Modélisation et résolution d'une application d'aide au déploiement d'antennes radio en programmation par contraintes sur le discret et le continu

Michaël Heusch

Université de Nantes

ÉCOLE DOCTORALE STIM
■ SCIENCES ET TECHNOLOGIES DE L'INFORMATION ET DES MATÉRIAUX ■

Année 2005

Modélisation et résolution d'une application d'aide au déploiement d'antennes radio en programmation par contraintes sur le discret et le continu

THÈSE

pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE NANTES

Discipline : INFORMATIQUE

présentée et soutenue publiquement par

Michael HEUSCH

le 30 janvier 2006

à l'UFR Sciences et Techniques, Université de Nantes

devant le jury ci-dessous

Président	:	Pr. PrénomPrésident NomPrésident	Institution
Rapporteurs	:	Gilles PESANT, Professeur	École Polytechnique de Montréal
		Michel RUEHER, Professeur	Université de Nice-Sophia Antipolis
Examinateurs	:	Nicolas BELDICEANU, Professeur	École des Mines de Nantes
		Frédéric GOUALARD, Maître de conférences	Université de Nantes
		Juliette MATTIOLI, Industriel	Thales

Directeur de thèse : Pr. Frédéric BENHAMOU
Encadrants de thèse : Frédéric BENHAMOU, Frédéric GOUALARD et Juliette MATTIOLI
Laboratoire : LINA (Laboratoire d'Informatique de Nantes Atlantique) N° ED 0366-XXX

MODÉLISATION ET RÉSOLUTION D'UNE APPLICATION D'AIDE AU DÉPLOIEMENT D'ANTENNES RADIO EN PROGRAMMATION PAR CONTRAINTES SUR LE DISCRET ET LE CONTINU

Modeling and solving of a radio antennas deployment support application by constraint programming over finite and continuous domains

Michael Heusch

favet neptunus eunti

Université de Nantes

Michael HEUSCH

Modélisation et résolution d'une application d'aide au déploiement d'antennes radio en programmation par contraintes sur le discret et le continu

xii+144 p.

Ce document a été préparé avec LaTeX2$_\varepsilon$ et la classe `these-IRIN` version 0.92 de l'association de jeunes chercheurs en informatique LOGIN, Université de Nantes. La classe `these-IRIN` est disponible à l'adresse :

`http://login.irin.sciences.univ-nantes.fr/`

Impression : template-these.tex – 5/12/2005 – 0:53

Révision pour la classe : $Id: these-IRIN.cls,v 1.3 2000/11/19 18:30:42 fred Exp

Sommaire

Liste des Symboles

$\check{Z}$	Centre d'un intervalle Z
Δ_{eer}	Constante d'écart de fréquence minimum d'inter-modulation entre un récepteur et deux émetteurs co-sites
Δ_{ee}	Constante d'écart de fréquence minimum entre deux émetteurs co-sites
Δ_{er}	Constante d'écart de fréquence minimum entre un émetteur un récepteur co-sites
Δ_h	Constante d'écart de fréquence minimum de compatibilité distante pour deux sites en portée
Δ_l	Constante d'écart de fréquence minimum de compatibilité distante pour deux sites à mi-portée
Δ_{rr}	Constante d'écart de fréquence minimum entre deux récepteurs co-sites
$\mathbb{F}$	L'ensemble des nombres flottants
$\lceil f \rceil$	plus petit entier plus grand ou égal au réel f
$\lfloor f \rfloor$	plus grand entier plus petit ou égal au réel f
$\overline{I} \equiv sup(I)$	Borne supérieure d'un intervalle $I = [\underline{I}, \overline{I}]$
$\underline{I} \equiv inf(I)$	Borne inférieure d'un intervalle $I = [\underline{I}, \overline{I}]$
d_h	distance au delà de laquelle deux sites sont à mi-portée
d_l	distance en deçà de laquelle deux sites sont à mi-portée
$dist$	La fonction distance euclidienne entre deux points
E_d	Ensemble des liaisons l_{ij} pour lesquelles on impose un écart duplexe Δ_d
E_h	Ensemble des liaisons l_{ij} pour lesquelles on impose un écart harmonique Δ_h
$E_{f_=}$	Ensemble des couples (t_{ij}, t_{kl}) pour lesquels on impose une fréquence égale
$E_{f_\neq}$	Ensemble des couples (t_{ij}, t_{kl}) pour lesquels on impose une fréquence différente
E_{f_i}	Ensemble des couples (t_{ij}, f_{i_k}) pour lesquels on impose une certaine fréquence f_{i_k} prédéfinie
$E_{p_=}$	Ensemble des couples (t_{ij}, t_{kl}) pour lesquels on impose une polarisation égale
$E_{p_\neq}$	Ensemble des couples (t_{ij}, t_{kl}) pour lesquels on impose une polarisation différente
E_{p_h}	Ensemble des trajets t_{ij} pour lesquels on impose une polarisation horizontale
E_{p_v}	Ensemble des trajets t_{ij} pour lesquels on impose une polarisation verticale
F_{ij}	Le domaine fréquentiel disponible pour le trajet t_{ij}

f_{ij}	La fréquence assignée au trajet t_{ij}
M	Matrice contenant en $M_{i,j}$ les distances (portées) maximales entre les sites S_i et S_j
m	Matrice contenant en $m_{i,j}$ la distance minimales entre les sites S_i et S_j
P_{ij}	Le domaine de polarisation disponible pour le trajet t_{ij}
p_{ij}	La polarisation assignée au trajet t_{ij}
PI	Point d'Implantation
S_i	Un Site du problème LocRLFAP
T	L'ensemble des trajets du problème
t_{ij}	Un trajet de S_i à S_j
ZD	Zone de Déploiement
ZI	Zone Interdite
ZPI	Zone Potentielle d'Implantation
`Rel`	Un symbole de relation $\in \{=, \neq, >, \geq, <, \leq\}$

Introduction

Depuis le début des années 90 la Programmation par Contraintes (PPC) a fait ses preuves de réussite dans la résolution de problèmes industriels d'optimisation combinatoire. C'est une technologie de choix pour le développement de systèmes d'aide à la décision utilisés dans le cadre de l'optimisation des ressources. Elle vise à améliorer la productivité dans des secteurs de métiers comme l'ordonnancement de la production, la planification de personnel, la logistique d'approvisionnement, dans lesquels elle offre une flexibilité significative et une efficacité réelle pour modéliser et résoudre des systèmes combinatoires complexes.

L'objectif de cette thèse est de développer des techniques permettant de traiter un problème opérationnel de déploiement d'antennes rencontré chez THALES dans le cadre de la PPC. Cette application associe le problème classique en Recherche Opérationnelle de l'allocation de fréquences avec un problème d'Analyse de Localisation. Dans le problème classique allocation de fréquences radio, des liaisons sont établies entre différentes antennes et l'objet est d'allouer une fréquence à chaque liaison en minimisant l'utilisation des fréquences et en respectant toutes les contraintes physiques liées au matériel. La distance entre les terminaux détermine *a priori* un certain nombre de contraintes d'interférence. Pour nous, les positions des antennes ne sont pas déterminées précisément à l'avance et il s'agit de trouver une allocation optimale à partir de la localisation la plus favorable. La nécessité de prendre en compte simultanément ces deux aspects, cumulant contraintes discrètes et continues, entraîne que les méthodes classiques ne sont pas exploitables.

La majorité des applications, pour lesquelles la PPC s'est distinguée, sont modélisées exclusivement par des contraintes discrètes. Le système de PPC appelé solveur met en œuvre les problèmes combinatoires dans un langage déclaratif qui exprime dans des énoncés logiques un ensemble de contraintes prédéfinies. Celles-ci peuvent être des expressions simples, par exemple des expressions arithmétiques, ou des contraintes symboliques exploitant des algorithmes de résolution spécialisés. La coopération des mécanismes de résolution autorise à envisager pour le problème global, une solution à la fois générique et efficace.

Pour les problèmes continus, des solveurs capables de résoudre des systèmes composés de contraintes linéaires ont été développés depuis les débuts de la PPC. L'introduction de techniques issues de l'arithmétique d'intervalles a rendu récemment possible la résolution de systèmes d'équations non-linéaires difficiles et constitue encore un axe de recherche prometteur.

Lorsqu'il s'agit de traiter des systèmes hybrides discrets-continus, la PPC rencontre des obstacles. L'application de déploiement d'antennes radio comprend des ressources de nature discrète soumises à des contraintes d'ordre géométrique, et en cela associe un problème de type combinatoire à des contraintes de type continu, ce qui la rend intrinsèquement hybride discrète-continue.

L'approche originelle pour modéliser cette application en contraintes discrétise entièrement le problème. Cette méthode est peu convaincante car elle ne tient pas compte de la sémantique des contraintes et entraîne dans certains cas une dégradation de la qualité des solutions. La résolution nécessite de disposer aussi bien de techniques efficaces pour résoudre le problème combinatoire d'allocation de ressources, que d'un cadre adapté pour modéliser la composante numérique du problème d'optimisation sous contraintes.

Un certain nombre de travaux autour du langage HCC [55, 54] ont été effectués sur la résolution de systèmes hybrides discrets-continus en PPC, mais ils s'intéressent principalement à la simulation en des points discrets de problèmes évoluant dans un temps continu. Pour nous, l'objectif visé est de résoudre des problèmes d'optimisation, qui comportent simultanément des contraintes de nature discrète et continue. Une façon de procéder est de les plonger dans un univers de calcul continu. Or si les solveurs de contraintes continues permettent de traiter des problèmes comportant à la fois des variables discrètes et continues, leur conception repose généralement sur l'utilisation d'un seul algorithme de résolution de type continu. Cette restriction limite de fait l'efficacité du mécanisme de résolution lorsque la composante discrète du problème est importante, comme cela se présente dans notre application. Notre approche est d'intégrer dans le solveur de domaines finis Eclair un algorithme générique de propagation de contraintes continues. La définition de contraintes globales continues de distance euclidienne, et la mise au point d'algorithmes de recherche adaptés améliorent alors la résolution du problèmes combinatoire. Ces composants logiciels nous donnent les clefs d'un outil d'aide à la décision pour le déploiement d'antennes radio.

Nous aborderons dans un premier chapitre un état de l'art de la programmation par contraintes, où sont mises en avant les similarités et les différences entre les techniques de résolution des contraintes discrètes et des contraintes continues. Après avoir donné un aperçu des principes généraux de la PPC, nous présentons les techniques les plus importantes propres à chacun des domaines de définition de contraintes. Ces bases nous permettent d'envisager les hybridations possibles entre ces deux types de méthodes.

Nous proposons dans un deuxième chapitre une nouvelle contrainte globale, appelée *distn*, qui maintient des relations de distance euclidienne entre n points. Cette contrainte, dont nous montrons l'utilité pour modéliser une variété assez hétérogène de problèmes, présente la nouveauté d'être implémentée sur des domaines continus. Deux algorithmes de filtrage sont détaillés pour cette contrainte, ils mettent en œuvre des raisonnements géométriques dérivés du problème de « packing de cercles ». L'utilisation de l'arithmétique d'intervalles dans l'implémentation de ces algorithmes garantit la correction des calculs effectués sur les domaines continus. Une analyse comparative avec les différents algorithmes de propagation existants mesure les performances de l'implémentation aussi bien en termes de rapidité du calcul que de nombre de backtracks nécessaires pour trouver les solutions de quelques benchmarks.

Notre dernier chapitre décrit le traitement de l'application de déploiement d'antennes avec allocation de fréquences radio. Nous proposons un modèle mathématique adapté à cette application que nous déclinons en contraintes selon un modèle discrétisé et selon un modèle hybride conjuguant contraintes discrètes et continues. La résolution de ces modèles met en évidence que l'hybridation naïve n'est pas suffisante car le filtrage des contraintes continues se montre trop faible. Nous proposons par conséquent la mise en œuvre de la contrainte *distn* dans chacun des deux modèles et le développement d'heuristiques de recherche avancées, moyens grâce auxquels l'hybridation est alors gagnante par rapport aux différents modèles entièrement discrets. Ces approches sont validées par une série d'exemples de test ; une comparaison qualitative et quantitative des résultats permet d'apprécier l'avantage de notre approche.

CHAPITRE 1

Approches discrètes et continues de la Programmation par Contraintes pour les Systèmes Hybrides discrets-continus

Ce chapitre présente l'approche choisie dans cette thèse pour traiter les problèmes d'optimisation combinatoire : la Programmation par Contraintes (PPC). Nous donnons d'abord un aperçu des principes de cette discipline, puis nous montrons comment elle traite les difficultés inhérentes aux contraintes discrètes d'une part, et continues d'autre part. Enfin, les modèles et les méthodes qui permettent d'envisager plus généralement des problèmes hybrides discrets-continus en PPC seront décrits.

1.1 Un aperçu de la Programmation par Contraintes

La PPC est une technologie adaptée à la résolution de problèmes d'optimisation combinatoires. Elle connaît depuis les années 90 un certain succès dans le milieu industriel où de nombreuses applications, dont l'abord est difficile avec les outils classiques de la Recherche Opérationnelle, ont été résolues. Plusieurs solveurs de PPC sont actuellement commercialisés (Ilog Solver[1], CHIP[2], ECL^{i}PS^e[3], Sicstus Prolog[4], Prolog IV[5], ConstraintWorks[6], Artelys Kalis[7] et Koalog Constraint Solver^{TM}[8]) et un certain nombre de grands groupes (Bouygues, EDF, Microsoft, Siemens, Thales) développent leurs outils de PPC pour un usage interne. Ce succès

[1] http://www.ilog.com/products/solver
[2] http://www.cosytec.com
[3] http://www.icparc.ic.ac.uk/eclipse
[4] http://www.sics.se/isl/sicstus
[5] http://prologianet.univ-mrs.fr
[6] http://www.actenum.com
[7] http://www.artelys.com/fr/produits/metis
[8] http://www.koalog.com

industriel est allé de pair avec celui qu'il a connu dans le milieu académique où cette discipline assez récente s'est développée en fédérant des chercheurs de domaines aussi divers que la Théorie des Langages, la Programmation Logique, l'Intelligence Artificielle, l'Analyse Numérique, la Recherche Opérationnelle et les Mathématiques Discrètes. Deux conférences internationales (CP depuis 1995 et CPAIOR depuis 2004) rassemblent tous les ans environ trois cents chercheurs.

La résolution d'un problème en Programmation par Contraintes suit un processus en deux étapes :

1. Le problème est modélisé comme un Problème de Satisfaction de Contraintes (CSP).
2. Un algorithme de recherche est ensuite élaboré pour résoudre efficacement cette formulation en contraintes.

La première étape se focalise sur une description logique du problème où sont définies l'ensemble des conditions nécessaires et suffisantes qui caractérisent ses solutions. On lui associe une formulation (si possible équivalente) en contraintes. Pour cela, on introduit un ensemble fini de variables de décision, définies sur des domaines représentant l'ensemble de leurs valeurs possibles. On spécifie ensuite un ensemble fini de relations (contraintes) que les valeurs de ces variables doivent satisfaire simultanément dans une solution. Ces contraintes définissent les combinaisons de valeurs que les variables peuvent prendre dans les domaines. La modélisation en contraintes se fait de façon déclarative (on énonce l'ensemble des contraintes sans qu'intervienne l'ordre de ces déclarations) et spécifie des prédicats dans un sous-ensemble de la logique du premier ordre. Ce schéma permet aussi de traiter des problèmes d'optimisation en introduisant une contrainte pour la fonction objectif.

La seconde étape s'occupe de mettre en œuvre des algorithmes appropriés pour calculer des solutions satisfaisant le modèle. Elle se fait en programmant un algorithme de recherche. Cet algorithme repose le plus souvent sur la construction d'un arbre qui décompose récursivement le problème en des sous-problèmes plus simples, suivant un raisonnement hypothétique. La décomposition, qui dans chaque sous-problème ajoute des hypothèses de résolution, est alternée avec l'application de techniques d'inférence dites de consistance locale[9] opérant un filtrage des valeurs incohérentes des domaines. Le solveur propage les déductions possibles sous ces hypothèses sur l'ensemble des contraintes, ce qui réduit progressivement la taille de l'espace contenant les solutions.

Cette séparation en deux étapes (modélisation et résolution) offre un grand avantage par rapport à une résolution au moyen d'une approche spécifique. Elle permet une grande flexibilité du point de vue du génie logiciel : les conséquences d'un changement dans les spécifications d'un problème se limitent (en principe) à une modification du modèle (en ajoutant et/ou supprimant des contraintes) et laissent le processus de résolution inchangé. De plus, la déclarativité est obtenue dans un langage d'une grande expressivité. Ceci comporte de nombreux avantages : on obtient une formulation concise des programmes dans un langage proche des spécifications du problème, le temps de développement et la difficulté à maintenir les applications sont réduits et la réutilisation d'un problème à l'autre des composants logiciels développés (*i.e.* des algorithmes de résolution de contraintes) est immédiate.

[9] il faudrait dire techniques de « cohérence », mais l'anglicisme « consistance » est consacré *de facto* par la communauté francophone de PPC

Opérationnellement, une approche en contraintes permet aussi de garder la structure originelle du problème (par exemple les symétries du problème) et de la prendre en compte pour améliorer sa résolution (par exemple en omettant la recherche de solutions symétriques lors de la résolution).

Notre présentation de la PPC est structurée comme suit : nous développons d'abord les modalités d'une modélisation en contraintes (Section 1.1.1) et les propriétés des techniques de propagation de contraintes (Section 1.1.2), puis nous décrivons les algorithmes de recherche de solutions et les différents modes de résolution de la PPC (Sections 1.1.3 et 1.1.4).

1.1.1 Modélisation d'un problème en contraintes

La modélisation d'un problème en contraintes passe par la définition d'un CSP dans un langage de contraintes donné. Ce cadre permet *a fortiori* de traiter des problèmes d'optimisation en associant au CSP une variable « objectif » contrainte selon la formule de la fonction objectif.

Les problèmes de satisfaction de contraintes sont apparus au début des années 70 en Intelligence Artificielle quand cette communauté a reconnu la possibilité de modéliser et résoudre des problèmes très variés sous cette forme. La première formalisation des CSP est attribuée à Montanari [91]. Nous donnons maintenant une définition formelle d'un CSP n-aire :

Définition 1.1. Un problème $\mathcal{P}$ est modélisé par un CSP en définissant un triplet $\mathcal{P} = (\mathcal{V}, \mathcal{D}, \mathcal{C})$ où

- $\mathcal{V} = \{X_1, \ldots, X_n\}$ est l'ensemble des variables du problème
- $\mathcal{D} = \{D_1, \ldots, D_n\}$ est l'ensemble associé des domaines de valeurs que peuvent prendre les variables : $\forall i\ X_i \in D_i$
- $\mathcal{C} = \{C_1, \ldots, C_m\}$ est un ensemble fini de relations (contraintes) portant sur ces variables.

On note $Vars(C_i)$ le sous-ensemble de $\mathcal{V}$ de variables présentes dans la contrainte C_i.

Une contrainte peut être définie

- explicitement, en spécifiant l'ensemble des tuples permis (ou interdits) par la contrainte,
  - par exemple avec $D_1 = D_2 = \{0, 1, 2\}$ la contrainte C_1 définie par $(X_1, X_2) \in \{(0, 1), (2, 0)\}$.
- implicitement à partir d'une équation, d'une inéquation ou d'une diséquation,
  - par exemple pour les mêmes domaines la contrainte C_2 définie par $X_1 < X_2$.
- par une combinaison logique de contraintes plus simples,
  - par exemple C_3 définie par $(X_1 < X_2) \vee (X_1 > X_2 + 1)$.,
- ou encore symboliquement par une relation prédéfinie.
  - par exemple une contrainte globale C_4 de cardinalité notée $gcc(\{X_1, X_2\}, l, u)$: elle associe chaque valeur $v_i \in Vars(C_4)$ à des entiers positifs l_i et u_i tels que pour tout tuple solution v, le nombre d'occurrences d'une valeur v_i dans v satisfait $l_i \leq v_i \leq u_i$.

Nous donnerons à la satisfaction d'une contrainte la sémantique suivante :

Définition 1.2. Une contrainte C avec $Vars(C) = \{X_1, \ldots, X_i\}$ est satisfaite par l'affectation $(v_1, \ldots, v_i)$ si et seulement si la relation $C(v_1, \ldots, v_i)$ est satisfaite.

Plus formellement, l'objet de la résolution d'un CSP peut être définie en ces termes :

Définition 1.3. Une solution du CSP est définie par une affectation $v = (v_1, \ldots, v_n)$ quand $\forall i,\ v_i \in D_i$ et la conjonction $C_1 \wedge \ldots \wedge C_m$ est satisfaite.

Une fois le problème modélisé comme un CSP, on le résout. Plusieurs objectifs sont possibles lors de la recherche de solution(s) :

- déterminer si le CSP a une solution, ou prouver sa non-satisfiabilité,
- déterminer une ou toutes ses solutions,
- déterminer une solution optimale selon une certaine fonction objectif.

De manière générale, savoir si un CSP est satisfaisable est un problème NP-complet (par réduction au problème NP-complet SAT [98]), et *a priori* il n'existe donc pas d'algorithme générique et efficace pour le résoudre. La solution d'un CSP (sur des variables discrètes) étant un point dans un espace discret fini, il est commode de voir un CSP discret comme un problème de recherche de solutions.

Pour les CSPs sur des variables continues, on se contente en général de solutions représentables en machine par des intervalles de nombre flottants ; on peut donc là aussi voir la résolution comme un problème de recherche de solutions dans un espace fini.

Cet espace de recherche étant de taille exponentielle en le nombre de variables, le plus souvent on ne peut envisager de le parcourir en entier. On utilise pour cette raison des algorithmes appelés solveurs qui éliminent des portions de l'arbre de recherche ne pouvant pas contenir de solutions.

1.1.2 Propagation des contraintes

La PPC réduit l'espace de recherche avec des algorithmes appelés solveurs – par abus de langage le système de PPC est lui même appelé solveur – qui éliminent du domaine des variables des valeurs ne pouvant pas participer à une solution.

Les solveurs procèdent pour chaque contrainte par le biais d'algorithmes de filtrage qui suppriment du domaine des variables de la contrainte des valeurs ne pouvant pas vérifier la relation qu'elle maintient. Ils agissent sur ce qu'on appelle le réseau de contraintes : l'(hyper)graphe dans lequel les variables sont les nœuds et les (hyper)arcs sont les contraintes.

La description des principaux types de solveurs utilisés nous conduit à indiquer par la suite comment ces solveurs coopèrent lors de la propagation des contraintes et quelles propriétés sont assurées par la résolution.

Un solveur est dit complet s'il garantit de détecter l'inconsistance d'un réseau de contraintes donné en entrée. Un exemple de solveur de contraintes continu complet est obtenu en utilisant l'algorithme du simplexe comme filtrage : étant donné un système d'inéquations, si on réduit pour chaque variable son domaine à la projection des bornes des simplexes, la complétude du solveur est bien vérifiée. La consistance du système complet est souvent coûteuse à assurer, il est donc nécessaire que lorsqu'on ajoute une contrainte c à un ensemble de contraintes E déjà résolu, le solveur ne résolve pas $E \cup \{c\}$ en partant de zéro. Un solveur qui satisfait cette propriété est dit incrémental.

À l'opposé, les solveurs incomplets basés sur des techniques dites de consistance locale ne peuvent détecter que tardivement l'inconsistance d'un réseau de contraintes – éventuellement seulement lorsque toutes les variables sont affectées.

En introduisant des variables intermédiaires, tout CSP discret peut être ramené à un CSP dont les contraintes sont d'arité inférieure ou égale à deux (appelé CSP binaire) défini par des

contraintes en extension[10] [107]; la majorité des premiers travaux en Intelligence Artificielle se sont par conséquent concentrés sur ce cadre. Ils ont porté sur des solveurs qui appliquent un filtrage générique pour l'ensemble des contraintes. Par exemple, le filtrage par consistance d'arc, pour une contrainte $C(X_i, X_j)$, supprime toutes les valeurs v_i du domaine $D(X_i)$ de la variable X_i qui n'admettent aucune valeur v_j telle que C soit satisfaite. Lorsqu'une telle déduction est faite, le solveur prend en charge la « propagation » de cette information au réseau de contraintes : le filtrage est réappliqué à toutes les contraintes utilisant cette variable.

Une forme importante de solveur est celle que l'on trouve dans une contrainte globale. Il s'agit de contraintes particulières qui implantent leur propre algorithme de filtrage, qui peut être complet ou incomplet. Ce concept est détaillé dans la Section 1.2.3.

La résolution de l'ensemble des contraintes par le système de PPC est obtenue par la propagation des contraintes : le solveur invoque successivement un ou plusieurs de ces algorithmes d'inférence, qui communiquent entre eux la modification du domaine des variables. Lorsque la propagation est terminée, *i.e.* que le filtrage ne peut plus supprimer de valeurs pour chacune des contraintes, le réseau a atteint un point fixe où l'on dit que le système est localement consistant.

Les propriétés fondamentales que l'on préserve dans le processus de propagation sont la correction (l'algorithme ne supprime jamais de point solution) et la confluence (l'ordre dans lequel l'algorithme s'exécute n'a pas d'incidence sur le résultat final). Elles permettent ainsi d'assurer la déclarativité de la pose d'un modèle en contraintes.

Plusieurs alternatives sont donc offertes pour simplifier un CSP jusqu'à sa résolution. Il faut alors prendre en considération la complexité algorithmique de chacun des solveurs utilisés ; le meilleur choix dépend cependant du problème traité, et il faut savoir déterminer un bon compromis (défini sur une base expérimentale) entre le temps pris par l'application de solveurs plus ou moins forts et l'élagage obtenu. Pis encore, seule une partie des solveurs est finalement réellement intéressante d'un point de vue pratique dans la mesure où la complexité des techniques de consistance fortes peut malheureusement parfois dépasser celle de l'exploration d'un arbre de recherche. La prochaine section s'attache pour cela à décrire une solution à ce dernier problème.

Les techniques de consistance locale, à l'exception de celles trouvées dans ALICE [74] n'ont généralement pas été intégrées dans des langages de programmation. C'est là précisément l'un des apports fondamentaux de la PPC.

Étant donné l'importance centrale de ces techniques en PPC, nous les détaillons davantage dans les sections 1.2 pour des variables discrètes, 1.3 pour des variables continues. Les approches existantes pour la combinaison des modes de résolution de contraintes discrètes et continues pourront alors être abordées dans la Section 1.4.

1.1.3 Algorithmes de recherche

La propagation des contraintes ne permet pas en général à elle seule de déterminer une solution en PPC. De ce fait, on procède à l'exploration (la plupart du temps en profondeur d'abord) d'un arbre de recherche qui décompose le problème en des sous-problèmes plus simples.

[10] *i.e.* par la liste des combinaisons autorisées de valeurs

Il est nécessaire dans ce couplage de trouver un bon compromis (défini sur une base expérimentale) entre le temps pris par l'application de techniques de maintien de la consistance locale (plus ou moins forte) et celui passé dans l'exploration de l'arbre de recherche.

Nous introduisons d'abord les principes des algorithmes naïfs de recherche purs, puis nous montrons comment ils peuvent être combinés avec les solveurs décrits ci-dessus en 1.1.2. Des méthodes de résolution efficaces peuvent ainsi être développées en les combinant avec des heuristiques.

Originellement, en Intelligence Artificielle, les CSP (discrets) étaient résolus par des méthodes d'énumération appelées « Génération et Test » (*generate & test*) et « Recherche par retour-arrière » (*backtracking*) [48] :

- Le *generate & test* se contente d'énumérer toutes les combinaisons de valeurs et de vérifier *a posteriori* si elles violent[11] ou non les contraintes.
- Le *backtracking* [48] est une amélioration du *generate & test* qui procède en parcourant « en profondeur d'abord » un arbre de recherche. La racine représente l'affectation vide et dans les feuilles de l'arbre toutes les variables sont affectées. Un nœud de l'arbre représente une affectation partielle des variables et les arcs représentent des transitions (ou choix). Le parcours énumère les valeurs possibles des variables et effectue un retour-arrière (backtrack) lorsque l'affectation partielle en un nœud de l'arbre viole une contrainte.

Or ces méthodes sont de complexité exponentielle et généralement peu utilisables. En observant les défaillances du *backtracking*, appelées *thrashing*, qui sont :

- la redécouverte perpétuelle de la validité ou de l'incohérence d'une contrainte,
- la découverte tardive des incohérences,
- le *backtrack* se fait vers un nœud sans rapport avec la cause de l'incohérence,

Waltz [116], Montanari [91] et Mackworth [80] ont introduit les premiers algorithmes (génériques) de consistance locale permettant de détecter des conflits sur des affectations partielles avant même d'énumérer des tuples inconsistants. Ce traitement supplémentaire introduit naturellement un surcoût, mais il réduit la taille de l'espace à explorer. Si les calculs restent exponentiels dans le pire cas, c'est cette combinaison qui permet d'obtenir de bons résultats ; la voici à présent détaillée.

Montrons comment les techniques de consistance peuvent être incorporées dans la construction d'un arbre de recherche en profondeur d'abord, et comment ce schéma s'adapte à différents modes de résolution pour la recherche complète ou partielle de solutions.

Nous utilisons dans la suite la notion de variable instanciée. Elle désigne dans le cas discret une variable qui est affectée à une valeur, et dans le cas continu une variable affectée à un intervalle soit canonique[12] soit d'une taille inférieure à la précision recherchée. Le schéma général de recherche est le suivant :

(1) Sélection d'une variable X_i non-affectée (selon l'heuristique de choix de variables)

(2) Décomposition du CSP en sous-problèmes (selon des hypothèses différentes sur X_i)

(3) Exploration d'un des sous-problèmes (sélectionné selon l'heuristique de choix de valeurs)

(4) Propagation des contraintes

[11] on parle de la violation d'une contrainte lorsque sa relation sous-jacente ne peut pas être vérifiée

[12] *i.e.* un intervalle qui contient au plus deux nombres flottants

(5) Traitement sur le nouveau nœud :

(a) Si le domaine d'une variable est vide
- Retour à l'étape (1).

(b) Sinon
- Si certaines variables n'ont pas encore été instanciées, retour à l'étape (1),
- Sinon on a trouvé une solution du CSP, la retourner.

La décomposition en (2) peut être vue comme la pose d'une disjonction, et le choix effectué en (3) comme la pose d'une hypothèse de travail sur l'une des branches de la disjonction. Pour des variables discrètes, ce procédé peut être mis en œuvre en énumérant les valeurs de leur domaine. D'autres options telles qu'un découpage par dichotomie du domaine de la variable ou encore le branchement sur une contrainte présente dans une disjonction du modèle sont aussi possibles. Pour des variables continues, on bisecte généralement l'intervalle courant. Observons au préalable que si en (2) on se contente de créer un sous-problème pour chaque valeur, et qu'au lieu de propager les contraintes on vérifie si l'une d'elles est violée par l'affectation partielle courante, on retrouve le schéma de *backtracking*.

L'algorithme de *backtracking* peut être amélioré en substituant l'utilisation *a posteriori* des contraintes à chaque nœud par un filtrage *a priori* des valeurs prises par les variables non encore affectées par une affectation partielle. On passe alors d'un usage passif à un usage actif des contraintes. Selon le type de propagation effectué en (4), on obtient plusieurs algorithmes classiques :

- Le *forward-checking* propage le choix effectué en (2) seulement depuis les variables instanciées vers les variables non encore affectées.
- *Maintaining Arc-Consistency (MAC)* – aussi appelé *look-ahead* – est plus fort et propage le choix effectué en (2) sur tout le réseau de contraintes. La décomposition du problème induit la perte de la consistance d'arc que le système vérifiait en (1) ; la propagation complète rétablit alors cet état et c'est pourquoi on parle ici de maintien de la consistance d'arc. Le même principe est bien sûr applicable pour d'autres formes de consistance du réseau de contraintes.

Des études effectuées sur un grand nombre de benchmarks aléatoires privilégient ce dernier [21].

Deux heuristiques peuvent être utilisées comme paramètres de ce schéma général :

- l'heuristique de choix de variables en (1) pour déterminer dans quel ordre les variables sont sélectionnées,
- l'heuristique de choix de valeurs en (3), applicable pour les variables discrètes, pour déterminer dans quel ordre les valeurs sont examinées.

Ces deux heuristiques permettent de guider la recherche selon un parcours qui vise à optimiser la propagation des contraintes. En conséquence, elles jouent souvent un rôle déterminant dans la rapidité de la résolution.

Ces heuristiques peuvent être appliquées statiquement en exploitant la structure du réseau de contraintes, par exemple en ordonnant les variables selon un nombre décroissant de voisins. L'approche la plus répandue est toutefois de les utiliser dynamiquement en vue de détecter des échecs au plus tôt ; ce principe est appelé le *first-fail* [57]. Des heuristiques simples de *first-fail* sont par exemple de choisir comme variable suivante, celle ayant le plus petit domaine (pour des variables discrètes) ou le plus grand domaine (pour des variables continues), ou encore celle ayant le plus grand nombre de voisins instanciés dans le réseau de contraintes. On pourra se

référer à [21] pour d'autres exemples et une analyse de la performance de ces heuristiques.

Notons pour terminer que l'exploration en largeur d'abord est une alternative parfois intéressante. Elle est bien plus performante pour certains problèmes continus où elle permet de suivre le principe de first-fail. Son champ d'application est pour autant limité à des problèmes de petite taille étant donné que sa consommation en mémoire explose très rapidement.

1.1.4 Modes de résolution

Qu'attendons-nous de la résolution d'un problème ? C'est que celle-ci détermine si ce problème admet une solution et prouve qu'il en admet une, qu'elle calcule une ou toutes ses solutions, ou qu'elle permette de trouver la meilleure solution possible par rapport à un objectif déterminé. Toutes ces façons d'aborder la résolution d'un problème peuvent être satisfaites en PPC.

La recherche est dite globale (ou complète), quand elle permet de prouver l'optimalité de la solution obtenue. *A contrario*, lorsqu'une preuve n'est pas nécessaire, ou que la recherche complète est trop coûteuse, une recherche partielle ou locale dans l'arbre de recherche permet parfois d'obtenir une « bonne solution ».

Après avoir montré comment le schéma introduit ci-dessus permet également de traiter des problèmes d'optimisation, nous expliquons le fonctionnement, les avantages et les limites des algorithmes de recherche globale, partielle et locale.

Un algorithme de *Branch & Bound* naïf peut être dérivé du parcours arborescent décrit ci-dessus. Supposons qu'une première solution ait été trouvée lors du parcours. Pour assurer que les solutions suivantes aient un meilleur coût, il suffit d'imposer une amélioration à une variable contrainte selon la fonction objectif. Ce procédé permet d'effectuer un branchement (un retour-arrière dans l'arbre) dès que la borne de la fonction objectif est violée.

Nous avons vu que lors de la construction de l'arbre de recherche, on traite la totalité des combinaisons d'affectations possibles. À chaque appel, le solveur réduit le CSP en un autre équivalent, au sens où les seules valeurs élaguées sont garanties de ne pas pouvoir participer à une solution. Ce parcours apporte donc bien une preuve du résultat obtenu.

Ces garanties sont préservées lorsque l'on résout des problèmes sur les variables continues car l'arithmétique d'intervalles permet de garantir la correction des résultats. Un effet pervers de la correction est toutefois que l'on peut être assuré que l'ensemble des solutions au problème (s'il en existe) se trouvent dans les domaines retournés, mais on n'a pas toujours la possibilité de prouver (pour des variables continues) l'existence de solution dans ces domaines.

La garantie de complétude dans la résolution a cependant l'inconvénient d'être parfois trop lente pour permettre de résoudre des problèmes d'optimisation de très grande taille. On peut dans ces cas se contenter d'une solution garantie dans un voisinage donné de l'optimum. Cette restriction permet d'accélérer notablement la résolution sur certains problèmes. On peut obtenir un tel résultat sur un algorithme de *Branch & Bound* en prouvant (par une recherche complète) qu'il n'existe pas de solution de coût inférieur à l'écart donné.

Outre les méthodes de recherche partielle basées sur des techniques de diversification du

parcours de recherche [13], les méthodes de recherche locale sont une alternative commune à la PPC pour traiter des problèmes d'optimisation de très grande taille. Elles opèrent sur des instanciations complètes des variables (appelés états). Leur principe est de chercher à améliorer progressivement la valeur d'une fonction de coût d'un état – par exemple le nombre de contraintes violées, pour un problème de satisfaction – en explorant son voisinage. On obtient ces configurations proches par un opérateur qui, en changeant les valeurs de quelques variables seulement, obtient d'autres solutions satisfaisant les contraintes. Les approches les plus utilisées sont le « Recuit-Simulé » [68] et la « Recherche Tabou » [47]. Cette dernière approche permet d'obtenir de très bons résultats par exemple sur des problèmes d'allocation de fréquence de très grande taille [42]. Les inconvénients principaux des méthodes de recherche locale sont toutefois qu'elles peuvent rester bloquées sur un minimum local de l'espace de recherche et qu'elles n'offrent aucune preuve de l'absence ou de l'optimalité des solutions.

1.1.5 Synthèse

Cette section a donné un aperçu du cadre général de la PPC en introduisant les principes de la modélisation, de la propagation des contraintes et de la résolution d'un problème par des algorithmes de recherche. Pour comprendre la problématique de notre thèse et la nature de notre contribution, il nous reste maintenant à détailler les techniques de résolution de contraintes dans les domaines discrets et continus ; cela nous permettra d'appréhender la dernière section de l'état de l'art sur les techniques permettant de combiner les deux domaines de résolution.

1.2 Résolution de contraintes discrètes

Nous avons précisé dans la section 1.1.1 différentes formes sous lesquelles les contraintes d'un CSP pouvaient s'exprimer. Pour un CSP discret on aura notamment :

- des tuples autorisés en extension (pour une contrainte très forte),
- des tuples interdits en extension (pour une contrainte très lâche),
- des contraintes définies implicitement par une relation (par exemple arithmétique),
- des contraintes « globales » ayant une sémantique prédéfinie,
- des combinaison logique de contraintes.

Notre but est ici d'exposer les méthodes permettant de les résoudre, ou du moins de réduire la taille de leurs domaines.

Les premiers travaux sur les CSPs se sont d'abord intéressés à la conception d'un algorithme efficace mais générique pour résoudre l'ensemble des contraintes sans leur attribuer de sémantique particulière. Cette approche s'est finalement avérée manquer de réalisme et il est entre-temps admis en PPC que suivant le type de contraintes discrètes, différents algorithmes de résolution doivent être utilisés pour la résolution efficace d'un problème.

Nous introduisons d'abord les techniques génériques des CSPs (Section 1.2.1), puis nous indiquons les classes particulières de contraintes pour lesquelles des solveurs spécifiques (Section 1.2.2) ou encore des contraintes globales (Section 1.2.3) ont été développées.

[13] Nous citons en particulier le *LDS (Limited Discrepancy Search)* [117]. Cependant cette méthode étant protégée par un brevet est peu utilisable en milieu industriel

1.2.1 Consistances locales génériques

Les travaux en Intelligence Artificielle sur les CSPs discrets se sont principalement intéressés à des méthodes génériques pour traiter des contraintes unaires et binaires sur des variables discrètes dont le domaine est défini en extension. Ces approches ont été ensuite étendues à des contraintes n-aires.

Bien que l'importance de cette approche n'est que anecdotique dans nos travaux, elle est l'analogue discret d'un algorithme que nous utilisons pour des contraintes continues (*cf.* Algorithme 2 p. 21) et il nous semble pour cela utile d'indiquer ses principaux résultats.

Nous nous intéressons en premier lieu aux contraintes binaires et nous indiquons en second lieu les limites liées à l'extension au cas n-aire des résultats.

1.2.1.1 Consistances génériques pour les contraintes binaires

Les algorithmes de consistance locale développés, poursuivant la quête [6] d'un algorithme de résolution « universel », ont d'abord été conçus pour des contraintes dépourvues de sémantique particulière. Les travaux ultérieurs visent à prendre en compte la spécificité de certaines sous-classes de contraintes et leur consacrent une spécialisation plus efficace de l'algorithme.

Définissons d'abord les formes les plus élémentaires de consistance locale :

- Une contrainte unaire $C(X_i)$ satisfait la consistance de nœud si
  - $\forall v_i \in D_i$, on a $v_i \in C$

  i.e. si toute valeur de $D(X_i)$ satisfait C. Un CSP satisfait la propriété de consistance de nœud si toutes ses contraintes unaires satisfont cette propriété.
- Une contrainte binaire $C(X_i, X_j)$ satisfait la consistance d'arc si
  - $\forall v_i \in D(X_i), \exists v_j \in D(X_j) \mid (v_i, v_j) \in C$
  - $\forall v_j \in D(X_j), \exists v_i \in D(X_i) \mid (v_i, v_j) \in C$

  Un CSP satisfait la propriété de consistance d'arc si toutes ses contraintes binaires satisfont cette propriété.

L'algorithme AC-3[14] (décrit dans l'Algorithme 1) développé par Mackworth [80] assure la consistance d'arc d'un système de contraintes binaires.

Algorithme 1 : Algorithme de propagation AC-3

Entrées : $\{C_1, \ldots, C_m\}$
Sorties : $D = D_1 \times \ldots \times D_n$
$Q = \{C_1, \ldots, C_m\}$
tant que $Q \neq \emptyset$ **faire**
  choisir et extraire une contrainte C_i de Q
  $D' := revise(C_i, D)$
  si $(D' = \emptyset)$ **alors retourner** $(\emptyset)$
  $Q := Q \cup \{C_j \mid \exists X_k \in Vars(C_j)$ pour laquelle $D'_k \neq D_k\}$
  $D := D'$
retourner (D)

AC-3 procède comme suit : un ensemble Q est initialisé avec l'ensemble des contraintes du problème. Une boucle est ensuite invoquée[15] tant qu'il reste des contraintes à « réviser » dans

[14] AC-3 est une améliorations des algorithmes naïfs AC-1 et AC-2 du même auteur.

[15] Dans le cas où le CSP contient aussi des contraintes unaires, il suffit d'itérer une fois sur ces contraintes avant

Q. À chaque itération, on choisit et retire une contrainte C_i de Q et on « révise » les domaines des variables de C_i en éliminant les valeurs ne satisfaisant pas la contrainte. On rajoute ensuite à Q les contraintes ayant une variable dont le domaine vient d'être modifié avant de réitérer la boucle.

En notant d la taille maximale des domaines et m le nombre de contraintes, la complexité algorithmique (dans le pire cas) de AC-3 est $\mathcal{O}(m^2d^3)$. Elle est améliorée à $\mathcal{O}(md^2)$ dans AC-4 [89], en remplaçant une approche orientée-variables par une approche orientée-valeurs : au lieu de réviser un arc du graphe, AC-4 révise le support de chaque valeur pour chaque variable, et supprime les valeurs dons le support devient vide. La complexité en moyenne de AC-4 est toutefois souvent plus élevée que celle d'AC-3. C'est pourquoi les successeurs d'AC-4 (AC-6 [19], AC-7 [20], et AC-2001 [23]) introduisent (à l'exception d'AC-2000 [23]) des structures de données sophistiquées qui s'efforcent d'améliorer sa complexité en moyenne et permettent de réduire la complexité spatiale à $\mathcal{O}(md)$ tout en conservant sa borne supérieure dans le pire cas, mais au final, le constat est que le coût lié au maintien des structures de données lors de l'algorithme de recherche ainsi que leur difficulté d'implantation, font que l'on préfère souvent utiliser AC-3 (ou AC-2000) plutôt que les variantes d'AC-4.

Dans certaines situations, la consistance d'arc n'arrive pas à prendre en compte la dépendance entre les contraintes et le filtrage obtenu est en conséquence très faible. Il est alors possible d'appliquer des formes de consistance d'ordre supérieur, considérant simultanément plusieurs contraintes. Si le compromis entre la complexité de la recherche et celle du raisonnement sur les contraintes tend à privilégier des formes de consistance plus faibles dans le cas discret, il en est parfois autrement dans le continu [102].

1.2.1.2 Consistances génériques pour les contraintes n-aires

Nous avons évoqué plus tôt (*cf.* p. 4) la possibilité de décomposer des contraintes n-aires en contraintes binaires. Pour des raisons d'occupation mémoire et d'efficacité, cette approche n'est cependant pas réaliste en pratique : la décomposition entraîne la perte d'une partie de la sémantique de la contrainte [104] et du même fait cause un filtrage plus faible.

L'analogue n-aire de la consistance d'arc est la consistance d'hyperarc, appelée aussi consistance d'arc généralisée (GAC). Elle assure dans toutes les contraintes que, pour toute valeur du domaine d'une variable de la contrainte, il existe une solution de la contrainte à laquelle cette affectation participe.

Alors qu'un grand nombre d'algorithmes génériques ont été proposés pour les contraintes binaires, nous n'en recensons que trois pour les contraintes n-aires [81, 90, 22]. Leur utilisation n'est souvent pas envisageable de par leur trop grande complexité. Nous verrons à la section suivante que GAC-schéma [22] est tout de même intéressant à des fins de prototypage.

1.2.2 Solveurs adaptés à la propagation de contraintes spécifiques

Nous décrivons maintenant une autre approche prise plus récemment en Programmation Logique avec Contraintes [85], procédant à l'implantation de filtrages incomplets spécifiques à

d'entrer dans la boucle.

chaque contrainte.

Si les méthodes génériques précisées ci-dessus peuvent être efficaces pour des problèmes donnés en extension, leur applicabilité se limite toutefois en général à des contraintes binaires et/ou à des problèmes de petite taille. Notamment, des contraintes importantes, apparaissant de façon récurrente dans les problèmes résolus en PPC, ont été identifiées, et des algorithmes spécialisées développés pour celles-ci s'avèrent bien plus efficaces que les précédents algorithmes génériques de cohérence d'arc. Mentionnons par exemple les contraintes :

- linéaires à variables et coefficients entiers,
- arithmétiques[16]à variables et coefficients entiers,
- de valeur absolue.

De façon générale, il semble naturellement plus avantageux de considérer des algorithmes spécialisés implantant leur propre mécanisme de propagation. Ceci est justifié par les raisons suivantes :

- on peut définir des règles de propagation qui évitent que chaque valeur du domaine d'une variable soit examinée indépendamment,
- l'appel à l'algorithme de filtrage est spécialisé pour (et limité à) certains types de modification du domaine d'une variable,
- un test rapide de la consistance de la contrainte peut être spécifié,
- le niveau de filtrage est spécialisé pour chaque contrainte.

Le filtrage utilisé pour ces contraintes se contente souvent d'une forme de consistance locale plus faible comme par exemple la consistance aux bornes (appelée aussi consistance d'intervalles) que nous définissons maintenant.

Différentes définitions sont possibles pour cette consistance, et pour éviter toute confusion [31] nous les précisons pour des domaines donnés en intension sur $\mathbb{Z}$ et $\mathbb{R}$ et en extension (que nous notons $\mathbb{D}$) :

- Une contrainte $C(X_i, \ldots, X_n)$ satisfait la condition de consistance aux bornes(b), avec $b \in \{\mathbb{D}, \mathbb{Z}, \mathbb{R}\}$, si :
  – $\forall v_i \in \{inf\{v \mid v \in D_i\}, sup\{v \mid v \in D_i\}\}, \forall j,\ 1 \leq j \leq n, j \neq i, \exists v_j \in D(X) \mid C(v_1, \ldots, v_n)$
  Un CSP satisfait la propriété de consistance aux bornes si toutes ses contraintes satisfont cette propriété.

Nous renvoyons à l'article [31] pour trouver des exemples de contraintes où ces différentes formes sont équivalentes.

En reprenant les exemples de contraintes identifiées ci-dessus, on connaît des algorithmes efficaces pour assurer la borne($\mathbb{R}$)-consistance d'une équations linéaire à variables et coefficients entiers, et la borne($\mathbb{D}$)-consistance de la contrainte produit $X.Y = Z$ [5]. Le traitement des contraintes arithmétiques peut être envisagé par une décomposition utilisant la contrainte linéaire et la contrainte produit. Nous donnons plus loin (Algorithme 12, p. 91) un invariant pour la contrainte de valeur absolue.

Opérationnellement, l'utilisation d'algorithmes spécifiques à chaque contrainte permet aussi d'ajuster dans la résolution le meilleur choix de niveau de consistance à adopter pour chaque type de contraintes du problème. On peut ainsi décider de combiner par exemple la consistance d'arc pour les contraintes arithmétiques avec la consistance aux bornes pour les contraintes de valeur absolue.

Ces principes peuvent être implémentés par une architecture du solveur de contraintes où chaque contrainte implémente des procédures (appelées demons) de révision de domaines spéci-

[16] *i.e.* les expressions de la forme $\sum_{j=1}^{m} \alpha_j \prod_{i=1}^{n} X_i^{\alpha_{i,j}}\ op\ \alpha$, où $\alpha_{i,j}, |\alpha_j|, |\alpha| \in \mathbb{N}$ et $op \in \{=, \neq, \geq, >, \leq, <\}$.

fiées pour chaque type d'« évènement » possible. Par exemple dans les solveurs Eclair© [73] et Choco [72] ce sont la modification de la borne inférieure, la modification de la borne supérieure, l'instanciation d'une variable et la suppression d'une valeur du domaine de la variable. Ainsi, le filtrage opéré par chaque contrainte interagit dans l'algorithme de propagation du solveur en réagissant à une série d'évènements. On suit pour cela un schéma proche de l'algorithme AC-3, en respectant une sémantique de point fixe. Dans Eclair©, l'algorithme de propagation appelle récursivement chacun des demons en réponse à chaque évènement. Dans Choco, la gestion de plusieurs files de priorité permet de maintenir plus finement un ordre dans la propagation. La performance générale du solveur est affectée par un compromis – dépendant des contraintes utilisées – entre le coût lié à la gestion de ces files et le gain en efficacité apporté.

1.2.3 Contraintes globales

Le concept de contrainte globale permet d'étendre le champ d'applicabilité des solveurs incomplets traités dans la dernière section. Ayant conduit à des gains de performance importants des solveurs de contraintes, en « packageant » des algorithmes de Recherche Opérationnelle, il est devenu une des clefs de voûte des systèmes de PPC depuis le milieu des années 90. On trouve aujourd'hui des contraintes globales dans la plupart des solveurs de contraintes, et la pertinence de l'utilisation d'un solveur de contraintes pour un problème donné se mesure d'une certaine façon à la disponibilité dans l'outil de contraintes globales adaptées.

Après avoir introduit le principe général des contraintes globales, nous indiquons quels algorithmes de propagation on peut envisager pour ces contraintes.

1.2.3.1 Principe général

Commençons en donnant un exemple. Beaucoup de problèmes comportent un ensemble de variables qui doivent prendre des valeurs différentes, comme c'est le cas pour un problème d'ordonnancement où les tâches exécutées par une même ressource doivent être affectées à des instants différents. La contrainte globale *alldifferent* permet de résoudre ce problème : *alldifferent*$(x_1, \ldots, x_n)$ exprime la condition $\forall i \neq j,\ x_i \neq x_j$: toutes les variables de la contrainte doivent prendre une valeur distincte.

Une contrainte globale peut être vue de manière informelle comme une contrainte spécifiant un motif apparaissant dans beaucoup de problèmes et portant sur un nombre paramétrable de variables. La relation maintenue par une contrainte globale peut le plus souvent s'exprimer de façon sémantiquement équivalente par une conjonction de contraintes élémentaires; dans le cas contraire on parle de globalité sémantique.

Outre le fait qu'une contrainte globale donne une plus grande expressivité au modèle, et donc permet de raisonner à un plus haut niveau d'abstraction, elle présente l'avantage opérationnel d'isoler un sous-problème combinatoire dans le problème traité. Cette vue globale permet d'exploiter la structure d'un sous-problème pour obtenir soit des déductions plus fortes que celles obtenues par la propagation indépendante de la conjonction des contraintes encodées, soit un algorithme de propagation plus rapide.

Reprenons l'exemple de la contrainte *alldifferent* portant sur trois variables X_1, X_2, X_3 de domaine identique égal à $\{0, 1\}$. Le système de contraintes $(X_1 \neq X_2) \vee (X_2 \neq X_3) \vee (X_3 \neq X_1)$ satisfait la condition de consistance d'arc, et pourtant il n'existe pas de solution au problème. La

modélisation de ces trois contraintes par une unique contrainte globale *alldifferent*(X_1, X_2, X_3) permet de confier le maintien d'une forme plus forte de consistance à un algorithme dédié.

Notons déjà que pour rendre le concept pertinent, ces contraintes doivent être suffisamment génériques pour permettre effectivement d'obtenir des constructeurs réutilisables, plutôt qu'un catalogue de problèmes. Les contraintes globales permettent ainsi d'introduire des primitives – éventuellement propres à un domaine de métier (par exemple la contrainte *cumulative* [2] pour l'ordonnancement et *cycle* [12, 25] pour les problèmes de routage) – servant de constructeurs de base pour la modélisation (et la résolution efficace) de différents problèmes.

1.2.3.2 Algorithmes de propagation

La forme la plus forte de consistance que l'on puisse espérer pour une contrainte globale est la consistance d'arc généralisée (*cf.* p. 11). De manière générale, assurer GAC est un problème NP-dur, et l'applicabilité de cette consistance se limite donc tout au plus aux contraintes pour lesquelles on peut vérifier l'existence d'une solution en temps polynomial. Pour les contraintes représentant un problème NP-complet, pour lesquels il n'existe donc pas de tel algorithme[17], on doit se contenter d'une forme de consistance plus faible.

Trois types importants d'algorithmes de filtrage peuvent être élaborés pour une contrainte globale [105] :

1. Pour une contrainte décomposable, il est parfois possible d'inférer des contraintes valides supplémentaires. La contrainte globale peut alors générer automatiquement ces contraintes, dont la propagation permet soit d'accélérer, soit d'augmenter la propagation des contraintes représentées par la contrainte :
   - Un exemple d'accélération est donné dans [8] pour une contrainte de distance entre n points lorsqu'il existe des couples de points entre lesquels aucune contrainte ne s'applique : la génération systématique d'une clique de contraintes, inférant d'après l'inégalité triangulaire, une contrainte entre des couples de points permet d'augmenter le filtrage et d'accélérer la résolution. Un autre exemple de raisonnement portant sur des contraintes de cardinalité est donné dans [105].
   - La génération de relaxations de certaines contraintes peut parfois accélérer les calculs si celles-ci ont un algorithme de filtrage plus rapide, et permet ainsi de détecter rapidement un état inconsistant. Par exemple, la contrainte de distance de Manhattan est une telle relaxation de la contrainte de distance euclidienne.

   Ce type de génération de contraintes est intéressant – lorsque les contraintes s'y prêtent ! – puisqu'il permet d'améliorer la résolution sans outils supplémentaires en exploitant seulement le modèle.
2. Si un algorithme efficace vérifiant la satisfiabilité de la contrainte globale est disponible – c'est le cas par exemple pour la contrainte *alldifferent*, l'algorithme GAC-schéma (évoqué parmi les consistances génériques pour les contraintes n-aires) peut être utilisé. Il permet d'obtenir la consistance d'hyperarc de la contrainte globale. On notera qu'en pratique, lorsque les problèmes atteignent une grande taille, ce schéma n'est plus applicable. Il présente toutefois un intérêt pratique important : il fournit un algorithme de prototypage générique pour des contraintes globales. En effet, à partir du nombre de *backtracks* que l'on obtient dans la résolution d'un problème avec GAC, on peut se faire une idée de la

[17] en supposant $P \neq NP$

pertinence qu'il y a à développer un algorithme de filtrage spécifique pour la contrainte globale donnée [22].

3. Lorsqu'aucune des deux solutions précédentes s'applique, on doit identifier ou élaborer un algorithme *ad hoc* exploitant la structure spécifique de la contrainte. Cette connaissance permet d'espérer un filtrage soit plus fort, soit plus rapide que celui de l'algorithme standard sur la forme décomposée de la contrainte. Cette forme d'algorithme permet d'exploiter *a fortiori* les avantages présentés dans la section précédente pour des solveurs incomplets spécifiques.

Le type le plus commun d'algorithme de filtrage pour une contrainte globale est malgré tout obtenu par le dernier schéma. Nous l'exploiterons au chapitre 2 pour le cas d'une contrainte globale de distance euclidienne étant donné que l'efficacité du premier schéma est assez limitée et que le second ne s'applique pas. Les algorithmes de filtrage utilisés sont le plus souvent issus de la Recherche Opérationnelle, notamment de la Théorie des Graphes, mais certains algorithmes de Géométrie Algorithmique s'appliquent également. Nous illustrons de quelle manière des algorithmes de ces disciplines peuvent s'intégrer pour améliorer une résolution en PPC par deux exemples :

- La contrainte de cardinalité maximale (*gcc*) contraint le nombre minimal et maximal d'occurrences d'un ensemble de valeurs pour un ensemble de variables. Elle apparaît de manière récurrente dans les problèmes d'emplois du temps. On peut établir pour une contrainte *gcc* un graphe orienté dont la capacité des arêtes est bornée. La contrainte est alors consistante si et seulement si il existe un flot réalisable sur ce graphe. La consistance d'arc de la contrainte peut être établie à partir du calcul des composantes connexes du graphe résiduel de ce flot. Comme il existe des algorithmes efficaces (qui plus est incrémentaux) pour ces problèmes de graphes, on obtient une contrainte globale efficace.
- La contrainte *NonOverlapping*[18] [11] impose à un ensemble de rectangles de ne pas s'intersecter. C'est une spécialisation en 2 dimensions de la contrainte *diffn* [12] qui est très utilisée pour des problèmes de placement et d'ordonnancement. Le filtrage de la contrainte *NonOverlapping* est fait suivant un principe de balayage, algorithme classique en Géométrie Algorithmique [38]. Étant donné un rectangle, la contrainte définit pour chacun des autres rectangles la région qui lui est interdite. L'algorithme de balayage permet alors d'évaluer de façon paresseuse l'intersection de ces régions[19].

On remarquera que les algorithmes de filtrage existants ne sont pas forcément directement adaptés en l'état ; par exemple si une contrainte est appelée un grand nombre de fois par des changements minimes de son domaine, il est désirable qu'un prétraitement puisse détecter à faible coût si ces modifications entraîneront un filtrage supplémentaire par la contrainte globale. Le coût élevé des appels à la contrainte (dans des situations sans intérêt) peut sinon compromettre l'utilité générale d'un filtrage supérieur de la contrainte.

On voit que ces schémas d'intégration d'algorithmes de filtrage ne sont pas immédiats. En général le développement de nouvelles contraintes globales exige une expertise dans ces domaines, et représente un investissement de développement important. En particulier, il est souhaitable de rendre l'algorithme de filtrage incrémental sans trop perdre en complexité.

[18] Une généralisation immédiate s'applique à *diff3D*, l'extension en 3 dimensions de cette contrainte

[19] Notons que cette notion est plus faible que la consistance d'arc, mais ce problème est NP-complet

1.2.4 Synthèse

Cette section a indiqué les principales techniques de résolution de contraintes discrètes, et montré comment l'efficacité d'un solveur de PPC dans la résolution de problèmes difficiles dépend de l'utilisation de contraintes spécialisées. C'est en particulier le cas dans nos travaux où nous utilisons des algorithmes de filtrage spécifiques pour les contraintes arithmétiques et de valeur absolue et nous définissons une contrainte globale de distance au chapitre 2.

La prochaine section va s'attacher à mettre en évidence un parallèle dans la résolution des contraintes continues.

1.3 Résolution de contraintes continues

Pour un CSP continu, les contraintes portant sur des variables à valeur réelle peuvent notamment s'exprimer comme :

- une relation implicite, par exemple une formule arithmétique,
- des fonctions transcendantales,
- des combinaison logique de contraintes.

Nous introduisons ici les techniques employées et les méthodes utilisées pour les résoudre.

Face à l'impossibilité d'énumérer les valeurs réelles et aux erreurs de calcul effectuées en machine, l'adaptation des techniques de résolution de contraintes sur les domaines discrets exige des outils supplémentaires.

Cette section présente d'abord les difficultés liées au traitement des réels (Section 1.3.1) et les réponses apportées par l'arithmétique d'intervalles pour effectuer des calculs tant d'un point de vue mathématique que pratique (Section 1.3.2). Nous exposons ensuite comment sont définies les contraintes d'intervalles (Section 1.3.3), et enfin comment des solveurs analogues à ceux développés pour les contraintes discrètes peuvent être élaborés (Sections 1.3.4 et 1.3.5).

1.3.1 Problèmes du calcul réel

Traiter des contraintes numériques sur un ordinateur présuppose de pouvoir effectuer en machine des calculs sur les réels de façon correcte.

Dans certains cas il est possible de réaliser les calculs avec des nombres de précision infinie (rationnels ou réels)[20] ; c'est le choix fait pour la résolution de contraintes arithmétiques dans les premiers solveurs continus. L'utilisation de ces nombres se heurte cependant rapidement à des problèmes d'efficacité ; pour les rationnels par exemple, il est nécessaire d'effectuer des calculs très coûteux de plus grand commun diviseur afin de garder une représentation compacte des nombres. Il faut donc le plus souvent se contenter de nombres de précision finie représentables en machine, ce qui pose bien sûr des problèmes de correction lorsque des arrondis sont introduits dans les calculs.

Les nombres flottants utilisés aujourd'hui[21] répondent à la norme IEEE 754 [65]. Cette norme

[20] On mentionnera par exemple les bibliothèques Gmpq (resp. Core) qui proposent une implémentation efficace des nombre rationnels (resp. réels).

[21] Dans les années 60 et 70 chaque fabriquant utilisait sa propre implantation des nombres flottants, et l'on trouvait selon les machines, des représentations binaires, décimales et hexadécimales. Cette diversité entraînait des problèmes de portabilité du code. Ces disparités se sont depuis aplanies à la fin des années 80, et la norme IEEE 754 [65] définit un standard de représentation interne et de manipulation des nombres flottants, respectée

donne notamment les bases pour pouvoir contrôler les erreurs numériques par des arrondis dirigés[22]. Il faut toutefois encore nous munir d'une méthode qui permette d'exploiter ces possibilités. Plusieurs techniques ont été proposées pour contrôler les problème liés au calcul en machine tout en conservant une représentation de taille fixe des nombres :

- La méthode CESTAC [115] se base sur des arguments probabilistes sur l'arrondi des calculs (l'hypothèse que les erreurs d'arrondi suivent une distribution normale). Elle procède à plusieurs exécutions successives du code effectuant des propagations d'arrondis différentes. Les différents résultats sont ensuite recombinés pour en déduire le résultat (probablement) correct ainsi que son nombre de chiffres significatifs.
- L'arithmétique d'intervalles effectue les calculs sur des intervalles de nombres (flottants) en assurant par des arrondis dirigés l'inclusion de la solution dans l'intervalle calculé.

La première offre plus de commodité à l'utilisateur, la seconde de vraies garanties.

Différentes possibilités sont donc offertes pour la résolution de contraintes continues. Cependant, la seule qui prouve la correction des calculs avec une efficacité raisonnable est l'arithmétique d'intervalles.

1.3.2 L'arithmétique d'intervalles

L'arithmétique d'intervalles a été introduite par Moore [92] dans les années soixante et a depuis entraîné un champ très riche de recherches[23]. Elle propose un ensemble d'outils et de méthodes pour résoudre, en présence de données ou de calculs incertains, des systèmes d'équations linéaires et non-linéaires. Nous en donnons ici une très courte introduction. On trouvera une présentation plus complète et une vaste bibliographie dans les ouvrages de référence [3, 93].

Le principe fondamental de l'arithmétique d'intervalles est de remplacer les calculs sur les réels par des calculs sur les bornes d'intervalles contenant ces nombres. Nous donnons d'abord un aperçu de l'arithmétique d'intervalles sur les nombres réels et voyons ensuite comment ces résultats peuvent s'adapter au cas de calculs effectués en machine avec des arrondis dirigés.

On définit $\mathbb{I}_\mathbb{R}$ comme l'ensemble des intervalles réels à bornes fermées, et on notera un intervalle X comme :

$$X \equiv [\underline{X}, \overline{X}] := \{x \in \mathbb{R} \quad | \quad \underline{X} \leq x \leq \overline{X}\}.$$

Si S est un sous-ensemble borné de $\mathbb{R}$, on note

$$\Box S = [inf(S), sup(S)]$$

l'enveloppe (ou *hull*) de S.

Les calculs dans l'arithmétique d'intervalles sont effectués à partir des règles suivantes. Pour les opérations élémentaires $\circ \in \Omega = \{+, -, *, /, **\}$, et des fonctions $\varphi \in \Phi = \{\sqrt{}, sqr, sin, cos, tan, atan, \ldots\}$, continues sur tout intervalle de leur domaine de définition noté $Dom(\phi)$, on définit :

$$\forall I, J \in \mathbb{I}_\mathbb{R}, \quad I \circ J = \Box\{i \circ j \quad | \quad i \in I, j \in J\}$$

par la quasi totalité des constructeurs, à l'exception de certaines machines CRAY.

[22] Signalons toutefois que les exigences de cette norme ne suffisent pas à garantir des résultats identiques sur deux machines différentes : même si le résultat d'un calcul est arrondi correctement sur chaque machine, il peuvent être calculés dans des registres de taille différente, ce qui peut entraîner des résultats différents [69].

[23] Neumaier recense déjà plus de 2000 publications sur le sujet en 1990 [93].

et

$$\forall I \in \mathbb{I}_{\mathbb{R}}, \quad \varphi(I) = \square\{\varphi(i) \quad | \quad i \in I\}, \text{ si } I \subseteq Dom(\phi)$$

On peut vérifier que $\forall [a,b],[c,d] \in \mathbb{I}_{\mathbb{R}}$, les propriétés algébriques suivantes sont vérifiées par l'arithmétique d'intervalles:

$$\begin{array}{lcll} [a,b] & + & [c,d] & = & [a+c, b+d] \\ [a,b] & - & [c,d] & = & [a-d, b-c] \\ [a,b] & * & [c,d] & = & [min(ac,ad,bc,bd), max(ac,ad,bc,bd)] \\ [a,b] & / & [c,d] & = & [min(\frac{a}{c},\frac{a}{d},\frac{b}{c},\frac{b}{d}), max(\frac{a}{c},\frac{a}{d},\frac{b}{c},\frac{b}{d})], \text{ si } 0 \notin [c,d] \end{array} \tag{1.1}$$

De façon plus générale, on peut étendre aux intervalles la définition de fonctions arbitrairement compliquées. Pour cela on s'appuie généralement sur la notion suivante :

Définition 1.4. Une fonction $F : \mathbb{I}_{\mathbb{R}}^n \to \mathbb{I}_{\mathbb{R}}$ est une extension aux intervalles de $f : \mathbb{R}^n \to \mathbb{R}$ (appelée aussi fonction d'inclusion de f) si et seulement si

$$\forall I = (I_1, \ldots, I_n) \in \mathbb{I}_{\mathbb{R}}^n \text{ et } (a_1, \ldots, a_n) \in I, \quad f(a_1, \ldots, a_n) \in F(I_1, \ldots, I_n)$$

On obtient l'extension naturelle aux intervalles F d'une fonction f en remplaçant chaque variable réelle par une variable d'intervalle et chaque opération arithmétique ou fonction élémentaire par celle qui lui est associée sur les intervalles.

Une des difficultés majeures que l'on rencontre en arithmétique d'intervalles est liée au problème de dépendance entre les variables d'une expression. Nous l'illustrons sur l'équation $X - X = 0$: avec $X = [0,1]$ on obtient $X - X = [0,1] - [0,1] = [-1,1]$ et non $X - X = [0,0]$. Les calculs perdent l'information relative à la dépendance des variables, et chaque occurrence est considérée comme indépendante ; on calcule donc ici $X - X = \{x - y \mid x \in X, y \in X\}$. En conséquence, l'extension aux intervalles d'une fonction dépend fortement de son expression :

Exemple 1.1. *Pour le carré de la distance entre deux pavés $X_1 \times Y_1$ et $X_2 \times Y_2$:*

$$(X_1 - X_2)^2 + (Y_1 - Y_2)^2 = X_1^2 + X_2^2 + Y_1^2 + Y_2^2 - 2(X_1X_2 + Y_1Y_2)$$

avec $X_1 \times Y_1 = [0,1] \times [0,1]$ et $X_2 \times Y_2 = [1,2] \times [1,2]$. On peut vérifier à partir des formules énoncées en (1.1) que l'on obtient $[0,2]^2 + [0,2]^2 = [0,8]$ pour la forme factorisée de l'équation et respectivement $[0,1]^2 + [0,1]^2 + [1,2]^2 + [1,2]^2 - 2 * [0,8] = [-14,10]$ pour sa forme développée.
Devant ces difficultés on se contente du théorème suivant :

Théorème 1.1. *Si l'expression d'une fonction f ne comporte que des occurrences simples de variables, l'extension naturelle aux intervalles donne l'encadrement exact du domaine de variation de la fonction :*

$$F(D) = \square\{f(v) \mid v \in D\}$$

Il n'est malheureusement pas toujours possible de ramener une équation à cette forme, ce qui nous confronte au problème de trouver une transformation algébrique avantageuse (*i.e.* la plus serrée possible) pour les équations comportant des occurrences multiples d'une même variable. Nous renvoyons à [93] pour une description des principales fonctions d'inclusion et à [112] pour une analyse expérimentale de leur performance.

Notons pour finir qu'il faut se munir de quelques précautions pour transposer des résultats valides sur des intervalles réels à des intervalles de nombre flottants, parce que l'ensemble des nombres flottants n'est pas fermé par les opérations arithmétiques élémentaires. Par exemple, pour a et b dans $\mathbb{F}$ l'ensemble des nombres flottants, on peut avoir $a+b \notin \mathbb{F}$, ce qui entraîne par exemple $a+(b+c) \neq (a+b)+c$.

Cependant, on peut tout de même – moyennant certains tests – implémenter des calculs en ayant à l'esprit la propriété d'inclusion, sans se soucier du détail des opérations de bas niveau, d'autant que des librairies d'arithmétique d'intervalles sont maintenant disponibles dans les principaux langages de programmation et pour la plupart des plateformes.

1.3.3 Contraintes d'intervalles

La notion d'extension aux intervalles d'une fonction que nous venons de voir nous permet d'introduire maintenant les contraintes d'intervalles.

Nous donnons un cadre théorique pour caractériser la sémantique de la résolution des CSPs continus et adoptons le formalisme défini par Benhamou [15]. Il nous permet de considérer ici des contraintes portant sur les domaines numériques ; nous verrons à la Section 1.4 qu'il s'étend naturellement à des domaines hétérogènes.

Contrairement au cas des contraintes discrètes, on ne peut généralement pas traiter directement les contraintes définies par des relations dans $\mathbb{R}^n$. On s'intéresse par conséquent à leur restriction à un domaine d'approximation :

Définition 1.5. A est un domaine d'approximation pour un domaine D s'il est un sous-ensemble de $\mathcal{P}(D)$ fermé pour l'intersection et tel que $D \in A$.

Dans la suite, pour $D = \mathbb{R}$ nous utiliserons notamment $A = \mathbb{F}$ pour les contraintes numériques, et $A = \mathcal{P}(E)$ avec E un sous-ensemble fini connexe de $\mathbb{Z}$ pour les contraintes sur les domaines finis.

Pour une relation $\rho \subseteq \mathbb{R}$, une première alternative opérationnelle est de travailler avec une fonction d'approximation sur A définie comme $\underline{apx}_A(\rho) = \{u \in A \mid \rho \subseteq u\}$. Sa définition se généralise trivialement aux relations n-aires où l'on prend le produit cartésien de la projection $\pi_k(\rho)$ sur chaque composante. L'approximation d'une relation satisfait les propriétés suivantes [18] :

Proposition 1.1. *Soit deux relations ρ et $\rho' \subseteq \mathbb{R}^n$, on a :*

1. $\rho \subseteq \rho' \Rightarrow \underline{apx}_A(\rho) \subseteq \underline{apx}_A(\rho')$
2. $\underline{apx}_A(\underline{apx}_A(\rho)) = \underline{apx}_A(\rho)$
3. $\underline{apx}_A(\rho \cup \rho') = \underline{apx}_A(\underline{apx}_A(\rho) \cup \underline{apx}_A(\rho'))$
4. $\underline{apx}_A(\rho \cap \rho') \subseteq \underline{apx}_A(\rho) \cap \underline{apx}_A(\rho')$

Une seconde alternative est de considérer l'extension aux intervalles d'une relation :

Définition 1.6. Une relation d'intervalles $C : \mathbb{I}^n \to \mathcal{B}$ est une extension aux intervalles de $c : \mathbb{R}^n \to \mathcal{B}$ si et seulement si

$$\forall I = (I_1, \ldots, I_n) \in \mathbb{I}^n \text{ et } (a_1, \ldots, a_n) \in I, \quad c(a_1, \ldots, a_n) \Rightarrow C(I_1, \ldots, I_n)$$

Définition 1.7. Soit c une contrainte définie par une relation $\rho \subseteq D^n$. On appelle opérateur de contraction associé à la relation ρ un opérateur $N_c : A^n \to A^n$ qui pour $\forall u, v \in A^n$ satisfait les propriétés suivantes :

$$\begin{array}{lcll} (1) & N_c(u) \subseteq u & \text{(Complétude)} \\ (2) & \rho \cap u \subseteq N_c(u) & \text{(Correction)} \\ (3) & u \subseteq v \Rightarrow N_c(u) \subseteq N_c(v) & \text{(Monotonie)} \end{array}$$

Proposition 1.2. *Soit c une contrainte définie par une relation $\rho \subseteq D^n$. et une fonction $N : A^n \to A^n$ telle que $\forall u \in A^n$ on a*

$$N(u) = \underline{apx}_A(\rho \cap u)$$

alors N est un opérateur de contraction pour c, et N est idempotent.

On définit un CSP étendu (ECSP) comme un couple

$$E = (S, D)$$

où $S = \{(C_1, N_1), \ldots, (C_m, N_m)\}$ associe les contraintes à leurs opérateurs de contraction, et X est un produit cartésien de domaines appartenant à A.

La sémantique déclarative d'un ECSP, que nous notons E^* est l'ensemble des tuples $x \in X$ satisfaisant l'ensemble des contraintes :

$$E^* = \bigcap_{i=1}^{m} C_i \cap X.$$

Cet ensemble n'étant en général pas calculable, on doit se contenter de la sémantique approchée d'un ECSP. En notant $fp(N_i)$ l'ensemble des points fixes de l'opérateur de contraction N_i, la sémantique approchée de E est définie par :

$$\overline{E} = max(\{u \in \bigcap_{i=1}^{m} fp(N_i) \mid u \subseteq X\}).$$

Elle nous permet de caractériser l'objet calculé par la propagation des contraintes :

Définition 1.8. On dit qu'un ECSP $E = (S, X)$ satisfait la consistance d'arc faible si $\overline{E} = X$.

Il nous reste pour finir à définir un algorithme qui calcule ce plus grand point fixe commun des opérateurs de contraction. C'est l'Algorithme 2, appelé Nar [15] (pour *narrowing*), qui est essentiellement une réécriture de AC-3 [80] (*cf.* p. 10) pour des opérateurs de contraction :

Algorithme 2 : Algorithme de propagation `Nar`

Entrées : $\{(c_1, N_1), \ldots, (c_m, N_m)\}$
Sorties : $D = D_1 \times \ldots \times D_n$
$Q = \{(c_1, \ldots, c_m\}$
tant que $(Q \neq \emptyset) \wedge (D \neq \emptyset)$ **faire**
  $c :=$ une contrainte c_i de Q
  $D' := N_i(D)$
  si $(D' \neq D)$ **alors**
    $Q := Q \cup \{c_j \mid \exists X_k \in Vars(c_j)$ pour laquelle $D'_k \neq D'_k\}$
    $D := D'$
  si N_i *idempotent* **alors**
    $Q := Q \backslash \{c\}$
retourner (D)

Sa particularité est qu'il applique pour chaque contrainte un opérateur de contraction propre à celle-ci. Cet algorithme satisfait les propriétés suivantes :

(1) Si les opérateurs de contraction N_c terminent, l'algorithme termine.

(2) L'algorithme est correct, *i.e.* l'ensemble solution est un sous-ensemble de celui calculé.

(3) `Nar` est confluent : l'ordre dans lequel les opérateurs de contraction sont invoqués n'a pas d'incidence sur le résultat final.

(4) Le résultat du calcul sur E est la fermeture $\overline{E}$.

Opérationnellement, pour des contraintes continues il est souvent plus avantageux de se limiter au calcul d'un « quasi point-fixe » [53] où la modification d'un domaine n'est propagée que si la réduction effectuée est supérieure à un certain seuil. On perd alors les propriétés (3) et (4), mais c'est à l'avantage d'une résolution plus rapide. Autrement, on encourt le coût d'une convergence lente [76] vers le point fixe exact, dans des cas où il serait plus intéressant de procéder à une bissection supplémentaire pour obtenir immédiatement les mêmes déductions.

1.3.4 Opérateurs de contraction génériques

Les opérateurs de contraction de contraintes fournissent l'analogue des algorithmes de consistance génériques développés pour des domaines discrets.

Donnons maintenant un panorama des opérateurs de contraction usuels. Nous distinguons d'abord les formes qui assurent une consistance faible (en appliquant les contraintes une par une), et ensuite celles considérant simultanément plusieurs contraintes à la fois. Nous terminons en donnant une classification selon le degré de contraction des opérateurs.

Les opérateurs de contraction continus visent à obtenir un schéma d'approximation efficace de la consistance d'arc. Calculer la consistance d'(hyper)arcs exacte par des unions d'intervalles est généralement très inefficace [44, 64, 29], et les algorithmes utilisés en pratique (2B [79], HC-4 [16], Box-consistance [17], Bounds [101], BC-4 [49] et BC-5 [51]) calculent donc une approximation plus grossière des relations réelles. Alors que certains opérateurs nécessitent une transformation préalable du système original P en P_{decomp} où les contraintes complexes sont

décomposées en contraintes primitives, d'autres permettent d'éviter de perdre ainsi une partie de la sémantique en se passant de cette réécriture. Nous nous limitons ici à décrire les deux consistances les plus usuelles, à savoir la 2B et la Box-consistance, et terminons en donnant un aperçu des améliorations que l'on obtient en combinant les deux approches.

La première forme de consistance locale pour des contraintes continues est la hull-consistance (aussi appelée 2B-consistance [79]) introduite par Cleary [33]. Étant donné une contrainte continue $c(x_1, \ldots, x_n)$ et un pavé I, on dira que c est hull-consistante sur I si

$$\forall i \in \{1, \ldots, n\}, \quad I_i = \square(\pi_k(\rho_c \cap I_i)) \tag{1.2}$$

De manière opérationnelle on s'intéresse à obtenir

$$\forall i \in \{1, \ldots, n\}, \quad I_i = \underline{apx}_A(\pi_k(\rho_c \cap I_i)) \tag{1.3}$$

Or on ne dispose pas du moyen de calculer cette approximation pour toute relation ; on désigne comme « primitive » une contrainte qui dispose d'un tel opérateur sur le domaine d'approximation A. Une manière de traiter les contraintes non-primitives est de les décomposer en primitives par l'introduction de variables intermédiaires[24]. Cette décomposition peut entraîner la perte des relations de dépendances entre les variables de la contrainte, et en conséquence un système décomposé hull-consistant n'est pas forcément hull-consistant pour la contrainte originelle [34].

Une alternative se passant de la décomposition en contraintes élémentaires est de se contenter d'une forme plus faible de consistance : la Box-consistance [17]. Étant donné c une contrainte n-aire, C une extension aux intervalles de c et un pavé I. On dit que c est Box-consistante par rapport à I si

$$\forall i \in \{1, \ldots, n\}, \quad I_i = \square\{x_i \in I_i \mid C(I_1, \ldots, I_{i-1}, \square\{x_i\}, I_{i+1}, \ldots, I_m)\} \tag{1.4}$$

L'algorithme de Box-consistance BC-3 (par analogie avec AC-3) calcule des domaines box-consistants pour une contrainte $c(x_1, \ldots, x_m)$ en utilisant des projections de contraintes. Il obtient $\forall i \in \{1, \ldots, m\}$ la projection C_{X_i} de C sur X_i par

$$C_{X_i} = C(D_1, \ldots, D_{i-1}, X_i, D_{i+1}, \ldots, D_m) \tag{1.5}$$

L'opérateur de contraction N_c calcule pour tout i le plus grand intervalle $[a, b] \subseteq D(X_i)$ tel que

$$\begin{cases} C_{X_i}([a^-, a)) \\ C_{X_i}((b, b^+]) \end{cases} \tag{1.6}$$

où a^- (resp. b^+) est le plus grand (resp. petit) flottant appartenant à $\mathbb{F}^\infty$ plus petit que a (resp. b). On peut trouver chacune de ces bornes en évaluant C selon un découpage dichotomique de $D(X_i)$. Il est possible de plus d'accélérer dans certains la recherche des bornes en utilisant la méthode de Newton étendue aux intervalles [114].

Entre ces deux formes, l'utilisation de la Box-consistance est avantageuse pour des variables de grande multiplicité, mais elle est en général plus lente sur les variables apparaissant une seule fois. Une stratégie avantageuse est de combiner les opérateurs de hull et de box-consistance, ce

[24] La perte de performances liée à l'introduction de variables supplémentaires peut être évitée avec l'algorithme HC-4 [16]. Il utilise un « arbre syntaxique » de la contrainte qu'il traverse de bas en haut.

que fait l'algorithme BC-4 [49], ou encore BC-5 [51] en appliquant en plus la méthode de Newton.

Tout comme pour les contraintes discrètes, dans certaines situations le filtrage obtenu par ces algorithmes est trop faible pour saisir la dépendance entre les contraintes, et entraîne à force de *splittings* une recherche de solutions dans un arbre très profond. On peut dans ce cas tenter d'appliquer des formes de consistance d'ordre supérieur : alors que les consistances locales d'ordre deux calculent une approximation de la hull-consistance, les consistances d'ordre supérieur portent sur les bornes calculées par ces dernières. Considérons un CSP (C, X, D) 2B-consistant. L'algorithme de 3B-consistance [79] révise à tour de rôle chaque variable X_i et assure que les systèmes $(C \cup \{X_i = \underline{X_i}\}, X, D)$ et $(C \cup \{X_i = \overline{X_i}\}, X, D)$ sont 2B-consistants. Le même principe peut être appliqué à la box-consistance et on obtient alors la Bounds-consistance.

Terminons en clarifiant la relation entre ces différentes consistances. Pour un $P = (\mathcal{V}, \mathcal{D}, \mathcal{C})$ un CSP continu. nous notons $\Phi_{cstce}(P) = (\mathcal{V}, \mathcal{D}^{cstce}, \mathcal{C})$ la fermeture de P par l'algorithme de consistance locale *cstce*. Introduisons la relation d'ordre $\preceq$ définissant $\Phi_{cstce_1}(P) \preceq \Phi_{cstce_2}(P)$ si et seulement si $\forall D_i \in \mathcal{D},\ D_i^{cstce_1} \subseteq D_i^{cstce_2}$. Collavizza *et. al* donnent dans [35] une classification de quelques formes de consistance-locale à laquelle nous ajoutons les formes développées depuis :

$$\Phi_{\text{Bounds}}(P) \preceq \Phi_{\text{3B}}(P_{decomp}) \preceq \begin{cases} \Phi_{\text{Box\_Newton}}(P) \\ \Phi_{\text{BC-4\_Newton}}(P) \end{cases} \preceq \begin{cases} \Phi_{\text{Box}}(P) \\ \Phi_{\text{BC-4}}(P) \end{cases} \preceq \begin{cases} \Phi_{\text{HC-3}}(P_{decomp}) \\ \Phi_{\text{HC-4}}(P) \end{cases} .$$

En plus de la preuve de ces propriétés, on trouvera dans [35] des contre-exemples illustrant les relations d'inclusion.

Dans nos travaux, les contraintes continues traitées sont exclusivement des contraintes de distance euclidienne. Sur nos exemples, la 3B-consistance apporte un filtrage assez fort, mais trop coûteux pour être avantageux. Comme on peut exprimer la distance euclidienne par une formule n'ayant que des occurrences simples des variables, HC-4 est toujours plus efficace que la box-consistance.

1.3.5 Solveurs spécialisés pour des contraintes spécifiques

Au même titre que pour les contraintes discrètes, il est souvent avantageux d'utiliser des solveurs spécialisés pour des contraintes continues particulières. Pour cela, plusieurs schémas de « *coopération ad hoc* » utilisant un algorithme de programmation linéaire ont été proposés.

Nous revenons d'abord aux solveurs de contraintes continus basés sur un solveur linéaire et déclinons les méthodes de collaboration entre des solveurs contraintes continues génériques et un solveur de contraintes linéaires, puis nous présentons comment ces principes s'étendent à des contraintes d'ordre supérieur. Nous nous comparons à cette dernière approche dans nos expérimentations au chapitre 2.

1.3.5.1 Contraintes linéaires

L'utilisation de solveurs de programmation linéaire pour la résolution de contraintes continues en PPC remonte aux premiers systèmes de Programmation Logique avec Contraintes [85]. Ainsi, CLP($\mathcal{R}$) et Prolog III prennent comme solveur de contraintes une version incrémentale de l'algorithme du simplexe.

Comme la restriction à des expressions linéaires limite toutefois sévèrement l'expressivité d'un langage de PPC, beaucoup de travaux sur les contraintes continues se sont tournés vers des techniques de consistance génériques. Or en retour, ces dernières peuvent s'avérer inefficaces pour résoudre un système aussi simple que $\{Y = X,\ Y = -X\}$ avec $D(X) = D(Y) = [-1, 1]$. Bien que ce problème soit rapidement résolu après quelques bissections dans un arbre de recherche, cette déficience peut être un réel handicap sur des problèmes plus grands.

Les deux mécanismes de coopération les plus courants[25] entre des consistances locales génériques continues et un algorithme de programmation linéaire sont :

1. une coopération séquentielle,
2. une coopération concurrente.

La première, la plus simple, laisse chaque solveur atteindre un point fixe, où les solveurs s'échangent de l'information (ajout de contraintes, inconsistance, changement de bornes, etc.). La seconde permet d'incorporer les échanges d'information aussi tôt que possible, et donc par exemple de couper court à des phénomènes de convergence lente.

La nature des informations échangées varie selon les implémentations :

- les bornes inférées pour les contraintes non-linéaires sont généralement répercutées dans le sous-problème linéaire pour les variables partagées,
- le programme linéaire, qui est un solveur complet, calcule si le système est réalisable,
- dans Prolog IV [36] et CIAL [30], le solveur linéaire signale en retour au solveur non-linéaire lorsqu'une variable a été fixée. On trouve dans [37] une discussion des différentes méthodes permettant ce calcul,
- CCC [108] choisit d'effectuer une coopération concurrente avec l'algorithme du simplexe comme solveur de réduction de domaines et propage les bornes inférées aux autres contraintes.[26]

Alors que Prolog IV utilise les algorithme de Gauss-Jordan et du simplexe sur des nombres rationnels et permet de résoudre des équations et inéquations linéaires, CIAL se limite à des équations et procède à la coopération de solveurs par intervalles par la méthode de Gauss-Seidel étendue aux intervalles.

1.3.5.2 Linéarisation de contraintes d'ordre supérieur

Une approche intéressante pour la résolution de contraintes d'ordre supérieur (quadratiques, puissances, etc.) est obtenue en procédant à une reformulation linéaire de ces expressions. Elle permet de capturer la sémantiques de ces contraintes en calculant itérativement une enveloppe linéaire concave (resp. convexe) des termes considérés. Une fois obtenue une expression linéaire des contraintes, on peut réutiliser un algorithme de programmation linéaire pour effectuer les calculs.

Cette méthode, aujourd'hui commune pour la résolution de MINLPs [110], a été introduite par Pesant et Boyer [100] pour des contraintes quadratiques comme sur-couche d'un solveur existant. Elle permet d'enrichir le champ d'application du solveur CLP($\mathcal{R}$), dont le traitement des contraintes non-linéaires se limite à retarder l'évaluation de ces contraintes jusqu'à un stade

[25] La résolution parallèle est une dernière forme de coopération de solveurs, qui combine le résultat des deux résolutions lorsque tous deux ont atteint leur point fixe.

[26] Les auteurs ne fournissent malheureusement pas de benchmarks qui permettent d'estimer des gains possibles en efficacité de cette approche

du calcul (éventuellement jamais atteint) où ces dernières avaient suffisamment de variables instanciées pour que la contrainte soit devenue linéaire.

L'utilisation de reformulations linéaire a plus récemment été généralisée dans l'algorithme de réduction de domaines *Quad* [78] pour

- des contraintes quadratiques et bilinéaires [78],
- des contraintes de puissance, multilinéaires et des fonctions univariées [77].

Quad effectue une reformulation des contraintes qui est spécifiée aux bornes des domaines de leurs variables. L'algorithme du simplexe est appelé sur le système linéarisé pour calculer le maximum et le minimum que chacune des variables peuvent prendre. *Quad* définit ainsi de nouvelles bornes aux variables de ces contraintes, et re-linéarise le système jusqu'à l'obtention d'un point fixe.

Bien que l'utilisation intensive de l'algorithme du simplexe requiert son implémentation avec des nombres flottants, on peut rendre ses résultats sûrs [75]. Précisément, on peut déjà par des arrondis dirigés définir des approximations linéaires extérieures ; ensuite la prise en compte d'un terme correctif proposé récemment par Neumaier [94] pour des programmes linéaires garantit que l'utilisation des calculs du solveur linéaire préserve la correction du calcul.

Cette approche a été utilisée en coopération avec des algorithmes de consistance locale génériques, et améliore substantiellement la résolution d'un grand nombre de benchmarks. Lebbah *et. al* [78] proposent pour cela d'utiliser cet algorithme comme une contrainte globale.

1.3.6 Synthèse

Nous avons vu comment les contraintes d'intervalles permettent de résoudre des contraintes continues de manière fiable. La même conclusion que pour les contraintes discrètes s'impose : les algorithmes génériques de consistance locale autorisent à résoudre un certain nombre de problèmes ; il est pourtant nécessaire d'utiliser des techniques spécialisées – prenant jusqu'ici la forme de coopérations de solveurs – pour résoudre les problèmes de grande taille. En contraste avec les contraintes discrètes, aucune contrainte globale n'a encore été développée pour la résolution de problèmes difficiles. C'est là une des contributions de cette thèse : nous introduisons au Chapitre 2 une contrainte globale de distance euclidienne.

1.4 Approches discrètes-continues en PPC

Alors que les contraintes discrètes et continues ont suscité un nombre important de travaux dans les vingt dernières années, l'hybridation d'approches discrètes et continues est comparativement restée peu étudiée.

L'hybridation de ces techniques de résolution peut être motivée par deux raisons : la recherche de méthodes plus efficaces tirant parti de plusieurs points de vue complémentaires sur un problème, ou plus simplement la nécessité de faire coopérer des algorithmes spécifiques à chaque composante discrète ou continue d'un problème pour le prendre en compte dans sa globalité.

Nous détaillons quelques méthodes proposées pour résoudre des problèmes discrets en tirant parti d'un solveur continu appliqué à une relaxation continue d'un ensemble de contraintes (Section 1.4.1), et donnons ensuite les techniques de résolution des problèmes hybrides faisant intervenir simultanément des variables entières et réelles (Section 1.4.2). Nos travaux s'inscrivent dans chacune des ces deux approches.

1.4.1 Relaxations continues en PPC

Les techniques de relaxation de contraintes sont d'une utilisation commune en Recherche Opérationnelle. Elles consistent à supprimer (temporairement) une partie des contraintes – par exemple la condition d'intégralité des solutions – lorsque ceci permet de définir un problème plus simple à résoudre.

Dans le cadre de la PPC, il n'est généralement pas avantageux d'utiliser les opérateurs génériques de consistance locale continue sur une relaxation continue de contraintes discrètes parce qu'elles calculent une approximation de la consistance d'arc et que leurs algorithmes ne sont pas plus rapides que dans le cas discret. Par contre, du fait d'algorithmes très efficaces pour résoudre des systèmes d'équations ou d'inéquations linéaires, la relaxation linéaire continue de certaines contraintes peut être très utile[27].

Dans un premier temps, nous donnons un aperçu des quelques travaux qui étudient les correspondances à établir entre PPC et programmation linéaire ; ils ne nous semblent difficilement applicables à l'application de notre thèse. C'est pourquoi nous montrons dans un deuxième temps comment on peut aussi exploiter avec profit d'autres types de relaxations continues dans une contrainte globale comme nous le ferons à partir de la contrainte continue *distn* pour résoudre notre problème sur les entiers au chapitre 3.

1.4.1.1 PPC, Programmation Linéaire et en Nombres Entiers

Un solveur de programmation linéaire peut calculer avantageusement des bornes resserrées ou détecter la fixation de certaines variables ou l'absence de solutions au sous-ensemble des contraintes linéaires d'un problème [26, 37]. Aussi, pour les problèmes d'optimisation dont la variable objectif est liée aux variables de décision de façon complexe, la propagation obtenue dans un *Branch & Bound* naïf est souvent faible. Or, pour de nombreux problèmes, on peut obtenir une relaxation continue serrée d'un sous-problème du modèle en contraintes, et par là donner des bornes étroites à la fonction objectif. On peut ainsi supprimer des portions prouvées sous-optimales de l'espace de recherche que l'on n'aurait détectées que tardivement sans cette relaxation. Plusieurs schémas apportant des gains importants d'efficacité ont été proposés pour appliquer cette technique :

- Rodosek [106] propose plus globalement une linéarisation complète du problème discret. Les deux solveurs échangent les bornes des variables partagées et la solution optimale du problème linéaire sert de guide dans la recherche du problème discret.
- Refalo [103] présente un schéma qui opère une relaxation (automatique) de toutes les contraintes globales du problème ; celles-ci sont ensuite traitées par un solveur linéaire séparé. Des relaxations pour de nombreuses contraintes globales ont effectivement été proposées, comme par exemple *alldifferent*, *among*, *element*, *cycle*, *cumulative*,...
- Focacci [45] considère un schéma où des contraintes globales « orientées optimisation » implantent leur propre relaxation du problème – bien que la relaxation continue soit dans ce cas plus faible – comme plusieurs contraintes globales ne partagent alors pas leurs relaxation – cela permet d'introduire des coupes spécialisées additionnelles dans la contrainte.

Ces approches nous semblent pour autant peu applicables pour aborder l'application de déploiement d'antennes qui nous intéresse dans cette thèse. Cette dernière comporte bien un

[27] On notera que ce n'est pas la seule forme de relaxation possible ; on a aussi par exemple la relaxation lagrangienne ou la relaxation discrète [62].

grand nombre de contraintes de valeur absolue et une contrainte de minimum pour lesquelles on peut définir une linéarisation continue, la relaxation de ces contraintes n'est pas serrée et laisse peu d'espoir d'être utile.

Aucune de ces techniques évoquées supra n'a été étudiée dans le cadre de problèmes hybrides discrets-continus, bien que celles-ci s'y appliquent. C'est précisément l'ambition du « cadre général de modélisation en logique pour l'optimisation » proposé par Hooker dans [62] en vue d'enrichir les méthodes d'optimisation par les méthodes de satisfaction de contraintes. Il est plus général que le cadre des travaux décrits supra et prend en compte des problèmes mixtes définis selon plusieurs classes de contraintes :

minimiser	$f(x) + r(y)$	fonction objectif
sujet à	$p_i(y)$	contraintes *vérifiables*
	$g_i(x)$	contraintes *solubles*
	$q_i(y) \rightarrow h_i(x)$	contraintes *dynamiques*
	$d_i(x, y)$	contraintes *définies*
	$x \in X$	variables *solubles*
	$y_j \in D_j, \forall j$	variables *de recherche*

La forme générale d'un problème exprimé dans ce modèle distingue deux types de variables. Les variables *de recherche* – sur lesquelles on doit brancher pour arriver à une solution – sont impliquées dans des contraintes *vérifiables*, dont on peut tester la satisfaction quand l'ensemble de ses variables sont affectées, mais auxquelles on peut sinon seulement assurer un certain degré de consistance (éventuellement par une contrainte globale). Les variables *solubles* sont quant à elles utilisées dans des contraintes plus simples, pour lesquelles on peut calculer directement les valeurs optimales. Le lien entre les deux se fait par des contraintes *dynamiques* pour lesquelles la validation d'une contrainte *vérifiable* entraîne la pose d'une contrainte *soluble* supplémentaire et par des contraintes *définies* portant sur des variables des deux types à la fois. Ces dernières sont tenues de pouvoir s'exprimer formellement comme une conjonction de contraintes *vérifiables*, *solubles* et *dynamiques*. Si en principe, les variables *de recherche* comme les variables *solubles* peuvent être discrètes ou continues, en pratique, les problèmes abordés par Hooker se limitent au cas de contraintes *vérifiables* portant uniquement sur des variables discrètes et de contraintes *solubles* se limitant à des contraintes linéaires ; ils ne nous aident donc pas d'avantage pour aborder nos travaux.

1.4.1.2 Résolution continue dans une contrainte globale discrète

Nous indiquons ici une autre opportunité qu'offre le recours à une relaxation continue à l'intérieur d'une contrainte globale discrète.

Pour certaines contraintes non-linéaires complexes portant sur des variables discrètes, on peut justifier d'utiliser pour sa propagation un algorithme continu générique – par exemple si la contrainte n'est pas disponible dans le solveur discret. Dans le cas d'une résolution dans un solveur de contraintes discret, l'interface d'une contrainte globale permet de gérer la propagation d'un sous-problème continu dans le solveur sans en modifier l'architecture. Plusieurs possibilités s'offrent pour mettre en œuvre une coopération à l'intérieur de la contrainte globale :

- avec un algorithme de filtrage continu *ad hoc*,
- en lançant un solveur de contraintes continu dans la contrainte globale.

Il suffit d'assurer que les résultats donnés en sortie soient ramenés à des entiers. Cette procédure est éventuellement répétée jusqu'à ce qu'un point fixe soit obtenu par ce filtrage.

Cette intégration est profitable pour la contrainte globale de distance euclidienne que nous présentons dans la suite (Chapitre 2). Elle s'appliquerait tout autant à des contraintes bénéficiant d'un algorithme spécialisé de filtrage continu comme *Quad* (*cf.* Section 1.3.5).

1.4.2 Résolution de contraintes hybrides en PPC

Intéressons nous maintenant à la résolution de problèmes hybrides discrets-continus au moyen de solveurs incomplets procédant par réduction de domaine.

Ces problèmes comportent des contraintes discrètes, continues et mixtes (*i.e.*portant sur les deux types de variables à la fois). Plus précisément, nous considérerons des contraintes continues portant sur des variables définies par des intervalles de flottants, et des contraintes discrètes portant sur des variables définies soit par des intervalles d'entiers, soit par des unions d'intervalles entiers. Nous devons exclure d'avance la possibilité d'utiliser des unions d'intervalles comme domaine de calcul pour les contraintes continues, puisque ce choix conduit à des pertes de performance trop importantes [44, 64, 29].

D'un point de vue théorique, la résolution de tels problèmes peut être décrite dans le cadre des « itérations chaotiques » introduit par Apt [4] ou celui de la résolution de contraintes hétérogènes (*cf.* Section 1.3.3) proposé par Benhamou [15]. En utilisant ce dernier, définissons les domaines d'approximation $A = \mathbb{F}$ pour les contraintes continues, et $A' = \mathcal{P}(E)$ pour les contraintes discrètes avec E un sous ensemble fini[28] de $\mathbb{Z}$. Il suffit d'utiliser l'Algorithme Nar (*cf.* p. 21) en transformant les opérateurs de contraction associés aux contraintes pour travailler sur $(A \cup A')^n$. L'application de l'opérateur de contraction pour une contrainte continue C sur $u \in (A \cup A')^n$ se fait via $N_C^{'}$ défini par $N_C^{'}(u) = N_C(\underline{apx}_A(u)) \cap u$ qui est lui aussi un opérateur de contraction. La transformation est symétrique pour les contraintes discrètes. La propagation des contraintes aboutit au calcul du plus grand point fixe commun à l'ensemble des opérateurs de contraction.

Nous prenons donc en compte des considérations qui sont surtout d'ordre pratique. Après avoir donné les principales formes de contraintes mixtes et les techniques de résolutions qui leurs sont associées, nous discutons des problèmes d'implémentation qui se posent.

1.4.2.1 Principales contraintes mixtes discrètes-continues

Les formes principales de contraintes mixtes discrètes-continues que nous avons identifiées pour la modélisation d'un problème hybride sont les suivantes :

1. des contraintes en extension associant des valeurs discrètes à des intervalles réels,
2. des contraintes dynamiques associant des valeurs discrètes à l'introduction de contraintes continues,
3. des contraintes continues utilisant un opérateur de discrétisation, comme par exemple :
   - $X_d = \lceil Y_c \rceil$, la contrainte de valeur supérieure associant une variable discrète X_d et une continue Y_c,
   - $X_d = \lfloor Y_c \rfloor$, la contrainte de valeur inférieure,
   - $X_d = round(Y_c)$, la contrainte de plus proche entier,

[28] Nous supposerons de plus E suffisamment petit pour que $A \cup A'$ soit fermé par intersection, *i.e.* que l'on a $sup\{|z| \mid z \in \mathbb{Z}\} \leq inf\{|f| \in \mathbb{F} \mid f + 1 \in \mathbb{F}\}$.

- $X_d = trunc(Y_c)$, la contrainte de plus proche entier orientée vers zéro,

4. des contraintes globales mélangeant les deux types de variables,
5. des contraintes continues quelconques faisant intervenir des variables discrètes,
6. des combinaison booléennes de contraintes discrètes et de contraintes continues.

Les trois premières apparaissent dans [46] pour des problèmes de configuration et de design conceptuel. L'utilisation de variables discrètes dans des contraintes continues générales est celle que l'on fait classiquement en PLNE. Hooker donne plusieurs exemples d'utilisation de contraintes globales sous une forme mixte dans [62]. Nous introduisons dans cette thèse des disjonctions entre contraintes discrètes et continues pour traiter un problème d'allocation de ressources avec déploiement géographique. Nous donnons ici pour chacun de ces types de contrainte mixtes des exemples d'utilisation :

1. Dans un problème de configuration de cuves de batteurs industriels : on peut rencontrer des contraintes en extension associant le type de cuve à la capacité du récipient. La contrainte $C_1(X_{cuve}, Y_{capacité})$ est donnée par la liste de ses tuples possible :

$$\{(réacteur, [0, 100.0]), (stockage, [0, 1000.0])\}$$

2. Pour le même problème de configuration, le type de récipient utilisé peut influer sur la contrainte définissant son volume :

$$\begin{cases} X_{cuve} = cuve\_sphérique \rightarrow V = \pi * R^3, \\ X_{cuve} = cuve\_cylindrique \rightarrow V = \pi * R^2 * h \end{cases}$$

3. Prenons un problème de design conceptuel de pont : un opérateur de discrétisation peut être utile pour associer une quantité physique à une variable donnant le nombre de ressources :

$$X_{nbPiliers} = \lceil Y_{longueur} / Y_{travée} \rceil$$

4. Une variable discrète peut apparaître dans une contrainte continue quelconque lorsque la notion de choix entre un nombre fini de valeurs intervient. Tous les exemples de PLNE rentrent dans ce cas de figure.
5. La contrainte globale *element* peut être très utile pour la modélisation d'un problème, dans la mesure où elle permet d'utiliser des variables comme indices pour écrire des expressions complexes. Un exemple où une version mixte de cette contrainte est bénéfique est donné dans le problème de voyageur de commerce : si la variable X_k indique la k^{e} ville visitée, la fonction de coût peut s'écrire $\sum_k c_{X_k, X_{k+1}}$. Hooker [62] définit une extension naturelle de cette contrainte[29] de la manière suivante :

$$element(X_d, (c_1, \ldots, c_n), Y_d)$$

où $c_1, \ldots, c_n$ sont des constantes réelles (éventuellement des intervalles) et la contrainte est satisfaite si et seulement si $Y_d = c_{X_d}$

[29] Hooker définit également une extension mixte naturelle pour la contrainte *somme*, dont l'expression est très proche de *element*.

6. Dans un problème de déploiement d'antennes (Chapitre 3), on doit déterminer pour ces antennes une position (dans un espace continu) et une fréquence d'émission (parmi un ensemble discret de fréquences possibles). Des contraintes d'interférence interviennent entre deux antennes S_i et S_j voisines dès lors que les fréquences qui leur sont allouées sont trop proches. Ce problème se traduit par une disjonction entre une contrainte continue et une contrainte discrète :

$$\{distance(S_i, S_j) \geq seuil\} \ \vee \ \{|fréquence(S_i) - fréquence(S_j)| \geq écart\}$$

1.4.2.2 Filtrage des contraintes mixtes discrètes-continues

Il nous reste maintenant à voir comment on peut implémenter un algorithme de filtrage pour ces contraintes.

1. Les contraintes mixtes données en extension ne font intervenir qu'un nombre fini normalement petit de points flottants, suivant lesquels on peut décomposer $\mathbb{R}$ en un ensemble d'intervalles disjoints. On peut donc, moyennant une opération de réécriture invoquée à chaque appel de la contrainte, propager cette contrainte à l'aide des techniques génériques de consistance locale discrète [46].
2. Les contraintes dynamiques sont traitées le plus naturellement en les ajoutant au système lors du parcours de l'arbre de recherche. Nous donnons maintenant une solution pour prendre en compte les deux autres cas de figure.
3. Les contraintes utilisant un opérateur de discrétisation peuvent être approchées (encadrées) par des fonctions continues [46]. Nous les définissons sur le tableau 1.1, en mettant en correspondance une variable discrète et une variable continue. Notons qu'elles requièrent l'implémentation de la relation d'inégalité stricte dans le solveur. Ces opérateurs s'expriment alors en introduisant encore celui d'intégralité donné ci-dessus.

Opérateur	Nom	Approximation
$X_d = \lceil Y_c \rceil$	valeur supérieure	$Y_c \leq X_d < Y_c + 1$
$X_d = \lfloor Y_c \rfloor$	valeur inférieure	$Y_c - 1 < X_d \leq Y_c$
$X_d = round(Y_c)$	valeur arrondie	$Y_c - \frac{1}{2} \leq X_d < Y_c + \frac{1}{2}$
$X_d = trunc(Y_c)$	valeur tronquée	$Y_c - 1 < X_d$ si $Y_c \geq 0$

Tableau 1.1 – Contraintes associant une variable discrète X_d à une variable continue Y_c

4. Lorsqu'une variable discrète apparaît dans une contrainte continue générique, on exprime le plus naturellement cette condition en considérant cette variable X comme étant continue et en introduisant une contrainte supplémentaire de la forme

$$Int(X) = [\lceil \underline{X} \rceil, \lfloor \overline{X} \rfloor],$$

où pour un nombre réel f, $\lfloor f \rfloor$ et $\lceil f \rceil$ désignent le plus grand (*resp.* plus petit) entier plus petit ou égal (*resp.* plus grand ou égal) à f.
Cette approche – introduite dans le langage CLP(BNR) (et reprise à titre indicatif dans Declic et Realpaver) – utilise alors les opérateurs de contraction génériques continus pour la contrainte.

5. Le cadre d'une contrainte globale mixte accorde dans certains cas l'obtention directe d'une propagation optimale sur les deux domaines de calcul. L'adaptation de l'algorithme de filtrage de la contrainte *element* à des variables mixtes est assez immédiat [62].
6. En ce qui concerne les contraintes booléennes $(\vee, \wedge, \neg)$, elles peuvent être retrouvées dans un solveur continu en exploitant la Propriété 1.1 p. 19 [18] avec des variables entières de domaine $\{0, 1\}$. Par exemple la disjonction s'obtient par la relation de maximum :

$$or = max = \{(x, y, z) \in \mathbb{R}^3 \mid max(x, y) = z\}$$

Le filtrage pour cette relation s'obtient à partir de la décomposition suivante :

$$\begin{aligned} max \quad = \quad & \{\{(x, y, z) \in \mathbb{R}^3 \mid x \geq y\} \cap \{(x, y, z) \in \mathbb{R}^3 \mid z = x\}\} \cup \\ & \{\{(x, y, z) \in \mathbb{R}^3 \mid x < y\} \cap \{(x, y, z) \in \mathbb{R}^3 \mid z = y\}\} \end{aligned}$$

Les autres contraintes booléennes peuvent être implémentées de façon analogue.

1.4.2.3 Stratégies de coopération

Il nous importe désormais de voir par quel mode de coopération entre solveurs discrets et continus on peut résoudre un système de contraintes portant sur différents domaines de calcul.

Il peut sembler que l'approche la plus raisonnable pour résoudre un problème hybride discret-continu en PPC est de le plonger entièrement dans le domaine continu, en ajoutant simplement des contraintes d'intégralité aux variables entières du problème. Cependant, cela nous empêcherait d'utiliser des domaines énumérés pour les variables discrètes, or ceux-ci permettent d'améliorer nettement la performance de résolution de certaines contraintes comme par exemple celles de valeur absolue.

A fortiori, contrairement à ce qu'on pourrait attendre, cette approche n'est pas toujours très performante dans la résolution des contraintes non-linéaires portant exclusivement sur des variables discrètes. Par exemple, pour trouver toutes les solutions de l'équation Diophantienne [18] :

$$X^2 + Y^2 + W^2 = Z^2$$

où X, Y, W, Z ont un domaine entier positif borné, la résolution avec le solveur de domaines finis Eclair est au moins dix fois plus rapide qu'avec Realpaver. Les raisons de cet écart peuvent s'expliquer du fait que contrairement à l'algorithme de propagation continu, dans le solveur discret les algorithmes de propagation pour le carré et la contrainte linéaire sont spécifiques à chaque type de modification des domaines (modification de la borne inférieure, supérieure, ou instanciation du domaine).

Nous utilisons pour cela deux solveurs de contraintes séparés qui se coordonnent par l'effet des contraintes mixtes – dans notre cas il s'agit des disjonctions évoquées au dernier point de la section précédente. Le solveur de contraintes de domaines finis Eclair© est utilisé en maître ; celui-ci appelle au besoin un module développé en Claire© implémentant l'algorithme de propagation HC4 pour les contraintes continues. Les contraintes discrètes sont propagées de manière préemptive (*cf.* Section 1.2.2). Lorsque celles-ci modifient le domaine d'une variable présente dans une contrainte mixte, le solveur continu est déclenché et la propagation des contraintes continues s'effectue jusqu'au point fixe, puis rend le contrôle au solveur discret. Ce dernier étudie alors en retour l'effet des modifications effectuées sur les contraintes mixtes. La performance de cette approche et les stratégies et heuristiques utiles dans la recherche de solutions sont développées au Chapitre 3 dans le cadre de notre application de déploiement d'antennes radio.

1.5 Conclusion

Nous avons montré les formes que prennent les approches hybrides discrètes-continues en PPC : la relaxation de contraintes discrètes et la résolution par coopération de solveurs sur des problèmes contenant des contraintes mixtes. La première approche a suscité récemment beaucoup d'intérêt en PPC. Nous utilisons une approche originale en utilisant non pas un solveur linéaire (la seule approche jusque-là) mais une contrainte globale. La seconde approche est bien fondée dans plusieurs cadres théoriques, mais elle a été très peu étudiée en pratique, et nous explorons sa faisabilité sur un nouveau problème.

Nos travaux s'inscrivent sur chacun des deux axes d'hybridation. Ils s'appuient sur *distn*, une contrainte globale continue de distance euclidienne introduite au Chapitre 2. Une première résolution de l'application que nous traitons au Chapitre 3 utilise *distn* sur la relaxation continue du problème. La seconde résolution étend l'espace de solutions d'une partie des variables aux domaines continus et coordonne la résolution de contraintes discrètes et continues par des contraintes mixtes.

CHAPITRE 2

distn : une contrainte globale continue

L'objectif de cette thèse est de proposer une modélisation et une résolution efficace d'un problème de déploiement d'antennes. Au cœur du modèle de cette application traitée au chapitre 3, nous trouvons l'expression d'une clique de contraintes de distance euclidienne, ce qui nous motive à proposer une contrainte globale pour ce sous-problème.

Après avoir donné la sémantique d'une telle contrainte et montré son intérêt plus général, nous exposons une interprétation géométrique du filtrage apporté par des contraintes de distance euclidienne qui nous amène à étudier deux algorithmes de packing de cercles comme algorithmes de filtrage adaptés à la contrainte.

Nous proposons un premier algorithme de filtrage qui réduit les domaines – représentés par des pavés – en des simplexes, afin de les ramener à des pavés plus petits. Ces réductions sont calculées à l'aide d'une extension originale à l'arithmétique d'intervalles d'un algorithme de programmation linéaire de complexité linéaire en le nombre de contraintes.

Nous poursuivons ensuite avec un second algorithme plus complexe qui conserve les simplexes intermédiaires et opère le filtrage directement de polygone en polygone. Cette approche nous a montré que l'utilisation de polygones est finalement plus efficace qu'une approche plus simple basée sur des pavés. Nos contributions à l'amélioration de cet algorithme sont les suivantes :

- Nous proposons une nouvelle caractérisation du filtrage qu'il est possible d'opérer et en dérivons une modification de l'algorithme originel qui élimine certains cas particuliers.
- Nous donnons une adaptation de l'algorithme à des domaines en trois dimensions.
- Nous donnons une analyse de sa complexité algorithmique.
- Nous rendons l'extension aux intervalles de l'algorithme plus simple en utilisant un calcul d'enveloppe convexe.
- Nous un test d'efficience de la procédure de filtrage.

Nous introduisons la généralisation de *distn* à *distn_var*, contrainte dont les distances données par des variables. Son algorithme de filtrage exploite la représentation des domaines par des polygones convexes.

Une série d'expérimentations compare les performances de *distn* à celles des algorithmes de filtrage concurrents.

Introduction

Des problèmes contenant des contraintes de distance euclidienne apparaissent dans de nombreux champs d'application de la PPC, allant de la robotique [88] à la chimie [71], en passant par les bases de données spatiales. Considérant la performance généralement faible des techniques de filtrage classiques sur des problèmes faisant intervenir un grand nombre de contraintes de ce type, plusieurs méthodes ont été proposées pour améliorer l'usage des solveurs de contraintes [78, 71, 8, 9, 59] :

- Krippahl et Barahona [71] proposent de combiner une approximation faible de la contrainte de distance euclidienne avec la contrainte globale *diffn*. Plus précisément, ils relaxent la relation de distance dans la métrique euclidienne vers la métrique de l_∞, en remplaçant la boule euclidienne (distance euclidienne exacte) par les carrés inscrits (distance l_∞ min) et circonscrits (distance l_∞ max) à celle ci.
- Pesant et Boyer [100] et Lebbah *et al.* [78] proposent un algorithme de filtrage, qui de façon plus générale, génère une relaxation linéaire serrée de tous les termes quadratiques et délègue leur traitement à un solveur linéaire. Batnini et Rueher [10] introduisent une décomposition sémantique des contraintes de distance euclidienne qui accélère l'algorithme *Quad* [78].
- Batnini [7] introduit systématiquement des contraintes redondantes données par les inégalités triangulaires et des relations faisant intervenir le barycentre de triplets de points, pour accélérer l'algorithme de 3B-consistance.

Cependant ces approches comportent encore des limitations. Elles apportent dans la première approche une approximation parfois très faible ; les suivantes ont une complexité qui croit très rapidement avec la taille des systèmes.

Nous proposons ici une contrainte globale n-aire qui considère les relations d'inter-distance entre n points. Nous utilisons ce point de vue pour mettre en œuvre des raisonnements géométriques qui permettent de résoudre ces limitations.

Nous présentons d'abord les motivations applicatives de la contraintes globale *distn* (Section 2.1) et poursuivons en donnant les motivations géométriques pour introduire une contrainte globale (Section 2.2). Un premier algorithme de filtrage est d'abord introduit (Section 2.3), suivi de l'algorithme que nous avons retenu pour *distn* (Section 2.4) et pour son extension à *distn_var* (Section 2.5). Nous terminons avec une série d'expérimentations (Section 2.6) et une conclusion (Section 2.7).

2.1 Motivations applicatives

L'intérêt d'une contrainte globale réside en ce qu'elle fournit un élément à un langage de contraintes pour modéliser et résoudre plus aisément des applications complexes. Aussi, le développement de *distn* a été motivé par un problème de localisation que nous avons rencontré dans une application industrielle.

Nous commençons ici par donner la sémantique de notre contrainte globale (Section 2.1.1) et poursuivons en présentant (Section 2.1.2) comment elle s'accorde à ses champs d'application les plus intéressants.

2.1.1 Sémantique de la contrainte *distn*

Intéressons nous au problème de résoudre un système de contraintes entre n points. Au lieu de considérer la clique des contraintes de distance isolément, nous proposons une modélisation du problème par le biais d'une contrainte globale qui les traite simultanément. Nous verrons comment cette approche permet de tirer parti de la géométrie du problème pour développer l'algorithme de filtrage de la contrainte. Nous décrivons d'abord la sémantique déclarative de la contrainte *distn* dans sa forme la plus simple.

La syntaxe de la contrainte *distn* est donnée par

$$distn(\texttt{list}_{<\texttt{CVar}>} : [\texttt{P}_1, \ldots, \texttt{P}_\texttt{n}] \ , \ \texttt{matrix}_{<\texttt{double}>} : \texttt{d} \ , \ \texttt{matrix}_{<\texttt{double}>} : \texttt{D}) \tag{2.1}$$

où $P_i = (X_i, Y_i)$ est le produit cartésien de variables continues de domaine (resp. des k-tuples quand on est en k dimensions), d et D sont des matrices symétriques positives ou nulles de taille $n \times n$, contenant les écarts minimaux et maximaux tolérés.

La contrainte (2.1) est vérifiée si et seulement si

$$\forall (i,j), \quad \begin{cases} \forall p_i = (x_i, y_i) \in P_i, \ \forall j, \ \forall p_j = (x_j, y_j) \in P_j & dist(p_i, p_j) \geq d_{i,j} \\ \forall p_i = (x_i, y_i) \in P_i, \ \forall j, \ \forall p_j = (x_j, y_j) \in P_j & dist(p_i, p_j) \leq D_{i,j} \end{cases} \tag{2.2}$$

i.e. si et seulement si pour tout couple de points pris dans P_i et P_j, la distance entre ces points est supérieure à $d_{i,j}$ tout en restant inférieure à $D_{i,j}$.

Lorsqu'on traite un problème pur de contraintes de distance maximale (resp. minimale), la matrice D (resp. d) doit être initialisée à $+\infty$ (resp. 0).

La contrainte peut être utilisée également pour des problèmes d'optimisation. Elle prend alors un argument supplémentaire : la variable (continue) d'optimisation o de domaine $[\underline{o}, \overline{o}]$:

$$distn(\texttt{list}_{<\texttt{CVar}>} : [\texttt{P}_1, \ldots, \texttt{P}_\texttt{n}] \ , \ \texttt{matrix}_{<\texttt{double}>} : \texttt{d} \ , \ \texttt{matrix}_{<\texttt{double}>} : \texttt{D} \ , \ \texttt{CVar} : \texttt{o}) \tag{2.3}$$

La variable o représente la dilatation (linéaire) appliquée aux matrices de distances, et cette nouvelle contrainte est vérifiée si et seulement si

$$\forall (i,j), \quad \begin{cases} \forall p_i = (x_i, y_i) \in P_i, \ \forall j, \ \forall p_j = (x_j, y_j) \in P_j & dist(p_i, p_j) \geq \underline{o} \ . \ d_{i,j} \\ \forall p_i = (x_i, y_i) \in P_i, \ \forall j, \ \forall p_j = (x_j, y_j) \in P_j & dist(p_i, p_j) \leq \overline{o} \ . \ D_{i,j} \end{cases} \tag{2.4}$$

i.e. si et seulement si les points contraints sont en accord avec les bornes en cours de la variable d'optimisation.

Nous verrons enfin, pourquoi il peut être intéressant de généraliser *distn* à *distn_var*, où les matrices d et D sont remplacées par une matrice de variables de distance. La syntaxe de cette dernière forme est donnée par :

$$distn\_var(\texttt{list}_{<\texttt{CVar}>} : [\texttt{P}_1, \ldots, \texttt{P}_\texttt{n}] \ , \ \texttt{matrix}_{<\texttt{CVar}>} : \texttt{dvar}) \tag{2.5}$$

et cette contrainte est vérifiée si et seulement si

$$\forall i \neq j \quad dist(p_i, p_j) = dvar_{i,j} \tag{2.6}$$

i.e. si et seulement si les points contraints sont en accord avec les variables de distance.

2.1.2 Différents domaines d'application de *distn*

Nous montrons ici comment des applications faisant intervenir des points mis en relation par des contraintes de distance peuvent aisément être modélisés par la contrainte *distn*.

Le problème académique dit de "packing de cercles" occupe d'abord notre propos. Étant donné que l'algorithme de filtrage de *distn* est inspiré d'un algorithme de packing de cercles, il est assez naturel que la contrainte globale puisse être utilisée pour modéliser élégamment ce problème d'optimisation géométrique. Nous nous tournons ensuite vers des problèmes d'intérêt industriel, nommément ceux appartenant à la classe des problèmes d'analyse de localisation, de conformation moléculaire, de robotique et de design d'expérimentations numériques.

2.1.2.1 Packing de cercles

Un packing de n cercles de rayon d dans un carré unité est donné par les coordonnés des centres des cercles, dans une position où ils sont contenus dans le carré et ne s'intersectent pas. Il est dit maximal lorsqu'il n'existe pas de packing de rayon supérieur pour n cercles.

Ce problème a été abondamment traité par les communautés d'arithmétique d'intervalles [83], d'optimisation globale [82], de recherche locale [50], des méta-heuristiques [63], de géométrie discrète [87], et de géométrie combinatoire [97].

Le problème est équivalent à celui de maximiser la distance minimale entre n points placés dans un carré [111]. En effet, la donnée de n points p_i satisfaisant la contrainte peut être ramenée de façon bijective aux centres des cercles. L'instanciation suivante de la contrainte (2.3) permet de modéliser le problème de satisfaction.

$$\begin{aligned} &\mathit{distn}([\mathtt{P_1},\ldots,\mathtt{P_n}],\mathtt{d},+\infty,\mathtt{o}) \\ &\quad\begin{cases}\mathtt{d_{i,j}} = \mathtt{1}, & \forall i \neq j \\ \mathtt{d_{i,i}} = \mathtt{0}, & \forall i\end{cases}\end{aligned}$$

La matrice d spécifie les couples de points à maintenir à une distance minimale de o ; nous avons ignoré les contraintes de distance maximale en ajustant les valeurs de la matrice D à $+\infty$.

La recherche d'un packing optimal peut être envisagée[1] en combinant un algorithme de *Branch & Prune* avec un algorithme de *Branch & Bound* sur la variable o : le domaine initial de o sera alors $[d_{min}, 1]$, où d_{min} est une borne inférieure connue du rayon maximal du problème.

2.1.2.2 Analyse de Localisation

L'Analyse de Localisation est une branche de la Recherche Opérationelle qui rassemble l'étude des problèmes visant à localiser des objets dans l'espace de façon optimale vis à vis d'une certaine fonction de coût. Pour nommer quelques applications étudiées, précisons qu'elle s'intéresse à la distribution d'installations aussi diverses que des entrepôts, des succursales d'une entreprise, des arrêts de bus, des écoles, des relais téléphoniques ou des décharges sur la trame urbaine.

Différentes grandes classes de problèmes ont été identifiées et caractérisées dans divers modèles discrets et continus [41]. Parmi celles-ci, un certain nombre d'articles [66, 14] abordent le

[1] Les approches existantes pour résoudre ce problème nécessitent toutefois de plus des techniques sophistiquées de brisure de symétrie dans l'exploration de l'arbre de recherche.

problème de localisation d'installations dites indésirables ou nuisibles[2]. Ces installations peuvent être par exemple des émetteurs radio, des centrales électriques ou des décharges. L'enjeu est de trouver un emplacement à ces équipements qui minimise leur impact négatif sur les villes voisines, des parcs naturels, etc.

Différents modèles physiques sont employés pour estimer la nuisance d'un site en fonction de sa position, tant dans un cadre discret que continu. Nous utilisons ici le modèle de "maximin", appelé aussi "problème de p-dispersion" par cette communauté.

Ce problème est principalement traité dans un cadre discret [41], où un nombre fini de sites candidats est prédéfini, et certaines installations sont déjà établies. Toutefois, dans certaines situations, ce problème peut s'avérer aussi intéressant à étudier dans un espace continu. La seule étude expérimentale publiée, se trouve dans [14, 67], où l'emplacement de deux nouvelles installations est localisée par le biais d'algorithmes de géométrie algorithmique.

Considérons le problème suivant : étant donné une région bornée R, et un ensemble de k points de demande (villes) $\{p_1, \ldots, p_k\}$ dans R. Déterminer le lieu de $(n-k)$ installations indésirables $\{p_{n-k+1}, \ldots, p_n\}$ dans les limites de R, avec comme contrainte que leur inter-distance doit être au moins r. De plus, on souhaite maximiser la distance minimale entre les installations et les points de demande.

Nous pouvons exprimer ces conditions avec le modèle suivant :

$$
\begin{array}{l}
\mathit{distn}([\mathtt{p_1}, \ldots, \mathtt{p_k}, \mathtt{P_{k+1}}, \ldots, \mathtt{P_n}], \mathtt{d}, +\infty, \mathtt{o}), \\
\begin{cases}
\mathtt{d_{i,j}} = 0, & \forall\ \mathtt{1} \leq \mathtt{i}, \mathtt{j} \leq \mathtt{k} \\
\mathtt{d_{i,j}} = 0, & \forall\ \mathtt{k+1} \leq \mathtt{i}, \mathtt{j} \leq \mathtt{n} \\
\mathtt{d_{i,j}} = 1 & \text{sinon}
\end{cases}
\end{array}
$$

$$
\begin{array}{l}
\mathit{distn}([\mathtt{P_{k+1}}, \ldots, \mathtt{P_n}], \mathtt{d'}, +\infty), \\
\begin{cases}
\mathtt{d'_{i,j}} = \mathtt{r}, & \forall i \neq j \\
\mathtt{d'_{i,i}} = 0, & \forall i
\end{cases}
\end{array}
$$

La première contrainte permet de modéliser le sous problème d'optimisation par le biais de la variable d'optimisation o et de l'encodage de la structure du problème dans la matrice d. Les points de demande sont représentés par des variables déjà instanciées. Des degrés variables de nuisibilité suivant les installations peuvent éventuellement être considérés en adaptant les facteurs de dilatation de la matrice d.

La seconde contrainte traduit la contrainte supplémentaire d'inter-distance minimale de r entre installations nuisibles. La restriction à R des positions des installations est simplement posée dans les domaines des variables P_i, où $k+1 \leq i \leq n$.

Si par ailleurs la donnée de l'association des installations aux villes est connue à l'avance (lorsque les installations doivent être tout de même relativement proches des villes comme dans le cas d'une usine), on peut traiter le contexte d'installations dites semi-nuisibles, en spécifiant aussi la matrice D.

La résolution peut alors être effectuée sur la base de ce modèle par un algorithme classique de *Branch & Bound*. L'aspect le plus important de la formulation d'un problème de localisation en

[2] *undesirable* ou *obnoxious facilities* en anglais.

contraintes est que nous pouvons facilement modifier le modèle en ajoutant ou retirant d'autres contraintes, qui autrement ne pourraient pas être intégrées aussi aisément dans un algorithme de recherche dédié de RO. Hamacher [56] rapporte que suivant l'expérience de la communauté d'analyse de localisation, la possibilité d'intégrer des contraintes annexes aux formulations classiques est généralement déterminante dans le succès de l'application de cette technologie.

On peut par exemple coupler le problème de localisation que nous venons de décrire avec celui d'allocation de fréquences ; c'est précisément l'application que nous étudions au Chapitre 3, où les installations à localiser comportent des antennes radio. Nous pensons pour cette raison que le cadre de modélisation de la PPC est particulièrement adapté pour la formulation de problèmes d'Analyse de Localisation, et que la contrainte *distn* offre un premier composant pour étudier cette riche famille de problèmes.

2.1.2.3 Conformation Moléculaire

La conformation moléculaire est un sujet de recherche très actif en biologie moléculaire et d'une importance majeure pour la médecine et les biotechnologies. Le problème posé consiste à déterminer précisément la configuration spatiale (la forme) de protéines. Les chercheurs peuvent définir leur fonction grâce à cette connaissance, qui se rapporte à l'analogie de clef et de serrure ; la compréhension de la structure moléculaire permet entre autres de déterminer des médicaments potentiels.

Alors qu'environ 100000 séquences de protéines sont connues, une détermination structurelle de seulement 10% d'entre elles est disponible. Le procédé principalement utilisé jusqu'à ce jour est la cristallographie aux rayons X, qui requiert une étape malaisée de cristallisation. L'analyse de résonance moléculaire nucléaire (NMR[3]) est une technique alternative qui croît actuellement en importance.

Les propriétés chimiques de la molécule impliquent que la longueur des liaisons entre certains atomes est connue, éventuellement de façon approximative. Mais comme beaucoup d'atomes laissent cours à des rotations le long de certaines liaisons, un degré trop élevé de liberté subsiste pour qu'il soit possible de déterminer la forme précise d'une protéine.

Les tests de NMR fournissent comme donnée complémentaire les paires d'atomes qui doivent être proches l'un de l'autre. La conjonction de ces deux informations donne le moyen d'établir la position de tous les atomes, et donc d'estimer la forme précise des conformations possibles de la molécule.

Les logiciels commerciaux existants n'utilisent à ce jour que des algorithmes d'optimisation pour approcher la structure de molécules, et leurs temps de calculs peuvent être très longs. Sachant que les mesures issues d'une étude par NMR peuvent être entâchées d'erreurs et risquent donc de fournir des données erronées, il est nécessaire de procéder à plusieurs expérimentations pour déterminer correctement la structure de la molécule.

Une approche en PPC est intéressante parce qu'elle autorise à éliminer au plus tôt ces tests de l'espace de recherche. Krippahl et Barahona [71] proposent de combiner une résolution en PPC avec de la recherche locale, et parviennent à obtenir des résultats similaires à ceux de logiciels d'optimisation sans contraintes, mais dans une fraction de leur temps de calcul.

[3] Nuclear Molecular Resonance.

Pour des raisons physiques, les noyaux ne peuvent pas se chevaucher, et imposent donc des contraintes de distance minimale et maximale entre eux. Les données de tests par NMR offrent de surcroît d'autres contraintes de distance, détaillant quels groupes d'atomes doivent être voisins. Comme les contraintes de distance euclidienne peuvent se révéler inefficaces pour résoudre des problèmes de grande taille, Krippahl [71] propose un modèle de PPC, qui utilise l'approximation suivante pour obtenir un filtrage rapide : la boule euclidienne est remplacée par son carré approximant. Plus précisément, les contraintes *In* (distance maximale) et *Out* (distance minimale) sont distinguées et remplacées par une approximation extérieure (par le carré circonscrit) et intérieure (par le carré inscrit) de la métrique euclidienne.

La sémantique de *distn* correspond aussi à cette situation : La contrainte *In* (resp. *Out*) est encodée dans la matrice d (resp. D). Toutes ces relations sont englobées dans la contrainte globale suivante :

$$distn([\mathtt{P_1}, \ldots, \mathtt{P_n}], \mathtt{d}, \mathtt{D}).$$

où pour $i = 1, \ldots, n$ les $P_i = (X_i, Y_i, Z_i)$ sont les produits cartésiens des variables de domaine continu. Des contraintes annexes, comme l'angle de torsion de certaines grappes d'atomes complètent ensuite le modèle.

2.1.2.4 Autres applications

Nous évoquons ici, sans rentrer toutefois dans les mêmes détails, deux autres domaines d'applications d'importance industrielle :

- Les problèmes de robotique exigent souvent de résoudre des systèmes d'équations non-linéaires pour lesquels aucune solution explicite n'est connue. Certains comportent des systèmes d'équations de distance, pour lesquels les consistances locales classiques requièrent beaucoup de *splittings*, et génèrent un grand nombre de boîtes ne comportant pas de solutions.
  La plateforme de Gough-Stewart [78] en est une illustration. Il s'agit d'un robot articulé, composé d'une plateforme mobile reliée à un socle fixe par six jambes télescopiques. On voudrait pour certaines configuration pouvoir déterminer la position de la plateforme suivant la longueur des jambes.
  Nous verrons plus loin, dans notre série d'expérimentations (Section 2.6), un autre exemple de manipulateur planaire.
- Les contraintes de distance ont aussi leur importance pour déterminer des statistiques relatives au design d'expérimentations numériques. Booker étudie dans [24] le cas d'une expérience de simulation des temps de réponse aéroélastiques et dynamiques d'une pale de rotor d'hélicoptère qui nécessite six heures de calcul sur un Cray Y-MP.
  L'enjeu d'un design d'expérimentations numériques est de remplacer des évaluations de fonction coûteuses par une évaluation pour des points de l'espace méticuleusement choisis. Une configuration de points sélectionnés de manière à maximiser leur inter-distance minimale est appelée un design maximin de distances par cette communauté [113, 40]. Ils sont perçus comme un critère désirable, du fait de leur capacité à "remplir" l'espace.

La résolution de tels problèmes avec une contrainte globale de distance, offrant un algorithme de propagation efficace, se présente ici encore comme un outil de modélisation souhaitable.

2.1.3 Synthèse

Nous avons vu sur quelques exemples, pourquoi la résolution des contraintes de distance euclidienne était centrale dans des problèmes applicatifs issus de domaines très divers, et comment la contrainte *distn* permet de les modéliser élégamment.

La prochaine section illustre sur un exemple les faiblesses des consistances locales classiques pour des contraintes de distance, et présente à cette occasion les améliorations nous proposons d'y apporter.

2.2 Motivations géométriques

Nous donnons ici les motivations géométriques pour introduire une contrainte globale de distance euclidienne en illustrant l'interprétation géométrique que l'on peut faire du filtrage obtenu.

Après avoir donné une caractérisation formelle du filtrage que l'on souhaite obtenir, nous étudions sur des exemples le cas d'une contrainte prise entre deux points, et montrons ensuite comment le filtrage justifie de s'intéresser à des consistances d'ordre supérieur.

2.2.1 Caractérisation du filtrage par des régions permises et interdites

Les contraintes de distance euclidienne autorisent une description géométrique intuitive du problème de satisfaction qu'elles posent.

Nous proposons pour cela une caractérisation du filtrage en termes de « régions d'exclusion » [13] appelées ici « régions interdites » et de régions « permises ». En anticipant sur la section 2.4, ces aspects sont définis en considérant plus généralement des contraintes de distance prises entre des points dont le domaine est défini par un polygone convexe. Cela nous mène à introduire d'abord une notion, appelée "Noyau de Rayon d d'un polygone", qui détermine la région d'exclusion d'une contrainte de distance minimale posée entre deux polygones convexes, puis nous définissons la Somme de Minkowski qui donnera le support pour des contraintes de distance maximale.

Définition 2.1. Soit une contrainte binaire $C(P_i, P_j)$ définie sur des domaines polygonaux $P_i \times P_j \subseteq \mathbb{R}^2 \times \mathbb{R}^2$.

On appelle $R_i \in \mathbb{R}^2$ une région interdite de la contrainte C par rapport à la variable P_j, si $\forall r_i \in R_i \cap P_i$ et $\forall r_j \in P_j$, le couple (r_i, r_j) viole la contrainte C. Nous noterons cette région $RI(P_j)$.

Inversement, on appelle $R_i \in \mathbb{R}^2$ une région permise de la contrainte C par rapport à la variable P_j, si $\forall (r_i, r_j) \in P_i \times P_j$, le couple (r_i, r_j) satisfait la contrainte C seulement si $r_i \in R_i$. Nous noterons cette région $RP(P_j)$.

Dans la suite, pour $x \in E$ un espace euclidien et $d \in \mathbb{R}$ nous utiliserons la notation $B(x, d)$ pour désigner la boule $\{y \in E \ / \ dist(x, y) \leq d\}$.

Définition 2.2. Le Noyau de Rayon d d'un Polygone convexe P, que nous noterons $N(P,d)$, est défini comme

$$N(P,d) = \{y \in \mathbb{R}^2 \ / \ y \in \bigcap_{x \in P} B(x,d)\}$$

i.e. comme l'intersection de toutes les boules $B(x,d)$ pour $x \in P$.

Cette définition nous intéresse parce qu'elle permet de caractériser le support d'une contrainte de distance euclidienne minimale.

Proposition 2.1. *En autorisant une représentation des domaines par des polygones, la région interdite de P_i pour la contrainte $dist(P_i,P_j) \geq d_{i,j}$ est donnée par*

$$RI(P_i) = P_i \cap N(P_j, d_{i,j}).$$

Preuve : La preuve est immédiate : si $p_i \in RI(P_i)$, alors $\forall p_j \in P_j,\ dist(p_i,p_j) \leq d_{i,j}$, et donc $p_i \in N(P_j,d_{ij})$. Inversement, si $p_i \in P_i \cap N(P_j,d_{i,j})$ alors $\exists p_j \in P_j$ tel que $p_i \in B(p_j,d_{i,j})$ et on a $support(p_i) = \emptyset$.
■

La région permise pour une contrainte de distance maximale est quant à elle définie à l'aide de la somme de Minkowski :

Définition 2.3. La Somme de Minkowski de deux ensembles A et B dans l'espace vectoriel $\mathbb{R}^2$ est définie comme l'ensemble $\{a+b : a \in A,\ b \in B\}$ et sera notée $A \oplus B$.

Dans toute la suite, nous notons $B(0,d)$ la boule centrée en l'origine et de rayon d :

$$B(0,d) = \{(x,y) \in \mathbb{R}^2 \mid x^2 + y^2 \leq d\}$$

Proposition 2.2. *En autorisant une représentation des domaines par des polygones, la région permise de P_i pour la contrainte $dist(P_i,P_j) \leq d_{i,j}$ est donnée par*

$$RP(P_j) = P_j \oplus B(0,d_{i,j}).$$

2.2.2 Filtrage obtenu par une seule contrainte

Élaborons maintenant la correspondance directe que l'on peut établir entre ces considérations d'ordre géométrique et le filtrage obtenu par la 2B-consistance pour les contraintes de distance euclidienne.

Considérons pour cela deux boîtes $\mathrm{P}_1 = X_1 \times Y_1$ et $\mathrm{P}_2 = X_2 \times Y_2$ qui représentent le domaine initial de points p_1 et p_2 que nous souhaitons placer à une distance $d \in [d_{min}, d_{max}]$ l'un de l'autre. La Figure 2.1 illustre le filtrage obtenu sur un exemple comportant une seule contrainte $dist(P_1,P_2)$ avec $d_{min} = d_{max}$.

La région « interdite » due à la contrainte de distance minimale est représentée en gris clair. Elle peut être construite comme l'intersection de quatre disques de rayon d_{min} centrés en chacun des coins de la boîte. En effet, pour tout point pris dans cette région, l'ensemble de tous les points pris dans P_2 se situent à une distance inférieure à d_{min}. C'est le noyau de rayon d du pavé P_2.

La région « permise » pour la distance maximale (l'union des régions en gris foncé et en gris clair) est quant à elle décrite par l'intérieur de la boîte P_2 à laquelle on ajoute la surface balayée

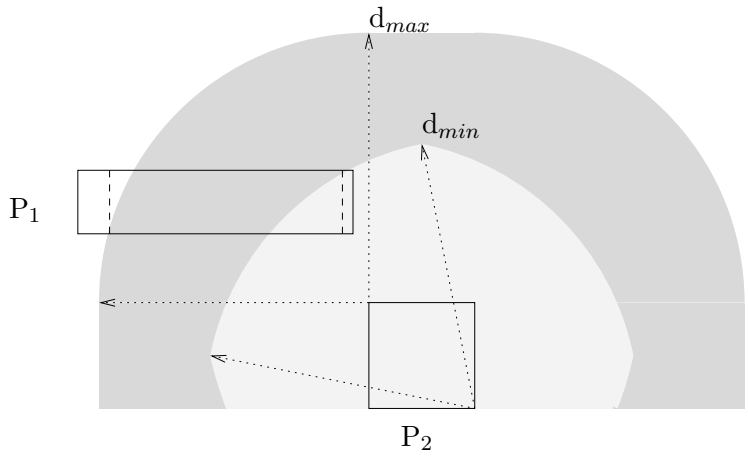


Figure 2.1 – Exemple du filtrage opéré par P_2 sur P_1 avec la 2B-consistance

par un disque de rayon d dont on fait parcourir le centre tout le long des côtés de la boîte. Autrement dit, on l'obtient par une translation – de longueur d_{max} et dirigée vers l'extérieur – de chacun des côtés du rectangle et en complétant ce contour par des quarts de cercles centrés aux coins. En effet, un point à une distance inférieure à d_{max} dans P_2 n'existe que dans ces limites. Il s'agit précisément de la somme de Minkowski de la boîte avec une boule centrée en l'origine et de rayon d_{max}

En considérant simultanément ces contraintes, on voit qu'un point $p_1 \in P_1$ ne les satisfait que s'il est à la fois compris dans l'aire en gris foncé et en dehors de celle donnée en gris clair sur la figure.

Un calcul par intervalles permet d'isoler directement l'invariant de consistance aux bornes maintenu par ces contraintes. Pour la distance minimale on a [83] :

$$X_i^k = \Box(X_i^k \setminus [\overline{X_j} - \sqrt{d^2 - \overline{(Y_i - Y_j)^2}}, \underline{X_j} + \sqrt{d^2 - \overline{(Y_i - Y_j)^2}}]) \tag{2.7}$$

Pour la distance maximale on a :

$$\begin{array}{rl} & (X_1 - X_2)^2 + (Y_1 - Y_2)^2 \leq d^2 \\ \Rightarrow & (X_1 - X_2)^2 \leq d^2 - \underline{(Y_1 - Y_2)^2} \\ \Rightarrow & |X_2 - X_1| \leq \sqrt{d^2 - \underline{(Y_1 - Y_2)^2}} \end{array}$$

d'où les invariants :

$$\begin{cases} X_2 \leq \overline{\overline{X_1} + \sqrt{d^2 - \underline{(Y_1 - Y_2)^2}}} \\ X_2 \geq \underline{\underline{X_1} - \sqrt{d^2 - \underline{(Y_1 - Y_2)^2}}} \end{cases} \tag{2.8}$$

Les bornes calculées sur P_1 par la 2B-consistance sont tracées en pointillés. Dans une certaine mesure, l'estimation obtenue par la 2B est satisfaisante en ce que ces bornes sont exactes. Ceci s'explique du fait que la formule donnant la contrainte de distance euclidienne peut s'exprimer en ne faisant intervenir que des occurrences simples de variables, ce qui permet de calculer exactement la consistance aux bornes (*cf.* Section 1.3.4).

On voit par contre qu'il reste une portion importante du domaine de P_1 dont on sait déjà qu'elle ne peut contenir de solutions, mais qu'on ne peut pas filtrer du fait de la représentation des domaines par un produit de variables indépendantes. Nous montrons maintenant comment cette information peut être exploitée si l'on considère plusieurs contraintes de distance à la fois.

2.2.3 Filtrage obtenu en considérant plusieurs contraintes

L'intérêt du cadre d'une contrainte globale est qu'il permet de considérer simultanément plusieurs contraintes à la fois dans le filtrage.

L'exemple qui motive notre approche est le suivant : considérons trois boîtes $P_i = X_i \times Y_i$ où $i = 1, 2, 3$ qui représentent le domaine initial de points que nous souhaitons placer avec une inter-distance minimale de $d_{i,j}$. Le système d'inéquations considéré s'écrit :

$$\begin{cases} dist(P_1, P_2) \geq d_{2,1} \\ dist(P_1, P_3) \geq d_{3,1} \\ dist(P_3, P_2) \geq d_{2,3} \end{cases}$$

Nous avons représenté sur la Figure 2.2, avec X (resp. Y) dans la direction horizontale (resp. verticale) le filtrage obtenu par des algorithmes de consistance locale classiques ordinairement disponibles dans les solveurs de contraintes par intervalles, et sur la Figure 2.3 celui produit par d'autres algorithmes de propagations spécialisés.

2.2.3.1 Consistances locales classiques

Les solveurs de contraintes par intervalles proposent habituellement quelques formes de 2 ou 3 consistances :

- Sur le schéma de gauche de la Figure 2.2, on peut voir qu'avec une représentation des domaines par un produit de deux variables, la 2B-consistance ne parvient à éliminer qu'une portion très faible de la boîte à réduire. Ceci est dû à ce qu'elles ne peuvent pas considérer l'effet de plusieurs contraintes prises simultanément.
- On peut néanmoins assurer une forme plus forte de consistance locale avec la 3B-consistance [79]. Une analyse détaillée de son mécanisme de filtrage montre que par des diminutions successives de la borne droite de X_1, illustrée par un segment vertical à droite sur la Figure 2.2, l'algorithme finit par atteindre le point limite, où l'effet simultané des deux contraintes de distance résulte en une solution vide pour la 2B-consistance sur le CSP $P \cup \{X_1 = \overline{X_1}\}$. Le problème est que rien ne guide l'algorithme vers cette borne, et qu'un grand nombre d'évaluations successives peut être nécessaire pour atteindre cette limite.

Alors que la propagation obtenue par la 3B-consistance est assez satisfaisante dans cet exemple, son coût de calcul peut la rendre inutilisable sur des problèmes de plus grande taille.

2.2.3.2 Consistances locales obtenues dans une contrainte globale

Des formes moins conventionnelles de représentation des domaines nous permettent d'approcher les secteurs angulaires suggérés par la Figure 2.2 :

- Krippahl et Barahona [71] proposent de remplacer le filtrage de distance euclidiennes par celui obtenu grâce à une approximation carrée intérieure et extérieure de la boule unité

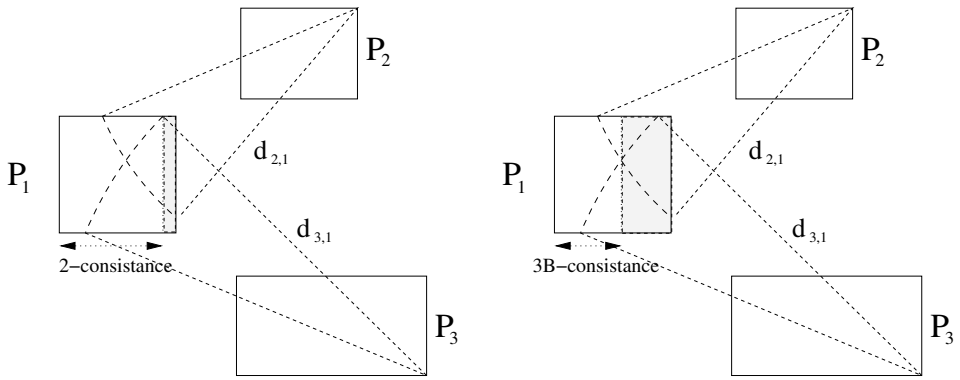


Figure 2.2 – Filtrage obtenu sur P_1 par la 2B et la 3B consistance pour la distance-min.

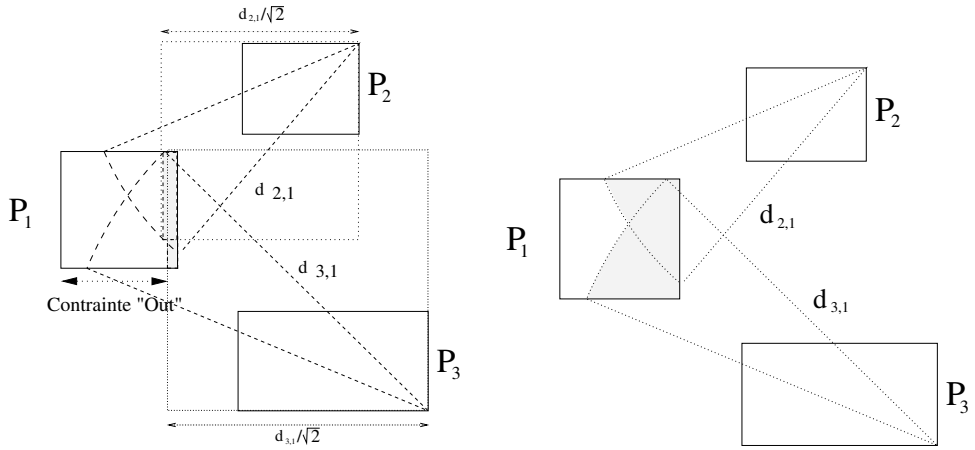


Figure 2.3 – Filtrage obtenu sur P_1 par la contrainte *Out* et région d'exclusion approximée par la Méthode des Aires Actives.

euclidienne. Pour la contrainte *Out*, cette reformulation est donnée ici en deux dimensions par

$$\sqrt{(X_i - X_j)^2 + (Y_i - Y_j)^2} \geq d_{i,j} \quad \Rightarrow \quad \{|X_i - X_j| \geq \frac{d_{i,j}}{\sqrt{2}} \quad \wedge \quad |Y_i - Y_j| \geq \frac{d_{i,j}}{\sqrt{2}}\}$$

Bien qu'il ne soit pas présenté comme tel, leur schéma correspond à ce que l'on attend d'une contrainte globale qui tire pleinement parti de la sémantique du problème traité. En plus de ce filtrage, un calcul des "no-goods" est effectué pour évaluer l'effet de toutes les régions d'exclusion prises simultanément. Cet algorithme est rapporté comme étant très efficace sur des instances de grande taille (de l'ordre de 300 points) en utilisant un espace discrétisé ; les résultat obtenus sur notre illustration ne sont cependant pas serrés (sur le schéma de gauche de la Figure 2.3).

- Peikert et De Groot ont observé [39] dans le cadre de la résolution du problème de packing de cercles qu'il serait plus efficace de supprimer directement du domaine de P_1 la totalité de l'aire grisée sur le schéma de droite de la Figure 2.3). Cette observation est à l'origine d'une part importante des travaux de la littérature traitant de ce problème, où elle est désignée comme la "Méthode des Aires Actives"[4].

Ce dernier schéma est au cœur de l'algorithme de filtrage de notre contrainte globale. Pour cette raison, nous donnons à présent un aperçu de cette méthode.

2.2.4 La méthode des aires actives

L'algorithme que nous proposons étend les travaux de plusieurs auteurs [39, 99, 96, 83, 84] sur le problème de packing de cercles (Section 2.1.2.1). L'idée originelle de De Groot *et. al* [39] pour obtenir un packing optimal de n points dans un carré unité est d'utiliser un algorithme de *Branch & Bound* pour déterminer le rayon maximal d_{max}. Le problème de satisfiabilité pour un rayon d donné est effectué en combinant une technique de pavage avec un algorithme de *Branch & Prune* classique en PPC.

Une difficulté majeure de ce problème de recherche de points est que l'espace de recherche pour chaque point couvre l'ensemble du carré. Or pour toute solution donnée dans le carré, il existe $n!$ solutions équivalentes par renommage des points, et encore 4 autres solutions obtenues en la transformant par des rotations du carré. La seule manière connue de briser ce problème de symétrie pour le problème de packing de cercles est de recourir à un pavage préalable de l'espace.

Supposons qu'on cherche à montrer la satisfiabilité d'une (nouvelle) borne inférieure d_{inf} au rayon optimal de packing. La méthode commence alors par départager le carré en des pavés dont la diagonale est de longueur inférieure à d_{inf}. La distance minimale entre deux points quelconques pris dans une tuile[5] est alors plus petite que d_{inf}. Une tuile contient donc au plus un point dans une solution au problème de packing.

La méthode génère pour cela en premier toutes les combinaisons de n tuiles prises parmi le pavage, en omettant toutes celles qui sont équivalentes. Un ordre lexicographique sur les tuiles permet d'éliminer toutes les configurations similaires à un renommage près. On peut de plus éliminer encore ce faisant les configurations déjà générées qui sont équivalentes par rotation.

[4] L'appellation de "Method of Active Areas" a été reprise dans les articles suivant celui de Peikert *et. al* [39].

[5] Une tuile est un pavé de dimension deux.

Si cette prémisse est acceptable dans le problème de packing de cercles, pour aborder des problèmes de conformation moléculaire (Section 2.1.2.3) ou d'analyse de localisation (Section 2.1.2.2), cette étape de pavage initial n'est pas envisageable parce que les contraintes de distance peuvent être toutes distinctes. Une tessellation naturelle de cet espace de recherche ne peut alors pas être extraite. Heureusement, les points sont généralement confinés dans un certain voisinage, si bien que l'on peut espérer que la taille de l'espace de recherche reste modérée.

Une fois le pavage de l'espace déterminé, il faut réduire par un algorithme de filtrage toutes les n-tuiles à des n-tuiles plus petites (éventuellement les éliminer ou les ramener à des points) pour extraire celles qui contiennent un packing de rayon supérieur à d_{inf}. Un algorithme de *Branch & Prune* classique – dont le filtrage est l'objet de nos recherches – détermine si la combinaison de tuiles courante peut contenir un packing de rayon d_{inf}.

Deux variétés de procédures de filtrage ont été développées pour l'algorithme de *Branch & Prune* mis en œuvre dans la "Méthode des aires actives". Elles visent toutes deux à considérer simultanément les contraintes appliquées à un pavé, et partagent de ce fait essentiellement une même technique de propagation. L'une utilise des domaines de type rectangulaire, l'autre les représente par des polygones. Chacune donne la base des algorithmes de filtrage pour la contrainte *distn* que nous développons dans les deux prochaines sections.

2.3 Un premier algorithme de filtrage pour *distn*

Cette section détaille les travaux liés à un algorithme de filtrage que nous avons proposé pour la contrainte globale *distn*.

Nous commençons par détailler l'algorithme de propagation de la Méthode des Aires Actives utilisant des domaines rectangulaires. Nous détaillons ensuite l'amélioration que nous proposons pour les contraintes de distance minimale et maximale. Ces travaux nous ont conduit à introduire une extension aux intervalles de l'algorithme de programmation linéaire de Seidel [109] que nous donnons alors. Enfin, un bilan indique les raisons qui nous ont fait choisir finalement un autre algorithme pour *distn*.

2.3.1 Méthode des aires actives appliquée à des domaines rectangulaires

L'algorithme de Markót [83], que nous présentons ici, est une variation de l'algorithme originellement proposé par De Groot *et. al* [39] pour effectuer la réduction des domaines dans la méthode des aires actives[6].

En partant d'une combinaison de n tuiles, il peut soit extraire la région contenant (potentiellement) une solution au système de contraintes de distance minimale posé, soit déduire qu'il n'existe pas de solutions dans la combinaison courante.

L'algorithme de propagation coordonne l'action d'une fonction de filtrage pour prendre en compte successivement l'ensemble des contraintes. Cette fonction est donnée par l'invariant (équivalent à l'action de la 2B-consistance) précisé par l'équation (2.7). Or nous avons vu à

[6] L'algorithme originel décompose chaque pavé P_i selon un quadrillage P_i^k. En considérant P_i, l'ensemble des carreaux P_i^k tels que $\forall k'\ dist(P_i^k, P_j^{k'}) < d$ sont supprimés du sous pavage de P_i

la section précédente que ce filtrage n'était malheureusement pas très efficace pour prendre en compte plusieurs contraintes simultanément. L'idée utilisée ici pour arriver à cette fin est de décomposer temporairement le pavé à réduire en des rubans horizontaux puis verticaux, d'appliquer le filtrage sur la décomposition, de retenir enfin le filtrage commun à tous les rubans.

Algorithme 3 : $MinDistancePruneX_{rect}$

Entrées : $\{P_i^k\}_{k=1,\ldots,t}$ un t-sous-pavage du rectangle P_i, et P_j le rectangle réducteur
Sorties : $\{P_i^k\}_{k=1,\ldots,t'}$ l'ensemble des sous-pavés non éliminées par P_j

pour $k = 1$ ***to*** t **faire**
  $X_i^k = \square(X_i^k \setminus [\overline{X_j} - \sqrt{d^2 - \overline{(Y_i - Y_j)^2}}, \underline{X_j} + \sqrt{d^2 - \overline{(Y_i - Y_j)^2}}])$

L'algorithme de propagation met en œuvre ce filtrage de la manière suivante : un prétraitement extrait pour chaque pavé P_i la liste de ses « voisins » qui peuvent filtrer une partie de son domaine, *i.e.* la liste des pavés P_j qui ne satisfont pas trivialement l'inéquation :

$$\overline{dist(P_i, P_j)} > d.$$

Une stratégie de round-robin analogue à AC-1 [80] est alors adoptée pour approcher (après $MaxIterations$) le point-fixe de la propagation des contraintes de distance. L'algorithme de propagation considère tour à tour chaque pavé P_i, le décompose temporairement au moyen d'une fonction $TempSubTiling$ en un ensemble de sous-pavés $\{P_i^k\}_{k=1,\ldots,t}$ auxquels on applique le filtrage $MinDistancePrune$ opéré par chacun des voisins appartenant à $Voisins(P_i)$. L'opération $BoîteEnglobante$ ramène ensuite le domaine P_i à la boîte englobante de l'ensemble des sous-pavés qui n'ont pas été éliminés.

Algorithme 4 : $Method\_of\_Active\_Areas_{rect}$

Entrées : Une combinaison de n tuiles
Sorties : Cette combinaison réduite par le filtrage

pour $i := 1$ ***to*** n **faire** $P_i = X_i \times Y_i$
pour $i := 1$ ***to*** n **faire** déterminer $Voisins(P_i)$
pour $k := 1$ ***to*** $MaxIterations$ **faire**
  pour $i := 1$ ***to*** n **faire**
    $\{P_i^k\}_{k=1,\ldots,t} := TempSubTilingX(P_i)$
    pour $j \in Voisins(P_i)$ **faire** $\{P_i^k\} = MinDistancePruneX_{rect}(\{P_i^k\}, P_j, d_{i,j})$
    $P_i = BoîteEnglobante(P_i^k)$
    $\{P_i^k\}_{k=1,\ldots,t} := TempSubTilingY(P_i)$
    pour $j \in Voisins(P_i)$ **faire** $\{P_i^k\} = MinDistancePruneY_{rect}(\{P_i^k\}, P_j, d_{i,j})$
    $P_i = BoîteEnglobante(P_i^k)$

Si un des domaines devient vide au cours de ce processus, l'algorithme déduit qu'il n'existe pas de solutions dans la n-tuile courante.

Une particularité importante de cet algorithme par rapport à AC-1 réside dans sa procédure de révision (*i.e* de filtrage) des domaines que nous appellerons $MinDistancePrune$. Son intérêt est d'être mise en oeuvre sur des domaines de points considérés en deux dimensions, alors que les consistances locales classiques opèrent de façon « aveugle ».

Nous proposons dans ce qui suit une amélioration aux Algorithmes 4 et 3, qui au lieu de considérer des partitions de la tuile courante, calcule directement une approximation de son aire active par un simplexe [59].

2.3.2 Premier algorithme proposé pour *distn*

Cette section décrit l'algorithme de filtrage que nous proposons pour la propagation des contraintes de distance minimale et maximale.

Nous motivons d'abord notre choix d'approximer l'aire active par un simplexe, puis nous procédons à une étude de cas qui permet de poser les inéquations définissant l'aire active pour des contraintes de distance minimale puis maximale. Nous terminons en nous assurant de la possibilité de garantir la correction des calculs.

2.3.2.1 Approximation de l'aire active par un simplexe

Nous proposons une amélioration à l'Algorithme 4, qui au lieu de considérer un sous pavage de la tuile courante, approche l'aire active par un simplexe et permet ainsi d'obtenir directement et rapidement une approximation serrée de la région active en considérant simultanément l'ensemble des contraintes portant sur un pavé.

Pour illustrer notre approche, nous reprenons sur la Figure 2.4 l'exemple de la Figure 2.1 qui donnait des régions « interdites » et « permises » pour les points de P_1. Nous calculons ici l'intersection de ces régions avec le pavé à réduire à partir d'un système d'inéquations linéaires définies par une étude de cas. La contrainte de distance minimale est approchée par un arc, la distance maximale est définie dans ce cas par trois tangentes.

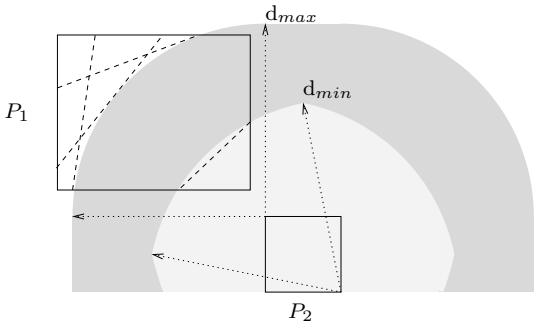


Figure 2.4 – Filtrage opéré par P_2 sur P_1 avec le premier filtrage de *distn*

Ce filtrage – qui est plus serré – n'a en soi pas d'intérêt par rapport à la 2B-consistance si on considère une seule contrainte, par contre il peut être cumulé en appliquant le même raisonnement aux autres contraintes posées sur P_1 et ainsi approcher l'effet global de toutes ces contraintes en définissant l'aire active de P_1 par un simplexe.

Un exemple illustrant cette idée pour des contraintes de distance minimale est donné sur la Figure 2.5. La tuile courante y est "entourée" par trois tuiles voisines, et l'effet des régions d'exclusion qu'elles engendrent est illustré par des arcs. Les cordes tracées pour approcher les arcs sont données sur la Figure 2.6, et l'on voit que la 2B-consistance ne permet pas de prendre en compte la région d'exclusion créée par P_3, alors qu'elle est prise en compte en représentant le domaine de P_1 par un simplexe.

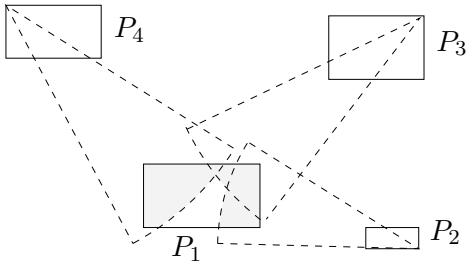


Figure 2.5 – Filtrage possible depuis les voisins de la tuile courante

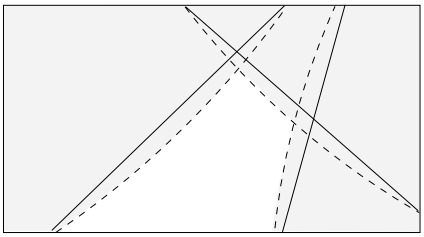


Figure 2.6 – Approximations linéaires des arcs

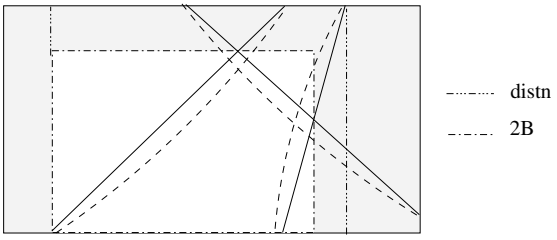


Figure 2.7 – Boîte englobante minimale du simplexe calculé

Une fois le simplexe connu, on peut obtenir ses bornes par un solveur linéaire et retourner sa boîte englobante pour propager ensuite cette modification. La réduction des domaines obtenues par le simplexe est présentée sur la Figure 2.7 où nous la comparons au filtrage de la 2B-consistance.

Une étude de cas donne l'expression des inéquations définissant le filtrage des contraintes de distance minimale et maximale est proposée en annexe p. 135.

2.3.2.2 Correction des calculs

Dans l'approximation que nous faisons de la contrainte de distance minimale ou maximale, nous voulons éliminer exclusivement les points de la zone grisée, et devons donc nous assurer que les demi-espaces calculés sont bien dans cette limite.

Définition 2.4. On dira qu'un calcul est sûr s'il n'élimine aucun point solution.

Bien que les calculs impliqués dans l'approximation par un simplexe peuvent sembler plutôt simples, l'obtention de résultats justes lors des opérations sur les nombres flottants est essentielle dans les trois étapes suivantes :

1. les points d'intersection doivent être sûrs,
2. les équations des demi-espaces obtenus doivent être sûres,
3. le solveur linéaire doit être fiable.

Ces trois points sont traités de la manière suivante :

1. Le premier est résolu en utilisant les arrondis dirigés de l'arithmétique d'intervalles pour calculer les points d'intersection des arcs avec les pavés : on choisit la borne de l'intervalle à l'extérieur des arcs (resp. à l'intérieur) pour la distance maximale (resp. minimale).
2. Le second point est analogue : en calculant le coefficient directeur de la pente des droites supportant ces deux points, l'arithmétique d'intervalles doit être utilisée, et une autre étude de cas détermine alors quelle borne de l'intervalle il faut conserver.
3. Le dernier comporte deux aspects :
   (a) les solveurs linéaires n'offrent pas de garantie sur la correction de leurs résultats[7],
   (b) les solveurs linéaires sont lents sur des problèmes de si petite taille.

Cette dernière difficulté nous a amené à développer une adaptation d'un algorithme de programmation linéaire qui résout ces deux aspects.

2.3.3 Extension aux intervalles de l'algorithme de Seidel

Des problèmes de géométrie algorithmique de dimension 2 ou 3 ont motivé l'étude d'algorithmes plus performants que ceux utilisés classiquement en programmation linéaire : l'algorithme du simplexe est de complexité exponentielle (en le nombre d'inéquations) dans le pire cas. Dyer [43] et Megiddo [86] ont l'un et l'autre découvert de façon indépendante que l'on pouvait résoudre les programmes linéaires en temps linéaire en n (le nombre d'inéquations) avec

[7] Dans [94] Neumaier et Shcherbina montrent un problème assez simple de programmation linéaire en nombre entiers (PLNE) qui met en échec la plupart des solveurs de PLNE existants à cause des problèmes d'instabilité numérique dans les calculs sur les nombres flottants.

une complexité doublement exponentielle en d (le nombre de variables). Par la suite, la borne du terme exponentiel a par la suite été encore améliorée par des algorithmes randomisés. Celui proposé par Seidel [109], dont la complexité en moyenne est $\mathcal{O}(nd!)$, présente en outre l'avantage d'être particulièrement simple.

Suite à nos expérimentations peu probantes avec les solveurs linéaires LPsolve et XPressMP, nous avons choisi de produire et mettre en oeuvre une implantation de l'algorithme de Seidel qui offre des garanties de robustesse des résultats. Étant donné que nous recherchons des bonnes performances dans cette implantation et que nous pouvons nous contenter d'une approximation extérieure, nous avons décidé d'utiliser le paradigme de l'arithmétique d'intervalles plutôt que celui du calcul exact.

Considérons le calcul de la boîte englobante du simplexe de la Figure 2.7 : Pour trouver des bornes sûres de la boîte, nous devons chercher une approximation extérieure du simplexe. Pour cela nous tolérerons donc de violer éventuellement certaines contraintes, auquel cas le point solution se trouvera bien dans l'espace dual. De façon générale, en considérant un problème de maximisation, nous voudrons obtenir au pire un point qui surestime la fonction objectif.

Nous utiliserons les notations suivantes pour notre description de l'algorithme : Étant donnés le vecteur $c = (c_1, \ldots, c_d) \in \mathbb{R}^d$, deux vecteurs l et $u \in \mathbb{R}^d$ tels que $l_i \leq u_i$ pour $1 \leq i \leq d$, un ensemble A de vecteurs de dimension $d+1$ et un point $x \in \mathbb{R}^d$, le programme linéaire consiste à maximiser le produit scalaire $c.x$ en respectant

$$\sum_{1 \leq i \leq d} a_i x_i < a_{d+1} \quad \text{pour } a = (a_1, \ldots, a_{d+1}) \in A \text{ et}$$
$$l_i \leq x_i \leq u_i \qquad \text{pour } 1 \leq i \leq d.$$

Nous proposons une extension aux intervalles de l'algorithme de Seidel (Algorithme 6). Cette réécriture diffère de l'algorithme originel [109] sur ses hypothèse de travail : les inéquations du programme linéaire peuvent être indécidables.

Ainsi, à chaque point de choix, si un test d'inconsistance est validé sur les intervalles, la même décision que pour des calculs réels est prise. Sinon, on se trouve dans une position de *statu quo*, et l'algorithme continue en conservant la valeur optimale précédente, ce qui permet de rester dans l'espace dual dans la suite des calculs.

Algorithme 5 : Algorithme randomisé de Seidel étendu aux intervalles (cas $d = 1$)

```
   LP (d, c, l, u, A)
 1 sup ← min { {a2/a1 | a ∈ A, (a1)⁻ > 0} ∪ {u1} }
 2 inf ← max { {a2/a1 | a ∈ A, (a1)⁺ < 0} ∪ {l1} }
 3 pour {i | (a1^i)⁻ ≤ 0 ≤ (a1^i)⁺} faire
 4     si (a1^i.x)⁺ > (a2)⁻ alors
 5         retourner pas de solution
 6 si sup < inf alors
 7     retourner pas de solution
 8 sinon
 9     si (c)⁻ > 0 alors retourner sup
10     sinon si (c)⁺ < 0 alors retourner inf
11     sinon retourner garder la solution courante
```

Algorithme 6 : Algorithme randomisé de Seidel étendu aux intervalles (cas $d > 1$)

LP (d, c, l, u, A)
// Calculer la solution x optimale par rapport aux contraintes l et u.
pour $1 < i < d$ **faire**
  si $(c_i)^- > 0$ **alors** $x_i = u_i$
  sinon si $(c_i)^+ < 0$ **alors** $x_i = l_i$
  sinon si $c_i = 0$ **alors** $x_i = l_i$
  sinon retourner *garder la solution courante*
// Ajouter les contraintes de A, une par une.
$B \leftarrow \{\}$
tant que $A \neq \{\}$ **faire**
  Choisir au hasard une contrainte $a \in A$ et la retirer de A
  // Si x viole certainement la nouvelle contrainte.
  si $(\sum_{1 \leq i \leq d} a_i x_i)^- > (a_{d+1})^+$ **alors**
    Soit $1 \leq k \leq d$ maximal tel que $(a_k)^+ \neq 0$
    si $\nexists$ *de tel* k **alors**
      retourner *pas de solution*
    sinon si $a_k \cap 0 \neq \emptyset$ **alors**
      retourner *garder la solution courante*
    // Éliminer la variable x_k des contraintes de B et de c.
    $A' \leftarrow \{(b - \frac{b_k}{a_k} a)$ avec la k^{e} composante supprimée $\mid b \in B\}$
    $c' \leftarrow (c - \frac{c_k}{a_k} a)$ avec la k^{e} composante supprimée
    si $\exists\ i\ /\ (c_i)^- \leq 0 \leq (c_i)^+$ **alors**
      retourner *garder la solution courante*
    sinon
      // Incorporer les contraintes $l_k < x_k < u_k$ dans A'.
      Soit f le (d+1)-vecteur avec $f_k = 1, f_{d+1} = u_k$ et sinon $f_i = 0$
      Soit g le (d+1)-vecteur avec $g_k = -1, g_{d+1} = -l_k$ et sinon $g_i = 0$
      $A' \leftarrow A' \cup \{(f - \frac{1}{a_k} a), (g - \frac{1}{a_k} a)$ avec k^{e} composante supprimée$\}$
      // Générer de nouvelles bornes pour les variables.
      Soit l' le (d - 1)-vecteur obtenu de l en omettant k^{e} composante
      Soit u' le (d - 1)-vecteur obtenu de u en omettant la k^{e} composante
      // Résoudre le problème à $(d-1)$-dimensions et relever la solution.
      Soit $x' = LP(d-1, c', l', u', A')$
      si $x' \neq$*"garder la solution courante"* **alors**
        Soit x le d-vecteur obtenu de x' en insérant 0 à la k^{e} composante
        $x_k = \frac{1}{a_k}(a_{d+1} - \sum_{1 \leq i \leq d} a_1 x_i)$
  $B = B \cup \{a\}$
retourner X

2.3.4 Synthèse

Nous avons présenté un algorithme de filtrage pour les contraintes de distance minimale de *distn*. Il s'agit d'une amélioration à la méthode des aires actives appliquée à des domaines rectangulaire. Son originalité est d'approximer directement l'aire active par un simplexe obtenu par une étude de cas. Le filtrage est obtenu sur ce simplexe par un programme linéaire sûr et de complexité linéaire en le nombre d'inéquations.

Nous avons cependant abandonné cet algorithme, au profit de la méthode des aires actives utilisant des polygones car Markót et Csendes [84] ont montré récemment qu'il était possible d'adapter la méthode polygonale au paradigme de l'arithmétique d'intervalles, et la comparaison avec cette nouvelle approche a démontré un gain important de performances. Cette dernière permet d'obtenir une propagation non seulement plus forte, mais aussi plus rapide. C'est pourquoi nous avons poursuivi nos travaux sur la base d'approximations polygonales de la région active.

2.4 Algorithme de filtrage de *distn*

Les algorithmes de filtrage de la contrainte globale *distn* sont détaillés dans cette section en utilisant maintenant une représentation polygonale des domaines. Nous produisons d'abord l'algorithme de filtrage pour des contraintes de distance minimale, puis celui proposé pour les contraintes de distance maximale.

2.4.1 Algorithme de filtrage de *MinDistance*

Nous présentons à nouveau les principes de la Méthode des Aires Actives, mais appliquée cette fois à des domaines polygonaux. L'algorithme à l'origine de ces travaux est décrit ; nous en donnons une analyse de complexité. Nous proposons alors une caractérisation du filtrage qu'il est possible d'opérer et en dérivons une modification de l'algorithme originel. Ce nouvel algorithme permet une adaptation plus aisée à des domaines en trois dimensions et une extension immédiate à l'arithmétique d'intervalles. Nous terminons en introduisant quelques améliorations à l'algorithme de propagation.

2.4.1.1 Méthode des aires actives appliquée à des domaines polygonaux

La méthode à l'origine des développements de la section précédente a évolué d'un filtrage des partitions de la tuile pour déterminer l'aire inactive (la zone grisée sur le schéma de droite de la Figure 2.3) vers une procédure plus sophistiquée basée sur une approximation de l'aire active par un polygone introduite par Nurmela et Östergård [95]. L'algorithme de propagation de la Méthode des Aires Actives pour des domaines polygonaux (Algorithme 7) reste essentiellement inchangé vis à vis de l'algorithme de propagation pour des domaines rectangulaires. La seule différence est que l'on se passe du calcul de la boîte englobante du simplexe (des polygones) dans les étapes intermédiaires (ligne 8 de l'Algorithme 4) ; cette évaluation est différée jusqu'à l'obtention d'un quasi point-fixe dans la propagation.

Algorithme 7 : $Method\_of\_Active\_Areas_{poly}$

Entrées : Une combinaison de n tuiles
Sorties : Cette combinaison réduite par le filtrage
1 **pour** $i := 1$ ***to*** n **faire** $P_i = Polygone(X_i \times Y_i)$
2 **pour** $i := 1$ ***to*** n **faire** déterminer $Voisins(P_i)$
3 **pour** $k := 1$ ***to*** $MaxIterations$ **faire**
4 **pour** $i := 1$ ***to*** n **faire**
5 **pour** $j \in Voisins(P_i)$ **faire**
6 $P_i = Réduction(P_i, P_j, d_{i,j})$
7 $X_i \times Y_i = BoîteEnglobante(P_i), \quad \forall i$

La représentation des domaines par des polygones permet un filtrage plus rapide et plus fort. Ceci est dû au fait que les opérations peuvent s'effectuer directement de polygone à polygone, ce qui évite des sur-approximations intermédiaires.

2.4.1.2 Un algorithme de réduction de domaines polygonaux

Commençons par expliquer le fonctionnement de la fonction de réduction de domaines en supposant l'ensemble des calculs comme étant exacts.

Pour chaque polygone – et la boîte rectangulaire de départ est effectivement un polygone convexe – l'algorithme proposé par Nurmela [96] élimine pour chacun de ses polygones voisins, l'aire induite par les contraintes de distance minimale[8]. On réduit ainsi la taille des polygones à chaque pas jusqu'à ce qu'un quasi point-fixe soit atteint. Cette méthode se fonde sur les lemmes et théorèmes suivants :

Lemme 2.1 (Lemme 1 de [96]). *Si un point p est à une distance inférieure à d de tous les sommets d'un polygone P, alors il est à une distance inférieure à d de tous les points de P.*

Théorème 2.1 (Théorème 1 de [96]). *Supposons que $p_{i_1}, p_{i_2}, \ldots, p_{i_k}$ sont des points distincts sur le bord de P_i, que $(p_{i_s}, p_{i_{s+1}})$ pour $2 \leq s \leq k-2$ sont des arêtes de P_i, et que (p_{i_1}, p_{i_2}) et $(p_{i_{k-1}}, p_{i_k})$ se trouvent sur des arêtes de P_i. Si les points $p_{i_s}, 1 \leq s \leq k$, sont à une distance inférieure à d de tous les sommets de P_j, alors tous les points dans le polygone $(p_{i_1}, p_{i_2}, \ldots, p_{i_k})$ obtenu sont à une distance inférieure à d de tous les points de P_j.*

Un algorithme de réduction de domaines découle assez naturellement du théorème précédent. Nous le donnons sur l'Algorithme 8 accompagné d'une illustration de son déroulement sur la Figure 2.8 ; aussi le présentons nous maintenant.

Chaque sommet de P_i prend une étiquette '+' ou '–' indiquant s'il est à une distance supérieure ou inférieure à $d_{i,j}$ de tous les sommets de P_j ; si tous les sommets de P_i sont marqués '+' (resp. '–'), aucune réduction de P_i n'est possible (resp. l'absence de solutions est signalée). Sinon, on commence par étiqueter les sommets du polygone réduit et on extrait une chaîne maximale $(p_{i_2}, \ldots, p_{i_{k-1}})$ étiquetée '–'. Deux points $p_{1'}$ (resp. $p_{k'}$) sont alors déterminés aussi loin que possible de $p_{2'}$ (resp. $p_{k-1'}$) sur l'arête reliant ces points à un sommet étiqueté '+', de telle façon qu'ils soient toujours à une distance inférieure à $d_{i,j}$ de P_j. D'après le Théorème 2.1,

[8] Cette fonction est reprise et appelée *Diminish_ij* par Markót [84].

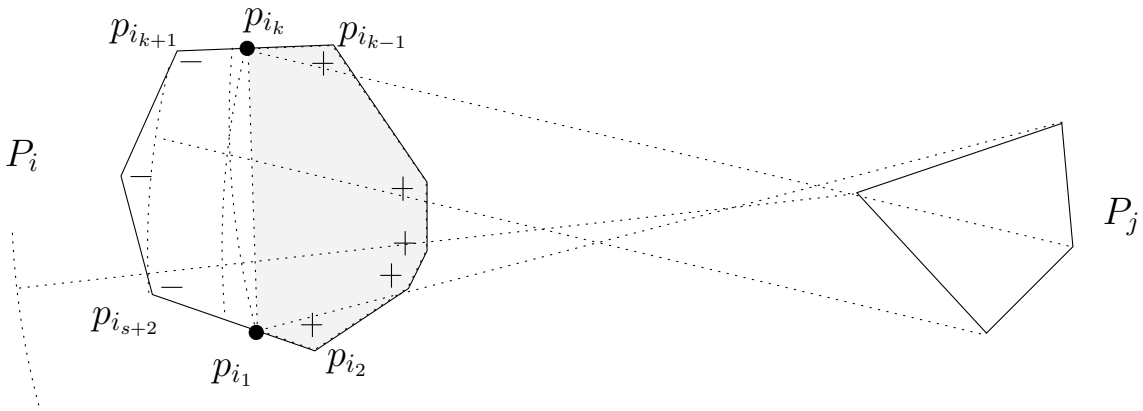


Figure 2.8 – Calcul de la chaîne de points inactifs sur le bord de P_i

le polygone formé par $(p_{i_1}, \ldots, p_{i_k})$ peut alors être retranché de P_i, et on retourne le polygone réduit de cette portion.

Algorithme 8 : Réduction des polygones

Entrées : Deux polygones $P_i = (p_{i_1}, \ldots, p_{i_s}),\ s \geq 3$ et $P_j = (p_{j_1}, \ldots, p_{j_t})$
Sorties : P_i réduit par P_j

pour $k := 1$ ***to*** s **faire**
  si $\forall p_{j_l} \in P_j,\ dist(p_{i_k}, p_{j_l}) > d_{i,j}$ **alors** $\acute{E}tiquette(p_{i_k}) =$ '+'
  sinon $\acute{E}tiquette(p_{i_k}) =$ '–'
si $\forall p_{i_k} \in P_i,\ \acute{E}tiquette(p_{i_k}) =$ '–' **alors**
  retourner *pas de solution*
sinon si $\forall p_{i_k} \in P_i,\ \acute{E}tiquette(p_{i_k}) =$ '+' **alors**
  retourner P_i
sinon
  Trouver une chaîne $p_{i_2}, \ldots, p_{i_{k-1}}$ de sommets de P_i étiquetés '–'
  Appeler p_{i_0} le point étiqueté '+' précédent p_{i_2}
  Appeler $p_{i_{k+1}}$ le point étiqueté '+' suivant $p_{i_{k-1}}$
  Calculer un point p_{i_1} sur le segment $\overline{p_{i_0}p_{i_2}}$ tel que $dist(p_{i_1}, p_{j_l}) > d,\ \forall l = 1, \ldots, t$
  Calculer un point p_{i_k} sur le segment $\overline{p_{i_{k-1}}p_{i_{k+1}}}$ tel que $dist(p_{i_k}, p_{j_l}) > d,\ \forall l = 1, \ldots, t$
  Ajuster $P_i = P_i \backslash (p_{i_1}, \ldots, p_{i_k})$
  retourner P_i

2.4.1.3 Analyse de complexité

Nous donnons ci-dessous les résultats de l'analyse de complexité algorithmique dans le pire cas de la méthode des aires actives.

Indiquons pour cela une estimation du coût de la k^{e} itération de la "boucle de filtrage" dans l'Algorithme 7. L'étiquetage impliqué par un appel à $MinDistancePrune_{poly}$ requiert de calculer les distances entre toutes les paires de sommets $(p_i, p_j) \in P_i \times P_j$. On doit donc compter combien de sommets les polygones P_i et P_j peuvent avoir à l'étape (k, i, j). À chaque appel à

$MinDistancePrune_{poly}$, P_i peut être intersecté avec un demi-espace, il aura donc au maximum $\mathcal{O}(kn)$ côtés après la k^{e} itération de la boucle extérieure. $\mathcal{O}(k^2n^2)$ calculs de distance sont ainsi nécessaires entre P_i et P_j, et cela mène à une complexité totale de $\mathcal{O}(k^2n^4)$ pour calculer la k^{e} itération de l'Algorithme 7. Les évaluations de chaînes (ligne 6) et de points (ligne 7) peuvent tous être effectués par un balayage linéaire des côtés du polygone.

Remarquons que cette analyse est par trop pessimiste parce qu'elle considère une clique de contraintes de distances alors qu'on pourrait raisonnablement prendre en compte le prétraitement effectué. Suite à celui-ci, chaque polygone a un nombre de voisins majoré par une constante m (bien plus petite que n), et cette appréciation réduit la complexité du calcul à $\mathcal{O}(k^2m^3n)$

2.4.1.4 Caractérisation par des régions interdites

La caractérisation par des régions interdites des contraintes de distance euclidienne que nous avons introduite à la section 2.2.1 nous mène à proposer une interprétation géométrique du filtrage de l'algorithme précédent et à y apporter une modification. Après avoir illustré le filtrage sur deux exemples, nous poursuivons avec notre modification et indiquons quelques propriétés vérifiées par l'algorithme.

Exemple 2.2. *Un exemple de région interdite $RI(P, d)$ d'un polygone est donné en gris clair sur la Figure 2.9. Les flèches (de longueur d) donnent le rayon des arcs tracés.*

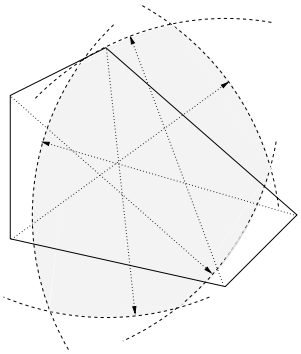


Figure 2.9 – Noyau de rayon d d'un polygone

Exposons dès lors quelques propriétés satisfaites par le noyau de rayon d d'un polygone.

Proposition 2.3. *Caractéristiques du noyau de rayon d de P:*

1. *Le noyau de rayon d de P est un ensemble convexe.*
2. *Le noyau de rayon d de P est réduit à l'ensemble vide si le diamètre de ce polygone a une longueur strictement supérieure à $2d$.*
3. *Le noyau de rayon d d'un polygone peut être construit en considérant l'intersection des boules centrées sur les sommets de P_j, i.e. $N(P, d) = \cap_{i=1}^{n} B(p_i, d)$.*

Preuve :

(1.) En tant qu'intersection d'ensembles convexes, le noyau de rayon d de P est lui-même un ensemble convexe.

(2.) Si le polygone a un diamètre supérieur à $2d$, alors il existe deux arêtes p_{i_1} et p_{i_2} telles que $dist(p_{i_1}, p_{i_2}) > 2d$. On a $B(p_{i_1}, d) \cap B(p_{i_2}, d) = \emptyset$. Comme $N(P, d) \subseteq B(p_{i_1}, d) \cap B(p_{i_2}, d)$ on a bien $N(P, d) = \emptyset$.

(3.) On a clairement $N(P, d) \subseteq \cap_{i=1}^{n} B(p_i, d)$. La réciproque résulte du Lemme 2.1 qui montre que si un point p est à une distance inférieure à d de tous les sommets d'un polygone P, alors il est à une distance inférieure à d de tous les points de P.

■

Rappelons que dans la méthode des aires actives l'algorithme 8 est toujours appelé sur des pavés de diamètre inférieur à d, et $N(P, d)$ est donc toujours non vide ; or ici nous n'en sommes pas sûrs *a priori*. Le premier point de la Proposition 2.3 suggère qu'on pourrait ajouter un test préalable à l'appel de la fonction de réduction de domaines : on s'assurerait de cette façon que le diamètre de P_j est supérieur à $2d$ avant d'invoquer inutilement l'algorithme complet. Il est possible en effet qu'au début de la propagation, le diamètre (la diagonale) d'un pavé soit supérieure à cette valeur, et qu'au fur et à mesure des réductions, le filtrage devienne effectif. Ce test n'est cependant pas nécessaire : si le noyau de rayon d est l'ensemble vide, l'étiquetage initial ne produira que des sommets marqués '+', et l'algorithme retourne immédiatement P_i inchangé.

En reprenant l'exemple précédent, nous illustrons maintenant que la réduction de domaines obtenue par l'algorithme de Nurmela correspond en fait à une approximation convexe de l'intersection de $RI(P_j, d_{i,j})$ avec P_i.

Exemple 2.3. *La région interdite $N(P_j, d_{i,j})$ est représentée en gris clair sur la Figure 2.10 ; l'aire effectivement éliminée sur P_i par l'Algorithme 9 est en gris foncé. Le cas présenté sur ce schéma parvient à capturer presque entièrement l'aire active.*

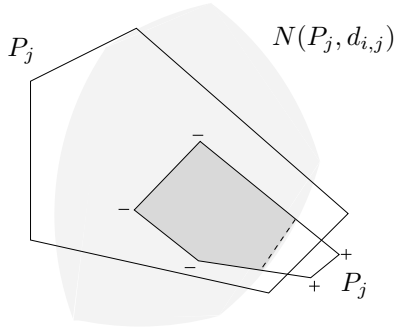


Figure 2.10 – Filtrage vers P_i du noyau de rayon $d_{i,j}$ de P_j.

Il est intéressant d'observer comment cet algorithme peut se montrer efficace sans même passer – comme il est d'usage pour résoudre les problèmes de packing de cercles – par une tessellation initiale des domaines.

2.4.1.5 Algorithme de filtrage $MinDistancePrune_{poly}$

Nous commencons la présentation de notre algorithme de filtrage par un exemple.

Exemple 2.4. *Sur la configuration présentée sur la Figure 2.11, l'algorithme nécessite l'application du filtrage sur deux chaînes de points étiquetés '–'. La configuration est de ce fait moins favorable et l'algorithme originel échoue dans l'élimination d'une partie importante de l'aire inactive du domaine de P_i parce que nous sommes restreints à une représentation connexe des domaines.*

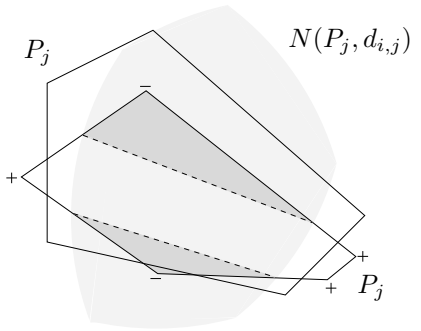


Figure 2.11 – Filtrage vers P_i du noyau de rayon $d_{i,j}$ de P_j.

Étant donné que Nurmela utilisait l'algorithme 8 sur des polygones disjoints (toujours issus d'une tessellation du carré), ce cas de figure a été considéré comme une forme d'anomalie dans l'algorithme. Dans la mesure où nous souhaitons définir un algorithme de filtrage pour des pavés sous des hypothèses de position générale, cette configuration où une approximation convexe de l'intersection de $N(P_j, d_{i,j})$ avec P_i implique plusieurs coupes, doit être vue comme une situation ordinaire.

Nous appelons $MinDistancePrune_{poly}$ l'algorithme que nous proposons pour appréhender le filtrage de pavés quelconques. Les huit premières lignes de cet algorithme sont identiques à celui de Nurmela. Au-delà, l'algorithme initialise un ensemble S_i avec tous les points de P_i étiquetés '+', puis calcule l'ensemble de tous les points d'intersection du bord de $N(P_j, d_{i,j})$ avec P_i pour les ajouter à S_i. L'algorithme termine en retournant l'enveloppe convexe des points de S_i. Nous montrerons dans la suite que l'algorithme d'enveloppe convexe présente l'avantage notable de rendre immédiate l'extension aux intervalles du filtrage.

L'algorithme retourne en sortie le statut du filtrage : en identifiant qu'aucun filtrage n'était possible, on pourra économiser par la suite des appels inutiles à cette procédure.

Algorithme 9 : $MinDistancePrune_{poly}$

Entrées : Deux polygones $P_i = (p_{i_1}, \ldots, p_{i_s}),\ s \geq 3$ et $P_j = (p_{j_1}, \ldots, p_{j_t})$
Sorties : Statut

pour $k := 1$ ***to*** s **faire**
  si $\forall p_{j_l} \in P_j,\ dist(p_{i_k}, p_{j_l}) > d_{i,j}$ **alors** $\acute{E}tiquette(p_{i_k}) =$ '–'
  sinon $\acute{E}tiquette(p_{i_k}) =$ '+'
si $\forall p_{i_k} \in P_i,\ \acute{E}tiquette(p_{i_k}) =$ '–' **alors**
  retourner *pas de solution*
sinon si $\forall p_{i_k} \in P_i,\ \acute{E}tiquette(p_{i_k}) =$ '+' **alors**
  retourner *pas de modification de* P_i
sinon
  $S_i := \{p_{i_k} \in P_i : \acute{E}tiquette(p_{i_k}) =$ '+'$\}$
  pour chaque *chaîne de sommets* $p_{i_2}, \ldots, p_{i_{k-1}}$ *étiquetée* '–' **faire**
    Trouver une chaîne $p_{i_2}, \ldots, p_{i_{k-1}}$ de sommets de P_i étiquetés '–'
    Appeler p_{i_0} le point étiqueté '+' précédent p_{i_2}
    Appeler $p_{i_{k+1}}$ le point étiqueté '+' suivant $p_{i_{k-1}}$
    Calculer un point p_{i_1} sur le segment $\overline{p_{i_0} p_{i_2}}$ tel que $dist(p_{i_1}, p_{j_l}) > d,\ \forall l = 1, \ldots, t$
    Calculer un point p_{i_k} sur le segment $\overline{p_{i_{k-1}} p_{i_{k+1}}}$ tel que $dist(p_{i_k}, p_{j_l}) > d,\ \forall l = 1, \ldots, t$
    $S_i = S_i \cup \{p_{i_1}, p_{i_{k-1}}\}$
  $P_i = EnveloppeConvexe(S_i)$
  retourner *modification de* P_i

Avec pour dessein d'inscrire la contrainte *distn* dans le cadre théorique donné à la section 1.3.3, on peut se demander si l'Algorithme 9 est un opérateur de contraction de contraintes (*cf.* Définition 1.7 p. 20). En effet sur $\mathbb{R}^2$ on peut montrer que $MinDistancePrune_{poly}$ est complet, correct et monotone :

- L'algorithme est complet par construction.
- La correction est prouvée par le Théorème 2.1.
- La monotonie est entraînée par le fait que l'algorithme calcule (par construction) l'enveloppe convexe de $P_i \cap N(P_j, d)^c$. En effet, puisque

$$P_j \subseteq P_j^{'} \Rightarrow N(P_j, d) \supseteq N(P_j^{'}, d),$$

  il s'en suit que

$$MinDistancePrune(P_i, P_j^{'}, d) \subseteq MinDistancePrune(P_i, P_j, d).$$

Nous verrons dans la suite, qu'en adaptant cet algorithme à l'arithmétique d'intervalles, on perd malheureusement la propriété de monotonie.

2.4.1.6 Extension à des domaines comme polytopes en 3 dimensions

Nous donnons ici un premier aperçu de la manière dont l'algorithme peut être étendu en 3 dimensions.

Contrairement à la situation en deux dimensions, il n'y a pas de manière triviale d'approcher (par un arc) le bord du noyau de rayon d avec le polygone P_i. Une fois tous les sommets de P_i étiquetés, on sait que des points d'intersection de P_i (qui est maintenant un polytope) avec le bord de $N(P_j, d_{i,j})$ peuvent être trouvés sur les arêtes partant d'un sommet étiqueté '–' vers un sommet marqué '+'. Cependant, en trois dimensions, le polytope qui s'en suit n'est pas défini de manière unique, ni nécessairement convexe.

Deux façons alternatives de procéder s'offrent naturellement :

1. La Figure 2.12 illustre sur un exemple le plus petit polytope que l'on puisse construire à partir des points marqués '+' et des points d'intersection du bord de $N(P_j, d_{i,j})$ avec les arêtes de P_i. La correction de cette approximation est une conséquence de la convexité de $N(P_j, d_{i,j})$. Le polytope résultant n'est cependant pas convexe.
2. Calculer l'enveloppe convexe des p_{i_k} avec l'ensemble des points étiquetés '+'. Cette alternative est illustrée sur la Figure 2.13.

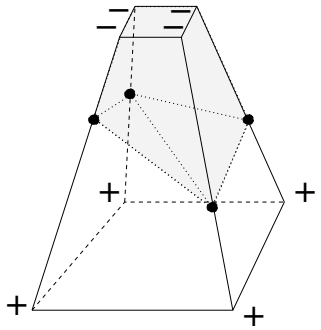


Figure 2.12 – Intersection du noyau de rayon d avec le polytope.

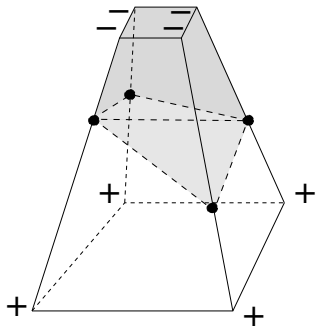


Figure 2.13 – Approximation de l'intersection de rayon d avec le polytope par une enveloppe convexe.

2.4.1.7 Extension aux intervalles de l'algorithme

Nous donnons maintenant l'ensemble des modifications qu'il faut apporter à l'algorithme précédent pour son implémentation en machine.

L'algorithme de filtrage que nous avons présenté à la dernière section nécessiterait le recours à une arithmétique rationnelle pour être correct. Or la recherche de bonnes performances nécessite

l'utilisation de nombres flottants dans son implémentation. Une grande vigilance doit donc être prise pour contourner les problèmes d'incorrection des calculs avec l'arithmétique flottante.

Cette section présente comment la procédure de filtrage peut être ajustée à l'arithmétique d'intervalles pour garantir des calculs corrects. Nous apportons ce faisant plusieurs améliorations à l'adaptation initiée par Markót [84].

Quoique Nurmela et Östergård [96] donnent une étude des cas pathologiques où les éventuelles erreurs de calcul doivent être corrigées, cette approche n'offre cependant pas de garanties sur la correction du filtrage. La source la plus importante d'incorrections dans les calculs provient des lignes 10 et 11 de l'algorithme 9. On doit porter une attention particulière au fait qu'il est possible que $p_1^{'}$ et $p_k^{'}$ ne puissent pas être représentables par des nombres flottants. Markót [84] apporte quant à lui des modifications pour effectuer tous les calculs en utilisant l'arithmétique d'intervalles, ce qui permet de prouver la correction de toutes les étapes de l'algorithme.

En utilisant l'arithmétique d'intervalles pour effectuer ces calculs, on ne pourra obtenir, au lieu de points, que des petites boîtes $P_1^{'}$ et $P_k^{'}$ englobant p_{i_1} et p_{i_k}. Une conséquence est que l'aire restante devra être légèrement élargie pour prendre en compte cet aspect. Nous abordons à présent deux manières de procéder pour résoudre ce problème.

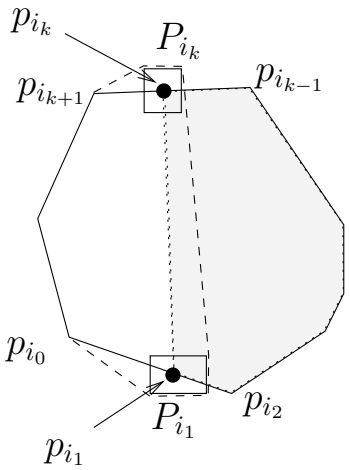


Figure 2.14 – Calcul par intervalles de la chaîne de points inactifs du bord

Markót y répond en conservant l'aire représentée sur la Figure 2.14 : l'enveloppe convexe de $p_{i_{k+1}}$, p_{i_0}, $P_k^{'}$ et $P_1^{'}$ y est « rattachée » à la chaîne $p_{i_{k+1}}, \ldots, p_{i_0}$. Il en découle que le polygone obtenu peut perdre sa propriété de convexité. Précisément, les angles obtenus en p_{i_0} et en $p_{i_{k+1}}$ peuvent devenir obtus. Markót propose d'accepter cette donnée et introduit pour cela une condition supplémentaire pour assurer la stabilité de l'algorithme. Nous nous aiderons pour l'expliquer de la Figure 2.15 : cette condition exige que l'intersection de l'enveloppe convexe de $P_1^{'}$ et $P_k^{'}$ ait une intersection vide avec la droite $(p_{i_{k+1}}, p_{i_0})$[9]. Ceci permet d'assurer que les polygones restent simples[10] et en pratique cette condition est le plus souvent vérifiée.

Nos expérimentations nous ont cependant montré que la violation de cette condition pouvait parfois être très gênante du fait que dans ce cas elle peut diminuer considérablement le filtrage.

[9] Elle est appelée *Separate* dans [84].

[10] Un polygone est simple si aucune de ses arêtes ne se touchent.

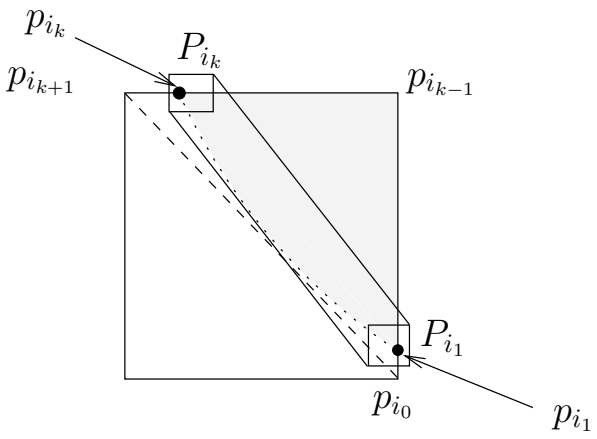


Figure 2.15 –

Sur l'exemple de la Figure 2.15, on voit aisément que si l'algorithme ne peut pas filtrer presque la moitié de P_i, le filtrage global sur l'ensemble des points est fortement amoindri.

Nous proposons pour cette raison de remplacer P_i par l'enveloppe convexe de $(p_{i_{k+1}}, \ldots, p_{i_0})$ avec $P_1^{'}$ et $P_k^{'}$. La condition de stabilité et la correction de l'algorithme sont préservées par ce calcul.

Cette nouvelle écriture présente les avantages suivants :

- Son extension aux intervalles est immédiate : Il suffit, pour chaque chaîne de points étiquetés '–', de remplacer aux lignes 13 et 14 les points d'intersection de $N(P_j, d_{i,j})$ avec les arêtes de P_i par les quatre points définissant une boîte englobante de cette intersection.
- La simplicité de cette implémentation permet d'envisager son implémentation en 3 dimensions.

L'algorithme global de *Branch & Prune* est généralement plus rapide (parfois de beaucoup – nous gagnons un facteur 10 sur un exemple) dans les expérimentations que nous avons menées, et cette écriture infirme d'une certaine façon l'intérêt qu'il y aurait à utiliser des polygones non-convexes pour obtenir une propagation plus forte.

2.4.1.8 Algorithme de propagation

Présentons enfin les quelques améliorations que comporte l'algorithme de propagation de *distn* (Algorithme 10) par rapport au schéma originel de propagation de la méthode des aires actives.

Dans l'algorithme 9, nous avons enrichi la valeur de retour de l'algorithme originel pour indiquer en plus si le filtrage a modifié les polygones. On peut enregistrer cette information dans un tableau de taille $n \times n$ qui donne la date de la dernière modification de P_i. Avant d'appeler l'algorithme de filtrage, il suffit de vérifier si l'on satisfait la fonction *Réviser* définie par

$$\mathit{Réviser}(t_i, t_j) \rightarrow (t_i > (k-1) * n^2 + i * n + j) \ \vee \ (t_j > (k-1) * n^2 + i * n + j)$$

où t_i est la date de dernière modification de P_i

Algorithme 10 : $MinDistancePropagation_{poly}$

Entrées : Une combinaison de n tuiles
Sorties : Cette combinaison réduite par le filtrage

```
1  pour i := 1 to n faire P_i = Polygone(X_i × Y_i)
2  pour i := 1 to n faire déterminer Voisins(P_i)
3  pour k := 1 to MaxIterations faire
4      pour i := 1 to n faire
5          pour j ∈ Voisins(P_i) faire
6              si Réviser(Date(P_i), Date(P_j)) alors
7                  statut := MinDistancePrune_poly(P_i, P_j, d_{i,j})
8                  si statut = "pas de solution" alors
                       retourner statut
9                  sinon si statut = "modification de P_i" alors
                       Date(P_i) = k * n^2 + i * n + j
10 Ajuster X_i × Y_i = BoîteEnglobante(P_i),   ∀i
```

La boucle principale itérant la réduction des boîtes dans l'algorithme de filtrage de la Méthode des Aires Actives (ligne 3, Algorithme 7 page 54) utilise un critère d'arrêt statique défini empiriquement. Nous avons rendu ce critère dynamique en lui substituant un critère de taux de filtrage. Il permet de laisser la boucle propager les réductions de domaines tant que ces opérations restent efficaces. Nous avons éprouvé plusieurs fonctions $TauxDeFiltrage$:

$$SumArea = \sum_{i=1...n} PolyArea(i)_{it=k} / \sum_{i=1...n} PolyArea(i)_{it=k-1}$$

$$MaxArea = \max_{i=1...n} (PolyArea(i)_{it=k} / PolyArea(i)_{it=k-1})$$

et les mêmes fonctions, appliquées aux boîtes englobantes des polygones. En utilisant la contrainte dans un algorithme de *Branch & Bound* standard, l'heuristique $MaxArea$ donnait les meilleurs temps, et produisait effectivement une grande variété de nombre d'itérations de la boucle. Il s'est néanmoins avéré utile d'ajouter également une borne supérieure sur le nombre d'itérations. Le gain obtenu par rapport à un critère d'arrêt statique est généralement de l'ordre de deux. Dans l'implantation actuelle, les constantes ont été fixées à $MaxIterations = 15$ et $\varepsilon = 0.9$.

Il pourrait également être avantageux d'introduire une étape supplémentaire de prétraitement à la contrainte, pour détecter plus rapidement des cas d'échec. L'algorithme par balayage de non-intersection de rectangles de Beldiceanu [11] nous semble être idoine pour ce test.

2.5 Généralisation à *distn_var*

La représentation des domaines par des polygones permet également pour la contrainte *distn_var*, de propager plus fortement des variables de distance minimale et maximale. Nous donnons ici les principes de l'algorithme de filtrage, puis nous détaillons notre approche pour calculer les sommes de Minkowski intervenant dans la procédure.

En imposant la contrainte *distn_var*([P$_1$, ..., P$_n$] , dvar) on peut déjà appliquer aux variables de position le filtrage de la contrainte suivante :

$$distn([\mathtt{P_1}, \ldots, \mathtt{P_n}]\ ,\ \mathtt{d}\ ,\ \mathtt{D}), \tag{2.9}$$

$$\text{où} \quad \forall (i,j) \quad \begin{cases} D_{i,j} = \overline{dvar_{i,j}} \\ d_{i,j} = \underline{dvar_{i,j}} \end{cases} \tag{2.10}$$

Le filtrage peut alors en retour être étendu aux variables de distance : une fois la propagation précédente achevée, il est en effet possible de mettre à jour les bornes déterminées par les domaines polygonaux. Nous l'illustrons sur la Figure 2.16, où il est remarquable que la distance entre tout couple de points solutions est majorée (resp. minorée) par la distance maximale (resp. minimale) entre tout couple de points pris dans les polygones. Une double flèche en pointillés (resp en trait plein) indique la valeur distance maximale (resp. minimale) entre P_i et P_j.

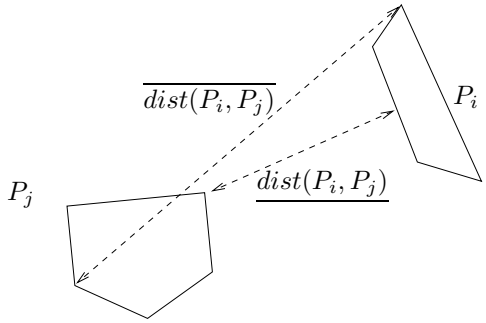


Figure 2.16 – Distance minimale et maximale entre deux polygones

L'algorithme naïf pour calculer les bornes sur les variables de distances est quadratique. Nous proposons ici un algorithme en $\mathcal{O}(n\ log(n))$.

Les bornes obtenues par le filtrages de *MinDistancePrune* peuvent en effet être obtenu en calculant la différence de Minkowski de tous les couples de polygones. Pour deux polygones convexes P_i et P_j, la différence de Minkowski de ces polygones que nous notons $P_i \ominus P_j$ est aussi un polygone convexe. Nous donnons un exemple de cette construction sur la Figure 2.17. Or les points de $P_i \ominus P_j$ sont l'ensemble des couples de la forme :

$$\left\{ (p_i^x - p_j^x, p_i^y - p_j^y) : p_i \in P_i, p_j \in P_j \right\},$$

si bien qu'en considérant l'ensemble des normes de cet ensemble de vecteurs, on obtient l'ensemble

$$\left\{ \sqrt{(p_i^x - p_j^x)^2 + (p_i^y - p_j^y)^2} : p_i \in P_i, p_j \in P_j \right\},$$

i.e. l'ensemble des valeurs possibles de la distance entre deux points dans P_i et P_j.

Pour calculer le minimum de cet ensemble, si l'origine n'est pas comprise dans le polygone $P_i \ominus P_j$, il suffit de calculer le minimum de la distance qu'il y a d'une arête de ce polygone à l'origine. Sinon, il existe deux points identiques $p_i \in P_i$ et $p_j \in P_j$ (les polygones s'intersectent) et le minimum est zéro. Précisons de plus que la somme de Minkowski construite est une approximation extérieure ; la distance minimale (resp. maximale) de l'origine à ce polygone est

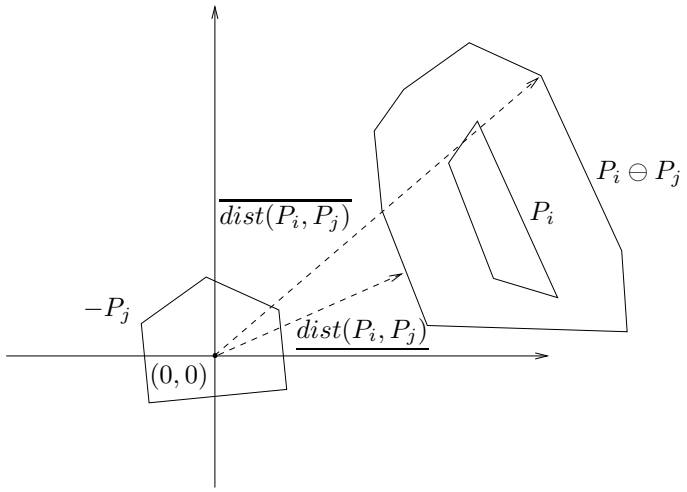


Figure 2.17 – Utilisation de la différence de Minkowski pour le calcul de la distance minimale entre deux polygones P_i et P_j.

pour cette raison aussi une borne inférieure (resp. supérieure) à la distance entre les polygones, si bien que la transformation préserve la correction des calculs.

On remarquera pour finir que les calculs nécessaires pour estimer $\overline{dvar_{i,j}}$ ont en fait déjà été effectués dans l'Algorithme 9. Lors de l'étape initiale d'étiquetage des polygones, il suffit d'ajouter au test d'écart une comparaison qui extrait la distance maximale entre les polygones.

Terminons en indiquant notre approche pour le calcul de la somme de Minkowski. Nous nous basons sur l'algorithme en $\mathcal{O}(m+n)$ pour des polygones à m et n côtés donné dans [38] que nous détaillons sur l'Algorithme 11. Il est fondé sur l'observation que tout point extrême dans une direction $\overrightarrow{d}$ appartenant à la somme de Minkowski de deux objets convexes dans le plan, est la somme de deux points extrêmes dans chacun des objets selon cette même direction. L'algorithme construit la somme en parcourant simultanément la liste des côtés (orientés) des deux polygones. La comparaison de l'angle formé par chacun des vecteurs avec l'axe positif permet de déterminer quels points il faut choisir dans la somme pour n'obtenir que des points extrêmes.

Algorithme 11 : Somme de Minkowski

Entrées : Deux polygones convexes $P = \{p_1, \ldots, p_n\}$ et $Q = \{q_1, \ldots, q_m\}$ donnés dans l'ordre trigonométrique, avec p_1 et q_1 les sommets de plus petite ordonnée et de plus petite abscisse.
Sorties : $P \oplus Q$
$i := 1,\ j := 1$
$p_{n+1} := p_1,\ q_{m+1} = q_1$
répéter
  ajouter $p_i + q_j$ à $P \oplus Q$
  si $angle(p_i, p_{i+1}) < angle(q_j, q_{j+1})$ **alors** $i := i + 1$
  sinon si $angle(p_i, p_{i+1}) > angle(q_j, q_{j+1})$ **alors** $j := j + 1$
  sinon $i := i + 1, j := j + 1$
jusqu'à $i = n + 1$ *et* $j = m + 1$

Pour rendre cet algorithme efficace nous combininons le calcul sur les rationnels avec l'arithmétique d'intervalles. Nous devons considérer deux sources éventuelles d'erreur avec des calculs flottants : le calcul du prédicat géométrique comparant l'angle à l'axe positif entre deux vecteurs et la construction des sommes de points.

Le calcul du prédicat ne fait intervenir que des comparaisons de signes et des différences et produits de vecteurs. Ces opérations sont aisément remplacées par des calculs sur des intervalles et lorsque l'arithmétique d'intervalles occasionne une surestimation, qui ne permet plus de décider du résultat du test, les mêmes calculs sont simplement repris en utilisant une arithmétique rationnelle.

Pour les opérations de sommation de points, afin de garantir la correction de l'algorithme de filtrage, on doit s'assurer que le polygone construit contient le polygone obtenu avec des calculs exacts. Nous effectuons pour cela les sommes avec l'arithmétique d'intervalles – nous pourrons de la sorte obtenir jusqu'à quatre points encadrant le point de sommation exacte – et terminons en retournant l'enveloppe convexe de tous les points calculés en lieu d'approximation extérieure du polygone égal à la somme exacte de Minkowski. Le calcul d'enveloppe convexe se fait au pire en $\mathcal{O}(n\ log(n))$.

2.6 Expérimentations

Cette section présente les expérimentations que nous avons menées pour donner une première évaluation de la performance de *distn* sur deux exemples assez simples résolus dans un cadre continu. Nous verrons dans le chapitre suivant une utilisation de *distn_var* dans une application plus complexe que les exemples abordés ici.

Nous avons incorporé la contrainte *distn* dans le solveur de contraintes continues Realpaver [52] qui implémente un mécanisme de recherche par *Branch & Prune*. Les résultats que nous présentons avec *distn* sont confrontés à ceux obtenus avec les 2-consistances disponibles dans Realpaver et pour certains tests aussi à ceux obtenus avec l'algorithme *Quad* [78]. Nous utilisons *distn* en tant que modèle redondant où la contrainte est propagée après un premier filtrage par les contraintes élémentaires. Les benchmarks ont été effectués sur un processeur de type Pentium M-735 à 1.6Ghz avec 512Mb de mémoire vive et la même quantité de mémoire virtuelle.

Dans nos expérimentations nous traitons d'abord un problème de robotique qui met en avant les limitations de l'utilisation de contraintes globales continues sur un problème de petite taille. Nous abordons ensuite un problème de configuration où le filtrage fort de *distn* permet de résoudre des problèmes de taille supérieure à ceux auxquels on doit se limiter avec des contraintes élémentaires.

2.6.1 Cinématique avant d'un robot planaire

Les techniques de PPC continue sont très utiles pour résoudre certains designs planaires de robots parallèles qui n'ont pas de solution algébrique connue. Nous avons représenté sur la Figure 2.18 un manipulateur 3-RPR [28], qui consiste en une plateforme planaire, dont la position des points D, E et F doit être en adéquation avec la longueur des vérins CF, AD, BE et EF. Sont donnés : $\mathtt{A} = (0, 0)$, $\mathtt{B} = (15.91, 0)$, $\mathtt{C} = (0, 10), \mathtt{CF} = 12$, $\mathtt{AD} = \mathtt{DE} = 14.98$, $\mathtt{DF} = 20.84$ et $\mathtt{BE} = 15.38$.

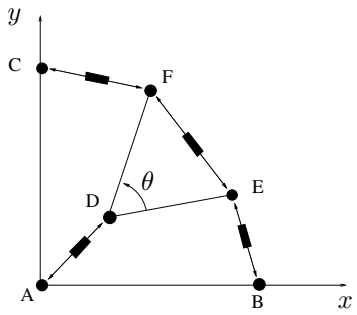


Figure 2.18 – Le manipulateur 3-RPR.

Le problème peut être modélisé en contraintes par le système suivant :

/* la matrice M spécifie les relations de distance entre tous les points */
$distn(((\texttt{xD}, \texttt{yD}), (\texttt{xE}, \texttt{yE}), (\texttt{xF}, \texttt{yF}), (\texttt{xA}, \texttt{yA}), (\texttt{xB}, \texttt{yB}), (\texttt{xC}, \texttt{yC})), \texttt{M})$,
/* utilisation d'une variable FE pour la longueur de F à E */
$(\texttt{xF} - \texttt{xE})^2 + (\texttt{yF} - \texttt{yE})^2 = \texttt{FE}^2$,
/* l'angle theta est aigu */
$\texttt{theta} = 0.882603$,
$\cos(\texttt{theta}) = (\texttt{DE}^2 + \texttt{DF}^2 - \texttt{FE}^2)/(2 * \texttt{DE} * \texttt{DF})$,
/* le produit vectoriel de DE et DF est positif */
$(\texttt{xE} - \texttt{xD}) * (\texttt{yF} - \texttt{yD}) - (\texttt{xF} - \texttt{xD}) * (\texttt{yE} - \texttt{yD}) >= 0$;

Ce problème n'est pas vraiment difficile à résoudre, et l'algorithme HC4 réussit à isoler l'ensemble des solutions rapidement. *distn* est légèrement plus lent même si la contrainte permet d'économiser quelques *splittings*. *Quad* ralentit notablement la résolution, du fait du coût élevé des appels au simplexe. Les temps de calcul pour trouver toutes les solutions du CSP sont donnés sur le tableau 2.1.

	Temps	Splittings	Boîtes
hc4	70ms	146	7
distn	110ms	135	9
Quad[11]	16s	?	?

Tableau 2.1 – Résultats de la résolution du manipulateur 3-RPR

2.6.2 La fractale de Sierpinski

La fractale de Sierpinski a été conçue par le mathématicien polonais Waclaw Sierpinski (1882-1969). Les premières itérations de son motif sont représentées sur la figure 2.19. Nous utilisons cette fractale pour exprimer une classe de CSPs représentant des corps rigides [32]. L'objet du CSP est de trouver les coordonnées cartésiennes des points en fonction du système de contraintes

[11] Les résultats de *Quad* ont été obtenus sur un Pentium IV à 2Ghz, 1GB Ram.

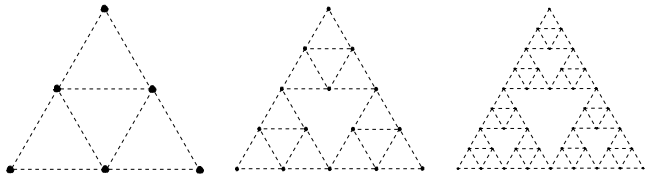


Figure 2.19 – Fractales de Sierpinski d'ordre 1, 2 et 3.

de distance (les lignes en pointillés sur la figure). Ce problème comporte un nombre fini de solutions dès lors que au moins deux coordonnées selon x et une coordonnée selon y sont fixées. Nous étudions successivement la résolution de trois instances ayant une solution unique et d'une autre instance comprenant plusieurs solutions.

Les trois premières instances que nous traitons sont des fractales d'ordre un, deux et trois telles que présentées sur la Figure 2.19. Sur chacune on fixe dans le modèle le point inférieur gauche, on contraint le point inférieur droit selon un axe vertical passant par ce point et on contraint le point supérieur selon un axe horizontal passant par ce point. Ces contraintes restreignent le problème à une solution unique tout en laissant libres les coordonnées des autres points. Les résultats numériques de la résolution de ces CSPs sont donnés sur les tableaux 2.2 et 2.3. en utilisant respectivement les consistances locales HC3 et HC4.

	Temps		Splittings		Boîtes	
	distn	naïve	distn	naïve	distn	naïve
Sierpinski(1)	0ms	0ms	1	16	1	1
Sierpinski(2)	690ms	780ms	336	1892	11	192
Sierpinski(3)	55,790ms	-	7507	-	6	-

Tableau 2.2 – Résultats sur une région comprenant une solution unique en utilisant HC4

	Temps		Splittings		Boîtes	
	distn	naïve	distn	naïve	distn	naïve
Sierpinski(1)	10ms	10ms	7	20	1	1
Sierpinski(2)	130ms	2.220ms	39	1836	1	196
Sierpinski(3)	18,450ms	-	1763	-	15	-

Tableau 2.3 – Résultats sur une région comprenant une solution unique en utilisant HC3

Le comportement par défaut de Realpaver choisit sur ces exemples l'algorithme HC4 et une bissection des boîtes en 3 parties égales avec une précision de $1.0e^{-8}$ et l'ordre *round-robin* de sélection des variables. La consistance forte de *distn* réussit à améliorer tous les calculs, parfois de plusieurs ordres de grandeur. En particulier, les consistances locales ordinaires sont trop faibles pour résoudre le problème Sierpinski(3) en une heure de calcul. La formulation naïve donne des meilleures performances avec HC4 par rapport à HC3, mais cet ordre est curieusement inversé en utilisant *distn*. Cette différence dans les résultats nous semble fortuite et nous l'attribuons à la non-confluence des calculs. Les temps obtenus avec la box-consistance ne sont pas indiqués, mais ils sont tous bien inférieurs à ceux présentés.

Considérons maintenant une fractale de Sierpinski d'ordre 2 dans laquelle on fixe les points inférieurs droits et inférieurs gauches et laisse les autres points libres sur le demi-plan supérieur défini par cette ligne. Ce système comprend au total 6 solutions. Les tableaux 2.4 et 2.5 indiquent les performances de résolution que nous obtenons avec un modèle naïf .

Temps		Splittings		Boîtes	
distn	naïve	distn	naïve	distn	naïve
140ms	510ms	28	1238	6	210

Tableau 2.4 – Résultats avec HC4 sur une fractale d'ordre 2 avec six solutions

Temps[12]	Splittings	Boîtes	Appels au simplexe
96s	15	11	4584

Tableau 2.5 – Résultats avec Quad sur une fractale d'ordre 2 avec six solutions

La formulation du problème à l'aide de *distn* apporte ici encore les meilleurs résultats. Si les temps entre le modèle naïf et celui utilisant la contrainte globale restent relativement proches, la globale permet d'isoler exactement les solutions du problème alors qu'il faudrait vérifier l'existence et l'unicité des solutions parmi le grand nombre de boîtes générées par le modèle naïf. Bien qu'il faille attribuer une partie des pertes de performances par rapports à nos résultats avec RealPaver à notre machine plus récente et à l'utilisation d'Ilog-Solver dans l'implémentation de *Quad*, le coût des appels au simplexes semble ici tout de même très pénalisant.

2.7 Conclusion

Nous avons présenté *distn*, une nouvelle contrainte globale sur les domaines continus, qui facilite la modélisation de plusieurs problèmes où interviennent des contraintes géométriques de distance euclidienne. Sa procédure de filtrage est adaptée d'un algorithme de packing de cercles et a été intégrée dans un solveur de contraintes continues.

Une caractérisation géométrique des contraintes de distance euclidienne par des régions permises et interdites nous a conduit à identifier comment améliorer le filtrage opéré par les formes classiques de consistance locale. Nous avons à cette fin modifié et étendu deux algorithmes de packing de cercles pour les adapter au cadre d'une utilisation dans un solveur de contraintes. Le premier algorithme effectue des linéarisations successives des régions à approximer. Le second surpasse cette approche en adoptant une représentation polygonale des domaines. Le recours à des techniques d'arithmétique d'intervalles garantit la correction de ces algorithmes.

Une première série de benchmarks a confirmé l'intérêt de cette approche. Nous verrons au chapitre suivant comment la contrainte *distn_var* présente un avantage décisif pour résoudre une application industrielle de déploiement de capteurs avec l'allocation de fréquences.

[12]Les résultats sont obtenus sur un portable Pentium III à 1Ghz,256MB

CHAPITRE 3

Application de *distn* à un problème hybride de déploiement progressif d'antennes

Nous présentons dans ce chapitre un outil d'aide au déploiement d'antennes sollicité par Thales. La description de l'application opérationnelle sera suivie d'un modèle mathématique du problème, que nous transcrivons parallèlement en deux modèles en contraintes. Le premier [60], entièrement discret, est mis en œuvre dans le cadre du solveur de contraintes utilisé originellement par Thales. Le second [61], entreprend une hybridation avec des contraintes d'intervalles. L'hybridation naïve n'améliore pas les résultats. Pour cette raison, nous développons des heuristiques qui rendent la résolution plus performante mais ne suffisent pas pour résoudre des problèmes de taille conséquente. Finalement, nous parvenons à résoudre les problèmes qui nous intéressent dans le cadre d'un espace de recherche hybride par l'introduction de la contrainte *distn_var* proposée au chapitre 2.

3.1 Description du problème applicatif

Le nouveau théâtre des opérations voit cohabiter, sur des zones géographiques parfois très limitées, une très grande quantité de systèmes de télécommunications aussi bien civils que militaires. De nouvelles configurations sont demandées au fur et à mesure du déploiement des forces, tandis que d'autres disparaissent, ce qui induit un problème de déploiement progressif d'antennes mobiles. Ces demandes en communication, soumises au problème classique d'allocation de fréquences radio, rendent de plus en plus difficile la tâche de déploiement, permettant soit de construire un réseau de toutes pièces, soit d'enrichir un réseau déjà existant. En fonction des objectifs de couverture, un outil d'aide au déploiement doit fournir à l'opérateur la position théorique des sites radio tout en garantissant les qualités de services requises (respect des contraintes d'interférences, etc.) et en s'adaptant aux contraintes géographiques du terrain (présence de zones interdites, etc.).

Nous introduisons d'abord quelques définitions utiles pour décrire les systèmes physiques que nous modélisons, puis nous décrivons de façon informelle les deux principales classes de contraintes physiques en jeu.

3.1.1 Définitions du problème

Le problème de déploiement d'antennes comporte deux sous-problèmes combinatoires distincts, à savoir le problème d'allocation de fréquences d'une part et celui du déploiement des sites d'autre part, qui sont reliés par une certaine classe de contraintes pour former le problème complet. Dans la suite, le sous-problème d'allocation de fréquences est noté RLFAP, pour *Radio-Link Frequency Assignment Problem* ; celui de déploiement est noté Loc en référence aux problèmes apparentés de *Facility Location* en Recherche Opérationnelle. Le problème complet d'allocation de fréquences avec déploiement sera alors noté LocRLFAP, pour Location and Radio-Link Frequency Assignment Problem. Notons que par anticipation de la présentation du LocRLFAP et en raison de la nature distincte de ce problème, notre présentation du RLFAP diffère légèrement de celles trouvées ordinairement dans la littérature.

3.1.1.1 Définition du sous-problème d'allocation de fréquences (RLFAP)

Décrivons d'abord la nature des contraintes radio-électriques entre antennes. Le problème induit par ces contraintes est celui de l'allocation des fréquences dans un réseau [1, 70, 27].

Définition 3.1. On définit un site de déploiement comme un lieu géographique assimilé à un point, sur lequel est implanté un nombre prédéfini d'antennes raccordées à des émetteurs ou à des récepteurs.

Un réseau est composé d'un ensemble de sites. Une liaison établit un lien entre deux sites géographiques distincts; elle peut être constituée de un ou de plusieurs trajets. Nous considérerons cependant sans perte de généralité qu'une liaison est composée d'un maximum de deux trajets - au besoin certains sites peuvent sinon être dédoublés. Un exemple de déploiement pour un réseau comportant 10 sites est présenté sur la Figure 3.1.

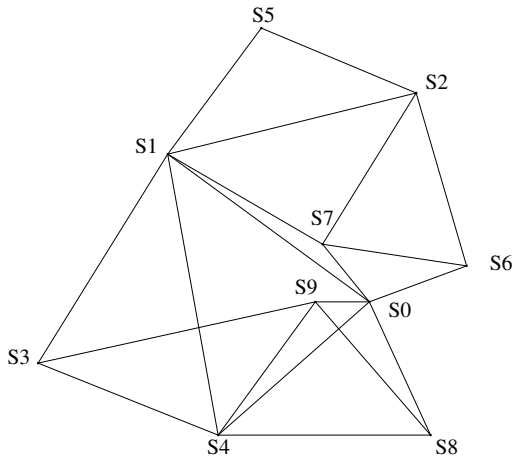


Figure 3.1 – Topologie d'un réseau à 10 sites.

Définition 3.2. On appelle trajet un bond radioélectrique unidirectionnel établi entre deux antennes sur des sites géographiques distincts à une fréquence et une polarisation données. On appellera T l'ensemble des trajets d'un problème donné.

Définition 3.3. Une liaison établit un lien bidirectionnel entre deux sites géographiques. Elle peut être constituée de un ou de plusieurs trajets dans chaque sens.

Pour chaque trajet t_{ij} d'un émetteur en S_i à un récepteur en S_j (les sites seront obligatoirement distincts), on définit d'une part un domaine fréquentiel noté F_{ij} qui contient l'ensemble des fréquences susceptibles d'être attribuées au trajet t_{ij} et d'autre part un domaine de polarisation P_{ij} qui décrit les polarisation admissibles pour l'onde du trajet (pour simplifier on ne considère que deux valeurs possibles de polarisation : verticale ou horizontale). Ce couple définit le domaine de ressources du trajet :

Définition 3.4. On définit le domaine de ressources d'un trajet comme l'ensemble des couples (fréquence, polarisation) qui sont susceptibles de lui être allouées.

On dira que la polarisation d'un trajet est libre s'il peut indifféremment prendre une polarisation verticale ou horizontale.

3.1.1.2 Définition du sous-problème de déploiement (Loc)

Étant donné une zone géographique, on veut y déployer un ensemble de n sites $(S_i)_{1\leq i\leq n}$ en assurant certains liens de communication entre sites au moyen d'antennes (composées d'un émetteur et/ou d'un récepteur) ayant des portées (couverture radio) distinctes tout en satisfaisant certaines contraintes spatiales de déploiement.

Nous donnons maintenant quelques notions relatives au déploiement.

Définition 3.5. On appelle Zone de Déploiement (ZD) toute portion de terrain dans laquelle l'utilisateur veut établir ses sites.

Définition 3.6. On appelle Zone Potentielle d'Implantation (ZPI) toute portion de terrain dans laquelle l'utilisateur veut déployer un site.

On notera le cas particulier suivant :

Définition 3.7. On appelle Point d'Implantation (PI) toute ZPI réduite à un point, défini manuellement ou par le calcul, sur lequel l'utilisateur veut déployer un site.

L'opérateur peut par exemple choisir d'utiliser pour PI des points hauts comme le sommet d'une colline.

Définition 3.8. On appelle Zone Interdite (ZI) toute portion de terrain dans laquelle on ne peut établir un site du fait des conditions de terrain.

À titre d'exemple, des étendues d'eau ou des pentes trop abruptes seront définies comme ZI.

Définition 3.9. On appelle Solution de déploiement le fait d'associer à chaque site à déployer une position sur le terrain qui satisfait l'ensemble des contraintes opérationnelles.

Une solution de déploiement doit en particulier respecter les contraintes de zones d'implantation. La ZD pourra donc être vue comme l'union des ZPI privée de l'union des ZI.

3.1.1.3 Définition du problème complet (LocRLFAP)

Allouer des fréquences consiste pour l'allocateur à trouver un couple (fréquence, polarisation) pour chaque trajet, parmi un domaine de ressources fréquentielles qui leur est associé (déterminé par les contraintes matérielles, de réglementation ou par des choix de l'opérateur) et satisfaisant l'ensemble des contraintes de compatibilité radioélectrique, tout en précisant en même temps la position des sites associés, en respectant les données du terrain. Le nombre d'antennes données dans le problème est fixé et constant a priori.

On définit une solution réalisable du problème d'allocation de fréquences comme l'affectation d'une ressource (fréquence, polarisation) à chaque trajet en satisfaisant l'ensemble des contraintes posées.

3.1.2 Description des contraintes radio-électriques (RLFAP)

Nous décrivons dans cette section les contraintes de compatibilité radioélectrique qui sont indépendantes du déploiement choisi : les contraintes co-site, de liaison, de bande et celles imposées par le matériel ou l'opérateur.

3.1.2.1 Contraintes co-site

La première catégorie d'interférences co-site correspond aux contraintes dues au couplage Émetteur-Récepteur au sein d'un même terminal (un terminal est composé d'un émetteur et d'un récepteur pour garantir la bidirectionnalité de la liaison) ou à la proximité d'un récepteur avec un autre émetteur implanté sur un même site. On parle alors de contraintes co-site d'interférence.

Lorsque deux ou plusieurs émetteurs sont présents sur un même site, leurs fréquences se combinent linéairement de façon couplée pour perturber les récepteurs présents en ce lieu. On parle alors de contraintes co-site d'inter-modulation.

3.1.2.2 Contraintes de liaison

Dans certains problèmes, le choix des terminaux peut imposer d'autres contraintes. Par exemple, l'écart entre les deux trajets d'une même liaison (écart duplexe) peut être fixé. De même, des contraintes concernant les harmoniques peuvent se traduire par des contraintes restreignant les valeurs que cet écart peut prendre.

3.1.2.3 Contraintes de bande

Le spectre disponible s'étend de 30 à 60 MHz, par pas de 25 kHz, et peut être entrecoupé de bandes interdites réservées à d'autres applications (télédiffusion, gendarmerie, etc.). Ces fréquences interdites devront être impérativement respectées par l'opérateur.

3.1.2.4 Autres contraintes de matériel et d'opérateur

Des types de contraintes, autres que radioélectriques, peuvent être prise en compte dans le problème. Certains matériels peuvent ajouter des contraintes d'utilisation supplémentaires :

- contraintes d'égalité de polarisation entre les deux trajets constituant une liaison,
- fonctionnement limité à un sous-ensemble du spectre disponible.

L'opérateur peut aussi désirer imposer des contraintes supplémentaires sur certains trajets. Ces contraintes peuvent prendre la forme suivante :

- contraintes de fréquence imposées,
- contraintes d'égalité ou d'inégalité de fréquence,
- contraintes de polarisation imposées,
- contraintes d'égalité ou d'inégalité de polarisation.

3.1.3 Description des contraintes de déploiement (Loc)

Nous donnons ici la description des contraintes de déploiement portant exclusivement sur la position géographique des sites.

La différence majeure qui caractérise le LocRLFAP d'un RLFAP classique, est que les antennes auxquelles nous devons attribuer des fréquences ne sont pas encore placées et que le graphe du réseau de contraintes d'interférence distante n'est pas défini *a priori*. En particulier, on s'autorisera à déplacer les antennes non-fixées *a priori* dans le but de réduire le nombre de fréquences allouées.

Notre description porte sur les contraintes d'interdistance minimale, de portée et de zones.

3.1.3.1 Contraintes d'interdistance minimale

Selon les matériels envisagés, on pourra vouloir assurer un certain périmètre de sécurité autour de chacun des sites. Nous prendrons en compte des contraintes d'interdistance minimale imposant que la distance (euclidienne) entre deux antennes soit supérieure à une valeur constante mais dépendante de la nature des sites concernés.

3.1.3.2 Contraintes de portée

La contrainte de liaison de communication entre antennes impose que la distance (euclidienne) entre les deux terminaux d'une liaison soit inférieure à une valeur constante mais dépendante de la portée de chacune des antennes concernées et ceci indépendamment de la fréquence allouée à l'antenne.

3.1.3.3 Contraintes de zones

Le problème applicatif est considéré dans un espace en deux dimensions. Ce cadre est approprié pour traiter des situations où le théâtre des opérations comporte un faible dénivelé ou à des configurations de surveillance maritime. Les ZD, ZPI et ZI sont spécifiées par des polygones simples.

3.1.4 Description des contraintes radioélectriques de déploiement (LocRLFAP)

Notre dernière catégorie de contraintes prend en compte les interférences potentielles entre deux terminaux éloignés ; c'est elle qui garantit la compatibilité entre l'allocation de fréquences (RLFAP) et le déploiement (Loc).

Ces contraintes, dites de compatibilité distante, traduisent le fait que l'écart de fréquences nécessaire entre deux trajets qui se perturbent dépend des polarisations qui leurs sont respectivement affectées. En règle générale, on aura une contrainte en polarisation croisée plus faible qu'en polarisation identique.

- Si la distance entre les deux antennes est inférieure à d_l km, les 2 sites se brouillent si l'écart de fréquence entre les émetteurs d'un site et les terminaux de l'autre est inférieur à Δ_l. On dit que les sites sont à mi-portée.
- Si cette distance est comprise entre d_l et d_h km, les 2 réseaux se brouillent si l'écart de fréquence entre les émetteurs d'un site et les terminaux de l'autre est inférieur à Δ_h, avec $\Delta_l \leq \Delta_h$. On dit que les sites sont en portée.
- Si cette distance est supérieure à d_h km, les 2 sites peuvent émettre sur les mêmes fréquences sans se brouiller. On dit qu'ils sont hors-portée.

3.1.5 Description des exemples de test

Pour valider notre approche, nous disposons de la topologie d'un réseau RLFAP (Figure 3.1) appelé R_{10}, constitué de 10 sites sur lesquels sont implantés des antennes établissant un maximum de 40 trajets.

3.1.5.1 Exemples de test pour le RLFAP

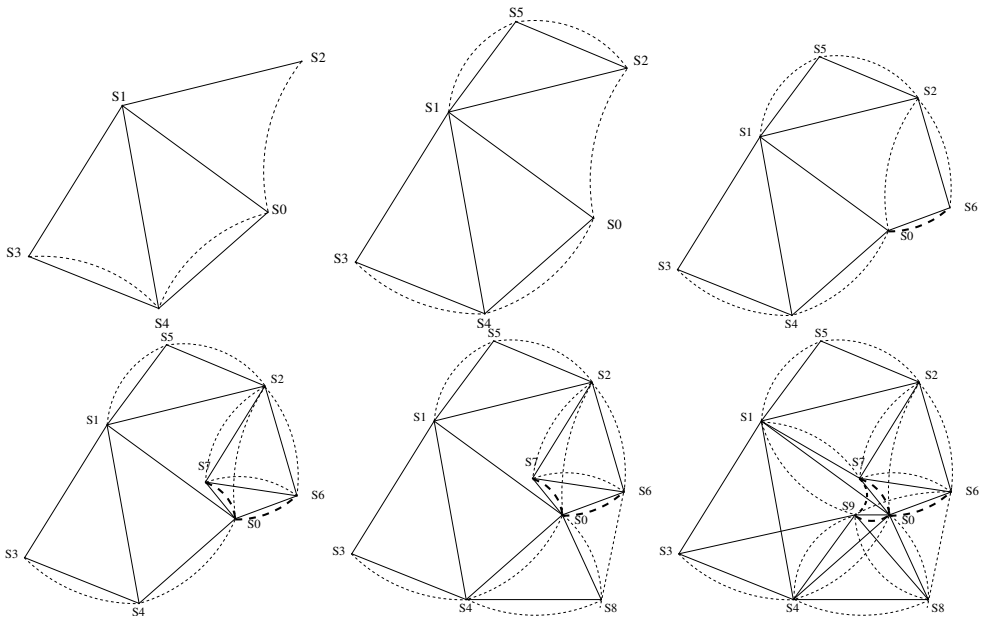


Figure 3.2 – Topologie des exemples de RLFAP de R_5 à R_{10} : réseaux de 5 à 10 sites.

Six instances de sous-réseaux de R_{10}, ordonnés par inclusion, d'une taille variant de 5 à 10 sites, sont étudiés en détail et illustrées sur la Figure 3.2. Les graphes de ces sous-réseaux, nommés de R_5 à R_{10} sont inclus l'un dans l'autre. Les liaisons sont représentées par des lignes pleines, les contraintes de perturbation entre des sites en portée sont données par des arcs en

pointillés et les contraintes (plus fortes) de perturbation entre des sites à mi-portée sont données par des arcs en pointillés gras.

3.1.5.2 Exemples de test pour le LocRLFAP

Les mêmes réseaux sont considérés pour résoudre le LocRLFAP, mais les positions des sites du RLFAP y ont été relâchées. Ils sont présentés sur la Figure 3.3. Seules les contraintes d'interférence distante sont représentées dans le but d'alléger la présentation. Chaque site est circonscrit dans un rectangle identique, centré en la position originelle du site du RLFAP, et tous sont par ailleurs contraints à se trouver dans la zone (rectangulaire) de déploiement.

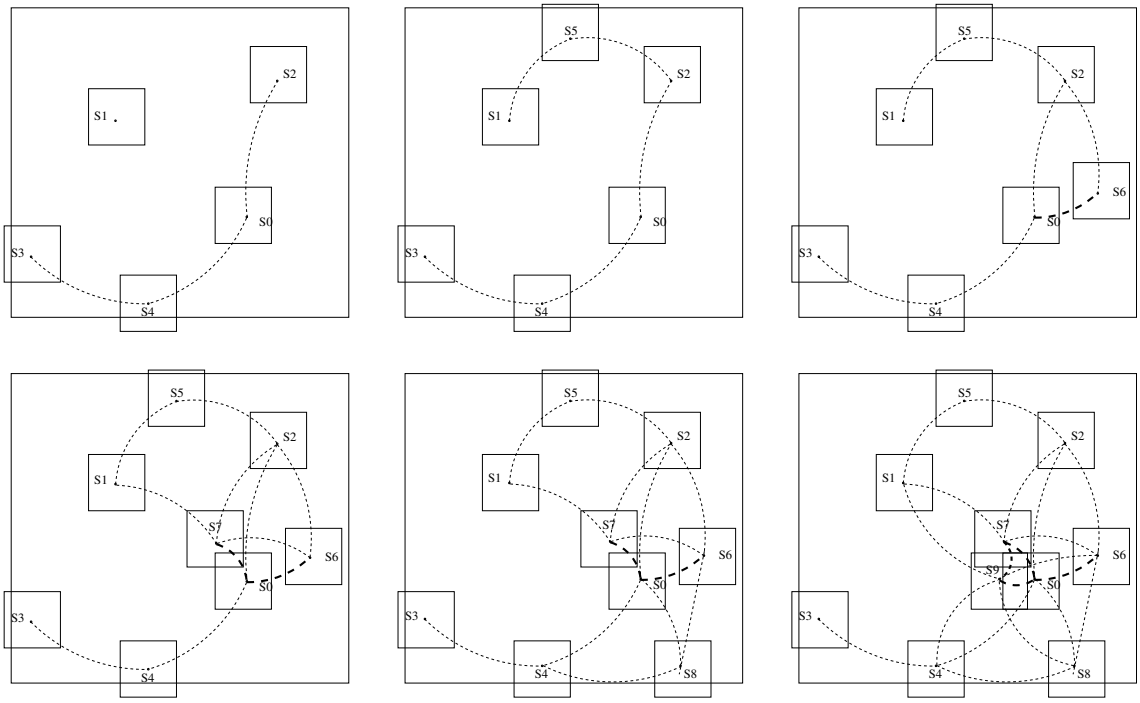


Figure 3.3 – Topologie des exemples de LocRLFAP de L_5 à L_{10} : réseaux de 5 à 10 sites.

3.2 Modèle mathématique du problème

Nous proposons ici un modèle mathématique pour décrire formellement notre problème de déploiement progressif d'antennes.

Cette étape initiale est importante car elle permet d'avoir une spécification rigoureuse de l'ensemble du problème à résoudre. La nature hétérogène des sous problèmes en jeu (allocation de fréquences et localisation des sites) accentue la pertinence d'un modèle mathématique qui s'affranchit de la donnée d'un langage de contraintes (discrètes ou continues).

Nous produisons le modèle en suivant l'ordre que nous avons utilisé dans la description de l'application (RLFAP, Loc puis LocRLFAP) avant de préciser le critère d'optimisation utilisé.

3.2.1 Données du problème

Le problème de déploiement progressif d'antennes est une problème de type mixte, dont les variables relatives à l'allocation de fréquences - les couples (fréquence, polarisation) - sont de type entier, et celles portant sur le déploiement (les coordonnées des antennes) sont de type continu. Définissons ces variables et leurs domaines.

On dispose de m antennes implantées sur n sites $(S_i)_{1\leq i\leq n}$ de position $S_i = (x_i, y_i)$, auxquels on veut attribuer une position et entre lesquels on souhaite établir des liaisons. Le domaine fréquentiel disponible est noté $F = [\ \underline{F}, \overline{F}\ ]$, une liaison entre S_i à S_j est notée l_{ij}, un trajet de S_i à S_j est noté t_{ij} et l'ensemble des trajets que l'opérateur doit pourvoir est noté T. On associe à t_{ij} le domaine fréquentiel $F_{ij} = \{f_{ij} \in F\}$ comme l'ensemble des fréquences admissibles parmi lesquelles on peut choisir la fréquence assignée au trajet. La position d'un site S_i est définie comme le point $(x_i, y_i) \in X_i \times Y_i$, où $X_i \times Y_i$ contient la ZPI.

3.2.2 Modélisation des contraintes radioélectriques

Nous définissons ici quelques notations et le modèle mathématique élaboré pour rendre compte de l'ensemble des contraintes de type radioélectrique qui sont indépendantes du déploiement adopté par les sites.

3.2.2.1 Contraintes co-site

À la pose du problème, l'opérateur définit un certain nombre de constantes en fonction de la sensibilité du matériel utilisé et des ressources disponibles :

- Δ_{er}, la constante d'écart de fréquence minimum entre un émetteur et un récepteur co-sites
- Δ_{ee}, la constante d'écart de fréquence minimum entre deux émetteurs co-sites
- Δ_{rr}, la constante d'écart de fréquence minimum entre deux récepteurs co-sites
- Δ_{eer}, la constante d'écart de fréquence minimum d'inter-modulation entre un récepteur et deux émetteurs co-sites.

Les contraintes de type radioélectrique co-site peuvent alors être modélisées comme suit :

- Contraintes co-site d'interférence :
  - Contraintes émetteur-récepteur :

$$\forall (i,j,k)\ /\ i \neq j, i \neq k, \quad |f_{ij} - f_{ki}| > \Delta_{er}$$

  - Contraintes émetteur-émetteur :

$$\forall (i,j,k)\ /\ i \neq j, i \neq k, j \neq k, \quad |f_{ij} - f_{ik}| > \Delta_{ee}$$

  - Contraintes récepteur-récepteur :

$$\forall (i,j,k)\ /\ i \neq j, i \neq k, j \neq k, \quad |f_{ji} - f_{ki}| > \Delta_{rr}$$

- Contraintes co-site d'inter-modulation (d'ordre 3) :

$$\forall(i,j,k,l) \ / \ i \neq j, i \neq k, j \neq k, l \neq i, \quad \begin{cases} |f_{ij} - 2f_{ik} - f_{li}| > \Delta_{eer} \\ |f_{ik} - 2f_{ij} - f_{li}| > \Delta_{eer} \end{cases}$$

3.2.2.2 Contraintes de liaison

L'opérateur définit aussi l'ensemble des constantes suivantes :

- E_d, l'ensemble des liaisons l_{ij} pour lesquelles on impose un écart duplexe Δ_d
- E_h, l'ensemble des liaisons l_{ij} pour lesquelles on impose un écart harmonique Δ_h

Les contraintes de type radioélectrique qui peuvent apparaître sur les deux trajets de certaines liaisons l_{ij} choisies peuvent alors être modélisées comme suit :

- Contraintes d'écart duplexe

$$\forall(i,j) \in E_d, \quad |f_{ij} - f_{ji}| = \Delta_d$$

- Contraintes d'écart harmonique

$$\forall(i,j) \in E_h, \quad |f_{ij} - f_{ji}| \neq \Delta_h$$

3.2.2.3 Contraintes de matériel ou d'opérateur

L'opérateur définit enfin l'ensemble des constantes suivantes :

- E_{f_i}, l'ensemble des couples (t_{ij}, f_{i_k}) pour lesquels on impose une certaine fréquence f_{i_k} prédéfinie,
- E_{p_h}, l'ensemble des trajets t_{ij} pour lesquels on impose une polarisation horizontale,
- E_{p_v}, l'ensemble des trajets t_{ij} pour lesquels on impose une polarisation verticale,
- $E_{f_=}$, l'ensemble des couples (t_{ij}, t_{kl}) pour lesquels on impose une fréquence égale,
- $E_{f_{\neq}}$, l'ensemble des couples (t_{ij}, t_{kl}) pour lesquels on impose une fréquence différente,
- $E_{p_=}$, l'ensemble des couples (t_{ij}, t_{kl}) pour lesquels on impose une polarisation égale,
- $E_{p_{\neq}}$, l'ensemble des couples (t_{ij}, t_{kl}) pour lesquels on impose une polarisation différente.

Ces ensembles donnés permettent d'imposer pour certaines valeurs (i,j) choisies, la compatibilité entre certains matériels spécifiques :

- Contrainte de fréquence f_{i_k} imposée :

$$\forall(i,j) \ / \ (t_{ij}, f_{i_k}) \in E_{f_i}, \quad f_{ij} = f_{i_k}$$

- Contrainte d'égalité ou d'inégalité de fréquence :

$$\forall(t_{ij}, t_{kl}) \in E_{f_=}, \quad f_{ij} = f_{kl}$$

$$\forall(t_{ij}, t_{kl}) \in E_{f_{\neq}}, \quad f_{ij} \neq f_{kl}$$

- Contrainte de polarisation imposée :

$$\forall t_{ij} \in E_{p_h}, \quad p_{ij} = horizontale$$

$$\forall t_{ij} \in E_{p_v}, \quad p_{ij} = verticale$$

- Contrainte d'égalité ou d'inégalité de polarisation :

$$\forall (t_{ij}, t_{kl}) \in E_{p_=}, \quad p_{ij} = p_{kl}$$

$$\forall (t_{ij}, t_{kl}) \in E_{p_{\neq}}, \quad p_{ij} \neq p_{kl}$$

3.2.2.4 Contraintes de bande

On utilisera ici encore les notations d'intervalles $I = [\underline{I}, \overline{I}]$. On prendra en compte l'ensemble des bandes interdites dans le spectre de fréquences disponible en le définissant par l'union ordonnée des n_b bandes admissibles F_{b_k}, $1 \leq k \leq n_b$

$$F = \bigcup_{k=1}^{n_b} [\ \underline{F_{b_k}}, \overline{F_{b_k}}\ ] \quad \text{avec} \quad \overline{F_{b_k}} < \underline{F_{b_{k+1}}}, \quad \forall\ 1 \leq k \leq n_b - 1,$$

Les fréquences sont contraintes à être espacées régulièrement sur F par un pas Δ_p de fréquences. En écrivant $a \equiv b$ pour $a = b\ (modulo\ \Delta_p)$ on aura donc :

$$\forall f_{ij} \in F, \quad f_{ij} \equiv \underline{F_{b_1}}$$

3.2.3 Modélisation des contraintes de déploiement

Nous abordons ici le sous-problème pur de déploiement de sites. La position des sites n'étant pas connue *a priori*, nous déterminons formellement les contraintes qui permettent de distinguer une configuration valide. Interviennent dans cette estimation l'inter-distance des sites et la donnée de zones admissibles.

3.2.3.1 Contraintes de distance

Le déploiement fait intervenir la fonction distance euclidienne dans un espace de dimension deux qui pourra être $\mathbb{Z}^2$ ou $\mathbb{R}^2$. Cette fonction notée *dist* est définie entre des sites S_i et S_j par :

$$dist(S_i, S_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}.$$

Les contraintes de portée sont déterminées par deux matrices m et M de taille $n \times n$, où la valeur $m_{i,j}$ (i-ème ligne et j-ème colonne) donne la distance minimale entre deux sites S_i et S_j, et où la valeur $M_{i,j}$ (i-ème ligne et j-ème colonne) donne la portée maximale de l'antenne en S_i servant à relier le site S_j.

On a alors les contraintes suivantes :

- Contraintes de distance minimale

$$\forall i \neq j, \quad dist(S_i, S_j) \geq m_{i,j}$$

- Contraintes de portée maximale.

$$\forall l_{ij}, \quad dist(S_i, S_j) \leq M_{i,j}$$

On pourra par convention définir $M_{ij} = +\infty$ si on ne souhaite pas établir de liaison entre S_i et S_j.

3.2.3.2 Contraintes de zones

Un polygone P_i peut être spécifié géométriquement de manière unique, par $(p_{i_1}, \ldots, p_{i_{q_i}})$ la liste ordonnée de ses sommets, où $p_{i_j} = (x_{i_j}, y_{i_j})$. Il faut cependant nous munir d'une représentation adaptée de cet espace pour l'exprimer dans un solveur de contraintes. Lorsque le polygone est convexe, on peut représenter son intérieur par l'intersection des demi-espaces définis par l'ensemble de ses arrêtes, et on peut sans perte de généralité supposer que les ZPI sont convexes[1].

On définira donc le modèle de zones de déploiement par

$$\begin{cases} \forall i \quad S_i \in ZPI_i \\ \forall i \quad ZPI_i = \bigwedge_{j=1}^{q_i} (a_{i_j} x + b_{i_j} y + c_{i_j} \geq 0), \end{cases}$$

où les $(a_{i_j}, b_{i_j}, c_{i_j})$ sont déterminés par la suite des sommets $(p_{i_j}, p_{i_{j+1}})_{(1 \leq j \leq q_i)}$ avec par convention $p_{i_{q_i+1}} = p_{i_1}$.

De même, si les zones interdites spécifiées sont des polygones non-convexes, on pourra les représenter par une union de polygones convexes. On représentera donc les contraintes de zones interdites par une union de n_{ZI} disjonctions de la forme :

$$\begin{cases} \forall i \quad ZI_i = \bigvee_{k=1}^{q_i} (d_{i_k} x + e_{i_k} y + f_{i_k} \geq 0), \\ \forall i \quad S_i \in \bigcup_{j=1}^{n_{ZI}} ZI_j \end{cases}$$

3.2.4 Modélisation des contraintes de déploiement avec allocation de fréquences

La donnée du problème fournit deux constantes Δ_l (resp. Δ_h) définissant l'écart de fréquence minimum exigé afin de garantir la compatibilité distante entre deux sites situés à mi-portée (resp. en portée). Les contraintes de type radioélectrique distantes sont modélisées comme :

- Contraintes de compatibilité radio à mi-portée

$$\forall (i,j,k,l) \ / \ i \neq j, i \neq k, j \neq l, \quad ((dist(S_i, S_j) \geq d_l) \ \vee \ (|f_{ik} - f_{lj}| > \Delta_l))$$

- Contraintes de compatibilité radio en portée

$$\forall (i,j,k,l) \ / \ i \neq j, i \neq k, j \neq l, \quad ((d_h \geq dist(S_i, S_j) \geq d_l) \ \vee \ (|f_{ik} - f_{lj}| > \Delta_h))$$

[1] En effet, il suffit sinon de considérer l'enveloppe convexe du polygone initialement spécifié et d'ajouter des ZI au modèle.

3.2.5 Modélisation du critère d'optimisation

L'objectif opérationnel est de minimiser la taille de la bande de fréquences nécessaire à une résolution du problème respectant l'ensemble des contraintes posées. Ce critère est atteint en introduisant une variable objectif k représentant le nombre de fréquences allouées au total pour résoudre le problème :

$$k = max(\bigcup_{i,j}\{f_{ij}\})$$

On pourra remarquer que contrairement au problème d'allocation de fréquences, le problème traité ne se réduit plus à celui de minimiser le nombre de couleurs nécessaires pour colorier un graphe. En effet, pour le problème d'allocation de fréquences pur, on modélise le réseau par un graphe dans lequel les antennes sont les noeuds et les arêtes représentent une interférence possible entre les deux antennes. Dans notre problème, les antennes ne sont pas toutes placées *a priori*, et l'on ne sait donc pas entre quels noeuds il y a des arêtes dans le graphe.

3.3 Modèles discrets et hybrides en contraintes

En nous basant sur le modèle défini en 3.2, nous proposons parallèlement deux transcriptions en contraintes du problème de déploiement d'antennes : l'une n'utilise que les contraintes disponibles originellement dans Eclair, l'autre fait intervenir la coopération avec un algorithme générique de résolution de contraintes continues développé en Claire.

La première approche montre la faisabilité d'un prototype industriel en contraintes et nous sert de modèle de référence. Bien que la nature du problème de positionnement soit foncièrement continue, puisque les sites doivent être placés dans des zones de $\mathbb{R}^2$, nous proposons d'abord une modélisation du problème dans le sous-espace $\mathbb{Z}^2$. Elle amène à sacrifier l'ensemble des solutions de déploiement dans $\mathbb{R}^2\backslash\mathbb{Z}^2$, et lorsque nous parlons de la position du site $S_i = (x_i, y_i)$ dans ce modèle, x_i et y_i sont des entiers et les optima atteints sont ceux que l'on obtient dans cet espace. En revanche, cette restriction rend possible la transcription du problème dans la plupart des solveurs de contraintes commercialisés, sans exiger de développements supplémentaires. En particulier, c'est l'approche la plus immédiate pour Thales, dont le solveur Eclair© implémente toutes les contraintes en jeu, pour résoudre le problème sur les entiers. La modélisation discrète se fait donc à l'aide de ce solveur.

Le modèle mathématique, défini pour le problème de déploiement d'antennes, nous autorise à proposer au même titre un modèle hybride et autorisant d'appréhender des solutions dans tout $\mathbb{R}^2$ plutôt que de se restreindre à des solutions de déploiement dans $\mathbb{Z}^2$. Parce que la restriction à des positions entières peut causer la perte de solutions optimales localisées sur des positions réelles, nous levons cette contrainte d'intégralité et étudions la pertinence d'une modélisation hybridant des contraintes discrètes pour l'allocation de fréquences, continues pour le déploiement et mixtes pour la compatibilité distante. Nous remplaçons dans cet autre modèle les variables donnant la position des sites et la distance entre sites par des variables continues. Nous avons développé dans Eclair© un module de propagation de contraintes par intervalles implémentant l'algorithme HC4 ; cet algorithme générique nous donne en particulier les moyens d'exprimer les contraintes de distance euclidienne intervenant dans notre application. Les contraintes discrètes d'allocation de fréquence restent inchangées dans le modèle hybride. Le solveur coordonne la propagation de contraintes discrètes pour l'allocation de fréquences et de contraintes continues

pour le déploiement, en garantissant la cohérence des disjonctions entre contraintes discrètes et continues.

Nous distinguons dans notre présentation les contraintes discrètes simples des contraintes de type non-linéaire discrètes et continues, puis nous introduisons celles posant une disjonction entre les premières et ces deux dernières. Enfin nous abordons la modélisation en contraintes du critère d'optimisation.

3.3.1 Modélisation des contraintes discrètes logiques et linéaires

Les contraintes discrètes du LocRLFAP s'expriment foncièrement toutes comme des combinaisons logiques de contraintes arithmétiques. Elles ont cependant chacune un mécanisme de propagation qui leur est propre, afin d'assurer une plus grande efficacité dans la résolution. Elles s'expriment comme suit.

3.3.1.1 Contraintes unaires

Les contraintes unaires d'(in)égalité ou de différence s'expriment comme :

$$(\texttt{u} : \texttt{Var})\ \texttt{Rel}\ (\texttt{c} : \texttt{integer}) \quad \text{vérifiée ssi} \quad u\ Rel\ c.$$

où $\texttt{Rel} \in \{=, \neq, >, \geq, <, \leq\}$. Ceci nous permet d'exprimer les contraintes de matériel ou d'opérateur.

3.3.1.2 Contraintes linéaires

Les contraintes ternaires d'inter-modulation s'expriment à l'aide de la contrainte linéaire

$$(\texttt{terme\_g})\ \texttt{Rel}\ (\texttt{terme\_d}) \quad \text{vérifiée ssi} \quad terme\_g\ Rel\ terme\_d$$

où $\texttt{Rel} \in \{=, \neq, >, \geq, <, \leq\}$ et où un terme gauche (resp. droit) `terme_g` (resp. `terme_d`) est une formule obtenue à partir de variables entières et de constantes avec les opérateurs $+$ et $-$.

3.3.1.3 Contraintes de valeur absolue

Les contraintes d'écart entre fréquences s'expriment à l'aide d'une contrainte de distance en une dimension

$$\texttt{distxyc}(\texttt{u} : \texttt{Var}\ ,\ \texttt{v} : \texttt{Var}\ ,\ \texttt{c} : \texttt{integer}\ ,\ \texttt{Rel} : \texttt{symbol}) \quad \text{vérifiée ssi} \quad |u - v|\ Rel\ c.$$

où $\texttt{Rel} \in \{=, \neq, >, \geq, <, \leq\}$. Ceci nous permet d'exprimer les contraintes co-site d'interférence et les contraintes d'écart duplexe ou harmonique.

3.3.1.4 Contraintes booléennes

Le solveur fournit en outre des contraintes booléennes pour poser la négation ou la conjonction de contraintes :

$$\begin{aligned} \texttt{or}(\texttt{c1} : \texttt{Constraint}\ ,\ \texttt{c2} : \texttt{Constraint}) &\quad \text{vérifiée ssi} \quad c1 \vee c2, \\ \texttt{and}(\texttt{c1} : \texttt{Constraint}\ ,\ \texttt{c2} : \texttt{Constraint}) &\quad \text{vérifiée ssi} \quad c1 \wedge c2. \end{aligned}$$

Ces constructeurs permettent ainsi d'exprimer les contraintes d'appartenance à une zone valide de déploiement. Une ZI s'écrit comme une disjonction entre des contraintes d'inéquations linéaires et les limites d'une ZPI s'expriment par une conjonction de contraintes d'inéquations linéaires.

3.3.2 Modélisation discrète des contraintes non-linéaires

La prise en compte de la distance euclidienne dans le LocRLFAP suppose la modélisation de contraintes non-linéaires dans le solveur. La contrainte de carré est en fait suffisante pour obtenir une implémentation « méta-programmée » de la distance euclidienne.

3.3.2.1 Contrainte de carré

Les contraintes non-linéaires de distance euclidienne du LocRLFAP s'expriment à l'aide de la contrainte `square`, contraignant le carré d'une variable :

$$\texttt{square(x : Var , xs : Var)} \quad \text{vérifiée ssi} \quad x^2 = xs$$

On notera que cette contrainte fournit un meilleur filtrage que la contrainte plus générale de produit de deux variables, comme elle permet de prendre en compte le problème de dépendance (*cf.* p. 18) que l'on aurait entre les deux variables du produit.

3.3.2.2 Contrainte de distance euclidienne

On obtient la contrainte de distance euclidienne par décomposition, en introduisant des variables intermédiaires et en utilisant la contrainte `square()` et des contraintes linéaires.

$$\begin{aligned}
\texttt{(dx : Var)} = (\texttt{(xi : Var)} - \texttt{(xj : Var)}) \quad &\text{vérifiée ssi} \quad dx = xi - xj \\
\texttt{(dy : Var)} = (\texttt{(yi : Var)} - \texttt{(yj : Var)}) \quad &\text{vérifiée ssi} \quad dy = yi - yj \\
\texttt{square(dx, dxs)} \quad &\text{vérifiée ssi} \quad (dx)^2 = dxs \\
\texttt{square(dy, dys)} \quad &\text{vérifiée ssi} \quad (dy)^2 = dys \\
\texttt{dist(xi : Var, xj : Var, yi : Var, yj : Var, Rel : symbol, c : integer)} \quad & \\
&\text{vérifiée ssi} \quad dxs + dys \; Rel \; c^2.
\end{aligned}$$

Nous utiliserons par commodité d'écriture la formule

$$\texttt{dist(Si : Site , Sj : Site , Rel : symbol , c : integer)} \quad \text{vérifiée ssi} \quad dist(S_i, S_j)^2 \; Rel \; c^2.$$

3.3.3 Modélisation continue des contraintes non-linéaires

Le module de propagation de contrainte permet d'exprimer des contraintes respectant la grammaire suivante :

$$\begin{aligned}
cts_constraint \quad ::= \quad & f = f \mid f >= f \mid f <= f \\
f \quad ::= \quad & f + f \mid f - f \mid f * f \mid f/f \mid \\
& sqr(f) \mid sqrt(f) \mid f^{\;entier} \mid \\
& (f) \mid -f \mid +f \mid \\
& nombre \mid intervalle \mid RVar \mid \\
& log(f) \mid exp(f) \mid \\
& trigo(f)
\end{aligned}$$

où *trigo* est une fonction trigonométrique quelconque. Les contraintes de distance peuvent donc être fournies directement au solveur par l'expression de la formule donnant la fonction distance euclidienne.

$$\begin{aligned}\texttt{cts\_dist}((\texttt{xi} : \texttt{RVar}, \texttt{yi} : \texttt{RVar}), (\texttt{xj} : \texttt{RVar}, \texttt{yj} : \texttt{RVar}), \texttt{Rel} : \texttt{symbol}, \texttt{c} : \texttt{float}) \\ := \ \texttt{sqr}(\texttt{xi} - \texttt{xj}) + \texttt{sqr}(\texttt{yi} - \texttt{yj}) \ \texttt{Rel} \ \texttt{sqr}(\texttt{c}) \\ \text{vérifiée ssi} \quad (x_i - x_j)^2 + (y_i - y_j)^2 \ \geq \ c^2.\end{aligned}$$

L'algorithme de propagation garantit alors pour tout point de choix la consistance aux bornes de la contrainte et par conséquent la validité de la contrainte en chaque point solution dans l'espace continu.

3.3.4 Modélisation des disjonctions

Le passage du modèle de RLFAP à celui de LocRLFAP introduit des contraintes de compatibilité distante qui posent un choix entre

- une contrainte sur la position relative des sites, soit des contraintes non-linéaires (Section 3.3.2) et
- une contrainte sur les valeurs que peuvent prendre les fréquences des antennes sur ces sites, soit une combinaison de contraintes logiques et linéaires (Section 3.3.1).

On a donc des disjonctions de nature hybride, la nature des premières étant essentiellement continue et celle des secondes foncièrement discrète. Bien que cette appellation soit quelque peu impropre, considérant que les deux types de contraintes restent discrètes, nous l'utilisons par anticipation de la suite de notre présentation où les disjonctions seront entre des contraintes portant sur des variables discrètes et continues.

3.3.4.1 Disjonctions du LocRLFAP discret

Dans le modèle discret du LocRLFAP, les disjonctions dans les contraintes de compatibilité distante en portée ou à mi-portée s'expriment sous la forme suivante :

$$\begin{aligned}\texttt{or}(\texttt{dist?}(\texttt{Si}, \texttt{Sj}, \geq, \texttt{d}_\texttt{l}) \ , \ \texttt{distxyc?}(\texttt{f}_\texttt{il}, \texttt{f}_\texttt{jk}, \geq, \Delta_\texttt{l}) \\ \text{vérifiée ssi} \quad dist(S_i, S_j)^2 \ \geq \ {d_l}^2 \quad \vee \quad |f_{il} - f_{jk}| \geq \Delta_l\end{aligned}$$

Pour le RLFAP, les positions des sites sont fixées *a priori*, et les disjonctions sont trivialement résolues à la pose du problème. Pour le LocRLFAP ceci n'est plus vrai et nécessite une modélisation plus fine.

3.3.4.2 Disjonctions du LocRLFAP hybride

Dans le modèle hybride du LocRLFAP, le solveur coordonne la propagation de contraintes discrètes pour l'allocation de fréquences et de contraintes continues pour le déploiement, en garantissant la cohérence des disjonctions hybrides

$$\begin{aligned}\texttt{or}(\texttt{cts\_dist?}(\texttt{Si}, \texttt{Sj}, \geq, \texttt{d}_\texttt{l}) \ , \ \texttt{distxyc?}(\texttt{f}_\texttt{il}, \texttt{f}_\texttt{jk}, \geq, \Delta_\texttt{l}) \\ \text{vérifiée ssi} \quad cts\_dist(S_i, S_j)^2 \ \geq \ {d_l}^2 \quad \vee \quad |f_{il} - f_{jk}| \geq \Delta_l\end{aligned}$$

L'algorithme suivant garantit la cohérence des deux systèmes de contraintes (discrets et continus) à tout moment : lorsque la branche continue d'une disjonction hybride devient fausse, la contrainte discrète est propagée suivant la manière usuelle dans le solveur discret. Lorsque la branche discrète d'une disjonction hybride devient fausse, la contrainte continue est ajoutée au store des contraintes continues, et la propagation de l'ensemble des contraintes continues s'effectue jusqu'à l'obtention d'un point fixe ; toutes les disjonctions hybrides sont alors révisées afin de vérifier si aucune de leurs branches continues n'a été violée au cours de la propagation du store de contraintes continues. Ce schéma s'inscrit dans le cadre introduit à la Section 1.4.2.

3.3.4.3 Factorisation des disjonctions du LocRLFAP

Si l'on veut éviter d'obtenir un modèle trop lourd, il vaut mieux factoriser les disjonctions hybrides du RLFAP. L'objectif qui nous intéresse alors est d'écrire une contrainte qui regroupe l'ensemble des contraintes sur les fréquences des antennes. On définit ainsi :

$$\texttt{Compat\_freq}(\texttt{Si} : \texttt{Site}\ ,\ \texttt{Sj} : \texttt{Site}\ ,\ \Delta : \texttt{integer}) \tag{3.1}$$

$$\text{vérifiée ssi} \quad \begin{cases} \forall (k,l)\ /\ k \neq j\ \vee\ l \neq i, \quad (|f_{ik} - f_{lj}| > \Delta) \\ \forall (k,l)\ /\ k \neq j\ \vee\ l \neq i, \quad (|f_{jk} - f_{li}| > \Delta) \end{cases}$$

Cette contrainte est donnée au solveur par une conjonction sur l'ensemble des contraintes élémentaires. Les deux types contraintes de compatibilité distante s'expriment alors par les expressions :

$$\begin{aligned} \texttt{Compat\_h(i,j)} \ &= \ (\texttt{or}(\texttt{dist?}(\texttt{Sites[i]}, \texttt{Sites[j]}, \geq, \texttt{d}_\texttt{h}), \\ &\qquad \texttt{Compat\_freq?}(\texttt{Sites[i]}, \texttt{Sites[j]}, \texttt{i}, \texttt{j}, \Delta_\texttt{h}, \texttt{T}) \end{aligned} \tag{3.2}$$

$$\text{vérifiée ssi} \quad \begin{cases} \forall (k,l)\ /\ k \neq j\ \vee\ l \neq i, \quad (|f_{ik} - f_{lj}| > \Delta_h) \vee ((dist(S_i, S_j) \geq d_h) \\ \forall (k,l)\ /\ k \neq j\ \vee\ l \neq i, \quad (|f_{jk} - f_{li}| > \Delta_h) \vee ((dist(S_i, S_j) \geq d_h) \end{cases} \quad \text{et}$$

$$\begin{aligned} \texttt{Compat\_l(i,j)} \ &= \ (\texttt{or}(\texttt{dist?}(\texttt{Sites[i]}, \texttt{Sites[j]}, \geq, \texttt{d}_\texttt{l}), \\ &\qquad \texttt{Compat\_freq?}(\texttt{Sites[i]}, \texttt{Sites[j]}, \texttt{i}, \texttt{j}, \Delta_\texttt{l}, \texttt{T}) \end{aligned} \tag{3.3}$$

$$\text{vérifiée ssi} \quad \begin{cases} \forall (k,l)\ /\ k \neq j\ \vee\ l \neq i, \quad (|f_{ik} - f_{lj}| > \Delta_l) \vee (dist(S_i, S_j) \geq d_l) \\ \forall (k,l)\ /\ k \neq j\ \vee\ l \neq i, \quad (|f_{jk} - f_{li}| > \Delta_l) \vee (dist(S_i, S_j) \geq d_l) \end{cases} .$$

On peut cependant remarquer que, si par exemple chacun des termes gauches de la disjonction est violé, alors le terme droit le moins contraignant sera posé inutilement. Il semble donc préférable de poser cette disjonction ternaire sur deux niveaux :

$$\begin{aligned} \texttt{Compat\_lh(i,j)} \ &= \ (\texttt{or}(\texttt{dist?}(\texttt{Sites[i]}, \texttt{Sites[j]}, \geq, \texttt{d}_\texttt{h}), \\ &\qquad \texttt{or?}(\texttt{and?}(\texttt{dist?}(\texttt{Sites[i]}, \texttt{Sites[j]}, \geq, \texttt{d}_\texttt{l}), \\ &\qquad\qquad \texttt{Compat\_freq?}(\texttt{Sites[i]}, \texttt{Sites[j]}, \texttt{i}, \texttt{j}, \Delta_\texttt{l})), \\ &\qquad\quad \texttt{Compat\_freq?}(\texttt{Sites[i]}, \texttt{Sites[j]}, \texttt{i}, \texttt{j}, \Delta_\texttt{h})))) \end{aligned} \tag{3.4}$$

$$\text{vérifiée ssi} \quad \begin{cases} \begin{cases} \forall (k,l)\ /\ k \neq j\ \vee\ l \neq i, \quad (|f_{ik} - f_{lj}| > \Delta_l) \vee (dist(S_i, S_j) \geq d_l) \\ \forall (k,l)\ /\ k \neq j\ \vee\ l \neq i, \quad (|f_{jk} - f_{li}| > \Delta_l) \vee (dist(S_i, S_j) \geq d_l) \end{cases} \\ \begin{cases} \forall (k,l)\ /\ k \neq j\ \vee\ l \neq i, \quad (|f_{ik} - f_{lj}| > \Delta_h) \vee ((d_h \geq dist(S_i, S_j) \geq d_l) \\ \forall (k,l)\ /\ k \neq j\ \vee\ l \neq i, \quad (|f_{jk} - f_{li}| > \Delta_h) \vee ((d_h \geq dist(S_i, S_j) \geq d_l) \end{cases} \end{cases}$$

Bien que cette forme permet de gagner en performance sur quelques exemples, elle en rend d'autres beaucoup plus difficiles à résoudre. Nous préférons donc utiliser la première forme dans nos modèles.

3.3.5 Modélisation en contraintes du critère d'optimisation

La résolution du problème d'optimisation se fait grâce à l'introduction dans le modèle d'une contrainte reliant les variables de décision du problème à une variable donnant la valeur de la fonction objectif. La contrainte `maxlist` établit le lien entre les variables donnant les fréquences allouées aux différents trajets et une variable contrainte au maximum d'entre elles. On pose :

$$\texttt{maxlist}(\texttt{LF} : \texttt{list}[\texttt{Var}]\,,\, \texttt{Fmax} : \texttt{Var}) \quad \text{vérifiée ssi} \quad \max_{f \in LF}(f) = Fmax$$

L'obtention d'une valeur minimale de la variable F_{max} donne alors une preuve d'optimalité pour une solution.

3.4 Première résolution et analyse

Nous abordons à présent une résolution du RLFAP et du LocRLFAP dans chacun des modèles en contraintes discrets et continus.

Nous commençons par produire les données complètes des exemples traités et détaillons ensuite l'expression de l'exemple le plus simple des réseaux. Nous justifions alors le choix du type de domaines utilisés et donnons les heuristiques mises en oeuvre dans l'algorithme de recherche développé pour notre première étude.

Nous exposons enfin les résultats numériques associés et concluons. Le modèle discret est nettement le plus avantageux sous cette première approche, mais pour autant nous verrons à la prochaine section que l'on peut définir une procédure de recherche plus efficace pour le modèle hybride.

3.4.1 Simplification et données du problème

Nous ne disposons pas de données numériques complètes pour aborder le LocRLFAP et nous proposons la résolution d'un problème simplifié. Ce modèle ne prend pas en compte :

- les contraintes d'inter-modulation,
- les contraintes de zones interdites,
- la polarisation des trajets,

et les zones potentielles d'implantation sont limitées à des régions rectangulaires dont les côtés sont parallèles aux axes du repère. Les données numériques utilisées sont les suivantes :

- Le spectre de fréquences disponibles est $F = [0, 400]$ avec un domaine identique pour chaque antenne. L'objectif est de minimiser la fréquence maximale utilisée.
- Les contraintes radio-électriques imposées sont les suivantes :
  (1): Contraintes co-site
  $\Delta_{er} = 16$ - contraintes émetteur récepteur
  $\Delta_{ee} = 0$ - contraintes émetteur émetteur
  $\Delta_{rr} = 0$ - contraintes récepteur récepteur

(2): Contraintes distantes
$\Delta_h = 4$ - contraintes entre un émetteur i et un récepteur j en portée
$\Delta_l = 8$ - contraintes entre un émetteur i et un récepteur j à mi-portée
(3): Contraintes duplexes
$\Delta_d = 20$ - contraintes entre les deux trajets d'une liaison
(4): Contraintes sur le spectre de fréquences
$\Delta_p = 4$ - contrainte de distance entre les fréquences du spectre
$F_{b_i} = [(2i-1) * \lfloor \frac{400}{12} \rfloor, 2i * \lfloor \frac{400}{12} \rfloor], \quad 1 \leq i \leq 6$ - contraintes de bande

- Les positions des sites sont données en coordonnées cartésiennes par $S_0 = (93, 44), S_1 = (37, 85), S_2 = (106, 102), S_3 = (1, 27), S_4 = (51, 7), S_5 = (63, 120), S_6 = (120, 54), S_7 = (80, 60), S_8 = (110, 7)$ et $S_9 = (78, 44)$.
- La zone de déploiement est donnée par $ZD = [0, 150] \times [0, 130]$ et pour chaque site localisé originellement en $S_i = (x_i, y_i)$ sa zone potentielle d'implantation est agrandie à $ZPI_i = [x_i - 15, x_i + 15] \times [y_i - 15, y_i + 15]$.
- Les liaisons à établir entre les sites sont présentées par des lignes pleines sur la Figure 3.2, et les contraintes de compatibilité distante à mi-portée et en portée sont définies par les constantes $d_l = 30$ et $d_h = 60$.

3.4.2 Expression des modèles L_5 et R_5 dans le modèle naïf

Exprimons en contraintes le modèle développé à la section 3.2 en l'adaptant aux données simplifiées de 3.4.1. On obtient pour L_5 (Figure 3.2, p.76) le système de contraintes suivant :

$$
\begin{array}{lllll}
or(dist(S_0,S_1)\geq 30, & or(dist(S_0,S_1)\geq 60, & or(dist(S_1,S_2)\geq 30, & or(dist(S_1,S_2)\geq 60, & or(dist(S_2,S_3)\geq 60,\\
\quad and(|f_{01}-f_{21}|\geq 8, & \quad and(|f_{01}-f_{21}|\geq 4, & \quad and(|f_{21}-f_{01}|\geq 8, & \quad and(|f_{21}-f_{01}|\geq 4, & \quad and(|f_{21}-f_{13}|\geq 4,\\
\qquad |f_{01}-f_{31}|\geq 8, & \qquad |f_{01}-f_{31}|\geq 4, & \qquad |f_{21}-f_{31}|\geq 8, & \qquad |f_{21}-f_{31}|\geq 4, & \qquad |f_{21}-f_{43}|\geq 4,\\
\qquad |f_{01}-f_{41}|\geq 8, & \qquad |f_{01}-f_{41}|\geq 4, & \qquad |f_{21}-f_{41}|\geq 8) & \qquad |f_{21}-f_{41}|\geq 4) & \qquad |f_{31}-f_{12}|\geq 4,\\
\qquad |f_{04}-f_{21}|\geq 8, & \qquad |f_{04}-f_{21}|\geq 4, & or(dist(S_1,S_3)\geq 30, & or(dist(S_1,S_3)\geq 60, & \qquad |f_{34}-f_{12}|\geq 4)\\
\qquad |f_{04}-f_{31}|\geq 8, & \qquad |f_{04}-f_{31}|\geq 4, & \quad and(|f_{10}-f_{43}|\geq 8, & \quad and(|f_{10}-f_{43}|\geq 4, & or(dist(S_2,S_4)\geq 60,\\
\qquad |f_{04}-f_{41}|\geq 8, & \qquad |f_{04}-f_{41}|\geq 4, & \qquad |f_{12}-f_{43}|\geq 8, & \qquad |f_{12}-f_{43}|\geq 4, & \quad and(|f_{21}-f_{04}|\geq 4,\\
\qquad |f_{10}-f_{40}|\geq 8, & \qquad |f_{10}-f_{40}|\geq 4, & \qquad |f_{13}-f_{43}|\geq 8, & \qquad |f_{13}-f_{43}|\geq 4, & \qquad |f_{21}-f_{14}|\geq 4,\\
\qquad |f_{12}-f_{40}|\geq 8, & \qquad |f_{12}-f_{40}|\geq 4, & \qquad |f_{14}-f_{43}|\geq 8, & \qquad |f_{14}-f_{43}|\geq 4, & \qquad |f_{21}-f_{34}|\geq 4,\\
\qquad |f_{13}-f_{40}|\geq 8, & \qquad |f_{13}-f_{40}|\geq 4, & \qquad |f_{31}-f_{01}|\geq 8, & \qquad |f_{31}-f_{01}|\geq 4, & \qquad |f_{40}-f_{12}|\geq 4,\\
\qquad |f_{14}-f_{40}|\geq 8) & \qquad |f_{14}-f_{40}|\geq 4) & \qquad |f_{31}-f_{21}|\geq 8, & \qquad |f_{31}-f_{21}|\geq 4, & \qquad |f_{41}-f_{12}|\geq 4,\\
or(dist(S_0,S_2)\geq 30, & or(dist(S_0,S_2)\geq 60, & \qquad |f_{31}-f_{41}|\geq 8, & \qquad |f_{31}-f_{41}|\geq 4, & \qquad |f_{43}-f_{12}|\geq 4)\\
\quad and(|f_{01}-f_{12}|\geq 8, & \quad and(|f_{01}-f_{12}|\geq 4, & \qquad |f_{34}-f_{01}|\geq 8, & \qquad |f_{34}-f_{01}|\geq 4, & or(dist(S_3,S_4)\geq 60,\\
\qquad |f_{04}-f_{12}|\geq 8, & \qquad |f_{04}-f_{12}|\geq 4, & \qquad |f_{34}-f_{21}|\geq 8, & \qquad |f_{34}-f_{21}|\geq 4, & \quad and(|f_{31}-f_{04}|\geq 4,\\
\qquad |f_{21}-f_{10}|\geq 8, & \qquad |f_{21}-f_{10}|\geq 4, & \qquad |f_{34}-f_{41}|\geq 8) & \qquad |f_{34}-f_{41}|\geq 4) & \qquad |f_{31}-f_{14}|\geq 4,\\
\qquad |f_{21}-f_{40}|\geq 8) & \qquad |f_{21}-f_{40}|\geq 4) & or(dist(S_1,S_4)\geq 30, & or(dist(S_1,S_4)\geq 60, & \qquad |f_{34}-f_{04}|\geq 4,\\
or(dist(S_0,S_3)\geq 30, & or(dist(S_0,S_3)\geq 60, & \quad and(|f_{10}-f_{04}|\geq 8, & \quad and(|f_{10}-f_{04}|\geq 4, & \qquad |f_{34}-f_{14}|\geq 4,\\
\quad and(|f_{01}-f_{13}|\geq 8, & \quad and(|f_{01}-f_{13}|\geq 4, & \qquad |f_{10}-f_{34}|\geq 8, & \qquad |f_{10}-f_{34}|\geq 4, & \qquad |f_{40}-f_{13}|\geq 4,\\
\qquad |f_{01}-f_{43}|\geq 8, & \qquad |f_{01}-f_{43}|\geq 4, & \qquad |f_{12}-f_{04}|\geq 8, & \qquad |f_{12}-f_{04}|\geq 4, & \qquad |f_{41}-f_{13}|\geq 4,\\
\qquad |f_{04}-f_{13}|\geq 8, & \qquad |f_{04}-f_{13}|\geq 4, & \qquad |f_{12}-f_{34}|\geq 8, & \qquad |f_{12}-f_{34}|\geq 4, & \qquad |f_{43}-f_{13}|\geq 4)\\
\qquad |f_{04}-f_{43}|\geq 8, & \qquad |f_{04}-f_{43}|\geq 4, & \qquad |f_{13}-f_{04}|\geq 8, & \qquad |f_{13}-f_{04}|\geq 4, & \\
\qquad |f_{31}-f_{10}|\geq 8, & \qquad |f_{31}-f_{10}|\geq 4, & \qquad |f_{13}-f_{34}|\geq 8, & \qquad |f_{13}-f_{34}|\geq 4, & or(dist(S_2,S_3)\geq 30,\\
\qquad |f_{31}-f_{40}|\geq 8, & \qquad |f_{31}-f_{40}|\geq 4, & \qquad |f_{14}-f_{04}|\geq 8, & \qquad |f_{14}-f_{04}|\geq 4, & \quad and(|f_{21}-f_{13}|\geq 8\\
\qquad |f_{34}-f_{10}|\geq 8, & \qquad |f_{34}-f_{10}|\geq 4, & \qquad |f_{14}-f_{34}|\geq 8, & \qquad |f_{14}-f_{34}|\geq 4, & \qquad |f_{21}-f_{43}|\geq 8\\
\qquad |f_{34}-f_{40}|\geq 8) & \qquad |f_{34}-f_{40}|\geq 4) & \qquad |f_{40}-f_{01}|\geq 8, & \qquad |f_{40}-f_{01}|\geq 4, & \qquad |f_{31}-f_{12}|\geq 8\\
or(dist(S_0,S_4)\geq 30, & or(dist(S_0,S_4)\geq 60, & \qquad |f_{40}-f_{21}|\geq 8, & \qquad |f_{40}-f_{21}|\geq 4, & \qquad |f_{34}-f_{12}|\geq 8\\
\quad and(|f_{01}-f_{14}|\geq 8, & \quad and(|f_{01}-f_{14}|\geq 4, & \qquad |f_{40}-f_{31}|\geq 8, & \qquad |f_{40}-f_{31}|\geq 4, & or(dist(S_2,S_4)\geq 30,\\
\qquad |f_{01}-f_{34}|\geq 8, & \qquad |f_{01}-f_{34}|\geq 4, & \qquad |f_{41}-f_{01}|\geq 8, & \qquad |f_{41}-f_{01}|\geq 4, & \quad and(|f_{21}-f_{04}|\geq 8\\
\qquad |f_{04}-f_{14}|\geq 8, & \qquad |f_{04}-f_{14}|\geq 4, & \qquad |f_{41}-f_{21}|\geq 8, & \qquad |f_{41}-f_{21}|\geq 4, & \qquad |f_{21}-f_{14}|\geq 8\\
\qquad |f_{04}-f_{34}|\geq 8, & \qquad |f_{04}-f_{34}|\geq 4, & \qquad |f_{41}-f_{31}|\geq 8, & \qquad |f_{41}-f_{31}|\geq 4, & \qquad |f_{21}-f_{34}|\geq 8\\
\qquad |f_{40}-f_{10}|\geq 8, & \qquad |f_{40}-f_{10}|\geq 4, & \qquad |f_{43}-f_{01}|\geq 8, & \qquad |f_{43}-f_{01}|\geq 4, & \qquad |f_{40}-f_{12}|\geq 8\\
\qquad |f_{41}-f_{10}|\geq 8, & \qquad |f_{41}-f_{10}|\geq 4, & \qquad |f_{43}-f_{21}|\geq 8, & \qquad |f_{43}-f_{21}|\geq 4, & \qquad |f_{41}-f_{12}|\geq 8\\
\qquad |f_{43}-f_{10}|\geq 8) & \qquad |f_{43}-f_{10}|\geq 4) & \qquad |f_{43}-f_{31}|\geq 8) & \qquad |f_{43}-f_{31}|\geq 4) & \qquad |f_{43}-f_{12}|\geq 8\\
 & & & & or(dist(S_3,S_4)\geq 30,\\
 & & & & \quad and(|f_{31}-f_{04}|\geq 8,\\
 & & & & \qquad |f_{31}-f_{14}|\geq 8,\\
 & & & & \qquad |f_{34}-f_{04}|\geq 8,\\
 & & & & \qquad |f_{34}-f_{14}|\geq 8,\\
 & & & & \qquad |f_{40}-f_{13}|\geq 8,\\
 & & & & \qquad |f_{41}-f_{13}|\geq 8,\\
 & & & & \qquad |f_{43}-f_{13}|\geq 8)
\end{array}
$$

$|f_{01}-f_{10}|=20$ $|f_{01}-f_{40}|\geq 16$ $|f_{12}-f_{01}|\geq 16$ $|f_{13}-f_{41}|\geq 16$ $|f_{34}-f_{13}|\geq 16$
$|f_{14}-f_{41}|=20$ $|f_{04}-f_{10}|\geq 16$ $|f_{12}-f_{31}|\geq 16$ $|f_{14}-f_{01}|\geq 16$ $|f_{40}-f_{14}|\geq 16$
$|f_{13}-f_{31}|=20$ $|f_{10}-f_{21}|\geq 16$ $|f_{12}-f_{41}|\geq 16$ $|f_{14}-f_{21}|\geq 16$ $|f_{40}-f_{34}|\geq 16$
$|f_{34}-f_{43}|=20$ $|f_{10}-f_{31}|\geq 16$ $|f_{13}-f_{01}|\geq 16$ $|f_{14}-f_{31}|\geq 16$ $|f_{41}-f_{04}|\geq 16$
$|f_{12}-f_{21}|=20$ $|f_{10}-f_{41}|\geq 16$ $|f_{13}-f_{21}|\geq 16$ $|f_{31}-f_{43}|\geq 16$ $|f_{41}-f_{34}|\geq 16$
$|f_{43}-f_{04}|\geq 16$
$|f_{43}-f_{14}|\geq 16$

$15\leq dist(S_0,S_1)\leq 80$ $15\leq dist(S_1,S_2)\leq 80$ $15\leq dist(S_2,S_3)$ $15\leq dist(S_3,S_4)\leq 80$
$15\leq dist(S_0,S_2)$ $15\leq dist(S_1,S_3)\leq 80$ $15\leq dist(S_2,S_4)$
$15\leq dist(S_0,S_3)$ $15\leq dist(S_1,S_4)\leq 80$
$15\leq dist(S_0,S_4)\leq 80$

Voyons maintenant comment ce modèle est simplifié quand on aborde le RLFAP en lieu du LocRLFAP. R_5 (Figure 3.3, p.77) est implanté en $S_0 = (93, 44), S_1 = (37, 85), S_2 = (106, 102), S_3 = (1, 27)$ et $S_4 = (51, 7)$. La distance euclidienne entre chaque site est donnée dans le tableau ci-dessous :

	S_0	S_1	S_2	S_3	S_4
S_0	-	69.40	59.44	93.56	55.97
S_1	-	-	71.06	68.26	79.25
S_2	-	-	-	129.03	109.77
S_3	-	-	-	-	53.85
S_4	-	-	-	-	-

On peut vérifier que le système de contraintes associé à ces positions ne comporte donc pas de contraintes à mi-portée et que l'expression en contraintes du réseau R_5 peut ainsi être simplifiée à partir de L_5 en le système infra :

$|f_{01}-f_{10}|=20$ $|f_{01}-f_{40}|\geq 16$ $|f_{12}-f_{01}|\geq 16$ $|f_{13}-f_{41}|\geq 16$ $|f_{34}-f_{13}|\geq 16$
$|f_{14}-f_{41}|=20$ $|f_{04}-f_{10}|\geq 16$ $|f_{12}-f_{31}|\geq 16$ $|f_{14}-f_{01}|\geq 16$ $|f_{40}-f_{14}|\geq 16$
$|f_{13}-f_{31}|=20$ $|f_{10}-f_{21}|\geq 16$ $|f_{12}-f_{41}|\geq 16$ $|f_{14}-f_{21}|\geq 16$ $|f_{40}-f_{34}|\geq 16$
$|f_{34}-f_{43}|=20$ $|f_{10}-f_{31}|\geq 16$ $|f_{13}-f_{01}|\geq 16$ $|f_{14}-f_{31}|\geq 16$ $|f_{41}-f_{04}|\geq 16$
$|f_{12}-f_{21}|=20$ $|f_{10}-f_{41}|\geq 16$ $|f_{13}-f_{21}|\geq 16$ $|f_{31}-f_{43}|\geq 16$ $|f_{41}-f_{34}|\geq 16$
$|f_{43}-f_{04}|\geq 16$
$|f_{43}-f_{14}|\geq 16$

$|f_{01}-f_{12}|\geq 4$ $|f_{01}-f_{14}|\geq 4$ $|f_{40}-f_{10}|\geq 4$ $|f_{31}-f_{14}|\geq 4$ $|f_{41}-f_{13}|\geq 4$
$|f_{04}-f_{12}|\geq 4$ $|f_{01}-f_{34}|\geq 4$ $|f_{41}-f_{10}|\geq 4$ $|f_{34}-f_{04}|\geq 4$ $|f_{43}-f_{13}|\geq 4$
$|f_{21}-f_{10}|\geq 4$ $|f_{04}-f_{14}|\geq 4$ $|f_{43}-f_{10}|\geq 4$ $|f_{34}-f_{14}|\geq 4$
$|f_{21}-f_{40}|\geq 4$ $|f_{04}-f_{34}|\geq 4$ $|f_{31}-f_{04}|\geq 4$ $|f_{40}-f_{13}|\geq 4$

3.4.3 Choix du type de domaines utilisés

Deux possibilités sont offertes dans Eclair pour la modélisation en contraintes discrètes : le choix de domaines énumérés (dits en extension) et celui de domaines définis en intension. Pour

la résolution de contraintes continues, on utilise exclusivement des variables dont le domaine est donné par un intervalle de nombres flottants.

L'utilisation de domaines en intension pour la contrainte de valeur absolue est implémenté par une disjonction de deux contraintes linéaires. Pour des variables u, v et une constante c, les contraintes sont propagées comme suit :

- pour la contrainte $|u - v| = c$ on pose :

$$\begin{cases} (\mathtt{u} \leq \mathtt{v} + \mathtt{c}) \\ (\mathtt{v} \leq \mathtt{u} + \mathtt{c}) \\ \mathtt{or}((\mathtt{u} = \mathtt{v} + \mathtt{c}), (\mathtt{v} = \mathtt{u} + \mathtt{c})) \end{cases}$$

- pour la contrainte $|u - v| \geq c$ on pose :

$$\mathtt{or}((\mathtt{u} \geq \mathtt{v} + \mathtt{c}), (\mathtt{v} \geq \mathtt{u} + \mathtt{c}))$$

Considérons la seconde contrainte qui a le plus grand nombre d'occurences dans le LocRLFAP. Avec des domaines donnés en intension, si aucune des branches de la disjonction n'est résolue, le filtrage possible pour la contrainte de valeur absolue est nul.

Lorsqu'on utilise une modélisation en extension pour les domaines des variables de trajets, les contraintes de valeur absolue assurent la consistance d'arc de ce sous-ensemble de contraintes.

Algorithme 12 : *abs_geq_revise : maintien des invariants de* $|u - v| \geq c$

si $(\underline{v} + c > \overline{u})$ **alors**
  si $(\underline{v} + c < \underline{u})$ **alors** (u is $\emptyset$)
  sinon (u isless $\overline{v} - c$)
sinon si $(\overline{v} - c < \underline{u})$ **alors**
  si $(\underline{v} + c > \overline{u})$ **alors** (u is $\emptyset$)
  sinon (u ismore $\underline{v} + c$)
sinon si $(\overline{v} - c + 1 \leq \underline{v} + c - 1)$ **alors**
  (u isnotin $[\overline{v} - c + 1, \underline{v} + c - 1]$)
retourner $dom(u)$

Cette propagation, qui se fait en créant des "trous" dans les domaines (*cf.* Algorithme 12) s'est avérée suffisamment avantageuse sur nos premiers tests pour déterminer le choix de ce type de représentation des domaines dans notre résolution étant donné la prépondérance de ces contraintes dans le RLFAP.

On notera que ce choix n'est pas exclusif à notre modèle. Nous pouvons en effet le combiner avec celui de domaines discrets ou flottants donnés en intension pour les variables de position du LocRLFAP.

3.4.4 Algorithmes de recherche

Nous détaillons les spécificités des algorithme de recherche élaborés pour le problème de déploiement d'antennes.

L'efficacité de la résolution dépend fortement de l'ordre du choix des variables et de valeurs lors de la construction de l'arbre de recherche. Nous indiquons les stratégies que nous avons mises en œuvre.

3.4.4.1 Heuristiques de choix de variable

Nous avons dans un premier temps comparé l'efficacité relative des différentes heuristiques disponibles dans le solveur pour des variables discrètes :

- La première, appelée `nextvar`, assigne les variables selon l'ordre arbitraire dans lequel elles sont données au solveur.
- La seconde, `mindomain`, choisit pour prochaine variable, à chaque point de choix, la variable ayant le plus petit domaine courant.

Cette dernière s'est montrée globalement plus efficace sur nos instances et c'est donc celle choisie pour la résolution des contraintes discrètes.

Pour les variables continues, la stratégie de « first-fail » la plus répandue est à l'inverse de celle employée sur les domaines discrets :

- La stratégie `maxdomain`, choisit pour prochaine variable, à chaque point de choix, la variable ayant le plus grand domaine courant.

Elle s'est montrée souvent plus efficace que `nextvar` sur nos exemples, et c'est celle que nous employons pour la résolution de ce jeu de tests.

3.4.4.2 Distinction de la nature des variables

Deux groupes de variables cohabitent dans le LocRLFAP :

- les variables de fréquence,
- les variables de position.

Nous les distinguons dans notre résolution étant donné que les contraintes qui leur sont appliquées sont de force différente. Plusieurs arguments penchent en faveur d'une énumération des variables commençant par les fréquences :

- La fonction objectif du problème d'optimisation porte sur les fréquences.
- Le problème est sous-contraint sur les variables de position.

Ce dernier point indique que l'on risque de rencontrer un grand nombre de solutions réalisables du sous-problème de déploiement. Une énumération commençant par les positions amènerait alors à explorer plusieurs feuilles équivalentes du point de vue des contraintes qu'elles impliquent sur les fréquences. Comme la consistance appliquée sur les contraintes n'est que partielle, il faut généralement explorer une certaine profondeur l'arbre de recherche avant de détecter une éventuelle incohérence du système. La condition d'amélioration de la fonction objectif du *Branch & Bound* risque donc de ne pas être assez forte pour éviter d'explorer inutilement des solutions ayant des positions très proches. Notre première résolution se fait pour cela en énumérant d'abord les variables de fréquence, puis à partir de celles de position.

3.4.4.3 Heuristiques de choix de valeur pour des variables discrètes

Pour les variables discrètes, nous avons expérimenté les stratégies standard suivantes d'énumération du sous-arbre de valeurs :

- `enum` consiste à énumérer dans l'ordre croissant sur toutes les valeurs du domaine
- `dicho` découpe le domaine de la variable courante par dichotomie. À chaque bissection, la propagation est alors déclenchée.

Le second mode d'énumération est le plus efficace dans les problèmes à résoudre et c'est celui que nous utilisons dans toute la suite.

3.4.4.4 Algorithme de recherche hybride

La recherche de solutions dans un espace hybride doit être envisagée avec certaines précautions étant donné les difficultés introduites par la portion continue de l'espace.

La résolution des CSPs continus est généralement obtenue par un algorithme de *Branch & Prune* calculant un pavage récursif de l'espace de recherche jusqu'à une précision donnée. Cet algorithme permet une résolution complète lors de laquelle aucune solution n'est perdue. Le solveur garantit que l'ensemble des solutions se trouvent à l'intérieur des boîtes retournées ; en revanche, il est plus difficile de déterminer si une boîte retournée par le solveur contient effectivement une solution. Sous certaines conditions de dérivabilité des fonctions impliquées dans le système d'équations à résoudre, le test de Moore et Nickel [93] permet de prouver l'existence d'une solution dans une boîte.

Cependant, dans le cas d'un système d'inéquations, ce test n'est pas applicable. Deux possibilités se présentent comme alternative :

- Si la boîte calculée se trouve à l'intérieur d'une région solution et que l'extension aux intervalles des fonctions impliquées est assez serrée, on peut prouver directement que la boîte ne contient que des solutions.
- On peut sinon tester un point pris dans la boîte calculée et vérifier si l'ensemble des contraintes sont satisfaites en ce point.

Dans le cas du problème de déploiement d'antennes, le système est généralement sous-contraint sur les variables continues et ces tests sont effectués avec succès dans chacun de nos exemples. D'un point de vue opérationnel, nous préférons nous contenter d'un algorithme qui n'est pas complet dans l'absolu, que de retourner une boîte dont on n'est pas sûr qu'elle contienne effectivement une solution. Nous utilisons par défaut un algorithme de *Branch & Prune* effectuant une bissection des boîtes jusqu'à une précision de $\varepsilon = 10^{-5}$, auquel succède un test de correction effectué au centre de la boîte si l'intervalle n'est pas canonique et sur la borne gauche sinon.

3.4.4.5 Algorithme de Branch & Bound

La construction de l'arbre de recherche se fait dynamiquement en suivant les heuristiques définies supra. Eu égard au schéma générique donné dans notre état de l'art (*cf.* p.5), la seule différence est donc que notre algorithme construit successivement deux arbres de recherche utilisant chacun des heuristiques qui lui sont propres.

L'algorithme de *Branch & Bound* élémentaire, que nous employons, contraint à chaque solution trouvée que la suivante atteigne une valeur meilleure, en ajoutant au modèle cette contrainte à la variable portant la fonction objectif.

3.4.5 Résultats numériques

Nous avons vu à la Section 3.3.4 que deux écritures du modèle en contraintes du RLFAP se présentaient. Nous rapportons d'abord les résultats expérimentaux obtenus sur le modèle classique de RLFAP, puis ceux donnés en posant le modèle qui anticipe sur le modèle du LocRLFAP en utilisant des disjonctions ; nous indiquons alors les résultats du LocRLFAP dans les modèles discrets et hybrides. Nous terminons en évaluant la qualité des solutions et en mettant en parallèle les résultats avec une analyse de la combinatoire des exemples.

3.4.5.1 Résolution du RLFAP classique

Nous comparons ici (voir 3.1) les performances dans la résolution du RLFAP classique en utilisant respectivement le modèle simple et le modèle du LocRLFAP où toutes les positions des sites sont fixées.

`heur(T)=mindomain, heur(P)=maxdomain`				
	modèle RLFAP		modèle LocRLFAP	
Instance	Temps_{ms}	Backtracks	Temps_{ms}	Backtracks
R_5	0	53	10	=
R_6	0	59	10	=
R_7	20	273	40	=
R_8	400	10.005	1.870	=
R_9	370	6.724	2.130	=
R_{10}	9.300	156.497	94.960	=

Tableau 3.1 – Résolution du RLFAP dans les modèles sans et avec disjonctions

Sans disjonctions, les instances de RLFAP sont résolues rapidement jusqu'à l'optimum avec un algorithme de branch and bound simple. La première solution est toujours trouvée en quelques millisecondes après un petit nombre de backtracks.

La résolution du RLFAP posé avec le modèle du LocRLFAP ne pose pas davantage de problèmes. On peut toutefois remarquer que pour les instances les plus grandes, la résolution est nettement plus coûteuse qu'avec le modèle classique.

3.4.5.2 Résolution des LocRLFAPs discrets et hybrides

Nous détaillons ici l'étude des instances de LocRLFAP illustrées sur la figure 3.3 en utilisant les modèles discrets et hybrides. Nos résultats sont donnés dans le tableau suivant :

`heur(T)=mindomain, heur(P)=maxdomain`				
	modèle discret		modèle hybride	
Instance	Temps_{ms}	Backtracks	Temps_{ms}	Backtracks
L_5	10	80	60	199
L_6	50	94	100	212
L_7	100	111	1.920	914
L_8	350	171	640	346
L_9	1.200	590	695.000	104.819
L_{10}	-	-	-	-

Tableau 3.2 – Résolution du LocRLFAP dans les modèles discrets et hybrides

Les résultats montrent que le modèle hybride est légèrement plus difficile à résoudre. La difficulté du problème semble exploser sur le problème à neuf sites, mais c'est surtout dans la portion purement continue de l'arbre de recherche que se situe la difficulté : seuls 0.1% des

backtracks sont effectués dans la portion hybride de l'arbre. De plus, le modèle se heurte aux mêmes difficultés que dans le cas purement discret, *i.e.* que l'on n'arrive pas d'avantage à résoudre le scénario à dix sites.

3.4.5.3 Analyse des résultats

Les valeurs de l'optimum obtenues dans les instance de RLFAP et de LocRLFAP sont ici identiques dans les modèles discrets et hybrides. Elles sont données dans le tableau suivant :

Sites	5	6	7	8	9	10
Valeur de l'optimum du RLFAP	89	89	97	221	221	285
Valeur de l'optimum du LocRLFAP	89	89	89	89	89	-

La comparaison de ces chiffres nous montre qu'en ayant une latitude de déploiement des sites, les optima atteints sont effectivement bien meilleurs que ceux permis par le RLFAP associé. On atteint sur tous ces exemples l'optimum que l'on aurait sans les contraintes de compatibilité distante. Dans le cas du déploiement de neuf sites on effectue un gain de 57% en fréquences.

De plus, la résolution du LocRLFAP se fait sur les cinq premiers exemples plus rapidement que celle du RLFAP dans le même modèle. Cependant une exception se présente sur le dernier exemple, où l'on n'arrive pas à calculer en une heure de temps CPU ne serait-ce que la première solution.

3.4.5.4 Analyse de la combinatoire de ces exemples

Étant donné l'explosion des temps de calculs sur le dernier exemple, il est intéressant d'analyser la combinatoire des modèles en contraintes pour tenter de comprendre cette difficulté. Les chiffres sont donnés pour le modèle discret dans le tableau suivant :

Problème	L_5	L_6	L_7	L_8	L_9	L_{10}
Variables	22	28	34	42	48	60
Contraintes d'écart duplexe	6	8	10	13	15	20
Contraintes d'interférence co-site	22	34	44	66	82	136
Contraintes d'interférence distante	212	402	670	1152	1568	2766
Contraintes de distance	36	53	73	97	123	155
Nombre de disjonctions	20	30	42	56	72	90

On a $n(n-1)$ disjonctions dans le modèle, étant donné que toute la clique des sites est considérée, et que pour chaque couple, les contraintes de compatibilité en portée et à mi-portée sont posées.

Bien que le dernier problème ait une combinatoire plus élevée que dans les précédents, il est étonnant que nous n'arrivions à trouver ne serait-ce même qu'une solution réalisable.

3.4.6 Bilan

Dans cette section nous avons pu estimer l'efficacité d'une première résolution du RLFAP et du LocRLFAP. La difficulté du RLFAP posé dans le modèle "disjonctif" croît plus vite que celle du RLFAP classique. Nous avons vu que de relâcher les positions de déploiement des sites permet

d'économiser jusqu'à 57% de la bande lors de l'allocation de fréquences. De plus, la résolution du LocRLFAP est plus aisée que celle du RLFAP sur les instances de 5 à 9 sites, mais ceci n'est pas vrai sur l'exemple le plus grand que nous n'arrivons alors étonnamment plus à résoudre. Nous poursuivons pour cela dans la section suivante en raffinant l'algorithme de recherche.

3.5 Améliorations de l'algorithme de recherche

Nous rapportons ici les pistes d'améliorations que nous avons explorées pour accroître la performance de l'algorithme de recherche.

Nous montrons d'abord pourquoi il n'est pas avantageux de brancher sur les disjonctions hybrides, puis nous proposons des heuristiques avancées de choix de variables et de valeurs qui permettent d'accélérer fortement la résolution.

3.5.1 Branchement sur les disjonctions dans l'algorithme de recherche

Puisque l'objectif est de minimiser l'utilisation de la plage de fréquences, on voudrait privilégier, lors de la construction de l'arbre de recherche, les configurations minimisant le nombre de contraintes posées portant sur des variables de fréquence. À cette fin, il semble intéressant de commencer la recherche des solutions par un branchement sur les disjonctions de compatibilité distante en supposant d'abord la branche portant sur la contrainte de distance dans les disjonctions `Compat_h` comme vraie et en continuant de même sur les disjonctions `Compat_l`. En poursuivant ensuite ce branchement sur les positions, on peut déterminer au plus tôt si les choix faits sur les disjonctions sont satisfiables.

Le fait que l'on ne dispose pas de bonnes bornes inférieures pour la fonction objectif afin d'éliminer les branches trop contraintes constitue une limitation à cette approche. Si l'on obtient assez rapidement de bons optima au problème par ce type de recherche, la réfutation des branches imposant les contraintes de compatibilité distante s'est montrée trop coûteuse. Par ailleurs, sur un problème comportant n sites, l'arbre des disjonctions est de profondeur $\mathcal{O}(n^2)$ et comporte donc $\mathcal{O}(3^{n^2})$ feuilles. Ceci explique, que même si une partie des noeuds est éliminée lors de la propagation, le nombre de fois où l'on doit réfuter des feuille de l'arbre formé par les disjonctions reste trop important au vu du coût de ce calcul, si le problème n'est pas suffisamment contraint par les distances.

3.5.2 Heuristiques de choix de variables

Pour des variables discrètes, la manière la plus simple d'implémenter le principe de *first-fail* (rappelons qu'il vise à obtenir les contradictions éventuelles sur un nœud situé le plus haut possible dans l'arbre de recherche) consiste à choisir comme variable suivante celle ayant le plus petit domaine. Étant donné que les domaines fréquentiels de toutes les antennes sont identiques dans notre problème, ce critère n'est pas suffisant parce-qu'il entraîne un choix arbitraire pour la première variable ainsi que pour tous les autres nœuds où plusieurs variables ont la même taille minimale de domaine. Cette faiblesse a pour suite que les temps d'exécution avec `mindomain` sont bien souvent pires que ceux obtenus en utilisant une énumération chronologique des variables.

Bessière et Régin présentent dans [21] des arguments qui suggèrent de privilégier l'heuristique `mindomain/maxdegree` à celle de `mindomain` pour effectuer le choix de variables, mais dans nos expérimentations cette heuristique ne s'est pas montrée intéressante sur nos exemples. Au lieu de cela, nous avons étudié la pertinence de choisir comme variable celle attachée au plus grand nombre de contraintes actives. Si ce choix à lui seul n'est pas avantageux, sa combinaison avec `mindomain` peut donner de bons résultats ; précisément, celui du plus petit domaine le plus contraint s'est montré être le meilleur sur nos exemples.

3.5.3 Heuristique de choix de valeurs

Étant donné que le modèle est le plus souvent sous-contraint sur les positions, on peut améliorer légèrement l'efficacité de l'algorithme de recherche dans le sous-arbre des positions par une heuristique proposée par Gelle pour des problèmes de configuration [46]. Cette heuristique de choix de valeurs – nous l'appelons `midvalue` – apporte un raffinement au principe de recherche dichotomique utilisé précédemment. En plus de découper en deux le domaine d'une variable en chaque point de choix, `midvalue` commence par explorer la valeur située au milieu du domaine. En orientant la recherche vers un point au centre de la boîte, on espère choisir la région dans laquelle il est le plus probable de trouver une solution. De plus, ceci permet d'éviter d'effectuer un grand nombre de bissections avant d'atteindre un point solution lorsque toute une région de l'espace est composée de solutions semblables.

3.5.4 Résultats numériques

Nous présentons sur le tableau 3.3 les résultats que l'on obtient grâce à ces améliorations de l'algorithme de recherche :

`heur(T)=mindomain-maxconstraints, heur(P)=maxdomain` `val-heur(P`$_{\texttt{continu}}$`)=midvalue` `val-heur(P`$_{\texttt{discret}}$`)=dicho`				
	modèle RLFAP		modèle LocRLFAP	
Instance	Temps$_{ms}$	Backtracks	Temps$_{ms}$	Backtracks
L_5	10	99	20	65
L_6	10	111	30	77
L_7	20	113	50	87
L_8	70	173	150	164
L_9	80	148	250	168
L_{10}	-	-	-	-

Tableau 3.3 – Résolution du LocRLFAP dans les modèles discrets et hybrides

Grâce à l'heuristique de choix de variables notamment, nous arrivons à obtenir des temps de résolution bien meilleurs et un nombre de backtracks réduits pour résoudre les cinq premiers exemples. Le modèle hybride est celui qui profite le plus des raffinements de l'algorithme de recherche, et ses performances sont maintenant proches de celles du modèle discret. En particulier, l'heuristique permet de surmonter les difficultés qu'il rencontrait sur L_7 et L_9.

Pour autant, ces heuristiques ne règlent l'impossibilité de trouver une solution réalisable sur L_{10}. Il nous faut pour cela modifier le modèle du LocRLFAP, ce que nous abordons maintenant.

3.6 Modélisation à l'aide de *distn* et *distn_var*

Nous proposons ici deux améliorations à chacun de nos modèles simples en contraintes discrètes et hybrides dans le but d'améliorer les performances de sa résolution.

L'écriture du problème global de déploiement par un ensemble déconnecté de contraintes élémentaires de distance euclidienne ne parvient pas à prendre en compte la sémantique des domaines. En conséquence, le filtrage opéré sur les contraintes donnant la position des sites est trop faible pour permettre une résolution efficace du modèle précédent. Nous proposons de répondre à ce problème en mettant en œuvre la contrainte globale développée au Chapitre 2 tout en conservent l'équivalence entre le modèle mathématique et le modèle en contraintes.

Nous introduisons d'abord la contrainte *distn* pour modéliser les contraintes de portée et d'interdistance minimale puis nous améliorons encore l'intégration de la contrainte globale dans la spécificité disjonctive du modèle. La contrainte *distn_var* permet de relier les contraintes d'interdistance avec celles de compatibilité distante.

3.6.1 Maintien des contraintes de distance min et max par la contrainte globale *distn*

Dans nos premiers modèles en contraintes naïfs, l'ensemble des contraintes de portée et la clique des contraintes de distance minimale étaient propagées de façon indépendante. Ces contraintes peuvent être regroupées dans une contrainte globale et cette encapsulation permet *a fortiori* une propagation plus forte.

Pour traiter le modèle hybride, on dispose de la contrainte continue *distn* définie au chapitre 2 comme :

$$\begin{aligned}\texttt{distn}(\texttt{list}[(\texttt{x}_1 : \texttt{RVar}\ ,\ \texttt{y}_1 : \texttt{RVar}), \ldots, (\texttt{x}_\texttt{n} : \texttt{RVar}\ ,\ \texttt{y}_\texttt{n} : \texttt{RVar})] \\ \texttt{m} : \texttt{matrix}[\texttt{double}], \\ \texttt{M} : \texttt{matrix}[\texttt{double}]),\end{aligned}$$

$$\text{vérifiée ssi } \begin{cases}\forall i \neq j, & dist((x_i, y_i), (x_j, y_j)) \geq\ m_{i,j} \\ \forall i \neq j, & dist((x_i, y_i), (x_j, y_j)) \leq\ M_{i,j}\end{cases}$$

En revanche, pour utiliser une modélisation discrète, on doit se restreindre à des variables $((x_1, y_1), \ldots, (x_n, y_n))$ entières. On utilise pour cela une modification dans l'algorithme de propagation de *distn* qui opère son filtrage dans le continu, tout en réduisant les domaines réels à des domaines entiers quand cela est avantageux, et qui à la sortie de l'algorithme assure l'intégralité des solutions. On utilisera donc la contrainte modifiée comme suit :

$$\texttt{distn}(\texttt{list}[\texttt{S}_1 : \texttt{Site}, \ldots, \texttt{S}_\texttt{n} : \texttt{Site}]\ ,\ \texttt{m} : \texttt{matrix}[\texttt{integer}]\ ,\ \texttt{M} : \texttt{matrix}[\texttt{integer}]),$$

$$\text{vérifiée ssi } \begin{cases}\forall i \neq j, & dist(S_i, S_j) \geq\ m_{i,j} \\ \forall l_{ij}, & dist(S_i, S_j) \leq\ M_{i,j}\end{cases}$$

où les domaine des sites sont définis par des variables entières. Il est par ailleurs avantageux d'ajouter au système de contraintes – de façon redondante – l'expression de *distn* par des contraintes élémentaires de distance, parce que cela permet d'accélérer la propagation du modèle.

3.6.2 Maintien des contraintes de distance min et max par la contrainte globale *distn_var*

Une faiblesse du modèle précédent est que les contraintes de distance apparaissent encore de façon décorrélée malgré l'introduction de la contrainte *distn* dans le modèle. On a en effet :

$$\forall i \neq j \begin{cases} (\texttt{or}(\texttt{dist?}(\texttt{S}_\texttt{i}, \texttt{S}_\texttt{j}, \geq, \texttt{d}_\texttt{h}), \texttt{Compat\_freq?}(\texttt{S}_\texttt{i}, \texttt{S}_\texttt{j}, \texttt{i}, \texttt{j}, \Delta_\texttt{h}, \texttt{T}))) \\ (\texttt{or}(\texttt{dist?}(\texttt{S}_\texttt{i}, \texttt{S}_\texttt{j}, \geq, \texttt{d}_\texttt{l}), \texttt{Compat\_freq?}(\texttt{S}_\texttt{i}, \texttt{S}_\texttt{j}, \texttt{i}, \texttt{j}, \Delta_\texttt{l}, \texttt{T}))) \\ \texttt{distn}(\texttt{list}[\texttt{S}_1, \ldots, \texttt{S}_\texttt{n}]\ ,\ \texttt{m}\ ,\ \texttt{M}) \end{cases}$$

où les deux premières contraintes posent des disjonctions entre positions et fréquences. La propagation des contraintes de distance impliquées dans ces disjonctions se fait de façon indépendante, et ne se répercute par conséquent que indirectement dans la contrainte globale *distn*.

Une possibilité pour remédier à cette insuffisance consiste à introduire des variables supplémentaires et à tirer parti d'une modélisation plus avancée avec *distn_var*. Introduisons pour chaque couple (i, j) une variable `vdist_ij` exprimant la distance entre S_i et S_j :

$$\forall i \neq j \quad \texttt{vdist\_ij}[\texttt{i}, \texttt{j}] : \texttt{Var}, \quad \texttt{Domain}(\texttt{vdist\_ij}[\texttt{i}, \texttt{j}]) = [\ 0, +\infty\ [$$

On peut alors utiliser la contrainte *distn_var* :

$$\texttt{distn\_var}(\texttt{list}[\texttt{S}_1 : \texttt{Site}, \ldots, \texttt{S}_\texttt{n} : \texttt{Site}]\ ,\ \texttt{vdist} : \texttt{matrix}[\texttt{Var}])$$
$$\text{vérifiée ssi } \forall i \neq j, \quad dist(S_i, S_j) \ = \ vdist[i, j]$$

En effet, les contraintes de distance apparaissant dans les disjonctions hybrides peuvent être connectées à la contrainte globale par l'intermédiaire des matrices de variables de distance en posant :

$$\begin{aligned} &\forall \texttt{i} \neq \texttt{j}, \quad \texttt{Domain}(\texttt{vdist\_ij}[\texttt{i}, \texttt{j}]) = [\ \texttt{m}_{\texttt{i,j}}, \texttt{M}_{\texttt{i,j}}\ [\\ &\forall \texttt{i} = \texttt{j}, \quad \texttt{Domain}(\texttt{vdist\_ij}[\texttt{i}, \texttt{j}]) = [\ 0, +\infty\ [\\ &\forall \texttt{i} \neq \texttt{j}, \quad \texttt{vdist}[\texttt{i}, \texttt{j}] = \texttt{vdist\_ij}[\texttt{i}, \texttt{j}] \end{aligned}$$

Cela permet d'avoir une communication bidirectionnelle entre les contraintes de distance des disjonctions et les contraintes de portée et de distance minimale :

- Si une contrainte `Compat_freq` est violée, la nouvelle borne inférieure de distance minimale se répercute immédiatement sur l'ensemble des contraintes de distance de la contrainte *distn*.
- Si l'ensemble des contraintes de distances considérées globalement dans la contrainte *distn* ne permettent pas l'assignation de la branche gauche d'une disjonction, alors ceci se répercute immédiatement en forçant la contrainte `Compat_freq` correspondante.

On peut donc réécrire l'ensemble des contraintes de distance de façon homogène dans le problème :

$$\forall i \neq j \begin{cases} \texttt{or}((\texttt{vdist\_ij}[\texttt{i},\texttt{j}] \geq ?\texttt{d}_\texttt{h}), \texttt{Compat\_freq}?(\texttt{S}_\texttt{i}, \texttt{S}_\texttt{j}, \texttt{i}, \texttt{j}, \Delta_\texttt{h}, \texttt{T})) \\ \texttt{or}((\texttt{vdist\_ij}[\texttt{i},\texttt{j}] \geq ?\texttt{d}_1), \texttt{Compat\_freq}?(\texttt{S}_\texttt{i}, \texttt{S}_\texttt{j}, \texttt{i}, \texttt{j}, \Delta_1, \texttt{T})) \\ \texttt{distn\_var}(\texttt{list}[\texttt{S}_1, \ldots, \texttt{S}_\texttt{n}]\ ,\ \texttt{vdist\_ij}[\texttt{i},\texttt{j}]) \end{cases}$$

Ici encore, nous posons de manière redondante l'expression de *distn_var* par des contraintes élémentaires de distance.

3.7 Résolution du problème de déploiement progressif d'antennes

Nous rapportons ici les résultats numériques associés à la résolution des exemples de test du problème de déploiement progressif d'antennes en utilisant les voies d'améliorations que nous avons proposées aux deux dernières sections concernant l'algorithme de recherche et la modélisation de l'application avec la contrainte globale *distn_var*.

Dans les instances de test précédentes, nous avons étudié les cas extrêmes du RLFAP et du LocRLFAP, où les positions des sites sont toutes respectivement fixées ou relâchées. Nous envisageons ici des cas où seule une partie des positions des sites peuvent effectivement être relâchées. Pour une étude exhaustive, il faudrait considérer tous les 2^n scénarios de localisation possibles. Cependant la durée des calculs nous conduit à ne nous intéresser qu'à une transition progressive donnée du RLFAP jusqu'au LocRLFAP. Nos tests partent d'une instance de résolution du RLFAP et relâchent progressivement l'ensemble de tous les sites dans l'ordre chronologique : on relâche S_0, puis S_0 et S_1, etc. jusqu'au scénario de LocRLFAP. Par exemple, pour un déploiement à 9 sites, nous nommons ces scénarios $D_9(\emptyset) = R_9$, $D_9(0)$, ..., $D_9(0..8) = L_9$. Une illustration des exemples de tests traités est donnée sur les Figures 3.4 et 3.5.

Nous abordons d'abord la résolution du problème de déploiement progressif d'antennes dans les modèles discrets et hybrides simples mais avec l'algorithme de recherche avancé puis nous montrons comment la combinaison de ce dernier avec le modèle en contrainte élaboré permet de résoudre nos jeux de tests. Enfin nous donnons une analyse qualitative des résultats de l'allocation de fréquence et concluons.

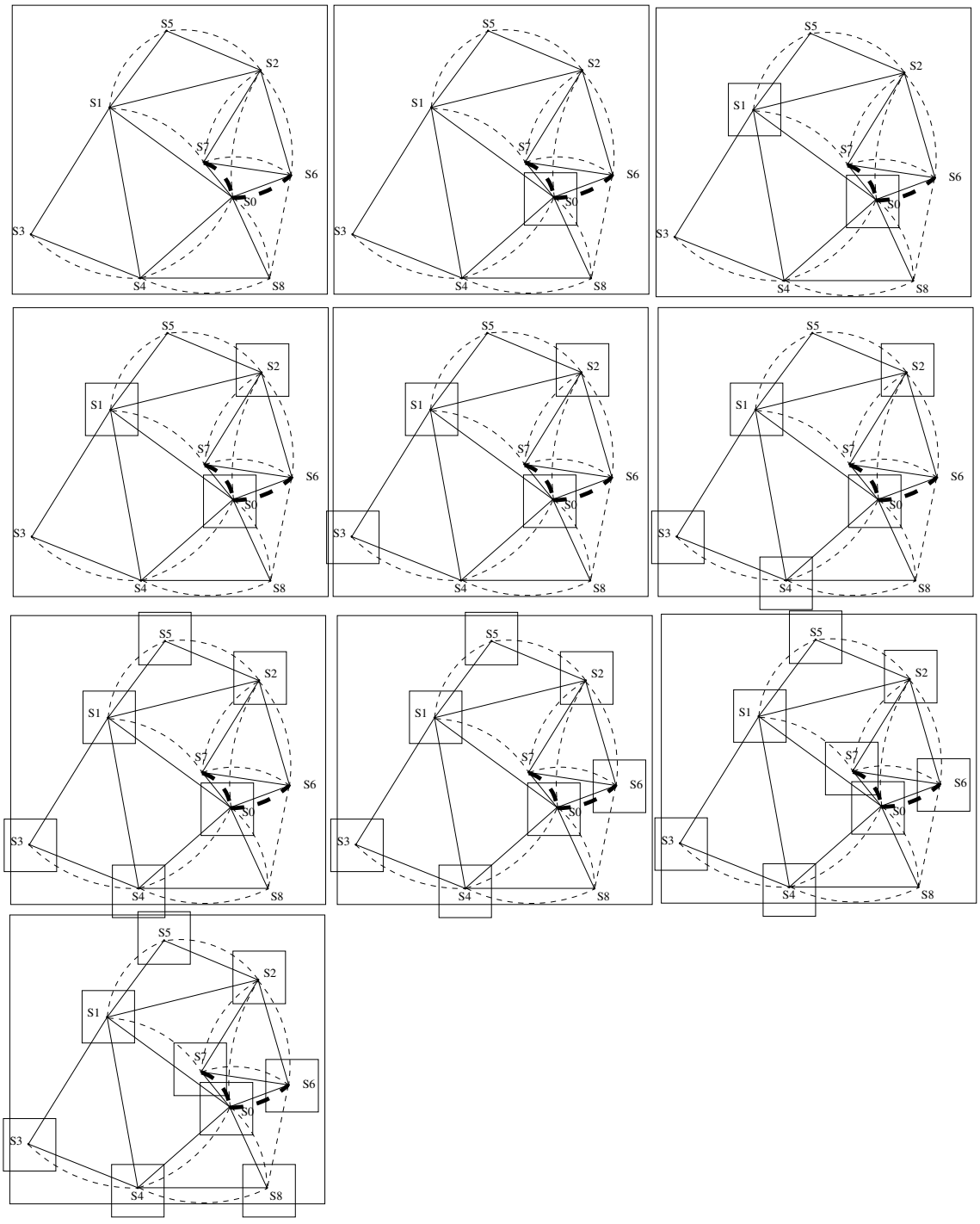


Figure 3.4 – Déploiement progressif de R_9 à L_9.

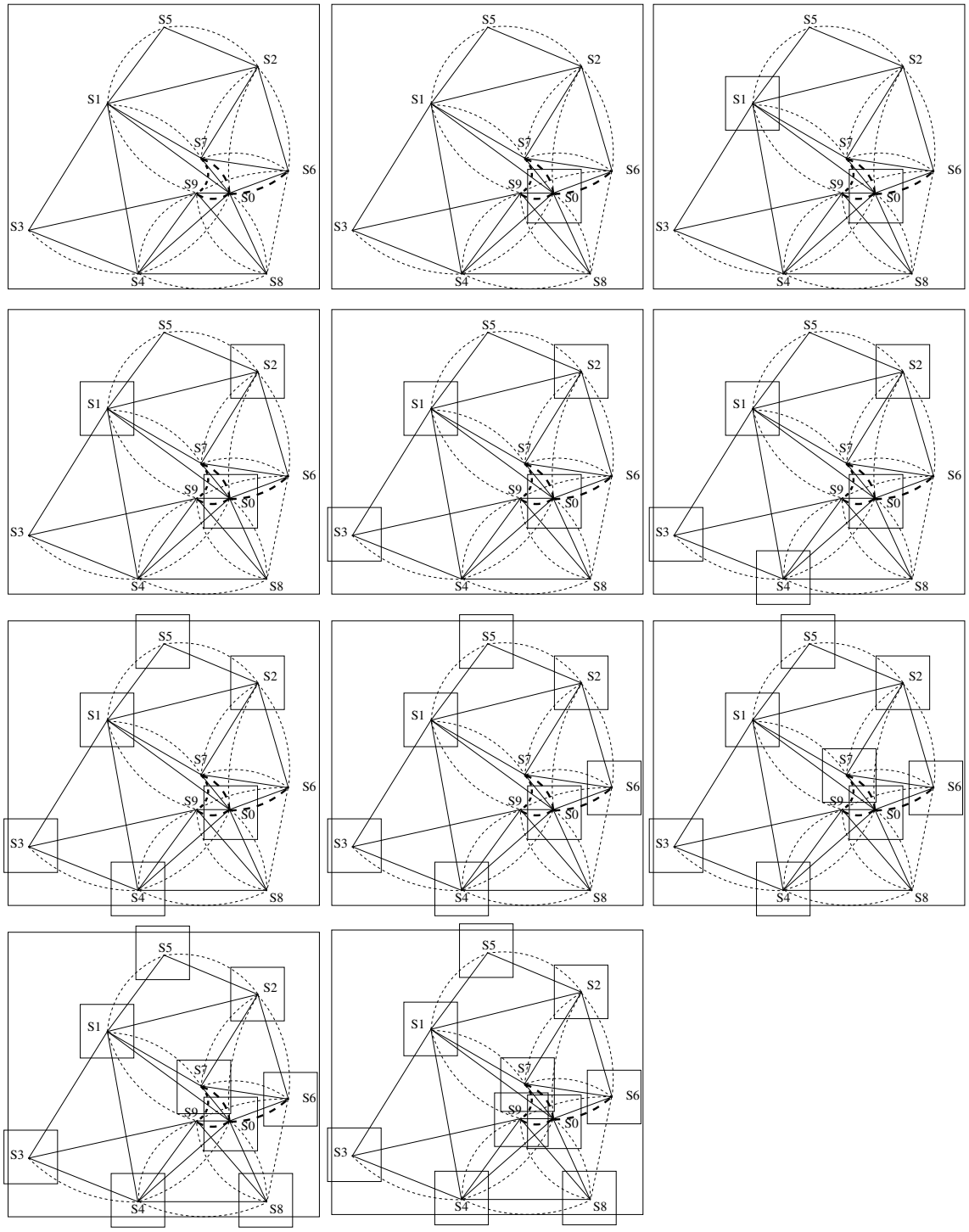


Figure 3.5 – Déploiement progressif de R_{10} à L_{10}.

3.7.1 Résultats numériques sur les modèles simples

Nous présentons ici les résultats que l'on obtient pour une résolution sans la contrainte globale :

heur(T)=mindomain-maxconstraints, heur(P)=maxdomain val-heur($P_{continu}$)=midvalue val-heur($P_{discret}$)=dicho				
	modèle discret		modèle hybride	
Instance	$Temps_{ms}$	Backtracks	$Temps_{ms}$	Backtracks
$D_9(\emptyset) = R_9$	3120	9192	5430	9196
$D_9(0)$	830	3270	1890	3265
$D_9(0..1)$	930	3319	1990	3306
$D_9(0..2)$	1410	6754	3880	6725
$D_9(0..3)$	1420	6768	3870	6733
$D_9(0..4)$	430	655	760	644
$D_9(0..5)$	440	646	750	634
$D_9(0..6)$	140	370	360	422
$D_9(0..7)$	70	140	285050	75737
$D_9(0..8) = L_9$	90	148	240	168

heur(T)=mindomain-maxconstraints, heur(P)=maxdomain val-heur($P_{continu}$)=midvalue val-heur($P_{discret}$)=dicho				
	modèle discret		modèle hybride	
Instance	$Temps_{ms}$	Backtracks	$Temps_{ms}$	Backtracks
$D_{10}(\emptyset) = R_{10}$	86.540	156.487	136.350	156.489
$D_{10}(0)$	2.894.490	8.422.085	7.034.800	8.397.394
$D_{10}(0..1)$	465.510	641.425	893.360	585.919
$D_{10}(0..2)$	210.940	253.606	358.930	253.204
$D_{10}(0..3)$	143.240	162.252	238.950	161.624
$D_{10}(0..4)$	11.720	16.146	17.110	12.639
$D_{10}(0..5)$	11.670	16.040	17.630	12.756
$D_{10}(0..6)$	3.330	8.529	8.590	9.058
$D_{10}(0..7)$	-	-	4.420	4.324
$D_{10}(0..8)$	-	-	-	-
$D_{10}(0..9) = L_{10}$	-	-	-	-

De façon remarquable la difficulté à résoudre le problème de déploiement progressif d'antennes varie fortement et de manière non monotone selon le nombre de sites relâchés. Sur le déploiement progressif de 9 sites, on constate des écarts allant jusqu'à presque trois ordres de grandeur entre les exemples les plus faciles et les plus difficiles.

Exception faite de $D_9(0..7)$, la difficulté est généralement analogue entre les modèles discrets et hybrides. Pour ce cas particulier, presque tous les backtracks (précisément 75146 sur 75737)

ont lieu dans la portion continue de l'arbre de recherche ; ceci suggère que la consistance locale assurée par HC4 sur les contraintes de distance continues n'est pas suffisante pour résoudre efficacement cette instance.

Dans six instances sur dix, le modèle hybride atteint l'optimum avec quelques backtracks de moins, mais l'écart n'est pas suffisant pour rendre le calcul plus rapide. Cette diminution du nombre de backtracks peut s'expliquer par le fait que dans le cas discret on approche parfois d'une solution réelle au problème, mais qu'on est amené à la rejeter du fait de la restriction à des solutions entières. À nombre de backtracks équivalents, la résolution hybride est entre une et trois fois plus lente. Si l'on exclut $D_9(0..7)$ et l'instance où tous les sites sont fixés[2], les temps de résolution sont en moyenne 2.4 fois plus lents.

Rappelons que l'une des insuffisances majeures du modèle et du mode de résolution précédents était qu'ils ne réussissaient pas le passage à l'échelle d'un déploiement à 10 sites. On constate ici que l'on arrive pour autant à résoudre les premières instances intermédiaires lors du déploiement progressif de dix sites. Sur les instances résolues par les deux modèles, le nombre de backtracks est sensiblement équivalent, mais la résolution est en moyenne 2.3 fois plus lente dans le modèle hybride. Dans ces modèles sans contraintes globales, le modèle hybride réussit cependant à résoudre une instance de plus que le modèle discret.

3.7.2 Résultats numériques avec *distn_var*

Nous présentons dans cette section les détails de la résolution du problème de déploiement d'antennes sur les instances à 9 et 10 sites. Nous analysons ici les temps de calcul obtenus.

Les temps et le nombre de backtracks sont montrés sur les Tableaux 3.4 et 3.5 pour chacune des heuristiques de recherche utilisées.

Sur l'exemple à 9 sites avec des heuristiques élémentaires, le modèle en contraintes hybrides requiert en moyenne 1.5 fois plus de backtracks et il est de ce fait 1.8 fois plus lent. Cependant les temps de résolution restent faibles dans les deux cas (toutes les calculs se font en quelques secondes) et l'utilisation de l'heuristique avancée `mindomain-maxconstraints` lisse ces différences et les performances sont alors très similaires.

En comparant maintenant ces derniers résultats avec ceux auxquels on parvenait avec des contraintes élémentaires (*cf.* 103), on voit que *distn_var* permet de diviser le nombre de backtracks par deux sur une moyenne de neuf instances, tant dans les modèles discrets que hybrides, mais le coût d'appel à la contrainte globale fait que l'on est 2 (resp. 1.2) fois plus lent sur le modèle discret (resp. hybride). On n'obtient plus l'irrégularité qu'on avait sur $D_9(0..7)$ et on accélère ainsi de deux ordres de grandeur sur cette instance si bien que sur la totalité des instances hybrides on divise le nombre de backtracks par 8 et on gagne un facteur temps de 20.

Sur les exemples à 10 sites que les deux modèles arrivent à résoudre avec des heuristiques élémentaires, le modèle en contraintes hybrides est un peu moins performant sur $D_{10}(0..3)$, par contre il requiert moins de backtracks sur l'ensemble des autres cas que l'on sait traiter, et il parvient à résoudre quatre instances supplémentaires.

En utilisant maintenant l'algorithme de recherche avancé, quand nous comparons les sept instances de déploiement à 10 sites que nous arrivons à résoudre dans les deux modèles, le mo-

[2]Pour R_9, le modèle de LocRLFAP n'a pas vraiment de justification et l'on pourrait tout aussi bien choisir le modèle sans disjonctions.

heur(T)=mindomain, heur(P)=maxdomain				
	modèle discret		modèle hybride	
Instance	Temps$_{ms}$	Backtracks	Temps$_{ms}$	Backtracks
$D_9(\emptyset) = R_9$	2.130	6.724	3.930	6728
$D_9(0)$	22.430	99.715	52.610	99.812
$D_9(0..1)$	40.070	103.149	128.290	176.029
$D_9(0..2)$	258.100	374.547	686.660	623.382
$D_9(0..3)$	22.240	27.074	62.500	52.013
$D_9(0..4)$	3.040	1.283	2.900	1.021
$D_9(0..5)$	3.640	1.273	3.110	892
$D_9(0..6)$	450	112	450	251
$D_9(0..7)$	1.490	347	1.530	532
$D_9(0..8) = L_9$	176.420	14.597	1.710	456

heur(T)=mindomain-maxconstraints, heur(P)=maxdomain val-heur(P$_{continu}$)=midvalue val-heur(P$_{discret}$)=dicho				
	modèle discret		modèle hybride	
Instance	Temps$_{ms}$	Backtracks	Temps$_{ms}$	Backtracks
$D_9(\emptyset) = R_9$	3120	9192	5540	9196
$D_9(0)$	1570	3284	2150	3278
$D_9(0..1)$	1580	3403	1910	2744
$D_9(0..2)$	1280	1851	2040	2096
$D_9(0..3)$	1300	1763	2100	2022
$D_9(0..4)$	2140	716	2180	698
$D_9(0..5)$	2690	707	2550	702
$D_9(0..6)$	780	266	530	389
$D_9(0..7)$	410	117	640	215
$D_9(0..8) = L_9$	690	117	830	237

Tableau 3.4 – Résultats avec 9 sites dans les modèles discrets et hybrides utilisant *distn_var*. Nous utilisons des heuristiques élémentaires en haut et des heuristiques avancées en bas.

heur(T)=mindomain, heur(P)=maxdomain				
	modèle discret		modèle hybride	
Instance	Temps$_{ms}$	Backtracks	Temps$_{ms}$	Backtracks
$D_{10}(\emptyset) = R_{10}$	92.880	156497	146.960	156.500
$D_{10}(0)$	-	-	2.772.640	2.000.610
$D_{10}(0..1)$	-	-	-	-
$D_{10}(0..2)$	-	-	-	-
$D_{10}(0..3)$	253.410	178.464	460.560	233.897
$D_{10}(0..4)$	30.060	15.420	33.500	15.107
$D_{10}(0..5)$	32.060	15.437	34.890	15.109
$D_{10}(0..6)$	4.560	2.013	3.900	2.011
$D_{10}(0..7)$	-	-	7.430	3.915
$D_{10}(0..8)$	-	-	39.540	6.238
$D_{10}(0..9) = L_{10}$	-	-	67.130	9.413

heur(T)=mindomain-maxconstraints, heur(P)=maxdomain val-heur($P_{continu}$)=midvalue val-heur($P_{discret}$)=dicho				
	modèle discret		modèle hybride	
Instance	Temps$_{ms}$	Backtracks	Temps$_{ms}$	Backtracks
$D_{10}(\emptyset) = R_{10}$	85.620	156.487	138.370	156.489
$D_{10}(0)$	351.020	852.318	150.170	87.144
$D_{10}(0..1)$	124.620	115.307	123.340	73.451
$D_{10}(0..2)$	239.700	147.809	325.530	163.440
$D_{10}(0..3)$	237.950	131.582	337.970	147.716
$D_{10}(0..4)$	28.310	16.830	28.420	13.758
$D_{10}(0..5)$	29.960	16.698	29.660	13.760
$D_{10}(0..6)$	11.360	8.052	3.210	1.931
$D_{10}(0..7)$	-	-	6.310	4.264
$D_{10}(0..8)$	-	-	218.220	100.298
$D_{10}(0..9) = L_{10}$	-	-	369.300	140.592

Tableau 3.5 – Résultats avec 10 sites dans les modèles discrets et hybrides utilisant *distn_var* Nous utilisons des heuristiques élémentaires en haut et des heuristiques avancées en bas.

dèle hybride nécessite 63% de backtracks de moins que le modèle discret mais ne se résout que 3% plus vite. Par contre, sur trois autres instances, le modèle hybride permet de résoudre le problème en moins de dix minutes, alors que le modèle discret n'arrive pas même à trouver de solution réalisable en plusieurs heures. En comparant à nouveau ces derniers résultats avec ceux auxquels on parvenait avec des contraintes élémentaires (*cf.* 103), on voit que *distn_var* permet de diviser le nombre de backtracks par 18.8 (resp. 7.4) dans le cas hybride (resp. discret) et de diviser les temps de calcul par 8.6 (resp. 3.7) sur une moyenne de sept instances.

Globalement, sur l'ensemble de tous les *benchmarks* abordés avec *distn_var*, l'heuristique de `mindomain-maxconstraints` obtient de meilleurs résultats dans 28 exemples sur 39, et l'hybridation diminue le nombre de backtracks nécessaires dans 22 cas sur 36.

En conclusion, l'hybridation à elle seule ne suffit pas, puisque les instances $D_{10}(0..1)$ et $D_{10}(0..2)$ ne peuvent pas être résolues avec des heuristiques élémentaires. C'est l'association du modèle hybride utilisant *distn_var* avec l'heuristique de `mindomain-maxconstraints` qui est essentielle pour résoudre l'ensemble de nos exemples.

3.7.3 Analyse qualitative des résultats

Nous analysons ici le gain en fréquences que permet le LocRLFAP comparé au RLFAP. Les données relatives à la qualité des solutions obtenues par chacune des résolutions sont regroupées sur le tableau 3.6.

	modèle discret		modèle hybride	
Instance	Optimum	Gain	Optimum	Gain
$D_9(\emptyset)=R_9$	221	0	=	=
$D_9(0)$	153	36%	=	=
$D_9(0..1)$	153	36%	=	=
$D_9(0..2)$	153	36%	=	=
$D_9(0..3)$	153	36%	=	=
$D_9(0..4)$	97	39%	=	=
$D_9(0..5)$	97	39%	=	=
$D_9(0..6)$	89	42%	=	=
$D_9(0..7)$	89	42%	=	=
$D_9(0..8)=L_9$	89	57%	=	=

	modèle discret		modèle hybride	
Instance	Optimum	Gain	Optimum	Gain
R_{10}	285	0	=	=
$D_{10}(0)$	221	24%	161	39%
$D_{10}(0..1)$	161	39%	=	=
$D_{10}(0..2)$	161	39%	=	=
$D_{10}(0..3)$	161	39%	=	=
$D_{10}(0..4)$	157	41%	=	=
$D_{10}(0..5)$	157	41%	=	=
$D_{10}(0..6)$	153	43%	=	=
$D_{10}(0..7)$	-	-	153	43%
$D_{10}(0..8)$	-	-	97	58%
$D_{10}(0..9)=L_{10}$	-	-	89	63%

Tableau 3.6 – Gains de fréquences grâce au déploiement progressif dans chacun des modèles

D'un point de vue qualitatif, la conclusion la plus intéressante est que notre nouveau modèle a un double avantage.

- Le modèle hybride permet d'atteindre des solutions économisant des fréquences par rapport aux solutions discrètes : nous obtenons un résultat meilleur de 15% sur $D_{10}(0)$. On ne peut rien dire pour les trois dernières instances dont nous n'arrivons pas à calculer l'optimum dans un espace discret.
- Nous arrivons à résoudre l'ensemble de nos problèmes de test dans ce cadre alors qu'aucune solution n'est trouvée en plus d'une heure sur trois de nos instances résolues dans un espace discret.

3.8 Conclusion

Nous avons présenté un problème opérationnel de déploiement progressif d'antennes radio mobiles et étudié sa modélisation et sa résolution en vue de proposer un outil d'aide au déploiement aux opérateurs chargés de l'affectation des ressources sur le terrain.

Nous avons défini un modèle mathématique à partir de la description de ce problème. Ce modèle a été dérivé en deux modèle simples en contraintes, l'un entièrement discret, l'autre hybride discret continu. Ils permettent de résoudre les instances de test les plus petites, mais sont incapable de résoudre l'ensemble des instances de test plus conséquentes, même avec un algorithme de recherche élaboré.

Nous avons modifié ces modèle en y intégrant la contrainte globale *distn* ; plus particulièrement, nous avons vu comment la contrainte *distn_var* adresse particulièrement bien à la nature disjonctive de problème. L'association d'un espace de définition hybride discret-continu avec cette modélisation avancée et une nouvelle heuristique de choix de variables permet de résoudre la totalité des exemples de tests étudiés. Ces résultats ont montré l'intérêt que peut présenter l'hybridation des techniques de résolution de contraintes discrètes et d'intervalles pour améliorer les performances de la résolution de problèmes combinatoires complexes. De plus, nos tests ont mis en avant certains scénarios où la modélisation par des contraintes hybrides présente de plus un avantage qualitatif en ce qu'elle autorise d'atteindre de meilleurs optima que dans un modèle entièrement discrétisé.

Conclusion et Perspectives

Les travaux de cette thèse ont porté sur la recherche de méthodes pour modéliser finement et résoudre efficacement un problème opérationnel de déploiement d'antennes avec allocation de fréquences dans le cadre de la PPC. Nous avons étudié d'une part, la pertinence de modéliser cette application sous la forme d'un problème d'optimisation sous contraintes hybrides discrètes-continues pour augmenter l'expressivité de son modèle, d'autre part l'avantage que présente l'introduction de contraintes globales continues dans la modélisation pour une résolution efficace de ce problème.

Nous avons montré qu'à l'instar des contraintes globales discrètes, des techniques issues de la géométrie algorithmique peuvent être utilisées avantageusement pour concevoir des algorithmes de filtrage de contraintes globales continues.

En partant d'un algorithme de packing de cercles, nous avons épousé une interprétation géométrique de la sémantique des domaines de contraintes de distance euclidienne. De la sorte, en proposant une première procédure de filtrage pour *distn* qui construit à partir d'une étude de cas une relaxation linéaire de la contrainte, nous avons abouti à l'extension à l'arithmétique d'intervalles d'un algorithme de programmation linéaire de complexité linéaire qui pourrait être utile pour résoudre d'autres problèmes.

N'ayant pas obtenu de résultats concluants avec ce premier algorithme de filtrage, nous avons finalement opté pour une procédure de filtrage plus efficace, déterminée à partir d'un second algorithme de packing de cercles. Nos principales contributions sur ce dernier ont été de donner une caractérisation géométrique de ce filtrage par une sémantique de point fixe, de donner son extension à des hypothèses de position générale des domaines de départ et d'apporter des simplifications à son extension aux intervalles.

Nous avons tiré parti de la représentation polygonale des domaines dans cet algorithme pour proposer, à l'aide de propriétés de la somme de Minkowski de polygones convexes, un filtrage adapté à notre contrainte globale de distance variable *distn_var*.

Dans ces algorithmes, la nécessité de préserver la correction des calculs pour leur utilisation en PPC présente des difficultés lorsqu'il est nécessaire de construire de nouveaux objets géométriques (par exemple des points d'intersection). Si l'arithmétique rationnelle permet en principe d'éliminer ces problèmes, elle entraîne dans certains cas des pertes de performance telles que cette solution n'est plus viable lorsque les calculs sont effectués en cascade. Nous utilisons pour cela une combinaison de l'arithmétique rationnelle avec l'arithmétique d'intervalles, et améliorons ainsi la performance des algorithmes tout en respectant la correction des résultats.

Les travaux sur notre contrainte globales laissent quelques perspectives pour des recherches supplémentaires :

- Une amélioration assez immédiate de *distn* serait d'autoriser la spécification de ses domaines par un simplexe. Cette possibilité, quoique assez spécifique au cas du déploiement d'antennes radio, pourrait augmenter l'expressivité de la contrainte et l'efficacité de l'al-

gorithme de filtrage. En effet pour ce champ d'application, les domaines des points sont spécifiés par des polygones décrivant une approximation du terrain ; or la résolution de ces contraintes par un solveur extérieur – ne communiquant avec *distn* que par la modification des bornes – revient à pâtir du même problème de dépendance auquel répondait justement la vision globale de *distn*.

- Si nous avons pu proposer un algorithme pour les contraintes de distance maximale dans le cadre de notre premier algorithme de filtrage pour *distn*, la définition d'un algorithme de filtrage $MaxDistancePrune_{poly}$ tirant parti d'une représentation polygonale des domaines serait intéressante car elle permettrait un couplage fort avec l'algorithme présenté ici pour *distn* et *distn_var*.
- L'aperçu que nous avons donné de l'extension en trois dimensions de l'algorithme de filtrage $MinDistancePrune_{poly}$ permet d'envisager les étapes essentielles de son implémentation. Cette tâche semble toutefois assez ardue au vu des difficultés que présente l'implémentation en deux dimensions. Le passage en trois dimensions permettrait notamment d'appréhender l'applicabilité de la contrainte à des problèmes importants. À ce titre, ceux de conformation moléculaire auraient sans doute les retombées pratiques les plus importantes. Le passage à l'échelle des algorithmes sur ces problèmes est une question intéressante qui reste ouverte à l'issue de cette thèse.

L'hybridation de méthodes discrètes et continues a été abordée tant dans la relaxation continue de contraintes discrètes que dans la coopération de modèles discrets et continus dans un modèle mixte.

Nous utilisons le premier mode d'hybridation dans la version discrète de la contrainte *distn* : nous y opérons de manière naïve sur deux niveaux en discrétisant le résultat du filtrage sur la relaxation continue de la contrainte : cette approche répète l'algorithme continu jusqu'à ce que la contraction des intervalles réels calculés à des intervalles entiers laisse le filtrage inchangé.

Il serait intéressant de mettre en oeuvre des méthodes plus fines pour ramener le domaine des variables à leur sémantique discrète. L'algorithme incrémental proposé par Harvey [58] calcule en temps optimal le plus petit ensemble d'inéquations linéaires qui définissent l'enveloppe convexe des points à coordonnées entières compris dans un polygone réel. Malheureusement son auteur ne fournit pas de résultats expérimentaux pour estimer la performance de cet algorithme en pratique.

L'hybridation plus complète utilisée dans l'étude de l'application de déploiement d'antennes présente au moins trois avantages :

Le sous-problème continu étant moins contraint, il est possible de trouver de meilleurs optima au problème d'optimisation. Ce cas se présente sur l'un de nos exemples où l'hybridation permet un gain de fréquences de 15% par rapport à un modèle discret.

La prise en compte d'une grille de discrétisation plus fine permettrait sans doute de retrouver ces solutions disparues dans le modèle discret, mais il suppose de connaître à l'avance la dégradation de qualité des solutions inhérente au pas de discrétisation. L'espace de résolution continu a l'avantage de s'affranchir dans la modélisation d'un pas de discrétisation imposé artificiellement.

La restriction à des valeurs entières peut entraîner que l'algorithme de recherche explore des points très proches d'une solution continue et le rejet de cette solution retarde alors l'obtention de solutions. L'inverse peut naturellement aussi se produire, et nous avons observé un exemple où l'on effectuait une exploration intensive du sous-espace de recherche continu alors que la restriction à des solutions discrètes élaguait très rapidement la branche problématique de l'arbre

de recherche. L'utilisation de la contrainte globale opérant un filtrage fort sur les contraintes continues éliminait cependant ce problème, et la résolution d'un nombre supérieur d'instances hybrides suggère que, moyennant une modélisation efficace, l'hybridation de l'espace de recherche est préférable : en faisant abstraction des heuristiques utilisées pour résoudre les déploiements à 10 sites, 18 tests sur 20 sont résolus dans l'espace de recherche hybride alors qu'ils ne sont que 11 dans l'espace discret.

Bibliographie

[1] K. I. Aardal, C. P. M. van Hoesel, A. M. C. A. Koster, C. Mannino et A. Sassano. Models and solution techniques for the frequency assignment problem. *4OR*, 1(4):261–317, 2003.

[2] Abderrahmane Aggoun et Nicolas Beldiceanu. Extending chip in order to solve complex scheduling and placement problems. *Journal of Mathematical and Computer Modelling*, 17(7):57–73, 1993.

[3] Götz Alefeld et Jürgen Herzberger. *Introduction to Interval Computations*. Academic Press Inc., New York, USA, 1983.

[4] Krzystof R. Apt. The essence of constraint propagation. *CWI Quarterly*, 11(2 & 3):215–248, 1998.

[5] Krzysztof Apt et Peter Zoeteweij. A comparative study of arithmetic constraints on integer intervals. In K.R. Apt, F. Fages, F. Rossi, P. Szeredi et J. Vancza, réds., *Recent Advances in Constraints*, volume 3010 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 2004.

[6] R. Barták. Constraint programming: In pursuit of the holy grail. In *Proceedings of WD99*, Prague, June 1999.

[7] Heikel Batnini. Introduction of redundant constraint for solving systems of distance equations. *Journal of the University of Saarbrucken*, 2002. CALCULEMUS Autumn School in Pisa,

[8] Heikel Batnini. Contraintes globales pour la résolution de contraintes de distance. Master's thesis, Université de Nice Sophia-Antipolis, 2003.

[9] Heikel Batnini. *Contraintes Globales et Techniques de Résolution pour les CSPs Continus*. Thèse de Doctorat, Université de Nice Sophia-Antipolis, décembre 2005.

[10] Heikel Batnini et Michel Rueher. Semantic decomposition for solving distance constraints. In *Proc. of CP-03*, Kinsale, County Cork, Ireland, 2003.

[11] Nicolas Beldiceanu et Mats Carlsson. Sweep as a generic pruning technique applied to the non-overlapping rectangles constraint. In *Principles and Practice of Constraint Programming,7th International Conference*, pages 377–391, 2001.

[12] Nicolas Beldiceanu et Evelyne Contejean. Introducing global constraints in CHIP. *Mathematical and Computer Modelling*, 12:97–123, 1994.

[13] Nicolas Beldiceanu, Qi Guo et Sven Thiel. Non-overlapping constraints between convex polytopes. In *Proc. of CP 2001*, Lecture Notes in Computer Science, pages 392–407. Springer, 2001.

[14] Boaz Ben-Moshe, Matthew J.Katz et Michael Segal. Obnoxious facility location: Complete service with minimal harm. *International Journal on Computational Geometry and Applications*, 10(6):581–92, 2000.

[15] Frederic Benhamou. Heterogeneous constraint solving. In *Algebraic and Logic Programming, 5th International Conference*, 1996.

[16] Frederic Benhamou, Frédéric Goualard, Laurent Granvilliers et Jean-François Puget. Revising Hull and Box Consistency. In *Procs. ICLP'99*, 1999.

[17] Frederic Benhamou, D. McAllester et Pascal Van Hentenryck. Clp(intervals) revisited. In *Proc. of the Internationnal Symposium on Logic Programming*, pages 124–138, 1994.

[18] Frédéric Benhamou et William J. Older. Applying interval arithmetic to real, integer and boolean constraints. *Journal of Logic Programming*, 32(1):1–24, 1997.

[19] C. Bessière et M.O. Cordier. Arc-Consistency and Arc-Consistency Again. In *Proc. of AAAI-93*, pages 108–113, Washington, DC, 1993.

[20] C. Bessière, E. C. Freuder et J-C Régin. Using Inference to Reduce Arc Consistency Computation. In *Proc. of IJCAI-95*, volume 1, pages 592–598, Montréal, Canada, 1995.

[21] Christian Bessière et Jean-Charles Régin. Mac and combined heuristics: Two reasons to forsake fc (and cbj?) on hard problems. *Proc. of CP-96*, 1996.

[22] Christian Bessière et Jean-Charles Régin. Arc-consistency for general constraint networks: Preliminary results. In *Proc. of IJCAI-97*, pages 398,404, Nagoya, Japan, 1997.

[23] Christian Bessière et Jean-Charles Régin. Refining the basic constraint propagation algorithm. In *Proc. of IJCAI-01*, pages 309–315, Seattle, WA, USA, 2001.

[24] A.J. Booker. Case studies in design and analysis of computer experiments. In *Proc. of the Section on Physical and Engineering Science*, pages 224–248. American Statistical Association, 1996.

[25] Eric Bourreau. *Traitement de contraintes sur les graphes en programmation par contraintes.* Thèse de Doctorat, University Paris 13, 1999. cycle,

[26] Henri Béringer et Bruno De Backer. Combinatorial problem solving in constraint logic programming with cooperative solvers. In North Holland, réd., *Logic Programming: Formal Methods and Practical Applications*, volume 11 of *Studies in Computer Science and Artificial Intelligence*, 1995.

[27] B. Cabon, S. De Givry, L. Lobjois, T. Schiex et J. P. Warners. Benchmarks problems: Radio link frequency assignment. *Constraints*, 4:79–89, 1999.

[28] C.Gosselin, J.Sefrioui et M.J. Richard. Polynomial solutions to the direct kinematic problem of planar three-degree-of-freedom parallel manipulators. *Mechanism and Machine Theory*, 27:107–119, 1992.

[29] G. Chabert, G. Trombettoni et B. Neveu. New light on arc-consistency over continous domains. In *Proc. CP'04 First International Workshop on Constraint Propagation and Implementation*, Toronto, Canada, 2004.

[30] Chong-Kan Chiu et Jimmy Ho-Man Lee. Efficient interval linear equality solving in constraint logic programming. Rapport technique, Dep. of Computer Sciences, The Chinese University of Hong-Kong, 1996.

[31] Chiu Wo Choi, Warwick Harvey, Jimmy Ho-Man Lee et Peter J. Stuckey. Finite domain bounds consistency revisited. Published as `arXiv:cs.AI/0412021` disponible sur http://arxiv.org/, December 2004,

[32] C.Jermann, B.Neveu et G.Trombettoni. Inter-block backtracking: Exploiting the structure in continuous csps. In *2nd International Workshop on Global Constrained Optimization and Constraint Satisfaction COCOS'03*, Lausanne, Swiss, 2003.

[33] John G. Cleary. Logical Arithmetic. *Future Computing Systems*, 2(2):125–149, 1987.

[34] Hélène Collavizza, François Delobel et Michel Rueher. Comparing Partial Consistencies. *Reliable Computing*, 5(3):213–228, 1999.

[35] Hélène Collavizza, François Delobel et Michel Rueher. Extending consistent domains of numeric CSP. In *Procs. of the 16th IJCAI*, volume 1, pages 406–411, Stockholm, Sweden, juillet 1999.

[36] Alain Colmerauer. Prolog IV Specifications. Esprit project 5246, 1995,

[37] Bruno De Backer et Henri Beringer. Cooperative solvers and global constraints: the case of linear arithmetic constraints. In *Procs. of the ILPS'95 Post-Conference Workshop on Constraints, Databases and Logic Programming*, Portland, OR, USA, 1995.

[38] Mark de Berg, Marc van Kreveld, Mark Overmars et Otfried Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, 2nd édition, 1997.

[39] C. de Groot, M. Monagan, R. Peikert et D. Würtz. Packing circles in a square: A review and new results. In *System Modelling and Optimization*, volume 180 of *Lecture Notes in Control and Information Sciences*, pages 45–54. Springer Verlag, 1992.

[40] Artan Dimnaku, Rex Kincaid et Michael W. Trosset. Approximate solutions of continuous dispersion problems. preprint, 2002,

[41] Z. Drezner, réd. *Facility location. A survey of applications and methods*. Springer Verlag, 1995.

[42] Audrey Dupont, Eric Alvernhe et Michel Vasquez. Efficient filtering and tabu search on a consistent neighbourhood for the frequency assignment problem with polarisation. *Annals of Operations Research*, 130(1-4):179–198, 8 2004.

[43] M.E. Dyer. Linear algorithms for two and three- variable linear programs. *SIAM J. on Computing*, 13:31–45, 1984.

[44] Boi Faltings. Arc Consistency for Continuous Variables. *Artificial Intelligence*, 65(2):363–376, 1994.

[45] Filippo Focacci, Andrea Lodi et Michela Milano. Optimization-oriented global constraints. *Constraints*, 7(3/44), 2002.

[46] Esther Gelle et Boi Faltings. Solving mixed and conditional constraint satisfaction problems. *Constraints*, 8(2):107–141, 2003.

[47] F. Glover et M. Laguna. Tabu Search. In *Modern Heuristic Techniques for Combinatorial Problems*, pages 70–141. Blackwell Scientific Publishing, Oxford, 1993.

[48] Baumert L.D. Golomb S.W.. Backtrack programming. *Journal of the ACM, 12:516-524*, 1965.

[49] Frédéric Goualard. *Langages et environnements en programmation par contraintes d'intervalles*. Thèse de Doctorat, Université de Nantes, 2000.

[50] R.L. Graham et B.D. Lubachevsky. Repeated patterns of dense packings of equal disks in a square. *The Electronic Journal of Combinatorics*, 3(16):211–227, 1996.

[51] Laurent Granvilliers. On the Combination of Interval Constraint Solvers. *Reliab. Comput.*, 7(6):467–483, 2001.

[52] Laurent Granvilliers. Realpaver. disponible sur http://www.sciences.univ-nantes.fr/info/perso/permanents/granvil/realpaver/, 2003. IRIN - Université de Nantes,

[53] Laurent Granvilliers et Frederic Benhamou. Progress in the Solving of a Circuit Design Problem. *J. Global Optim.*, 20(2):155–168, 2001.

[54] Vineet Gupta, Radha Jagadeesan et Vijay A. Saraswat. Computing with continuous change. *Science of Computer Programming*, 30(1-2):3–49, 1998.

[55] Vineet Gupta, Radha Jagadeesan, Vijay A. Saraswat et Daniel G. Bobrow. Programming in hybrid constraint languages. In *Hybrid Systems*, pages 226–251, 1994.

[56] Horst Hamacher. Meeting of the working group "location and transportation". Rapport technique 54, Mathematisches Forschungsinstitut Oberwolfach, 2001.

[57] Robert M. Haralick et Gordon L. Elliott. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence*, 14:263–313, 1980.

[58] Warwick Harvey. Computing two-dimensional integer hulls. *Society for Industrial and Applied Mathematics*, 28(6):2285–2299, 1999.

[59] Michael Heusch. distn: An euclidean distance global constraint. In Francesca Rossi, réd., *Proc. of CP 2003*, volume 2833 of *Lecture Notes in Computer Science*, page 975. Springer, 2003.

[60] Michael Heusch. Aide au déploiement d'antennes radio par une contrainte globale de distance euclidienne. In *Actes de ROADEF-2005, 6^e congrès de la société Française de Recherche Opérationnelle et d'Aide à la Décision*, pages 213–214, Tours, France, 2005.

[61] Michael Heusch. Modélisation hybride discrète-continue d'un problème d'aide au déploiement d'antennes radio. In *Actes de ROADEF-2006, 7^e congrès de la société Française de Recherche Opérationnelle et d'Aide à la Décision*, Lille, France, 2006.

[62] John Hooker. *Logic-Based Methods for Optimization.* Wiley Interscience, 2000.

[63] Wen Qi Huang, Yu Li, Bernard Jurkowiak, Chu Min Li et Ru Chu Xu. A two-level search strategy for packing unequal circles into a circle container. In Francesca Rossi, réd., *Proc. of CP 2003*, volume 2833 of *Lecture Notes in Computer Science*. Springer, 2003.

[64] Eero Hyvönen. Constraint Reasoning based on Interval Arithmetic. The Tolerance Propagation Approach. *Artificial Intelligence*, 58:71–112, 1992.

[65] IEEE. IEEE Standard for Binary Floating-Point Arithmetic. Rapport technique IEEE Std 754-1985, 1985. Reaffirmed 1990,

[66] J.Krarup, D.Pisinger et F.Plastria. Discrete location problems with push-pull objectives. *Discrete Applied Mathematics*, 123:363–378, 2002.

[67] Matthew J. Katz, Klara Kedem et Michael Segal. Improved algorithms for placing undesirable facilities. *Computers & Operations Research*, 29(13):1859–72, 2002.

[68] S. Kirkpatrick, C.D. Gelatt et M.P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, May 1983.

[69] Donald E. Knuth. *Seminumerical Algorithms*, volume 2 of *The Art of Computer Programming.* Addison-Wesley, 2^e édition, 1981.

[70] A. M. C. A. Koster. *Frequency Assignment - Models and Algorithms.* Thèse de Doctorat, Maastricht University, 1999.

[71] Ludwig Krippahl et Pedro Barahona. PSICO: Solving protein structures with constraint programming and optimization. *Constraints*, 7(3-4):317–331, 2002.

[72] François Laburthe et Le Projet OCRE. Choco : implémentation du noyau d'un système de contraintes. In Christian Bessière, réd., *actes des 6e Journées Nationales sur la résolution de Problèmes NP-Complets*, pages 151–165, `http://www.lirmm.fr/~bessiere/jnpc00/laburthe.doc`, juin 2000. ONERA.

[73] François Laburthe, Pierre Savéant, Simon de Givry et Jean Jourdan. Eclair: A library of constraints over finite domains. Rapport technique ATS 98-2, Thomson-CSF LCR, Orsay, France, 1998.

[74] Jean-Louis Laurière. *Intelligence Artificielle*. EYROLLES, 1987.

[75] Y. Lebbah, C. Michel et M. Rueher. Global filtering algorithms based on linear relaxations. In *2nd International Workshop on Global Constrained Optimization and Constraint Satisfaction (Cocos'03)*, Lausanne, Suisse, novembre 2003.

[76] Yahia Lebbah et Olivier Lhomme. Accelerating filtering techniques for numeric csps. *Artif. Intell.*, 139(1):109–132, 2002.

[77] Yahia Lebbah, Claude Michel, Michel Rueher, David Daney et Jean-Pierre Merlet. Efficient and safe global constraints for handling numerical constraint systems. *Accepted for publication in the SIAM Journal on Numerical Analysis*, 2005.

[78] Yahia Lebbah, Michel Rueher et Claude Michel. A global filtering algorithm for handling systems of quadratic equations and inequations. In *Principles and Practice of Constraint Programming,8th International Conference*, 2002.

[79] Olivier Lhomme. Consistency Techniques for Numeric CSPs. In *Procs. IJCAI'93*, pages 232–238, 1993.

[80] Alan Mackwotrth. Consistency in networks of relations. *Artificial intelligence*, 8(1):99–118, 1977.

[81] Alan Mackwotrth. On reading sketch maps. In *Proc. of IJCAI-77*, pages 598–606, Cambridge, MA, 1977.

[82] C. D. Maranas, C. A. Floudas et P. M.Pardalos. New results in the packing of equal circles in a square. *Discrete Mathematics*, (142):287–293, 1995.

[83] Míhaly C. Markót. An interval method to validate optimal solutions of the "packing circles in a unit square" problems. *Central European Journal of Operations Research*, 8:63–78, 2000.

[84] Míhaly C. Markót et Tibor Csendes. A new verified optimization technique for the "packing circles in a unit square" problems. *SIAM J. Optimization*, *accepted*.

[85] K. Marriott et P. J. Stuckey. *Programming with Constraints: An Introduction*. The MIT Press, 1998.

[86] N. Megiddo. Linear-time algorithms for linear programming in $\mathbb{R}^3$ and related problems. *SIAM Journal on Computing*, 12:759–776, 1983.

[87] J. Melissen et P. Schuur. Improved coverings of a square with six and eight equal circles. *Electronic Journal of Combinatorics*, 3(R32), 1996.

[88] J.-P. Merlet. Solving the forward kinematics of a gough-type parallel manipulator with interval analysis. *The International Journal of Robotics Research*, 23:221–235, 2004.

[89] Roger Mohr et Thomas C. Henderson. Arc and path consistency revisited. *Artificial Intelligence*, 28(2):225–33, 1986.

[90] Roger Mohr et Gérald Masini. Good old discrete relaxation. In *ECAI'88*, pages 651–656, 1988.

[91] U. Montanari. Networks of constraints: Fundamental properties and applications to picture processing. *Information Sciences*, 7:95–132, 1974.

[92] Ramon E. Moore. *Interval Analysis*. Prentice-Hall, Englewood Cliffs, NJ, 1966.

[93] Arnold Neumaier. *Interval Methods for Systems of Equations*. Cambridge University Press, 1990.

[94] Arnold Neumaier et Oleg Shcherbina. Safe bounds in linear and mixed-integer programming. to appear in Mathematical Programming,

[95] Kari J. Nurmela et Patric R.J. Östergård. Optimal packings of equal circles in a square. In *Proc. of the 8th Quadrennial International Conference on Graph Theory, Combinatorics, Algorithms and Applications*, 1996.

[96] Kari J. Nurmela et Patric R.J. Östergård. More optimal packings of equal circles in a square. *Discrete and Computational Geometry, submitted*, 22:439–457, 1999.

[97] J. Pach et P. Agarwal. *Combinatorial Geometry*. Wiley-Interscience, 1995.

[98] C. H. Papadimitriou. *Computational complexity*. Addison-Wesley Publishing company, première édition, 1994.

[99] R. Peikert. Dichteste packungen von gleichen kreisen in einem quadrat. *Elem. Math. 49*, pages 16–26, 1994.

[100] Gilles Pesant et Michel Boyer. Reasoning about solids using constraint logic programming. *J. Autom. Reason.*, 22(3):241–262, 1999.

[101] Jean-François Puget et Pascal Van Hentenryck. A constraint satisfaction approach to a circuit design problem. Research Report CS-96-34, Brown University, December 1996.

[102] Jean-François Puget et Pascal Van Hentenryck. A constraint satisfaction approach to a circuit design problem. *Journal of Global Optimization*, 13(1):75–93, 1998.

[103] Philippe Refalo. Linear formulation of constraint programming models and hybrid solvers. In *Proc. of CP-00*, pages 369–383, Singapore, September 2000. Springer-Verlag.

[104] Jean-Charles Régin. A filtering algorithm for constraints of difference in CSPs. In *Proceedings AAAI94*, pages 362–367, Seattle, Washington, 1994.

[105] Jean-Charles Régin. Global constraints and filtering algorithms. In Michela Milano, réd., *CONSTRAINT AND INTEGER PROGRAMMING, Toward a Unified Methodology*, volume 27 of *Operations Research/Computer Science Interfaces*. Kluwer Academic Publishers, 2004.

[106] R. Rodosek, M. G. Wallace et M. T. Ha jian. A new approach to integrating mixed integer programming and constraint logic programming. *Annals of Operations Research*, 86:63 – 87, 1999.

[107] F. Rossi, V. Dahr et C. Petrie. On the equivalence of constraint satisfaction problems. In *Proceedings. European Conference on Artificial Intelligence, ECAI90*, Stockholm, August 1990. Also: MCC Technical Report ACT-AI-222-89,

[108] Michel Rueher et Christine Solnon. Concurrent cooperating solvers over reals. *Reliable Computing*, 3(3):325–333, Oktober 1997.

[109] Raimund Seidel. Linear programming and convex hulls made easy. In *Proc. of the sixth annual symposium on Computational geometry*, pages 211–215, 1990.

[110] Hanif D. Sherali et Warren P. Adams. *A Reformulation-Linearization Technique for Solving Discrete and Continuous Nonconvex Problems*. Number 31 in Nonconvex Optimization and its Applications. Kluwer Publisher, December 1999.

[111] Péter Gabór Szabó. Some new structures for the "equal circles packing in a square" problem, special issue: Proceedings of the xxiv. hungarian operations research conference. *Central European Journal of Operations Research*, 8:79–91, 2000.

[112] Boglárka Tóth et Tibor Csendes. Empirical investigation of the convergence speed of inclusion functions. In *10th GAMM - IMACS International Symposium on Scientific Computing, Computer Arithmetic, and Validated Numerics (SCAN 2002)*, Paris, September 2002.

[113] Michael W. Trosset. Approximate maximin distance designs. In *Proc. of the Section on Physical and Engineering Science*, pages 223–227. American Statistical Association, 1999.

[114] Pascal Van Hentenryck, David McAllester et Deepak Kapur. Solving polynomial systems using a branch and prune approach. *SIAM Journal on Numerical Analysis*, 34(2):797–827, avril 1997.

[115] J. Vignes. A stochastic arithmetic for reliable scientific computation. *Math. Comput. Simul.*, 35(3):233–261, 1993.

[116] D. L. Waltz. Generating semantic descriptions from drawings of scenes with shadows. Technical Report AI-TR-271, MIT, 1972.

[117] Matthew L. Ginsberg William D. Harvey. Limited discrepancy search. In Chris S. Mellish, réd., *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95); Vol. 1*, pages 607–615, Montréal, Québec, Canada, August 20-25 1995. Morgan Kaufmann, 1995.

Liste des tableaux

Table des figures

Table des matières

Index

Annexe

Études de cas du premier algorithme de filtrage de *distn*

Étude de cas pour des contraintes de distance minimale

Les travaux sur le problème de packing de cercles ont été effectués en appliquant un filtrage sur des boîtes disjointes et dont la diagonale est de longueur inférieure à d[3]. Dans l'article [83], Markót indique que la méthode des aires actives ne peut pas réduire les domaines lorsque les pavés considérés s'intersectent ou quand ils sont trop grands, ce qui justifie en soi le recours à un pavage du carré.

Nous montrons ici que ce constat est par trop pessimiste et qu'il est possible d'utiliser les principes de cette technique de filtrage dans un cadre plus vaste ; ce point est essentiel, car le cadre d'une tessellation est incompatible avec la portée générale de la sémantique de la contrainte globale et les particularités des domaines d'applications qu'elle vise. Le filtrage que nous proposons est donc effectif – même si la diagonale des domaines est supérieure à la distance minimale à propager – et le reste encore pour des boîtes se trouvant sous des conditions de position générale.

Nous étudions ici le filtrage opéré par un pavé $P_g = X_g \times Y_g$ (il est situé sur la gauche des schémas qui suivent) sur le pavé $P_d = X_d \times Y_d$ (le pavé de droite). Introduisons aussi la notation $\check{Z}$ pour désigner le centre d'un intervalle Z. Notre seule hypothèse de travail est que le centre du pavé à réduire est situé « en haut à droite » du centre du pavé réducteur, *i.e.* on a $\check{X}_g < \check{X}_d$ et $\check{Y}_g < \check{Y}_d$ (les centres des pavés sont illustrés par des lignes en pointillés sur la Figure 8). L'étude se rapporte à ce cas pour chacune des trois autres positions relatives des centres par une rotation de la figure. L'étude de cas se décompose sous ces hypothèses en quatre sous-cas :

$$\begin{array}{ll} \text{(i)} & \check{X}_g < \underline{X_d} \text{ et } \check{Y}_g > \underline{Y_d} \\ \text{(ii)} & \check{X}_g > \underline{X_d} \text{ et } \check{Y}_g > \underline{Y_d} \\ \text{(iii)} & \check{X}_g < \underline{X_d} \text{ et } \check{Y}_g < \underline{Y_d} \\ \text{(iv)} & \check{X}_g > \underline{X_d} \text{ et } \check{Y}_g < \underline{Y_d} \end{array}$$

En écartant la situation immédiatement non-satisfiable où

$$\overline{dist(P_g, P_d)} < d \tag{5}$$

[3]Ces boites sont extraites du pavage initial du carré. L'intérêt principal de cette tessellation est qu'il assure que chaque pavé contient un seul point.

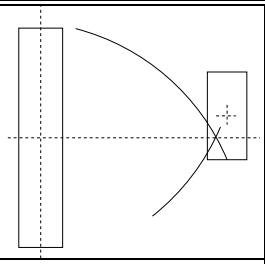
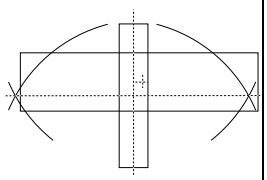
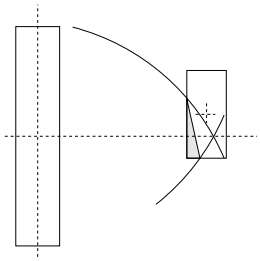
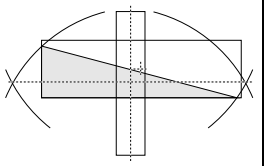
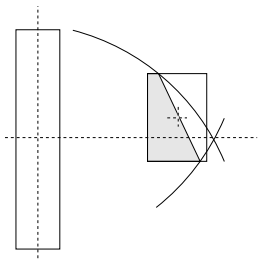
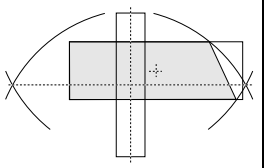
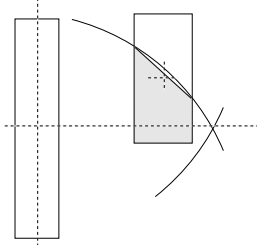
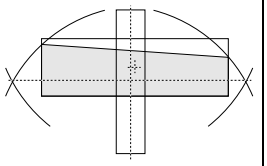
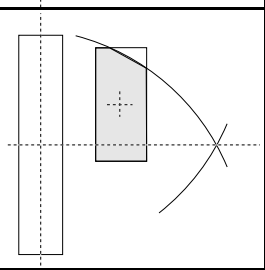
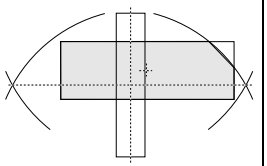
(i)		$\check{X}_g < \underline{X_d}$ et $\check{Y}_g > \underline{Y_d}$	(ii)		$\check{X}_g > \underline{X_d}$ et $\check{Y}_g > \underline{Y_d}$
1		$\left\{ dist((\underline{X_g},\overline{Y_g}),(\underline{X_d},\underline{Y_d}))^2 > d^2 \right.$	1		$\begin{cases} dist((\underline{X_g},\overline{Y_g}),(\overline{X_d},\underline{Y_d}))^2 > d^2 \\ dist((\overline{X_g},\overline{Y_g}),(\underline{X_d},\underline{Y_d}))^2 > d^2 \end{cases}$
2		$\begin{cases} dist((\underline{X_g},\overline{Y_g}),(\underline{X_d},\underline{Y_d}))^2 < d^2 \\ dist((\underline{X_g},\overline{Y_g}),(\overline{X_d},\underline{Y_d}))^2 > d^2 \\ dist((\underline{X_g},\underline{Y_g}),(\underline{X_d},\overline{Y_d}))^2 > d^2 \end{cases}$ $\Rightarrow$ $p_g = (\underline{X_d}, \underline{Y_g} + \sqrt{d^2 - (\underline{X_d} - \underline{X_g})^2})$ $p_d = (\underline{X_g} + \sqrt{d^2 - (\overline{Y_g} - \underline{Y_d})^2}, \underline{Y_d})$	2		$\begin{cases} dist((\overline{X_g},Y_g),(\underline{X_d},\overline{Y_d}))^2 > d^2 \\ dist((\overline{X_g},\overline{Y_g}),(\underline{X_d},\underline{Y_d}))^2 < d^2 \end{cases}$ $\Rightarrow$ $p_g = (\underline{X_d}, \underline{Y_g} + \sqrt{d^2 - (\overline{X_g} - \underline{X_d})^2})$ $p_d = (\underline{X_g} + \sqrt{d^2 - (\overline{X_g} - \underline{X_d})^2}, \underline{Y_d})$
3		$\begin{cases} dist((\underline{X_g},\overline{Y_g}),(\underline{X_d},\underline{Y_d}))^2 < d^2 \\ dist((\underline{X_g},\overline{Y_g}),(\overline{X_d},\underline{Y_d}))^2 > d^2 \\ dist((\underline{X_g},\underline{Y_g}),(\underline{X_d},\overline{Y_d}))^2 < d^2 \end{cases}$ $\Rightarrow$ $p_g = (\underline{X_g} + \sqrt{d^2 - (\overline{Y_d} - \underline{Y_g})^2}, \overline{Y_d})$ $p_d = (\underline{X_g} + \sqrt{d^2 - (\overline{Y_g} - \underline{Y_d})^2}, \underline{Y_d})$	3		$\begin{cases} dist((\underline{X_g},\overline{Y_g}),(\overline{X_d},\underline{Y_d}))^2 > d^2 \\ dist((\overline{X_g},\underline{Y_g}),(\underline{X_d},\overline{Y_d}))^2 < d^2 \end{cases}$ $\Rightarrow$ voir cas 3 (i)
4		$\begin{cases} dist((\underline{X_g},\overline{Y_g}),(\overline{X_d},\underline{Y_d}))^2 < d \\ dist((\overline{X_g},\underline{Y_g}),(\underline{X_d},\overline{Y_d}))^2 > d^2 \end{cases}$ $\Rightarrow$ $p_g = (\underline{X_d}, \underline{Y_g} + \sqrt{d^2 - (\underline{X_d} - \underline{X_g})^2})$ $p_d = (\overline{X_d}, \underline{Y_g} + \sqrt{d^2 - (\overline{X_d} - \underline{X_g})^2})$	4		$\begin{cases} dist((\underline{X_g},\overline{Y_g}),(\overline{X_d},\underline{Y_d}))^2 < d^2 \\ dist((\overline{X_g},\underline{Y_g}),(\underline{X_d},\overline{Y_d}))^2 > d^2 \end{cases}$ $\Rightarrow$ $p_g = (\underline{X_d}, \underline{Y_g} + \sqrt{d^2 - (\overline{X_g} - \underline{X_d})^2})$ $p_d = (\underline{X_g} + \sqrt{d^2 - (\underline{Y_d} - \underline{Y_g})^2}, \underline{Y_d})$
5		$\begin{cases} dist((\underline{X_g},\overline{Y_g}),(\overline{X_d},\underline{Y_d}))^2 < d^2 \\ dist((\underline{X_g},\underline{Y_g}),(\underline{X_d},\overline{Y_d}))^2 < d^2 \\ dist((\underline{X_g},\underline{Y_g}),(\overline{X_d},\overline{Y_d}))^2 > d^2 \end{cases}$ $\Rightarrow$ $p_g = (\underline{X_g} + \sqrt{d^2 - (\overline{Y_d} - \underline{Y_g})^2}, \overline{Y_d})$ $p_d = (\overline{X_d}, \underline{Y_g} + \sqrt{d^2 - (\overline{X_d} - \underline{X_g})^2})$	5		$\begin{cases} dist((\underline{X_g},\overline{Y_g}),(\overline{X_d},\underline{Y_d}))^2 < d^2 \\ dist((\overline{X_g},\underline{Y_g}),(\underline{X_d},\overline{Y_d}))^2 < d^2 \\ dist((\underline{X_g},\underline{Y_g}),(\overline{X_d},\overline{Y_d}))^2 > d^2 \end{cases}$ $\Rightarrow$ voir cas 5 (i)

Figure 6 – Étude des cas (i) et (ii) avec $\check{X}_g < \check{X}_d$ et $\check{Y}_g < \check{Y}_d$. La corde sur le pavé réduit est définie par les points p_g (le plus à gauche) et p_d.

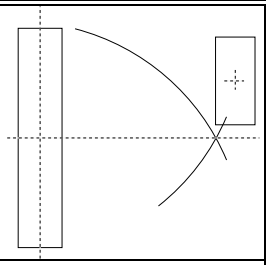
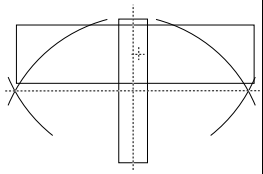
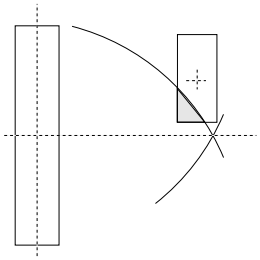
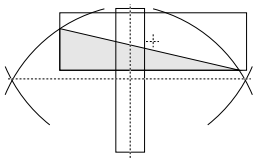
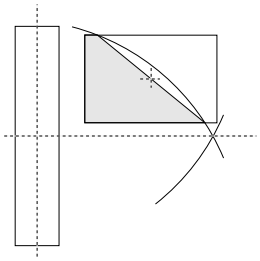
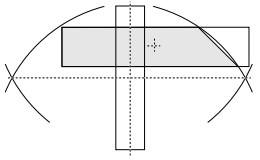
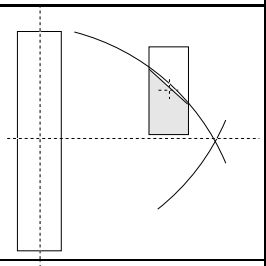
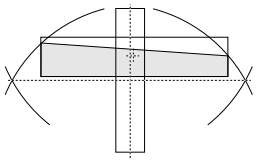
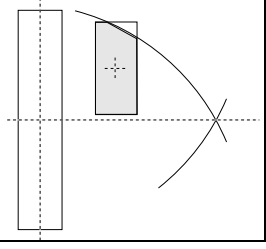
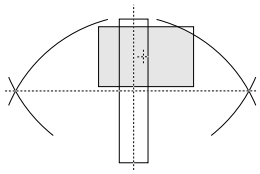
(iii)		$\check{X}_g < \underline{X_d}$ et $\check{Y}_g < \underline{Y_d}$	(iv)		$\check{X}_g > \underline{X_d}$ et $\check{Y}_g < \underline{Y_d}$
1		$\left\{ dist((\underline{X_g}, \overline{Y_g}), (\underline{X_d}, \underline{Y_d}))^2 > d^2 \right.$	1		$\begin{cases} dist((\underline{X_g}, \overline{Y_g}), (\overline{X_d}, \underline{Y_d}))^2 < d^2 \\ dist((\underline{X_g}, \underline{Y_g}), (\overline{X_d}, \overline{Y_d}))^2 > d^2 \\ dist((\overline{X_g}, \underline{Y_g}), (\underline{X_d}, \overline{Y_d}))^2 > d^2 \\ dist((\underline{X_d}, \overline{Y_g}), (\overline{X_g}, \underline{Y_d}))^2 < d^2 \end{cases}$
2		$\begin{cases} dist((\underline{X_d}, \underline{Y_d}), (\underline{X_g}, \overline{Y_g}))^2 \leq d^2 \\ dist((\overline{X_d}, \underline{Y_d}), (\underline{X_g}, \overline{Y_g}))^2 > d^2 \\ dist((\underline{X_d}, \overline{Y_d}), (\underline{X_g}, \underline{Y_g}))^2 > d^2 \end{cases}$ $\Rightarrow$ $p_g = (\underline{X_d}, \underline{Y_g} + \sqrt{d^2 - (\underline{X_d} - \underline{X_g})^2})$ $p_d = (\underline{X_g} + \sqrt{d^2 - (\overline{Y_g} - \underline{Y_d})^2}, \underline{Y_d})$	2		$\begin{cases} dist((\underline{X_g}, \overline{Y_g}), (\overline{X_d}, \underline{Y_d}))^2 > d^2 \\ dist((\underline{X_g}, \underline{Y_g}), (\overline{X_d}, \overline{Y_d}))^2 > d^2 \\ dist((\overline{X_g}, \underline{Y_g}), (\underline{X_d}, \overline{Y_d}))^2 > d^2 \\ dist((\overline{X_g}, \overline{Y_g}), (\underline{X_d}, \underline{Y_d}))^2 < d^2 \end{cases}$ $\Rightarrow$ $p_g = (\underline{X_d}, \underline{Y_g} + \sqrt{d^2 - (\overline{X_g} - \underline{X_d})^2})$ $p_d = (\underline{X_g} + \sqrt{d^2 - (\overline{X_g} - \underline{X_d})^2}, \underline{Y_d})$
3		$\begin{cases} dist((\underline{X_g}, \overline{Y_g}), (\underline{X_d}, \underline{Y_d}))^2 \leq d^2 \\ dist((\underline{X_g}, \overline{Y_g}), (\overline{X_d}, \underline{Y_d}))^2 > d^2 \\ dist((\underline{X_g}, \underline{Y_g}), (\underline{X_d}, \overline{Y_d}))^2 \leq d^2 \end{cases}$ $\Rightarrow$ $p_g = (\underline{X_g} + \sqrt{d^2 - (\overline{Y_d} - \underline{Y_g})^2}, \overline{Y_d})$ $p_d = (\underline{X_g} + \sqrt{d^2 - (\overline{Y_g} - \underline{Y_d})^2}, \underline{Y_d})$	3		$\begin{cases} dist((\underline{X_g}, \underline{Y_g}), (\underline{X_d}, \underline{Y_d}))^2 \leq d^2 \\ dist((\underline{X_g}, \underline{Y_g}), (\underline{X_d}, \overline{Y_d}))^2 > d^2 \end{cases}$ $\Rightarrow$ voir cas 3 (iii)
4		$\begin{cases} dist((\underline{X_g}, \overline{Y_g}), (\overline{X_d}, \underline{Y_d}))^2 < d \\ dist((\underline{X_g}, \underline{Y_g}), (\underline{X_d}, \overline{Y_d}))^2 > d^2 \end{cases}$ $\Rightarrow$ voir cas 4 (i)	4		$\begin{cases} dist((\underline{X_g}, \underline{Y_g}), (\underline{X_d}, \underline{Y_d}))^2 \leq d^2 \\ dist((\underline{X_g}, \underline{Y_g}), (\underline{X_d}, \overline{Y_d}))^2 \leq d^2 \end{cases}$ $\Rightarrow$ voir cas 4 (ii)
5		$\begin{cases} dist((\underline{X_g}, \overline{Y_g}), (\overline{X_d}, \underline{Y_d}))^2 < d^2 \\ dist((\underline{X_g}, \underline{Y_g}), (\underline{X_d}, \overline{Y_d}))^2 < d^2 \end{cases}$ $\Rightarrow$ voir cas 5 (i)	5		$\begin{cases} dist((\underline{X_g}, \overline{Y_g}), (\overline{X_d}, \underline{Y_d}))^2 > d^2 \\ dist((\underline{X_g}, \underline{Y_g}), (\overline{X_d}, \overline{Y_d}))^2 > d^2 \\ dist((\overline{X_g}, \underline{Y_g}), (\underline{X_d}, \overline{Y_d}))^2 > d^2 \end{cases}$ $\Rightarrow$ voir cas 5 (i)

Figure 7 – Étude des cas (iii) et (iv) avec $\check{X}_g < \check{X}_d$ et $\check{Y}_g < \check{Y}_d$. La corde sur le pavé réduit est définie par les points p_g (le plus à gauche) et p_d.

il reste en tout les 20 possibilités décrites sur les Figures 6 et 7 p. 136 et 137. Elles donnent les conditions nécessaires et suffisantes pour décrire les points supportant la corde qui approche l'aire active.

Pour rendre la lecture du tableau tout à fait claire, nous détaillons séparément sur la Figure 8 le cas $3(i)$ de ce tableau. Les autres cas sont tout à fait analogues.

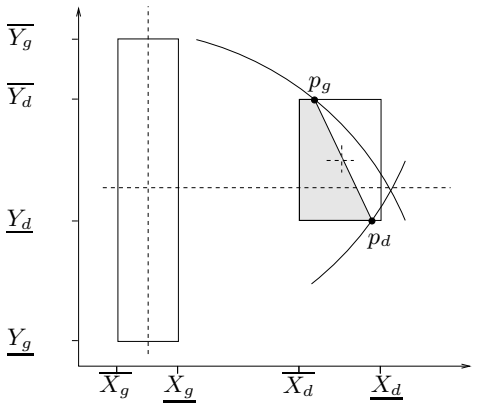


Figure 8 – Filtrage pour la distance minimale du cas $3(i)$

Expliquons comment on peut construire l'ensemble des droites définissant l'approximation de la région permise :

- La condition (5) implique les conditions suivantes pour le point d'intersection du noyau de rayon d avec P_g.

$$\left\{ dist((\underline{X_g}, \underline{Y_g}), (\underline{X_d}, \overline{Y_d}))^2 < d^2 \right. \quad \Rightarrow \quad p_g = (\underline{X_g} + \sqrt{d^2 - (\overline{Y_d} - \underline{Y_g})^2}, \overline{Y_d})$$

- Le point de droite est quant à lui donné par

$$\begin{cases} dist((\underline{X_g}, \overline{Y_g}), (\underline{X_d}, \underline{Y_d}))^2 < d^2 \\ dist((\underline{X_g}, \overline{Y_g}), (\overline{X_d}, \underline{Y_d}))^2 > d^2 \end{cases} \quad \Rightarrow \quad p_d = (\underline{X_g} + \sqrt{d^2 - (\overline{Y_g} - \underline{Y_d})^2}, \underline{Y_d})$$

Si l'on est comme ici avec $\check{X}_g < \check{X}_d$ et $\check{Y}_g < \check{Y}_d$, la droite associée est toujours décroissante, et on s'intéresse alors au filtrage défini par le demi-plan :

$$(p_g(x) - p_d(x))\ x + (p_g(y) - p_d(y))\ y \leq d^2.$$

Étude de cas pour des contraintes de distance maximale

L'étude de cas pour les contraintes de distance maximale est un peu plus simple. Elle s'applique également de manière symétrique, mais cette fois-ci en additionnant l'approximation faite en chacun des quatre coins du pavé réducteur.

Nous la présentons ici dans le coin $(\overline{X_2}, \overline{Y_2})$ en considérant le filtrage du pavé $P_2 = X_2 \times Y_2$ sur $P_1 = X_1 \times Y_1$. Nous y imposons *a priori* les conditions suivantes :

$$\begin{cases} \underline{dist(P_g, P_d)} \leq d \\ \overline{X_2} < \overline{X_1} \\ \overline{Y_2} < \overline{Y_1} \end{cases} \tag{6}$$

La violation de la première inéquation traduit un cas immédiatement non-satisfiable, et la violation des deux suivantes entraîne qu'aucune approximation n'est à introduire en ce coin. Il existe alors pour le pavé P_1 exactement 10 façons possibles d'intersecter la frontière de la zone « permise » par la contrainte de distance maximale de P_2 au coin $(\overline{X_2}, \overline{Y_2})$. L'étude de cas complète est présentée sur la Figure 10 page 141. Le pavé P_2 n'est pas représenté sur la figure ; seules apparaissent les coordonnées d'un de ses coins dans un repère, la courbe donnant la zone permise par la contrainte et l'approximation qui en est faite par trois tangentes en ce coin.

Pour y apporter d'avantage de clarté, nous détaillons séparément sur la Figure 9 le cas 7 de ce tableau. Les autres cas sont tout à fait analogues.

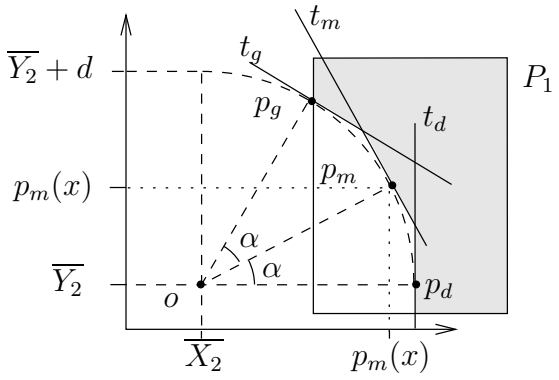


Figure 9 – Filtrage pour la distance maximale en $(\overline{X_2}, \overline{Y_2})$

Expliquons comment on peut construire l'ensemble des droites définissant l'approximation de la région permise. L'étude au point $o = (\overline{X_2}, \overline{Y_2})$ introduit une approximation de la contrainte de distance maximale par trois tangentes au quart de cercle centré en o. Ces droites, notées t_g, t_m et t_d passent par les points p_g, p_m et p_d définis de la manière suivante :

- le point p_g d'intersection de la frontière de la région permise avec P_1 situé sur la gauche est défini par :

$$\begin{cases} \underline{X_1} > \overline{X_2} \\ dist((\overline{X_2}, \overline{Y_2}), (\underline{X_1}, \overline{Y_1})) > d \end{cases} \quad \Rightarrow \quad p_g = (\underline{X_1}, \overline{\overline{Y_2} + \sqrt{d^2 - (\underline{X_1} - \overline{X_2})^2}})$$

- le point p_d d'intersection de la frontière de la région permise avec P_1 situé le plus à droite est défini par :

$$\begin{cases} \overline{X_1} > \overline{X_2} + d \\ \underline{Y_1} < \overline{Y_2} \end{cases} \quad \Rightarrow \quad p_d = (\overline{\overline{X_2} + d}, \overline{Y_2})$$

- p_m est calculé comme le point d'intersection du quart de cercle avec la médiane de l'angle formé par les points p_g, o et p_d.

Les coordonnées du point p_m peuvent être obtenues à partir des relations trigonométriques suivantes : en notant $\overline{p_g p_d}$ la longueur du segment de p_g à p_d, les égalités suivantes se présentent :

$$\begin{cases} \frac{p_g(y) - p_d(y)}{\overline{p_g p_d}} = \frac{p_m(x) - \overline{X_2}}{d} \\ \frac{p_g(x) - p_d(x)}{\overline{p_g p_d}} = \frac{p_m(y) - \overline{Y_2}}{d} \end{cases}$$

L'équation d'une tangente en (x_i, y_i) au quart de cercle centré en o étant donnée par

$$x_i(x - \overline{X_2}) + y_i(y - \overline{Y_2}) = d^2$$

nous en déduisons les inéquations correspondant à t_g, t_2 et t_d. En $(\overline{X_2}, \overline{Y_2})$ on s'intéresse aux demi plans situés en dessous des droites toujours décroissantes t_i ; l'approximation en p_i sera pour cela donnée par

$$x_i(x - \overline{X_2}) + y_i(y - \overline{Y_2}) \leq d^2$$

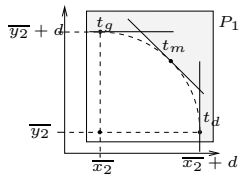
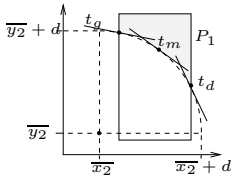
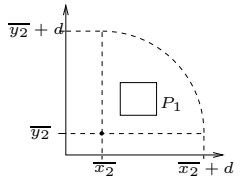
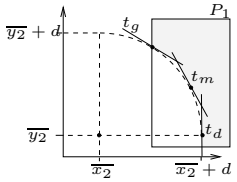
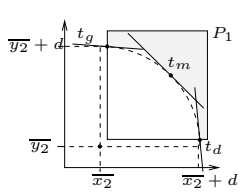
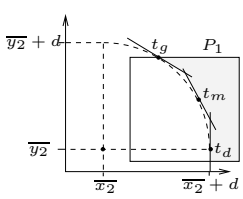
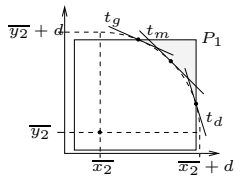
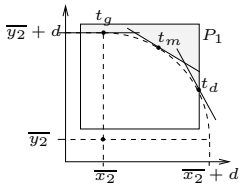
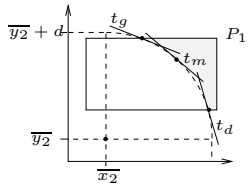
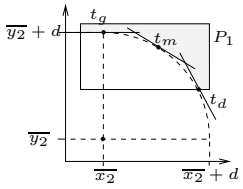
1		$\begin{cases}\underline{X_1} < \overline{X_2} \\ \underline{Y_1} < \overline{Y_2} \\ \overline{X_1} > \overline{X_2}+d \\ \overline{Y_1} > \overline{Y_2}+d \end{cases}$ $\Rightarrow$ $p_g = (\overline{X_2}, \overline{Y_2}+d)$ $p_d = (\overline{X_2}+d, \overline{Y_2}+d)$	6		$\begin{cases}\underline{X_1} > \underline{X_2} \\ \overline{X_1} < \overline{X_2}+d \\ dist((\overline{X_2}, \overline{Y_2}), (\underline{X_1}, \overline{Y_1})) > d \\ dist((\overline{X_2}, \overline{Y_2}), (\overline{X_1}, \uparrow \underline{Y_1})) < d \end{cases}$ $\Rightarrow$ $p_g =$ voir cas 3 $p_d =$ voir cas 4
2		$\begin{cases}\underline{X_1} < \overline{X_2} \\ \underline{Y_1} < \overline{Y_2} \\ dist((\overline{X_2}, \overline{Y_2}), (\overline{X_1}, \overline{Y_1})) < d \end{cases}$	7		$\begin{cases}\underline{X_1} > \overline{X_2} \\ \overline{X_1} > \overline{X_2}+d \\ \underline{Y_1} < \overline{Y_2} \\ dist((\overline{X_2}, \overline{Y_2}), (\underline{X_1}, \overline{Y_1})) > d \end{cases}$ $\Rightarrow$ $p_g =$ voir cas 3 $p_d = (\overline{X_2}+d, \overline{Y_2})$
3		$\begin{cases}\underline{X_1} > \overline{X_2} \\ \underline{Y_1} > \overline{Y_2} \\ dist((\overline{X_2}, \overline{Y_2}), (\underline{X_1}, \overline{Y_1})) > d \\ dist((\overline{X_2}, \overline{Y_2}), (\overline{X_1}, \underline{Y_1})) > d \end{cases}$ $\Rightarrow$ $p_g = (\underline{X_1}, \overline{Y_2}+\sqrt{d^2-(\underline{X_1}-\overline{X_2})^2})$ $p_d = (\overline{X_2}+\sqrt{d^2-(\underline{Y_1}-\overline{Y_2})^2}, \underline{Y_1})$	8		$\begin{cases}\overline{Y_1} < \overline{Y_2}+d \\ \overline{X_1} > \overline{X_2}+d \\ \underline{Y_1} < \overline{Y_2} \\ dist((\overline{X_2}, \overline{Y_2}), (\uparrow \underline{X_1}, \overline{Y_1})) < d \end{cases}$ $\Rightarrow$ $p_g =$ voir cas 4 $p_d = (\overline{X_2}+d, \overline{Y_2})$
4		$\begin{cases}\overline{X_1} < \overline{X_2}+d \\ \overline{Y_1} < \overline{Y_2}+d \\ dist((\overline{X_2}, \overline{Y_2}), (\underline{X_1}, \uparrow \overline{Y_1})) < d \\ dist((\overline{X_2}, \overline{Y_2}), (\uparrow \overline{X_1}, \underline{Y_1})) < d \end{cases}$ $\Rightarrow$ $p_g = (\underline{X_1}, \overline{Y_2}+\sqrt{d^2-(\underline{X_1}-\overline{X_2})^2})$ $p_d = (\overline{X_2}+\sqrt{d^2-(\underline{Y_1}-\overline{Y_2})^2}, \underline{Y_1})$	9		$\begin{cases}\underline{X_1} < \overline{X_2} \\ \overline{Y_1} > \overline{Y_2}+d \\ \overline{X_1} < \overline{X_2}+d \\ dist((\overline{X_2}, \overline{Y_2}), (\overline{X_1}, \uparrow \underline{Y_1})) < d \end{cases}$ $\Rightarrow$ $p_g = (\overline{X_2}, \overline{Y_2}+d)$ $p_d =$ voir cas 4
5		$\begin{cases}\overline{Y_1} < \overline{Y_2}+d \\ \underline{Y_1} > \overline{Y_2} \\ dist((\overline{X_2}, \overline{Y_2}), (\uparrow \underline{X_1}, \overline{Y_1})) < d \\ dist((\overline{X_2}, \overline{Y_2}), (\overline{X_1}, \underline{Y_1})) > d \end{cases}$ $\Rightarrow$ $p_g =$ voir cas 4 $p_d =$ voir cas 3	10		$\begin{cases}\underline{X_1} < \overline{X_2} \\ \overline{Y_1} > \overline{Y_2}+d \\ \underline{Y_1} > \overline{Y_2} \\ dist((\overline{X_2}, \overline{Y_2}), (\overline{X_1}, \underline{Y_1})) > d \end{cases}$ $\Rightarrow$ $p_g = (\overline{X_2}, \overline{Y_2}+d)$ $p_d =$ voir cas 3

Figure 10 – Étude de cas pour la distance maximale dans le coin $(\overline{X_2}, \overline{Y_2})$.

Modélisation et résolution d'une application d'aide au déploiement d'antennes radio en programmation par contraintes sur le discret et le continu

Michael Heusch

Résumé

La programmation par contraintes rencontre depuis le milieu des années 90 un certain succès dans la résolution d'applications combinatoires complexes. Son extension aux contraintes d'intervalles est une approche prometteuse pour traiter des contraintes non-linéaires. La résolution de systèmes hybrides discrets-continus est pour autant restée essentiellement inexplorée. Cette thèse exploite un modèle hybride pour s'attaquer à une application en permettant d'éviter la discrétisation du problème. Les deux problèmes traités sont les suivants :

1. *Contraintes globales continues :* la définition de contraintes globales a permis d'améliorer substantiellement l'expressivité et l'efficacité des solveurs de contraintes discrets. Nous spécifions ici une première contrainte globale dans un solveur continu. Elle maintient des contraintes de distance euclidienne entre n points par un algorithme géométrique.
2. *Résolution d'une application hybride discrète-continue :* l'aide au déploiement d'antennes radio est un métissage du problèmes d'allocation de fréquences radio et d'un problème d'analyse de localisation. Nous utilisons notre contrainte globale de distance euclidienne pour obtenir une résolution hybride discrète-continue efficace de ce problème.

Mots-clés : programmation par contraintes, contraintes globales, systèmes hybrides discrets-continus, allocation de fréquences, analyse de localisation, packing de cercles

Abstract

Since the mid 90's constraint programming has proved successful in solving complex combinatorial applications. Its extension to interval constraints is a promising approach to handle non-linear constraints. However, the solving of discrete-continuous hybrid systems has remained mostly unexplored. This thesis takes advantage of a hybrid model to tackle an application while avoiding the discretization of this problem. Two main issues are adressed:

1. *Continuous global constraints:* the introduction of global constraints has substantially enhanced the expressiveness and efficiency of discrete constraint solvers. We specify a first continuous global constraint in a continuous solver. It maintains Euclidean distance constraints between n points by means of a geometric algorithm.
2. *Solving of a discrete-continuous hybrid application:* deployment support of radio antennas is a crossbreeding between a radio link frequency assignment problem and a location analysis problem. Our Euclidean distance global constraint enables us to obtain an efficient discrete-continuous hybrid resolution of this problem.

Keywords: constraint programming, global constraints, discrete-continuous hybrid systems, frequency assignment, location analysis, circle packing