente une occurrence multiple pour la variable x et n'est pas projetable sur x. Nous posons donc la variable z telle que $z = \ln(x)$ et $D_z = \{[\ln(2), \ln(5)]\}$. Le système décomposé $\mathbb{SC}_{decomp}$ est donc composé de deux contraintes :

$$\mathbb{SC}_{decomp} : \left\{ \begin{array}{ll} c_1: & y = x \times z \\ c_2: & z = \ln(x) \end{array} \right.$$

Le système $\mathbb{SC}_{decomp}$ contient uniquement des contraintes qui sont projetables sur chacune de leurs variables et dont les fonctions de projection sont monotones :

$$\mathbb{SC}_{decomp} : \left\{ \begin{array}{ll} \pi_y(c_1): & y = x \times z \\ \pi_x(c_1): & x = y/z \\ \pi_z(c_1): & z = y/x \\ \pi_z(c_2): & z = \ln(x) \\ \pi_x(c_2): & x = e^z \end{array} \right.$$

[2]La monotonie des contraintes a été définie par Lhomme et Rueher (1997). Une contrainte est monotone si elle est non disjonctive et si toutes ses fonctions de projection sont monotones. Une contrainte est disjonctive si elle associe à l'une de ses variables un domaine multi-intervalles : $x^2 = y$ est disjonctive sur $D_x = \cup\{[-2, -1], [1, 2]\} \rightsquigarrow [-2, 2]$ et $D_y = [1, 4]$.

4.2.2 Comparaison de méthodes de filtrage

Il existe plusieurs méthodes de filtrage sur les contraintes numériques basées sur l'arithmétique des intervalles. Nous les exposons succinctement et les comparons en termes de puissance et de degré de filtrage.

Les méthodes de propagation sont :

2B-cohérence La 2B-cohérence ou $2B$ (Lhomme, 1993) est une approximation de l'arc-cohérence qui considère uniquement les bornes des domaines. Cette méthode nécessite des contraintes monotones projetables et par conséquent une décomposition des contraintes en contraintes primitives lors de multi-occurrences de variables.

Box-cohérence La Box-cohérence ou *Box* (Benhamou et al., 1994) est une approximation de l'arc-cohérence qui utilise une extension aux intervalles de la méthode de Newton accélérée par une forme de *Domain Splitting*. Cette méthode ne nécessite pas de décomposition des contraintes en fonctions primitives.

3B-cohérence La 3B-cohérence ou $3B$ (Lhomme, 1993) est une approximation de la 3-cohérence qui vérifie que les bornes des intervalles filtrés par 2B-cohérence appartiennent aux solutions. Cette méthode nécessite des contraintes monotones projetables et par conséquent une décomposition des contraintes en contraintes primitives lors de multi-occurrences de variables.

Bound-cohérence La Bound-cohérence ou *Bound* (Benhamou, 1996) est une généralisation de la Box-cohérence qui vérifie si les bornes des intervalles filtrés par Box-cohérence appartiennent aux solutions.

kB-cohérence La kB-cohérence est l'équivalent de la k-cohérence discrète. Ce type de filtrage possède le plus haut degré, mais reste essentiellement théorique en raison du coût élevé des calculs.

Le filtrage d'un *CSP* $P = (\mathbb{V}, \mathbb{D}, \mathbb{C})$ par une méthode de filtrage mf est noté $\Phi_{mf}(P)$. Dans ses travaux, Delobel (2000) a comparé les puissances de filtrage de ces méthodes à partir de la comparaison des *CSPs* obtenus après filtrage.

Définition 14 : *comparaison de CSP*

Un CSP $P = (\mathbb{V}, \mathbb{D}, \mathbb{C})$ *est plus petit (ou égal) qu'un* CSP $P' = (\mathbb{V}, \mathbb{D}', \mathbb{C})$ *si et seulement si* $\mathbb{D} \subseteq \mathbb{D}'$. *Dans ce cas, nous notons cette relation* $P \preceq P'$.

Considérons deux méthodes de filtrage mf_1 et mf_2. $\Phi_{mf_1}(P) \preceq \Phi_{mf_1}(P)$ indique que la méthode de filtrage mf_1 réduit plus fortement les domaines de définition des variables du problème P que la méthode mf_2.

Il en résulte (Delobel, 2000) que pour un même problème P filtré par les différentes méthodes de propagation :

- $\Phi_{2B}(P) \preceq \Phi_{Box}(P)$ pour un système ne nécessitant pas de décomposition en fonctions primitives,
- $\Phi_{2B}(P_{decomp}) \succeq \Phi_{Box}(P)$ pour un système P_{decomp} devant se décomposer en fonctions primitives pour utiliser la 2B-cohérence,

- $\Phi_{3B}(P) \preceq \Phi_{2B}(P)$ pour un système devant ou non se décomposer en fonctions primitives,
- $\Phi_{3B}(P_{decomp}) \preceq \Phi_{Box}(P)$ pour un système P_{decomp} devant se décomposer en fonctions primitives (ce qui est dû à des degrés de filtrage différents),
- $\Phi_{3B}(P_{decomp}) \succeq \Phi_{Bound}(P)$ pour un système P_{decomp} devant se décomposer en fonctions primitives,
- $\Phi_{Bound}(P) \preceq \Phi_{Box}(P)$ pour un système P quelconque.

4.2.3 Synthèse

Cette section a présenté de manière succincte plusieurs méthodes de filtrage basées sur l'arithmétique des intervalles. Rappelons que la 2B-cohérence et la 3B-cohérence nécessitent la projection des contraintes sur chacune de leurs variables et la décomposition de celles-ci en contraintes primitives lors de la présence d'occurrences multiples de variables. Les méthodes de propagation ont été comparées par Delobel (2000) :

- sur un système non décomposé : $\Phi_{3B}(P) \preceq \Phi_{2B}(P) \preceq \Phi_{Box}(P)$
- sur un système décomposé : $\Phi_{Bound}(P) \preceq \Phi_{3B}(P_{decomp}) \preceq \Phi_{Box}(P) \preceq \Phi_{2B}(P_{decomp})$

Dans le cadre de nos travaux, nous devons privilégier le côté interactif de l'outil d'aide à la conception. De plus, les contraintes numériques des modèles de connaissances ne présentent pas d'occurrences multiples de variables et sont projetables sur chacune d'entre elles. De ce fait, nous optons pour la méthode de filtrage par 2B-cohérence car :

- son degré de filtrage faible, du fait de calculs moins conséquents, est compatible avec des temps d'interaction,
- sa puissance de filtrage, meilleure à degré de filtrage équivalent, élimine le plus grand nombre de valeurs incohérentes avec l'atteinte d'une solution dans le cas où les contraintes n'ont pas besoin d'être décomposées.

Nous notons que les choix de modélisation des contraintes numériques jouent un rôle important sur l'intervalle résultat fourni après un filtrage par 2B-cohérence.

4.3 2B-cohérence

Nous présentons dans cette section la méthode de filtrage par 2B-cohérence que nous avons retenue pour notre application. Nous allons, dans un premier temps, définir ce qu'est un *CSP* 2B-cohérent. Puis nous reviendrons sur les exigences qu'impose l'utilisation de cette méthode de filtrage et nous présenterons ses limites dues en partie à l'utilisation de l'arithmétique des intervalles.

4.3.1 Définition

La 2B-cohérence se base sur l'arithmétique des intervalles pour filtrer les variables continues aux bornes de leurs domaines de définition. Lhomme (1993) fut le premier à en donner une définition formelle. Une contrainte $\boldsymbol{c}$ est 2B-cohérente si et seulement si pour toute variable x

de domaine $D_x = \{[\underline{x}, \overline{x}]\}$, il existe au moins une valeur dans le domaine des autres variables qui satisfait $\boldsymbol{c}$ lorsque x est instanciée à $\underline{x}$ puis à $\overline{x}$.

Définition 15 : *2B-cohérence (Lhomme et Rueher, 1997)*

Soient $P = (\mathbb{V}, \mathbb{D}, \mathbb{C})$ un CSP *et $\boldsymbol{c} \in \mathbb{C}$ une contrainte numérique d'arité k portant sur les variables continues $(v_1, \ldots, v_k)$. La contrainte $\boldsymbol{c}$ est 2B-cohérente si et seulement si les relations ci-dessous sont vérifiées pour toutes les variables v_i de la contrainte :*

- $\boldsymbol{c}(D_{v_1}, \ldots, D_{v_{i-1}}, \underline{v_i}, D_{v_{i+1}}, \ldots, D_{v_k})$
- $\boldsymbol{c}(D_{v_1}, \ldots, D_{v_{i-1}}, \overline{v_i}, D_{v_{i+1}}, \ldots, D_{v_k})$

Par exemple, considérons la contrainte $\boldsymbol{c}_1 : x + y = 2$ avec, dans un premier cas, $D_x = \{[0, 1]\}$ et $D_y = \{[1, 2]\}$, puis, dans un second cas, $D_x = \{[0, 1]\}$ et $D_y = \{[0, 2]\}$. Dans le premier cas, la contrainte $\boldsymbol{c}_1$ est 2B-cohérente car :

- il existe une valeur $(y = 2)$ dans le domaine de y qui satisfait $\boldsymbol{c}_1$ lorsque x est valuée à sa borne inférieure $(x = 0)$,
- il existe une valeur $(y = 1)$ dans le domaine de y qui satisfait $\boldsymbol{c}_1$ lorsque x est valuée à sa borne supérieure $(x = 1)$,
- il existe une valeur $(x = 1)$ dans le domaine de x qui satisfait $\boldsymbol{c}_1$ lorsque y est valuée à sa borne inférieure $(y = 1)$,
- et il existe une valeur $(x = 0)$ dans le domaine de x qui satisfait $\boldsymbol{c}_1$ lorsque y est valuée à sa borne supérieure $(y = 2)$.

Dans le second cas, la contrainte $\boldsymbol{c}_1$ n'est pas 2B-cohérente car il n'existe pas de valeur dans le domaine de x lorsque y est valuée à sa borne inférieure $(y = 0)$ telle que $\boldsymbol{c}_1$ soit satisfaite.

Un *CSP* $P = (\mathbb{V}, \mathbb{D}, \mathbb{C})$ est 2B-cohérent si et seulement si toutes ses contraintes prises une à une sont 2B-cohérentes.

Définition 16 : *CSP 2B-cohérent*

Soit $P = (\mathbb{V}, \mathbb{D}, \mathbb{C})$. P est 2B-cohérent si et seulement si toutes ses contraintes sont 2B-cohérentes.

4.3.2 Réquisits

L'utilisation de la 2B-cohérence requiert un certain nombre d'exigences. Elle fait partie des méthodes de filtrage qui nécessitent la propriété de projetabilité et de monotonie (cf. sous-section 4.2.1). Elle requiert donc la décomposition des contraintes en fonctions primitives lorsque la projection n'est pas réalisable du fait de la présence d'occurrences multiples de variables.

La procédure de filtrage par 2B-cohérence (Lhomme et Rueher, 1997), dérivée de l'AC3 et nommée In, est présentée par l'algorithme 5 page ci-contre. Cette procédure fait appel à la

Alg. 5 IN(ENSEMBLE DES CONTRAINTES : $\mathbb{C}$, ENSEMBLE DES VARIABLES : $\mathbb{V}$)

– $\therefore$ – ***Cet algorithme filtre un* CSP *par 2B cohérence.***
– $\therefore$ – $\mathbb{V}'$ *représente la liste des variables réduites*
$\mathbb{V}' \leftarrow \emptyset$
– $\therefore$ – *Queue représente la liste des contraintes à filtrer*
Queue $\leftarrow \mathbb{C}$
Tant Que (*Queue* $\neq \emptyset$) **Faire**
    Dépiler une contrainte numérique $\boldsymbol{c}$ de *Queue*
    – $\therefore$ – *On calcule les domaines des variables potentiellement réduites par la contrainte* $\boldsymbol{c}$
    $\mathbb{V}' \leftarrow$ NARROW$(\boldsymbol{c}, \mathbb{V})$ (algorithme 6)
    – $\therefore$ – *si des variables ont été réduites par* $\boldsymbol{c}$
    Si ($\mathbb{V}' \neq \mathbb{V}$) **Alors**
        – $\therefore$ – *On ne filtre à nouveau qu'à partir des variables réduites*
        $\mathbb{V} \leftarrow \mathbb{V}'$
        – $\therefore$ – *On empile dans Queue les contraintes qui portent sur les variables réduites de* $\mathbb{V}'$
        $Queue \leftarrow Queue \cup \{\boldsymbol{c'} \in \mathbb{C} \mid Var(\boldsymbol{c}) \cap Var(\boldsymbol{c'}) \neq \emptyset\}$
    Fin Si
Fin Tant Que

Alg. 6 NARROW(CONTRAINTE : $\boldsymbol{c}$, ENSEMBLE DES VARIABLES : $\mathbb{V}$)

– $\therefore$ – ***Cet algorithme filtre une contrainte par 2B cohérence.***
– $\therefore$ – F *est l'ensemble des fonctions de projection*
– $\therefore$ – $F_{\boldsymbol{c}}$ *est l'ensemble des fonctions de projection issues de* $\boldsymbol{c}$
– $\therefore$ – $\mathbb{V}'$ *est l'ensemble des variables réduites*
$\mathbb{V}' \leftarrow \emptyset$
– $\therefore$ – *Queue est l'ensemble des fonctions de projection de* $\boldsymbol{c}$
$Queue \leftarrow F_{\boldsymbol{c}} \subseteq F$
Tant Que (*Queue* $\neq \emptyset$) **Faire**
    Dépiler une fonction f_i de *Queue*
    – $\therefore$ – *Calcul du domaine* D_{v_i} *de* v_i *par la fonction de projection* $f_i(\mathbb{V})$
    $D_{v_i} \leftarrow D_{v_i} \cap f_i(\mathbb{V})$
    – $\therefore$ – *Si le domaine de la variable est réduit par* $f_i(\mathbb{V})$
    Si (le domaine de v_i est réduit) **Alors**
        – $\therefore$ – *On indique que la variable* v_i *a été réduite*
        $\mathbb{V}' \leftarrow \mathbb{V}' \cup \{v_i\}$
    Fin Si
Fin Tant Que
Retourner $\mathbb{V}'$

fonction NARROW, algorithme 6 page précédente, qui réduit les domaines des variables d'une contrainte $\boldsymbol{c}$, $Var(\boldsymbol{c})$ jusqu'à ce que $\boldsymbol{c}$ soit 2B-cohérente.

La procédure IN appliquée à un *CSP* défini par $\mathbb{V} = \{x, y\}$ et contenant la contrainte précédente $\mathbb{C} = \{\boldsymbol{c}_1 : x + y = 2\}$ fournit les résultats suivants sur les deux domaines de définition :

- $\mathbb{D} = \{D_x = \{[0, 1]\}, D_y = \{[1, 2]\}\}$, les intervalles de validité des variables x et y sont inchangés,
- $\mathbb{D} = \{D_x = \{[0, 1]\}, D_y = \{[0, 2]\}\}$, l'intervalle de validité de x est inchangé, celui de y est réduit à $\{[1, 2]\}$

4.3.3 Limites

Le filtrage par 2B-cohérence possède quelques limites en termes d'efficacité de filtrage. Certaines limites découlent directement de l'utilisation de l'arithmétique des intervalles, d'autres sont dues au faible degré de filtrage de la 2B-cohérence et à sa politique de gestion des intervalles résultats, alors que d'autres résultent de l'utilisation des flottants. Nous allons illustrer certaines de ces limites à partir d'exemples tirés de Rueher (2002).

4.3.3.1 Limites dues à l'arithmétique des intervalles

Certaines limites de la technique de filtrage par 2B-cohérence sont directement liées à l'utilisation de l'arithmétique des intervalles (cf. sous-section 4.1.3).

Forme de la contrainte La forme syntaxique des contraintes influe sur l'intervalle résultat fourni. Ceci est dû au passage de l'arithmétique des réels vers l'arithmétique des intervalles où les opérateurs $+$ et $\times$ perdent leur propriété de distributivité.

Multi occurence de variables Les variables d'occurrences multiples dans une contrainte sont gérées comme des variables indépendantes. Ceci est dû à la décorrélation des occurrences d'une même variable dans l'arithmétique des intervalles.

4.3.3.2 Limites dues au faible degré de filtrage

Certaines limites de la technique de filtrage par 2B-cohérence sont liées à son faible degré de filtrage et à sa politique de gestion des intervalles résultats.

Pas de prise en compte simultanée des contraintes Le filtrage par 2B-cohérence est dérivé de l'arc-cohérence. Les contraintes sont donc filtrées les unes après les autres. La non prise en compte simultanée de toutes les contraintes ne permet pas de réduire suffisamment les domaines des variables numériques.

Considérons, par exemple, le *CSP* $P_1 = (\mathbb{V}_1, \mathbb{D}_1, \mathbb{C}_1)$, illustré par la figure 4.1, où :

$$\begin{aligned} \mathbb{V}_1 = &\quad \{x, y\} \\ \mathbb{D}_1 = &\quad \{D_x = \{[0, 100]\}, D_y = \{[0, 100]\}\} \\ \mathbb{C}_1 = &\quad \{\boldsymbol{c_1} : x + y = 2, \boldsymbol{c_2} : y \leq x + 1, \boldsymbol{c_3} : y \geq 1 + \ln(x)\} \end{aligned}$$

Le filtrage $\Phi_{2B}(P_1)$ donne les domaines $D_x = \{[0, 2]\}$ et $D_y = \{[0, 2]\}$. Or, les bornes des intervalles réduits ne sont pas solutions. Il existe pourtant des combinaisons qui respectent les trois contraintes prises une à une, mais pas simultanément sur les valeurs des bornes.

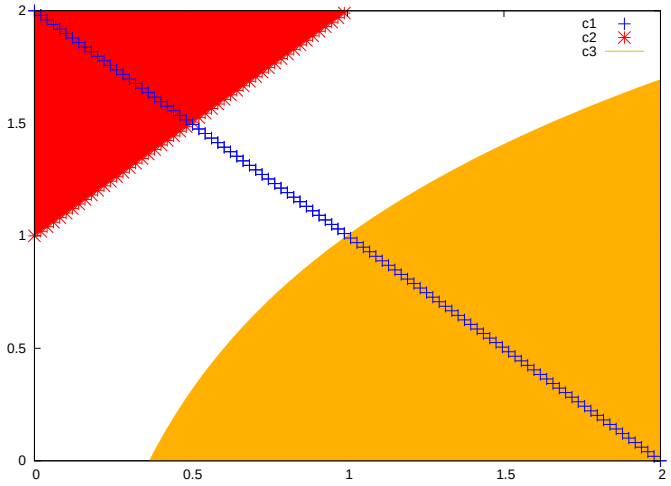


FIG. 4.1 – Problème P_1 2B-cohérent, avec bornes des intervalles non solutions

Non garantie de l'existence d'une solution Comme l'arc-cohérence, la méthode par 2B-cohérence possède un filtrage de faible degré. Elle ne permet donc pas toujours de détecter qu'un problème n'a pas de solution.

Considérons le *CSP* $P_2 = (\mathbb{V}_2, \mathbb{D}_2, \mathbb{C}_2)$ défini par le *CSP* précédent P_1 augmenté de la contrainte $\boldsymbol{c_4} : y = \frac{2\times(e^x-1)}{e^2-1}$:

$$\begin{aligned} \mathbb{V}_2 = &\quad \{x, y\} \\ \mathbb{D}_2 = &\quad \{D_x = \{[0, 100]\}, D_y = \{[0, 100]\}\} \\ \mathbb{C}_2 = &\quad \{\boldsymbol{c_1} : x + y = 2, \boldsymbol{c_2} : y \leq x + 1, \boldsymbol{c_3} : y \geq 1 + \ln(x), \boldsymbol{c_4} : y = \tfrac{2\times(e^x-1)}{e^2-1}\} \end{aligned}$$

Le problème P_2, illustré par la figure 4.2, est 2B-cohérent, mais ne possède pas de solution.

Politique de gestion des intervalles Le filtrage par 2B-cohérence nécessite la projection des contraintes sur chacune des variables. Or, le calcul de la projection d'une contrainte sur une variable peut détruire la continuité des domaines. La présence d'intervalles multiples peut être évitée en préservant la continuité des domaines. L'approche la plus utilisée consiste à approximer le domaine D_i d'une variable par son enveloppe englobante $\square D_i$.

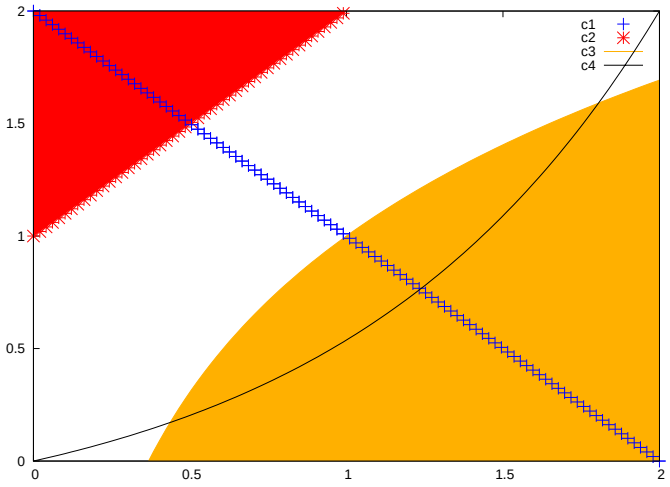


FIG. 4.2 – Problème 2B-cohérent sans solution

Par exemple, dans le cas de la contrainte disjonctive $\boldsymbol{c} : x^2 = [4, 16]$, la fonction de projection sur x est $\pi_x(\boldsymbol{c}) : x = \pm\sqrt{[4, 16]}$.

$$\begin{array}{rl} \boldsymbol{c} : & x^2 = [4, 16] \\ \pi_x(\boldsymbol{c}) : & x = \pm\sqrt{[4, 16]} \\ \pi_x(\boldsymbol{c}) : & x = \{[-4, -2], [2, 4]\} \rightsquigarrow [-4, 4] \end{array}$$

4.3.3.3 Problèmes de stabilité numérique

Le filtrage par 2B-cohérence s'effectue sur une représentation des nombres réels par des nombres flottants. Les flottants correspondent à la représentation informatique des réels selon la norme IEEE754 (1985). L'ensemble des nombres flottants $\mathbb{F}$ est un sous-ensemble fini de $\mathbb{R}^\infty$.

Les nombres flottants nf sont définis dans une base de numération b (en général en base 2) à l'aide d'une mantisse m et d'un exposant p : $nf = \pm m \cdot b^p$. La mantisse m est décrite par un nombre maximum N de chiffres significatifs. Les calculs sur les nombres flottants sont donc arrondis, contrairement à ceux sur les réels. Cette approximation crée des problèmes de stabilité numérique, car deux expressions mathématiques identiques sur $\mathbb{R}$ ne le sont pas forcément sur $\mathbb{F}$.

Considérons la représentation flottante suivante : $b = 10$ et $N = 3$. Soient la contrainte $\boldsymbol{c}_1 : h = x + (y + z)$ et $\pi_z(\boldsymbol{c_1}) : z = h - (x + y)$ la fonction de projection de $\boldsymbol{c}_1$ sur z. Le filtrage par 2B-cohérence de la contrainte $\boldsymbol{c}_1$ sur les domaines $D_x = \{0.822 \cdot 10\}$, $D_y = \{0.317 \cdot 10^{-2}\}$, $D_z = \{0.432 \cdot 10^{-2}\}$ et $D_h = \{[0, 1]\}$ donne les résultats suivants :

$$\begin{array}{rll} \boldsymbol{c_1} : & h = x + (y + z) & \\ \boldsymbol{c_1} : & h = 0.822 \cdot 10 + (0.317 \cdot 10^{-2} + 0.432 \cdot 10^{-2}) & \\ \boldsymbol{c_1} : & h = 0.822 \cdot 10 + 0.749 \cdot 10^{-2} & \\ \boldsymbol{c_1} : & h = 0.823 \cdot 10 & \text{car } 0.822749 \cdot 10 \text{ arrondi à } 0.823 \cdot 10 \end{array}$$

$$
\begin{array}{lll}
\pi_z(\boldsymbol{c_1}): & z = h - (x + y) & \\
\pi_z(\boldsymbol{c_1}): & z = 0.823 \cdot 10 - (0.822 \cdot 10 + 0.317 \cdot 10^{-2}) & \\
\pi_z(\boldsymbol{c_1}): & z = 0.823 \cdot 10 - 0.822 \cdot 10 & \text{car } 0.822317 \cdot 10 \text{ arrondi à } 0.822 \cdot 10 \\
\pi_z(\boldsymbol{c_1}): & z = 1 \cdot 10^{-2} &
\end{array}
$$

La contrainte $\boldsymbol{c_1}$ devient incohérente, puisque la variable z a pour domaine de définition $D_z = \{0.432 \cdot 10^{-2}\}$ et qu'elle est réduite à $z = \{1 \cdot 10^{-2}\}$ par la fonction de projection $\pi_z(\boldsymbol{c_1})$.

4.3.4 Synthèse

Cette section a présenté le filtrage par 2B-cohérence que nous retenons pour notre application pour :

- son faible degré de filtrage qui est compatible avec des délais d'interaction,
- sa puissance de filtrage supérieure à la Box-cohérence pour les contraintes ne nécessitant pas de décomposition en fonctions primitives (Delobel, 2000).

Cependant, il est important de garder à l'esprit que, dans le cas général :

- l'utilisation de l'arithmétique des intervalles fait que les intervalles résultats, surestimant parfois l'enveloppe des contraintes, dépendent de la syntaxe des contraintes,
- le degré de filtrage faible de la 2B-cohérence qui
  - ne réduit pas toujours de manière complète les domaines des variables,
  - ne permet pas toujours de détecter qu'un problème n'a pas de solution,
- la 2B-cohérence nécessite des contraintes projetables sur chacune de leurs variables,
- la 2B-cohérence ne gère pas les domaines multi-intervalles (politique commune à toutes les méthodes de filtrage actuelles),
- l'utilisation de flottants crée des problèmes de stabilité numérique.

4.4 Extension de la 2B-cohérence aux multi-intervalles

Nous présentons dans cette section une extension de l'arithmétique des intervalles prenant en compte les domaines multi-intervalles. Nous allons dans un premier temps exposer quels sont les besoins du multi-intervalles dans un outil d'aide à la conception interactif. Puis nous présenterons l'arithmétique des intervalles adaptée aux unions d'intervalles, ainsi que les limites de cette extension.

4.4.1 Besoin du multi-intervalles

Dans notre application, les variables numériques peuvent être liées à tout autre type de variables par des contraintes de divers types. Les tables de compatibilité mixtes par exemple, présentées dans la section 3.2, permettent de lister des combinaisons de valeurs et d'intervalles autorisées pour des variables discrètes et continues.

Dans un outil d'aide à la décision interactif, les choix peuvent aussi bien porter sur les variables discrètes que continues. Les choix de l'utilisateur peuvent alors faire perdre la continuité des domaines continus. Pour éviter de proposer à un utilisateur des choix qui ne sont plus compatibles avec l'atteinte d'une solution, nous ne pouvons rétablir la continuité des domaines continus après filtrage. Nous faisons donc le choix de conserver les domaines multi-intervalles.

Pour illustrer nos propos, nous proposons un scénario sur un système de contraintes tiré de notre application qui conduit à la perte de la continuité des domaines continus par la réduction d'une variable discrète par un utilisateur.

4.4.1.1 Système de contraintes

Considérons le système de contraintes constitué de deux contraintes de types différents. La première, représentée par une table de compatibilité mixte 4.1 nommée $\boldsymbol{c}_1$, lie la taille d'un cylindre, variable symbolique $Taille$, à son diamètre en millimètre, variable continue d. La seconde, de type fonction numérique monotone projetable nommée $\boldsymbol{c}_2$, définit que le cylindre est semi-infini, c'est-à-dire que sa hauteur, variable continue h, est égale à trois fois son diamètre, variable continue d : $h = 3 \times d$.

Nous avons donc le *CSP* $P = (\mathbb{V}, \mathbb{D}, \mathbb{C})$ suivant :

- $\mathbb{V} = \{Taille, d, h\}$
- $\mathbb{D} = \{D_{Taille} = \{petit, moyen, grand\}, D_d = \{[0, +\infty[\}, D_h = \{[0, +\infty[\}\}$
- $\mathbb{C} = \{\boldsymbol{c}_1, \boldsymbol{c}_2\}$

TAB. 4.1 – Contrainte mixte liant la taille au diamètre

$Taille$	d
petit	$[0, 15]$
moyen	$[15, 30]$
grand	≥ 30

4.4.1.2 Scénario

Faisons l'hypothèse que l'utilisateur ne peut, pour une raison quelconque, réaliser la trempe de pièces de taille moyenne. Il va donc réduire le domaine de la variable $Taille$ à $\{petit, grand\}$. Cette réduction va se répercuter, *via* la contrainte $\boldsymbol{c}_1$, sur la variable continue d et la réduire à $D_d = \{[0, 15], [30, +\infty[\}$. La réduction du domaine de d va, à son tour, se transmettre à la variable h par la contrainte $\boldsymbol{c}_2$ et après propagation par 2B-cohérence, $D_h = \{[0, 45], [90, +\infty[\}$.

Si nous rétablissons la continuité des domaines de d et de h avec leur enveloppe englobante $\Box D$ présentée dans le paragraphe 4.3.3.2, alors $D_d = \{[0, +\infty[\}$ et $D_h = \{[0, +\infty[\}$. L'utilisateur se voit ainsi proposer des valeurs incohérentes avec le problème courant comme $]15, 30[$ pour d et $]45, 90[$ pour h. Il peut alors choisir une valeur incompatible avec l'atteinte d'une solution, par exemple valuer la hauteur h à 60 mm, alors que celle-ci aurait dû être retirée des valeurs cohérentes par ses choix précédents.

4.4.2 Extension aux multi-intervalles

Pour prendre en compte la non continuité des domaines des variables, nous proposons d'étendre l'arithmétique des intervalles aux unions d'intervalles ou multi-intervalles. Les extensions optimales des fonctions élémentaires ne portent plus sur des intervalles, mais sur des unions d'intervalles. Pour éviter une explosion combinatoire, l'arithmétique des multi-intervalles unifie les intervalles résultats par l'opérateur d'union ensembliste.

Soient $\boldsymbol{x} = \cup\{\boldsymbol{x}_1, \boldsymbol{x}_2\}$ et $\boldsymbol{y} = \cup\{\boldsymbol{y}_1, \boldsymbol{y}_2\}$:

Addition

$$\begin{array}{lrcl} \boldsymbol{f}: & (\boldsymbol{x}, \boldsymbol{y}) & \mapsto & \boldsymbol{x} \oplus \boldsymbol{y} \\ \boldsymbol{f}: & (\cup\{\boldsymbol{x}_1, \boldsymbol{x}_2\}, \cup\{\boldsymbol{y}_1, \boldsymbol{y}_2\}) & \mapsto & \bigcup\{\boldsymbol{x}_1 \oplus \boldsymbol{y}_1, \boldsymbol{x}_1 \oplus \boldsymbol{y}_2, \boldsymbol{x}_2 \oplus \boldsymbol{y}_1, \boldsymbol{x}_2 \oplus \boldsymbol{y}_2\} \end{array}$$

Soustraction

$$\begin{array}{lrcl} \boldsymbol{f}: & (\boldsymbol{x}, \boldsymbol{y}) & \mapsto & \boldsymbol{x} \ominus \boldsymbol{y} \\ \boldsymbol{f}: & (\cup\{\boldsymbol{x}_1, \boldsymbol{x}_2\}, \cup\{\boldsymbol{y}_1, \boldsymbol{y}_2\}) & \mapsto & \bigcup\{\boldsymbol{x}_1 \ominus \boldsymbol{y}_1, \boldsymbol{x}_1 \ominus \boldsymbol{y}_2, \boldsymbol{x}_2 \ominus \boldsymbol{y}_1, \boldsymbol{x}_2 \ominus \boldsymbol{y}_2\} \end{array}$$

Multiplication

$$\begin{array}{lrcl} \boldsymbol{f}: & (\boldsymbol{x}, \boldsymbol{y}) & \mapsto & \boldsymbol{x} \otimes \boldsymbol{y} \\ \boldsymbol{f}: & (\cup\{\boldsymbol{x}_1, \boldsymbol{x}_2\}, \cup\{\boldsymbol{y}_1, \boldsymbol{y}_2\}) & \mapsto & \bigcup\{\boldsymbol{x}_1 \otimes \boldsymbol{y}_1, \boldsymbol{x}_1 \otimes \boldsymbol{y}_2, \boldsymbol{x}_2 \otimes \boldsymbol{y}_1, \boldsymbol{x}_2 \otimes \boldsymbol{y}_2\} \end{array}$$

Division

$$\begin{array}{lrcl} \boldsymbol{f}: & (\boldsymbol{x}, \boldsymbol{y}) & \mapsto & \boldsymbol{x} \oslash \boldsymbol{y} \\ \boldsymbol{f}: & (\cup\{\boldsymbol{x}_1, \boldsymbol{x}_2\}, \cup\{\boldsymbol{y}_1, \boldsymbol{y}_2\}) & \mapsto & \bigcup\{\boldsymbol{x}_1 \oslash \boldsymbol{y}_1, \boldsymbol{x}_1 \oslash \boldsymbol{y}_2, \boldsymbol{x}_2 \oslash \boldsymbol{y}_1, \boldsymbol{x}_2 \oslash \boldsymbol{y}_2\} \end{array}$$

Exponentielle

$$\begin{array}{lrcl} \boldsymbol{f}: & \boldsymbol{x} & \mapsto & \boldsymbol{e}^{\boldsymbol{x}} \\ \boldsymbol{f}: & \cup\{\boldsymbol{x}_1, \boldsymbol{x}_2\} & \mapsto & \bigcup\{\boldsymbol{e}^{\boldsymbol{x}_1}, \boldsymbol{e}^{\boldsymbol{x}_2}\} \end{array}$$

Par exemple, le calcul de l'intensité de déformation d'une pièce trempée, variable continue I_f, est réalisé à partir de la multiplication d'une intensité de distorsion potentielle, variable continue I_p (déterminée, par exemple, par les caractéristiques géométriques de l'axe trempé) par le produit de n coefficients de modulation $\forall j \in [1 \dots n], m^j$ (déterminés par les conditions de traitement thermique) :

$$I\_f = I\_p \times \prod_{j=1}^{n} m^j$$

Les intensités de déformation peuvent être multi-intervalles, tout comme les coefficients de modulation. Si nous considérons une intensité potentielle multi-intervalles $D_{I\_p} = \cup\{[0, 1], [3, 4]\}$

et deux coefficients de modulation m^1 de domaine $D_{m^1} = \{[1,2]\}$ et m^2 de domaine $D_{m^2} = \cup\{[0.5,1],[4,5]\}$, la déformation finale I_f sera comprise entre :

$$\begin{array}{rl} & \bigcup\{[0,1] \otimes [1,2] \otimes [0.5,1], [0,1] \otimes [1,2] \otimes [4,5], [3,4] \otimes [1,2] \otimes [0.5,1], [3,4] \otimes [1,2] \otimes [4,5]\} \\ = & \bigcup\{[0,2],[0,10],[1.5,8],[12,40]\} \end{array}$$

L'union ensembliste permet d'unifier les intervalles résultats : $\bigcup\{[0,2],[0,10],[1.5,8],[12,40]\} = \cup\{[0,10],[12,40]\}$ pour éviter de conserver inutilement des domaines multi-intervalles.

Il est à noter que lorsque les domaines sont très morcelés, l'unification des intervalles peut ne pas réduire la combinatoire des domaines.

4.4.3 Limites

L'utilisation de l'arithmétique des intervalles adaptée aux unions d'intervalles pour un filtrage par 2B-cohérence lève bien le problème des domaines multi-intervalles, mais ne résout pas les réserves évoquées en fin de synthèse précédente (page 53).

Il est à noter que la prise en considération de domaines multi-intervalles peut engendrer une grande combinatoire de calcul et ralentir par conséquent l'interactivité de l'outil d'aide à la décision. Cependant, l'unification des intervalles résultats permet de réduire cette combinatoire. De plus, les domaines de définitions des variables très morcelés ne sont pas fréquents dans les applications de conception.

4.5 Synthèse

L'arithmétique des intervalles est la base du filtrage des contraintes numériques. Celle-ci étend les notions de fonctions numériques réelles aux fonctions numériques sur des intervalles. Mais ce passage des réels vers les intervalles fait que les intervalles résultats dépendent de la forme syntaxique des contraintes. Les choix d'écriture syntaxique des contraintes numériques sont donc importants pour obtenir un filtrage qui surestime le moins possible les intervalles résultats.

Plusieurs méthodes de propagation sont basées sur l'arithmétique des intervalles. Elles permettent de filtrer des contraintes numériques de forte arité, mais certaines méthodes nécessitent des contraintes monotones, continues et projetables sur chacune de leurs variables. La classification de ces méthodes, en termes de puissance de filtrage, réalisée par Delobel (2000), montre l'intérêt du filtrage par 2B-cohérence. Nous avons opté pour un filtrage par 2B-cohérence pour :

- son faible degré de filtrage qui est compatible avec des délais d'interaction,
- sa puissance de filtrage qui est satisfaisante lorsque les contraintes ne présentent pas de multi-occurrence de variables et sont projetables, ce qui est le cas dans notre application.

D'autres applications peuvent nécessiter la prise en compte de contraintes ne possédant pas les propriétés de monotonie, de continuité et de projetabilité. Dans ce cas, nous nous orienterions vers les deux méthodes de filtrage suivantes :

– soit la méthode de discrétisation, que nous présenterons dans le chapitre 5, pour les contraintes de faible arité,
– soit la méthode de *Box*-cohérence, évoquée dans la section 4.2.2, pour les contraintes d'arité quelconque.

La projection des contraintes, nécessaire au filtrage par 2B-cohérence, peut entraîner la perte de la continuité des intervalles et fournir une union d'intervalles : celle-ci est alors approximée par une enveloppe contenant l'union de tous les intervalles. Dans une problématique d'aide à la conception interactive, nous ne pouvons présenter à l'utilisateur des intervalles qui ne sont pas totalement cohérents avec le problème courant. Nous avons donc étendu l'arithmétique des intervalles continus aux unions d'intervalles pour prendre en compte la combinatoire des domaines multi-intervalles. Pour éviter une explosion combinatoire, la surdéfinition des opérateurs multi-intervalles est contractante, car elle réalise l'union ensembliste des multi-intervalles résultats.

Chapitre 5

Expressions mathématiques et structures arborescentes

Il existe deux grands courants de méthodes de filtrage sur les expressions mathématiques. Le premier, présenté dans le chapitre 4, est basé sur l'arithmétique des intervalles, le second sur une discrétisation de l'espace de solutions, représentée sous forme de 2^k-arbres où k correspond à l'arité des contraintes numériques continues. Les 2^k-arbres sont des structures de données arborescentes d'hypercubes, de k dimensions, construits à partir d'une décomposition récursive de l'espace (Samet, 1984) jusqu'à l'obtention d'hypercubes unitaires possédant un grain prédéfini.

Sam (1995) fut la première à utiliser cette technique de représentation pour les contraintes numériques continues d'arités quelconques ne pouvant être filtrées par 2B-cohérence, c'est-à-dire les contraintes non monotones ou non projetables : les contraintes sont discrétisées et mises sous forme arborescente, délimitant ainsi les zones cohérentes et incohérentes. Les contraintes discrétisées de faible arité sont représentées par des arbres quaternaires (Finkel et Bentley, 1974) ou *quad tree* (structure arborescente de rectangles) pour les contraintes binaires et en arbres octaux ou *octree* (structure arborescente de cubes) pour les contraintes ternaires. Au-delà d'une arité de trois, les contraintes numériques k-aires sont décomposées en contraintes binaires et ternaires dont les arbres sont générés. Ces derniers sont ensuite combinés pour représenter les 2^k-arbres associés aux contraintes k-aires initiales. Cette représentation des contraintes, empruntée au traitement d'images, devrait donc être parfaitement adaptée à l'intégration d'abaques expérimentaux 2D dans les problèmes de satisfaction de contraintes. Or, il est extrêmement difficile de décrire des abaques par une seule expression mathématique continue. Il est plus aisé de les approximer par des expressions mathématiques définies par morceaux. Or, la méthode actuelle de génération d'arbres n'a pas été pensée pour prendre en compte ce type de contraintes définies par morceaux.

Nous allons, dans la section 5.1, définir les arbres quaternaires en termes de structure de données et de méthodes de génération à partir des définitions symboliques des contraintes continues numériques binaires. Puis, nous présenterons la technique dite de *fusion* permettant la prise en compte simultanée de plusieurs contraintes portant sur le même ensemble de variables. Les techniques de filtrage associées aux arbres quaternaires seront abordées.

Nous verrons que la représentation de contraintes sous forme de 2^k-arbres possède un certain nombre de limites en termes :

- de précision et de temps de génération,
- d'explorations inutiles de l'espace,
- d'impossibilité de prise en compte de contraintes continues numériques binaires définies par morceaux.

Nous présenterons, dans la section 5.2, deux méthodes permettant la génération d'arbres quaternaires à partir de contraintes continues numériques binaires définies par morceaux d'égalité ou d'inégalité. Ces méthodes reposent sur l'identification des degrés d'information pertinente des nœuds permettant de déduire leur cohérence, puis, dans le cas des inégalités par morceaux, sur la propagation des zones cohérentes et incohérentes par voisinage. La prise en compte de contraintes continues numériques binaires définies par morceaux ne modifie en rien les techniques de *fusion* et de filtrage des arbres quaternaires développées par Sam (1995). La représentation des contraintes sous forme d'arbre nécessite de trouver un compromis entre la précision désirée et le temps de génération des arbres.

La phase de propagation des zones cohérentes et incohérentes nécessite de pouvoir se déplacer de manière simple dans les 2^k-arbres : les nœuds sont donc étiquetés de manière unique suivant la courbe de Péano ordonnée par Morton. Ce système d'identification des nœuds est explicité dans la sous-section 5.1.3.

Nous proposerons, dans la section 5.3, une méthode réduisant les temps de génération et de *fusion* des arbres par une exploration opportuniste de l'espace de solution.

5.1 Arbres quaternaires

Les arbres quaternaires sont des structures de données empruntées au traitement d'images qui permettent de représenter des contraintes continues numériques binaires discrétisées dynamiquement. Nous allons, dans un premier temps, définir cette structure d'arbre en indiquant comment elle est intégrée aux problèmes de satisfaction de contraintes. Puis, nous exposerons la méthode de génération des arbres quaternaires à partir de la définition symbolique des contraintes continues numériques binaires. La prise en compte simultanée de contraintes binaires portant sur la même paire de variables se fait par un mécanisme de *fusion* de leur arbre quaternaire, présenté dans la sous-section 5.1.4. Les méthodes de filtrage associées à ce type de représentation ont été développées par Sam (1995) et seront abordées dans la sous-section 5.1.5.

5.1.1 Définition

Chaque contrainte continue numérique binaire discrétisée $\boldsymbol{C}(x, y)$ peut être mise sous forme d'un arbre quaternaire. Un arbre quaternaire est une structure arborescente où chaque nœud est père de quatre fils. Chacun des nœuds représente un sous-espace (d_n^x, d_n^y) de l'espace de

solution (D_x, D_y) dont il faut déterminer la cohérence avec la contrainte continue numérique binaire qui lui est associée.

Définition 17 : *arbre quaternaire*

Un arbre quaternaire représentant une contrainte continue numérique binaire $\boldsymbol{C}(x, y)$ est défini par le fait que :

- *la contrainte porte sur un espace de solution ou de recherche initial ($D_x^{\boldsymbol{C}}$, $D_y^{\boldsymbol{C}}$),*
- *chaque nœud n porte sur un sous-espace (d_n^x, d_n^y) de l'espace de recherche inclus dans l'espace de solution initial : $d_n^x \subseteq D_x^{\boldsymbol{C}}$ et $d_n^y \subseteq D_y^{\boldsymbol{C}}$,*
- *chaque nœud n est contraint par $\boldsymbol{C}(x, y)$,*
- *la cohérence des nœuds est matérialisée par des couleurs :*
  - *les nœuds totalement cohérents avec $\boldsymbol{C}(x, y)$ se changent en feuilles et se colorent en blanc,*
  - *les nœuds totalement incohérents avec $\boldsymbol{C}(x, y)$ se changent en feuilles et se colorent en bleu,*
  - *les nœuds composés de zones cohérentes et incohérentes avec $\boldsymbol{C}(x, y)$ se colorent en gris,*
- *chaque nœud gris est père de quatre nœuds fils nommés* Nord-Ouest *(NO),* Sud-Ouest *(SO),* Sud-Est *(SE) et* Nord-Est *(NE) dont il faut tester la cohérence,*
- *les variables de la contrainte possèdent un degré de précision ε_x et ε_y portant respectivement sur les variables x et y qui définissent le grain des nœuds unitaires,*
- *lorsque l'un des degrés de précision est atteint, les nœuds gris deviennent :*
  - *cohérents avec la contrainte numérique binaire et se colorent en blanc, si seul ce qui est totalement incohérent avec $\boldsymbol{C}(x, y)$ est à exclure,*
  - *incohérents avec la contrainte numérique binaire et se colorent en bleu, si seul ce qui est totalement cohérent avec $\boldsymbol{C}(x, y)$ est à garder.*
- *lorsque les quatre fils d'un nœud ont la même couleur de feuillage (couleur terminale blanche ou bleue), leur nœud père les absorbe et se change en feuille monochrome.*

La figure 5.1 présente la contrainte $\boldsymbol{C} : y - x^3 > 0$ avec $\epsilon_x = 0.0625$ et $\epsilon_y = 0.0625$ décomposée en nœuds de couleur blanche et bleue où les pères ont absorbé leurs fils monochromatiques (par exemple le nœud couvrant le sous-espace $(d_n^x = [1, 1.25], d_n^y = [1.75, 2])$), ainsi qu'une partie de sa structure arborescente.

Dans le cadre de notre application, seul ce qui est totalement incohérent avec les contraintes est à exclure. Les nœuds gris unitaires se colorent donc en blanc.

5.1.2 Génération

La génération des arbres quaternaires passe par une décomposition récursive de l'espace de recherche en sous-espaces dont il faut déterminer la cohérence avec les contraintes binaires

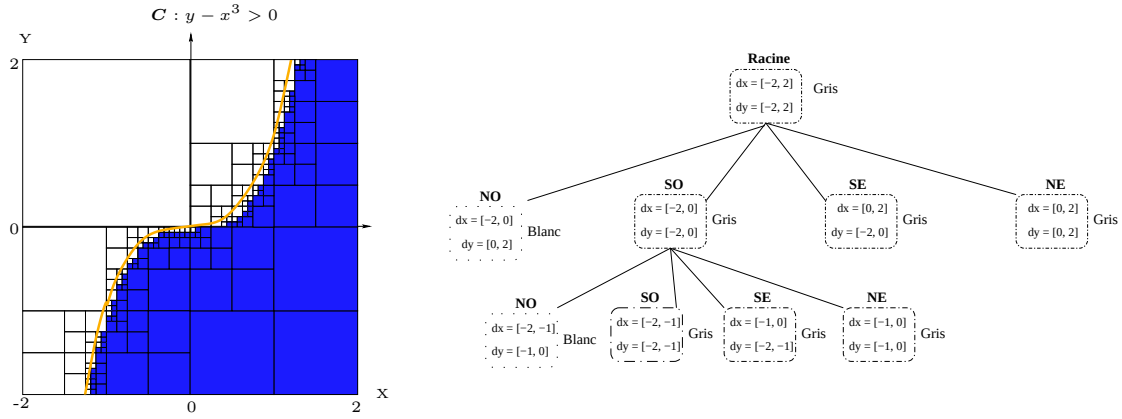


FIG. 5.1 – Contrainte binaire sous forme d'arbre quaternaire

discrétisées. La cohérence d'un nœud n est donc fonction de la contrainte continue numérique binaire $\boldsymbol{C}(x, y)$ et des intervalles de valeurs définissant son sous-espace (d_n^x, d_n^y).

Pour déterminer la cohérence d'un nœud, Sam (1995) recherche les intersections entre la contrainte continue numérique binaire et le rectangle défini par le sous-espace (d_n^x, d_n^y) du nœud traité. Des études par analyse numérique sont menées afin de déterminer si la contrainte coupe l'une des arêtes du rectangle. Si des intersections sont détectées, le nœud se colore en gris et doit être décomposé. Sinon le nœud devient une feuille. Pour déterminer sa cohérence et le colorer, il suffit de connaître la cohérence de l'un des points du rectangle (d_n^x, d_n^y) (coin, milieu...). Cette méthode de coloration des nœuds a une faible complexité, mais elle dépend fortement de celle de la méthode numérique utilisée pour déterminer si la contrainte passe par le rectangle associé à un nœud. Sa complexité vaut au pire $(2^{2h-1} + 2^h) \times \Theta_{\text{méthode numérique}}$ où h équivaut à la hauteur totale de l'arbre, $(2^{2h-1} + 2^h)$ équivaut au nombre d'arêtes et $\Theta_{\text{méthode numérique}}$ à la complexité de la détermination des intersections avec une arête pour la méthode numérique employée.

Il existe une autre méthode basée sur l'arithmétique des intervalles, proposée par Lottaz (2000), qui évite l'utilisation des méthodes d'analyse numérique pour déterminer la cohérence des nœuds :

- dans le cas de contrainte numérique binaire décrite par une fonction numérique de $\mathbb{R} \mapsto \mathbb{R}$, seule la comparaison de l'image $\boldsymbol{Y}$ de l'intervalle d_n^x par la contrainte $\boldsymbol{C}(x, y)$ à l'intervalle d_n^y est nécessaire. Si $\boldsymbol{Y} \cap d_n^y = \emptyset$, le nœud est totalement incohérent : il devient une feuille bleue. Si $\boldsymbol{Y} \cap d_n^y = d_n^y$, le nœud est totalement cohérent : il devient une feuille blanche. Sinon, il se colore en gris qui doit être décomposé.
- dans le cas de contrainte numérique binaire définie par une fonction numérique de $\mathbb{R}^2 \mapsto \mathbb{R}$, il faut vérifier que le couple d'intervalles d_n^x et d_n^y vérifient totalement la contrainte $\boldsymbol{C}(x, y)$. Si tel est le cas, le nœud se colore en blanc. Si le nœud ne vérifie la contrainte que partiellement, il se colore en gris et doit être, à nouveau, décomposé. Sinon, il est totalement incohérent avec la contrainte et il devient une feuille bleue.

Si nous appliquons cette méthode à trois nœuds de la contrainte $\boldsymbol{C} : y - x^3 > 0$, illustrée par la figure 5.1, nous vérifions la cohérence des trois nœuds par rapport à la contrainte :

– le nœud n couvrant le sous-espace $([0, 0.5], [0.5, 1])$ est blanc, puisque ses intervalles $[0, 0.5]$ et $[0.5, 1]$ vérifient totalement la contrainte $\boldsymbol{C}(x, y)$:

$$\begin{array}{cccl} d_n^y & \ominus & (d_n^x)^3 & > \mathbf{0} \\ [0.5, 1] & \ominus & [0, 0.5]^3 & > [0, 0] \\ [0.5, 1] & \ominus & [0, 0.125] & > [0, 0] \\ & [0.475, 1] & > [0, 0] & \text{vrai pour tout l'intervalle résultat} \end{array}$$

– le nœud n couvrant le sous-espace $([1, 2], [-1, 0])$ est bleu, puisque ses intervalles $[1, 2]$ et $[-1, 0]$ ne vérifient pas du tout la contrainte $\boldsymbol{C}(x, y)$:

$$\begin{array}{cccl} d_n^y & \ominus & (d_n^x)^3 & > \mathbf{0} \\ [-1, 0] & \ominus & [1, 2]^3 & > [0, 0] \\ [-1, 0] & \ominus & [1, 8] & > [0, 0] \\ & [-9, -1] & > [0, 0] & \text{faux pour tout l'intervalle résultat} \end{array}$$

– le nœud n couvrant le sous-espace $([1, 2], [1, 2])$ est gris et doit être décomposé, puisque ses intervalles $[1, 2]$ et $[1, 2]$ ne vérifient que partiellement la contrainte $\boldsymbol{C}(x, y)$:

$$\begin{array}{cccl} d_n^y & \ominus & (d_n^x)^3 & > \mathbf{0} \\ [1, 2] & \ominus & [1, 2]^3 & > [0, 0] \\ [1, 2] & \ominus & [1, 8] & > [0, 0] \\ & [-9, 1] & > [0, 0] & \text{vrai pour une partie de l'intervalle résultat} \end{array}$$

La fonction Arbre quaternaire, présentée dans l'algorithme 7, discrétise une contrainte numérique $\boldsymbol{C}_{(x,y)}$ sur un espace de solution et retourne l'arbre quaternaire associé. Cette fonction fait appel à la fonction Générer arbre quaternaire, présentée par l'algorithme 8, qui décompose l'espace de recherche et génère l'arbre quaternaire à partir d'un nœud.

Dans le cadre de notre application, seul ce qui est totalement incohérent avec les contraintes est à exclure. Les nœuds gris unitaires deviennent des feuilles blanches.

Alg. 7 Arbre quaternaire(Contrainte : $\boldsymbol{C}_{(x,y)}$, Intervalle : $D_x^{\boldsymbol{C}}$, Intervalle $D_y^{\boldsymbol{C}}$)

– ∴ – ***Cet algorithme génère un arbre quaternaire*** $T_{(x,y)}$ ***à partir d'une contrainte*** $\boldsymbol{C}_{(x,y)}$ ***et d'un espace de recherche*** $D_x^{\boldsymbol{C}}$, $D_y^{\boldsymbol{C}}$.
– ∴ – $D_x^{\boldsymbol{C}}$ *correspond au domaine de recherche suivant* x
– ∴ – $D_y^{\boldsymbol{C}}$ *correspond au domaine de recherche suivant* y
Création du nœud racine r de $T_{(x,y)}$ contraint par $\boldsymbol{C}_{(x,y)}$ et portant sur $D_x^{\boldsymbol{C}}$ et $D_y^{\boldsymbol{C}}$
$r \leftarrow$ Générer Arbre quaternaire(r) (algorithme 8 page suivante)
– ∴ – *Retourner l'arbre associé à la contrainte* $\boldsymbol{C}_{(x,y)}$
Retourner $(T_{(x,y)})$

L'un des problèmes connus de l'arithmétique des intervalles, présentée en 4.1.3, est la surestimation des intervalles résultats lors de la présence d'occurrences multiples d'au moins une variable. Appliquée à la génération d'arbres quaternaires, celle-ci peut conduire à des explorations inutiles de zones qui, par une méthode d'analyse numérique, auraient directement été détectées incohérentes. Malgré un nombre d'explorations plus important, cette méthode reste tout de même plus rapide que celle utilisant des méthodes d'analyse numérique (Lottaz, 2000).

Alg. 8 GÉNÉRER ARBRE QUATERNAIRE(NOEUD : n)

– $\therefore$ – ***Cet algorithme construit l'arbre quaternaire*** $T_{(x,y)}$ ***d'une contrainte*** $\boldsymbol{C}_{(x,y)}$ ***à partir d'un nœud.***
– $\therefore$ – ϵ_x *correspond au degré de précision de la variable* x
– $\therefore$ – ϵ_y *correspond au degré de précision de la variable* y
– $\therefore$ – $d_n^x = [\overline{x}, \underline{x}]$ *et* $d_n^y = [\overline{y}, \underline{y}]$ *correspondent au sous-espace du nœud courant*
Si $(\overline{x} - \underline{x} > \epsilon_x \vee \overline{y} - \underline{y} > \epsilon_y)$ **Alors**
  – $\therefore$ – *La précision n'est pas atteinte en* x *ou en* y
  Création et étiquetage des quatre fils du nœud courant n
  Pour chacun des nœuds fils f de n **Faire**
    Si (f est totalement incohérent avec $\boldsymbol{C}_{(x,y)}$) **Alors**
      – $\therefore$ – *Le nœud* f *se colore en bleu*
      $couleur(f) \leftarrow$ bleu
    Sinon
      Si (f est totalement cohérent avec $\boldsymbol{C}_{(x,y)}$) **Alors**
        – $\therefore$ – *Le nœud* f *se colore en blanc*
        $couleur(f) \leftarrow$ blanc
      Sinon
        – $\therefore$ – *Le nœud* f *est partiellement cohérent avec* $\boldsymbol{C}_{(x,y)}$
        – $\therefore$ – *Il faut à nouveau le décomposer*
        $couleur(f) \leftarrow$ gris
        $f \leftarrow$ GÉNÉRER ARBRE QUATERNAIRE(f) (algorithme 8)
      Fin Si
    Fin Si
  Fin Pour
  Absorption des nœuds fils f monochromes blancs et bleus par leur nœud père n
Sinon
  – $\therefore$ – *Le nœud* n *est unitaire*
  Si (n est partiellement cohérent avec $\boldsymbol{C}_{(x,y)}$) **Alors**
    – $\therefore$ – *Le nœud* n *se colore en blanc*
    $couleur(n) \leftarrow$ blanc
  Sinon
    – $\therefore$ – *Le nœud* n *est incohérent et se colore en bleu*
    $couleur(n) \leftarrow$ bleu
  Fin Si
Fin Si
– $\therefore$ – *Retourner le nœud courant*
Retourner (n)

Dans la cadre de notre application, nous optons pour la méthode de Lottaz (2000) à base d'arithmétique des intervalles pour son fonctionnement systématique, quelque soit la forme syntaxique de la contrainte. L'utilisation de l'arithmétique des intervalles, pour la génération des arbres quaternaires, nécessite, tout de même, la surdéfinition des fonctions élémentaires constituant les contraintes à discrétiser.

5.1.3 Étiquetage des nœuds

Les nœuds et les feuilles des arbres quaternaires sont étiquetés de manière unique à partir de la courbe remplissante de Peano en N et de l'ordre de Morton, dont nous pouvons trouver la définition dans Briggs et Peat (1991). L'étiquette d'un nœud dépend de sa profondeur dans l'arbre quaternaire. Cette numérotation unique permet de parcourir l'arborescence à partir des coordonnées des nœuds et des feuilles.

Courbe de Péano La courbe remplissante de Péano est une courbe continue qui traverse chaque point d'une surface bidimensionnelle, en répétant toujours le même motif. La figure 5.2 présente une courbe remplissante de Péano avec un motif en N. Ce type de courbe possède deux propriétés des plus intéressantes :

- elle passe une et une seule fois à travers tous les points de l'espace de départ,
- deux points voisins de la courbe le sont aussi dans l'espace.

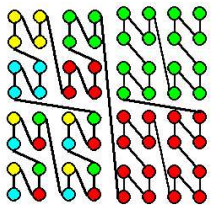


FIG. 5.2 – Courbe de Péano avec un motif en N

Ordre de Morton La courbe de Péano en N ordonnée suivant Morton, figure 5.3, permet d'étiqueter de manière unique les nœuds à partir de leurs coordonnées géographiques.

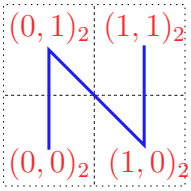


FIG. 5.3 – Ordre de Morton appliqué au motif en N de Péano

Les coordonnées (x, y) vont permettre de positionner le nœud traité dans l'arbre quaternaire, tel que le montre la figure 5.4. Le nombre de bits utilisés pour coder les coordonnées d'un

nœud est défini par $2 \times h$, où h représente la hauteur du nœud dans l'arbre. Nous notons $(x, y)_h$ le couple de coordonnées codé sur h bits chacun.

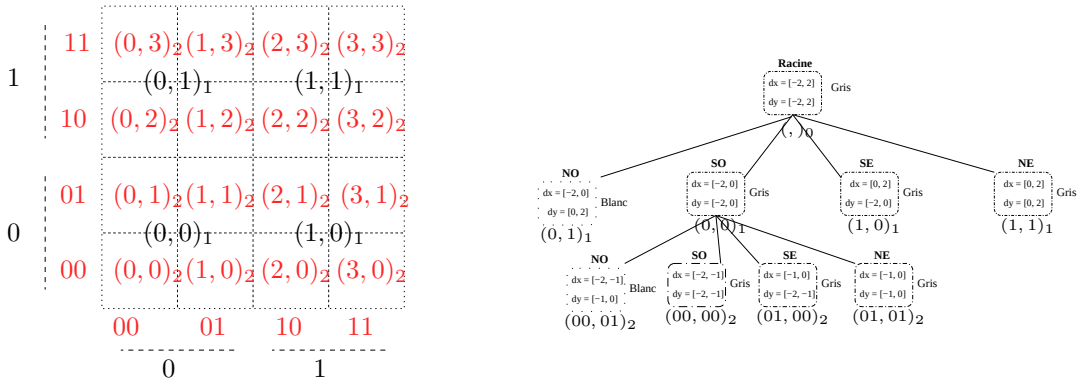


FIG. 5.4 – Étiquetage des nœuds de l'exemple 5.1 suivant la courbe de Péano ordonnée selon Morton

Ascendants d'un nœud Pour passer d'un nœud de hauteur h (de code $2 \times h$ bits) à son ascendant de hauteur $h - i$ (de code $2 \times (h - i)$), il suffit de considérer les $h - i$ bits de poids fort de ses coordonnées.

Si les coordonnées d'un nœud sont codées sur $2 \times h$, il faut :

- $fils_h \rightarrow$ père$_{h-1}$: considérer les $h - 1$ bits de poids fort des coordonnées du nœud pour connaître celles de son père.
- $fils_h \rightarrow$ grand-père$_{h-2}$: considérer les $h - 2$ bits de poids fort des coordonnées du nœud pour connaître celles de son grand-père.

Par exemple, les aïeuls du nœud $(5, 2)_3$, représentés sur la figure 5.5, sont :

- $(5, 2)_3 \Leftrightarrow (101, 010)_3 \rightarrow (10, 01)_2 \Leftrightarrow (2, 1)_2$ pour son père et
- $(5, 2)_3 \Leftrightarrow (101, 010)_3 \rightarrow (1, 0)_1 \Leftrightarrow (1, 0)_1$ pour son grand-père.

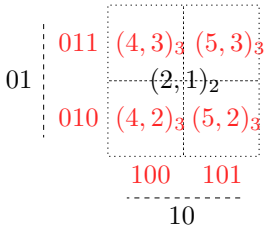


FIG. 5.5 – Père du nœuds $(5, 2)_3$

Descendants d'un nœud Pour passer d'un nœud de hauteur h (de code $2 \times h$) à ses descendants de hauteur $h+i$ (de code $2 \times (h+i)$), il faut ajouter i bits à ses coordonnées. Les bits ajoutés sont les bits de faible poids. Ces bits supplémentaires sont alors valués alternativement à 0 ou à 1 pour retrouver les étiquettes binaires de ses descendants :

- père$_h$ $\rightarrow$ fils$_{h+1}$: ajout d'un bit supplémentaire pour coder les lignes et les colonnes, valués successivement à 0 et 1.
- père$_h$ $\rightarrow$ petit-fils$_{h+2}$: ajout de 2 bits supplémentaires pour coder les lignes et les colonnes, valués successivement à 0 et 1.

Par exemple, les quatre fils du nœud $(1,0)_1$ sont :

- $(\mathbf{1}0,0\mathbf{0})_2 \Leftrightarrow (2,0)_2$,
- $(\mathbf{1}0,0\mathbf{1})_2 \Leftrightarrow (2,1)_2$,
- $(\mathbf{1}1,0\mathbf{0})_2 \Leftrightarrow (3,0)_2$ et
- $(\mathbf{1}1,0\mathbf{1})_2 \Leftrightarrow (3,1)_2$

Et ses seize petits-fils sont :

- $(\mathbf{1}00,0\mathbf{0}0)_3 \Leftrightarrow (4,0)_3$, $(\mathbf{1}00,0\mathbf{0}1)_3 \Leftrightarrow (4,1)_3$, $(\mathbf{1}01,0\mathbf{0}0)_3 \Leftrightarrow (5,0)_3$, $(\mathbf{1}01,0\mathbf{0}1)_3 \Leftrightarrow (5,1)_3$,
- $(\mathbf{1}00,0\mathbf{1}0)_3 \Leftrightarrow (4,2)_3$, $(\mathbf{1}00,0\mathbf{1}1)_3 \Leftrightarrow (4,3)_3$, $(\mathbf{1}01,0\mathbf{1}0)_3 \Leftrightarrow (5,2)_3$, $(\mathbf{1}01,0\mathbf{1}1)_3 \Leftrightarrow (5,3)_3$,
- $(\mathbf{1}10,0\mathbf{0}0)_3 \Leftrightarrow (6,0)_3$, $(\mathbf{1}10,0\mathbf{0}1)_3 \Leftrightarrow (6,1)_3$, $(\mathbf{1}11,0\mathbf{0}0)_3 \Leftrightarrow (7,0)_3$, $(\mathbf{1}11,0\mathbf{0}1)_3 \Leftrightarrow (7,1)_3$,
- $(\mathbf{1}10,0\mathbf{1}0)_3 \Leftrightarrow (6,2)_3$, $(\mathbf{1}10,0\mathbf{1}1)_3 \Leftrightarrow (6,3)_3$, $(\mathbf{1}11,0\mathbf{1}0)_3 \Leftrightarrow (7,2)_3$ et $(\mathbf{1}11,0\mathbf{1}1)_3 \Leftrightarrow (7,3)_3$

C'est donc l'ajout et le retrait des bits de faible poids qui vont nous permettre de nous déplacer dans l'arborescence ordonnée suivant Morton.

5.1.4 Fusion

La prise en compte simultanée de plusieurs contraintes continues numériques binaires portant sur la même paire de variables, sous forme de conjonction, est réalisée à partir du mécanisme de *fusion* des arbres quaternaires. La *fusion* d'arbres, notée $\Cap$, ne peut être effectuée que sur des arbres ayant les mêmes domaines de définition et possédant, par conséquent, le même découpage de l'espace de solutions.

Les couleurs des nœuds sont ordonnées :

$$blanc < gris < bleu.$$

La couleur d'un nœud n commun à k contraintes est déterminée par la comparaison de ses k couleurs où $couleur(n)_{\boldsymbol{C_k}(x,y)}$ correspond à la couleur du nœud n dans la contrainte $\boldsymbol{C_k}(x,y)$. La couleur fusionnée d'un nœud est donnée par :

$$couleur(n) = Max(couleur(n)_{\boldsymbol{C_1}(x,y)}, couleur(n)_{\boldsymbol{C_2}(x,y)}, \ldots, couleur(n)_{\boldsymbol{C_k}(x,y)})$$

La figure 5.6 présente la *fusion* de deux contraintes portant sur la même paire de variables. La première est définie par $\boldsymbol{C_1} : y < x + 1$, la seconde par $\boldsymbol{C_2} : y > -x + 2$. Toutes deux portent sur les intervalles $D_x = [0,5]$ et $D_y = [0,5]$ et ont une précision de $\epsilon_x = 0.0625$ et $\epsilon_y = 0.0625$.

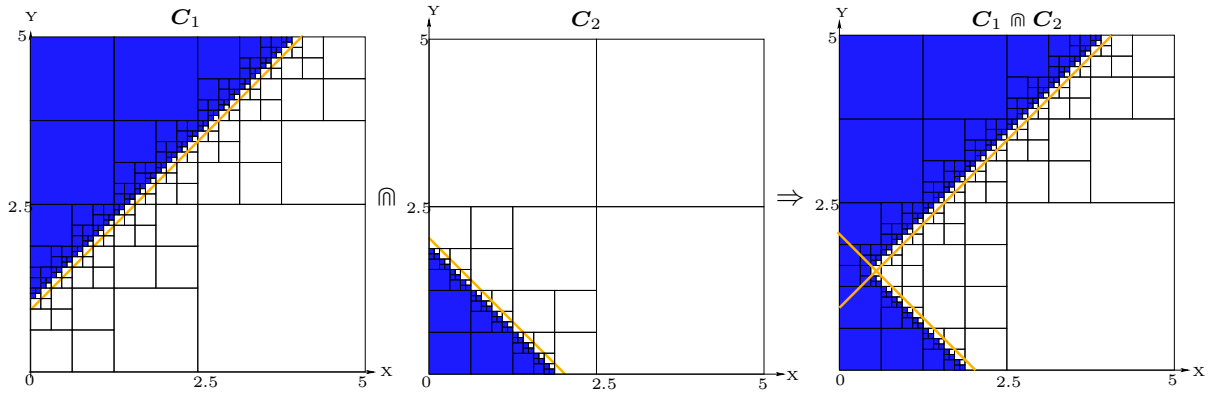


FIG. 5.6 – Fusion de contraintes binaires

5.1.5 Filtrage sur les arbres quaternaires

L'objectif principal des méthodes de filtrage est la détection et le retrait des domaines des variables, de valeurs qui ne peuvent mener à une solution. La représentation des contraintes continues numériques binaires sous forme arborescente découpe l'espace de recherche en régions compatibles blanches et incompatibles bleues. Sam (1995) a développé plusieurs algorithmes de filtrage de différents degrés. Dans un contexte d'aide à la décision interactif, nous devons privilégier les filtrages de faible degré. Nous utilisons donc la procédure développée par Sam (1995), nommée $\tau - AC - 3$ et présentée par l'algorithme 11 page 71, qui filtre par arc-cohérence des contraintes n-aires représentées par des 2^k-arbres. Nous l'avons, cependant, restreint aux arbres quaternaires. Le retrait des valeurs ne menant à aucune solution est effectué en réalisant l'union des feuilles cohérentes blanches de l'arbre quaternaire intersectée avec les domaines courants des variables de la contrainte[1].

La fonction DOMAINE COMPATIBLE, présentée par l'algorithme 9 page suivante, retourne l'union des feuilles cohérentes d'un arbre quaternaire $T_{(x,y)}$ portant sur la paire de variables x et y.

Lorsque le domaine D_x d'une variable x appartenant à plusieurs arbres quaternaires $T_{(x,j)}$ est réduit, il faut propager cette information aux contraintes continues numériques binaires $\boldsymbol{C}(x,j)$ représentées par les arbres $T_{(x,j)}$. Ceci revient à élaguer les branches des arbres $T_{(x,j)}$ qui ont perdu leur cohérence avec le problème courant et à décomposer celles qui pourraient ne plus l'être totalement. Les branches devenues totalement incompatibles sont détectées à partir de la détermination de l'intersection du nouveau domaine D_x de la variable réduite x avec le domaine d_n^x du nœud traité n pour la variable x de l'arbre $T_{(x,y)}$. Si $D_x \cap d_n^x = \emptyset$, alors la branche doit être élaguée : le nœud n perd ses fils et se colore en bleu. Sinon, soit la branche se trouve être une feuille blanche et elle doit être décomposée pour éliminer d'éventuelles zones incohérentes en générant un nouvel arbre quaternaire sur ce nœud, par l'appel à la procédure GÉNÉRER ARBRE QUATERNAIRE, présentée par l'algorithme 8 page 64, soit la branche est un nœud gris dont il faut vérifier les fils. La procédure PROPAGER RÉDUCTION DOMAINE, présentée par l'algorithme 10 page 70 ébranche un arbre $T_{(x,y)}$ et explore ses feuilles blanches pour éliminer les zones incohérentes avec le domaine réduit de la variable x.

[1] Ce qui revient à retirer des domaines des variables l'union des nœuds bleus, comme dans l'algorithme originel de Sam (1995).

Alg. 9 DOMAINE COMPATIBLE(ARBRE : $T_{(x,y)}$)

– ∴ – ***Cet algorithme renvoie les domaines cohérents*** D_x ***et*** D_y ***des variables*** x ***et*** y ***avec l'arbre*** $T_{(x,y)}$***.***
– ∴ – *Noeuds correspond aux nœuds restant à parcourir.*
$Noeuds \leftarrow$ la racine de $T_{(x,y)}$
– ∴ – D_x *correspond au domaine compatible de la variable* x *de* $T_{(x,y)}$
$D_x \leftarrow \emptyset$
– ∴ – D_y *correspond au domaine compatible de la variable* y *de* $T_{(x,y)}$
$D_y \leftarrow \emptyset$
Tant Que ($Noeuds \neq \emptyset$) **Faire**
    Dépiler un nœud n de $Noeuds$
    – ∴ – *Si le nœud est cohérent*
    Si (la couleur de n est blanche) **Alors**
        – ∴ – d_n^x *et* d_n^y *représentent le sous-espace de recherche du nœud* n *courant*
        – ∴ – *Union des intervalles compatibles pour* x *et* y *de* $T_{(x,y)}$
        $D_x \leftarrow D_x \cup d_n^x$
        $D_y \leftarrow D_y \cup d_n^y$
    Sinon
        Si (la couleur de n est grise) **Alors**
            – ∴ – n *est un nœud qui est père de quatre fils*
            – ∴ – *Il faut donc continuer à explorer ses fils*
            Empiler les quatre fils de n dans $Noeuds$
        Fin Si
    Fin Si
Fin Tant Que
– ∴ – *Retourner les domaines compatibles*
Retourner (D_x, D_y)

Alg. 10 PROPAGER RÉDUCTION DOMAINE(ARBRE : $T_{(x,y)}$, VARIABLE : x)

– ∴ – ***Cet algorithme filtre un arbre*** $T_{(x,y)}$ ***à partir d'une variable*** x***.***
– ∴ – *L'arbre quaternaire* $T_{(x,y)}$ *est associé à la contrainte* $\boldsymbol{C}(x, y)$
– ∴ – *Noeuds correspond aux nœuds restant à parcourir.*
$Noeuds \leftarrow$ la racine de $T_{(x,y)}$
– ∴ – D_x *correspond au domaine courant de la variable* x
– ∴ – d_n^x *et* d_n^y *représentent le sous-espace de recherche du nœud* n *courant*
Tant Que ($Noeuds \neq \emptyset$) **Faire**
  Dépiler un nœud n de $Noeuds$
  Si (la couleur de n n'est pas bleue) **Alors**
    Si ($d_n^x \cap D_x = \emptyset$) **Alors**
      – ∴ – *Si l'intersection est vide, la branche doit être élaguée*
      Suppression de tous les fils du nœud n
      $couleur(n) \leftarrow$ bleu
    Sinon
      Si (la couleur de n est blanche) **Alors**
        – ∴ – *Il faut explorer le nœud pour supprimer d'éventuelles zones incohérentes*
        – ∴ – *L'arbre quaternaire associé à la contrainte* $x = D_x$ *est généré sur le sous-espace* d_n^x, d_n^y
        – ∴ – *La contrainte associée au nœud à explorer devient* $x = D_x$
        $n \leftarrow$ GÉNÉRER ARBRE QUATERNAIRE(n) (algorithme 8 page 64)
      Sinon
        – ∴ – *Il faut continuer à explorer les fils de* n
        Empiler les quatre fils de n dans $Noeuds$
      Fin Si
    Fin Si
  Fin Si
Fin Tant Que

Alg. 11 τ-AC-3-QUATERNAIRE(CSP : P)

– $\therefore$ – ***Cet algorithme filtre tous les arbres*** $T_{(x,y)}$ ***d'un*** **CSP.**
– $\therefore$ – T = *liste des arbres quaternaires de* P
$T \leftarrow$ tous les arbres quaternaires de P
Tant Que ($T \neq \emptyset$) **Faire**
  Dépiler un arbre $T_{(x,y)}$ de T
  – $\therefore$ – $D_{x'}$ = *domaine compatible de la variable* x *pour la contrainte* $T_{(x,y)}$
  – $\therefore$ – $D_{y'}$ = *domaine compatible de la variable* y *pour la contrainte* $T_{(x,y)}$
  $(D_{x'}, D_{y'}) \leftarrow$ DOMAINE COMPATIBLE($T_{(x,y)}$) (algorithme 9 page 69)
  – $\therefore$ – *Réduction des domaines courant* D_x *et* D_y *des variables* x *et* y
  $D_x \leftarrow D_x \cap D_{x'}$
  $D_x \leftarrow D_y \cap D_{y'}$
  Si (D_x est réduit) **Alors**
    Pour les contraintes $T_{(x,j)}$ **Faire**
      PROPAGER RÉDUCTION DOMAINE($T_{(x,j)}$,x) (algorithme 10 page précédente)
      Empiler $T_{(x,j)}$ dans T
    Fin Pour
  Fin Si
  Si (D_y est réduit) **Alors**
    Pour les contraintes $T_{(y,j)}$ **Faire**
      PROPAGER RÉDUCTION DOMAINE($T_{(y,j)}$,y) (algorithme 10 page ci-contre)
      Empiler $T_{(y,j)}$ dans T
    Fin Pour
  Fin Si
Fin Tant Que

La procédure τ-AC-3 restreinte aux arbres quaternaires, nommée τ-AC-3-QUATERNAIRE et présentée par l'algorithme 11, revient à réduire les domaines des variables à partir de la structure arborescente blanche et bleue, puis à propager ces réductions de domaines aux autres arbres quaternaires portant, au moins, sur l'une de ces variables.

5.1.6 Limites

La représentation des contraintes continues numériques binaires sous forme d'arbres quaternaires pose un certain nombre de limites. Les méthodes de filtrage sur ce type de contraintes sont approchées aux degrés de précision près. L'utilisation de l'arithmétique des intervalles pour générer les arbres est certes rapide, mais elle peut engendrer des décompositions inutiles. Enfin, la méthode de génération des arbres quaternaires n'a pas été pensée pour prendre en compte des contraintes continues numériques binaires définies par morceaux.

5.1.6.1 Solutions approchées dues au degré de précision

La représentation de contraintes continues numériques binaires sous forme d'arbres quaternaires passe par une discrétisation récursive des contraintes suivant des degrés de précision.

Ceux-ci sont directement liés aux variables continues et définissent le grain des nœuds unitaires. Ils introduisent par conséquent des approximations des domaines cohérents. Si nous reprenons l'exemple tiré de Rueher (2002), illustré par la figure 4.1 page 51, : $\{\boldsymbol{C_1} : x + y = 2, \boldsymbol{C_2} : y \leq x + 1, \boldsymbol{C_3} : y \geq 1 + \ln(x)\}$ sur les domaines $D_x = [0, 100], D_y = [0, 100]$, en considérant plusieurs degrés de précision, nous obtenons les résultats présentés dans le tableau 5.1. Nous prenons comme temps de référence celui associé à $\epsilon_x = \epsilon_y = 10$.

TAB. 5.1 – Précision et temps de génération

$\epsilon_x = \epsilon_y$	Temps	D_x filtré	D_y filtré
10	1	[0, 6.25]	[0, 6.25]
5	3.75	[0,3.125]	[0, 3.125]
1	55.72	[0, 1.5625]	[0, 2.34375]
0.5	220	[0.390625, 1.5625]	[0.78125, 1.953125]
0.1	3551.74	[0.48125, 1.0742]	[0.8798, 1.71875]

Plus la précision est élevée, plus le temps de génération des arbres est important. Il faut donc arriver à trouver un compromis entre la précision souhaitée et un temps de calcul compatible avec une interaction.

Nous remarquons, de plus, que pour un petit degré de précision, les domaines filtrés par discrétisation sont plus précis que filtrés par 2B-cohérence.

5.1.6.2 Explorations inutiles de l'espace de recherche

L'utilisation de l'arithmétique des intervalles pour tester la cohérence des nœuds permet de réduire le temps de génération des arbres quaternaires par rapport à l'utilisation de méthodes d'analyse numérique. Mais celle-ci entraîne des explorations inutiles lors de la présence d'occurrences multiples de variables.

Considérons la contrainte continue numérique binaire : **zéro** : $y = x - x$ sur les intervalles $D_x = [-1, 3]$ et $D_y = [-1, 3]$ où x et y ont une précision de 0.25. La figure 5.7 présente sur sa partie gauche l'ensemble des sous-espaces explorés avant absorption des fils monochromes par leur père et sur sa partie droite, l'arbre quaternaire obtenu après absorption.

La méthode de génération de Sam (1995) utilisant des méthodes d'analyses numériques aurait uniquement exploré la partie droite de la figure 5.7.

5.1.6.3 Contraintes continues numériques binaires définies par morceaux

La méthode de décomposition en arbre quaternaire proposée par Sam (1995) n'a pas été conçue pour prendre en compte des fonctions définies par morceaux où seules des zones bien définies possèdent de l'information. La figure 5.8 présente une contrainte d'inégalité, nommée **vague**,

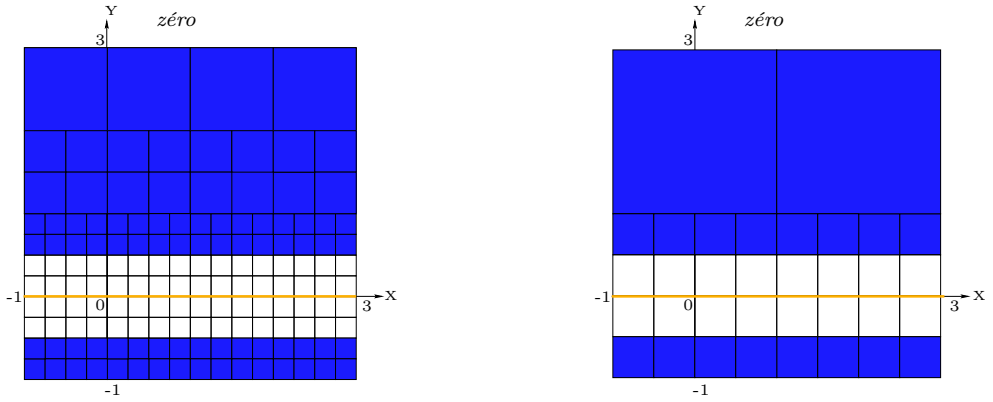


FIG. 5.7 – Exploration inutile de certains sous-espaces

définie par quatre morceaux :

$$\boldsymbol{vague}(x,y) \left\| \begin{array}{l} D_x^{\boldsymbol{vague}} = [0,4] \\ D_y^{\boldsymbol{vague}} = [0,4] \end{array} \right. = \bigcup \left\{ \begin{array}{lll} \boldsymbol{m_1}(x,y): & (x-1)^2+(y-1)^2 \leq 1 & \left\| \begin{array}{l} D_x^{m_1} = [0,2] \\ D_y^{m_1} = [1,2] \end{array} \right. \\ \boldsymbol{m_2}(x,y): & (x-1.5)^2+y^2 \geq 1 & \left\| \begin{array}{l} D_x^{m_2} = [1,2] \\ D_y^{m_2} = [0,1] \end{array} \right. \\ \boldsymbol{m_3}(x,y): & y \geq 0 & \left\| \begin{array}{l} D_x^{m_3} = [0,2] \\ D_y^{m_3} = 0 \end{array} \right. \\ \boldsymbol{m_4}(y,x): & x \geq 0 & \left\| \begin{array}{l} D_x^{m_4} = 0 \\ D_y^{m_4} = [0,2] \end{array} \right. \end{array} \right.$$

où le degré de précision est de $\epsilon_x = \epsilon_y = 0.05$. La méthode de génération de Sam (1995) considère les morceaux d'une contrainte définie par morceaux comme définies sur l'ensemble de l'espace de recherche. Or, il se peut que ces morceaux ne définissent pas d'espace globalement cohérent sur l'ensemble de l'espace.

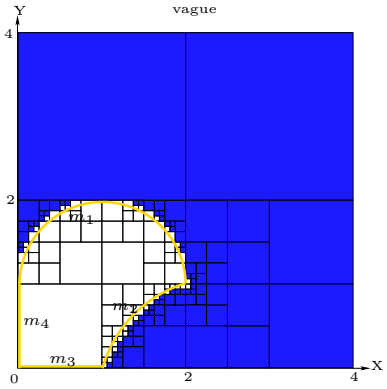


FIG. 5.8 – Exemple de contrainte définie par morceaux

5.1.7 Synthèse

Les arbres quaternaires permettent de représenter et d'intégrer dans les *CSPs* des contraintes continues numériques binaires discrétisées qui ne possèdent pas forcément les propriétés de

monotonie et de projetabilité indispensables pour un filtrage par 2B-cohérence.

Leur génération à partir de la définition symbolique des contraintes binaires $\boldsymbol{C}(x, y)$ est réalisée en exploitant l'arithmétique des intervalles. L'espace de recherche est découpé récursivement en rectangles (d_n^x, d_n^y), associés aux nœuds de l'arbre, jusqu'à l'atteinte d'une précision prédéfinie. La cohérence de chaque nœud est vérifiée au fur et à mesure du découpage et chacun se voit affecter :

- la couleur terminale bleue s'il est totalement incohérent avec $\boldsymbol{C}(x, y)$,
- la couleur terminale blanche s'il est totalement cohérent avec $\boldsymbol{C}(x, y)$,
- la couleur grise s'il est partiellement cohérent avec $\boldsymbol{C}(x, y)$ et s'il n'est pas unitaire, le nœud doit être à nouveau décomposé.

L'attribution d'une étiquette unique à chacun des nœuds à partir de ses coordonnées facilite la recherche de ses ascendants et de ses descendants et par conséquent le parcours de cette structure arborescente.

La prise en compte de plusieurs contraintes continues numériques binaires portant sur la même paire de variables et sur le même espace de définition est réalisée par le mécanisme de *fusion*. L'arbre résultat contenant la conjonction de toutes les contraintes binaires est construit à partir de la comparaison des couleurs de chacun des nœuds des arbres quaternaires associés aux contraintes suivant l'ordre établi : $blanc < gris < bleu$.

Le filtrage des contraintes continues numériques binaires mises sous forme arborescente consiste à élaguer les branches devenues entièrement incompatibles avec le problème courant et à décomposer celles qui pourraient ne plus l'être totalement.

La génération d'arbres quaternaires à partir de la définition syntaxique de contraintes continues numériques binaires possède un certain nombre de limites dues :

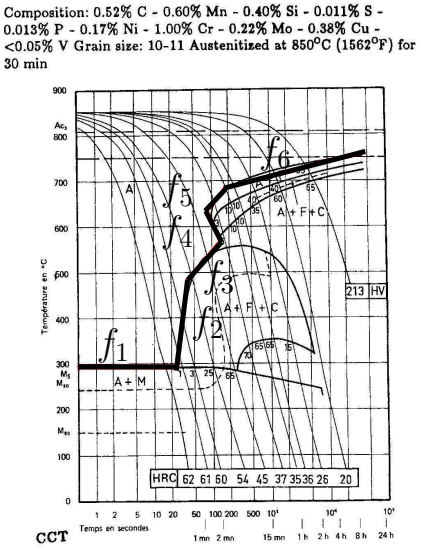
- d'une part à la structure arborescente récursive nécessitant une condition d'arrêt de découpage définie par des degrés de précision minimum : les résultats de filtrage sont donc approchés,
- à des explorations inutiles dans le cas où la cohérence des nœuds est déterminée par l'utilisation de l'arithmétique des intervalles,
- et d'autre part, à la méthode de génération qui n'a pas été conçue pour traiter les contraintes continues numériques binaires définies par morceaux.

5.2 Arbres quaternaires par morceaux

Nous présentons dans cette section deux méthodes de génération d'arbres quaternaires à partir de l'expression symbolique de contraintes continues numériques binaires définies par morceaux.

La prise en compte d'abaques expérimentaux dans un modèle de connaissances à base de contraintes peut être nécessaire dans une problématique d'aide à la conception. Les abaques doivent, avec les méthodes actuelles, être décrits par une seule expression mathématique continue qu'il n'est pas toujours évident d'identifier. Il est souvent plus aisé de les approximer par

des expressions mathématiques définies par morceaux, comme l'illustre la figure 5.9. Pour intégrer ce type de contraintes particulières dans un problème de satisfaction de contraintes, nous complétons la méthode de génération d'arbres proposée par Sam (1995) (Aldanondo et al. (2005a), Vareilles et al. (2005)).



$$\boldsymbol{TRC}(x,y) = \bigcup \begin{cases} \boldsymbol{f_1}(x,y) \text{ sur } D_x^{f_1} \text{ et } D_y^{f_1} \\ \boldsymbol{f_2}(x,y) \text{ sur } D_x^{f_2} \text{ et } D_y^{f_2} \\ \boldsymbol{f_3}(x,y) \text{ sur } D_x^{f_3} \text{ et } D_y^{f_3} \\ \boldsymbol{f_4}(x,y) \text{ sur } D_x^{f_4} \text{ et } D_y^{f_4} \\ \boldsymbol{f_5}(x,y) \text{ sur } D_x^{f_5} \text{ et } D_y^{f_5} \\ \boldsymbol{f_6}(x,y) \text{ sur } D_x^{f_6} \text{ et } D_y^{f_6} \end{cases}$$

FIG. 5.9 – Représentation d'une courbe d'un abaque expérimental sous forme d'une contrainte continue numérique binaire définie par six morceaux

Nous allons, dans un premier temps, définir ce que nous entendons par contraintes continues numériques binaires définies par morceaux, ainsi que les hypothèses que nous avons posées pour pouvoir générer les arbres quaternaires associés à ce type de contraintes. Nous présenterons dans un deuxième temps deux méthodes de génération d'arbres sur des contraintes binaires définies par morceaux. Celles-ci, basées sur la méthode proposée par Sam (1995) et utilisant l'arithmétique des intervalles, permettent de construire les arbres associés à des contraintes binaires d'égalité et d'inégalité par morceaux. Ces méthodes reposent sur l'identification de degrés d'information pertinente des nœuds et sur leur propension à déterminer ou non leur cohérence. En effet, seules certaines régions de l'espace de solution possèdent de l'information. La représentation des contraintes continues numériques binaires définies par morceaux sous forme arborescente ne modifie en rien les techniques de *fusion* et de filtrage des arbres quaternaires développées par Sam (1995).

5.2.1 Définition des contraintes numériques continues définies par morceaux

Nous définissons dans cette sous-section la notion de contraintes continues numériques binaires définies par morceaux. Celles-ci se déclinent en deux familles distinctes : elles peuvent soit définir des égalités par morceaux, soit des inégalités par morceaux. Des hypothèses, visant à garantir la connexité des zones cohérentes et incohérentes, doivent être posées sur la définition du contour des contraintes définies par morceaux pour pouvoir générer les arbres quaternaires associés à ce type de contraintes.

5.2.1.1 Définition

Une contrainte définie par morceaux $\boldsymbol{C}(x, y)$ sur l'espace de recherche $(D_x^{\boldsymbol{C}}, D_y^{\boldsymbol{C}})$ est caractérisée par l'union ou collection de n contraintes numériques continues nommées morceaux $\boldsymbol{c_i}(x, y)$ couvrant chacune une région bien déterminée de l'espace de recherche $(D_x^{c_i}, D_y^{c_i})$ telle que $D_x^{c_i} \subseteq D_x^{\boldsymbol{C}}$ et $D_y^{c_i} \subseteq D_y^{\boldsymbol{C}}$. Une région D_X, D_Y peut être couverte par plusieurs contraintes $\boldsymbol{c_i}(x, y)$.

Définition 18 : ***contrainte continue numérique binaire définie par morceaux***

L'union de n contraintes numériques continues binaires $\boldsymbol{c_i}(x, y)$ couvrant chacune un espace bien déterminé $(D_x^{c_i}, D_y^{c_i})$ définit la contrainte continue numérique binaire définie par morceaux nommée $\boldsymbol{C}(x, y)$ sur l'espace de recherche $D_x^{\boldsymbol{C}}$, $D_y^{\boldsymbol{C}}$ où $\forall i = 1, \ldots, n$, $D_x^{c_i} \subseteq D_x^{\boldsymbol{C}}$ et $D_y^{c_i} \subseteq D_y^{\boldsymbol{C}}$:

$$\bigcup_{i=1}^{n} \{\boldsymbol{c_i}(x, y) \text{ sur } (D_x^{c_i}, D_y^{c_i})\} = \boldsymbol{C}(x, y) \text{ sur } (D_x^{\boldsymbol{C}}, D_y^{\boldsymbol{C}})$$

Les contraintes $\boldsymbol{c_i}(x, y)$ peuvent :

- être des contraintes d'égalité ou d'inégalité,
- représenter des fonctions numériques binaires de $\mathbb{R} \mapsto \mathbb{R}$ ou de $\mathbb{R}^2 \mapsto \mathbb{R}$.

Par exemple, la contrainte définissant l'extérieur d'une forme F, nommée $\boldsymbol{F}(x, y)$ sur l'espace $D_x^{\boldsymbol{F}} = [0, 6.5]$, $D_y^{\boldsymbol{F}} = [0, 7]$ et représentée par la figure 5.10, est modélisée sous forme d'une collection de quatre contraintes numériques couvrant chacune un espace bien défini :

$$\boldsymbol{F}(x, y) \left\| \begin{array}{l} D_x^{\boldsymbol{F}} = [0, 6.5] \\ D_y^{\boldsymbol{F}} = [0, 7] \end{array} \right. = \bigcup \left\{ \begin{array}{lll|l} \boldsymbol{f_1}(x, y) : & (x-3)^2 + (y-4)^2 \geq 4 & & \begin{array}{l} D_x^{f_1} = [3, 5] \\ D_y^{f_1} = [4, 6] \end{array} \\ \hline \boldsymbol{f_2}(x, y) : & y \leq 3x - 11 & & \begin{array}{l} D_x^{f_2} = [4, 5] \\ D_y^{f_2} = [1, 4] \end{array} \\ \hline \boldsymbol{f_3}(x, y) : & y \leq -x + 5 & & \begin{array}{l} D_x^{f_3} = [1, 4] \\ D_y^{f_3} = [1, 4] \end{array} \\ \hline \boldsymbol{f_4}(x, y) : & y \geq x + 3 & & \begin{array}{l} D_x^{f_4} = [1, 3] \\ D_y^{f_4} = [4, 6] \end{array} \end{array} \right.$$

5.2.1.2 Hypothèses

La méthode de génération des arbres quaternaires associés aux contraintes définies par morceaux est basée sur l'identification des zones possédant assez d'information pour déduire leur cohérence. En effet, seuls les espaces couverts par les contraintes numériques continues $\boldsymbol{c_i}(x, y)$ possèdent de l'information. Nous posons donc un certain nombre d'hypothèses visant à garantir la connexité des zones cohérentes et incohérentes. Dans le cas d'égalités par morceaux, il est nécessaire de définir les points extrémités des contraintes, si celles-ci ont un contour

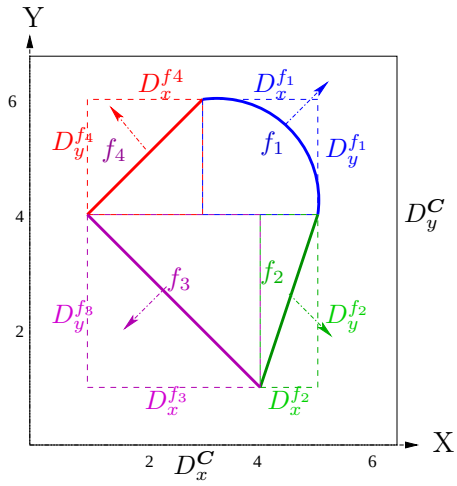


FIG. 5.10 – Exemple de contrainte d'inégalité définie par morceaux

ouvert. Dans le cas des inégalités, la frontière entre les zones cohérentes et incohérentes doit être clairement déterminée : le contour des contraintes d'inégalité doit être obligatoirement fermé. Les contraintes d'inégalité caractérisent des surfaces compatibles qu'il faut arriver à délimiter correctement. Les contraintes $\boldsymbol{c_i}(x, y)$ les composants doivent, par conséquent, être cohérentes entre elles et ne doivent pas se croiser. Nous tenons à noter que le respect de ces hypothèses sont supposées vérifiées avant la discrétisation des contraintes.

Points extrémités et contour fermé Une contrainte définie par morceaux est caractérisée par une collection de contraintes numériques continues binaires couvrant chacune un espace particulier. L'union de ces contraintes définit les zones cohérentes et incohérentes. La frontière entre ces deux zones doit être clairement définie pour éviter de conserver, ou inversement de supprimer, des régions qui n'auraient pas dû l'être. Les égalités par morceaux doivent être bornées par leurs points extrémités (si elles ne présentent pas un contour fermé) et les inégalités doivent obligatoirement avoir un contour explicitement fermé.

La figure 5.11 présente une contrainte binaire d'égalité par morceaux dont le contour est ouvert, composée de deux contraintes f_1 et f_2 sur l'espace de solution $(D_x^{\boldsymbol{C}}, D_y^{\boldsymbol{C}})$ et de deux points extrémités P_1 et P_2.

Morceaux non croisés Si deux morceaux $\boldsymbol{c_j}(x, y)$ et $\boldsymbol{c_k}(x, y)$ composant une contrainte d'inégalité définie par morceaux se croisent, il n'y a plus conservation d'une frontière cohérente entre l'intérieur et l'extérieur de la contrainte définie par morceaux, comme l'illustre la figure 5.12. Les contraintes $\boldsymbol{c_i}(x, y)$ ne doivent donc pas se croiser.

Morceaux cohérents Les morceaux doivent être cohérents entre eux. Dans le cas d'une contrainte d'inégalité définie par morceaux, les morceaux $\boldsymbol{c_i}(x, y)$ doivent définir correctement l'intérieur et l'extérieur. Dans le cas d'une égalité par morceaux, tous les morceaux $\boldsymbol{c_i}(x, y)$ doivent être des contraintes d'égalité.

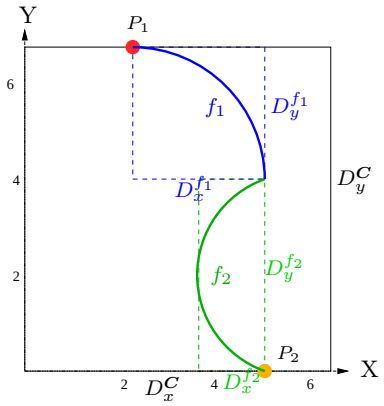


FIG. 5.11 – Exemple de contrainte d'égalité définie par morceaux

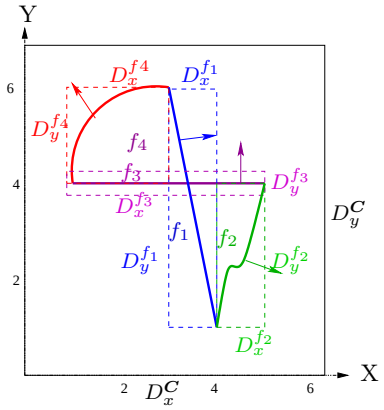


FIG. 5.12 – Exemple de contraintes croisées

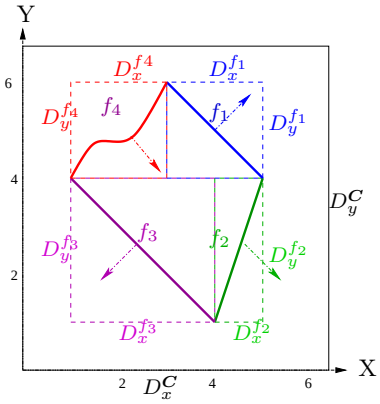


FIG. 5.13 – Exemple de contraintes incohérentes

Par exemple, la contrainte f_4 n'est pas cohérente avec l'ensemble des autres dans la figure 5.13.

5.2.2 Génération

Nous présentons dans cette sous-section deux méthodes de génération d'arbres quaternaires à partir de contraintes définies par morceaux. Celles-ci utilisent la méthode de génération de contrainte numérique continue développée par Sam (1995) et l'arithmétique des intervalles. Dans le cas de contraintes définies par morceaux, seuls les espaces couverts par les contraintes numériques continues $\boldsymbol{c_i}(x, y)$ possèdent de l'information. Il faut donc arriver à identifier ces espaces porteurs d'information pour en déduire les zones cohérentes et incohérentes.

La génération d'arbres quaternaires à partir de contraintes définies par morceaux est basée sur la détection des nœuds porteurs d'information pertinente leur permettant de déduire leur cohérence et l'identification des autres. Nous définissons, dans la première sous-section, les différents degrés d'information pertinente des nœuds : *vide*, *sous-informé*, *frontière* et *sur-frontière*. La première méthode traite les égalités définies par morceaux. Elle est présentée dans la sous-section 5.2.2.2, la seconde, présentée dans la sous-section 5.2.2.3, est dédiée aux inégalités. La méthode de génération des arbres quaternaires associés aux inégalités définies par morceaux nécessite, en plus de l'identification des nœuds ne possédant pas suffisamment d'information, une phase de propagation des zones cohérentes et incohérentes pour déterminer leur cohérence. Les nœuds ne pouvant connaître seuls leur cohérence sont informés de celle-ci par leurs voisins (hypothèse de cohérence par connexité).

5.2.2.1 Degré d'information pertinente

Les méthodes de génération des arbres quaternaires sur des contraintes définies par morceaux, $\boldsymbol{C}(x, y)$ sur $(D_x^{\boldsymbol{C}}, D_y^{\boldsymbol{C}}) = \bigcup_{i=1}^{n} \{\boldsymbol{c_i}(x, y)$ sur $(D_x^{c_i}, D_y^{c_i})\}$ se basent sur la détection des nœuds possédant de l'information pertinente et sur l'identification des autres. En effet, les informations sont réparties dans l'espace de solution : seules les régions de l'espace où les contraintes numériques $\boldsymbol{c_i}(x, y)$ sont définies en possèdent.

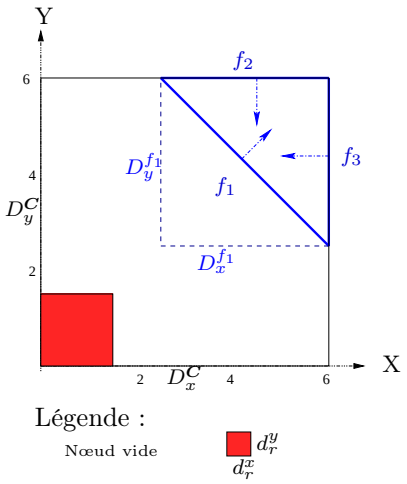


FIG. 5.14 – Exemple de nœud *vide*

Nous distinguons quatre sortes de nœuds à partir de la quantité d'information exploitable qu'ils possèdent pour déduire leur cohérence :

- les nœuds qui ne possèdent aucune information, appelés nœuds *vides*,
- les nœuds qui ne possèdent pas assez d'information, appelés nœuds *sous-informés*,
- les nœuds qui possèdent suffisamment d'information, appelés nœuds *frontières*,
- les nœuds qui possèdent de multiples informations, appelés nœuds *sur-frontières*.

Nous posons les notations suivantes pour définir ces quatre types de nœuds :

- $d_n^x \cap D_x^{c_i} \neq \emptyset \wedge d_n^y \cap D_y^{c_i} \neq \emptyset$ indique qu'un nœud intersecte partiellement ou totalement le domaine de définition de la contrainte $\boldsymbol{c_i}$,
- $\boldsymbol{c_i} \cap n \neq \emptyset$ indique que la contrainte $\boldsymbol{c_i}$ intersecte le nœud n.

Définition 19 : *nœud vide*

Un nœud (d_n^x, d_n^y) *est* vide *s'il n'intersecte aucun domaine de définition de contrainte* $\boldsymbol{c_i}(x, y)$ *définie sur* $(D_x^{c_i}, D_y^{c_i})$ *:*

$$\forall i = 1, \ldots, k \mid (d_n^x \cap D_x^{c_i} = \emptyset) \wedge (d_n^y \cap D_y^{c_i} = \emptyset)$$

La figure 5.14 présente un exemple où un nœud qualifié de *vide* ne peut déduire sa cohérence, puisqu'il ne possède aucune information.

Définition 20 : *nœud sous-informé*

Un nœud n *est* sous-informé *s'il ne possède pas suffisamment d'information lui permettant de définir sa cohérence. C'est-à-dire que le nœud* (d_n^x, d_n^y) *intersecte au moins partiellement ou totalement le domaine de définition* $(D_x^{c_i}, D_y^{c_i})$ *d'une contrainte* $\boldsymbol{c_i}(x, y)$ *sans qu'aucune contrainte ne l'intersecte :*

$$(\exists i \geq 1 \mid (d_n^x \cap D_x^{c_i} \neq \emptyset) \wedge (d_n^y \cap D_y^{c_i} \neq \emptyset)) \wedge ((\forall i = 1 \ldots k, \boldsymbol{c_i} \cap n = \emptyset))$$

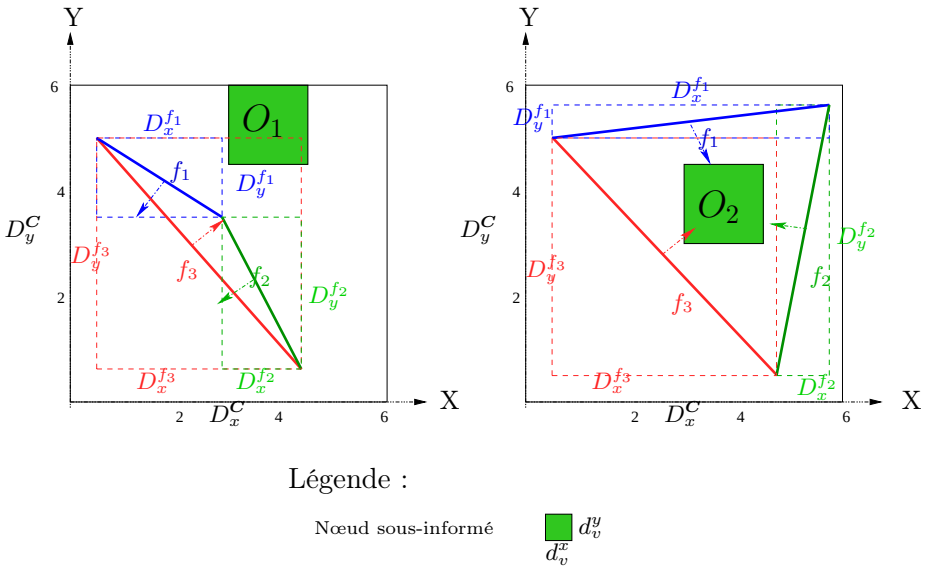


FIG. 5.15 – Exemples de nœuds *sous-informés* intersectant le domaine d'un seul morceaux

La figure 5.15 présente deux exemples où des nœuds qualifiés de *sous-informés* ne peuvent déduire leur cohérence de la seule information fournie par la contrainte sur laquelle ils sont définis :

- dans l'exemple de gauche, le nœud O_1 défini sur (d_v^x, d_v^y) intersecte le domaine de définition de $\boldsymbol{f_3}$. Il est partiellement cohérent avec $\boldsymbol{f_3}$, mais il n'appartient pas à la région cohérente définie par la contrainte d'inégalité par morceaux,
- inversement, dans l'exemple de droite, le nœud O_2 défini sur (d_v^x, d_v^y) intersecte le domaine de définition de $\boldsymbol{f_3}$. Il est cohérent avec $\boldsymbol{f_3}$ et il appartient à la région cohérente définie par la contrainte d'inégalité par morceaux.

Le fait de vérifier la contrainte sur laquelle un nœud est défini n'est donc pas suffisant pour pouvoir statuer sur sa cohérence.

La figure 5.16 présente deux nœuds *sous-informés* intersectant les domaines de deux morceaux qui ne peuvent avec les informations, dont ils disposent décider de leur cohérence :

- la feuille O_3 défini sur (d_v^x, d_v^y) intersecte les domaines de définition des contraintes $\boldsymbol{f_1}$ et $\boldsymbol{f_2}$. O_3 ne vérifie que partiellement la contrainte $\boldsymbol{f_1}$. O_3 appartient au domaine cohérent avec la contrainte d'inégalité par morceaux et sa couleur finale doit être blanche,
- la feuille O_4 défini sur (d_v^x, d_v^y) intersecte les domaines de définition des contraintes $\boldsymbol{f_2}$ et $\boldsymbol{f_3}$. O_4 ne vérifie que partiellement la contrainte $\boldsymbol{f_2}$. O_2 appartient au domaine incohérent avec la contrainte d'inégalité par morceaux et sa couleur finale doit être bleue.

Le nombre de contraintes vérifiées n'est donc pas discriminant pour déduire la couleur terminale des nœuds *sous-informés*.

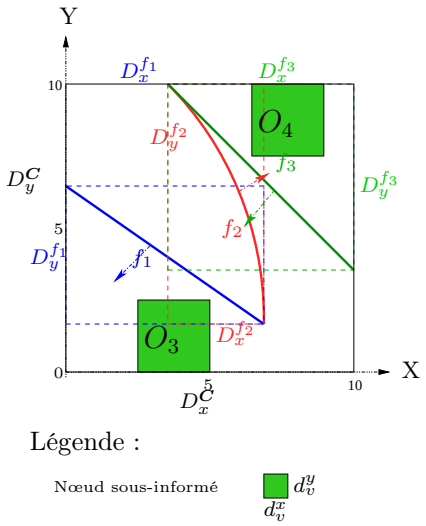


FIG. 5.16 – Nœuds *sous-informés* intersectant les domaines de deux morceaux

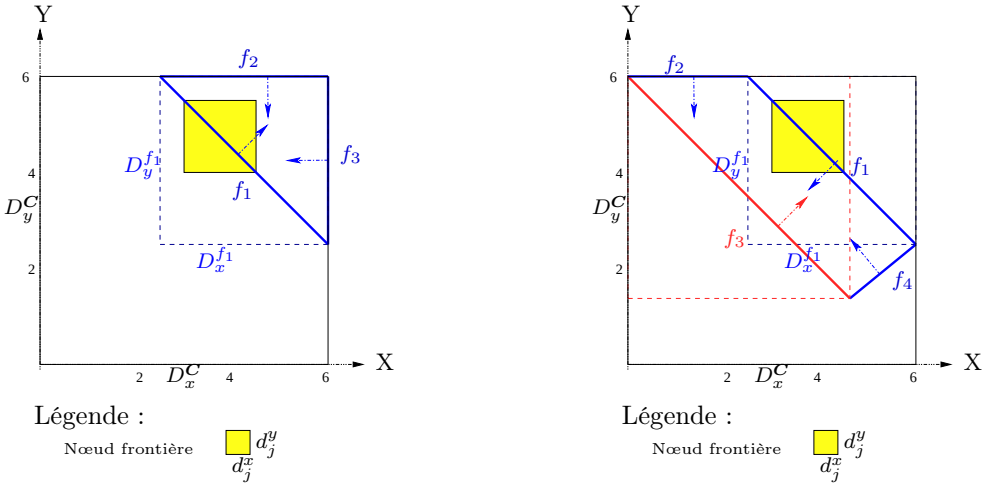


FIG. 5.17 – Exemple de nœuds *frontières*

Définition 21 : *nœud frontière*

Un nœud est porteur d'information si une et une seule contrainte $\boldsymbol{c_i}(x, y)$ *l'intersecte. Le nœud est alors qualifié de* frontière *:*

$$\exists i = 1 \mid ((\boldsymbol{c_i} \cap n \neq \emptyset) \wedge (d_n^x \cap D_x^{c_i} \neq \emptyset) \wedge (d_n^y \cap D_y^{c_i} \neq \emptyset))$$

Il est l'équivalent d'un nœud gris de la méthode de Sam (1995).

La figure 5.17 présente deux exemples de nœuds *frontières*. Ce type de nœud sait qu'il contient des zones cohérentes et incohérentes délimitées par la contrainte qui l'intersecte. Sur la figure de gauche, le nœud *frontière* n'appartient qu'au domaine de la contrainte f_1 et il est intersecté par f_1. Sur la figure de droite, le nœud *frontière* appartient aux domaines de définition de f_1 et de f_3, mais il n'est intersecté que par f_1.

Définition 22 : *nœud sur-frontière*

Un nœud n est sur-frontière *s'il est* frontière *de plus d'une contrainte.*

$$\exists i \geq 2 \mid ((\boldsymbol{c_i} \cap n \neq \emptyset) \wedge (d_n^x \cap D_x^{c_i} \neq \emptyset) \wedge (d_n^y \cap D_y^{c_i} \neq \emptyset))$$

La figure 5.18 présente un exemple de nœuds *sur-frontières*. Ce type de nœud possède une multitude d'informations.

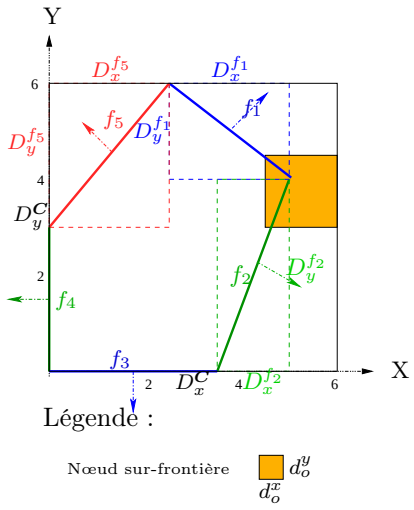


FIG. 5.18 – Exemple de nœud *sur-frontière*

Le tableau 5.2 récapitule les différents degrés d'information pertinente des nœuds illustrés sur deux contraintes $\boldsymbol{c_1}$ et $\boldsymbol{c_2}$. Il peut bien sûr être étendu à plusieurs contraintes. $\boldsymbol{c_i}$ correspond à la contrainte et D^{c_i} correspond au domaine de définition de la contrainte $\boldsymbol{c_i}$. Le marquage « $= \emptyset$ » indique que l'intersection entre le nœud et le domaine de définition $(d_n^x \cap D_x^{c_i})$ ou bien le nœud et la contrainte $\boldsymbol{c_i}$ $(n \cap \boldsymbol{c_i})$ est vide ; le marquage « $\neq \emptyset$ » indique que l'intersection n'est pas vide et « $\forall$ » indique que quelque soit la valeur de l'intersection, le nœud est tout de même du type mentionné.

TAB. 5.2 – Degrés d'information pertinente

Nœud	D^{c_1}	$\boldsymbol{c_1}$	D^{c_2}	$\boldsymbol{c_2}$
vide	$= \emptyset$	$= \emptyset$	$= \emptyset$	$= \emptyset$
sous-informé	$\neq \emptyset$	$= \emptyset$	$\forall$	$= \emptyset$
frontière	$\neq \emptyset$	$\neq \emptyset$	$\forall$	$= \emptyset$
sur-frontière	$\neq \emptyset$	$\neq \emptyset$	$\neq \emptyset$	$\neq \emptyset$

L'espace de solution est ainsi partitionné suivant les quatre degrés d'information pertinente. La figure 5.19 illustre ce partitionnement de l'espace sur la contrainte *Penta* définie par morceaux.

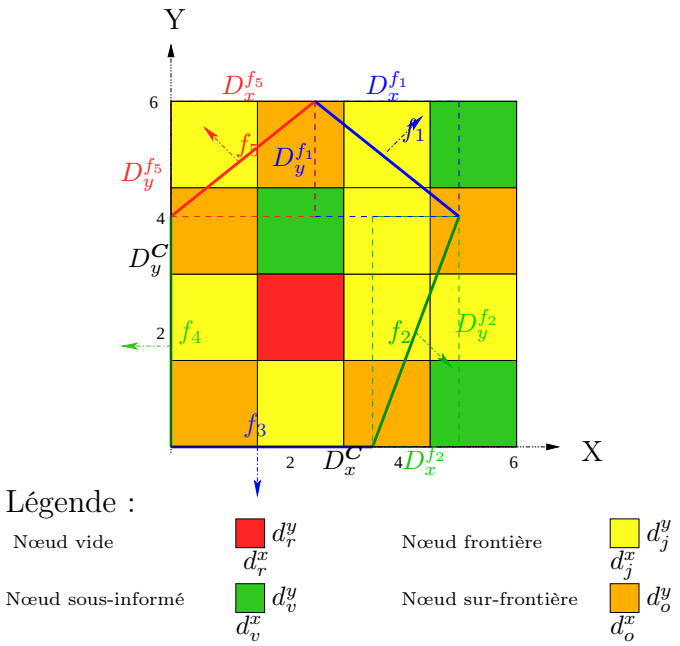


FIG. 5.19 – Degré d'information pertinente des nœuds sur la contrainte $\boldsymbol{Penta}(x, y)$

$$\boldsymbol{Penta}(x,y) \left\| \begin{array}{l} D_x^{\boldsymbol{Penta}} = [0,6] \\ D_y^{\boldsymbol{Penta}} = [0,6] \end{array} \right. = \bigcup \left\{ \begin{array}{lll} \boldsymbol{f_1}(x,y): & y \geq -0.8 \times x + 8 & \left\| \begin{array}{l} D_x^{f_1} = [2.5,5] \\ D_y^{f_1} = [4,6] \end{array} \right. \\ \boldsymbol{f_2}(x,y): & y \leq \frac{8}{3} \times x - \frac{28}{3} & \left\| \begin{array}{l} D_x^{f_2} = [3.5,5] \\ D_y^{f_2} = [0,4] \end{array} \right. \\ \boldsymbol{f_3}(x,y): & y \leq 0 & \left\| \begin{array}{l} D_x^{f_3} = [0,3.5] \\ D_y^{f_3} = 0 \end{array} \right. \\ \boldsymbol{f_4}(y,x): & x \leq 0 & \left\| \begin{array}{l} D_x^{f_4} = 0 \\ D_y^{f_4} = [0,4] \end{array} \right. \\ \boldsymbol{f_5}(x,y): & y \geq \frac{4}{5} \times x + 4 & \left\| \begin{array}{l} D_x^{f_5} = [0,2.5] \\ D_y^{f_5} = [4,6] \end{array} \right. \end{array} \right.$$

5.2.2.2 Génération des égalités par morceaux

Les contraintes d'égalité définissent comme zone cohérente l'enveloppe des contraintes. Dans le cas de contraintes d'égalité définies par morceaux $\boldsymbol{C}(x, y)$ sur $(D_x^{\boldsymbol{C}}, D_y^{\boldsymbol{C}}) = \bigcup_{i=1}^{n} \{\boldsymbol{c_i}(x, y)$ sur $(D_x^{c_i}, D_y^{c_i})\}$, seules les feuilles *frontières* et *sur-frontières* sont cohérents avec $\boldsymbol{C}(x, y)$. La génération d'arbres quaternaires à partir d'égalités définies par morceaux est donc basée sur la détection des nœuds *frontières* et *sur-frontières*.

Les nœuds *vides* et *sous-informés* deviennent des feuilles bleues, puisqu'ils ne définissent pas l'enveloppe de la contrainte définie par morceaux.

Les nœuds *frontières* peuvent se colorer de deux manières, suivant leur grain :

– si le nœud *frontière* n n'est pas unitaire, il devient gris et doit être décomposé : l'arbre quaternaire associé à la contrainte numérique isolée $\boldsymbol{c_i}(x, y)$ peut être généré sur l'espace (d_n^x, d_n^y) du nœud par l'appel à la fonction GÉNÉRER ARBRE QUATERNAIRE(n), présentée par l'algorithme 8 page 64. Le nœud est ainsi décomposé en zones cohérentes et incohérentes avec $\boldsymbol{c_i}(x, y)$.

– si le nœud est unitaire, il devient une feuille blanche, puisqu'il définit l'enveloppe de la contrainte.

Les nœuds *sur-frontières* peuvent être colorés de plusieurs manières :

– si le nœud *sur-frontière* n n'est pas unitaire, il devient gris et doit être décomposé pour isoler les nœuds :

  – *vides* et *sous-informés* qui deviennent des feuilles bleues,

  – *frontières* qui doivent être décomposés en zones cohérentes et incohérentes, comme défini précédemment,

  – *sur-frontières* qui doivent à nouveau être décomposés par rapport à la contrainte définie par morceaux.

– si le nœud *sur-frontière* est unitaire, il devient une feuille blanche, puisqu'il définit l'enveloppe de la contrainte.

Le phénomène d'absorption de nœuds fils monochromatiques de couleur terminale blanche ou bleue par leur père existe aussi dans la méthode de génération des arbres quaternaires sur des contraintes d'égalité par morceaux. Lorsqu'un nœud père a ses quatre fils de la même couleur, il les absorbe et se change en feuille monochrome.

La procédure QT ÉGAL, présentée par l'algorithme 12, construit l'arbre quaternaire $T_{(x,y)}$ associé à une contrainte d'égalité binaire définie par morceaux $\boldsymbol{C}(x, y)$ sur $(D_x^{\boldsymbol{C}}, D_y^{\boldsymbol{C}}) = \bigcup_{i=1}^{n} \{\boldsymbol{c_i}(x, y)$ sur $(D_x^{c_i}, D_y^{c_i})\}$. Cette fonction fait appel à la fonction GÉNÉRER ÉGAL, présentée par l'algorithme 13, qui décompose l'espace de recherche à partir d'un nœud.

Alg. 12 QT ÉGAL(CONTRAINTE : $\boldsymbol{C}_{(x,y)}$, INTERVALLE : $D_x^{\boldsymbol{C}}$, INTERVALLE : $D_y^{\boldsymbol{C}}$)

– ∴ – ***Cet algorithme génère un arbre quaternaire*** $T_{(x,y)}$ ***à partir d'une contrainte d'égalité définie par morceaux*** $\boldsymbol{C}_{(x,y)}$ ***et d'un espace de recherche*** $D_x^{\boldsymbol{C}}$, $D_y^{\boldsymbol{C}}$.
– ∴ – $D_x^{\boldsymbol{C}}$ *correspond à l'intervalle de solution suivant* x
– ∴ – $D_y^{\boldsymbol{C}}$ *correspond à l'intervalle de solution suivant* y
Création du nœud racine r de $T_{(x,y)}$ contraint par $\boldsymbol{C}_{(x,y)}$ et portant sur $(D_x^{\boldsymbol{C}}, D_y^{\boldsymbol{C}})$
$r \leftarrow$ GÉNÉRER ÉGAL(r) (algorithme 13 page suivante)
– ∴ – *Retourner l'arbre associé à la contrainte* $\boldsymbol{C}_{(x,y)}$
Retourner $(T_{(x,y)})$

Les zones cohérentes et incohérentes avec la contrainte $\boldsymbol{C}(x, y)$ sur $(D_x^{\boldsymbol{C}}, D_y^{\boldsymbol{C}}) = \bigcup_{i=1}^{n} \{\boldsymbol{c_i}(x, y)$ sur $(D_x^{c_i}, D_y^{c_i})\}$ sont ainsi identifiées. Il faut alors reconstruire les domaines des variables participant à la contrainte définie par morceaux et les propager aux autres contraintes. Ces mécanismes, communs aux contraintes d'égalité et d'inégalité par morceaux, seront présentés dans la sous-section 5.2.3.

Alg. 13 GÉNÉRER ÉGAL(NOEUD : n)

– $\therefore$ – ***Cet algorithme construit l'arbre quaternaire*** $T_{(x,y)}$ ***d'une contrainte d'égalité définie par morceaux*** $\boldsymbol{C}_{(x,y)}$ ***à partir d'un nœud.***
– $\therefore$ – ϵ_x *correspond au degré de précision de la variable* x
– $\therefore$ – ϵ_y *correspond au degré de précision de la variable* y
– $\therefore$ – $d_n^x = [\overline{x}, \underline{x}]$ *et* $d_n^y = [\overline{y}, \underline{y}]$ *correspondent au sous-espace du nœud courant*
Tant Que $(\overline{x} - \underline{x} > \epsilon_x \vee \overline{y} - \underline{y} > \epsilon_y)$ **Faire**
  – $\therefore$ – *La précision n'est pas atteinte en* x *ou en* y
  Création et étiquetage des quatre fils du nœud courant n
  Pour tous les fils f du nœud courant n **Faire**
    Contraintes $\leftarrow$ liste des contraintes $\boldsymbol{c_i}$ intersectant le nœud f : $\boldsymbol{c_i} \cap f \neq \emptyset$
    Si $(Contraintes = 0)$ **Alors**
      – $\therefore$ – *Le nœud* f *est* vide *ou* sous-informé
      – $\therefore$ – *Le nœud* f *se colore donc en bleu*
      $couleur(f) \leftarrow$ bleu
    Sinon
      – $\therefore$ – *Une seule contrainte* $\boldsymbol{c_i}(x, y)$ *intersecte le nœud* f
      Si $(Contraintes = 1)$ **Alors**
        – $\therefore$ – *Le nœud* f *est* frontière *d'une seule contrainte* $\boldsymbol{c_i}(x, y)$
        – $\therefore$ – *Génération de l'arbre quaternaire correspondant à la contrainte isolée* $\boldsymbol{c_i}(x, y)$ *sur* (d_f^x, d_f^y)
        $f \leftarrow$ GÉNÉRER ARBRE QUATERNAIRE(f) (algorithme 8 page 64)
      Sinon
        – $\therefore$ – *Plusieurs contraintes intersectent le nœud* f
        – $\therefore$ – *Le nœud* f *est donc* sur-frontière
        – $\therefore$ – *Il faut le décomposer*
        $f \leftarrow$ GÉNÉRER ÉGAL(f) (algorithme 13)
      Fin Si
    Fin Si
  Fin Pour
  Absorption des nœuds fils f monochromes blancs et bleus par leur nœud père n
Fin Tant Que
– $\therefore$ – *Le nœud* n *est unitaire*
Si (le nœud n est *frontière unitaire* ou *sur-frontière unitaire*) **Alors**
  – $\therefore$ – *Il enveloppe la contrainte* $C(x, y)$
  $couleur(n) \leftarrow$ blanc
Sinon
  – $\therefore$ – *Dans tous les autres cas, le nœud n'enveloppe pas la contrainte* $C(x, y)$
  $couleur(n) \leftarrow$ bleu
Fin Si
– $\therefore$ – *Retourner le nœud courant*
Retourner n

Prenons l'exemple d'un carré de côté de longueur 1.2 représenté par une contrainte d'égalité définie par morceaux $\boldsymbol{Carre}(x,y)$ portant sur $D_x^{\boldsymbol{C}} = [0,2]$, $D_y^{\boldsymbol{C}} = [0,2]$ où $\epsilon_x = \epsilon_y = 0.125$:

$$\boldsymbol{Carre}(x,y) \left\| \begin{array}{l} D_x^{\boldsymbol{Carre}} = [0,2] \\ D_y^{\boldsymbol{Carre}} = [0,2] \end{array} \right. = \bigcup \left\{ \begin{array}{lll} \boldsymbol{c_1}(y,x): & x = 1.6 & \left\| \begin{array}{l} D_x^{c_1} = 1.6 \\ D_y^{c_1} = [0.4, 1.6] \end{array} \right. \\ \boldsymbol{c_2}(x,y): & y = 1.6 & \left\| \begin{array}{l} D_x^{c_2} = [0.4, 1.6] \\ D_y^{c_2} = 1.6 \end{array} \right. \\ \boldsymbol{c_3}(y,x): & x = 0.4 & \left\| \begin{array}{l} D_x^{c_3} = 0.4 \\ D_y^{c_3} = [0.4, 1.6] \end{array} \right. \\ \boldsymbol{c_4}(x,y): & y = 0.4 & \left\| \begin{array}{l} D_x^{c_4} = [0.4, 1.6] \\ D_y^{c_4} = 0.4 \end{array} \right. \end{array} \right.$$

Nous obtenons la représentation sous forme d'arbre quaternaire présentée par la figure 5.20.

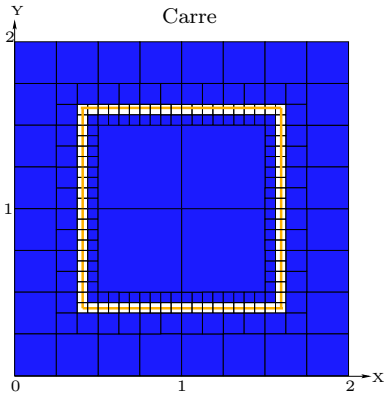


FIG. 5.20 – Arbre quaternaire associé à la contrainte d'égalité définie par morceaux $\boldsymbol{Carre}$

5.2.2.3 Génération des inégalités par morceaux

Les contraintes d'inégalité définissent des surfaces compatibles qu'il faut arriver à identifier. Les nœuds ne possédant pas suffisamment d'information pertinente doivent arriver à déduire leur cohérence : soit ils appartiennent aux zones compatibles avec la contrainte définie par morceaux, soit non. Les feuilles *informatrices*, possédant de l'information pertinente, vont les informer de leur cohérence et propager ainsi, par voisinage, les zones cohérentes et incohérentes. Il faut, pour cela, que la taille minimale du grain unitaire permette d'isoler ces feuilles *informatrices*. Si le grain est trop élevé, les feuilles *informatrices* ne pourront être identifiées et la totalité de l'espace de recherche sera considéré comme cohérent.

La méthode de génération des arbres quaternaires sur des inégalités définies par morceaux nécessite de renseigner sur leur cohérence les nœuds qui ne peuvent la déterminer seuls. Les différents degrés d'information pertinente sont matérialisés par des couleurs particulières. Lorsque le graphe est coloré de ces couleurs particulières et des couleurs terminales (blanc et bleu), la propagation d'informations a lieu. Nous proposons plusieurs règles de coloriage dépendant de la couleur initiale des feuilles *informatrices*. La méthode de recherche de voisins est basée sur l'étiquetage des nœuds suivant la courbe de Péano en N ordonnée par Morton. Nous présentons

l'algorithme de génération d'arbres quaternaires à partir de contraintes numériques d'inégalité définies par morceaux qui identifie et matérialise, dans un premier temps, le degré d'information pertinente de chaque nœud et propage, dans un deuxième temps, les informations des feuilles *informatrices* vers celles qui ne peuvent connaître seules leur cohérence.

5.2.2.4 Matérialisation des degrés d'information pertinente

Nous avons défini, dans la sous-section 5.2.2.1, différents degrés d'information pertinente des nœuds :

- les nœuds qui ne possèdent aucune information, appelés nœuds *vides*,
- les nœuds qui ne possèdent pas assez d'information, appelés nœuds *sous-informés*,
- les nœuds qui possèdent suffisamment d'information, appelés nœuds *frontières*,
- les nœuds qui possèdent de multiples informations, appelés nœuds *sur-frontières*.

Les différents degrés d'information pertinente des nœuds sont matérialisés par des couleurs particulières. Nous associons :

- la couleur **rouge** aux nœuds *vides*,
- la couleur **verte** aux nœuds *sous-informés*,
- la couleur **jaune** aux nœuds *frontières unitaires* (degré de précision atteint) et
- la couleur **orange** aux nœuds *sur-frontières unitaires* (degré de précision atteint).

L'attribution d'une couleur à un nœud découle de son degré d'information pertinente et de son grain :

- les nœuds *vides* deviennent des feuilles rouges,
- les nœuds *sous-informés* deviennent des feuilles vertes,
- les nœuds *frontières* peuvent se colorer de deux manières suivant leur grain :
  - si le nœud n n'est pas unitaire, il devient gris et doit alors être décomposé : l'arbre quaternaire associé à la contrainte numérique isolée $\boldsymbol{c_i}(x, y)$ peut être généré sur l'espace (d_n^x, d_n^y) du nœud par l'appel à la fonction Générer Arbre quaternaire(n), présentée par l'algorithme 8 page 64. Le nœud est ainsi décomposé uniquement en zones cohérentes (colorées en blanc) et incohérentes (colorées en bleu) avec $\boldsymbol{c_i}(x, y)$, ce qui explique que la frontière des contraintes soit blanche par endroit.
  - si le nœud est *frontière unitaire*, il devient une feuille jaune,
- les nœuds *sur-frontières* peuvent être colorés de plusieurs manières :
  - si le nœud *sur-frontière* n n'est pas unitaire, il devient gris et doit être décomposé pour isoler les nœuds :
    - *vides* qui deviennent des feuilles rouges,
    - *sous-informés* qui deviennent des feuilles vertes,
    - *frontières* qui doivent être décomposés en zones cohérentes et incohérentes avec la contrainte isolée par l'appel à la fonction Générer Arbre quaternaire(n), présentée par l'algorithme 8 page 64,

– *sur-frontières* qui doivent à nouveau être décomposés pour isoler les nœuds *vides*, *sous-informés*, *frontières* et *sur-frontières* par rapport à la contrainte définie par morceaux.

– si le nœud est *sur-frontière unitaire*, il devient une feuille orange.

Nous illustrons ces règles de propagation sur deux contraintes définies par morceaux : ***Losange*** et ***Borne***.

$$\boldsymbol{Losange}(x,y) \left\| \begin{array}{l} D_x^{\boldsymbol{L}} = [0.2, 2.2] \\ D_y^{\boldsymbol{L}} = [0.2, 2.2] \end{array} \right. = \bigcup \left\{ \begin{array}{lll} \boldsymbol{f_1}(x,y): & y \geq x + 0.6 & \left\| \begin{array}{l} D_x^{f_1} = [0.4, 1] \\ D_y^{f_1} = [1, 1.6] \end{array} \right. \\ \boldsymbol{f_2}(x,y): & y \geq 2.6 - x & \left\| \begin{array}{l} D_x^{f_2} = [1, 1.6] \\ D_y^{f_2} = [1, 1.6] \end{array} \right. \\ \boldsymbol{f_3}(x,y): & y \leq 1.4 - x & \left\| \begin{array}{l} D_x^{f_3} = [0.4, 1] \\ D_y^{f_3} = [0.4, 1] \end{array} \right. \\ \boldsymbol{f_4}(x,y): & y \leq x - 0.6 & \left\| \begin{array}{l} D_x^{f_4} = [1, 1.6] \\ D_y^{f_4} = [1, 1.6] \end{array} \right. \end{array} \right.$$

$$\boldsymbol{Borne}(x,y) \left\| \begin{array}{l} D_x^{\boldsymbol{B}} = [0, 4] \\ D_y^{\boldsymbol{B}} = [0, 4] \end{array} \right. = \bigcup \left\{ \begin{array}{lll} \boldsymbol{f_1}(x,y): & (x-2)^2 + (y-2)^2 \leq 1 & \left\| \begin{array}{l} D_x^{f_1} = [1, 3] \\ D_y^{f_1} = [2, 3] \end{array} \right. \\ \boldsymbol{f_2}(x,y): & y \geq 1 & \left\| \begin{array}{l} D_x^{f_2} = [1, 3] \\ D_y^{f_2} = 1 \end{array} \right. \\ \boldsymbol{f_3}(y,x): & x \geq 1 & \left\| \begin{array}{l} D_x^{f_3} = 1 \\ D_y^{f_3} = [1, 2] \end{array} \right. \\ \boldsymbol{f_4}(y,x): & x \leq 3 & \left\| \begin{array}{l} D_x^{f_4} = 3 \\ D_y^{f_4} = [1, 2] \end{array} \right. \end{array} \right.$$

Les arbres quaternaires associés aux contraintes d'inégalité définies par morceaux ***Losange*** définissant l'extérieur d'un losange avec $\epsilon_x = \epsilon_y = 0.125$ et ***Borne*** définissant l'intérieur d'une borne avec $\epsilon_x = \epsilon_y = 0.25$ sont présentés par la figure 5.21 avant propagation des couleurs des nœuds.

5.2.2.5 Règles de propagation de couleurs

Nous proposons plusieurs règles de propagation d'informations, fonction de la couleur initiale des feuilles *informatrices*. Les règles que nous proposons sont appliquées dans l'ordre de présentation pour s'assurer que la frontière entre les zones cohérentes et incohérentes avec la contrainte globale $\boldsymbol{C}(x,y)$ reste bien délimitée.

Trois types de feuilles sont des feuilles *informatrices* : les feuilles *frontières unitaires* jaunes, les feuilles incohérentes bleues et les feuilles cohérentes blanches. Les feuilles *sur-frontières unitaires* oranges ne peuvent indiquer leur couleur à leurs feuilles voisines, puisqu'elles se

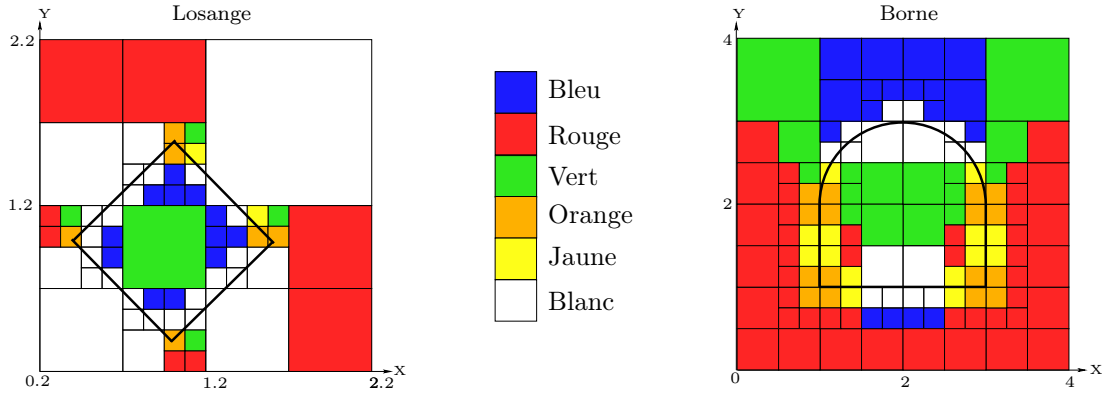


FIG. 5.21 – Matérialisation des degrés d'information pertinente des nœuds sur les contraintes ***Losange*** et ***Borne***

trouvent à la jonction de plusieurs contraintes. Elles ne peuvent choisir la contrainte qui délimite de manière irréfutable l'espace globalement cohérent.

Nous illustrons ces règles sur les contraintes d'inégalité par morceaux définies précédemment $\boldsymbol{Losange}(x, y)$ et $\boldsymbol{Borne}(x, y)$.

Propagation des feuilles jaunes Les feuilles *frontières* jaunes savent qu'elles sont traversées par une seule contrainte $\boldsymbol{c_i}(x, y)$. Elles peuvent donc indiquer à leurs feuilles voisines *sous-informées* vertes et *vides* rouges portant sur les sous-espaces (d_n^x, d_n^y), qui ne connaissent pas leur cohérence, si ces dernières se trouvent du bon ou du mauvais côté de la frontière définie par $\boldsymbol{c_i}(x, y)$. Il faut, pour cela, vérifier qu'un point du nœud (d_n^x, d_n^y) vérifie la contrainte $\boldsymbol{c_i}(x, y)$. Si tel est le cas, la feuille est cohérente et se colore en blanc. Sinon, elle est incohérente et se colore en bleu.

La règle précédemment énoncée est réalisée par l'algorithme 14 nommé FRONTIÈRE.

Le phénomène de propagation des feuilles jaunes sur les feuilles vertes et rouges est illustré par la figure 5.22.

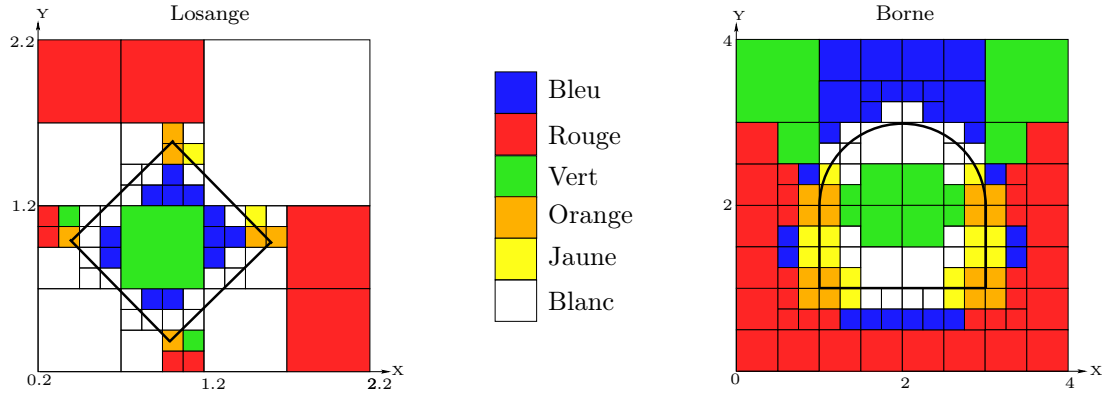


FIG. 5.22 – Propagation des informations des feuilles jaunes sur leurs feuilles voisines vertes et rouges

Alg. 14 FRONTIÈRE(ARBRE : $T_{(x,y)}$)

– $\therefore$ – ***Cet algorithme permet de colorer les feuilles vides et sous-informées à partir des feuilles frontières unitaires.***
– $\therefore$ – *$T_{(x,y)}$ correspond à l'arbre quaternaire associé à $\boldsymbol{C}(x, y)$*
– $\therefore$ – *$Noeuds$ correspond aux nœuds restant à explorer*
$Noeuds \leftarrow$ la racine de $T_{(x,y)}$
– $\therefore$ – *Tant qu'il reste des nœuds à explorer*
Tant Que ($Noeuds \neq \emptyset$) **Faire**
    Dépiler un nœud n de $Noeuds$
    Si (la couleur de n est jaune) **Alors**
        – $\therefore$ – *$Voisins$ correspond à toutes les feuilles voisines de n*
        $Voisins \leftarrow$ la liste des feuilles voisines de n
        Pour toutes les feuilles vertes et rouges v de $Voisins$ **Faire**
            – $\therefore$ – *$c_i(x, y)$ correspond à la contrainte qui traverse le nœud jaune n*
            Tester la cohérence de v par rapport à la contrainte associée à n
            Si (v est cohérent) **Alors**
                – $\therefore$ – *v fait partie de la zone cohérente avec la contrainte définie par morceaux, il se colore en blanc*
                $couleur(v) \leftarrow$ blanc
            Sinon
                – $\therefore$ – *v fait partie de la zone incohérente avec la contrainte définie par morceaux, il se colore en bleu*
                $couleur(v) \leftarrow$ bleu
            Fin Si
        Fin Pour
    Sinon
        – $\therefore$ – *n n'est pas une feuille jaune*
        Si (la couleur de n est grise) **Alors**
            – $\therefore$ – *n est un nœud dont il faut explorer les fils*
            Empiler dans $Noeuds$ les quatre fils de n
        Fin Si
    Fin Si
Fin Tant Que

Propagation des feuilles bleues Les feuilles incohérentes bleues savent qu'elles appartiennent à l'espace incohérent avec la contrainte globale $\boldsymbol{C}(x, y)$. Elles vont donc informer leurs feuilles voisines *sous-informées* vertes et *vides* rouges qu'elles se trouvent elles aussi du mauvais côté de la frontière.

La règle précédemment énoncée est réalisée par l'algorithme 15 nommé BLEUIR.

Alg. 15 BLEUIR(ARBRE : $T_{(x,y)}$)

```
– ∴ – Cet algorithme propage, par voisinage, les zones incohérentes.
– ∴ – T_(x,y) correspond à l'arbre quaternaire associé à C(x, y)
– ∴ – Noeuds correspond aux nœuds restant à parcourir.
Noeuds ← la racine de T_(x,y)
– ∴ – Tant qu'il reste des nœuds à explorer
Tant Que (Noeuds ≠ ∅) Faire
    Dépiler un nœud n de Noeuds
    Si (la couleur de n est bleue) Alors
        – ∴ – Voisins correspond à toutes les feuilles voisines de n
        Voisins ← la liste des feuilles voisines de n
        Pour toutes les feuilles voisines verts et rouges v de Voisins Faire
            couleur(v) ← bleu
            Empiler dans Noeuds le nouveau nœud incohérent v
        Fin Pour
    Sinon
        – ∴ – n n'est pas une feuille bleue
        Si (la couleur de n est grise) Alors
            – ∴ – n est un nœud dont il faut explorer les fils
            Empiler dans Noeuds les quatre fils de n
        Fin Si
    Fin Si
Fin Tant Que
```

Le phénomène de propagation des feuilles bleues sur les feuilles vertes et rouges est illustré par la figure 5.23.

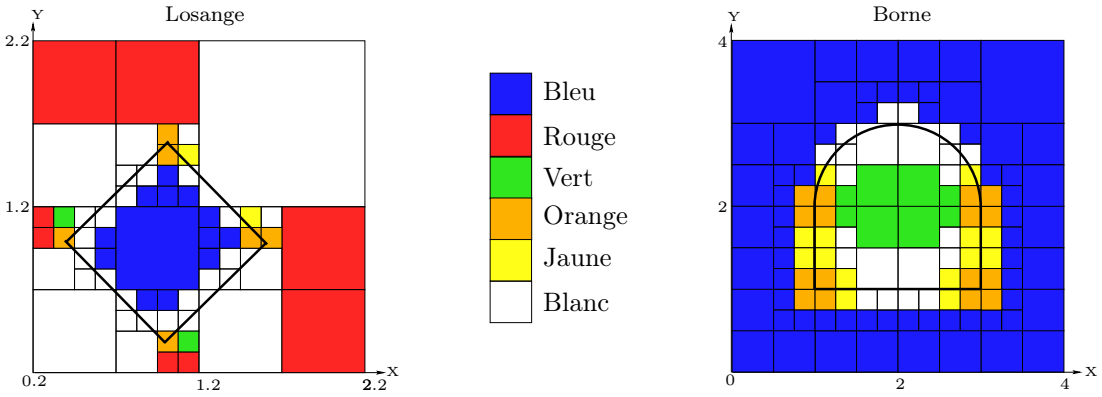


FIG. 5.23 – Propagation des feuilles bleues vers leurs feuilles voisines vertes et rouges

Propagation des feuilles blanches Les feuilles cohérentes blanches savent qu'elles appartiennent à l'espace cohérent avec la contrainte globale $\boldsymbol{C}(x, y)$. Elles vont donc informer leurs feuilles voisines *sous-informées* vertes et *vides* rouges qu'elles se trouvent elles aussi du bon côté de la frontière.

La règle précédemment énoncée est réalisée par l'algorithme 16 nommé BLANCHIR.

Alg. 16 BLANCHIR(ARBRE : $T_{(x,y)}$)

```
– ∴ – Cet algorithme propage, par voisinage, les zones cohérentes.
– ∴ – T(x,y) correspond à l'arbre quaternaire associé à C(x, y)
– ∴ – Noeuds correspond aux nœuds restant à parcourir.
Noeuds ← la racine de T(x,y)
– ∴ – Tant qu'il reste des nœuds à explorer
Tant Que (Noeuds ≠ ∅) Faire
    Dépiler un nœud n de Noeuds
    Si (la couleur de n est blanche) Alors
        – ∴ – Voisins correspond à toutes les feuilles voisines de n
        Voisins ← la liste des feuilles voisines de n
        Pour toutes les feuilles voisines vertes et rouges v de Voisins Faire
            couleur(v) ← blanc
            Empiler dans Noeuds le nouveau nœud cohérent v
        Fin Pour
    Sinon
        – ∴ – n n'est pas une feuille blanche
        Si (la couleur de n est grise) Alors
            – ∴ – n est un nœud dont il faut explorer les fils
            Empiler dans Noeuds les quatre fils de n
        Fin Si
    Fin Si
Fin Tant Que
```

Le phénomène de propagation des feuilles blanches sur les feuilles vertes et rouges est illustré par la figure 5.24.

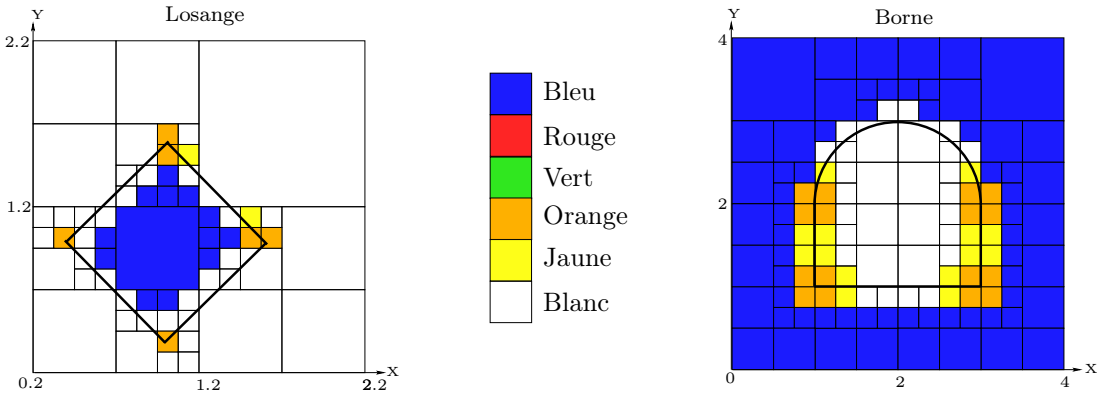


FIG. 5.24 – Propagation des feuilles blanches sur leurs feuilles voisines vertes et rouges

Règles de coloriage des feuilles jaunes et oranges Les feuilles *frontières* jaunes et *sur-frontières* oranges deviennent soit cohérentes avec la contrainte globale, si seul ce qui est totalement incohérent avec la contrainte est à exclure et se colorent en blanc, soit incohérentes avec la contrainte, si seul ce qui est totalement cohérent est à garder et se colorent en bleu.

La coloration des feuilles *frontières* et *sur-frontières* est réalisée par la procédure CONTOUR, présentée par l'algorithme 17. Dans notre application, seul ce qui est totalement incohérent avec $\boldsymbol{C}(x, y)$ est à exclure : les feuilles jaunes et oranges se colorent donc en blanc.

Alg. 17 CONTOUR(ARBRE : $T_{(x,y)}$)

– ∴ – ***Cet algorithme colore les feuilles frontières et sur-frontières en blanc.***
– ∴ – $T_{(x,y)}$ *correspond à l'arbre quaternaire associé à* $C(x, y)$
– ∴ – *Noeuds correspond aux nœuds restant à parcourir.*
Noeuds ← la racine de $T_{(x,y)}$
– ∴ – *Tant qu'il reste des nœuds à explorer*
Tant Que (*Noeuds* $\neq \emptyset$) **Faire**
  Dépiler un nœud n de *Noeuds*
  Si (la couleur de n est jaune ou orange) **Alors**
    – ∴ – *Il faut le colorer en blanc*
    couleur(n) ← blanc
  Sinon
    – ∴ – n *n'est ni une feuille* frontière*, ni une feuille* sur-frontière
    Si (la couleur de n est grise) **Alors**
      – ∴ – n *est un nœud dont il faut explorer les fils*
      Empiler dans *Noeuds* les quatre fils de n
    Fin Si
  Fin Si
Fin Tant Que
Absorption des nœuds fils monochromes blancs par leur nœud père

Les zones cohérentes avec la contrainte globale définie par morceaux sont ainsi identifiées. La figure 5.25 présente le résultat obtenu sur les contraintes $\boldsymbol{Losange}(x, y)$ et $\boldsymbol{Borne}(x, y)$ dans le cas où seul ce qui est totalement incohérent est à exclure.

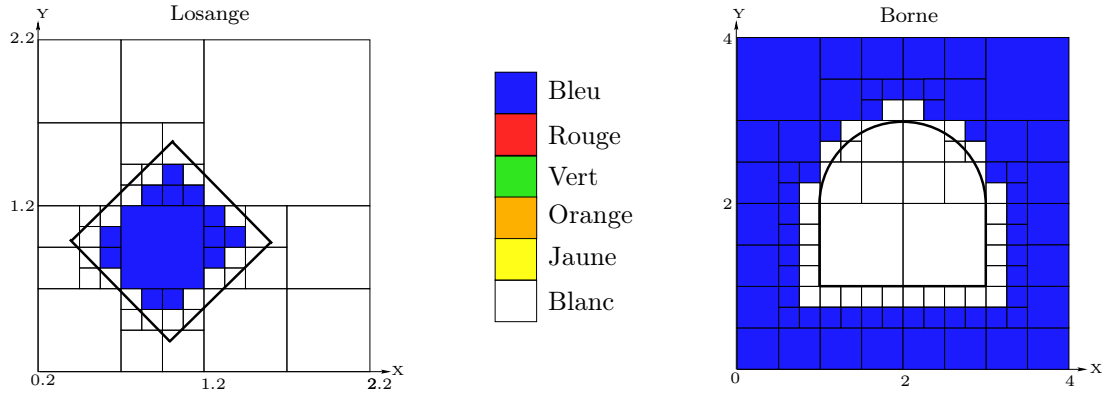


FIG. 5.25 – Arbres quaternaires finaux : règle de coloriage des feuilles unitaires jaunes et oranges

5.2.2.6 Notion de voisinage

La propagation d'informations permettant de définir les zones cohérentes et incohérentes avec une inégalité définie par morceaux se fait à partir des feuilles *informatrices* jaunes, bleues et blanches vers leurs feuilles voisines vertes et rouges. Une feuille peut avoir des feuilles voisines d'un grain différent du sien : soit supérieur, soit inférieur à elle.

Nous considérons comme voisin d'une feuille *informatrice* uniquement les **feuilles** qui lui sont voisines (et pas les nœuds intermédiaires). C'est, en effet, sur les feuilles colorées de l'arbre que la propagation d'informations a lieu. La méthode de recherche de voisins est basée sur l'étiquetage des nœuds suivant la courbe de Péano en N ordonnée par Morton introduite dans la section 5.1.3.

Voisins de même grain Les coordonnées des quatre voisins de même grain d'un nœud s'obtiennent en retranchant (resp. ajoutant) une unité à ses coordonnées.

Par exemple, les voisins du nœud $(4,4)_3$ sont les nœuds de coordonnées :

- voisin de droite : $(100,100) + (001,000) = (101,100) \Leftrightarrow (5,4)_3$
- voisin de gauche : $(100,100) - (001,000) = (011,100) \Leftrightarrow (3,4)_3$
- voisin du dessus : $(100,100) + (000,001) = (100,101) \Leftrightarrow (4,5)_3$
- voisin du dessous : $(100,100) - (000,001) = (100,011) \Leftrightarrow (4,3)_3$

comme le montre le tableau 5.3.

TAB. 5.3 – Voisins de même grain

	$(4,5)_3$	
$(3,4)_3$	$(4,4)_3$	$(5,4)_3$
	$(4,3)_3$	

Voisins de plus gros grain Dans le cas où le nœud adjacent n'est pas de décomposition aussi fine que le nœud dont nous cherchons les voisins, l'ordre de Morton nous permet de remonter jusqu'à son premier aïeul existant (première feuille de cette branche).

Par exemple, si le nœud de coordonnées $(3,4)_3$ n'existe pas, il faut remonter à ses ascendants. Le nœud de coordonnées $(3,4)_3 \Leftrightarrow (011,100)_3$ a pour père le nœud $(01,10)_2 \Leftrightarrow (1,2)_2$ qui constitue, s'il existe, le voisin de plus gros grain du nœud $(4,4)_3$ comme le montre le tableau 5.4.

Voisins de plus petit grain Si un nœud possède des fils, ce sont eux qu'il faut considérer comme voisins, puisque les informations se propagent uniquement sur les feuilles des arbres. Pour trouver les voisins de hauteur $h+1$ d'un nœud de hauteur h, il faut tout d'abord ajouter un bit supplémentaire pour coder les coordonnées des voisins de plus bas maillage. Puis, la sélection des voisins se fait de la manière suivante :

TAB. 5.4 – Voisins de plus gros grain

$(1,2)_2$	$(4,5)_3$	
	$(4,4)_3$	$(5,4)_3$
	$(4,3)_3$	

– si les coordonnées du voisin de même niveau h sont obtenues par le retrait d'une unité en ligne (resp. en colonne), le bit supplémentaire de la ligne (resp. colonne) doit être valué à 1 et le bit supplémentaire de la colonne (resp. ligne) à 0 ou 1.
– si les coordonnées du voisin de même niveau h sont obtenues par l'ajout d'une unité en ligne (resp. en colonne), le bit supplémentaire de la ligne (resp. colonne) doit être valué à 0 et le bit supplémentaire de la colonne (resp. ligne) à 0 ou 1.

S'il est nécessaire de descendre d'un ou plusieurs niveaux supplémentaires, il faut réitérer le processus à chaque niveau.

Si le nœud $(4,3)_3$ possède des fils, il faut identifier ceux qui sont voisins du nœud $(4,4)_3$. Les fils de $(4,3)_3 \Leftrightarrow (100,011)_3$ voisins de $(4,4)_3$ sont : $(100\mathbf{0},\ 011\mathbf{1})_4 \Leftrightarrow (8,7)_4$ et $(100\mathbf{1},\ 011\mathbf{1})_4 \Leftrightarrow (9,7)_4$ comme l'illustre le tableau 5.5. Le nœud $(4,4)_3$ possède ainsi cinq voisins de grain différent.

TAB. 5.5 – Voisins de plus petit grain

$(1,2)_2$	$(4,5)_3$		
	$(4,4)_3$		$(5,4)_3$
	$(8,7)_4$	$(9,7)_4$	

5.2.2.7 Algorithme de génération des inégalités

La méthode de génération d'arbres quaternaires à partir de la définition symbolique de contraintes d'inégalité définies par morceaux $\boldsymbol{C}(x,y)$ repose sur une première étape d'identification et de marquage des degrés d'information pertinente des nœuds, puis sur une seconde étape de propagation d'information des nœuds qui connaissent leur cohérence avec la contrainte $\boldsymbol{C}(x,y)$ vers ceux qui ne peuvent la déterminer par manque d'information pertinente.

La fonction QT INÉGAL, présentée par l'algorithme 18, construit l'arbre quaternaire $T_{(x,y)}$ associé à une contrainte d'inégalité définie par morceaux $\boldsymbol{C}(x,y)$ sur $(D_x^{\boldsymbol{C}}, D_y^{\boldsymbol{C}}) = \bigcup_{i=1}^{n} \{\boldsymbol{c_i}(x,y)$ sur $(D_x^{c_i}, D_y^{c_i})\}$. Cette fonction fait appel à plusieurs fonctions ou procédures :

– GÉNÉRER INÉGAL, présentée par l'algorithme 19 page 98, qui partitionne l'espace de recherche d'un nœud suivant les différents degrés d'information pertinente,
– FRONTIÈRE, présentée par l'algorithme 14 page 91, qui permet d'informer sur leur cohérence les feuilles *vides* et *sous-informées* voisines de feuilles *frontières unitaires*,

Alg. 18 QT INÉGAL(CONTRAINTE : $\boldsymbol{C}_{(x,y)}$, INTERVALLE : $D_x^{\boldsymbol{C}}$, INTERVALLE : $D_y^{\boldsymbol{C}}$)

– ∴ – ***Cet algorithme génère un arbre quaternaire*** $T_{(x,y)}$ ***à partir d'une contrainte d'inégalité définie par morceaux*** $\boldsymbol{C}_{(x,y)}$ ***et d'un espace de recherche*** $D_x^{\boldsymbol{C}}$, $D_y^{\boldsymbol{C}}$.
– ∴ – $D_x^{\boldsymbol{C}}$ *correspond à l'intervalle de solution suivant* x
– ∴ – $D_y^{\boldsymbol{C}}$ *correspond à l'intervalle de solution suivant* y
Création du nœud racine r de $T_{(x,y)}$ portant sur $(D_x^{\boldsymbol{C}}, D_y^{\boldsymbol{C}})$
– ∴ – *La contrainte associée au nœud racine* r *est* $\boldsymbol{C}_{(x,y)}$
$r \leftarrow$ GÉNÉRER INÉGAL(r) (algorithme 19 page suivante)
– ∴ – $T_{(x,y)}$ *est coloré des couleurs particulières (rouge, vert, jaune, orange) et terminales (blanc, bleu).*
– ∴ – *Il faut maintenant propager les informations des feuilles informatrices vers les autres*
– ∴ – *Les feuilles* frontières *jaunes informent de leur cohérence leurs feuilles voisines vertes et rouges*
FRONTIÈRE($T_{(x,y)}$) (algorithme 14 page 91)
– ∴ – *Les feuilles incohérentes informent leurs feuilles voisines vertes et rouges de leur incohérence*
BLEUIR($T_{(x,y)}$) (algorithme 15 page 92)
– ∴ – *Les feuilles cohérentes informent leurs feuilles voisines vertes et rouges de leur cohérence*
BLANCHIR($T_{(x,y)}$) (algorithme 16 page 93)
– ∴ – *Les feuilles* frontières unitaires *jaunes et* sur-frontières unitaires *oranges se colorent en blanc*
CONTOUR($T_{(x,y)}$) (algorithme 17 page 94)
Retourner $T_{(x,y)}$

- BLEUIR, présentée par l'algorithme 15 page 92, qui propage par voisinage les zones incohérentes sur les feuilles *vides* et *sous-informées*,
- BLANCHIR, présentée par l'algorithme 16 page 93, qui propage par voisinage les zones cohérentes sur les feuilles *vides* et *sous-informées*,
- CONTOUR, présentée par l'algorithme 17 page 94, qui colore les feuilles unitaires *frontières* et *sur-frontières* en blanc.

Les zones cohérentes et incohérentes avec la contrainte $\boldsymbol{C}(x,y)$ sur $(D_x^{\boldsymbol{C}}, D_y^{\boldsymbol{C}}) = \bigcup_{i=1}^{n} \{\boldsymbol{c_i}(x,y)$ sur $(D_x^{c_i}, D_y^{c_i})\}$ sont ainsi identifiées. Il faut alors reconstruire les domaines des variables x et y participant à la contrainte définie par morceaux et les propager aux autres contraintes $\boldsymbol{C}(x,j)$ et $\boldsymbol{C}(y,j)$.

5.2.3 Fusion et filtrage des arbres quaternaires par morceaux

Les méthodes de *fusion* et de filtrage développées par Sam (1995) et présentées respectivement dans les sous-sections 5.1.4 et 5.1.5 s'appliquent de manière identique aux arbres quaternaires générés à partir de contraintes numériques binaires définies par morceaux. Ces méthodes sont uniquement basées sur les couleurs terminales des nœuds (blanc et bleu) et non sur les définitions symboliques des contraintes.

5.2.3.1 Fusion

La *fusion* d'arbres permet de prendre en compte simultanément des contraintes portant sur la même paire de variables. C'est la comparaison des couleurs de chacun des nœuds des arbres

Alg. 19 GÉNÉRER INÉGAL(NOEUD : n)

– $\therefore$ – ***Cet algorithme construit l'arbre quaternaire*** $T_{(x,y)}$ ***d'une contrainte d'inégalité définie par morceaux*** $C_{(x,y)}$ ***à partir d'un nœud.***
– $\therefore$ – *ϵ_x correspond au degré de précision de la variable x*
– $\therefore$ – *ϵ_y correspond au degré de précision de la variable y*
– $\therefore$ – *$d_n^x = [\overline{x}, \underline{x}]$ et $d_n^y = [\overline{y}, \underline{y}]$ correspondent au sous-espace du nœud courant n.*
– $\therefore$ – *La précision n'est pas atteinte en x ou en y*
Tant Que $(\overline{x} - \underline{x} > \epsilon_x \vee \overline{y} - \underline{y} > \epsilon_y)$ **Faire**
  Création et étiquetage des quatre fils du nœud courant n
  Pour tous les fils f du nœud courant n **Faire**
    Contraintes $\leftarrow$ liste des contraintes $\boldsymbol{c_i}$ définies sur le nœud f : $d_f^x \cap D_x^{c_i} \neq \emptyset \wedge d_f^y \cap D_y^{c_i} \neq \emptyset$
    Si $(Contraintes = 0)$ **Alors**
      – $\therefore$ – *Le nœud f est un nœud* vide
      – $\therefore$ – *Le nœud f devient une feuille rouge*
      $couleur(f) \leftarrow$ rouge
    Sinon
      – $\therefore$ – *Au moins une contrainte $\boldsymbol{c_i}(x, y)$ est définie sur le nœud f.*
      Si (le nœud f est *frontière*) **Alors**
        – $\therefore$ – *Génération de l'arbre quaternaire correspondant à la contrainte isolée $\boldsymbol{c_i}(x, y)$ sur (d_f^x, d_f^y)*
        $f \leftarrow$ GÉNÉRER ARBRE QUATERNAIRE(f) (algorithme 8 page 64)
      Sinon
        Si (le nœud f est *sur-frontière*) **Alors**
          – $\therefore$ – *Le nœud f doit être décomposé*
          $f \leftarrow$ GÉNÉRER INÉGAL(f) (algorithme 19)
        Sinon
          – $\therefore$ – *Le nœud f est* sous-informé
          – $\therefore$ – *Le nœud f devient une feuille verte*
          $couleur(f) \leftarrow$ vert
        Fin Si
      Fin Si
    Fin Si
  Fin Pour
Fin Tant Que
– $\therefore$ – *Le nœud n est unitaire*
Si (le nœud n est *frontière unitaire* d'une seule contrainte $\boldsymbol{c_i}(x, y)$) **Alors**
  $couleur(n) \leftarrow$ jaune
Sinon
  Si (le nœud n est *frontière unitaire* de plusieurs contraintes $\boldsymbol{c_i}(x, y)$) **Alors**
    $couleur(n) \leftarrow$ orange
  Sinon
    – $\therefore$ – *Dans tous les autres cas, le nœud ne peut déduire sa couleur*
    $couleur(n) \leftarrow$ vert
  Fin Si
Fin Si
Retourner n

à fusionner, selon l'ordre établi ($blanc < gris < bleu$), qui détermine la couleur du nœud de l'arbre résultat. La *fusion* ne peut être effectuée que sur des arbres ayant les mêmes domaines de définition et possédant, par conséquent, le même découpage de l'espace de solution.

La figure 5.26 présente la *fusion* de deux contraintes portant sur la même paire de variables. La première est définie par $\boldsymbol{C_1} : y > \frac{-6}{7} \times x + 3$, la seconde est une contrainte définie par morceaux nommée *Pentagone* :

$$\boldsymbol{Pentagone}(x,y) \left\| \begin{array}{l} D_x^{\boldsymbol{P}} = [0,6] \\ D_y^{\boldsymbol{P}} = [0,6] \end{array} \right. = \bigcup \left\{ \begin{array}{ll} \boldsymbol{f_1}(x,y): & y \leq -0.8 \times x + 8 \\ \boldsymbol{f_2}(x,y): & y \geq \frac{8}{3} \times x - \frac{28}{3} \\ \boldsymbol{f_3}(x,y): & y \leq \frac{6}{5} \times x + 3 \\ \boldsymbol{f_4}(x,y): & y \geq 0 \\ \boldsymbol{f_5}(y,x): & x \geq 0 \end{array} \right. \left\| \begin{array}{l} D_x^{f_1} = [2.5,5] \\ D_y^{f_1} = [4,6] \\ \hline D_x^{f_2} = [3.5,5] \\ D_y^{f_2} = [0,4] \\ \hline D_x^{f_3} = [0,2.5] \\ D_y^{f_3} = [3,6] \\ \hline D_x^{f_4} = [0,3.5] \\ D_y^{f_4} = 0 \\ \hline D_x^{f_5} = 0 \\ D_y^{f_5} = [0,3] \end{array} \right.$$

Toutes deux portent sur les intervalles $D_x = [0,6]$ et $D_y = [0,6]$ et ont une précision de $\epsilon_x = 0.0625$ et $\epsilon_y = 0.0625$.

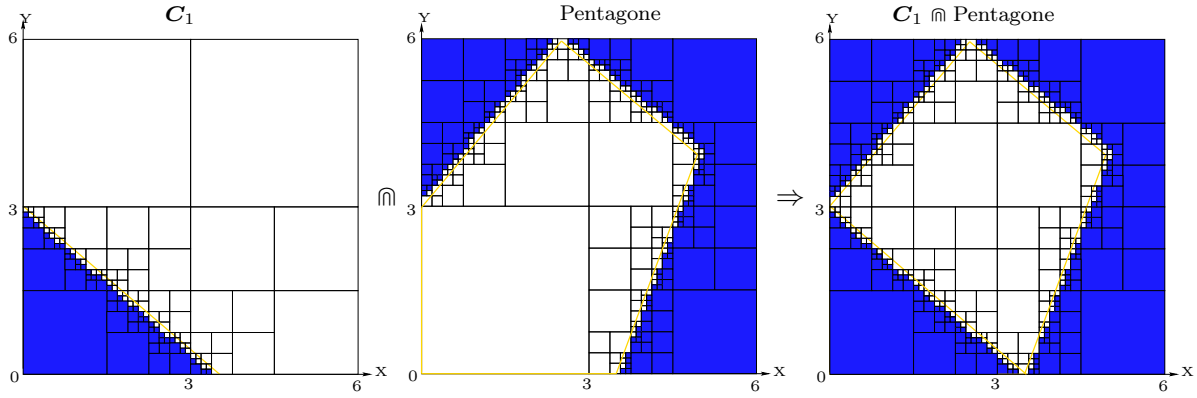


FIG. 5.26 – Fusion de contraintes

5.2.3.2 Filtrage

L'objectif principal des méthodes de filtrage est la détection et le retrait des domaines des variables de valeurs qui ne peuvent mener à une solution. La représentation des contraintes continues numériques binaires sous forme arborescente découpe l'espace de recherche en régions compatibles blanches et incompatibles bleues.

Lorsque les arbres quaternaires associés aux contraintes définies par morceaux $\boldsymbol{C}(x,y)$ sur $(D_x^{\boldsymbol{C}}, D_y^{\boldsymbol{C}}) = \bigcup_{i=1}^{n} \{\boldsymbol{c_i}(x,y) \text{ sur } (D_x^{c_i}, D_y^{c_i})\}$ sont générés et colorés des couleurs terminales bleues et blanches, les zones cohérentes et incohérentes avec les contraintes $\boldsymbol{C}(x,y)$ sont

identifiées. Il faut alors reconstruire les domaines des variables participant aux contraintes définies par morceaux. Nous pouvons faire appel à la fonction DOMAINE COMPATIBLE, algorithme 9 page 69, qui reconstitue, à partir de la couleur des feuilles d'arbre quaternaire, les domaines cohérents des variables.

Lorsque le domaine D_x d'une variable x appartenant à plusieurs arbres quaternaires $T_{(x,j)}$ est réduit, il faut propager cette information aux contraintes continues numériques binaires définies ou non par morceaux $\boldsymbol{C}(x, j)$ représentées par les arbres $T_{(x,j)}$. Il faut par conséquent élaguer les branches qui ne sont plus compatibles avec le problème courant et décomposer celles qui pourraient ne plus l'être totalement. La détection des branches incompatibles se fait de manière identique à celle des arbres associés aux contraintes continues par la détermination de l'intersection du domaine de la variable réduite D_x avec le domaine d_n^x du nœud traité n pour la variable x de l'arbre $T_{(x,j)}$. La décomposition des branches pouvant contenir d'éventuelles zones incohérentes est réalisée à partir de la contrainte $x = D_x$.

La procédure PROPAGER RÉDUCTION DOMAINE, présentée par l'algorithme 10 page 70, s'applique elle aussi aux arbres quaternaires représentant des contraintes définies par morceaux. Cette procédure élague un arbre $T_{(x,y)}$ et explore ses feuilles blanches pour éliminer les zones incohérentes avec le domaine réduit de la variable x.

La procédure τ-AC-3-QUATERNAIRE, présentée par l'algorithme 11 page 71, s'applique elle aussi aux arbres quaternaires représentant des contraintes numériques définies par morceaux. Cette procédure réduit les domaines des variables à partir de structures arborescentes blanches et bleues représentant des contraintes définies ou non par morceaux, et propage ces réductions de domaines aux autres arbres quaternaires portant sur au moins l'une de ces variables.

5.2.4 Synthèse

Dans cette section, nous avons présenté deux méthodes de génération d'arbres quaternaires pour des contraintes d'égalité et d'inégalité définies par morceaux $\boldsymbol{C}(x, y)$ sur $(D_x^{\boldsymbol{C}}, D_y^{\boldsymbol{C}}) = \bigcup_{i=1}^{n} \{\boldsymbol{c_i}(x, y) \text{ sur } (D_x^{c_i}, D_y^{c_i})\}$. De manière identique à la méthode développée par Sam (1995), ces méthodes de génération partitionnent de manière récursive l'espace de recherche initial en rectangles, jusqu'à l'atteinte d'une précision prédéfinie. Comme seules les zones (D_x^{ci}, D_y^{ci}) couvertes par les contraintes $\boldsymbol{c_i}$ possèdent de l'information et délimitent les zones cohérentes et incohérentes, cette méthode identifie les degrés d'information pertinente des nœuds (associés aux rectangles) puis propage, par voisinage, les zones cohérentes et incohérentes.

Un certain nombre d'hypothèses doivent être posées pour générer les arbres quaternaires sur les contraintes définies par morceaux afin de garantir la connexité des zones cohérentes et incohérentes.

Nous définissons différents degrés d'information pertinente correspondant à la propension des nœuds à déterminer leur cohérence : les nœuds *vides*, les nœuds *sous-informés*, les nœuds *frontières* et les nœuds *sur-frontières*.

Dans le cas d'égalité par morceaux, seuls les nœuds *frontières unitaires* et les nœuds *sur-frontières unitaires*, définissant l'enveloppe de la contrainte, sont cohérents. Ce sont donc les

seuls à se colorer en blanc. Tous les autres sont incohérents avec la contrainte et se colorent, par conséquent, en bleu.

Dans le cas d'inégalité par morceaux, la méthode de génération nécessite une phase de propagation d'informations des feuilles *frontières unitaires*, des feuilles incohérentes et des feuilles cohérentes vers les feuilles *vides* et les feuilles *sous-informées*. Les degrés d'information pertinente sont donc matérialisés par des couleurs particulières :

- rouge pour les feuilles *vides*,
- vert pour les feuilles *sous-informées*,
- jaune pour les feuilles *frontières unitaires* et
- orange pour les feuilles *sur-frontières unitaires*.

Différentes campagnes d'informations sont lancées pour propager les zones cohérentes et incohérentes. Les feuilles *frontières unitaires* jaunes commencent par informer leurs feuilles voisines vertes et rouges de leur cohérence en fonction du côté de la frontière auquel elles appartiennent. Puis, les feuilles bleues et blanches indiquent à leurs feuilles voisines vertes et rouges qu'elles sont respectivement incohérentes et cohérentes avec la contrainte définie par morceaux. Enfin, les feuilles unitaires jaunes et oranges deviennent cohérentes avec la contrainte et se colorent en blanc, si seul ce qui est totalement incohérent avec la contrainte est à exclure. Les zones cohérentes et incohérentes avec la contrainte continue numérique binaire définie par morceaux sont ainsi délimitées.

La propagation des zones cohérentes et incohérentes est réalisée par voisinage sur les feuilles des arbres. Les feuilles voisines d'un nœud sont identifiées à partir de son étiquette. Les feuilles voisines d'un nœud peuvent être d'un grain différent du sien.

La *fusion* et le filtrage de ce type de contraintes sont identiques à ceux des contraintes continues, puisque ces mécanismes sont uniquement basés sur les couleurs terminales blanches et bleues des nœuds.

5.3 Amélioration de la génération par une *fusion* opportuniste

La prise en compte simultanée de contraintes est réalisée par le mécanisme de *fusion*. Celui-ci construit à partir de chacun des arbres représentant des contraintes une contrainte équivalente à leur conjonction. Pour éviter de générer autant d'arbres que de contraintes portant sur la même paire de variables, nous proposons le mécanisme de *fusion opportuniste*.

Un seul arbre est généré par paire de variables : les arbres ne sont plus rattachés aux contraintes, mais à des paires de variables. Lorsque plusieurs contraintes portent sur la même paire de variables, un premier arbre est généré sur la totalité de l'espace de solution pour une seule contrainte. Puis, au lieu de générer tous les arbres, les contraintes sont discrétisées sur les espaces cohérents de l'arbre (les feuilles blanches) représentant leur conjonction ou arbre de conjonction. En effet, la couleur d'un nœud n commun à k contraintes est déterminée par la comparaison de ses k couleurs :

$$couleur(n) = Max(couleur(n)_{\boldsymbol{C_1}(x,y)}, couleur(n)_{\boldsymbol{C_2}(x,y)}, \ldots, couleur(n)_{\boldsymbol{C_k}(x,y)})$$

Lorsqu'un nœud est détecté incohérent pour l'une des contraintes, son nœud équivalent dans l'arbre de conjonction l'est aussi. Il est donc inutile de tester la cohérence des contraintes, restant à prendre en compte sur les nœuds incohérents.

Si un nœud cohérent dans l'arbre de conjonction de n contraintes se révèle être incohérent pour la $n+1^{\text{ième}}$ contrainte, le nœud devient incohérent et se colore en bleu. Sinon, il reste cohérent avec la conjonction des $n+1$ contraintes.

La procédure FUS GEN, présentée par l'algorithme 20, se base sur les feuilles cohérentes d'un arbre, pour générer l'arbre équivalent à la conjonction de deux contraintes portant sur la même paire de variables.

Les zones cohérentes et incohérentes avec l'ensemble des contraintes sont ainsi identifiées. Il faut alors reconstruire les domaines des variables rattachées à l'arbre de conjonction et les propager aux autres contraintes.

Considérons l'exemple tiré de Rueher (2002) où : $\{\boldsymbol{C_1} : x + y = 2, \boldsymbol{C_2} : y \leq x + 1, \boldsymbol{C_3} : y \geq 1 + \ln(x)\}$ sur les domaines $D_x = [0, 100], D_y = [0, 100]$. L'arbre de conjonction va être totalement généré sur $D_x = [0, 100], D_y = [0, 100]$ pour la contrainte $\boldsymbol{C_1}$. Puis, la cohérence des deux autres contraintes sera évaluée uniquement sur les feuilles blanches de cet arbre de conjonction. En considérant plusieurs degrés de précision et en comparant la *fusion* classique à la *fusion* opportuniste, nous obtenons les résultats présentés dans le tableau 5.6. Nous prenons comme temps de référence celui associé à $\epsilon_x = \epsilon_y = 10$ pour une *fusion* classique.

TAB. 5.6 – Comparaison *fusion* et *fusion* opportuniste

$\epsilon_x = \epsilon_y$	Temps génération puis *fusion*	Temps *fusion* opportuniste
10	1	0.056
5	3.75	0.069
1	55.72	0.156
0.5	220	0.343
0.1	3551.74	1.1

L'ordre de prise en compte des contraintes influe sur la vitesse de génération de l'arbre de conjonction dans le cas de la *fusion* opportuniste. Plus la contrainte initiale a de zones incohérentes, plus la vitesse de génération de l'arbre de conjonction est rapide. Le tableau 5.7 présente les résultats obtenus suivant l'ordre de prise en compte des contraintes. Dans le premier cas, l'arbre quaternaire est totalement généré sur la contrainte d'égalité $\boldsymbol{C_1}$, puis la *fusion* opportuniste est réalisée sur $\boldsymbol{C_2}$ puis sur $\boldsymbol{C_3}$. Dans le deuxième cas, l'arbre quaternaire est totalement généré sur la contrainte d'inégalité $\boldsymbol{C_2}$, puis la *fusion* opportuniste est réalisée sur $\boldsymbol{C_1}$ puis sur $\boldsymbol{C_3}$. Dans le troisième cas, l'arbre quaternaire est totalement généré sur la contrainte d'inégalité $\boldsymbol{C_2}$, puis la *fusion* opportuniste est réalisée sur $\boldsymbol{C_3}$ puis sur $\boldsymbol{C_1}$. Nous prenons comme temps de référence celui associé à $\epsilon_x = \epsilon_y = 10$ pour une *fusion* classique.

Il apparaît clair que le cas le plus favorable à la *fusion opportuniste* est celui où la contrainte d'égalité $\boldsymbol{C_1}$ est discrétisée en premier. Le cas le moins favorable est celui où la contrainte d'égalité est considérée en dernier. Mais ce cas est tout de même plus rapide qu'une *fusion* classique.

Alg. 20 FUS GEN(ARBRE : $T_{(x,y)}$,CONTRAINTE : $\boldsymbol{C_2}(x,y)$)

– ∴ – ***Cet algorithme construit l'arbre quaternaire de conjonction $T_{(x,y)}$ de contraintes à partir d'un arbre existant et d'un nouvelle contrainte $\boldsymbol{C_2}(x,y)$ à prendre en considération.***
– ∴ – *Noeuds = liste des nœuds à explorer*
Noeuds ← racine de $T_{(x,y)}$
Tant Que (*Noeuds* $\neq \emptyset$) **Faire**
  – ∴ – *Si n est une feuille cohérente, il faut réaliser la fusion de $T_{(x,y)}$ et de $\boldsymbol{C_2}(x,y)$ sur n*
  Si (la couleur de n est blanche) **Alors**
    – ∴ – *La contrainte $\boldsymbol{C_2}(x,y)$ n'est pas définie par morceaux*
    Si ($\boldsymbol{C_2}(x,y)$ est une contrainte non définie par morceaux) **Alors**
      – ∴ – *Il faut tester sa cohérence sur l'espace défini par le nœud n : son arbre quaternaire est généré sur d_n^x, d_n^y*
      $n \leftarrow$ GÉNÉRER ARBRE QUATERNAIRE(n) (algorithme 8 page 64)
    Sinon
      Si ($\boldsymbol{C_2}(x,y)$ est une contrainte d'égalité par morceaux) **Alors**
        – ∴ – *Il faut tester sa cohérence sur l'espace défini par le nœud n : son arbre quaternaire est généré sur d_n^x, d_n^y*
        $n \leftarrow$ GÉNÉRER ÉGAL(n) (algorithme 13 page 86)
      Sinon
        – ∴ – *La contrainte $\boldsymbol{C_2}(x,y)$ est une contrainte d'inégalité par morceaux.*
        – ∴ – *Il faut tester sa cohérence sur l'espace défini par le nœud n : son arbre quaternaire est généré sur d_n^x, d_n^y*
        $n \leftarrow$ GÉNÉRER INÉGAL(n) (algorithme 19 page 98)
      Fin Si
    Fin Si
  Sinon
    Si (la couleur de n est grise) **Alors**
      – ∴ – *n est un nœud dont il faut explorer les fils*
      Empiler dans *Noeuds* les quatre fils de n
    Fin Si
  Fin Si
Fin Tant Que
Si (la contrainte $\boldsymbol{C_2}(x,y)$ est une contrainte d'inégalité par morceaux) **Alors**
  – ∴ – *Il faut propager les zones cohérentes et incohérentes*
  FRONTIÈRE($T_{(x,y)}$) (algorithme 14 page 91)
  BLEUIR($T_{(x,y)}$) (algorithme 15 page 92)
  BLANCHIR($T_{(x,y)}$) (algorithme 16 page 93)
  CONTOUR($T_{(x,y)}$) (algorithme 17 page 94)
Fin Si
Retourner $T_{(x,y)}$

TAB. 5.7 – Influence de l'ordre des contraintes dans la *fusion* oppportuniste

$\epsilon_x = \epsilon_y$	*Fusion* classique	$\boldsymbol{C_1, C_2, C_3}$	$\boldsymbol{C_2, C_1, C_3}$	$\boldsymbol{C_2, C_3, C_1}$
10	1	0.056	0.45	0.79
5	3.75	0.069	1.46	2.91
1	55.72	0.156	19.64	38.79
0.5	220	0.343	75.38	150.18
0.1	3551.74	1.1	1174.1	2370.63

5.4 Synthèse

Les arbres quaternaires permettent de représenter et d'intégrer dans les *CSPs* des contraintes numériques continues qui ne possèdent pas forcément les propriétés de monotonie et de projetabilité indispensables pour un filtrage par 2B-cohérence.

Leur génération à partir de la définition symbolique des contraintes binaires $\boldsymbol{C}(x, y)$ est réalisée en découpant récursivement l'espace de solution en rectangles, définissant les nœuds de l'arbre, jusqu'à l'atteinte d'une précision prédéfinie. Chaque nœud est coloré suivant sa cohérence avec la contrainte :

- bleu s'il est totalement incohérent avec $\boldsymbol{C}(x, y)$ et devient feuille,
- blanc s'il est totalement cohérent avec $\boldsymbol{C}(x, y)$ et devient feuille,
- gris s'il est composé de zones cohérentes et incohérentes : il doit être à nouveau décomposé.

La méthode de génération des arbres quaternaires pour des contraintes continues n'a pas été conçue pour prendre en compte des contraintes numériques continues définies par morceaux $\boldsymbol{C}(x, y)$ sur $(D_x^{\boldsymbol{C}}, D_y^{\boldsymbol{C}}) = \bigcup_{i=1}^{n} \{\boldsymbol{c_i}(x, y)$. Nous proposons deux méthodes de génération d'arbres quaternaires sur ce type de contraintes, basées sur celle de Sam (1995). Ces méthodes de génération partitionnent de manière récursive l'espace de recherche initial jusqu'à l'atteinte d'une précision prédéfinie. Comme seules les zones (D_x^{ci}, D_y^{ci}) couvertes par les contraintes $\boldsymbol{c_i}(x, y)$ possèdent de l'information et délimitent les zones cohérentes et incohérentes, il faut identifier le degré d'information pertinente de chaque nœud, puis propager par voisinage les zones cohérentes et incohérentes.

Nous avons défini différents degrés d'information pertinente des nœuds, matérialisés par des couleurs spécifiques :

- rouge pour les nœuds *vides* d'information,
- vert pour les nœuds *sous-informés*,
- jaune pour les nœuds *frontières unitaires* et
- orange pour les nœuds *sur-frontières unitaires*.

Dans le cas d'égalités par morceaux, seules les feuilles *frontières unitaires* et les feuilles *sur-frontières unitaires*, définissant l'enveloppe de la contrainte, sont cohérentes. Ce sont donc

les seules à se colorer en blanc. Toutes les autres feuilles sont incohérentes avec la contrainte d'égalité définie par morceaux et se colorent en bleu.

Dans le cas d'inégalité par morceaux, l'arbre quaternaire est coloré de ces couleurs spécifiques rouges, verts, jaunes et oranges et des couleurs terminales blanches et bleues. Des campagnes d'informations doivent être lancées pour propager les zones cohérentes blanches et incohérentes bleues. Cette propagation exploite pleinement l'étiquetage des nœuds à partir de la courbe de Péano ordonnée par Morton.

La prise en compte de plusieurs contraintes (définies ou non par morceaux) portant sur la même paire de variables et sur le même espace de définition est réalisée à partir de la comparaison des couleurs des nœuds, suivant l'ordre établi $blanc < gris < bleu$.

Pour éviter de générer autant d'arbres quaternaires que de contraintes, nous proposons le mécanisme de *fusion opportuniste* qui ne teste la cohérence des contraintes que sur les feuilles cohérentes de l'arbre de conjonction. L'ordre de prise en compte des contraintes joue un rôle important dans la rapidité de génération des arbres de conjonction, mais la *fusion oppportuniste* reste tout de même plus rapide.

Le filtrage des contraintes (définies ou non par morceaux) mises sous forme arborescente consiste à élaguer les branches devenues entièrement incompatibles avec le problème courant et à décomposer celles qui pourraient ne plus l'être totalement.

La représentation de contraintes sous forme d'arbres quaternaires possède un certain nombre de limites dues :

- aux degrés de précision qui approchent les résultats de filtrage ;
- à l'utilisation de l'arithmétique des intervalles qui peut engendrer des explorations inutiles.

Chapitre 6

Contraintes d'activation et extension

Les contraintes d'activation permettent d'ajuster un modèle de connaissances à un problème de conception particulier par la prise en compte exclusive des éléments indispensables à sa résolution. Elles modifient la structure du modèle par l'ajout et le retrait d'éléments (variables et contraintes) suivant les différents choix de conception. Les contraintes d'activation se définissent en deux parties : une prémisse (condition logique) et un conséquent (ajout ou retrait d'un ensemble d'éléments) liées par un opérateur d'implication $\rightarrow$. Lorsque la prémisse est vérifiée, il y a application du conséquent : la structure du modèle est enrichie ou appauvrie en variables et contraintes. Sinon, sa structure reste inchangée.

Les premières notions de contraintes d'activation furent introduites dans les *CSPs* dynamiques par Mittal et Falkenhainer (1990). À l'origine, ces contraintes étaient purement discrètes et ne permettaient de modifier que les modèles de connaissance sans structure hiérarchique. Il est vite apparu que la modélisation de problèmes réels « à plat » était délicate et que les prémisses discrètes étaient insuffisantes pour modéliser la diversité des conditions d'activation. Pour modéliser la structure hiérarchique des problèmes, le concept de *CSP* composite fut proposé par Sabin et Freuder (1996). Les variables des *CSPs* composites peuvent être substituées par des sous-problèmes (sous-*CSP*). Pour éviter de perdre une vue globale du modèle, les *CSPs* à états furent proposés par Veron (2001). Les *CSPs* à états recouvrent les problématiques des *CSPs* dynamiques et composites, à savoir l'activation de variables et la modélisation structurée.

L'expressivité des prémisses et des conséquents des contraintes d'activation fut étendue par la prise en compte d'autres conditions logiques portant sur des variables numériques (Gelle et Weigel, 1995), sur leur type *objet* (Gelle, 1998) ainsi que sur l'activation de sous-ensembles de variables des conséquents (Soininen et Gelle, 1999).

Nous allons, dans la première section, présenter les différentes formes que peuvent prendre les contraintes d'activation. Puis, nous recenserons les besoins rencontrés lors de la formalisation des connaissances, ainsi que les choix d'implémentation des contraintes d'activation que nous avons effectués pour y répondre.

6.1 État de l'art

Les contraintes d'activation permettent de considérer uniquement les éléments pertinents d'un modèle de connaissance. Elles modifient sa structure en ajoutant (resp. en retirant) les variables et les contraintes indispensables (resp. inutiles) à sa résolution. La notion de contraintes d'activation a évolué au fil des besoins, passant de modèles de connaissances « à plat » à des structures hiérarchiques et de prémisses discrètes à des conditions d'activation plus complexes. Nous présentons, dans cette section, les différentes formes des contraintes d'activation.

6.1.1 Variables à existence conditionnée

Deux types de *CSPs* permettent d'adapter la structure d'un modèle à un problème donné par l'ajout de variables : les *CSPs* dynamiques de Mittal et Falkenhainer (1990) et les *CSPs* à états de Veron (2001). Ces deux types sont basés sur une définition explicite des contraintes d'activation. Nous les présentons dans cette section.

6.1.1.1 CSPs dynamiques

Les premières contraintes d'activation sont apparues dans les *CSPs* dynamiques de Mittal et Falkenhainer (1990), renommés *CSPs* conditionnels par Sabin et Freuder (1999) pour ne pas les confondre avec les *CSPs* dynamiques de Dechter et Dechter (1988). Ces derniers incluent dans l'ensemble des contraintes de compatibilité les contraintes unaires de restriction de domaines des variables. Ce type de *CSP* est alors défini comme une séquence de *CSPs* où chaque élément *CSP* diffère du précédent par l'ajout ou le retrait de contraintes. Contrairement aux *CSPs* conditionnels, il n'existe pas dans les *CSPs* dynamiques de Dechter et Dechter (1988) de contraintes d'activation à proprement parler : l'ensemble des modifications possibles d'un *CSP* (ajout ou retrait de contraintes, ajout ou retrait de variables) s'exprime en termes d'ajout et retrait de contraintes unaires (Verfaillie et Schiex, 1995). Les contraintes d'activation de Mittal et Falkenhainer (1990) permettent, par contre, explicitement d'ajouter et de retirer d'un problème des ensembles de variables.

Définition 23 : CSP *conditionnels*

Les CSPs *conditionnels sont définis comme un triplet* $(\mathbb{V}, \mathbb{D}, \mathbb{C})$ *où :*

- $\mathbb{V} = \{v_1, v_2, ..., v_l\}$ *est un ensemble fini de variables composé de deux sous-ensembles disjoints :*
  - *un ensemble* $\mathbb{V}_a$ *non vide de variables initialement actives,*
  - *un ensemble* $\mathbb{V}_i$ *de variables initialement inactives.*
- $\mathbb{D} = \{d_1, d_2, ..., d_l\}$ *est un ensemble fini de domaines de définition des variables de* $\mathbb{V}$*,*
- $\mathbb{C}$ *est un ensemble fini de contraintes de deux natures possibles :*
  - *un ensemble* $\mathbb{C}_C = \{c_1^c, c_2^c, \ldots, c_n^c\}$ *de contraintes de compatibilité,*
  - *un ensemble* $\mathbb{C}_A = \{c_1^a, c_2^a, \ldots, c_k^a\}$ *de contraintes d'activation.*

Les contraintes d'activation sont composées d'une prémisse $P(\mathbb{V}_p)$ et d'un conséquent $\mathbb{V}_c$. L'ensemble des variables $\mathbb{V}_p$ de la prémisse et l'ensemble des variables $\mathbb{V}_c$ du conséquent doivent être disjoints. Les contraintes d'activation de variables sont de la forme :

$$P(\mathbb{V}_p) \rightarrow (\text{dés})\text{activation} : \mathbb{V}_c$$

Chaque variable possède un état *actif* ou *inactif* qui indique si celle-ci doit être ou non prise en compte dans le problème courant. Les contraintes de compatibilité prennent part au filtrage lorsque toutes les variables sur lesquelles elles portent sont actives. Sinon, elles sont considérées comme trivialement satisfaites.

Une solution d'un *CSP* conditionnel correspond à un ensemble de variables valuées, contenant obligatoirement l'ensemble $\mathbb{V}_a$ des variables initialement actives, respectant toutes les contraintes de compatibilité $\mathbb{C}_C$ du modèle.

Définition 24 : *solution d'un* CSP *conditionnel*

Une solution d'un CSP *conditionnel se caractérise par le fait que :*

- *une instanciation A d'un ensemble de variables respecte toutes les contraintes de $\mathbb{C}_C$,*
- *toutes les variables initialement actives $\mathbb{V}_a$ sont valuées dans A,*
- *toutes les variables activées de $\mathbb{V}_i$ sont valuées dans A.*

Les prémisses des contraintes d'activation peuvent porter sur les valeurs de variables ou sur les états des variables. Les contraintes *RV*, pour *Required Variables* et *RN*, pour *Required Not* portent sur les valeurs des variables :

RV Les contraintes *RV* indiquent qu'un ensemble de variables $\mathbb{V}_c = \{v_k, \ldots, v_l\}$ doit être ajouté au problème si la combinaison de valeurs des variables $\mathbb{V}_p = \{v_1, v_2, \ldots, v_j\}$ est vérifiée :

$$(v_1 = a_1 \wedge v_2 = a_2 \wedge \cdots \wedge v_j = a_j) \rightarrow \text{activation} : \{v_k, \ldots, v_l\}$$

RN Les contraintes *RN* indiquent qu'un ensemble de variables $\mathbb{V}_c = \{v_k, \ldots, v_l\}$ doit être retiré du problème si la combinaison de valeurs des variables $\mathbb{V}_p = \{v_1, v_2, \ldots, v_j\}$ est vérifiée :

$$(v_1 = a_1 \wedge v_2 = a_2 \wedge \cdots \wedge v_j = a_j) \rightarrow \text{désactivation} : \{v_k, \ldots, v_l\}$$

Les contraintes *ARV*, pour *Always Required Variables*, et *ARN*, pour *Always Required Not*, ont été introduites pour éviter de lister, de manière exhaustive, la combinatoire des arrangements de valeurs autorisés pour l'activation et la désactivation de variables. Ces contraintes d'activation portent donc sur l'état des variables :

ARV Les contraintes *ARV* indiquent qu'un ensemble de variables $\mathbb{V}_c = \{v_k, \ldots, v_l\}$ doit être ajouté au problème si toutes les variables de la prémisse $\mathbb{V}_p = \{v_1, v_2, \ldots, v_j\}$ sont actives :

$$(v_1 : \mathit{active} \wedge v_2 : \mathit{active} \wedge \cdots \wedge v_j : \mathit{active}) \rightarrow \text{activation} : \{v_k, \ldots, v_l\}$$

ARN Les contraintes *ARN* indiquent qu'un ensemble de variables $\mathbb{V}_c = \{v_k, \ldots, v_l\}$ doit être retiré du problème si toutes les variables de la prémisse $\mathbb{V}_p = \{v_1, v_2, \ldots, v_j\}$ sont actives :

$$(v_1 : \mathit{active} \wedge v_2 : \mathit{active} \wedge \cdots \wedge v_j : \mathit{active}) \rightarrow \text{désactivation} : \{v_k, \ldots, v_l\}$$

6.1.1.2 CSPs à états

Les *CSPs* à états de Veron (2001) permettent d'ajouter et de retirer des variables du problème courant. Les *CSPs* à états reprennent les notions présentées dans les *CSPs* conditionnels de Mittal et Falkenhainer (1990), mais en associant à chacune des variables de base v_b du problème une variable booléenne d'état v_e. La valeur 0 (resp. 1) d'une variable d'état v_e indique que la variable de base associée v_b ne participe pas (resp. participe) à la solution. Les variables de base dont les variables d'état ne sont pas réduites à un singleton, sont dans un état indéterminé.

Définition 25 : *CSP à états*

Les CSPs *à états sont définis comme un triplet* $(\mathbb{V}, \mathbb{D}, \mathbb{C})$ *où :*

- $\mathbb{V} = \{(v_{b1}, v_{e1}), (v_{b2}, v_{e2}), ..., (v_{bl}, v_{el})\}$ *est un ensemble fini de couples variables composé :*
  - *d'un ensemble* $\mathbb{V}_b = \{v_{b1}, v_{b2}, \ldots, v_{bl}\}$ *de variables de base,*
  - *d'un ensemble* $\mathbb{V}_e = \{v_{e1}, v_{e2}, \ldots, v_{el}\}$ *de variables booléennes d'état.*
- $\mathbb{D} = \{D_b, D_e\}$ *est un ensemble fini de domaines de définition des variables de* $\mathbb{V}$ *où :*
  - D_b *est l'ensemble des domaines des variables* $\mathbb{V}_b$*,*
  - D_e *est l'ensemble des domaines booléens des variables* $\mathbb{V}_e$*.*
- $\mathbb{C}$ *est un ensemble fini de contraintes portant sur l'ensemble des variables* $\mathbb{V}$ *de deux natures possibles :*
  - *un ensemble* $\mathbb{C}_C = \{c_1^c, c_2^c, \ldots, c_n^c\}$ *de contraintes de compatibilité,*
  - *un ensemble* $\mathbb{C}_A = \{c_1^a, c_2^a, \ldots, c_k^a\}$ *de contraintes d'activation.*

La solution d'un *CSP* à états est définie comme une instanciation A des variables de $\mathbb{V}$, telle que A contienne toutes les variables d'état v_{ei} de $\mathbb{V}_e$ valuées à 0 ou à 1 et toutes les variables v_{bi} actives de $\mathbb{V}_b$, satisfaisant les contraintes de compatibilité $\mathbb{C}_C$. De manière identique aux *CSPs* conditionnels, les contraintes de compatibilité qui possèdent au moins une variable inactive sont considérées comme trivialement satisfaites.

Définition 26 : *solution d'un CSP à états*

Une solution d'un CSP *à états se caractérise par le fait que :*

- $\forall v_{ei} \in \mathbb{V}_e, v_{ei} = 0 \vee v_{ei} = 1$,
- $\forall v_{bi} \in \mathbb{V}_b \mid v_{ei} = 1$, v_{bi} *est valuée et sa valeur vérifie l'ensemble des contraintes de* $\mathbb{C}_C$*.*

Les prémisses des contraintes d'activation des *CSPs* à états reprennent celles définies dans les *CSPs* conditionnels en utilisant les variables de base et les variables d'état. Elles s'assurent, de plus, que les domaines des variables de base activées contiennent encore une valeur cohérente avec le problème courant :

RV Les contraintes *RV* se traduisent par :

$$(v_{b1} = a_1 \wedge v_{b2} = a_2 \wedge \cdots \wedge v_{bj} = a_j) \rightarrow \text{activation} : \quad \{v_{ek} = 1, \ldots, v_{el} = 1\} \text{ ssi } D_{v_{bk}} \neq \emptyset \wedge \cdots \wedge D_{v_{bl}} \neq \emptyset$$

RN Les contraintes RN se traduisent par :

$$(v_{b1} = a_1 \wedge v_{b2} = a_2 \wedge \cdots \wedge v_{bj} = a_j) \rightarrow \text{désactivation} : \quad \{v_{ek} = 0, \ldots, v_{el} = 0\}$$

ARV Les contraintes ARV se traduisent par :

$$(v_{e1} = 1 \wedge v_{e2} = 1 \wedge \cdots \wedge v_{ej} = 1) \rightarrow \text{activation} : \quad \{v_{ek} = 1, \ldots, v_{el} = 1\} \text{ ssi } D_{v_{bk}} \neq \emptyset \wedge \cdots \wedge D_{v_{bl}} \neq \emptyset$$

ARN Les contraintes ARN se traduisent par :

$$(v_{e1} = 1 \wedge v_{e2} = 1 \wedge \cdots \wedge v_{ej} = 1) \rightarrow \text{désactivation} : \quad \{v_{ek} = 0, \ldots, v_{el} = 0\}$$

Deux types de *CSPs* permettent donc d'adapter un modèle de connaissances à un problème particulier à l'aide de contraintes d'activation : les *CSPs* dynamiques (Mittal et Falkenhainer, 1990) et les *CSPs* à états (Veron, 2001). Ces deux types sont équivalents pour des problèmes « à plat » (Veron, 2001), puisqu'ils sont basés sur la notion d'activité des variables et possèdent les mêmes contraintes d'activation. Veron (2001) a traduit, dans sa thèse, l'exemple de configuration de la voiture de Mittal et Falkenhainer (1990), défini comme un *CSP* conditionnel « à plat », sous forme d'un *CSP* à états sans structure hiérarchique.

6.1.2 Composition hiérarchique

Les modèles de connaissances nécessitant une structure hiérarchique ne peuvent pas être adaptés à un problème particulier par les contraintes d'activation proposées par Mittal et Falkenhainer (1990). Or, la vision de problèmes réels de grande taille « à plat » est complexe et délicate à modéliser. Le premier formalisme permettant une représentation hiérarchique des problèmes modélisés par des contraintes fut proposé dans le cadre des *CSPs* composites de Sabin et Freuder (1996). Les valeurs des variables des *CSPs* composites peuvent être substituées par des sous-problèmes : la structure du modèle n'est plus localement modifiée (comme dans les *CSPs* conditionnels), mais globalement.

Pour éviter de perdre une vue globale du problème, les *CSPs* à états, définis par Veron (2001), permettent de représenter la structure hiérarchique de problèmes par la définition de méta-problèmes regroupant des variables et des contraintes. Contrairement aux *CSPs* composites de Sabin et Freuder (1996), les valeurs des variables du problème ne sont pas substituées par des sous-problèmes, mais le problème est enrichi en variables, contraintes et méta-problèmes.

Tout comme les variables, les méta-problèmes des *CSPs* à états possèdent des variables d'état v_e qui indiquent si ceux-ci participent ou non à la solution. Différentes règles régissent les états des méta-problèmes et des éléments (variables et méta-problèmes) qu'ils contiennent :

- un élément ne peut être actif si le méta-problème qui le contient ne l'est pas,
- si tous les éléments d'un méta-problème sont inactifs, alors le méta-problème est inactif,
- si un élément est actif, alors le méta-problème qui le contient l'est aussi,
- un méta-problème actif possède au moins un élément actif.

Veron et Aldanondo (2000) ont traduit l'exemple de configuration de la voiture de Mittal et Falkenhainer (1990), défini comme un *CSP* conditionnel « à plat », sous forme d'un *CSP* à états composé de deux méta-problèmes. Les auteurs explorent également des pistes concernant les mécanismes de filtrage sur ce type de *CSPs*.

6.1.3 Enrichissement des prémisses et des conséquents

Peu de travaux portent sur les conditions d'activation. Ce n'est que lorsqu'un nouveau besoin apparaît que celles-ci sont enrichies. Nous avons relevé dans la littérature deux types de conditions différentes de celles proposées par Mittal et Falkenhainer (1990) :

- Gelle et Weigel (1995) proposent des prémisses d'activation RV de type fonction numérique :
$$(v_1 \geq 10) \rightarrow \text{activation} : \mathbb{V}_c$$
- Gelle (1998) propose un type de prémisse d'activation basé sur le type *objet* des variables. Si les variables de la prémisse sont des instances d'un objet de type T_1, alors il y a ajout ou retrait d'un ensemble de variables ou d'un objet de type T_2 :
$$(M \text{ est du type } T_1) \rightarrow \text{activation} : \{T_2\}$$

Soininen et Gelle (1999) proposent un type particulier de conséquent qui permet de gérer les multi-instances de variables. Les contraintes d'activation qu'ils proposent expriment en une seule expression différentes activations de variables :

$$(c_1, \ldots, c_j, \neg c_{j+1}, \ldots, \neg c_k) \rightarrow \text{activation} : m\{v_1, \ldots, v_l\}n \text{ où } 0 \leq m \leq n$$

Ce type de contrainte d'activation exprime le fait que si les contraintes $c_1, \ldots, c_j$ sont satisfaites par une instanciation A et que les contraintes $c_{j+1}, \ldots, c_k$ ne le sont pas ou ne sont pas actives, alors seul un sous-ensemble des variables appartenant au conséquent de la contrainte est activé. Le cardinal de l'ensemble des variables activées est compris entre m et n :

- si $m = 1$ et si n correspond au nombre de variables du conséquent, alors la contrainte correspond à un *OU logique* : au moins une variable du conséquent doit être activée,
- si $m = n = 1$, alors la contrainte correspond à un *OU exclusif* : une seule variable du conséquent doit être activée,
- si $m = 0, \forall n \neq 0$, alors les variables du conséquent sont considérées comme optionnelles : n variables du conséquent sont potentiellement activables.

6.1.4 Approches existantes et activation de contraintes

L'activation de contraintes de compatibilité des *CSPs* conditionnels de Mittal et Falkenhainer (1990) et des *CSPs* à états de Veron (2001) se fait de manière transparente. Les contraintes de compatibilité ne prennent part au filtrage que lorsque toutes les variables sur lesquelles elles portent sont actives. Sinon, elles sont considérées comme trivialement satisfaites.

Pour garder la maîtrise des contraintes de compatibilité prises en compte dans les mécanismes de filtrage, une variable supplémentaire « de prise en compte », initialement inactive, peut être ajoutée artificiellement. Lorsque toutes les variables de la contrainte et la variable « de prise en compte » sont actives, la contrainte participe au mécanisme de filtrage. Sinon, elle est considérée comme trivialement satisfaite.

S'il est facile d'ajouter cette variable artificielle dans les contraintes de compatibilité de type table, cette idée est plus délicate pour les contraintes de compatibilité de type fonction numérique.

6.1.5 Synthèse

Les contraintes d'activation introduites par Mittal et Falkenhainer (1990) permettent d'ajuster un modèle de contraintes à un problème particulier. Seuls les éléments indispensables au problème participent ainsi à sa résolution. L'activation (resp. la désactivation) de variables peut avoir des répercussions sur la prise en compte (resp. la non prise en compte) de contraintes. Les contraintes de compatibilité ne sont pas explicitement ajoutées ou retirées des problèmes courants. Elles sont prises en compte dans le mécanisme de filtrage lorsque toutes leurs variables sont actives. Sinon, elles sont considérées comme trivialement satisfaites.

Pour éviter d'inclure des variables qui ne peuvent pas être cohérentes avec le problème courant, les *CSPs* à états proposés par Veron (2001) ajoutent une condition supplémentaire d'activation sur les domaines des variables inactives : une variable ne peut être activée que si la prémisse de la condition d'activation est vérifiée et si son domaine de définition contient au moins une valeur compatible avec le problème courant. Il est démontré par Veron (2001) que les *CSPs* à états sont équivalents aux *CSPs* conditionnels pour des problèmes « à plat ».

Les *CSPs* à états permettent de représenter les structures hiérarchiques des problèmes par l'emploi de méta-problèmes. Les méta-problèmes possèdent un état qui indique s'ils appartiennent ou pas à la solution. L'activation des méta-problèmes est régie par quatre règles portant sur l'état des éléments du méta-problème.

Les prémisses des conditions d'activation se déclinent en deux types différents qui possèdent chacun une sémantique bien définie. Le premier est défini sur les valeurs des variables et regroupe les prémisses de type table de compatibilité et de type fonction numérique. Le second est défini sur l'état des variables et correspond à un raccourci d'écriture des prémisses du premier type.

Il est à remarquer que très peu de travaux portent sur l'activation de variables, de contraintes et de méta-problèmes, bien que ce type de problématique se pose fréquemment en aide à la conception.

6.2 Choix de techniques d'activation par rapport à notre application et extension

Nous avons évoqué dans les sections précédentes l'activation de variables, de contraintes et de méta-problèmes. Notre application nécessite ces trois formes d'activation. Dans cette section, nous présentons, pour chacune d'entre elles, le principe d'implémentation retenu que nous illustrons par des exemples tirés de notre application.

6.2.1 Activation de variables

6.2.1.1 Principe retenu

Les contraintes *RN*, définies par $P(\mathbb{V}_p) \rightarrow$ désactivation : $\{\mathbb{V}_c\}$, peuvent être exprimées sous forme d'une contrainte de compatibilité liant les variables de la prémisse $\mathbb{V}_p$ aux variables

des prémisses des autres contraintes d'activation activant les variables du conséquent $\mathbb{V}_c$. Les contraintes d'activation *RN* évitent ainsi de lister de manière exhaustive la combinatoire des arrangements interdits entre les variables des prémisses des contraintes qui activent et qui désactivent un même sous-ensemble de variables. Les travaux de Haselböck (1993) ont montré que les contraintes *RN* peuvent être contournées de manière similaire.

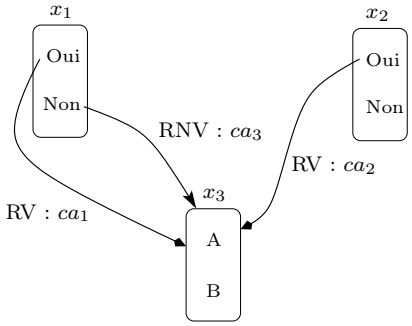
Considérons, par exemple, le *CSP* $P_1 = (\mathbb{V}, \mathbb{D}, \mathbb{C})$ défini par :

- un ensemble de variables initialement actives $\mathbb{V}_a = \{x_1, x_2\}$ et un ensemble de variables initialement inactives $\mathbb{V}_i = \{x_3\}$
- de domaine $D_{x_1} = \{Oui, Non\}, D_{x_2} = \{Oui, Non\}, D_{x_3} = \{A, B\}$
- trois contraintes d'activation ca_1, ca_2 et ca_3 :

ca_1		ca_2		ca_3	
x_1		x_2		x_1	
Oui	$\rightarrow$: activation$\{x_3\}$	Oui	$\rightarrow$: activation$\{x_3\}$	Non	$\rightarrow$: désactivation$\{x_3\}$

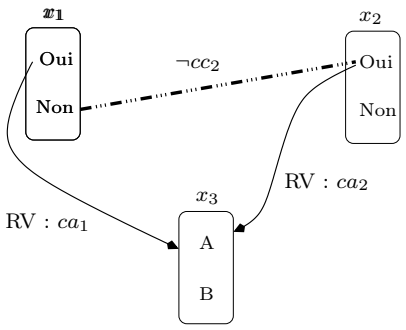
Les solutions du *CSP* P_1, illustré par la figure 6.1, sont équivalentes à celles du *CSP* $P_2 = P_1$, illustré par la figure 6.2, dans lequel la contrainte d'activation ca_3 de P_1 est exprimée sous forme d'une contrainte de compatibilité de type table :

x_1	x_2
Oui	Oui
Oui	Non
Non	Non



Légende :
- Contraintes d'activation de type *RN*
- Contraintes d'activation de type *RV*

FIG. 6.1 – P1



Légende :
- Contraintes d'activation de type *RV*
- Combinaison de valeurs interdites

FIG. 6.2 – P2

Il est plus fréquent, en modélisation, d'autoriser l'ajout d'éléments dans les modèles que de l'interdire, cette interdiction étant de plus contournable. Dans notre application, nous utilisons donc uniquement des contraintes d'activation de variables *RV*. Elles sont soit de type table de compatibilité, soit de type fonction numérique.

6.2.1.2 Exemples applicatifs

Exemple de contrainte d'activation de variables de type table

Par exemple, la cémentation consiste à chauffer l'acier au contact d'un cément[1] pour lui faire acquérir certaines propriétés. Un tel traitement thermique, variable symbolique *Cémentation*, nécessite la prise en compte de l'épaisseur de cémentation à 0.35 % de carbone, variable continue *Ec*, du rapport de l'épaisseur cémentée sur l'épaisseur totale de la pièce, variable continue *Ratio_Ec_Ep*, ainsi que l'atmosphère du four de cémentation, variable symbolique *Afc*. Ceci se traduit par la contrainte d'activation de type table de compatibilité présentée par le tableau 6.1.

TAB. 6.1 – Activation des variables nécessaires à une trempe avec cémentation

Cémentation
oui

$\rightarrow$ *activation* : $\{$ *Ec, Ratio_Ec_Ep, Afc* $\}$

Exemple de contrainte d'activation de variables de type fonction numérique

Les axes trempés peuvent présenter des épaulements, tels que l'illustre la figure 6.3.

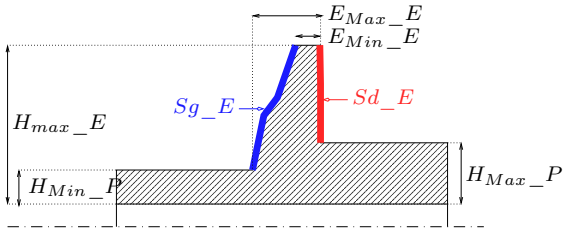


FIG. 6.3 – Exemple d'axe cylindrique alésé avec épaulement

La présence d'un épaulement peut entraîner l'apparition d'une déformation dite « parapluie », variable symbolique D_5. Cette déformation se décline en deux types : « parapluie gauche » et « parapluie droit ». La déformation de type « parapluie gauche » est illustrée sur la partie gauche de la figure 6.4, tandis que celle de type « parapluie droit » sur la partie droite de la figure 6.4.

Cette déformation « parapluie » se détermine en fonction de deux variables qui correspondent à la surface gauche de l'épaulement, variable continue Sg_E et à sa surface droite, variable continue Sd_E. Seuls les épaulements significatifs sont pris en compte.

Un épaulement est significatif si le rapport hauteur maximum de l'épaulement moins la hauteur moyenne de la pièce sur l'épaisseur moyenne de l'épaulement est supérieur à 2. Ceci se traduit par la contrainte d'activation de type fonction numérique suivante :

$$\left(\frac{2H_{max}\_E-(H_{Max}\_P+H_{Min}\_P)}{(E_{Max}\_E+E_{Min}\_E)} \geq 2\right) \rightarrow activation : \{Sg\_E,\ Sd\_E, D_5\}$$

[1]Un cément est une substance qui, chauffée au contact d'un métal, diffuse certains de ses éléments plus ou moins profondément dans le métal (Le petit Robert, 1996).

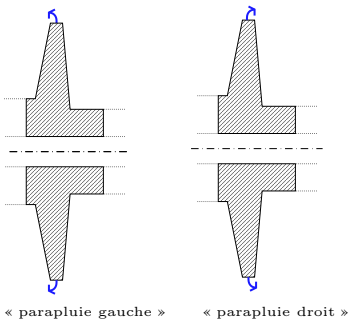


FIG. 6.4 – Déformation de type « parapluie »

6.2.2 Activation de contraintes

6.2.2.1 Principe retenu

Les contraintes de compatibilité et d'activation sont prises en compte de manière transparente dès que toutes leurs variables sont actives. Pour garder la maîtrise des contraintes utilisées dans la résolution sans ajouter de variable artificielle, nous proposons d'étendre les contraintes d'activation à l'activation de contraintes.

De manière identique aux variables, nous associons aux contraintes un état *actif* ou *inactif* qui indique si celles-ci participent ou non à la résolution du problème. Ces contraintes d'activation activent aussi bien des contraintes de compatibilité $\mathbb{C}_C$ que d'activation $\mathbb{C}_A$. Lorsque leur prémisse $P(\mathbb{V}_p)$ est vérifiée, les contraintes $\mathbb{C}_c$ appartenant au conséquent sont ajoutées au problème. Elles sont alors prises en compte pour sa résolution. Les contraintes d'activation de contraintes sont de la forme :

$$P(\mathbb{V}_p) \rightarrow activation : \{\mathbb{C}_c\} \text{ où } \mathbb{C}_c = \{\mathbb{C}_C \cup \mathbb{C}_A\}$$

Comme pour les contraintes d'activation de variables, les prémisses des contraintes d'activation de contraintes peuvent porter sur les valeurs des variables (RV ou RN) ou sur leur état (ARV, ARN).

Dans le cadre de notre application, seules les contraintes d'activation de contraintes permettant l'ajout de contraintes (RC, pour *Required Constraints*) sont utilisées. Celles-ci sont soit de type table de compatibilité, soit de type fonction numérique.

6.2.2.2 Exemples applicatifs

Par exemple, afin de caractériser la dureté après trempe et d'approximer les déformations potentielles, les métallurgistes exploitent deux abaques expérimentaux correspondant au diagramme de phases (diagramme TRC) spécifique à l'acier trempé et à un faisceau de lois de refroidissement. Ces deux abaques lient la température, variable continue *Température*, au temps, variable continue *Temps*. La superposition de ces deux abaques permet d'estimer les temps de premier changement de phases à cœur et en surface de la pièce. Plus le laps de temps

entre les premiers changements de phases à cœur et en surface est important, plus la pièce aura tendance à se déformer. Pour déterminer les temps de premier changement de phases, nous modélisons les diagrammes TRC par des contraintes numériques binaires définies par morceaux et les lois de refroidissement par des polynômes de degré 4. Les arbres quaternaires relatifs aux deux abaques sont alors générés et fusionnés pour déterminer ces deux points particuliers.

Le choix de l'acier utilisé est réalisé par l'utilisateur, parmi une large gamme de matériaux. Le faisceau de lois de refroidissement est sélectionné à partir de la nature du fluide de trempe et de sa température. La loi de refroidissement relative au problème est choisie dans le faisceau sélectionné suivant le diamètre de l'axe trempé. Il existe donc un grand nombre de contraintes portant sur la même paire de variables. Or, il est impossible de prendre en compte de manière simultanée toutes ces contraintes dans le mécanisme de filtrage, puisqu'elles ne sont pas cohérentes entre elles. Elles sont donc initialement inactives. Ce n'est que lorsque l'acier, variable symbolique TRC, la nature du fluide de trempe, variable symbolique FdT, sa température, variable continue T_FdT, et le diamètre de l'axe, variable continue d, sont valuées que les contraintes numériques correspondantes au problème sont activées et prises en compte.

La détermination des temps de premier changement de phase une fois les choix du matériau et de certaines caractéristiques du procédé de traitement thermique décidés n'est pas aberrante. Il s'avère, en effet, que les métallurgistes ne peuvent conclure sur la présence, par exemple, d'une déformation de type « bobine-tonneau » qu'après avoir choisi leur acier, leur nature de fluide de trempe, sa température et le diamètre de l'axe.

Dans notre application, la contrainte permettant d'activer un diagramme TRC est de type table de compatibilité et elle est de la forme :

TRC			
$30CrNiMo8$	$\rightarrow$	$activation:$	$\{\boldsymbol{TRC\_30CrNiMo8}(x,y) = \bigcup_{i=1}^{n}(f_i(x,y))\}$
$42CrMo4$	$\rightarrow$	$activation:$	$\{\boldsymbol{TRC\_42CrMo4}(x,y) = \bigcup_{i=1}^{l}(g_i(x,y))\}$
$90MnV8$	$\rightarrow$	$activation:$	$\{\boldsymbol{TRC\_90MnV8}(x,y) = \bigcup_{i=1}^{k}(h_i(x,y))\}$

La contrainte d'activation des lois de refroidissement est de type table de compatibilité et elle est de la forme :

FdT	T_FdT	d			
eau	< 25	$[30,60]$	$\rightarrow$	$activation:$	$\{LR1 = a_1x^4 + b_1x^3 + c_1x^2 + d_1x + e_1\}$
$huile$	< 25	$[30,60]$	$\rightarrow$	$activation:$	$\{LR2 = a_2x^4 + b_2x^3 + c_2x^2 + d_2x + e_2\}$
gaz	< 25	$[30,60]$	$\rightarrow$	$activation:$	$\{LR3 = a_3x^4 + b_3x^3 + c_3x^2 + d_3x + e_3\}$

Comme à chaque acier correspond un diagramme TRC, le choix d'un matériau va activer une contrainte numérique définie par morceaux. Cette contrainte définie par morceaux va décrire la frontière entre la phase austénitique et toutes les autres phases du diagramme de phases, comme l'illustre la figure 6.5. Les caractéristiques du fluide de trempe (nature et température) et le diamètre de l'axe définissent les lois de refroidissement à cœur et en surface à prendre en compte. La description du fluide de trempe et du diamètre de l'axe activeront les deux contraintes numériques correspondant aux lois de refroidissement à considérer. Il faudra

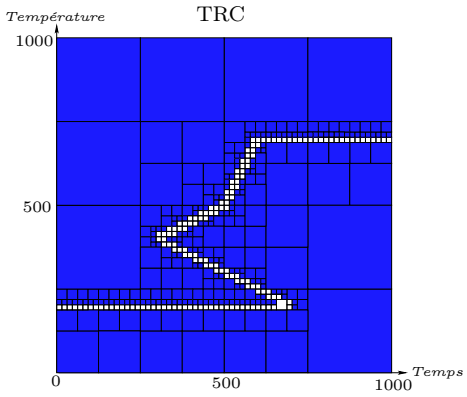


FIG. 6.5 – Exemple de diagramme *TRC*

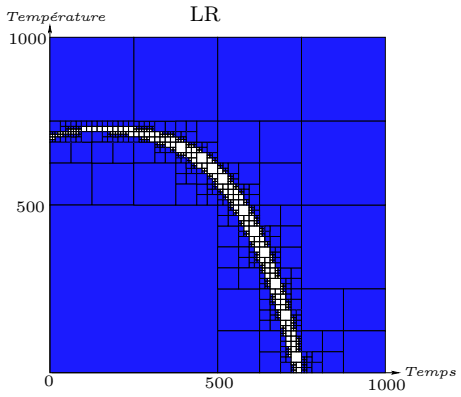


FIG. 6.6 – Exemple de loi de refroidissement

générer deux arbres quaternaires correspondant au refroidissement à cœur et en surface. L'une de ces deux lois est illustrée par la figure 6.6.

La première contrainte activée va générer son arbre quaternaire. La seconde ne sera générée que partiellement sur les zones cohérentes de la première contrainte par l'appel à la fonction FUS GEN, présentée par l'algorithme 20 page 103.

Par exemple, si l'utilisateur décrit d'abord les caractéristiques du fluide de trempe et du diamètre d'axe, les arbres quaternaires associés aux lois de refroidissement à cœur et en surface vont être générés. Puis, lorsque l'utilisateur choisit le matériau, la contrainte numérique définie par morceaux correspondant au diagramme *TRC* est activée. Comme les arbres quaternaires portant sur les variables *Temps* et *Température* existent déjà, les arbres quaternaires associés au diagramme *TRC* ne sont générés que sur les zones cohérentes définies par les lois de refroidissement. La fusion opportuniste des arbres détermine ainsi les points d'intersection entre le diagramme *TRC* et les lois de refroidissement. Si plusieurs points d'intersection existent, seul le premier est exploité dans le raisonnement.

L'activation de contraintes ne s'effectue que sur des contraintes qui lient la même paire de variables sans pour autant être cohérentes entre elles. Il est à noter que cette activation crée une orientation au niveau de la propagation. Cela ne gène en rien la prédiction des distorsions à partir de la description d'un procédé de traitement thermique qui « coule » dans le bon sens ; tandis que la conception d'un procédé de traitement thermique (caractérisation du matériau et du fluide de trempe) à partir de restrictions sur les distorsions est à contre-courant et se révèle impossible à conduire.

6.2.3 Activation de sous-problèmes

6.2.3.1 Principe retenu

Les choix de conception peuvent conduire à ajouter dans le problème courant des variables et des contraintes constituant des sous-problèmes. Pour éviter de lister l'ensemble des variables et des contraintes des conséquents des contraintes d'activation, nous les organisons en groupes.

Un groupe G se définit comme un ensemble de variables, de contraintes et de sous-groupes. Les groupes permettent ainsi de structurer un modèle de connaissances afin de faciliter sa modélisation et sa compréhension.

De manière identique aux variables et aux contraintes, nous associons aux groupes un état *actif* ou *inactif* qui indique si ceux-ci sont à considérer dans la résolution du problème courant. Nous élargissons ainsi le champ des contraintes d'activation aux groupes. Notons $\mathbb{G}$ l'ensemble des groupes du problème. Les contraintes d'activation sont maintenant de la forme :

$$P(\mathbb{V}_p) \rightarrow activation : \{\mathbb{V}_c \cup \mathbb{C}_c \cup \mathbb{G}_c\}$$

Contrairement aux sous-problèmes des *CSPs* à états de Veron (2001), les groupes contiennent un ensemble d'éléments localement actifs. Lorsqu'un groupe devient actif, seuls ses éléments (variables, contraintes, sous-groupes) localement actifs sont pris en compte dans le problème. Les variables, les contraintes et les groupes possèdent donc deux états : un état local, qui indique leur état par rapport au groupe qui les contient et un état global, qui indique s'ils participent ou non à la solution. Nous ne conservons que la première règle des *CSPs* à état (Veron, 2001) que nous traduisons de la manière suivante.

Un élément (variable, contrainte et groupe) est globalement actif si :

- le groupe qui le contient éventuellement est globalement actif,
- son état local est actif.

6.2.3.2 Exemples illustratifs

Par exemple, la présence d'un épaulement à l'extrémité gauche de l'axe cylindrique peut entraîner l'apparition d'une déformation dite « parapluie ». Celle-ci peut être modulée (amplifiée ou réduite) par la présence d'un trou à l'extrémité gauche. En effet, les trous peuvent générer aux extrémités une déformation de type « écartement-resserrement ». La présence d'une déformation de type « resserrement » diminue (resp. augmente) la déformation « parapluie droit » (resp. « parapluie gauche »), comme l'illustre la partie droite de la figure 6.7, alors qu'une déformation de type « écartement » augmente (resp. diminue) la déformation « parapluie droit » (resp. « parapluie gauche »), comme l'illustre la partie gauche de la figure 6.7. La présence d'un trou aux extrémités doit donc être prise en compte pour caractériser l'intensité de la déformation « parapluie ».

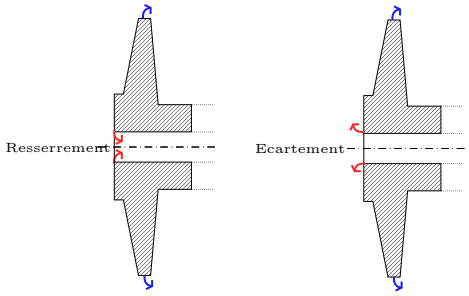


FIG. 6.7 – Amplification du « parapluie droit »

La présence éventuelle d'un épaulement est modélisée par le groupe localement inactif *Eeg*, figure 6.8, contenant :

- cinq variables localement actives caractérisant :
  - la surface gauche de l'épaulement, variable continue *Sg*,
  - la surface droite de l'épaulement, variable continue *Sd*,
  - le rapport des deux surfaces, variable continue *R*,
  - le type de déformation, variable symbolique *D*,
  - et l'intensité potentielle de déformation, variable continue I_p,
- deux contraintes localement actives :
  - la première de type fonction numérique, nommée *Ratio*, calculant le rapport des surfaces de l'épaulement :

$$R = \frac{Sg}{Sd}$$

  - la seconde de type table de compatibilité mixte, nommée *Déformation*, caractérisant le type de déformation en fonction du rapport des surfaces de l'épaulement :

R	D
1	Pas de déformation
> 1	« parapluie droit »
< 1	« parapluie gauche »

- et un sous-groupe localement inactif, nommé *Tg*, correspondant à la présence d'un trou au niveau de l'extrémité gauche, contenant :
  - deux variables localement actives caractérisant :
    - la position de l'épaulement par rapport au trou, variable symbolique *Position*,
    - la modulation de l'intensité de déformation, variable continue m^1,
  - et une contrainte localement active de type table de compatibilité, nommée *Moduler*, permettant de moduler l'intensité de déformation de l'épaulement à partir de sa position :

Position	m^1
proche	$[1.5, 3]$
éloigné	1

Une contrainte finale, qui n'est pas représentée sur la figure 6.8, calcule l'intensité finale de déformation, variable continue I_f, en fonction de l'intensité potentielle de déformation, I_p et des coefficients de modulation, m^j :

$$I_f = I_p \times \prod_{j=1}^{k} m^j$$

La contrainte d'activation des éléments relatifs à la présence d'un épaulement à l'extrémité gauche se traduit par l'activation du groupe *Eeg* :

$$\left(\frac{2H_{max\_Eg} - (H_{Max\_P} + H_{Min\_P})}{E_{Max\_Eg} + E_{Min\_Eg}} \geq 2\right) \quad \rightarrow activation : \quad \{\text{Eeg}\}$$

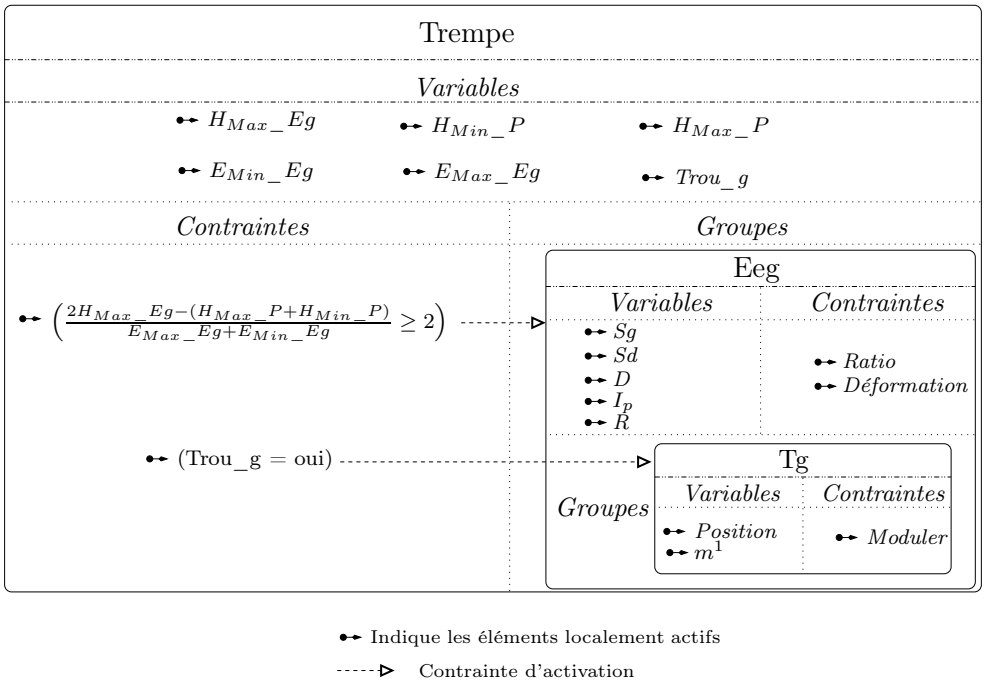


FIG. 6.8 – Exemple de sous-groupe : *Eeg* et *Tg*

L'activation du sous-groupe *Eeg*, sans présence de trou à l'extrémité gauche, intègre les variables *Sg*, *Sd*, *R*, *D* et I_p à la solution et les contraintes *Ratio* et *Déformation* au mécanisme de résolution. La présence supplémentaire d'un trou à l'extrémité gauche, contrainte (*Trou_g* = oui) vérifiée, intégrera au problème les éléments du sous-groupe *Tg* : les variables *Position* et m^1 ainsi que la contrainte *Moduler*.

6.3 Synthèse

Les contraintes d'activation permettent d'adapter un modèle de connaissances à un problème particulier par l'ajout et le retrait d'éléments. Les premières contraintes d'activation, introduites par Mittal et Falkenhainer (1990), ajoutent et retirent d'un problème courant « à plat » des ensembles de variables, sans vérifier que les variables activées sont cohérentes avec le problème. Les *CSPs* à états, introduits par Veron (2001), empêchent l'activation de variables incohérentes et permettent de modéliser le problème de manière structurée par la définition de méta-problèmes.

Pour garder la maîtrise des contraintes participant au mécanisme de résolution, nous étendons la notion de contrainte d'activation à l'activation de contraintes. Les contraintes peuvent être ainsi explicitement ajoutées au problème courant par des contraintes d'activation *RC* de type table de compatibilité et de type fonction numérique :

$$P(\mathbb{V}_p) \rightarrow activation : \{\mathbb{C}_c\}$$

Pour structurer le modèle de connaissances, nous définissons la notion de groupes composés

de variables, de contraintes et de sous-groupes. L'ajout d'un groupe au problème courant est réalisé explicitement par les contraintes d'activation de groupes. Contrairement aux méta-problèmes de Veron (2001), les groupes possèdent des éléments initialement localement actifs. Ceux-ci sont pris en compte lorsque leur état global est actif.

Afin de faciliter l'écriture du modèle, nous regroupons dans une seule et même contrainte les contraintes d'activation de variables, les contraintes d'activation de contraintes et les contraintes d'activation de groupes. Les contraintes d'activation sont alors de la forme :

$$P(\mathbb{V}_p) \rightarrow activation : \{\mathbb{V}_c \cup \mathbb{C}_c \cup \mathbb{G}_c\}$$

Chapitre 7

Application et modèle de raisonnement

Les travaux présentés dans ce mémoire se sont déroulés dans le cadre du projet européen *VHT*[1], concernant la mise au point d'un environnement d'aide à la conception d'opérations de traitement thermique sans recourir à la simulation *FEM*. Cet environnement regroupe deux outils d'aide à la conception, l'un à base de cas (qui n'est pas exposé dans ce mémoire), l'autre à base de contraintes. La mise au point de l'outil à base de contraintes a duré trois ans et a nécessité différentes phases menées en parallèle :

- extraction des connaissances à partir d'entretiens réguliers avec les experts industriels et scientifiques du domaine,
- validation de ces connaissances par des essais,
- formalisation des connaissances validées sous forme de *CSP*,
- spécification et mise au point du moteur de propagation de contraintes exploitant les connaissances formalisées.

En s'inspirant de modèles de traitement thermique tirés de la littérature, comme celui de Oliveira et al. (1986), plusieurs modèles de connaissances portant sur la famille des axes cylindriques ont été élaborés. Les premiers modèles (David et al. (2003), Vareilles et al. (2003)), relativement simples, ont permis d'identifier les différents types de connaissances à prendre en compte dans les modèles. Cette identification a débouché sur la sélection des concepts *CSPs* à utiliser et à adapter pour mettre au point le moteur de propagation de contraintes. Un modèle plus complexe (Aldanondo et al. (2005c), Aldanondo et al. (2005b)) présenté dans ce chapitre, a ensuite été élaboré. Celui-ci permet à la fois de concevoir une opération de traitement thermique et de prédire les distorsions.

Nous allons, dans un premier temps, présenter l'architecture générale du modèle de raisonnement en termes de variables, de contraintes et de groupes. Puis, nous exposerons le fonctionnement du moteur de propagation en présentant la manière dont sont agencées les différentes méthodes de filtrage. Nous aborderons, dans la deuxième section, les deux modes d'exploitation possible de ces modèles, à savoir la prédiction des distorsions à partir de la description

[1]Virtual Heat Treatment, projet n° G1RD-CT-2002-00835.

d'une opération de traitement thermique et la conception d'une opération minimisant les risques de déformations. Nous terminerons, dans la troisième section, par les différentes pistes d'améliorations de l'outil d'aide à la conception en termes de modèles de raisonnement et d'exploitation.

7.1 Élaboration du modèle et du moteur de propagation

Nous présenterons dans cette section l'architecture d'un des modèles de connaissance (Aldanondo et al. (2005c)). Celle-ci regroupe les différents concepts de *CSPs* présentés dans les chapitres précédents. Puis nous présenterons le fonctionnement général du moteur de propagation, à partir des différents algorithmes de filtrage.

7.1.1 Modèle de connaissance

Nous présentons ici l'architecture générale du modèle de connaissances en termes de variables, de contraintes et de groupes. Ce modèle possède une centaine de variables, rassemblées dans une dizaine de groupes, une vingtaine de contraintes d'activation, une cinquantaine de contraintes de compatibilité de type table et une soixantaine de contraintes de compatibilité de type fonction numérique.

7.1.1.1 Architecture générale

Le modèle de connaissances met en œuvre des variables caractérisant l'opération de traitement thermique et des attributs nécessaires à la quantification des déformations. Il a été établi, lors de la phase d'extraction, que :

- la déformation est un vecteur à cinq composantes (notée D_i),
- un ensemble de variables $\{P_1\}$ induit l'existence de chaque composante de déformation et permet une première évaluation dit *potentielle* de chaque composante de la déformation (attributs de déformation potentielle I_p^i),
- un nombre plus conséquent de variables $\{P_2\}$ amplifie ou réduit les valeurs de toutes les composantes de manière identique, à l'aide de vingt-six multiplicateurs (attributs de modulation m^j),
- l'intensité de déformation dite *finale* de chaque composante (attributs de déformation I_f^i) correspond au produit de l'intensité potentielle par composante (attributs de déformation I_p^i) par les attributs de modulation (attributs de modulation m^j) :

$$D = \bigcup_{i=1}^{5} \{D_i : I_f^i = I_p^i \times \prod_{j=1}^{26} m^j\}$$

- il est à noter qu'un ensemble de variables appartient à la fois à l'ensemble P_1 et P_2 et qu'il existe des contraintes entre les variables de $\{P_1\}$ et les variables de $\{P_2\}$ qui correspondent aux faisabilités techniques.

La figure 7.1 synthétise ces éléments pour un exemple comportant deux composantes de déformation D_1 et D_2, d'intensité potentielle I_p^1 et I_p^2 et finale I_f^1 et I_f^2, deux attributs de modulation m^1 et m^2, cinq variables appartenant à $\{P_1\}$ et six variables appartenant à $\{P_2\}$.

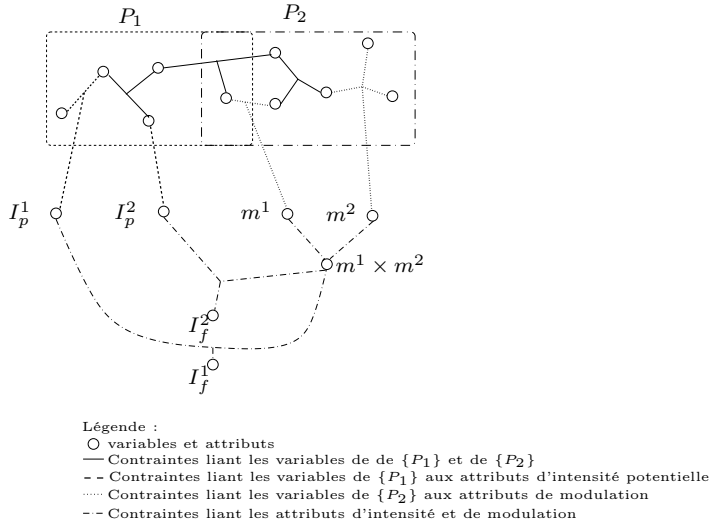


FIG. 7.1 – Architecture générale du modèle de connaissance

7.1.1.2 Variables de traitement thermique

Le premier dimensionnement des déformations potentielles par composante est lié aux variables de l'ensemble $\{P_1\}$. Ces variables caractérisent :

- la morphologie de l'axe de la pièce (épaisseur, diamètre, épaulement, trou, ...),
- le type de calage de l'axe lors de la trempe (suspendu, posé, calé, ...),
- la direction du fluide de trempe (perpendiculaire, parallèle),
- l'acier utilisé (30CrNiMo8, 42CrMo4, 90MnV8, ...).

Une vingtaine de variables de type P_1 ont été identifiées pour définir les intensités potentielles de chacune des cinq composantes de déformation I_p^i.

La modulation des intensités de chacune des déformations potentielles est réalisée à partir des variables $\{P_2\}$. Ces variables sont rassemblées en six ensembles caractérisant :

- le fluide de trempe (type de fluide, vélocité, agitation, ...),
- la préparation de la charge (perméabilité du support, distance entre les pièces trempées, rapport poids charge/poids maximum four, ...),
- la géométrie générale (nombre d'axe de symétrie de la pièce, rapport épaisseur maximale/diamètre maximum, variation épaisseur,...),
- les caractéristiques métallurgiques (trempabilité du matériau, cémentation, ...),
- l'historique du matériau (processus d'obtention du brut, relaxation, ...),
- le cycle de chauffe (classe du four de chauffe, vitesse de chauffe, ...).

De l'ordre de 70 variables de type P_2 ont été identifiées. Celles-ci sont associées à 26 attributs de modulation m^j.

7.1.1.3 Attributs de déformation

La déformation est un vecteur à cinq composantes notées D_i. Celles-ci correspondent aux déformations :

« bobine-tonneau » pour D_1 Pour des raisons essentiellement métallurgiques, un axe cylindrique peut prendre la forme d'une bobine ou d'un tonneau, tel que l'illustre la figure 7.2.

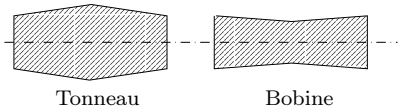


FIG. 7.2 – Déformation « bobine-tonneau »

« banane » pour D_2 Pour des raisons liées entre autre à la direction de circulation du fluide de trempe et à la présence d'une dissymétrie, un axe cylindrique peut prendre la forme d'une banane, tel que l'illustre la figure 7.3.

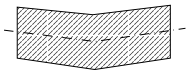


FIG. 7.3 – Déformation « banane »

« écartement-resserrement » pour D_3 En présence de trous aux extrémités, un phénomène d'écartement ou de resserrement de l'extrémité peut être observé, tel que l'illustre la figure 7.4.

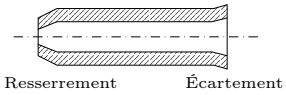


FIG. 7.4 – Déformation « écartement-resserrement »

« ovalisation » pour D_4 En présence d'un trou débouchant et d'une pièce mal positionnée, un phénomène d'ovalisation peut être observé, tel que l'illustre la figure 7.5.

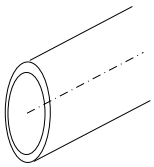


FIG. 7.5 – Déformation « ovalisation »

« parapluie » pour D_5 En présence d'un épaulement mince, celui-ci peut se déformer suivant une forme « parapluie » dont la poignée est orientée à droite ou à gauche, tel que l'illustre la figure7.6.

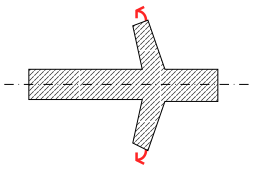


FIG. 7.6 – Déformation « parapluie » avec poignée orientée à gauche

Seules deux composantes sont systématiquement à considérer : la composante « bobine-tonneau » (D_1) et la composante « banane » (D_2). Les composantes « écartement-resserrement » (D_3) et « ovalisation » (D_4) sont susceptibles d'être prises en compte lors de la présence de trous aux extrémités, la composante « parapluie » (D_5) lors de la présence d'épaulements significatifs.

7.1.1.4 Contraintes

Les variables de $\{P_1\}$ et $\{P_2\}$ sont liées entre elles par des contraintes de compatibilité discrètes, mixtes et continues et des contraintes d'activation. Les variables de $\{P_1\}$ et $\{P_2\}$ sont liées respectivement aux attributs d'intensité potentielle I_p^i et aux attributs de modulation m^j par des contraintes de compatibilité mixtes de type table. Les attributs d'intensité potentielle I_p^i, d'intensité finale I_f^i et de modulation m^j sont liés par des contraintes de compatibilité de type fonction numérique. Chacune des contraintes du modèle résulte des connaissances et de l'expérience des experts en traitement thermique. Quatre emplois de différentes contraintes sont illustrés dans ce qui suit.

Activation de groupes La description de la pièce trempée et du procédé de traitement thermique peut nécessiter la prise en compte de variables, de contraintes et de groupes qui ne l'étaient pas jusque-là. Par exemple, la présence d'un trou le long de la pièce, variable symbolique *Trou* valuée à $\{oui\}$, va activer deux groupes de déformations correspondant au type « écartement-resserrement », groupe *ERg* et au type « ovalisation », groupe *Og*. Ces groupes contiennent les attributs d'intensités finale I_f^3 et I_f^4 et potentielle I_p^3 et I_p^4 correspondant aux composantes D_3 et D_4 du vecteur de déformation, ainsi que les variables de $\{P_1\}$, les contraintes et les groupes indispensables à la détermination des intensités potentielles I_p^3 et I_p^4 associées.

Activation de contraintes Certaines contraintes ne sont prises en compte qu'à un moment précis du raisonnement. Ces contraintes découlent directement des choix de conception. Par exemple, la détermination de l'intensité de déformation potentielle de la composante « bobine-tonneau » nécessite la connaissance du premier temps de changement de phases à cœur et en surface. Pour cela, deux abaques correspondant au diagramme *TRC* et aux lois de refroidissement à cœur et en surface sont superposés. Le choix du diagramme *TRC* est uniquement lié au choix du matériau. La sélection des lois de refroidissement repose sur les valeurs des variables correspondant à la nature du fluide de trempe, à sa température et au diamètre de l'axe trempé. Deux contraintes d'activation de type table permettent d'inclure dans le problème le diagramme *TRC* et les lois de refroidissement. Le diagramme *TRC* est modélisé par

une contrainte numérique binaire définie par morceaux, alors que les lois de refroidissement, le sont par des polynômes de degrés 4. Lorsque les contraintes deviennent actives, leur arbre quaternaire de conjonction est généré, voir le chapitre 5.

Tables de compatibilité mixtes Les attributs d'intensité potentielle I_p^i et les attributs de modulation m^j sont liés aux variables de $\{P_1\}$ et $\{P_2\}$ par des contraintes de compatibilité mixtes de type table. Ce sont les combinaisons de valeurs des variables de $\{P_1\}$ et $\{P_2\}$ qui vont valuer les attributs d'intensité et de modulation. Par exemple, le sixième attribut de modulation m^6, appartenant au groupe correspondant au fluide de trempe, dépend de la nature du fluide, variable symbolique N_FdT, de son agitation, variable symbolique A_FdT et de sa direction par rapport à l'axe de la pièce trempée, variable symbolique D_FdT. La table 7.1 présente cette relation. La valeur symbolique « $\neq$ rien » indique que quelque soit la valeur de la variable correspondante, le tuple est vérifié. Cette valeur particulière permet d'éviter de lister toute la combinatoire pour les variables symboliques.

TAB. 7.1 – Exemple de table de compatibilité mixte

N_FdT	A_FdT	D_FdT	m^6
liquide	turbulent	$\neq$ rien	$[1.047, 1.093]$
liquide	orienté	parallèle	$[1.163, 1.187]$
liquide	orienté	perpendiculaire	$[1.187, 1.467]$
gaz	turbulent	$\neq$ rien	$[1, 1.093]$
gaz	orienté	parallèle	$[1.093, 1.163]$
liquide	orienté	perpendiculaire	$[1.187, 1.28]$

Fonction numérique « simple » L'intensité finale de déformation I_f^i de chaque composante de déformation correspond au produit de l'intensité de déformation potentielle I_p^i par les 26 attributs de modulation m^j :

$$D = \bigcup_{i=1}^{5} \{D_i : I_f^i = I_p^i \times \prod_{j=1}^{26} m^j\}$$

Les valeurs des cinq attributs de déformation potentielle I_p^i et des 26 attributs de modulation m^j ont été évaluées par les experts en traitement thermique. Puis, elles ont été normalisées pour que l'intensité de déformation finale I_f^i de chaque composante D_i soit comprise dans l'intervalle $[1, 1000]$. La valeur 1 correspond à une intensité de déformation négligeable, la valeur 1000 a une intensité maximale de déformation.

Pour cela, l'influence des groupes de variables $\{P_1\}$ et $\{P_2\}$ a été analysée par les experts en traitement thermique. Il est apparu que :

- l'influence du groupe $\{P_1\}$ est de 30%. Les attributs d'intensité potentielle I_p^i sont donc compris dans l'intervalle $[1, 20]$.
- l'influence du groupe $\{P_2\}$ est de 70%. Le produit des 26 attributs de modulation m^j est compris dans l'intervalle $[1, 50]$.

La répartition de l'influence des six ensembles de variables de $\{P_2\}$ a de même été étudiée. Le tableau 7.2 présente les résultats obtenus.

TAB. 7.2 – Influence des six ensembles de variables de $\{P_2\}$

Ensemble	Influence	Nombre d'attributs de modulation	Valeurs
Fluide de trempe	35 %	6	$[1, 3.82]$
Préparation de charge	20%	3	$[1, 2.16]$
Métallurgie	15%	3	$[1, 1.78]$
Géométrie	15%	2	$[1, 1.78]$
Historique du matériau	10%	8	$[1, 1.53]$
Cycle de chauffe	5%	4	$[1, 1.24]$

Le modèle de connaissances possède finalement :

- 20 variables de type $\{P_1\}$, 70 variables de type $\{P_2\}$,
- 26 coefficients de modulation m^j dont le produit appartient à l'intervalle $[1, 50]$,
- 5 intensités potentielles de déformation I_p^i comprises dans l'intervalle $[1, 20]$,
- 5 intensités finales de déformation I_f^i égales au produit des I_p^i par les m^j. L'intensité de déformation finale de chaque composante est ainsi comprise dans l'intervalle $[1, 1000]$,
- une vingtaine de contraintes d'activation de variables, de contraintes et de groupes,
- une cinquantaine de contraintes de type table de compatibilité (discrète, continues et mixtes),
- une soixantaine de contraintes de type fonction numérique (dites « simples » et « complexes »),

Nous notons ici que les contraintes numériques « simples » doivent être projetées sur chacun des attributs pour pouvoir être filtrées par 2B-cohérence, d'où le nombre important de contraintes de type fonction numérique du modèle de connaissances.

7.1.2 Moteur de propagation

Le moteur de propagation, présenté par l'algorithme 21 page suivante, intègre les différentes méthodes de filtrage présentées dans les chapitres précédents. Il permet ainsi de répercuter sur l'ensemble des variables du modèle, *via* les contraintes, chaque choix de conception. Chaque choix correspond à une réduction du domaine d'une variable fournie par l'utilisateur.

Tout d'abord, nous incluons, grâce aux contraintes d'activation, dans le modèle courant les variables, les contraintes et les groupes qui ne l'étaient pas auparavant, appel à la fonction CA présentée par l'algorithme 22 page 132. Puis, les différentes méthodes de filtrage sont appliquées suivant les types de contraintes de compatibilité, par l'appel à la fonction CC présentée par l'algorithme 23 page 133.

La fonction CA, présentée par l'algorithme 22 page 132, teste les prémisses des contraintes d'activation auxquelles les variables actives nouvellement valuées (comprenant éventuellement

Alg. 21 MOTEUR(CSP : P)

– ∴ – ***Cet algorithme présente le fonctionnement général du moteur de propagation.***
– ∴ – P $= (\mathbb{V}, \mathbb{D}, \mathbb{C}_C, \mathbb{C}_A)$
– ∴ – Var_{filt} *est l'ensemble des variables à considérer pour filtrer les contraintes*
$Var_{filt} \leftarrow \emptyset$
– ∴ – Var_{val} *est l'ensemble des variables actives nouvellement valuées (domaine réduit à un singleton)*
$Var_{val} \leftarrow \emptyset$
Répéter
  – ∴ – *Présentation de l'ensemble des variables actives à valuer à l'utilisateur (de domaine non réduit à un singleton)*
  – ∴ – *Réduction d'une variable v par l'utilisateur* D'_v
  $D_v \leftarrow D_v \cap D'_v$
  Si (D_v est réduit à un singleton) **Alors**
    – ∴ – *v est nouvellement valuée*
    $Var_{val} \leftarrow v$
  Fin Si
  – ∴ – *La variable doit être considérée pour le filtrage*
  $Var_{filt} \leftarrow v$
  Répéter
    – ∴ – *Propagation sur les contraintes d'activation actives pour inclure de nouvelles variables, de nouvelles contraintes et de nouveaux groupes au problème*
    – ∴ – *Prise en compte dans le problème courant des variables nouvellement activées par ajout de celles-ci à l'ensemble* Var_{filt}
    $Var_{filt} \leftarrow (Var_{filt} \cup \mathrm{CA}(Var_{val}))$ (algorithme 22 page 132)
    – ∴ – *Filtrage sur les contraintes de compatibilité actives pour toutes les variables réduites*
    $Var_{val} \leftarrow \mathrm{CC}(Var_{filt})$ (algorithme 23 page 133)
  Jusqu'à (il ne reste plus de variables actives nouvellement valuées)
Jusqu'à (il ne reste plus de variables actives à valuer)

la saisie utilisateur) participent. Si les prémisses sont vérifiées, la fonction CA active les variables, les contraintes et les groupes des conséquents. Lorsque la prémisse active une contrainte « complexe » (correspondant, par exemple, à un abaque), il faut générer son arbre quaternaire et reconstruire les domaines de validité de ses variables :

- si un arbre existe déjà sur la paire de variables, l'arbre de conjonction est mis à jour par l'appel à la fonction FUS GEN, présentée par l'algorithme 20 page 103,
- si aucun arbre portant sur la paire de variable de la contrainte activée n'existe déjà, il faut générer l'arbre quaternaire sur l'ensemble de l'espace de recherche :
  - si la contrainte n'est pas définie par morceaux, son arbre est généré par l'appel à la fonction ARBRE QUATERNAIRE, présentée par l'algorithme 7 page 63,
  - si la contrainte est une contrainte d'égalité par morceaux, son arbre est généré par l'appel à la fonction QT ÉGAL, présentée par l'algorithme 12 page 85,
  - si la contrainte est une contrainte d'inégalité par morceaux, son arbre est généré par l'appel à la fonction QT INÉGAL, présentée par l'algorithme 18 page 97.
- une fois les arbres quaternaires générés, il faut reconstruire les domaines de compatibilité des variables par l'appel à la fonction DOMAINE COMPATIBLE, présentée par l'algorithme 9 page 69.

La fonction CA retourne l'ensemble des variables nouvellement actives qui sont ajoutées à l'ensemble des variables réduites sur lesquelles le filtrage a lieu. Les variables, les contraintes et les groupes sont des variables globales du problème.

La fonction CC, présentée par l'algorithme 23 page 133, filtre les contraintes à partir de la liste des variables actives nouvellement réduites (incluant l'ensemble des variables nouvellement actives). Les méthodes de filtrage utilisées dépendent du type des contraintes de compatibilité :

- les tables de compatibilité sont filtrées par la fonction REVISE_C, présentée par l'algorithme 3 page 35,
- les fonctions numériques « simples » sont filtrées par 2B-cohérence, appel à la fonction NARROW, présentée par l'algorithme 6 page 49,
- les fonctions numériques « complexes » (les contraintes discrétisées) sont filtrées par la procédure PROPAGER RÉDUCTION DOMAINE, présentée par l'algorithme 10 page 70, puis les domaines sont reconstruits par l'appel à la fonction DOMAINE COMPATIBLE, présentée par l'algorithme 9 page 69.

La fonction CC retourne l'ensemble des variables actives nouvellement valuées.

Une maquette développée en *Perl* nous a permis de valider les concepts présentés dans les chapitres précédents. Elle est accessible sur Internet à l'adresse suivante :

http://iena.enstimac.fr:20000/cgi-bin/vht.pl

Plusieurs modèles sont actuellement en ligne. Celui exposé dans la section précédente se nomme *Modele_VHT_2_version4*.

Le délivrable final du projet correspond au progiciel d'aide à la décision. Celui-ci est composé de deux outils : le premier à base de cas contenant un ensemble de cas industriels et une fonction de mesure de similarité, le second à base de contraintes contenant divers modèles de connaissances et un moteur de propagation de contraintes. Ce progiciel est en cours d'industrialisation par l'un des partenaires du consortium, *S&CC*[2].

[2]Sciences and Computers Consultants, Saint Étienne : http://www.scconsultants.com

Alg. 22 CA(LISTE : Var_{val})

– ∴ – ***Cet algorithme active, à partir de la liste des variables nouvellement valuées*** Var_{val}***, les variables, les contraintes et les groupes à inclure dans le problème courant.***
– ∴ – *Var_{filt} est l'ensemble des variables nouvellement actives*
– ∴ – *$\mathbb{C}_c$ est l'ensemble des contraintes de compatibilité actives de $\mathbb{C}_C$*
– ∴ – *$\mathbb{C}_a$ est l'ensemble des contraintes d'activation actives de $\mathbb{C}_A$*
$Var_{filt} \leftarrow \emptyset$
– ∴ – *Il faut vérifier les prémisses des contraintes actives auxquelles les variables actives nouvellement valuées participent*
Pour $v_v \in Var_{val}$ **Faire**
  Pour toutes les c^a actives dont v_v appartient aux variables de la prémisse $\mathbb{V}_p^{C^a}$ **Faire**
    – ∴ – *Si la prémisse de la contrainte d'activation est vraie*
    Si $(\mathrm{P}(\mathbb{V}_p^{C^a}) = \top)$ **Alors**
      Activer les variables, les contraintes et les groupes du conséquent
      $\mathbb{C}_c \leftarrow \mathbb{C}_c \cup$ ensemble des contraintes de compatibilité activées par C^a
      $\mathbb{C}_a \leftarrow \mathbb{C}_a \cup$ ensemble des contraintes d'activation activées par C^a
      $Var_{filt} \leftarrow$ ensemble des variables activées
      Si (le conséquent contient des fonctions numériques « complexes » : $\mathbb{C}_{complexe}$) **Alors**
        – ∴ – *Il faut générer leur arbre quaternaire*
        Pour toutes les contraintes $C_{(x,y)}$ de $\mathbb{C}_{complexe}$ **Faire**
          Si (un arbre $T_{(x,y)}$ existe déjà sur cette paire de variables) **Alors**
            – ∴ – *Utilisation de la* fusion opportuniste
            $T_{(x,y)} \leftarrow$ FUS GEN$(T_{(x,y)}C_{(x,y)})$ (algorithme 20 page 103)
          Sinon
            – ∴ – *Il faut générer son arbre totalement*
            Si ($C_{(x,y)}$ n'est pas définie par morceaux) **Alors**
              $T_{(x,y)} \leftarrow$ ARBRE QUATERNAIRE$(C_{(x,y)}, D_x^C, D_y^C)$ (algorithme 7 page 63)
            Sinon
              Si ($C_{(x,y)}$ est une contrainte d'égalité par morceaux) **Alors**
                $T_{(x,y)} \leftarrow$ QT ÉGAL$(C_{(x,y)}, D_x^C, D_y^C)$ (algorithme 12 page 85)
              Sinon
                – ∴ – *$C_{(x,y)}$ est une contrainte d'inégalité par morceaux*
                $T_{(x,y)} \leftarrow$ QT INÉGAL$(C_{(x,y)}, D_x^C, D_y^C)$ (algorithme 18 page 97)
              Fin Si
            Fin Si
          Fin Si
          $(D_x, D_y) \leftarrow (D_x, D_y) \cap$ DOMAINE COMPATIBLE$(T_{(x,y)})$ (algorithme 9 page 69)
          $Var_{filt} \leftarrow Var_{filt} \cup \{x, y\}$
        Fin Pour
      Fin Si
    Fin Si
  Fin Pour
Fin Pour
Retourner (Var_{filt})

Alg. 23 CC(LISTE : Var_{filt})

– ∴ – ***Cet algorithme filtre les contraintes de compatibilité actives à partir de leur type.***
– ∴ – $\mathbb{C}_c$ *est l'ensemble des contraintes de compatibilité actives de* $\mathbb{C}_C$
– ∴ – Var_{val} *est l'ensemble des variables actives nouvellement valuées*
$Var_{val} \leftarrow \emptyset$
– ∴ – *il reste des variables réduites à filtrer*
Tant Que ($Var_{filt} \neq \emptyset$) **Faire**
  $v_r \leftarrow$ dépiler une variable de Var_{filt}
  Pour toutes les contraintes de compatibilité actives $c^c \in \mathbb{C}_c$ auxquelles v_r appartient **Faire**
    – ∴ – *La méthode de filtrage dépend du type de la contrainte* c^c
    Si (c^c est de type table de compatibilité) **Alors**
      $Var_{filt} \leftarrow Var_{filt} \cup$ REVISE_C(c^c) (algorithme 3 page 35)
      $Var_{val} \leftarrow Var_{val} \cup$ ensemble des variables actives valuées par REVISE_C (algorithme 3 page 35)
    Sinon
      – ∴ – *La contrainte* c^c *est de type fonction numérique*
      Si (c^c est de type fonction numérique « simple ») **Alors**
        – ∴ – *La contrainte* c^c *est filtrée par 2B-cohérence*
        $Var_{filt} \leftarrow Var_{filt} \cup$ NARROW(c^c,$\mathbb{V}$) (algorithme 6 page 49)
        $Var_{val} \leftarrow Var_{val} \cup$ ensemble des variables actives valuées par NARROW (algorithme 6 page 49)
      Sinon
        – ∴ – c^c *est de type fonction numérique « complexe », son arbre quaternaire* T_{c^c} *est filtré*
        PROPAGER RÉDUCTION DOMAINE(T_{c^c},v_r) (algorithme 10 page 70)
        $(D_{v_r}, D_z) \leftarrow (D_{v_r}, D_z) \cap$ DOMAINE COMPATIBLE(T_{c^c}) (algorithme 9 page 69)
        Si (v_r ou z sont réduites) **Alors**
          $Var_{filt} \leftarrow Var_{filt} \cup (v$ ou $r)$
        Fin Si
        Si (v_r ou z sont valuées) **Alors**
          $Var_{val} \leftarrow Var_{val} \cup (v$ ou $r)$
        Fin Si
      Fin Si
    Fin Si
  Fin Pour
Fin Tant Que
Retourner (Var_{val})

7.2 Exploitation

L'outil d'aide à la conception permet à la fois de concevoir une opération de traitement thermique et de prédire les distorsions. Nous présentons dans cette section les deux modes d'exploitation de cet outil. Nous évoquerons, également, certaines de ses limites en termes d'exploitation de résultats et de garantie de cohérence.

7.2.1 Mode 1 : prédiction des déformations

En mode « prédiction des déformations », l'outil d'aide à la conception fournit des résultats conformes aux attentes des industriels :

- les choix de conception se répercutent *via* les contraintes sur l'ensemble du modèle en restreignant, de manière cohérente, les domaines de définition des variables,
- les temps de réponse sont tout à fait compatibles avec un mode interactif (de l'ordre de la seconde),
- et globalement, les résultats fournis sont cohérents avec les prédictions de déformations faites par les experts.

Nous illustrons ce mode de fonctionnement sur un exemple de traitement thermique de trempe (présenté plus en détail en annexes). La pièce étudiée est un axe sans trou à ses extrémités et sans épaulement en acier 42CrMo4. Les réductions des intensités de déformations finales I_f des composantes « bobine-tonneau » et « babane », suite à la saisie des valeurs décrivant :

1. la géométrie de la pièce,
2. le fluide de trempe,
3. la préparation de la charge,
4. le matériau et son historique avant traitement,
5. les caractéristiques métallurgiques et le cycle de chauffe.

sont illustrées sur la figure 7.7.

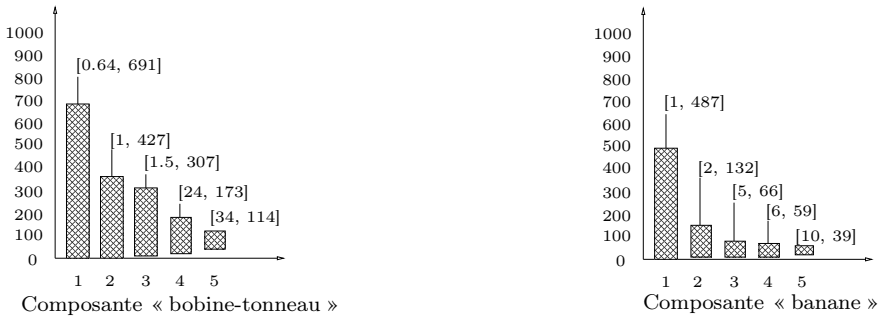


FIG. 7.7 – Réduction des intensités de déformation

L'exploitation des résultats reste, cependant, délicate. Les intensités finales fournies pour chaque composante de déformation, I_f^i, sont le plus souvent proches de l'intervalle de valeurs $[50, 200]$. En effet, les choix de conception réalisés par les utilisateurs expérimentés sont

rarement irraisonnés. Les intensités de déformation sont, par conséquent, comprises dans une fourchette relativement restreinte.

Cette forme de résultat permet de comparer différentes opérations de traitement thermique en termes de répercussions sur les déformations, mais elle ne permet pas actuellement de caractériser dans l'absolu l'importance de la déformation. Un travail d'analyse des résultats doit être mené pour définir les différents degrés de déformation (très faible, faible, moyenne, forte, très forte) associés aux valeurs des intensités finales de déformation par composante I_f^i.

Par exemple, une contrainte de compatibilité mixte, par composante de déformation, de type table permettrait de lier les intensités de déformation finale, I_f^i, à un indicateur qualitatif de déformation, variable symbolique Q^{D_i}, tel que l'illustre la table 7.3 :

TAB. 7.3 – Table d'interprétation qualitative des résultats

I_f^i	Q^{D_i}
< 50	très faible
$[50, 120]$	faible
$]120, 150]$	moyen
$]150, 180]$	forte
> 180	très forte

7.2.2 Mode 2 : conception d'opérations de traitement thermique

Le mode « conception d'opérations de traitement thermique » autorise l'utilisateur à contraindre les intensités de déformation finales I_f^i pour laisser l'outil caractériser certains paramètres de l'opération de traitement thermique (choix de l'orientation de la pièce, du cycle de chauffe).

En mode « conception d'opérations de traitement thermique », l'outil fournit des résultats satisfaisants, mais possède certaines limites. Ce mode de fonctionnement peut générer des problèmes de cohérence. En effet, les méthodes de filtrage employées pour garantir l'interactivité de l'outil sont de faible degré. Les choix utilisateurs sont donc répercutés de manière locale *via* les contraintes de compatibilité. Les choix de conception peuvent alors conduire le problème à devenir incohérent.

Pour illustrer nos propos, nous déroulons un scénario sur un système de contraintes où la réduction de l'intensité de déformation finale I_f^1 par un utilisateur conduit à la perte de cohérence du problème.

Système de contraintes

Considérons le système de contraintes, illustré par la figure 7.8, constitué de :

- trois variables de type $\{P_1\}$ et $\{P_2\}$, définies en 2.2.1.1 : la variable symbolique *Matériau* $\in \{P_1\}$ de domaine $D_{Matériau} = \{42CrMo4, 90MnV8\}$, la variable numérique discrète *Classe_de_four* $\in \{P_1 \cap P_2\}$ de domaine $D_{Classe\_de\_four} = \{7, 9\}$ et la variable symbolique $FdT \in \{P_2\}$ de domaine $D_{FdT} = \{\text{eau, huile}\}$,

– un attribut d'intensité de déformation potentiel I_p^1 de domaine $D_{I_p^1} = \{[1, 4]\}$, un attribut de modulation m^1 de domaine $D_{m^1} = \{[1, 3]\}$ et un attribut de déformation finale I_f^1 de domaine $D_{I_f^1} = \{[1, 12]\}$,

– trois contraintes de compatibilité de type table :

c_1

Classe_de_four	*FdT*
7	*eau*
9	*huile*

c_2

Matériau	*Classe_de_four*	I_p^1
$42CrMo4$	7	$[3, 4]$
$90MnV8$	9	$[1, 2]$

c_3

FdT	m^1
eau	$[1, 2]$
huile	$[2, 3]$

– une contrainte numérique « simple » $\boldsymbol{c_4}$ calculant l'intensité finale de déformation :

$$I_f^1 = I_p^1 \times m^1$$

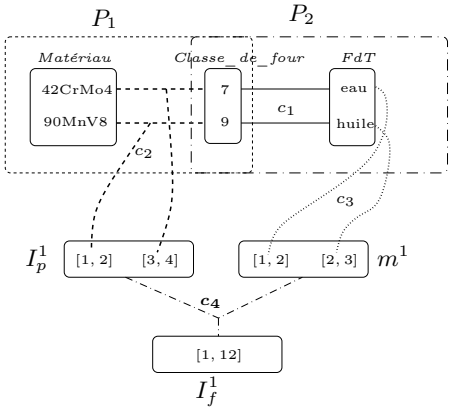


FIG. 7.8 – Architecture du système de contraintes

Scénario

Si l'utilisateur souhaite une faible intensité de déformation, il peut restreindre l'intensité de déformation finale I_f^1 à $\{< 2\}$. Cette réduction va se répercuter, *via* la contrainte numérique « simple » $\boldsymbol{c_4}$, sur les variables continues I_p^1 et m^1. Celles-ci seront réduites, par 2B-cohérence, à :

– $\{[1, 2]\}$ pour la variable continue I_p^1 :

$$\begin{aligned} D_{I_p^1} &= D_{I_p^1} \cap (D_{I_f^1} \oslash D_{m^1}) \\ D_{I_p^1} &= \mathbf{[1, 4]} \cap (\mathbf{[1, 2]} \oslash \mathbf{[1, 4]}) \\ D_{I_p^1} &= \mathbf{[1, 4]} \cap [\mathbf{\frac{1}{4}, 2}] \\ D_{I_p^1} &= \mathbf{[1, 2]} \end{aligned}$$

– et $\{[1, 2]\}$ pour la variable continue m^1 :

$$\begin{aligned} D_{m^1} &= D_{c_m^1} \cap (D_{I_f^1} \oslash D_{I_p^1}) \\ D_{m^1} &= \mathbf{[1, 3]} \cap (\mathbf{[1, 2]} \oslash \mathbf{[1, 2]}) \\ D_{m^1} &= \mathbf{[1, 3]} \cap [\mathbf{\frac{1}{2}, 2}] \\ D_{m^1} &= \mathbf{[1, 2]} \end{aligned}$$

La réduction du domaine de I_p^1 à $\{[1,2]\}$ va, à son tour, se transmettre aux variables *Matériau* et *Classe_de_four* par la contrainte c_2 et après propagation par REVISE_C (algorithme 3 page 35), $D_{Matériau} = \{90MnV8\}$ et $D_{Classe\_de\_four} = \{9\}$.

La réduction du domaine de m^1 à $\{[1,2]\}$ va, ensuite, se transmettre à la variable FdT par la contrainte c_3 et après propagation par REVISE_C (algorithme 3 page 35), $D_{FdT} = \{\text{eau}\}$.

La réduction du domaine de *Matériau* à {90MnV8} n'a aucune répercussion sur le système.

La réduction du domaine de *Classe_de_four* à {9} va, à son tour, se transmettre à la variable FdT par la contrainte c_1 et après propagation par REVISE_C (algorithme 3 page 35), $D_{Classe\_de\_four} = \{\emptyset\}$. En effet, le couple de valeurs (*Classe_de_four* = 9, FdT = eau) ne fait pas partie des combinaisons autorisées. Le problème ne possède plus de solution.

L'utilisation de méthodes de plus haut degré permettraient d'éviter ce problème d'incohérence. Mais, ces méthodes de filtrage compromettraient l'interactivité du système. L'adoption d'une démarche *essai-erreur* peut permettre de contourner ce problème.

7.3 Pistes d'améliorations

L'outil d'aide à la conception réalisé est conforme aux attentes des industriels en termes de prédiction qualitative des déformations, d'aide à la conception de procédés et de temps de réponse. Cependant, il doit encore être amélioré sur différents points.

Modèle et moteur de propagation

Plusieurs améliorations sont envisageables sur les modèles de connaissances et sur le moteur de propagation de contraintes.

Les experts ont noté que la restriction de l'intensité de déformation à une faible valeur, par exemple I_f^i à $\{< 20\}$, peut conduire à la conception d'une opération de traitement thermique minimisant bien les déformations mais n'améliorant que faiblement les caractéristiques mécaniques de l'acier trempé. Il ne faut pas oublier que le but originel des procédés de traitement thermique de trempe est l'amélioration des caractéristiques mécaniques des aciers et plus particulièrement leur dureté. Aucun attribut ne permet pour l'instant de caractériser la dureté après trempe. Un attribut visant à évaluer la dureté après trempe, variable continue DaT, sera prochainement intégré aux modèles de connaissances. Lorsque cet attribut sera inclus dans les modèles, un compromis entre l'augmentation de la dureté des aciers et la diminution des déformations pourra alors être recherché.

Pour l'instant, les modèles élaborés permettent de concevoir une opération de traitement thermique et de prédire les déformations sur des pièces de type axe cylindrique. Une autre famille de pièces fréquemment traitées chez les partenaires industriels du projet regroupe les engrenages. Un premier travail d'analyse a été mené sur ce type de pièces et il a été établi que deux vecteurs de déformations seraient nécessaires : le premier pour caractériser la déformation globale de la pièce, le second pour caractériser les déformations des dents des engrenages.

La cohérence des modèles de connaissances est actuellement vérifiée, lors de leur compilation, par des mécanismes de faible degré (équivalent à celui de l'arc-cohérence). C'est lors de l'utilisation de l'outil que les « erreurs » de modélisation sont détectées, analysées puis corrigées. Un mécanisme de vérification de cohérence de haut degré doit être mis en place pour limiter cette phase de mise au point des modèles. Comme cette vérification est réalisée lors de la compilation des modèles et non lors de leur utilisation, il est tout à fait envisageable d'utiliser des méthodes de plus haut degré.

Exploitation

Deux améliorations peuvent être apportées au niveau de l'utilisation de l'outil d'aide à la conception. La première porte sur l'interprétation des résultats. Les intensités de déformation de chacune des cinq composantes de déformation sont estimées par des intervalles de valeurs. Ces intervalles de valeurs permettent bien de comparer deux opérations de traitement thermique, mais pas d'interpréter, de manière absolue, les effets d'une opération de traitement thermique sur la déformation finale. Pour cela, nous devons ajouter aux modèles une contrainte de compatibilité mixte de type table liant les intensités finales de déformation à un indicateur de déformation. Un travail d'analyse des intensités de déformation pour chaque composante doit être réalisé.

La seconde amélioration porte sur l'ergonomie de la maquette *Web*. Celle-ci a été mise de côté, puisque la maquette avait pour but principal la validation des différents concepts utilisés et étendus. L'ergonomie est donc très simple. Chacune des variables du problème est néanmoins décrite par une légende qui indique sa nature, son domaine de définition et si elle vient d'être valuée, réduite ou activée. Actuellement, les variables apparaissent sous forme d'une liste. Il est, par conséquent, difficile d'avoir une vue générale du problème étant donné le nombre de variables. De plus, les déformations et leurs intensités apparaissent en fin de page et il est vite lassant de descendre vérifier les conséquences d'un choix de conception sur les déformations.

Un travail important doit donc être réalisé pour améliorer l'ergonomie de la maquette *Web*. Malgré la simplicité de son *IHM*, les industriels du projet l'ont cependant vite prise en main pour valider les modèles de connaissances.

Interface CBR-CSP

Le projet européen *VHT* concerne la mise au point d'un environnement d'aide à la conception regroupant deux outils : l'un à base de cas nommé *CBR*, l'autre à base de contraintes nommé *CSP*. Il pourrait être intéressant de coupler ces deux outils pour :

- aider la phase d'adaptation du cas résultat fourni par le *CBR* à l'aide du système à base de contraintes, comme le préconise Geneste et Ruet (2001) à partir de contraintes métier,
- valider l'opération de traitement thermique conçue à l'aide du système à base de contraintes par la recherche de cas similaires présents dans le *CBR*.

L'intérêt des deux possibilités n'a pas été évalué à ce jour, mais constitue un axe de recherche intéressant.

7.4 Synthèse

Ce chapitre a présenté l'un des modèles de connaissances que nous avons réalisé avec les experts industriels et scientifiques du projet. Ce modèle fait intervenir des connaissances se formalisant de différentes manières : tables de compatibilité discrètes, continues et mixtes, fonctions numériques « simples » et « complexes » et contraintes d'activation de variables, de contraintes et de groupes.

Le moteur de propagation de contraintes assemble donc différentes techniques de filtrage et les emploie de manière appropriée selon le type des contraintes à filtrer. Les choix de conception se répercutent d'abord sur les contraintes d'activation pour inclure les nouveaux éléments au problème courant, puis sur les contraintes de compatibilité pour propager les choix utilisateurs.

Les modèles de connaissances permettent de concevoir une opération de traitement thermique et de prédire, de manière qualitative, les déformations en résultant. Si le mode « prédiction des déformations » ne pose pas de problème particulier, le mode « conception de traitement thermique » ne garantit pas l'atteinte d'une solution. Ceci est simplement due à l'utilisation de méthodes de filtrage de faible degré, qui ne répercutent que localement les choix de conception.

Deux faiblesses peuvent être notées sur les deux modes d'exploitation. D'une part, les intensités de déformation pour chacune des composantes de déformation sont représentées par des intervalles de valeurs qui n'ont actuellement aucun sens dans l'absolu. Seule la comparaison d'opérations est véritablement exploitable. D'autre part, aucune information sur la dureté de l'acier après trempe n'est fournie à l'utilisateur. Or, le but des traitements de trempe est d'améliorer la dureté superficielle des aciers.

L'outil d'aide à la conception à base de contraintes répond aux attentes des industriels en termes de résultats fournis et de temps de réponses. Différentes améliorations sont néanmoins à mettre en œuvre :

- les modèles doivent être complétés pour :
  - interpréter les résultats dans l'absolu,
  - estimer la dureté après trempe d'un acier traité,
  - prendre en compte d'autres familles de pièces comme celle des engrenages,
- le moteur doit être complété pour :
  - vérifier plus fortement la cohérence des modèles,
  - trouver le compromis entre la minimisation des déformations et l'amélioration de la dureté,
- l'ergonomie doit être améliorée pour garder une vue générale des modèles.

L'environnement d'aide à la conception se compose d'un *CBR* et d'un *CSP*. Il serait intéressant de coupler ces deux outils pour aider à l'adaptation des cas résultats du *CBR* et pour valider les opérations de traitement thermique conçues à l'aide du système à base de contraintes.

Chapitre 8

Synthèse générale et perspectives

Nous présentons, dans ce chapitre, une synthèse générale de nos travaux de recherche. Dans un premier temps, nous positionnons le cadre et la problématique de nos travaux. Dans un deuxième temps, nous résumons notre contribution en méthodes de filtrage. Enfin, nous conclurons par quelques perspectives de recherche.

8.1 Cadre des travaux et problématique

Nos travaux se positionnent :

- en conception routinière de produits et de procédés,
- dans une problématique d'aide à la conception interactive,
- s'appuyant sur des raisonnements à base de contraintes.

La mise au point de l'outil interactif d'aide à la conception à base de contraintes nécessite l'emploi de différents types de *CSPs* pour répondre à la diversité des connaissances mises en jeu. Nous assemblons donc différents types de *CSPs* pour construire les modèles de connaissances :

- les *CSPs* discrets liant des variables discrètes (symboliques ou numériques discrètes),
- les *CSPs* continus liant des variables continues,
- les *CSPs* mixtes liant des variables discrètes et continues,
- les *CSPs* dynamiques qui permettent d'ajuster un modèle de contraintes à un problème particulier.

Pour garantir l'interactivité de l'outil, différentes méthodes de filtrage de faible degré (arc-cohérence, 2B-cohérence, discrétisation) sont donc utilisées. Elles sont assemblées dans un seul moteur de propagation.

Le choix des méthodes de filtrage à utiliser est fonction de la classe, de la nature et du type de la contrainte courante. Pour cela, une typologie des contraintes est réalisée. Les contraintes peuvent être :

- de trois classes différentes :

    - discrètes : liant des variables discrètes (symbolique ou numérique discrète),
    - continues : liant des variables numériques continues,
    - mixtes : liant à la fois des variables discrètes et continues,
- de deux natures différentes :
    - compatibilité : permettant de définir les combinaisons de valeurs autorisées entre les variables.
    - activation : permettant d'ajuster un modèle de connaissance à un problème particulier par l'ajout ou le retrait d'un ensemble d'éléments (variables, contraintes, groupes).
- de trois types différents :
    - type table : permettant de lister toutes les combinaisons de valeurs autorisées entre variables sous forme tabulaire. Ces contraintes sont filtrées par arc-cohérence,
    - type fonction numérique « simple » : correspondant aux expressions mathématiques monotones et projetables. Ces contraintes sont filtrées par 2B-cohérence,
    - type fonction numérique « complexe » permettant de :
        - filtrer les contraintes numériques ne possédant pas forcément les propriétés de monotonie, de continuité et de projetabilité indispensables pour un filtrage par 2B-cohérence,
        - représenter et de prendre en compte dans les *CSPs* des abaques expérimentaux et des résultats d'expérimentations.

L'ensemble de ces travaux est sous-tendu par une problématique de mise au point d'un outil d'aide à la conception interactif d'opérations de traitement thermique.

8.2 Contributions

Notre contribution porte sur l'ensemble des types de contraintes utilisées.

Nous proposons, pour les contraintes de compatibilité de type table, une méthode de filtrage par arc-cohérence qui puise les valeurs à tester directement dans les tuples des contraintes. Cette méthode permet de filtrer des contraintes de compatibilité de type table des trois classes précédemment énoncées, sans convertir les valeurs des variables continues en labels discrets.

Pour les contraintes de compatibilité de type fonction numérique « simple », nous proposons d'étendre l'arithmétique des intervalles continue aux unions d'intervalles, pour prendre en compte les domaines muli-intervalles non continus, et d'adapter la méthode de filtrage par 2B-cohérence pour prendre en compte la combinatoire des domaines multi-intervalles. Pour éviter une explosion combinatoire, la surdéfinition des opérateurs multi-intervalles est contractante.

Nous proposons, pour les contraintes de compatibilité de type fonction numérique « complexe » définies par morceaux, deux méthodes de décomposition récursive et de mise sous forme d'arbre quaternaire. La première est dédiée aux contraintes d'égalité définies par morceaux, la seconde aux inégalités définies par morceaux. Ces deux méthodes nécessitent un certain nombre d'hypothèses sur la définition des contraintes et sur les degrés de discrétisation minimum. La

discrétisation des contraintes définies par morceaux d'égalité et d'inégalité repose sur l'identification de degrés d'information pertinente des zones de l'espace. Dans le cas d'inégalité, une étape supplémentaire de propagation, par voisinage, des zones cohérentes et incohérentes est nécessaire. La prise en compte simultanée de contraintes numériques « complexes » est réalisée par le mécanisme de fusion d'arbres quaternaires : ce mécanisme compare, de manière descendante, la cohérence d'arbres déjà discrétisés. Dans une problématique d'aide à la conception interactive où les contraintes ne sont discrétisées qu'à un moment précis du raisonnement, nous proposons le mécanisme de fusion opportuniste. La fusion opportuniste réalise la fusion d'arbres à partir d'un arbre de conjonction généré à partir de l'une des contraintes. La cohérence de chacune des autres contraintes n'est vérifiée que sur les zones cohérentes de l'arbre de conjonction qui est ainsi mis à jour.

L'ajustement d'un modèle de connaissances structuré à un problème particulier est réalisé *via* des contraintes d'activation. Pour garder la maîtrise des éléments (variables, contraintes, groupes) à prendre en compte dans les mécanismes de filtrage, nous proposons d'étendre les contraintes d'activation de variables à l'activation de contraintes et de groupes en associant à chaque élément deux états.

Les modèles de connaissances rassemblent divers types de contraintes. Le moteur de propagation de contraintes intègre donc toutes les méthodes de filtrage adaptées. Une maquette a été réalisée pour tester et valider les différentes méthodes. Un progiciel, reprenant son architecture générale, est actuellement en cours d'industrialisation.

8.3 Perspectives

Plusieurs pistes de travail se dégagent des travaux présentés dans ce mémoire.

Tout d'abord, la comparaison des performances de la méthode de filtrage proposée par rapport à celle existante pour les contraintes de compatibilité de type table doit être effectuée. Pour cela, un ensemble de problèmes faisant intervenir les trois classes de contraintes (discrètes, continues, mixtes) doit être identifié. Il serait, d'autre part, intéressant d'étendre cette méthode de filtrage à des degrés supérieurs à l'arc-cohérence.

Les abaques expérimentaux 2D sont intégrés aux *CSPs* par l'intermédiaire d'une structure de données arborescente, générée à partir de la définition syntaxique des contraintes. La méthode de génération de ces structures arborescentes a été étendue pour prendre en compte les abaques décrits comme des contraintes numériques binaires définies par morceaux. La prise en compte de résultats d'expérimentations à plusieurs dimensions (3D ou 4D) est réalisable si ceux-ci sont décrits par une seule contrainte continue. Il serait intéressant d'étendre les méthodes de génération proposées pour les contraintes binaires définies par morceaux aux contraintes définies par morceaux d'arité supérieure à 2.

Les modèles de connaissances sont structurés à partir de la notion de groupes d'éléments activables. Il serait intéressant d'intégrer à nos modèles et au moteur de propagation des groupes à occurrences multiples comprenant des contraintes entre occurrences.

Les solutions retenues se montrant performantes pour l'aide à la conception de procédés de traitement thermique, il serait intéressant de les appliquer à d'autres types de problèmes

d'ingénierie. L'aide à la conception dans un autre domaine d'ingénierie devra probablement nécessiter des contraintes numériques différentes de celles gérées actuellement, telles que des lois de mécanique du vol (Mulyanto, 2002), des calculs de résistance des matériaux (Fischer, 2000) ou des lois thermiques (Vernat, 2004). Le filtrage de ces contraintes nécessitera alors l'intégration d'autres méthodes de filtrage telles que la Box-cohérence. Ces méthodes enrichiront ainsi la maquette développée.

Bibliographie

Aldanondo, M., Vareilles, E., Hadj-Hamou, K., et Gaborit, P. (2005a). Aiding design with constraints : an extension of quad trees dealing with piecewise functions. In *International Conference on Industrial Engineering and Systems Management*, Marrakech, Maroc.

Aldanondo, M., Vareilles, E., Lamesle, P., Hadj-Hamou, K., et Gaborit, P. (2005b). Interactive configuration and evaluation of a heat treatment operation. In *Workshop on configuration, International Joint Conference on Artificial Intelligence*, Edinburgh, Écosse.

Aldanondo, M., Vareilles, E., Lamesle, P., Hadj-Hamou, K., et Gaborit, P. (2005c). Modélisation et exploitation des connaissances en design for x : une application en traitement thermique. In $6^{ième}$ *congrès International de Génie Industriel*, Besançon, France.

Barták, R. (1998). *On line guide to Constraint Programming*. http://kti.ms.mff.cuni.cz/~bartak/constraints/.

Beaumont, O. (1999). *Comment obtenir un résultat sûr quand les données sont incertaines*. Thèse de doctorat, Université de Rennes, France.

Benhamou, F. (1996). Heterogeneous constraint programming. In Verlag, S., editor, *5th international conference on algebraic and logic programming*, pages 62–76, Aachen, Allemagne.

Benhamou, F., Mc Allester, D., et Van Hentenryck, P. (1994). Clp(intervals) revisited. In *ILPS'94*, pages 1–21.

Bessière, C. et Cordier, M. (1993). Arc-consistency and arc-consistency again. In *AAAI*, pages 108–113.

Bessière, C., Meseguer, P., Freuder, E., et Larrosa, J. (1999). On forward checking for non-binary constraint satisfaction. In *Proceedings CP*, Alexandria VA.

Bessière, C. et Régin, J. (1994). An arc-consistency algorithm optimal in the number of constraint checks. In *European Conference on Artificial Intelligence, Workshop on Constraint Processing*, Amsterdam, Pays Bas.

Briggs, J. et Peat, D. (1991). *Un miroir turbulent - Guide illustré de la théorie du chaos*. Harper and Row publishers. traduit de l'américain par D. Stoquart.

Chandrasekaran, B. (1990). Design problem solving : a task analysis. In *Artificial Intelligence Magazine*, volume 11, pages 59–71.

Codognet, P. et Rossi, F. (2000). Solving and programming with soft constraints : Theory and practice. In *European Conference on Artificial Intelligence*, Berlin, Allemagne.

David, P., Veaux, M., Vareilles, E., et Maury, J. (2003). Virtual heat treatment tool for monitoring and optimising heat treatment process. In 2^{nd} *International Conference on Thermal Process Modelling and Computer Simulation*, Nancy.

Davis, E. (1987). Constraint propagation with intervals labels. *Artificial Intelligence*, 32(3) : pages 281–331.

Dechter, A. et Dechter, R. (August 1988). Belief maintenance in dynamic constraints networks. In *Seventh National Conference on Artificial Intelligence*, pages 37–42, St Paul, USA.

Delobel, F. (2000). *Résolution de systèmes de contraintes réelles non linéaires.* Thèse de doctorat, Université Sophia Antipolis, France.

Faltings, B. (1994). Arc consistency for continuous variables. In *Artificial Intelligence*, volume 65, pages 363–376.

Finkel, R. et Bentley, J. (1974). Quadtrees : a data structure for retrieval on composite keys. In *Acta Infomatica*, pages 1–9.

Fischer, X. (2000). *Statégie de conduite du calcul pour l'aide à la décision en conception mécanique intégrée : application aux appareils à pression.* Thèse de doctorat, École Nationale Supérieure d'Arts et Métiers, France.

Freuder, E. (1978). Synthesizing constraint expressions. In *Communications of the ACM*, volume 21(11), pages 958–966.

Gelle, E. (1998). *On the generation of locally consistent solution spaces in mixed dynamic constraint problems.* Thèse de doctorat, École Polytechnique Fédérale de Lausanne, Suisse.

Gelle, E. et Weigel, R. (1995). Interactive configuration based on incremental constraint satisfaction. In *IFIP*, pages 117–126.

Geneste, L. et Ruet, M. (2001). Experience based configuration. In *International Joint Conference on Artificial Intelligence Workshop on Configuration*, Seattle, USA.

Glover, F. et Laguna, M. (1993). Tabou search. In *Modern Heuristic Techniques for Combinatorial Problems*, Oxford, England.

Golumb, S. et Baumbert, L. (1965). Backtrack programming. In *Journal of ACM*, volume 12, pages 516–524.

Hadj-Hamou, K. (2002). *Contribution à la conception de produits à forte diversité et de leur chaîne logistique : une approche par contraintes.* Thèse de doctorat, Institut National Polytechnique de Toulouse, France.

Hansen, E. (1992). Global optimization using intervals analysis. In *Marcel Dekker*, New York, USA.

Haralick, R. et Elliot, G. (1980). Increasing tree search efficiency for constraint satisfaction problem. In *Articial Intelligence*, volume 14, pages 263–313.

Haselböck, A. (1993). *Knowledge-based Configuration and Advanced Constraint Technologies*. Thèse de doctorat, Université technologique de Vienne.

Hyvönen, E. (1989). Constraint reasoning based on interval arithmetic. In *International Joint Conference on Artificial Intelligence*, pages 1193–1198, Detroit, USA.

IEEE754 (1985). *ANSI/IEEE. IEEE standard for binary floating point arithmetic*.

Kearfott, R. (1996). *Rigorous Global Search : Continuous Problems*. Kluwer Academic Publisher.

Kirkpatrick, S., Gelatt, C., et Vecchi, M. (1983). Optimization by simulated annealing. In *Science*, volume 200, pages 671–680.

Kolodner, J. (1993). Case-based reasoning. In *Morgan Kaufmann Publishet*.

Le petit Robert (1996). *Dictionnaire de la langue française*.

Lhomme, O. (1993). Consistency techniques for numeric CSP. In *International Joint Conference on Artificial Intelligence*, pages 232–238, Chambéry, France.

Lhomme, O. et Rueher, M. (1997). Application des techniques CSP au raisonnement sur les intervalles. *Revue d'intelligence artificielle*, 11/3 : pages 283–311.

Lobjois, L. et Lemaitre, M. (1997). Coopération entre méthodes complètes et incomplètes pour la résolution de (V)CSP : une tentative d'inventaire. In *Journées Nationales de la Résolution Pratique de Problèmes NP-complets*, pages 67–73, Rennes, France.

Lottaz, C. (2000). *Collaborative Design using Solution Spaces*. Thèse de doctorat, École Polytechnique Fédérale de Lausanne, Suisse.

Mackworth, A. (1977). Consistency in networks of relations. In *Artificial Intelligence*, volume 8(1), pages 99–118.

Mackworth, A. et Freuder, E. (1985). The complexity of some polynomial network-consistency algorithms for constraint satisfaction problems. In *Artificial Intelligence*, volume 25, pages 65–74.

Mittal, S. et Falkenhainer, B. (1990). Dynamic constraint satisfaction problems. In *AAAI*, pages 25–32, Boston, US.

Montanari, U. (1974). Networks of constraints : fundamental properties and application to picture processing. In *Information sciences*, volume 7, pages 95–132.

Moore, R. (1966). *Interval Analysis*. Prentice-Hall.

Mulyanto, T. (2002). *Utilisation des techniques de programmation par contraintes pour la conception d'avions*. Thèse de doctorat, École Nationale Supérieure de l'Aéronautique et de l'Espace, Toulouse, France.

Oliveira, M., Moreira, A., Loureiro, A., Denis, S., et Simon, A. (1986). Effet du mode de refroidissement sur les déformations produites par la trempe martensitique. In *5^e congrès international sur les traitements thermiques des matériaux*, pages 1814–1821, Budapest, Hongrie.

Pahl, G. et Beitz, W. (1996). *Engineering Design : a Systematic Approach*. Springer-Verlag.

Rossi, F., Petrie, C., et Dhar, V. (1990). On the equivalence of constraint satisfaction problems. In *European Conference on Artificial Intelligence*, pages 550–556, Stockholm, Sweden.

Rueher, M. (2002). *Contraintes sur les domaines continus*. Notes de cours.

Ruet, M. (2002). *Capitalisation et réutilisation d'expériences dans un contexte multiacteur*. Thèse de doctorat, Institut National Polytechnique de Toulouse, France.

Sabin, D. et Freuder, E. (1996). Configuration as composite constraint satisfaction. In *Artificial Intelligence and Manufacturing Research Planning Workshop*, pages 153–161.

Sabin, M. et Freuder, E. (1999). Detecting and resolving inconsistency and redundancy in conditional constraint satisfaction problems. In *American Association for Artificial Intelligence*.

Sam, D. (1995). *Constraint Consistency Techniques for Continuous Domains*. Thèse de doctorat, École Polytechnique Fédérale de Lausanne, Suisse.

Samet, H. (1984). The quadtree and related hierarchical structures. In *Computing surveys*, volume 16, pages 187–260.

Saucier, A. (1997). *Un modèle multi-vues du produit pour le développement et l'utilisation de systèmes d'aide à la conception en ingénierie mécanique*. Thèse de doctorat, École Normale Supérieure de Cachan, France.

Soininen, T. et Gelle, E. (1999). Dynamic constraint satisfaction in configuration. In *American Association for Artificial Intelligence, Workshop on Configuration*, Orlando, US.

Vareilles, E., Aldanondo, M., Hadj-Hamou, K., Gaborit, P., Lamesle, P., et Maury, J. (2003). Modélisation des connaissances et aide à la décision : une application en traitement thermique. In $5^{ième}$ *Congrès International de Génie Industriel*, Québec, Canada.

Vareilles, E., Hadj-Hamou, K., Aldanondo, M., et Gaborit, P. (2005). Extension des quadtrees pour la représentation et le filtrage des contraintes numériques définies par morceaux. In *Journées Francophones de Programmation par Contraintes*, Lens, France.

Vargas, C. (1995). *Modélisation du processus de conception en ingeniérie des systèmes mécaniques. Mise en œuvre basée sur la propagation de contraintes. Application à la conception d'une culasse automobile*. Thèse de doctorat, École Normale Supérieure de Cachan, France.

Veaux, M. (2001). *Prévision des cinétiques de transformation de phases, des contraintes et des déformations lors de la transformation bainitique*. Thèse de doctorat, École des Mines de Nancy, France.

Verfaillie, G. et Schiex, T. (1995). Maintien de solution dans les problèmes dynamiques de satisfaction de contraintes : bilan de quelques approches. *Intelligence Artificielle*, 9(3) : pages 269–309.

Vernat, Y. (2004). *Formalisation et qualification de modèles par contraintes en conception préliminaire*. Thèse de doctorat, École Nationale des Arts et Métier, Bordeaux, France.

Veron, M. (2001). *Modélisation et résolution du problème de configuration industrielle : utilisation des techniques de satisfaction de contraintes*. Thèse de doctorat, Institut National Polytechnique de Toulouse, France.

Veron, M. et Aldanondo, M. (2000). Yet another approach to CCSP for configuration problem. In *European Conference on Artificial Intelligence, Workshop on configuration*, pages 59–62, Berlin, Allemagne.

Waltz, D. (1975). Understanding line drawings of scenes with shadows. In Winston, P., editor, *The psychology of Computer Vision*, pages 19–91.

Yannou, B. (1998). Les apports de la programmation par contraintes en conception. In Hermés, editor, *Conception de produits mécaniques méthodes, modèles et outils*, pages 457–486.

Annexes

Exemple de pièce trempée

Nous présentons le cas d'axe trempé conduisant aux intensités de déformation finale des composantes « bobine-tonneau » et « banane » présentées en 7.2.1 page 134. Les valeurs des variables à saisir dans la maquette *WEB*, à l'adresse http://iena.enstimac.fr:20000/cgi-bin/vht.pl, pour aboutir aux mêmes résultats sont fournies.

Géométrie de la pièce

La pièce trempée, illustrée par la figure 1, ne possède pas de trou débouchant, ni de méplat, ni d'épaulement significatif. Elle possède plus de trois plans de symétrie. Son diamètre est de 30 mm et elle a une variation d'épaisseur supérieure à 1.5.

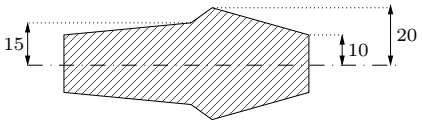


FIG. 1 – Exemple de pièce trempée

La géométrie de la pièce est donc décrite de la façon suivante :

- pas de présence de trou à l'extrémité gauche : *Hole at the left extremity*= {"no"}
- pas de présence de trou à l'extrémité droite : *Hole at the right extremity*= {"no"}
- présence d'un méplat : *Main dissymetry*= {"no"}
- épaulement significatif : *Significant shoulder*= {"none"}
- nombre de plans de symétrie : *Case of symetry*= {"case4"}
- diamètre de la pièce : *Diameter*= {30}
- variation d'épaisseur : *Thickness variation*= {">1.5"}

Fluide de trempe

La pièce est trempée dans de l'eau à 40°C. Le fluide de trempe a une vitesse de 0.8 mètre seconde, sa température s'élève au maximum de 20°C. Le fluide arrive parallèlement à l'axe de la pièce de manière orientée. Son intensité de trempe est inférieure à 10 HRC et son taux de pénétration est inférieur à 5 HRC.

La description du fluide de trempe est donc la suivante :

- un type de fluide : *Quenching Fluid*= {"water"}

– une température du fluide : *Temperature of quenching fluid*= {40}
– une vitesse du fluide : *Speed of the quenching fluid*= {"> 0.8"}
– une température d'élévation maximale du fluide : *Temperature of elevation of quenching fluid*= {"< 20"}
– une direction du fluide de trempe : *Quenching Fluid Direction*= {"parallel to axis"}
– une agitation du fluide : *Fluid agitation*= {"oriented"}
– une intensité de trempe : *Quenching intensity in HRC*= {"< 10"}
– une pénétration de trempe : *Quenching penetration in HRC*= {"< 5"}

Préparation de la charge

Les pièces trempées sont calées perpendiculairement à la gravité, telle que l'illustre la figure 2. Le taux de perméabilité du support est inférieure à 70 %. La distance entre les pièces est inférieure à 5 mm. Le poids des pièces est inférieur à la capacité maximale du bac de trempe. La symétrie du bac est compatible avec les pièces trempées.

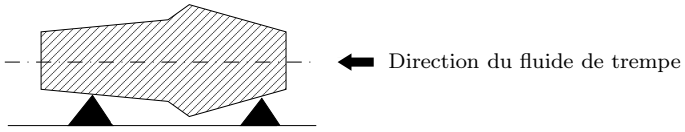


FIG. 2 – Exemple de calage de pièce trempée

La préparation de la charge est donc la suivante :
– direction de la gravité : *Gravity*= {"perpendicular to axis and good chock"}
– perméabilité du panier : *Basket permeability*= {"< 70 %"}
– distance entre les pièces trempées : *Distance between parts*= {"< 5mm"}
– ratio poids des pièces trempées / poids maximum admis par le bac de trempe : *Weight ratio*= {"< 70 %"}
– symétrie du bac de trempe par rapport aux pièces trempées : *Symetry aroud the part*= {"rather good"}

Matériau et historique

Le matériau se caractérise par :
– son nom : *Material*= {"42CrMo4"}
– son pourcentage de carbone : *Carbon content*= {"> 0.1 and < 0.2"}
– une structure en bande : *Banded structure*= {"yes"}
– sa structure avant traitement : *Structure before heat treatment*= {"Unsoft steel"}
– sa variation de rugosité en surface : *Roughness variation*= {"< 2.4"}
– son procédé d'obtention du brut : *Forming process*= {"casting"}
– une absence de relaxation avant traitement : *Relaxation before heat treatment*= {"no"}
– le non respect de l'axe de l'acier brut : *Raw part axis*= {"different from structure axis"}
– une profondeur de décarburation superficielle : *Decarburized depth* = " $< 0.3mm$"

– une dissymétrie de décarburation superficielle : *Symetry of decarburized area* = "*no*"

Caractéristiques métallurgiques et cycle de chauffe

Les caractéristiques métallurgiques et le cycle de chauffe sont caractérisés par :

– une absence de relaxation après traitement : *Relaxation after forming and before heat treatment*= {"no"}

– une atmosphère entre le four de chauffe et le bac de trempe : *Atmosphere*= {"Neutral"}

– une différence de température entre la sortie du four et θ_1 : *Furnace temperature minus teta 1*= {"> 300"}

– une différence entre Ms et θ_2 : *Teta2 minus Ms*= {"> -250 and < -50"}

– un type de traitement thermique de trempe avec ou sans cémentation : *Heat treatment type*= {"Quenching"}

– une absence de préchauffage de la pièce : *Pre heating of the load*= {"no"}

– une classe de four de chauffe : *Furnace class* = {"3 and 5"}

– la température du four avant chargement : *Furnace temperature* = {"> Ac1 and < Ac3"}

– une vitesse de chauffe : *Heating rate* = {"> 20°/mm"}

– des paliers durant le chauffage : *Step during heating* = {"Temp step > Ac1-50 and < Ac1"}

Valorisation des compétences, un nouveau chapitre de thèse

Ce chapitre présente une synthèse des compétences non scientifiques acquises durant les trois années de thèse. Cette synthèse a été réalisée dans le cadre de l'exercice « Valorisation des compétences, un nouveau chapitre de thèse » proposé par l'association Bernard Gregory. Le travail proposé a été validé par le mentor qui m'était assigné, M. Alexandre MARTINEZ que je remercie pour ses précieux conseils.

1 - Cadre général et enjeux de ma thèse

Mes travaux de recherche se déroulent au centre Génie Industriel de l'École des Mines d'Albi-Carmaux, sous la direction de Michel Aldanondo. Le mémoire résumant ces trois années de recherche s'intitule « Conception et approches par propagation de contraintes : mise œuvre d'un outil d'aide interactif ». Ce titre indique ma participation au développement d'un outil interactif d'aide à la conception basé sur une technique d'Intelligence Artificielle (Intelligence Artificielle) : les problèmes de satisfaction de contraintes (*CSP*). Ce travail est supporté par une problématique industrielle regroupant plusieurs partenaires européens et internationaux.

Le monde des entreprises fait actuellement face à un important problème de gestion de ses connaissances (savoir-faire techniques et compétences technologiques). Ce problème se caractérise de plusieurs manières :

- Perte de savoir et de savoir-faire antérieur (turn-over, départ à le retraite),
- Méconnaissance des travaux effectués ailleurs,
- Localisation monopolistique des connaissances et de l'expertise.

Les problèmes de satisfaction de contraintes ont été introduits en 1974 par Montanari (1974) pour capitaliser, formaliser et exploiter ces connaissances. L'exploitation des connaissances est primordiale dans un grand nombre de domaines industriels (aéronautiques, automobiles, etc) pour éviter, par exemple, de commettre à nouveaux les mêmes erreurs que dans le passé ou pour concevoir « plus rapidement » de nouveaux produits sur la base de produits déjà existants (seulement 5 années pour concevoir l'A380 pour Airbus en utilisant son savoir-faire et ses compétences en conception avionique). Les connaissances des entreprises peuvent être de différentes formes (lois physiques, plans, retour d'expériences, etc) : les *CSPs* ont donc évolué pour modéliser et exploiter ces types de connaissances. Mes travaux portent sur l'intégration de différents *CSPs* dans un seul outil de capitalisation de connaissances afin d'aider à la conception de cycles de traitement thermique pour des équipementiers automobiles et des métallurgistes.

Ma thèse s'inscrit dans la continuité de travaux déjà menés au sein de mon équipe de recherche. Je bénéficie donc de l'expérience acquise par l'équipe dans ce domaine. Pour des besoins très particuliers, j'ai dû faire appel à des compétences extérieures au laboratoire. J'ai ainsi contacté d'autres chercheurs du domaine (français ou étrangers) pour trouver de la bibliographie, pour obtenir des réponses à mes questions ou pour avoir un point de vue extérieur. Pour acquérir ces compétences et les retransmettre à mon équipe, tous les moyens financiers (déplacements en France et à l'étranger) et technologiques (courrier, mail, vidéoconférence) ont été mis à ma disposition.

Mon choix de sujet de thèse est surtout dû à mon expérience de DEA. Après mon stage de DEA très fondamental, je me suis rendu compte que la recherche purement théorique ne me correspondait pas. Ayant fait un stage de 5 mois en industrie qui s'était très bien passé sur

les plans professionnels et personnels, j'ai cherché un sujet plutôt appliqué à un problème industriel. Le sujet de thèse proposé correspondait exactement à mes attentes. Le projet se découpait clairement en deux parties : l'une plutôt industrielle (développement d'un outil informatique), l'autre plutôt académique (recherche et adaptation des techniques d'Intelligence Artificielle à utiliser pour réaliser cet outil). Après avoir déposé mon dossier de candidature et avoir passé l'entretien de recrutement, ma thèse a commencé au mois de septembre 2002.

2 - Déroulement, gestion et coût de mon projet

Mon sujet de thèse se découpant en deux parties, j'ai sollicité un grand nombre de personnes à la fois dans le milieu industriel et académique. Le projet regroupe 18 entreprises réparties sur 3 continents telles que Scania (Suède), Nissan Motor (RCP) Japon, Nitrex (Canada). Chaque entreprise a mobilisé en moyenne 3 personnes. J'ai donc travaillé avec environ 50 personnes du milieu industriel. Au niveau académique, il faut compter les membres de mon laboratoire (environ 25 personnes) et les personnes extérieures que j'ai sollicitées (environ 75 personnes), soit 100 personnes en milieu académique. Le domaine d'application de mon sujet m'était totalement inconnu. J'ai donc demandé de l'aide bibliographique à un grand nombre de personnes autant en milieu industriel qu'académique, français ou étrangers.

2.1 - Bilan financier :

Ce projet est financé par l'Europe. Le montant alloué à l'École des Mines d'Albi-Carmaux se monte à 100 000 euros par an. Cette somme a servi à couvrir mon salaire, les déplacements liées au projet et aux conférences ainsi que les équipements rattachés au projet. Pour être un peu plus précise, le tableau 1 présente la totalité des sommes engagées :

Tab. 1 – Bilan financier

Frais de personnel	Coût sur 3 ans	Coût total
1 Salaire d'ingénieur CFR	32 000 E chargés par an * 3	96000 E
1 Secrétariat	1 jour par semaine (57 E) * 52 * 3	8892 E
3 Encadrants	2 jours par semaine (420 E) * 52 * 3	65520 E
		170412 E
Déplacements	5000 E * 3	15000 E
Équipement	50000 E amortis sur 5 ans	30000 E
Frais infrastructure		
Téléphone	45 h * 3 * 1 E	135 E
Bureau	100 E * 12* 3	3600 E
Ordinateur	1000 E amortis sur 3 ans	1000 E
		4735 E
Coût total :		220147 E

2.2 - Montage du projet :

La phase de montage de ce projet de thèse a débuté avant mon recrutement, dans le courant de l'année 2001. Ce sont plusieurs personnes de mon laboratoire avec une petite partie des partenaires industriels qui ont répondu à l'appel d'offre du projet européen. Le dossier a été accepté et la thèse lancée.

Ce projet étant un projet européen, les travaux effectués ne peuvent être confidentiels. Le but du projet est certes la mise au point d'un outil informatique d'aide à la conception mais surtout le transfert de connaissances du milieu académique vers l'industrie.

2.3 - Jalons du projet :

Un calendrier de travail a été réalisé par le coordinateur du projet (SCC, France) sur les trois années. Les jalons du projet sont les suivants :

- Septembre 2002 : (Lyon, France)
  - lancement du projet
- Septembre 2003 : (Stockholm, Suède)
  - spécification de l'outil d'aide interactif, développement de la maquette
  - premiers modèles de connaissances réalisés avec plusieurs industriels dont EMTT, SCC (France), Metallorgraphica (Espagne)
  - collecte des cas industriels et extraction de connaissances (nombreuses réunions à Montpellier, à Lyon et sur Albi)
- Février 2004 : (Turin, Italie)
  - validation de la maquette par certains des partenaires : Scania (Suède), CRF (Italie)
  - validation des modèles de connaissances par certains des partenaires : École des Mines de Nancy (France), IWT (Allemagne)
  - collecte des cas industriels et extraction des connaissances (nombreuses réunions à Montpellier, à Lyon et sur Albi)
  - validation du travail effectué et restant à faire par le responsable du projet au niveau européen
- Septembre 2004 : (Breme, Allemagne)
  - première version du logiciel industriel : SCC (France)
  - collecte des cas industriels et extraction des connaissances (nombreuses réunions à Montpellier, à Lyon et sur Albi) : SCC, EMTT, École des Mines d'Albi-Carmaux (France), Metallographica (Espagne)
- Février 2005 : (Albi, France et Kyoto, Japon)
  - validation du logiciel industriel : Scania (Suède), CRF (Italie)
  - collecte des cas industriels et extraction des connaissances (nombreuses réunions à Montpellier, à Lyon et sur Albi) SCC, EMTT, École des Mines d'Albi-Carmaux (France), Metallographica (Espagne)
- Août 2005 : (Creusot, France)
  - validation des résultats du projet par le responsable du projet au niveau européen, fin du projet

Le projet n'a pas trop dérivé en termes de délais et le retard pris sur certaines tâches a pu être finalement rattrapé. Mes travaux de thèse ont consisté à définir les spécifications et à développer la maquette du logiciel industriel. J'ai donc travaillé à plein temps durant 18 mois sur le projet industriel, 12 mois sur la partie scientifique et 6 mois sur la rédaction et sur la préparation de la soutenance de la thèse.

2.4 - Planification :

La conduite du projet s'est déroulée à partir de réunions bimensuelles avec mon directeur de thèse et mes deux tuteurs. Chaque réunion a fait l'objet d'un compte rendu (rédigé généralement par mon directeur de thèse et validé par tous) qui nous permettait de réaliser un suivi précis de chaque tâche. Chacune de ces réunions débutait par la description de l'ordre du jour (conférence, articles à rédiger, travail de thèse), puis chaque point était passé en revue en termes de travail accompli et de travail restant à faire. Pour chacun des points importants, une discussion bilatérale était engagée. Ces réunions se concluaient par le recensement des tâches à réaliser, par l'établissement d'un planning, par l'affectation des tâches aux personnes concernées (encadrants et doctorant) et par la planification de la prochaine réunion. Tous les points scientifiques de ma thèse ont ainsi été débattus et résultent d'un travail d'équipe. De plus, nous avons eu des réunions de bilan d'avancement de ma thèse tous les six mois qui ont permis de faire le point sur le travail réalisé et de recadrer le travail restant à faire.

Lorsque des problèmes que je n'arrivais pas à résoudre seule sont apparus, j'ai sollicité mes encadrants qui ont toujours répondu présent : soit, nous pouvions discuter de mon problème sur-le-champ (souvent, devant un bon café), soit nous planifions une réunion de travail en petit comité.

L'un des plus gros problèmes auquel j'ai été confrontée, peut se résumer par le fait que l'une des solutions que nous proposions était partiellement erronée. Nous nous en sommes aperçu quelques jours avant la présentation de nos travaux de recherche à l'ensemble du laboratoire. Nous avons donc décidé de soulever ce problème lors de la présentation pour essayer d'y trouver une solution. Celle-ci m'est apparue, en partie, en discutant avec d'autres thésards du laboratoire et surtout avec un professeur agrégé de mathématiques extérieur à l'École des Mines d'Albi-Carmaux. Cette solution a été discutée avec les membres de l'équipe, validée puis intégrée dans la maquette.

3 - Compétences, savoir-faire, qualités professionnelles et personnelles illustrées par des exemples

3.1 - Informatique :

Durant ces trois années, j'ai acquis des compétences et de l'expérience sur les plans professionnels et personnels. Sur le plan technique, j'ai approfondi mes connaissances dans la technique d'Intelligence Artificielle utilisée (développement d'une maquette à base des techniques de filtrage des contraintes), j'ai appris un nouveau langage de prototypage rapide (*Perl*), à développer un site web et à utiliser LaTeXpour rédiger ma thèse.

3.2 - Langue anglaise :

Sur le plan linguistique, le fait que le projet soit un projet européen impliquant des partenaires de plusieurs nationalités (allemands, suédois, espagnols, italiens), m'a permis d'améliorer mon anglais technique mais surtout courant. Lors des premières réunions, je n'ai pas trop eu de mal à suivre les conversations professionnelles. Mais lorsque les repas arrivaient et que tout le monde parlait de sa vie personnelle (voyages, famille), j'ai eu beaucoup de mal à m'intégrer dans la conversation par manque de vocabulaire et de pratique. J'ai donc décidé, à ce moment là, de prendre des cours particuliers d'anglais. Je commence depuis à avoir de l'assurance, à expliquer mon travail et à converser avec nos partenaires lors des repas.

3.3 - Enseignement :

Sur le plan pédagogique, j'ai aussi beaucoup appris. J'avais tendance, lors des premières séances de TDs ou de TPs, à un peu trop "materner" les élèves. Je faisais systématiquement des rappels approfondis du cours et perdais beaucoup de temps sur les horaires établis. J'arrive maintenant à rappeler plus synthétiquement les connaissances indispensables à la réussite des TDs ou des TPs et à mieux gérer les délais. Je reste toujours à la disposition des élèves pour répondre à leurs questions durant ou non les heures encadrées.

3.4 - Relationnel :

Sur le plan relationnel, j'ai toujours entretenu de bonnes relations avec les membres du laboratoire. Lors de mes déplacements ou d'événements particuliers, je ramène toujours un petit quelque chose pour l'ensemble du laboratoire : spécialités régionales, crêpes, chocolat ... Cette convivialité permet de discuter plus facilement avec les personnes, d'échanger des idées de manière informelle et aussi de discuter des problèmes rencontrés par chacun lors de cette petite pause culinaire. Je pense qu'il est très important d'avoir de bonnes relations avec ses collègues pour favoriser le travail de chacun.

3.5 - Persévérance :

Durant ces trois années de thèse, j'ai su utiliser plusieurs qualités personnelles. Je suis plutôt persévérante et je ne me laisse pas facilement décourager par un problème dont la solution ne semble pas évidente. Lorsque nous nous sommes aperçus que l'une de nos propositions était erronée, j'ai sollicité plusieurs personnes internes ou extérieures au laboratoire pour essayer de trouver la réponse à ce problème. Plusieurs pistes ont ainsi été exploitées avant de trouver la bonne solution.

3.6 - Capacité d'organisation :

Ayant des capacités d'organisation et de synthèse, j'arrive à gérer plusieurs projets en parallèle (déplacements, cours, réunions de travail) et à accéder rapidement aux informations dont j'ai besoin. Je suis rarement en retard sur l'un des projets en cours. Par exemple, ma période de rédaction de thèse a été planifiée très finement : chacun des chapitres avait une date de rendu bien précise et tout devait être terminé initialement pour le 13 mai. J'avais durant cette période des enseignements à donner, que je devais préparer, et des réunions de travail avec les industriels du projet. Bien que la date de rendu du manuscrit final ait été avancée au 4 mai, j'ai su conduire ces différentes tâches dans les temps sans trop empiéter sur mes soirées et mes week-ends. J'ai su réutiliser les synthèses réalisées au cours des trois années pour écrire mon mémoire et préparer les réunions de travail. Ces capacités me permettent d'être réactive face à des nouveaux problèmes. Par exemple, lorsque les industriels ont validé notre maquette, plusieurs erreurs ont été détectées. J'ai pu dégager rapidement (généralement dans la journée) du temps sur mon emploi du temps pour corriger les erreurs et en informer les industriels.

3.7 - Ouverture d'esprit :

Étant curieuse et perfectionniste, j'ai acquis de solides compétences dans des domaines qui m'étaient auparavant inconnus. L'application industrielle du projet portait sur un domaine scientifique dont je n'avais eu que de légers aperçus durant mon parcours scolaire (métallurgie).

Je me suis donc investi pour combler ces lacunes en lisant des ouvrages, des articles et en posant des questions aux personnes compétentes. Cette mise à niveau m'a permis de parler le même langage que les industriels du projet européen et de mieux comprendre les difficultés technologiques du problème à résoudre.

3.8 - Développement du réseau :

Le fait d'aller vers les autres pour résoudre mes problèmes ou pour confronter mes idées m'a permis d'élargir mon réseau industriel (partenaires du projet) et académique (École des Mines de Nantes, École Polytechnique Fédérale de Lausanne), français (Sofia Antipolis, Nantes, Nancy) et étranger (Suisse, Allemand, Américain). Pour mener à bien ma thèse, j'avais besoin de connaître les bases d'un autre domaine scientifique (métallurgie). J'ai donc, d'une part, fait appel à mes connaissances industrielles et académiques, et d'autres part, à des personnes que je ne connaissais pas jusque là (certains partenaires du projet). J'ai pratiquement toujours eu des réponses à mes questions et à mes demandes. Lorsque aucune réponse n'arrivait au bout d'une à deux semaines, je relançais, dans un premier temps les personnes concernées et dans un second temps, je cherchais une autre personne compétente à contacter.

4 - Résultats et impact de la thèse

Les résultats de ma thèse sont très positifs. D'un point de vue projet européen, nous avons tenu nos objectifs : l'outil d'aide à la conception a vu le jour et les industriels en sont très satisfaits en termes de temps de réponse et de résultats fournis.

D'un point de vue recherche, mes travaux de thèse ont fait l'objet de quatorze publications dont sept avec actes édités (Industrial Engineering and Systems Management, IESM'05, International conference on Distortion Engineering, IDE'05) dont certaines n'étaient pas dans les habitudes de l'équipe. J'ai eu la chance de pouvoir les présenter dans plusieurs congrès et ainsi d'échanger des idées avec d'autres chercheurs. Notre maquette, développée dans le cadre du projet européen, est réutilisée dans le laboratoire et sert maintenant de base à d'autres projets (thèse de T. van Oudenhove de Saint Géry).

D'un point de vue professionnel et personnel, ma thèse m'a énormément apporté. Elle m'a conforté dans mon souhait de devenir enseignant-chercheur soit dans le public, soit dans le privé (il est en effet possible de donner des vacations lorsque l'on est chercheur dans l'industrie). J'aime, d'une part, le contact avec les élèves, transmettre mon savoir et répondre à leurs questions et d'autre part, être confrontée à des problèmes concrets. La résolution de ce type de problèmes est un travail passionnant qui permet d'acquérir de nouvelles compétences par la recherche de documentations, de travaux déjà existants et par la rencontre avec de nouvelles personnes d'horizons différents (génie industriel, matériaux, Intelligence Artificielle).

5 - Principaux enseignements

Le choix de faire une thèse doit être l'aboutissement d'une réflexion personnelle. Je me suis rendue compte en discutant avec d'autres thésards, et notamment lors des doctoriales de l'INPT, que beaucoup faisaient une thèse sans vraiment réfléchir parce que la thèse apparaissait comme une suite logique à leurs études. Être en thèse constitue, en général, le premier pas dans la vie professionnelle. Personnellement, j'ai signé mon premier contrat de travail lors de ma thèse. Par conséquent, je pense que la thèse se rapproche plus d'un travail professionnel

que des études. Certes, les doctorants sont encore étudiants, mais ce sont aussi et surtout des professionnels. Je pense que cet aspect professionnel est souvent oublié par les doctorants qui ne sont pas directement plongés dans un milieu industriel.

Après avoir fait le choix de faire une thèse, il faut se consacrer au sujet, à l'équipe et au laboratoire. Le choix du sujet de thèse est le plus important. La durée d'une thèse est de trois ans et cela peut paraître vraiment long lorsque le sujet n'est pas adapté au doctorant. Le sujet doit être suffisamment précis pour éviter de changer en cours de thèse. Si le sujet enthousiasme le futur doctorant, alors il sera plus à même d'argumenter son souhait de faire cette thèse lors du jury de recrutement.

Lorsque le sujet est choisi, il faut se concentrer sur l'équipe et le laboratoire. J'ai beaucoup de chance de faire ma thèse à l'École des Mines d'Albi-Carmaux car le cadre est véritablement propice au travail :

- Petit laboratoire,
- Petite équipe de recherche,
- Bonne ambiance,
- Directeur de thèse disponible,
- Moyens matériels et financiers.

Plusieurs de mes amis sont en thèse dans de gros laboratoires. Ils ne croisent leurs encadrants que de temps en temps, ceux-ci ayant un grand nombre de thésards à encadrer. Ils font leur recherche tout seuls et tous les six mois, font un rapport d'avancement qui est soumis à leurs encadrants. Ce type de fonctionnement est à double tranchant : soit le doctorant est autonome, a de très bonnes idées et peut conduire sa thèse seul (ce qui est rarement le cas), soit le doctorant se sent perdu, a des difficultés à faire avancer son travail de recherche et ne peut généralement pas parler de ses difficultés à ses encadrants. Il est parfois difficile de se faire une idée de l'ambiance du laboratoire lors de l'entretien. Je pense qu'il ne faut pas hésiter à prendre contact avec d'autres thésards pour prendre la température du laboratoire. Je pense que la réussite d'une thèse est basée sur l'adéquation du sujet avec l'environnement et le doctorant.

J'ai commencé à préparer mon après thèse véritablement au bout de deux années. Je me suis, alors, renseignée sur les différentes possibilités professionnelles après l'obtention du grade de docteur pour devenir enseignant-chercheur dans le public. Je n'ai vraiment préparé mon après thèse que lorsque mon manuscrit a été terminé (c'est-à-dire, il n'y pas longtemps) : j'ai postulé sur un poste de maître-assistant dans mon équipe de recherche et sur plusieurs postes d'attaché temporaire d'enseignement et de recherche (ATER). Je n'ai pas encore regardé les possibilités professionnelles dans le milieu industriel. Mais ce type de débouché me paraît encore assez flou et va me demander un travail de réflexion pour arriver à cibler les industries qui pourraient être intéressées par mes compétences (éditeur de logiciel, par exemple).

Si c'était à refaire, je recommencerai bien volontiers. Mon travail de thèse m'a vraiment passionné et s'est déroulé dans de très bonnes conditions autant matérielles que relationnelles. Il m'a beaucoup apporté sur les plans personnel et professionnel. J'ai beaucoup changé car j'ai eu l'occasion d'échanger mes idées avec d'autres personnes d'environnement différent, de culture différente et d'ouvrir mon esprit à d'autres domaines scientifiques.

Conception et approches par propagation de contraintes : contribution à la mise en œuvre d'un outil d'aide interactif

Les travaux présentés dans cette thèse s'inscrivent dans une problématique d'aide à la conception interactive de procédés. L'outil d'aide à la conception réalisé repose sur un modèle de connaissances décrit comme un problème de satisfaction de contraintes (*CSP*). La recherche de solutions étant interactive, cet outil exploite les méthodes de filtrage des *CSPs*. La diversité des connaissances à exploiter nous conduit à intégrer différents types de *CSPs* (discrets, continus, mixtes, dynamiques) ainsi que leurs méthodes de filtrage (arc-cohérence, 2B-cohérence, discrétisation). Une problématique de conception d'opérations de traitement thermique constitue le support industriel de ces travaux.

La première partie de la thèse positionne nos travaux et expose la problèmatique industrielle. Dans la deuxième partie, une typologie des contraintes indispensables à la construction des modèles de connaissances est réalisée. Cette typologie permet d'identifier les différents types *CSPs* à utiliser. Chacun d'entre eux est présenté en termes d'état de l'art, de besoins rencontrés et de solutions apportées.

La prise en compte d'abaques expérimentaux 2D dans les modèles à base de contraintes nécessite l'utilisation d'une structure de données nommée arbre quaternaire. L'intégration de cette structure de données ne pose pas de problème particulier pour les abaques décrits par des contraintes numériques continues. Nous l'étendons pour prendre en compte les abaques décrits par des contraintes numériques définies par morceaux.

Le dernier chapitre présente l'architecture générale d'un modèle de connaissances réalisé avec les experts en traitement thermique et le fonctionnement général du moteur de propagation.

Mots-clés : conception, contraintes, filtrage, arbre quaternaire, traitement thermique

Constraint-based design : contribution to the development of an interactive support tool

The work presented in this thesis deals with interactive aiding design of process. The design support tool is based on a knowledge model described as a Constraint Satisfaction Problem (*CSP*). The search for solutions being interactive, the tool makes use of *CSP* filtering methods. The range of knowledge used leads us to integrate different *CSP* types (discrete, continuous, mixed, dynamic) as well as their filtering methods (arc-consistency, 2B-consistency, discretization). This work is based on an industrial problem relevant to the design of heat treatment operations.

The first part gives the background of our works and presents the industrial problem. In the second part, a typology of the constraints necessary to build knowledge models is made. This typology allows us to identify the different CSP types to use. Each one is presented in terms of its state of art, needs and adopted solutions.

Taking into account 2D graphs in constraints-based models necessitates the use of a specific data structure named a quad tree. The integration of this data structure does not raise any particular problem for the graphs described as a single numerical constraint. We extend it in order to take into account the graphs described as piecewise numerical constraints.

The last part presents the general architecture of a constraint-based model built in collaboration with heat treatment experts and shows how the various filtering engines are gathered.

Keywords : conception, constraints, filtering, quad tree, heat treatment