L'apport des contraintes globales pour la modélisation et la résolution d'applications industrielles

A. Aggoun, N. Beldiceanu, E. Bourreau, H. Simonis
COSYTEC SA, Parc Club Orsay Université
4, rue Jean Rostand, F-91893 Orsay Cedex
cosytec@cosytec.fr

Résumé : L'article se propose de présenter l'apport des contraintes globales de CHIP pour la modélisation et la résolution des problèmes combinatoires rencontrés dans l'industrie. La première partie décrit les contraintes globales : **cumulative** (planification et ordonnancement à capacité finie : contraintes disjonctives et cumulatives), **diffn** (problèmes de placement et d'affectation), **cycle** (problèmes des tournées et réglages des machines) et **sequence** (contraintes réglementaires en planification des ressources humaines). La seconde partie explique les principaux apports : abstraction, puissance expressive, multiple interprétations, méthodes de résolution adaptées, combinaison d'algorithmes, saturation, évolution, relaxation.

Mots clés : Programmation Par Contraintes (PPC), CHIP, Recherche Opérationnelle, Modélisation, Ordonnancement, Affectation, Rotation.

Abstract: This paper summarizes the contribution of CHIP's global constraints in modeling and solving combinatorial problems in the industry. The first part describes global constraints **cumulative** (planning and scheduling with finite capacity: disjunctive and cumulative constraints), **diffn** (placement problems and assignment) and **cycle** (vehicle routing and machine setup) and **sequence** (regulation rules for scheduling human resources). The second part explains the main contributions: abstraction, expressiveness, multiple interpretations, adequate handling methods, algorithms combination, saturation, evolution and relaxation.

Keywords: Constraint Programming (CP), CHIP, Operation Research, Scheduling, Assignment, Rotation.

1. Introduction

L'article présente l'apport des contraintes globales de CHIP pour la modélisation et la résolution des problèmes combinatoires rencontrés dans l'industrie. Nous présentons tout d'abord CHIP en tant que langage de modélisation. Nous discutons les contraintes syntaxiques, contraintes dites de première génération. Puis nous introduisons les contraintes globales (ou sémantiques), dites contraintes de deuxième génération. La dernière partie de l'article est dédiée à l'apport des contraintes globales.

2. Langage de modélisation

Les concepts fondamentaux de la PPC [Jaffar 94] sont : la forme déclarative et relationnelle des contraintes [Colmerauer 90], le niveau d'abstraction des contraintes [Simonis 98a], le non-déterminisme et enfin les domaines de calcul [Dincbas 88, Laurière 78]. Les contraintes sont des relations entre variables. Les variables sont les liens entre les contraintes. Ces liens sont la base des méthodes de propagation des contraintes.

Le non-déterminisme introduit la programmation incrémentale. Les contraintes sont ajoutées une à une. A chaque ajout la contrainte est soit entièrement satisfaite, partiellement satisfaite ou insatisfaite. Ce dernier cas est intéressant, en effet insatisfaction signifie que les conséquences de celle-ci sont en contradiction avec l'ensemble des contraintes existantes déjà dans le système. La contrainte est automatiquement retirée ainsi que ses conséquences sur les domaines des variables. Ce type de programmation, dite programmation incrémentale, a un prix : quelles sont les informations à mémoriser à chaque ajout de contrainte pour pouvoir à tout moment revenir à l'état antérieur ? En revanche, elle libère l'utilisateur de la gestion des ajouts dynamiques des contraintes et des points de choix pendant la phase d'énumération.

2.1 Domaines de calcul

Les domaines de calcul sont inspirés des applications d'aide à la décision. Par exemple dans CHIP les domaines les plus utilisés sont les domaines finis et l'algèbre Rationnelle[Dincbas88]. Il existe d'autres domaines de calcul comme l'algèbre de Boole ou les intervalles. Cet article n'aborde que les domaines finis. Ceux ci permettent de modéliser d'une façon élégante les problèmes combinatoires discrets. Un programme

CHIP comprend trois parties : la déclaration des variables domaines (de décision), la pose incrémentale des contraintes et enfin l'énumération.

2.2 Le concept de domaine

Une variable domaine est une variable qui prend ses valeurs dans un ensemble fini d'entiers. Ces variables sont directement utilisées par les contraintes comme les contraintes arithmétiques sur des termes linéaires ou des inégalités. L'exemple suivant illustre un cas de déclaration de variable domaine. La ligne 1 indique que T11 et T12 sont des variables domaines pouvant prendre des valeurs entre 0 et 100. La ligne 2 exprime une contrainte (arithmétique) de précédence.

[T11, T12] :: 0..100,	% Ligne 1	résultat après la pose de la contrainte et propagation :
T12 # ≥ T11 + 5,	% Ligne 2	*T12 in {5..100}, T11 in {0..95}*

2.3 Les contraintes syntaxiques

Parmi les contraintes syntaxiques [Dincbas 88] on peut citer les contraintes arithmétiques linéaires (≥,≤, >, <, =), la non-égalité (X ≠ Y, X et *Y* ne peuvent pas prendre la même valeur), les contraintes comme alldifferent, element, atleast et atmost.

Contrainte alldifferent : alldifferent(L), L est une liste de variables domaines. Elle contraint ses variables à prendre des valeurs différentes.

Contrainte element : element(N, [V1, …, Vn], C), N et C sont des variables domaines, Vi des entiers. Elle contraint C à être le Nième élément de la liste [V1,…,Vn].

2.4 L'énumération

Le processus d'énumération consiste à définir une stratégie sur l'ordre de choix des variables et des valeurs. CHIP intègre des primitives de choix de variable et de valeur et offre également des moyens à l'utilisateur de programmer facilement ses propres heuristiques.

Choix des variables : L'utilisateur doit trouver un compromis entre des objectifs assez souvent contradictoires, par exemple minimiser la durée du projet (quelle tâche choisir ?) tout en maintenant une certaine équité en ce qui concerne l'utilisation des ressources. Les heuristiques utilisées proviennent des expériences en Recherche Opérationnelle et en Intelligence Artificielle.

Choix des valeurs : Le choix des valeurs dépend des objectifs fixés par le besoin : terminer le plus tôt possible, terminer juste à temps. Par exemple la primitive non-déterministe indomain(V*i*) permet d'énumérer les valeurs de la variable V*i*. Au premier appel elle fixe la variable à la plus petite valeur dans son domaine. En cas de retour arrière, elle essaie la valeur qui suit dans le domaine, etc.

La propagation : c'est un mécanisme [Mackworth 77, Montanari 74] qui provient de l'Intelligence Artificielle et principalement des CSP (Constraint Satisfaction Problems). Ce mécanisme détermine la manière dont les contraintes sont réveillées. Les méthodes utilisées pour résoudre les contraintes sont généralement incomplètes. Quand la contrainte est postée, elle est soit insatisfaite, partiellement ou entièrement résolue. Quand elle est partiellement satisfaite, elle peut de nouveau être appelée par le système si une de ses variables a été modifiée. Les contraintes peuvent être sollicitées plusieurs fois. Les contraintes du problème forment un réseau connecté par les variables. La propagation est sollicitée de manière dynamique lors de l'énumération.

3. Les contraintes globales

La version CHIP V5 repose sur la deuxième génération d'outils de Programmation Par Contraintes : les contraintes globales [Aggoun 93, Beldiceanu 94]. Elles ont largement contribué à la progression de la technologie PPC. Il s'agit de primitives de 'haut niveau' destinées à modéliser des problèmes complexes utilisant des algorithmes de résolution spécifiques. Elles regroupent un ensemble de prétraitements et de propagations appropriés. Certaines contraintes très proches des besoins exprimés dans des applications réelles apparaissent comme étant des conditions 'globales' et sont difficilement exprimables de façon compacte et concise avec des contraintes syntaxiques. Les contraintes globales (*cumulative*, *diffn*, *cycle* et *sequence*) ont permis une avancée technologique pour plusieurs raisons :

- un niveau d'abstraction permettant de modéliser les problèmes d'une façon beaucoup plus concise et proche de la réalité,
- ces contraintes sont génériques et complémentaires,
- les performances - temps de calcul et consommation mémoire - sont très nettement améliorées,
- certains types de problèmes sont aujourd'hui bien résolus [Beldiceanu 96, Caseau 94],
- la taille du code étant extrêmement réduite, la maintenance évolutive de l'application est grandement facilitée. En effet, une ligne d'appel à une contrainte globale remplace plusieurs centaines de lignes de modélisation avec des contraintes syntaxiques.

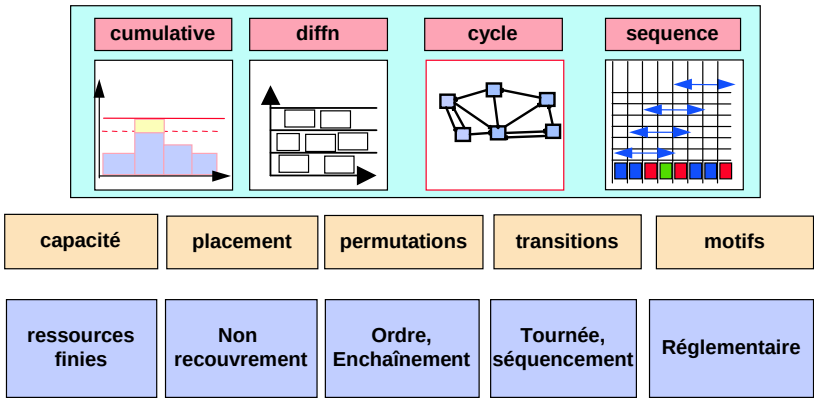


Figure 1 : Les contraintes globales et leurs domaines d'application

3.1 La contrainte Cumulative

La contrainte *cumulative* [Aggoun 93] permet de résoudre des problèmes d'ordonnancement tels que: l'informatique (affectation des tâches aux processeurs), la construction (gestion et suivi d'un projet [Creemers 95]), l'industrie de production (problèmes de gestion d'ateliers [Bellone 92, Bisdorf 95]), l'administration (gestion des emplois du temps [Boizumault 94], rotation et affectation du personnel). Elle exprime le fait qu'à tout instant, le total des ressources utilisées par un ensemble de tâches pour une machine donnée ne dépasse pas une certaine limite *Limite*.

Syntaxe : *cumulative(Débuts, Durées, Ressources, Limite, Fin)*

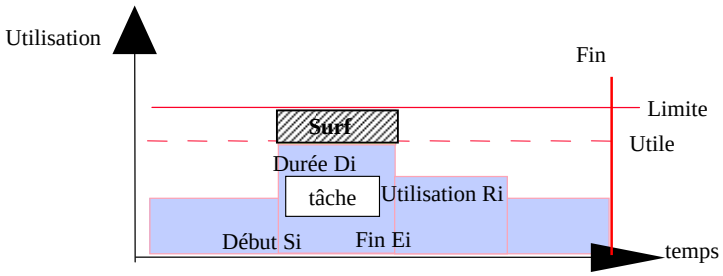


Figure 2 *:* illustration de la contrainte *cumulative*

Débuts est la liste des dates de début S_i des tâches, *Durées* la liste de leurs durées D_i, *Ressources* la liste des quantités de ressource R_i qu'elles utilisent, *Limite* le maximum de ressources disponibles, et *Fin* la date de fin de toutes les tâches.

Utilisation de cumulatives pour les problèmes d'ordonnancement : Prenant l'exemple suivant : Soient 4 projets P1, P2, P3 et P4. Chaque ligne de la table ci-dessous décrit une tâche : le nom du projet, son nom, sa durée et le nombre de personnes requises. La tâche T11 appartient au projet P1, sa durée est de 5 et nécessite 2 personnes. Les contraintes sur les tâches d'un projet Pi sont telles que $\forall j$ Ti,j précède Ti,j+1. Dans l'exemple le nombre de personnes disponibles à tout moment est de 6. Le problème à résoudre consiste à planifier les tâches Ti,j tout en respectant les contraintes de précédence entre tâche dans un projet, le cumul des ressources (personnes) partagées par les tâches afin de ne pas dépasser la disponibilité maximum (ex : 6 personnes) et enfin minimiser les dates de fin des projets.

Projet	Tâche	Durée	Pers.
P1	T11	5	2
P1	T12	7	3
P2	T21	3	4
P2	T22	5	3
P2	T23	4	3
P3	T31	5	2
P4	T41	7	4

```
top:-
  Fin = 100,  Personnel = 6,
  [T11,T12,T21,T22,T23,T31,T41] :: 0..Fin,
  [E1,E2,E3,E4] :: 0..Fin,
  [D11,D12,D21,D22,D23,D31,D41] = [5,7,3,5,4,5,7],
  [U11,U12,U21,U22,U23,U31,U41] = [2,3,4,3,3,2,4],
  T12 #>= T11+D11,
  T22 #>= T21+D21,  T23 #>= T22+D22,
  E1 #= T12+D12,  E2 #= T23+D23,  E3 #= T31+D31,  E4 #= T41+D41,
  Limite :: 0..Personnel,

cumulative([T11,T12,T21,T22,T23,T31,T41],[D11,D12,D21,D22,D23,D31,D41],
            [U11,U12,U21,U22,U23,U31,U41], Limite),
  min_max(labeling([T11,T12,T21,T22,T23,T31,T41]), [E1,E2,E3,E4]).
```

Figure 3 : exemple d'utilisation de *cumulative*

Toutes les tâches Ti,j commencent entre 1 et Fin, les variables Ei représentent les fin des projets Pi, Di les durées respectives (fixes), Ui les besoins en personnel, Limite1 (inconnue) indique le nombre maximum de personne nécessaire pour la complétion des projets. Un exemple de solution optimale est le suivant : Tous les projets se terminent avant la date 19. Les débuts des tâches T11,T12,T21,T22,T23,T31,T41 sont respectivement 0, 5, 0, 3, 8, 12, 12.

3.2 La contrainte Diffn

La contrainte *diffn* permet de résoudre des problèmes de non recouvrement apparaissant dans des applications d'ordonnancement, de découpe ou de placement géométrique. La contrainte *diffn* généralise la contrainte *alldifferent* en imposant qu'un ensemble de segments, rectangles ou parallélépipède ne se coupent pas deux à deux. Dans les problèmes d'ordonnancement, la contrainte *cumulative* permet de calculer (lisser la charge) des plannings en fonction des disponibilités des ressources partagées par les tâches. En revanche, elle ne détermine pas l'affectation : qui fait quoi ? Dans ce contexte, la contrainte *diffn* apporte cette complémentarité en résolvant le problème d'affectation.

Syntaxe: *diffn(Rectangles)* où Rectangles est une liste de m rectangles de dimension n (m, n >= 1), chaque rectangle étant décrit par une liste de longueur 2*n (n ≥ 1) comportant ses origines et ses longueurs sur chacun des n axes. Ainsi, en dimension 2, Rectangles est de la forme : [[$O_{1,1}$, $O_{1,2}$, $L_{1,1}$, $L_{1,2}$], ..., [$O_{m,1}$, $O_{m,2}$, $L_{m,1}$, $L_{m,2}$]] où les $O_{i,j}$ et les $L_{i,j}$ sont des entiers ou des variables domaine.

L'affectation et la contrainte diffn : Reprenons l'exemple donné ci-dessus. En plus du partage du personnel, l'exemple suivant traite aussi de l'affectation des tâches aux machines. Les tâches se partagent toutes les machines. Dans la contrainte *diffn* les tâches d'un projet Pi sont représentées par des rectangles [Tij, Mij, Dij, 1], des objets à 2-dimensions. Tij est la date de début (axe temporel) , Mij est la machine choisie (axe des machines), Dij est la durée (la longueur du rectangle sur l'axe temporel) de la tâche (fonction de Mij) et 1 est la largeur du rectangle (axe des machines).

```
top:-
  Fin = 100, Personnel = 6, Limite :: 0.. Personnel,
 [T11,T12,T21,T22,T23,T31,T41] :: 0..Fin,
 [D11,D12,D21,D22,D23,D31,D41] = 1..Fin,
 [E1,E2,E3,E4] :: 0..Fin,
 [U11,U12,U21,U22,U23,U31,U41] = [2,3,4,3,3,2,4],
 [M11,M12,M21,M22,M23,M31,M41] :: 1..3,
 T12 #>= T11+D11, T22 #>= T21+D21, T23 #>= T22+D22,
 E1 #= T12+D12, E2 #= T23+D23, E3 #= T31+D31, E4 #= T41+D41,
 diffn([[T11,M11,D11,1],[T12,M12,D12,1],
 [T21,M21,D21,1],[T22,M22,D22,1],[T23,M23,D23,1],
 [T31,M31,D31,1],[T41,M41,D41,1]]),
 cumulative([T11,T12,T21,T22,T23,T31,T41],[D11,D12,D21,D22,D23,D31,D41],
             [U11,U12,U21,U22,U23,U31,U41], _, _, Limite),
 min_max(labeling([T11,T12,T21,T22,T23,T31,T41,M11,M12,M21,M22,M23,M31,M41]),
          [E1,E2,E3,E4]).
```

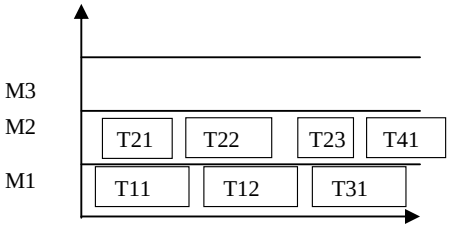


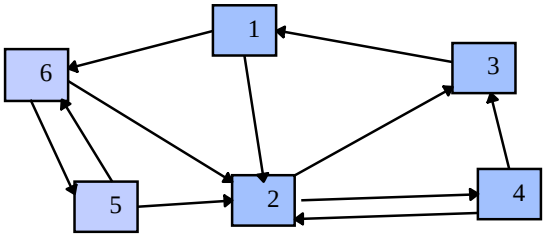
Figure 4 : illustration de la contrainte *diffn*

3.3 La contrainte Cycle

La contrainte *cycle* est adaptée aux problèmes de calcul de tournées, de rotations et de séquencement. Sans cette contrainte, il n'était guère possible d'aborder cette classe de problèmes. Elle permet d'exprimer directement le nombre de tournées à calculer, les contraintes de poids sur chaque tournée, les incompatibilités entre des sites et les fenêtres de temps de visite par site. Elle est complémentaire de la contrainte *cumulative* dans les applications d'ordonnancement où elle permet d'exprimer des contraintes de délais dues à des changements d'outils.

Syntaxe : cycle(Nb, Xs).

La contrainte cycle possède deux paramètres. le premier (Nb) contraint le nombre de circuits devant recouvrir le graphe. Le second (Xs) est le graphe donné sous la forme d'une liste de variables domaines, chaque variable représentant un nœud du graphe, chaque valeur dans le domaine des variables représentant l'arc exprimant la relation successeur. Par exemple le nœud 1 est modélisé par la variable X1 : : [2, 6], 2 signifie que X2 est un successeur potentiel de X1, de même pour X6. En effet la valeur de X1 indique le successeur du nœud 1.



Nb :: 1..6,
X1 :: [2, 6], X2 :: [3, 4], X3 :: [1],
X4 :: [2, 3], X5 :: [2, 6], X6 :: [2, 5],
cycle(Nb,[X1, X2, X3, X4, X5, X6]).

Exemples de solution:
cycle(2,[2,4,1,3,6,5])
cycle(1,[6,4,1,3,2,5])

Figure 5 : illustration de la contrainte *cycle*

De même contraindre le nombre de circuits apporte des informations permettant de fixer toutes les autres variables de la contrainte. Il existe plusieurs variantes de cette contrainte permettant d'exprimer des poids, sur les nœuds et sur les arcs, des poids minimum et maximum sur les circuits formés, des nœuds incompatibles (appartenant à des cycle différents). et des fenêtres temporelles. Pour des exemples montrant d'une part l'utilisation de cycle dans le transport, la minimisation des réglages nous vous recommandons les articles [Simonis 98a].

3.4 La contrainte *sequence*

La contrainte *sequence* est adaptée pour modéliser les contraintes réglementaires comme par exemple les contraintes légales de gestion des ressources humaines (obligation d'un jour de repos après chaque période de 5 journées consécutives travaillées) [Chan 98].

Syntaxe : sequence(Liste, Motifs).

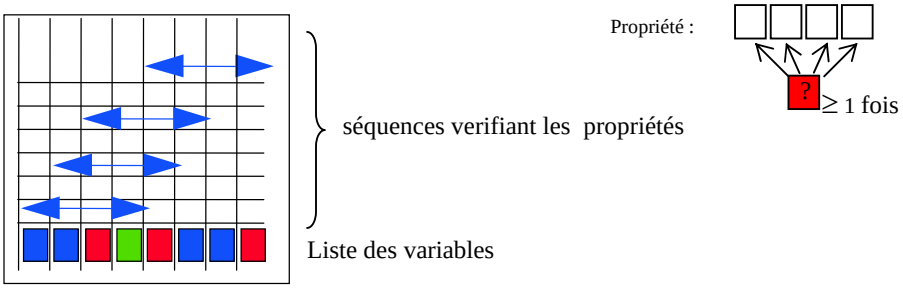


Figure 6 : illustration de la contrainte *sequence*

Cette contrainte généralise la contrainte *among***.** Cette dernière contraint le nombre d'occurrences d'une valeur dans une liste de variables. *Sequence* impose à une liste de variables de respecter certaines propriétés définies à l'aide de motifs. Les différents motifs sont définis à partir d'une grammaire.

4. Apport des contraintes globales

Cette section discute les principaux apports des contraintes globales pour la modélisation et la résolution d'applications industrielles.

Abstraction et Puissance d'expression : Les contraintes globales offrent un niveau assez élevé d'abstraction. Elles diminuent la distance entre le problème traité et les primitives du langage [Simonis 96]. L'idée est de concilier la concision et la généralité offerte par les langages de programmation (logique, fonctionnel, objet) avec l'efficacité des algorithmes ad-hoc. Les contraintes offrent une description compacte et concise des problèmes. Les contraintes étant très proches du discours de l'utilisateur, le partage de ressource devient synonyme de *cumulative* ou *diffn*, les tournées et changement d'outils sont synonymes de *cycle.*

Multiple interprétations : Afin d'augmenter le pouvoir expressif des contraintes, tous leurs paramètres sont des variables domaines. Par exemple, dans la contrainte cumulative la fin générale, la hauteur maximale d'utilisation des ressources, les variables débuts-durées-ressources des tâches sont des variables. Les paramètres sous forme de variables domaines et la combinaison de ces paramètres permettent différentes interprétations de la contrainte. Par exemple la contrainte *cycle* s'interprète comme une permutation, une couverture du graphe par des circuits ou une affectation dans un graphe biparti.

Méthodes de résolution : Bien que l'approche directe consistant à reprendre un ensemble d'algorithmes et à leur donner une interface contrainte peut s'avérer efficace, elle présente les inconvénients de l'approche algorithmique (manque de généricité, difficulté de combinaison). Certes l'implantation des contraintes globales tire profit du savoir-faire en RO et en MD. L'exploitation de ce savoir-faire nécessite des adaptations et des améliorations pour tenir compte de l'incrémentalité. L'implantation des contraintes globales soulève la recherche de conditions nécessaires pour satisfaire ces besoins ainsi que des nouveaux algorithmes pour tester ces conditions, ce qui contribue à l'enrichissement des techniques de résolution des problèmes combinatoires.

Combinaison d'algorithmes : Les contraintes globales sont un exemple parfait de collaboration de plusieurs techniques de résolution (algorithmes) provenant des Mathématiques Discrètes (MD) ou de la Recherche Opérationnelle (RO) [Pinson 88, Jacquet-Lagrèze 98]. L'objectif étant que ces techniques doivent tenir compte de plusieurs paramètres telles que la sémantique de la combinaison de ses différents paramètres, l'incrémentalité et la performance. La collaboration de plusieurs techniques engendre diverses déductions utiles à l'ensemble des contraintes.

Saturation incrémentale : L'utilisation combinée de ces déductions augmentée des combinaisons d'algorithmes est exploitée par la contrainte courante. L'ensemble des variables touchées par ces déductions alimente également l'algorithme de propagation de contraintes. Les contraintes impactées par les variables touchées sont de nouveau réveillées afin de bénéficier de cet apport d'information. Ce procédé de propagation des informations sur les contraintes est répété jusqu'à saturation (impossibilité de faire une nouvelle déduction).

Evolution fonctionnelle : Prenons l'exemple de la contrainte cycle. A l'origine cette contrainte était définie par l'unique paramètre qu'est la liste des nœuds du graphe : cycle(Xs). L'interprétation de cette contrainte se résume à vérifier l'existence d'un circuit hamiltonien. La généralisation est cycle(Nb, Xs) dans laquelle Nb est une variable domaine indiquant le nombre de circuits. Cette extension a permis d'appréhender d'autres problèmes.

Evolution technique : Elle concerne l'amélioration des algorithmes mis en œuvre pour résoudre les contraintes. Ces améliorations proviennent soit des nouvelles découvertes sur des cas similaires ou des cas nouveaux. Ces résultats sont facilement intégrables dans les solveurs sans conséquence pour les utilisateurs (aucune modification du code). L'intégration de l'arbitrage des disjonctions permet de mieux prendre en compte certains cas de contraintes cumulatives. La détection de ces cas peut se faire soit à la pose de la contrainte soit dynamiquement pendant la phase de propagation. L'ajout ou la combinaison des paramètres peut engendre des particularités de cas non encore étudiés ou bien des structures de problèmes pour lesquels il existe des méthodes spécifiques très performantes.

Evolution de l'application : L'application développée en CHIP avec les contraintes globales peut tirer profit des évolutions techniques et fonctionnelles décrites ci-dessus. De nos jours les applications d'aide à la décision évoluent régulièrement et de nouveaux besoins apparaissent : acquisition de nouvelles machines

plus performantes, nouveaux produits, méthodes de travail, flexibilité du temps de travail, nouvelle stratégie commerciale, etc. L'utilisation des contraintes globales rend le code concis ; en conséquence la taille du code devient faible, facile à maîtriser où à faire évoluer. En général la partie résolution dans une application est de l'ordre de quelques pages. L'apport des contraintes globales dans le domaine de la maintenance et évolutivité est considérable.

Relaxation : La relaxation intervient principalement dans les problèmes sur contraints. Par exemple un aléa comme la panne d'une ressource peut déstabiliser la production : des commandes annulées ou des retards dans les livraisons. Le planificateur dispose de plusieurs leviers : annuler ou retarder certaines commandes, prévoir des ressources supplémentaires, etc. Ces solutions peuvent engendrer des coûts si elles ne sont pas justifiées. La relaxation consiste à éviter les échecs d'une part pendant la pose des contraintes et d'autre part en amont de la phase d'énumération. L'idée est d'intégrer directement dans les contraintes globales des moyens modélisant la relaxation. Par exemple dans *Cumulative* l'utilisateur peut spécifier le nombre maximum de ressources. Il peut également contraindre les dépassements de ces ressources. Cet exemple est illustré par la surface hachurée de la figure 2. Il n'est pas concevable qu'une application pratique ne retourne pas de solution.

5. Les applications

ATLAS (*Prix de la Meilleure Application IA'94*) est un système d'ordonnancement pour le contrôle de plusieurs lignes de fabrication, conditionnement et palettisation d'herbicides [Simonis 95].

FORWARD-C (TECHNIP) traite de la planification de la production des raffineries. FORWARD-C permet de gérer au jour le jour le processus complexe de la production d'une raffinerie depuis l'arrivée du brut jusqu'à la livraison des produits finis.

EVA est un système de planification des transports de combustibles nucléaires usés au départ des centrales EDF(Electricité De France). Le but du système est d'établir le planning sur six mois des ressources nécessaires à l'acheminement vers les ateliers des usines du combustible usé produit par les réacteurs d'EDF. Environ deux cents évacuations sont ainsi effectuées annuellement.

TACT est un système d'aide à la décision dans le domaine de la logistique d'approvisionnement. Développé par COSYTEC pour l'un des plus grands producteurs de viande de volaille de Grande Bretagne. TACT optimise l'utilisation de toutes les ressources intervenant dans le processus d'approvisionnement de ses usines. La nature hautement périssable des biens à transporter, le nombre important de sources d'approvisionnement, la diversité des ressources humaines et non humaines utilisées font la difficulté de ce problème. TACT est capable de fournir une solution hebdomadaire en quelques minutes impliquant plusieurs milliers de tâches et plus de cent ressources [Simonis 98b].

GYMNASTE est un progiciel de planning des services infirmiers développé en partenariat par COSYTEC et UJF-PRAXIM [Chan 98]. Pour chaque période de temps : (1) affecte une ressource individuelle à un horaire, un poste, un cycle ou une rotation; (2) respecte les contraintes individuelles, le desiderata du personnel et les règles de gestion; (3) assure la satisfaction de besoins. Il permet de faire de la planification prévisionnelle et de la planification réactive.

6. Conclusion

Cet article a montré l'apport des contraintes globales tant au niveau modélisation que résolution. Elles ont contribué à la résolution des problèmes industriels complexes comprenant plusieurs types de contraintes. La contribution des contraintes globales en tant que langage de modélisation est significative. Cette maturité à conduit CHIP à devenir un langage de modélisation utilisé aussi par des ingénieurs possédant une connaissance métier. Elles ont permis à des non- spécialistes en optimisation d'utiliser la PPC pour s'attaquer à des problèmes d'aide à la décision dans de nombreux domaines.

Références :

Aggoun, A. Beldiceanu, N 1993. Extending CHIP in Order to Solve Complex Scheduling Problems, Journal of Mathematical and Computer Modeling, Vol. 17, No. 7, pages 57-73, Pergamon Press, 1993.

Beldiceanu, N. Contejean, E. 1994, Introducing Global Constraints in CHIP, Journal of Mathematical and Computer Modeling, Vol 20, No 12, pp 97-123, 1994.

Beldiceanu, N. Bourreau, D.Rivreau, D. Simonis, H. 1996. Solving Resource-Constrained Project Scheduling Problems with CHIP, Fifth International Workshop on Project Management and Scheduling, Poznan, Poland, April 1996.

Bellone, J. Chamard, A. Pradelles, C. 1992. PLANE -An Evolutive Planning System for Aircraft Production. First International Conference on the Practical Application of Prolog. 1-3 April 1992, London.

Bisdorff, R. Laurent, S. Pichon, E. 1995. Knowledge Engineering with CHIP - Application to a Production Scheduling Problem in the Wire-Drawing Industry PAP95, Paris, April 1995.

Boizumault, P. Delon, Y. Peridy, L. 1994. Planning Exams Using Constraint Logic Programming, 2nd Conf Practical Applications of Prolog, London, April 1994.

Caseau, Y. Laburthe, F. 1994. Improved CLP Scheduling with Task Intervals, Proc 11th ICLP 1994, Italy, June 1994. MIT Press.

Chan, P. Heus, K. Weil, G. 1998. Nurse Scheduling with Global Constraints in CHIP : GYMNASTE, PAPPACT 98, London.

Colmerauer, A. 1990. An Introduction to Prolog III, Communications of the ACM 33(7), 52-68, July 1990.

Creemers, T. Giralt, L.R. Riera, J. Ferrarons, C. Rocca, J. Corbella, X. 1195. Constrained-Based Maintenance Scheduling on an Electric Power-Distribution Network, PAP95, Paris, April 1995.

Dincbas, M. Van Hentenryck, P. Simonis, H. Aggoun, A. Graf, , T. Berthier, F. 1988. The Constraint Logic Programming Language CHIP. In Proceedings of the International Conference on Fifth Generation Computer Systems (FGCS'88), pages 693-702, Tokyo, 1988.

Jaffar, J. Maher, M. 1994. Constraint Logic Programming: A Survey. Journal of Logic Programming, 19/20:503-581, 1994

Jacquet-Lagrèze, E. 1998, Hybrid Methods for Large Scale Optimization Problems : an OR perspective, PAPPACT98, London.

Lauriere, J. 1978. A Language and a Program for Stating and Solving Combinatorial Problems, Artificial Intelligence, 10, 29-127, 1978

Mackworth, A. K. 1977. "Consistency in networks of relations", Artificial Intelligence 8, 1977.

Montanari, U. 1974. "Networks of constraints: Fundamental properties and applications to picture processing", Information Science 7(2) pp 95-132, 1974.

Pinson, E. 1988. Le Probléme de Job Shop, Thèse de Doctorat de Univ Paris VI, 1988.

Simonis, H. Cornelissens, T. 1995. Modeling Producer/Consumer Constraints. Proc. Principles and Practice of Constraint Programming, Cassis, France, September 1995.

Simonis, H. 1996. A Problem Classification Scheme for Finite Domain Constraint Solving, Proc workshop on constraint applications, CP96,Boston, August 1996.

Simonis, H. 1998a. Standard Models 2 for Finite Domain Constraint Solving, PAPPACT98, London.

Simonis, H. Charlier, P. Kay, P. 1998b. An integrated Transportation Problem solved with CHIP, PAPPACT98, London.

Sommaire :

A. Aggoun, N. Beldiceanu, E. Bourreau, H. Simonis

L’apport des contraintes globales pour la modélisation et la résolution d’applications industrielles,

FRANCORO II, Deuxièmes Journées Francophones de Recherche Opérationnelle,
Sousse, Tunisie, 6-8 Avril 1998.