Programmation par contraintes pour l'optimisation de plans de test de satellites

Caroline Maillet Gérard Verfaillie Bertrand Cabon
ONERA
2 avenue Edouard Belin, BP 4025, 31055 Toulouse Cedex, France
Astrium
31 rue des Cosmonautes, Z.I. des Palays, 31402 Toulouse Cedex 4, France
{Caroline.Maillet,Gerard.Verfaillie}@onera.fr, {Caroline.Maillet,Bertrand.Cabon}@astrium.eads.net

Résumé

Les tests physiques des charges utiles des satellites de télécommunication en chambre à vide durent plusieurs semaines et mobilisent des équipes importantes. Ces tests sont organisés selon des transitions thermiques permettant chacune de tester un sous ensemble d'équipements sous différentes contraintes. L'objectif de réduction du nombre de transitions thermiques est un enjeu crucial pour Astrium. Pour modéliser ce problème, nous avons élaboré un modèle en programmation par contraintes. Ce modèle peut induire des milliers de variables sur lesquelles s'appliquent plusieurs centaines de milliers de contraintes binaires et n-aires. Pour le résoudre, nous avons développé des algorithmes spécifiques utilisant des mécanismes de retour-arrière dirigé par les conflits, des heuristiques adaptatives originales pondérant les variables et des mécanismes de redémarrage. Les expérimentations ont montré que ces mécanismes apportent des gains significatifs en termes de qualité des plans produits en un temps de calcul borné.

Abstract

Communication satellite payload tests last several weeks in thermal vacuum chambers, and mobilise large validation teams. These tests are organised over several thermal transitions, with each transition allowing the testing of a subset of the equipment under different constraints. Minimising the number of transitions is a crucial objective for Astrium. Constraint Programming was used to elaborate a mathematical model of the problem. This model may induce thousands of variables with hundreds of thousands of binary and n-ary constraints. To solve it, specific algorithms were developed, using Conflict Directed Backjumping mechanisms, new adaptive variable weighting heuristics, and restarting strategies. Experiments showed significative gains in terms of quality of the plans produced within a limited computing time.

1 Contexte industriel

Astrium est la filiale d'EADS en charge de la conception et de la fabrication de satellites d'observation et de télécommunication. Les satellites de télécommunication sont des relais géostationnaires qui assurent des services tels que la téléphonie, la télévision haute définition ou encore internet. Situé à 36 000 km de la Terre, ils reçoivent avec une très faible puissance les signaux radio fréquence (RF) émis par les stations sol. Les signaux reçus sont amplifiés, convertis en fréquence pour éviter les interférences entre signaux reçus et signaux émis. Ils sont enfin filtrés et réamplifiés avant d'être émis vers le sol. Pour assurer cette mission, les charges utiles des satellites de télécommunication sont constituées d'un grand nombre d'amplificateurs, de filtres, de convertisseurs de fréquence et de connections reliant ces différents équipements. Tous ces équipements sont redondés pour permettre des reconfigurations en cas de pannes. La charge utile d'un satellite de télécommunication dépend de la mission à assurer. L'augmentation de la complexité des missions (nombre de signaux RF) se traduit par une augmentation de la complexité des charges utiles.

Avant lancement, les milliers d'équipements d'une charge utile doivent être testés dans des conditions proches de l'environnement spatial. Pour cela, l'essentiel des tests sont réalisés en chambre à vide dans des conditions extrêmes en température. Les tests réalisés sont organisés suivant une séquence de transitions

thermiques ascendantes ou descendantes en température. A chacune des transitions thermiques correspond une certaine configuration de la charge utile. Chaque transition permet de satisfaire une partie des exigences de test définies par Astrium. L'ensemble des transitions doit assurer la couverture de l'ensemble des exigences de test.

Les opérations de test associées à une transition durent plus d'une dizaine d'heures. Au total, ces opérations de test mobilisent sur plusieurs semaines des équipes importantes présentes 24h/24. L'objectif de limitation du nombre de transitions thermiques permettant d'assurer la couverture de l'ensemble des exigences de test est donc un objectif crucial pour Astrium.

2 Problème d'optimisation des plans de test des charges utiles de télécommunication

A chaque transition thermique correspond une configuration de la charge utile. A une configuration est associé un ensemble des chemins. Un chemin correspond à un signal RF traversant la charge utile depuis une antenne réceptrice vers une antenne émettrice et passant par un ensemble d'équipements (amplificateurs, filtres, convertisseurs, routeurs et connexions entre eux). Les routeurs sont des équipements configurables qui permettent de diriger les signaux RF vers les équipements appropriés. Par exemple, la figure 1 montre une partie du chemin rouge qui utilise les routeurs 3508 et 4227, l'amplificateur 5005, les routeurs 7988 et 7990 et le filtre CH17. Lors d'une transition thermique, tous les chemins de la configuration associée et donc tous les équipements présents sur ces chemins peuvent être testés en parallèle.

Cependant l'ensemble des chemins associés à une transition doit respecter certaines contraintes.

Premièrement, tous les chemins doivent être compatibles deux à deux en termes de positions des routeurs. Par exemple sur la figure 1, les chemins rouge et bleu sont compatibles car ils n'utilisent aucun équipement en commun. Les chemins vert et bleu sont aussi compatibles car le seul équipement commun est le routeur 3507 utilisé dans la même position par les deux chemins. Par contre, les chemins rouge et vert sont incompatibles car ils utilisent tous deux le routeur 7990 dans des positions différentes.

Deuxièmement, du fait de contraintes thermiques et de contraintes de mesures, chaque transition est limitée en capacité : le nombre de chemins associés doit être inférieur à un nombre maximum.

Troisièmement, du fait de contraintes thermiques locales, chaque zone de la charge utile est soumise à des limitations thermiques : le nombre d'équipements allumés doit rester inférieur à un nombre maximum. Par exemple, seuls deux équipements parmi les amplificateurs 5005, 5006, 5007, 5008 et 5009 de la figure 1, peuvent être utilisés par des chemins sur une transition descendante en température.

Enfin, certains chemins doivent être obligatoirement testés et donc apparaître dans au moins une transition. Pour d'autres chemins, certaines transitions sont interdites.

Les exigences de test portent sur l'ensemble des transitions thermiques. Elles prennent différentes formes.

Les plus courantes imposent de tester un équipement dans au moins une transition thermique, ce qui impose qu'au moins un chemin passant par cet équipement soit testé dans au moins une transition thermique. Par exemple sur la figure 1, l'utilisation du chemin rouge ou du chemin vert permet de satisfaire l'exigence de test du routeur 7990. Au minimum, l'ensemble des équipements de la charge utile doit être testé dans au moins une transition thermique.

D'autres exigences imposent de réaliser un nombre minimum (supérieur à un) de tests de certains équipements.

D'autres imposent de tester un équipement sous certaines conditions thermiques, ce qui limite les transitions utilisables.

D'autres enfin imposent de tester un équipement sous certaines conditions fréquentielles, ce qui limite les chemins utilisables. Par exemple, sur la figure 1, si le routeur 7990 doit être testé en fréquence haute, le chemin vert qui passe par le filtre CH11 de fréquence trop basse ne permet pas de satisfaire cette exigence. Par contre, le chemin rouge qui passe par le filtre CH17 de fréquence haute permet de la satisfaire.

A noter que l'appartenance d'un chemin à une transition permet de satisfaire un ensemble d'exigences portant sur les équipements utilisés par ce chemin.

Le problème d'optimisation des plans de test peut être envisagé selon deux objectifs différents :

- la minimisation du nombre de transitions thermiques utilisées sous contrainte de satisfaction de l'ensemble des exigences de test,
- la maximisation du nombre d'exigences de test satisfaites à nombre de transitions thermiques utilisées fixé.

Comme le nombre de transitions nécessaires est de l'ordre de la dizaine, il est possible d'itérer manuellement sur ce nombre. C'est pourquoi l'étude s'est

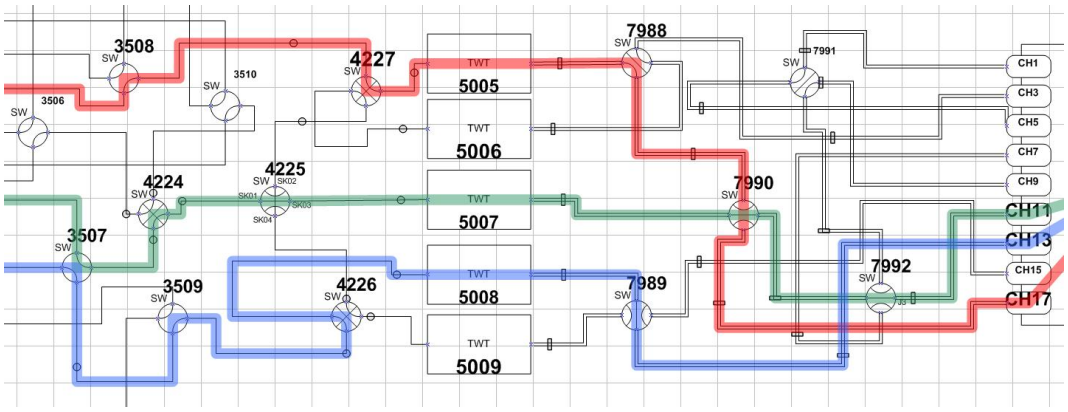

FIGURE 1 – Extrait d'une charge utile de satellite de télécommunication

focalisée sur le second objectif : maximisation du nombre d'exigences de test satisfaites à nombre de transitions fixé.

Les données de ce problème sont les suivantes :

- un nombre de transitions utilisables (de l'ordre de la dizaine),
- un ensemble de chemins utilisables (plusieurs milliers),
- un ensemble d'exigences de test (plusieurs milliers),
- un ensemble de contraintes sur les chemins utilisés :
  - contraintes binaires de compatibilité entre chemins (plusieurs centaines de milliers),
  - contraintes n-aires de capacité et de limitation thermique sur les transitions (plusieurs centaines),
  - contraintes unaires de restriction sur l'affectation des chemins aux transitions (plusieurs centaines).

La sortie du problème est un plan de test qui indique pour chaque chemin la transition dans laquelle il apparaît, éventuellement nulle s'il n'apparaît dans aucune transition. Cette sortie doit être produite en quelques minutes seulement pour répondre au besoin des utilisateurs.

A notre connaissance, ce type d'application n'a pas été traité dans la littérature scientifique. Le problème s'apparente à un problème de sacs à dos multiples et multidimensionnels avec conflits [14, 6]. Les transitions thermiques sont les sacs à dos et les chemins sont les objets. Les conflits sont les contraintes binaires d'incompatibilités entre chemins et les contraintes unaires de restrictions sur l'affectation des chemins. Les dimensions sont les contraintes n-aires de capacité et de limitation thermique des transitions. Par contre classiquement dans les problèmes de sac à dos, les utilités des objets sont supposées être additives. Ce n'est pas le cas dans notre problème : l'utilité d'un chemin est définie par l'ensemble des exigences de test qu'il peut satisfaire. Une exigence de test peut être satisfaite de façon équivalente par plusieurs chemins. Ces utilités ne sont pas additives. C'est pourquoi nous avons créé un modèle mathématique dédié au problème d'optimisation des plans de test des charges utiles de télécommunication.

3 Modèle mathématique

Dans un premier temps, nous avons construit un modèle mathématique en Programmation Par Contraintes (PPC) [5] utilisant CP Optimizer [1]. Ce modèle associe une variable à chaque chemin qui représente la transition dans lequel ce chemin est placé. Son domaine de valeurs est l'ensemble des transitions utilisables auquel on ajoute la valeur 0 pour représenter l'absence de sélection du chemin.

3.1 Données

- Nombre de chemins : P
- Nombre d'équipements : U
- Appartenance des équipements aux chemins :

$$UP[u][p], \quad \forall u \in \{1, \ldots, U\}, \forall p \in \{1, \ldots, P\}$$

- Nombre de transitions thermiques : T
- Capacité des transitions en nombre de chemins :

$$CAP[t], \quad \forall t \in \{1, \ldots, T\}$$

- Compatibilité entre chemins :

$$COMP[p][p'], \quad \forall p, p' \in \{1, \ldots, P\}$$

– Restriction d'affectation des chemins aux transitions :

$$REST[p][t], \quad \forall p \in \{1, \ldots, P\}, \forall t \in \{0, \ldots, T\}$$

– Nombre d'exigences de test : R
– Nombre minimum de chemins à sélectionner pour satisfaire les exigences de test :

$$MIN_R[r], \quad \forall r \in \{1, \ldots, R\}$$

– Appartenance des chemins aux exigences de test :

$$PR[p][r], \quad \forall p \in \{1, \ldots, P\}, \forall r \in \{1, \ldots, R\}$$

– Appartenance des transitions aux exigences de test :

$$TR[t][r], \quad \forall t \in \{1, \ldots, T\}, \forall r \in \{1, \ldots, R\}$$

– Nombre de limitations thermiques : L
– Nombre maximum d'équipements utilisés par transition pour respecter les limitations thermiques :

$$MAX_L[l], \quad \forall l \in \{1, \ldots, L\}$$

– Appartenance des équipements aux limitations thermiques :

$$UL[u][l], \quad \forall u \in \{1, \ldots, U\}, \forall l \in \{1, \ldots, L\}$$

– Appartenance des transitions aux limitations thermiques :

$$TL[t][l], \quad \forall t \in \{1, \ldots, T\}, \forall l \in \{1, \ldots, L\}$$

3.2 Variables

Pour chaque chemin, la transition thermique utilisée ou la valeur 0 pour désigner l'absence de sélection :

$$aff[p] \in \{0, \ldots, T\}, \quad \forall p \in \{1, \ldots, P\}$$

3.3 Expressions

– Pour chaque équipement, le fait qu'il soit utilisé ou pas :

$$\forall u \in \{1, \ldots, U\}, \forall t \in \{1, \ldots, T\},$$
$$usedU[u][t] = \Big(\sum_{p=1}^{P/UP[u][p]=1} (aff[p] = t) \Big) > 0$$

– Pour chaque exigence de test, le nombre de chemins sélectionnés satisfaisant cette exigence :

$$\forall r \in \{1, \ldots, R\},$$
$$testR[r] = \sum_{p=1,t=1}^{\substack{P/PR[p][r]=1,\\ T/TR[t][r]=1}} (aff[p] = t)$$

– Pour chaque exigence de test, son niveau de satisfaction :

$$\forall r \in \{1, \ldots, R\},$$
$$satR[r] = \min(testR[r], MIN_R[r])$$

– Critère à optimiser (somme des niveaux de satisfaction des exigences de test) :

$$obj = \sum_{r=1}^{R} satR[r]$$

3.4 Objectif

maximize obj

3.5 Contraintes

Contraintes unaires Restriction d'affectation des chemins aux transitions :

$$\forall p \in \{1, \ldots, P\}, \forall t \in \{0, \ldots, T\}/REST[p][t] = 0,$$
$$aff[p] \neq t$$

Contraintes binaires Compatibilité entre chemins affectés à une même transition :

$$\forall p, \forall p' \in \{1, \ldots, P\}/C[p][p'] = 0,$$
$$\Big(aff[p] \neq aff[p']\Big) \| \Big((aff[p] = 0) \&\& (aff[p'] = 0)\Big)$$

Contraintes n-aires
– Capacité de chaque transition :

$$\forall t \in \{1, \ldots, T\}, \quad \sum_{p=1}^{P} (aff[p] = t) \leq CAP[t]$$

– Limitations thermiques :

$$\forall l \in \{1, \ldots, L\}, \forall t \in \{1, \ldots, T\}/TL[t][l] = 1,$$
$$\sum_{u=1}^{U/UL[u][l]=1} (usedU[u][t] = 1) \leq MAX_L[r]$$

A l'aide de CP Optimizer [1], ce modèle mathématique a été expérimenté sur des instances correspondant aux satellites en cours de conception. Sur des instances de taille modeste, il permet de produire rapidement des solutions optimales et parfois d'en prouver l'optimalité. Un autre modèle en Programmation Linéaire en Nombre Entiers [11] impliquant des variables binaires a également été expérimenté avec CPLEX [1]. Sur les mêmes instances, il permet plus souvent de prouver l'optimalité au prix d'un temps de résolution nettement plus élevé. Sur des instances de plus

grande taille, ces solveurs génériques (CP Optimizer et CPLEX) se heurtent à des problèmes d'insuffisance de mémoire. Les mêmes problèmes auraient été rencontrés avec des solveurs SAT (nombre astronomique de clauses induites par les contraintes n-aires de capacité et de limitation thermique, fonction exponentielle du nombre de chemins). C'est pourquoi nous avons fait le choix de développer un algorithme spécifique. Pour éviter les problèmes de mémoire, cet algorithme gère les contraintes de façon implicite grâce à des opérations sur des vecteurs de bits. Il réutilise par ailleurs de nombreuses techniques de PPC.

4 Algorithmes

L'algorithme spécifiquement développé par Astrium pour résoudre ce problème d'optimisation est un algorithme de recherche arborescente en profondeur d'abord, comportant :

- un mécanisme de propagation à base de Forward Checking [12],
- un calcul de majorant mettant à jour le domaine de la variable objectif *obj* (voir §3.3),
- un mécanisme de retour-arrière à base de Backtrack chronologique (BT) [3],
- un ordonnancement dynamique des variables,
- un ordonnancement dynamique des valeurs suivant une valeur décroissante de l'impact immédiat sur la valeur minimum de la variable *obj*.

Ces mécanismes ont été ensuite améliorés en ayant recours à :

- des mécanismes de retour-arrière intelligent : Backjumping (BJ) [7], Conflict Directed Backjumping (CBJ) [18, 21] avec production d'explications sur contraintes binaires et n-aires,
- des heuristiques d'ordonnancement de variables dynamiques, spécifiques et adaptatives : apprentissage des impacts des contraintes sur les variables inspiré du weighted degree (wdeg) [4] et du wvar [13], utilisation du dernier conflit [16],
- un mécanisme de redémarrage (Rs) : Restart avec bruitage des heuristiques [9, 22], Restart adaptatif avec utilisation des impacts [10] et Restart adaptatif bruité.

Bien que le mécanisme de propagation à base de Maintien d'Arc Cohérence (MAC) [20] soit généralement très performant, il n'a pas été implémenté dans ce solveur spécifique. A cause de la présence de la valeur 0 dans le domaine de la plupart des variables, toute valeur a un support suivant toute contrainte et le problème initial est arc-cohérent. Pour les mêmes raisons, pratiquement à chaque noeud de l'arbre de recherche le problème est arc-cohérent après filtrage par Forward Checking.

4.1 Mécanismes de retour-arrière

Trois mécanismes de retour-arrière ont été implémentés. Pour chacun de ces mécanismes, on appelle conflit le fait que l'affectation d'une valeur à une variable soit incohérente avec l'affectation courante. Un conflit sur l'affectation a d'une valeur à une variable est détecté soit par la propagation d'une affectation précédente, soit par la propagation de a qui a vidé le domaine d'une autre variable, soit par le fait que a produise un sous-arbre en échec. On appelle échec le fait que toutes les valeurs d'une variable soient en conflit. Quelle que soit la raison de l'échec, il faut remonter dans l'arbre pour débloquer le problème.

Le Backtrack (BT) [3] consiste à revenir sur l'affectation de la variable précédente.

Le Backjumping (BJ), introduit par J. Gaschnig [7], consiste à revenir sur la variable la plus récente impliquée dans le dernier conflit ayant conduit à l'échec. En cas d'échecs imbriqués, le premier retour-arrière est un backjump, les autres sont des backtracks classiques.

Le Conflict Directed Backjumping (CBJ), introduit par P. Prosser [18] et par T. Schiex et G. Verfaillie [21], mémorise l'explication de chaque conflit. Cette explication est constituée de l'ensemble des variables affectées responsables du conflit. Lors de l'échec d'une variable, un backjump est effectué sur la variable la plus récente appartenant aux explications de la variable en échec. Le CBJ permet de multiples backjumps imbriqués.

4.2 Heuristiques d'ordonnancement des variables

4.2.1 Heuristiques d'ordonnancement dynamiques et spécifiques

En PPC, les heuristiques dynamiques d'ordonnancement de variables permettent de choisir la future variable à affecter en fonction des affectations précédentes. Ces heuristiques favorisent les variables menant le plus probablement à un échec. Une heuristique classique reposant sur ce principe d'échec d'abord est $MinDomain$ [12]. Elle choisit dynamiquement la variable ayant la plus petite taille de domaine courant. Cependant, l'existence de la valeur 0 dans le domaine des variables rend cette heuristique peu pertinente car il faut donner la priorité aux variables n'ayant pas 0 dans leur domaine (sélection obligatoire dans une des transitions) par rapport aux variables ayant la valeur 0 (possibilité de non sélection).

Pour les variables n'ayant pas la valeur 0 dans leur domaine, l'heuristique $MinDomain$ est pertinente. Pour les autres, on peut chercher à s'inspirer des heuristiques utilisées dans les problèmes de sac à dos. L'heuristique la plus classique consiste à ordonnan-

cer statiquement les objets suivant une valeur décroissante du ratio entre leur utilité et leur poids [14]. Dans notre problème, il est difficile de définir ce que sont l'utilité et le poids d'un chemin. Pour cette raison, les variables ayant la valeur 0 dans leur domaine sont ordonnancées dynamiquement suivant une valeur décroissante de l'impact immédiat sur le critère : heuristique $MaxObjectif$. Cet impact est la valeur maximale de l'augmentation du niveau de satisfaction des exigences qu'il est possible d'obtenir en affectant une valeur à cette variable.

En s'inspirant de ces principes, une heuristique dynamique et spécifique d'ordonnancement des variables a été implémentée. Trois types de variables sont identifiées et ordonnancées de manière hiérarchique :

1. les variables ayant une seule valeur dans leur domaine sont affectées d'abord,
2. les variables n'ayant pas la valeur 0 dans leur domaine suivant l'heuristique $MinDomain$,
3. les autres variables ayant la valeur 0 dans leur domaine suivant l'heuristique $MaxObjectif$.

Cet ordonnancement a été raffiné afin de prendre en compte la compatibilité initiale des chemins : pour chaque chemin le nombre de chemins avec lesquels il est compatible. Cette information permet de choisir la variable la moins (resp. la plus) compatible avec les autres variables : heuristique $MinCompatible$ (resp. $MaxCompatible$). L'heuristique dynamique et spécifique (HDS) résultante est :

1. les variables ayant une seule valeur dans leur domaine sont affectées d'abord,
2. les variables n'ayant pas la valeur 0 dans leur domaine suivant $MinDomain$,
3. les variables n'ayant pas la valeur 0 dans leur domaine et ayant la même taille de domaine suivant $MinCompatible$,
4. les variables ayant la valeur 0 dans leur domaine suivant $MaxObjectif$,
5. les variables ayant la valeur 0 dans leur domaine et ayant le même impact sur le critère suivant $MaxCompatible$.

4.2.2 Heuristiques d'ordonnancement dynamiques, spécifiques et adaptatives

HDS ne prend pas en compte les contraintes n-aires du problème. Ces contraintes sont gérées de façon implicite pour éviter les problèmes de dépassement de mémoire. De ce fait, il n'est pas envisageable d'utiliser des heuristiques basées sur le degré ou le degré dynamique des variables [2], qui reposent sur des calculs de nombre de contraintes pesant sur les variables et qui favorisent les variables impliquées dans le plus grand nombre de contraintes.

Heuristique basée sur la pondération des variables responsables des échecs Dans un premier temps, nous avons recherché une heuristique permettant de prendre en compte les impacts des contraintes (binaires ou n-aires), telle que le weighted degree (wdeg) [4] et le wvar [13]. Le wdeg et le wvar ont été définis dans le cadre du Forward Checking avec Backtrack. Pour le wdeg, un poids est associé à chaque contrainte. Ce poids est incrémenté chaque fois que la propagation d'une contrainte provoque un échec. Le poids d'une variable est alors défini comme la somme des poids des contraintes actives pesant sur cette variable (contraintes ayant au moins deux variables non affectées). Pour le wvar, un poids est associé à chaque variable. Ce poids est incrémenté chaque fois que la variable est en échec. Ces apprentissages sont utilisés lors de l'ordonnancement dynamique des variables pour pondérer la taille du domaine : les heuristiques $Min\frac{Domain}{wdeg}$ ou $Min\frac{Domain}{wvar}$ remplacent l'heuristique $MinDomain$.

Toujours à cause des problèmes de dépassement de mémoire, il n'est pas envisageable d'utiliser l'heuristique $Min\frac{Domain}{wdeg}$. L'heuristique $Min\frac{Domain}{wvar}$ est moins coûteuse mais ne semble pas assez riche pour prendre en compte les contraintes n-aires. Nous avons donc développé un nouvel apprentissage dénommé weighted culprit variable (wcvar) inspiré du wdeg et du wvar. Comme le wvar, le wcvar associe un poids à chaque variable. Ce poids est incrémenté chaque fois qu'une variable est responsable du dernier conflit ayant provoqué un échec.

Cet apprentissage est utilisé pour l'ordonnancement dynamique, spécifique et adaptatif (HDSA_WCVar) :

- des variables n'ayant pas la valeur 0 dans leur domaine, en favorisant les plus contraintes : $Min\frac{Domain}{wcvar}$,
- des autres variables présentant la valeur 0 dans leur domaine, en favorisant les moins contraintes : $Max\frac{Objectif}{wcvar}$.

Heuristique basée sur le dernier conflit Dans un deuxième temps, une heuristique dynamique, spécifique et adaptative basée sur le dernier conflit (HDSA_LC) a été implémentée. Le raisonnement à partir du dernier conflit [16] consiste, en cas de retour-arrière, à affecter en priorité la dernière variable en échec. L'heuristique HDSA_LC résultante est :

1. la dernière variable en échec est affectée d'abord,
2. les variables ayant une seule valeur dans leur domaine sont affectées ensuite,

3. les variables n'ayant pas la valeur 0 dans leur domaine suivant $MinDomain$,
4. les variables n'ayant pas la valeur 0 dans leur domaine et ayant la même taille de domaine suivant $MinCompatible$,
5. les variables ayant la valeur 0 dans leur domaine suivant $MaxObjectif$,
6. les variables ayant la valeur 0 dans leur domaine et ayant le même impact sur le critère suivant $MaxCompatible$.

Combinaison Cette deuxième heuristique adaptative peut être combinée avec la première. Cette combinaison (HDSA_WCVar_LC) implique l'ordonnancement suivant :

1. la dernière variable en échec est affectée d'abord,
2. les variables ayant une seule valeur dans leur domaine sont affectées ensuite,
3. les variables n'ayant pas la valeur 0 dans leur domaine suivant $Min\frac{Domain}{wcvar}$,
4. les variables n'ayant pas la valeur 0 dans leur domaine et ayant la même taille de domaine suivant $MinCompatible$,
5. les variables ayant la valeur 0 dans leur domaine suivant $Max\frac{Objectif}{wcvar}$,
6. les variables ayant la valeur 0 dans leur domaine et ayant le même impact sur le critère suivant $MaxCompatible$.

4.3 Mécanismes de Restart

Malgré l'utilisation de mécanismes de retour-arrière intelligent et d'heuristiques dynamiques, spécifiques et adaptatives, les premières affectations restent déterminantes du fait du très grand nombre de variables. Les mécanismes de Restart permettent de remettre en cause ces premières affectations. Après un certain nombre de retour-arrière, la recherche est relancée depuis le début. Restart après restart, le nombre de retour-arrières autorisés croît de façon géométrique [22]. Le nombre croissant de retour-arrières autorisés garantit la complétude de l'algorithme. Cependant, il faut s'assurer de ne pas reparcourir la même partie de l'arbre de recherche. Pour cela il suffit de modifier l'ordonnancement dynamique des variables. Plusieurs types de Restart ont été développés.

Le Restart bruité (Rs_Rand) bruite les heuristiques d'ordonnancement [9]. Pour cela, la valeur heuristique des variables est bruitée aléatoirement de plus ou moins un certain pourcentage. En plus d'assurer un parcours d'arbre différent, ce bruitage permet une exploration aléatoire autour de l'heuristique initiale. Il est donc possible de bruiter les quatre heuristiques HDS, HDSA_WCVar, HDSA_LC et HDSA_WCVar_LC pour obtenir quatre Rs_Rand différents. Le HDS_Rs_Rand est un Restart bruité, les trois autres sont des Restarts adaptatifs bruités.

Le Restart adaptatif avec utilisation de l'apprentissage wdeg a été introduit par D. Grimes [10]. Dans son travail, il a montré l'intérêt de modifier l'ordonnancement uniquement grâce à l'apprentissage wdeg. Un Restart adaptatif similaire est proposé sur la base de l'heuristique wcvar (HDSA_WCVar_Rs). HDSA_WCVar_Rs n'utilise aucun mécanisme de bruitage. Seul l'apprentissage wcvar assure un parcours d'arbre différent après chaque restart. Un Restart adaptatif sans bruitage à base d'une combinaison de wcvar et de dernier conflit (HDSA_WCVar_LC_Rs) a également été développé.

L'ensemble de ces mécanismes de retour-arrière, d'ordonnancement et de restart ont été expérimentés sur des instances correspondant aux satellites en cours de conception.

5 Expérimentations

Les expérimentations ont été menées sur cinq charges utiles de télécommunication associées chacune à une base d'exigences, appelées ici SatA, SatB, SatD, SatE et SatF. Afin d'assurer une grande variété d'instances de test, plusieurs dizaines d'instances ont été dérivées aléatoirement de ces instances de base. Pour cela, les exigences de tests prises en compte par chaque instance ont été tirées aléatoirement à partir de chaque base d'exigences. Les instances de test de type SatA et SatB impliquent plusieurs centaines de variables, celles de types SatD, SatE et SatF en comptent plusieurs milliers. Sur ces instances de test, douze algorithmes ont été comparés. Tous sont à base de recherche en profondeur d'abord et de propagation par Forward Checking. Ils se différencient par les mécanismes de retour-arrière (§4.1), les heuristiques (§4.2) et les mécanismes de Restart (§4.3) utilisés. Ces douze algorithmes sont les suivants :

- Backtrack et heuristique dynamique et spécifique (BT_HDS),
- Backjumping et heuristique dynamique et spécifique (BJ_HDS),
- Conflict Directed Backjumping (CBJ) et heuristique dynamique et spécifique (CBJ_HDS).
- CBJ et heuristique dynamique, spécifique et adaptative avec le wcvar (CBJ_HDSA_WCVar),
- CBJ et heuristique dynamique, spécifique et adaptative avec le dernier conflit (CBJ_HDSA_LC),
- CBJ et heuristique dynamique, spécifique et adaptative avec le wcvar et le dernier conflit

(CBJ_HDSA_WCVar_LC),
- CBJ, heuristique dynamique et spécifique et Restart bruité (CBJ_HDS_Rs_Rand),
- CBJ, heuristique dynamique, spécifique et adaptative avec le wcvar et Restart (CBJ_HDSA_WCVar_Rs),
- CBJ, heuristique dynamique, spécifique et adaptative avec le wcvar et Restart bruité (CBJ_HDSA_WCVar_Rs_Rand),
- CBJ, heuristique dynamique et spécifique et adaptative avec le dernier conflit et Restart bruité (CBJ_HDSA_LC_Rs_Rand),
- CBJ, heuristique dynamique et spécifique et adaptative avec le wcvar et le dernier conflit et Restart (CBJ_HDSA_WCVar_LC_Rs),
- CBJ, heuristique dynamique et spécifique et adaptative avec le wcvar et le dernier conflit et Restart bruité (CBJ_HDSA_WCVAR_LC_Rs_Rand).

Ces douze algorithmes ont également été comparés au solveur CP Optimizer d'IBM Ilog [1] pour les instances de type SatA, SatB et SatE n'ayant pas provoqué de dépassement de mémoire. Ce solveur générique a été utilisé en mode :
- profondeur d'abord (CP_Optimizer),
- Restart (CP_Optimizer_Rs).

Pour chaque instance et chaque algorithme, on mesure le nombre d'exigences de test satisfaites du meilleur plan de test produit en temps borné : une minute pour les instances simples de type SatA et SatB, cinq minutes pour les autres. Pour chaque type d'instance et chaque algorithme, une moyenne de ces mesures est réalisée. Ces moyennes sont normalisées entre 0 et 1 pour fournir un indice de qualité. Ces normalisations sont effectuées à partir des algorithmes spécifiques puisque CP Optimizer ne produit pas de résultats pour les instances de type SatD et SatE. Ainsi, le meilleur algorithme a une note de 1 et le plus mauvais de 0. Les indices de qualité des solutions trouvées par CP Optimizer sont également calculés par rapport aux solutions des algorithmes spécifiques. Pour cette raison, il est possible d'avoir un indice supérieur à 1 si la solution trouvée par CP_Optimizer est meilleure que celles produites par les algorithmes spécifiques, inférieur à 0 si elle est moins bonne. Le tableau 1 regroupe les indices de qualités obtenus par les douze algorithmes spécifiques et par le solveur générique pour chaque type d'instance. La figure 2 représente la moyenne de ces indices de qualité sur les cinq types d'instances pour les algorithmes spécifiques.

Résultats des mécanismes de retour-arrière Le tableau 1 montre que le BackJumping (BJ_HDS) permet d'améliorer très légèrement la qualité des solutions trouvées par rapport au Backtrack (BT_HDS). On note jusqu'à $0,43$ de gain d'indice pour les instances de type SatE. Pour la plupart des types d'instances de test, le CBJ (CBJ_HDS) apporte une amélioration importante à la qualité des solutions produites. Jusqu'à 0.63 de gain est obtenu pour les instances de type SatE par rapport au BT_HDS. La figure 2 confirme ces résultats. Bien que le CBJ_HDS soit le plus coûteux de ces mécanismes de retour-arrière, il permet d'obtenir un meilleur indice moyen que BJ_HDS et le BT_HDS pour le même temps d'exécution. Pour cette raison, les heuristiques et les Restarts ont été expérimentés uniquement sur la base du CBJ.

Résultats des mécanismes d'heuristiques d'ordonnancement des variables Les solutions trouvées par l'heuristique adaptative wcvar sans Restart (CBJ_HDSA_WCVar) ne sont pas meilleures que celles obtenues par l'heuristique spécifique (CBJ_HDS) (voir tableau 1 et figure 2), à l'exception d'un léger gain pour les instances de type SatF. L'heuristique dirigé par le dernier conflit (CBJ_HDSA_LC) permet un léger gain, $0,02$, pour les instances de type SatE par rapport à l'heuristique spécifique. L'heuristique adaptative combinée (CBJ_HDSA_WCVar_LC) détériore de près de $0,20$ en moyenne les solutions par rapport à CBJ_HDS. Pour les instances de type SatA, cette heuristique mixte dégrade même les solutions par rapport au BT_HDS. Ces apprentissages ne semblent donc pas être utilisés à bon escient sans remise en cause des premières affectations.

Résultats des mécanismes de Restart Parmi les algorithmes spécifiques, le tableau 1 montre que les meilleurs résultats sont obtenus :
- pour les instances de type SatA, par le Restart adaptatif bruité dirigé par le dernier conflit (CBJ_HDSA_LC_Rs_Rand),
- pour les instances de type SatB, à égalité par le Restart bruité (CBJ_HDS_Rs_Rand) et le Restart adaptatif bruité dirigé par le dernier conflit (CBJ_HDSA_LC_Rs_Rand),
- pour les instances de type SatD, SatE et SatF, par le Restart adaptatif wcvar (CBJ_HDSA_WCVar_Rs).

En moyenne, le Restart adaptatif wcvar bruité (CBJ_HDSA_WCVar_Rs_Rand) et le Restart adaptatif wcvar bruité dirigé par le dernier conflit (CBJ_HDSA_WCVar_LC_Rs_Rand) ne permettent pas d'améliorer les résultats obtenus par le CBJ_HDS (voir figure 2). Ces Restarts dégradent même les résultats obtenus par le BT_HDS pour les instances de type SatA. En moyenne, l'utilisation du dernier conflit (CBJ_HDSA_LC_Rs_Rand ou respecti-

Algorithme	SatA	SatB	SatD	SatE	SatF
BT_HDS	0,76	0	0	0	0,46
BJ_HDS	0,76	0,03	0,01	0,43	0,47
CBJ_HDS	0,81	0,14	0,08	0,63	0,48
CBJ_HDSA_WCVar	0,81	0,09	0,08	0,51	0,49
CBJ_HDSA_LC	0,81	0,14	0,08	0,65	0,48
CBJ_HDSA_WCVar_LC	0	0,07	0,08	0,53	0,49
CBJ_HDS_Rs_Rand	0,93	1	0,53	0,79	0,05
CBJ_HDSA_WCVar_Rs	0,9	0,74	1	1	1
CBJ_HDSA_WCVar_Rs_Rand	0,13	0,71	0,08	0,55	0,15
CBJ_HDSA_LC_Rs_Rand	1	1	0,44	0,86	0
CBJ_HDSA_WCVar_LC_Rs	0,9	0,72	0,97	0,99	0,99
CBJ_HDSA_WCVar_LC_Rs_Rand	0,13	0,71	0,08	0,52	0,14
CP_Optimizer	-23,5	-8,47	NA	NA	-17,94
CP_Optimizer_Rs	0,36	1,26	NA	NA	0,7

Légende :

Les cellules grisées NA correspondent à des tests non applicables dus à des problèmes de dépassement de mémoire.

TABLEAU 1 – Indice de qualité de chaque algorithme pour chaque type d'instances.

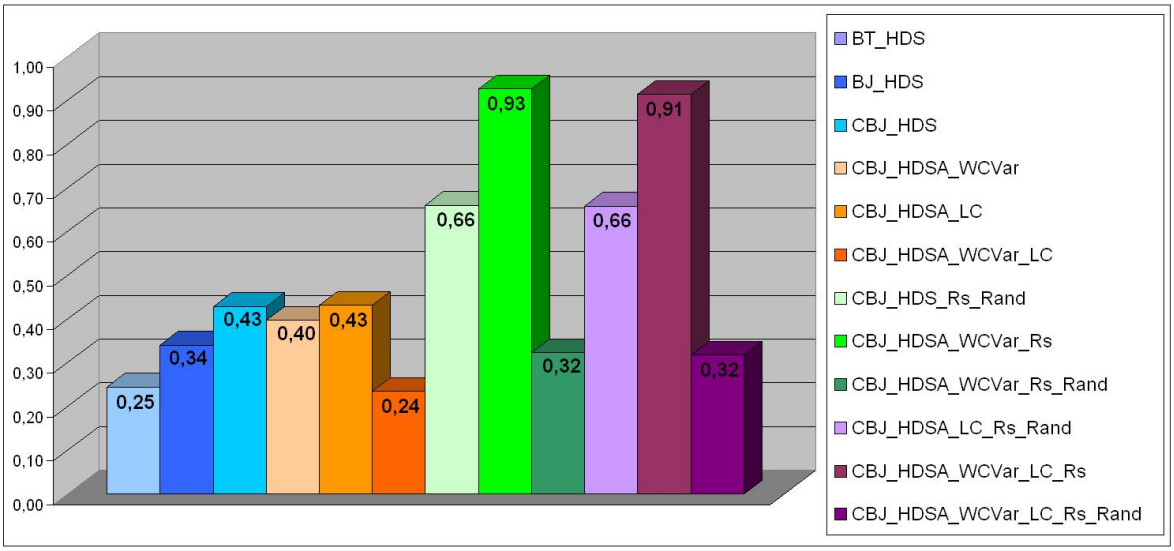

FIGURE 2 – Indice moyen de qualité pour chaque algorithme spécifique.

vement CBJ_HDSA_WCVar_LC_Rs) dégrade le restart bruité (CBJ_HDS_Rs_Rand) ou respectivement le restart adaptatif wcvar (CBJ_HDSA_WCVar_Rs). Bien qu'en combinaison avec le restart bruité (CBJ_HDSA_LC_Rs_Rand), elle permet d'obtenir les meilleurs résultats pour les instances de type SatA et SatB. En moyenne, le Restart adaptatif wcvar (CBJ_HDSA_WCVar_Rs) permet d'obtenir les meilleurs indices de qualité.

Cette analyse permet d'affirmer l'intérêt des mécanismes de CBJ et de Restart pour ce problème d'optimisation des plans de test avec un net avantage pour le Restart adaptatif wcvar. Cet apprentissage wcvar basé sur les variables responsables des échecs, s'avère très fructueux lors de l'utilisation du Restart.

Résultats du solveur générique : CP Optimizer Le solveur générique CP Optimizer d'IBM Ilog [1] a été utilisé à titre indicatif sur les instances de types SatA, SatB et SatF. Il n'a pas pu être utilisé sur les autres instances du fait de problème de dépassement de mémoire. Le tableau 1 montre que CP Optimizer en profondeur d'abord (CP_Optimizer) dégrade énormément l'ensemble des plans de test. Comme nos algorithmes utilisent le même mécanisme de recherche en profondeur d'abord, ce résultat montre l'impact positif du CBJ et des heuristiques spécifiques développés pour ce problème. Le Restart de CP Optimizer (CP_Optimizer_Rs) est plus performant. Il permet de trouver de meilleurs solutions pour les instances de type SatB. Par contre, il est très loin des solutions optimales pour les instances de type SatA, plus de 0.60

d'écart d'indices. Pour obtenir de meilleurs résultats pour les instances de type SatB, il serait intéressant d'implémenter d'autres mécanismes de Restart adaptatif basés sur l'apprentissage de la réduction de l'arbre de recherche. Ce type d'apprentissage proposé par P. Refalo [19] consiste à apprendre l'impact de chaque affectation sur la taille de l'arbre de recherche.

6 Conclusion

Pour ce nouveau domaine applicatif, le modèle mathématique que nous avons proposé permet de représenter le besoin d'optimisation des plans de test des charges utiles des satellites de télécommunication. Cependant, il engendre un très grand nombre de variables (plusieurs milliers) et de contraintes (plusieurs centaines de milliers). Sa résolution par des solveurs génériques peut s'avérer impossible.

Seuls les algorithmes spécifiques implémentés ont permis la résolution de toutes les instances de test. Au sein de ces algorithmes, les mécanismes de CBJ et de Restart se sont révélés bénéfiques. Pour les instances les plus complexes, la nouvelle heuristique wcvar s'est révélée très performante en combinaison avec le Restart. Pour un type d'instances simples, les Restarts mis en place ne sont pas aussi performants que CP Optimizer en mode Restart. Il semblerait intéressant de développer un apprentissage basé sur l'impact des affectations sur la taille de l'arbre de recherche [19].

Ce problème d'optimisation pourrait par ailleurs être traité par des algorithmes à base de recherche locale tels que le Min Conflict [17], la recherche Tabou [8] ou encore le Recuit Simulé [15].

Références

[1] IBM ILOG OPL Development Studio. http ://www-01.ibm.com/software/integration/optimization/opl-dev-studio/.

[2] C. Bessière and J.C. Régin. MAC and Combined Heuristics : Two Reasons to Forsake FC (and CBJ ?) on Hard Problems. In *CP*, pages 61–75, 1996.

[3] J. R. Bitner and E. M. Reingold. Backtrack Programming Techniques. *Commun. ACM*, 18(11) :651–656, 1975.

[4] F. Boussemart, F. Hemery, C. Lecoutre, and L. Sais. Boosting Systematic Search by Weighting Constraints. In *ECAI*, 2004.

[5] R. Dechter. *Constraint Processing*. Morgan Kaufmann Publishers Inc., 2003.

[6] L. Epstein, A. Levin, and R. van Stee. Multidimensional Packing with Conflicts. In *FCT*, pages 288–299, 2007.

[7] J. G. Gaschnig. *Performance measurement and analysis of certain search algorithms*. PhD thesis, 1979.

[8] F. Glover and M. Laguna. Tabu Search. In *Modern Heuristic Techniques for Combinatorial Problems*, pages 70–141. 1993.

[9] C.P. Gomes, B. Selman, and H. Kautz. Boosting combinatorial search through randomization. In *AAAI*, pages 431–437, 1998.

[10] D. Grimes. A Study of Adaptive Restarting Strategies for Solving Constraint Satisfaction Problems. In *AICS*, 2008.

[11] C. Guéret, C. Prins, and M. Sevaux. *Programmation linéaire*. Eyrolles, 2000.

[12] R. M. Haralick and G. L. Elliott. Increasing Tree Search Efficiency for Constraint Satisfaction Problems. *Artificial Intelligence*, 14(3) :263–313, 1980.

[13] W. Karoui. *Méthodes à divergences pour la résolution de problèmes de satisfaction de contraintes et d'optimisation combinatoire*. PhD thesis, 2010.

[14] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems*. Springer, Berlin, Germany, 2004.

[15] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimization by Simulated Annealing. *Science*, 220(4598) :671–680, 1983.

[16] C. Lecoutre, L. Saïs, S. Tabary, and V. Vidal. Reasoning from Last Conflict(s) in Constraint Programming. *AIJ*, 173 :1592–1614, 2009.

[17] S. Minton, M. D. Johnston, A. B. Philips, and P. Laird. Solving large-scale constraint satisfaction and scheduling problems using a heuristic repair method. In *AAAI*, pages 17–24, 1990.

[18] P. Prosser. Hybrid Algorithms for the Constraint Satisfaction Problem. *Computational Intelligence*, 9 :268–299, 1993.

[19] P. Refalo. Impact-Based Search Strategies for Constraint Programming. In *CP*, volume 3258, pages 557–571, 2004.

[20] D. Sabin and E. C. Freuder. Contradicting conventional wisdom in constraint satisfaction. In *ECAI*, pages 125–129, 1994.

[21] T. Schiex and G. Verfaillie. Nogood Recording for Static and Dynamic Constraint Satisfaction Problems. In *ICTAI*, 1993.

[22] T. Walsh. Search in a Small World. In *IJCAI*, pages 1172–1177, 1999.