8 Latif : résolveur d'une hiérarchie de contraintes à sorties multiples utilisant plusieurs critères.

Ce chapitre décrit une autre partie de notre travail. On s'intéresse à la coopération de différents résolveurs afin de résoudre une hiérarchie ayant les contraintes de son dernier niveau associées à un comparateur global et celles des niveaux supérieurs associées à un critère local. Une des approches prometteuses est celle qui utilise les techniques de la propagation locale pour partitionner un système de contraintes en des sous-ensembles de contraintes qui sont résolus par ces différents résolveurs.

Dans ce chapitre, on s'intéresse au cas où il est nécessaire de résoudre les contraintes simultanément (c.à.d. contraintes formant un cycle et/ou un conflit). On s'intéresse également au cas où les modes de combinaison peuvent varier selon les niveaux. Par exemple, dans une hiérarchie contenant trois niveaux d'importance, on peut utiliser pour les deux premiers niveaux un critère local pour lequel une solution sera meilleure qu'une autre si elle satisfait un sur-ensemble des contraintes satisfaites par la seconde. Pour le troisième niveau, un comparateur global plus fin, comme la somme des carrés des erreurs, qui impose un ordre total sur les configurations, peut être utilisé pour départager les solutions. Ce troisième niveau, utilisant des erreurs numériques, ne peut plus être résolu par propagation locale.

Des travaux précédents sur ce problème avaient abouti à la réalisation d'un algorithme (nommé LSCS [HKS+94]) pour la résolution d'une hiérarchie de contraintes dont les méthodes ne possèdent qu'une seule variable de sortie. Les hiérarchies manipulées par l'algorithme initial utilisé consistent en des hiérarchies ayant au dernier niveau des contraintes associées au comparateur global *Moindres-Carrés-Métrique*, et aux niveaux précédents, des contraintes associées au comparateur *Localement-Prédicat-Meilleur*.

Notre recherche s'est orientée vers la généralisation de cet algorithme. Cette généralisation présente des nouvelles définitions et une extension du mécanisme initial utilisé dans LSCS, ceci dans le but de pouvoir prendre en compte des contraintes possédant des méthodes recalculant plusieurs variables à la fois, ainsi que de piloter différents résolveurs spécifiques aux critères utilisés.

Ce chapitre est composé de quatre parties dont chacune décrit en parallèle l'existant et notre apport pour la généralisation[1]. Dans la première partie, on montre *via* des exemples le besoin de résoudre les contraintes simultanément ainsi que celui de résoudre les contraintes ayant des méthodes possédant plusieurs variables de sortie. On finira cette première partie par une vue générale sur le résolveur que nous proposons (nommé Latif).

La deuxième partie décrit des outils de construction d'un graphe admissible. La troisième partie décrit le processus à utiliser pour obtenir un graphe solution à partir d'un graphe admissible. Cette partie décrit également un exemple pour bien illustrer ce mécanisme. La quatrième partie décrit l'algorithme généralisé pour la résolution d'une hiérarchie lors de l'ajout ou du retrait d'une contrainte.

8.1 Motivations et vue générale sur Latif

8.1.1 Cycles et conflit

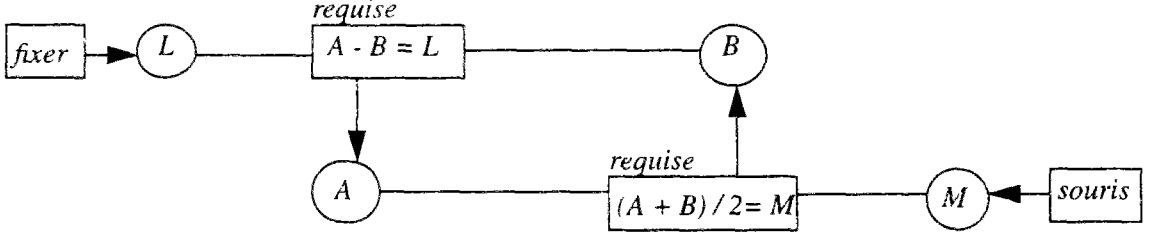
L'algorithme de la propagation locale est très efficace pour des hiérarchies utilisant un critère de comparaison basé sur l'erreur prédicat et ne formant pas de circuits de méthodes des contraintes qui la compose. Malheureusement, beaucoup de hiérarchies du monde réel contiennent des contraintes où leurs méthodes forment des circuits. Ces hiérarchies utilisent aussi des critères basés sur l'erreur métrique. Ces deux cas ne peuvent pas être résolus par la propagation locale. Par exemple, considérons le système de contraintes de la figure 46. Ce système représente une situation typique où le point-milieu *M* d'un segment ayant pour extrémités les points *A* et *B* est déplacé avec une souris. Le point *M* est marqué par la notation *écriture-seulement* et *L* (qui est la longueur du segment *[A,B]*) est marqué par la notation *lecture-seulement* (car on souhaite que cette longueur reste fixe). Pour avoir une solution à ce problème, on est ramené à résoudre un circuit de méthodes. La figure 47 illustre ce fait.

FIGURE 46 : Modélisation par contraintes du déplacement d'un segment par la souris

requise A - B = L?

requise (A + B) / 2 =M!

FIGURE 47 : Graphe de méthodes des contraintes du déplacement d'un segment



Un autre exemple de hiérarchie ne pouvant pas être résolue par la propagation locale est celle de la figure 48 où le comparateur *Moindres-Carrés* est utilisé. Le graphe solution de cette hiérarchie contient un conflit comme le montre la figure 49. Puisque le comparateur *Moindres-Carrés* utilise l'erreur métrique, la solution à cette hiérarchie est $x = 2$.

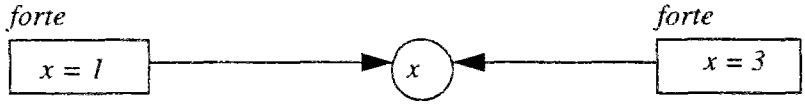
1. Pour des raisons de compréhension, nous avons préféré faire une présentation en parallèle de ce qui existe et de la généralisation.

FIGURE 48 : *Hiérarchie de contraintes conflictuelles.*

forte x = 1

forte x = 3

FIGURE 49 : *Graphe de méthodes d'une hiérarchie de contraintes conflictuelles*



D'une manière générale :

- lors de l'utilisation d'un comparateur basé sur l'erreur métrique, les méthodes des contraintes formant un ou plusieurs conflits doivent être résolues ensemble.
- lors de l'utilisation de n'importe quel critère, si les méthodes des contraintes forment un circuit, alors ces méthodes doivent être résolues ensemble.

8.1.2 Le besoin de considérer des contraintes ayant des méthodes multi-sorties.

Une méthode d'une contrainte est dite multi-sorties si elle détermine plusieurs variables. L'utilisation de ce type de contrainte est considérée pour la résolution des problèmes de cycles. Elle est considérée aussi pour une meilleure modélisation de certaines applications comme par exemples TRIp et TRIpII [KK91,SSA91] qui sont deux applications d'interfaces graphiques. Plusieurs articles de la littérature présentent les avantages et la facilité d'utilisation des contraintes ayant plusieurs variables de sortie [ROS94, San94, Hil93].

Généralement, la modélisation par des contraintes ayant des méthodes multi-sorties a pour but de donner un aspect élégant pour l'expressivité des contraintes du problème traité. Par exemple, considérons les variables x et y qui représentent les coordonnées cartésiennes d'un point, et les variables ρ et θ qui représentent les coordonnées polaires de ce même point. Pour garder ces deux représentations consistantes, on peut définir une contrainte possédant deux méthodes à deux-sorties qui sont m_{1c1} dans un sens et m_{2c1} dans l'autre sens définies dans la figure 50.

FIGURE 50 : *Relation entre coordonnées polaires et cartésiennes d'un point.*

$m_{1c1} = (x,y) \leftarrow (\rho \cos\theta, \rho \sin\theta)$.

$m_{2c1} = (\rho,\theta) \leftarrow \left(\sqrt{x^2 + y^2}, \arctan(y,x)\right)$.

Les méthodes multi-sorties sont aussi utilisées pour accéder aux éléments des structures de données composées. Par exemple, pour décomposer l'objet composé pointCartesien en deux variables, on utilise la contrainte contenant les deux méthodes m_{1c1} et m_{2c1} définis dans la figure 51.

FIGURE 51 : *Décomposition d'un objet composé.*

(x, y)← (point . x, point . y).

point ← Créerpoint(x, y).

8.1.3 Vue générale sur Latif

L'algorithme que nous allons présenter dans ce chapitre est une extension de LSCS présenté dans [HKS+94]. Notre algorithme résout une hiérarchie de contraintes multi-directionnelles et multi-sorties, et de plus, manipule différents types de contraintes.

On s'intéresse au cas où les modes de combinaison peuvent varier selon les niveaux. Par exemple, dans une hiérarchie contenant *n* niveaux d'importance, on peut utiliser pour les *n-1* premiers niveaux un critère local, et pour le $n^{iéme}$ niveau un critère global plus fin qui impose un ordre total sur les configurations pour départager les solutions. Partant d'ici, les hiérarchies pouvant être traitées par Latif sont celles ayant à leur dernier niveau des contraintes associées au comparateur global *Moindre-Carrés-Métrique* ou au comparateur global *Pire-Cas-Métrique* et celles des niveaux supérieurs sont associées à un comparateur local.

Latif peut être vu comme un système de pilotage de résolveurs spécifiques pour le traitement d'une hiérarchie contenant plusieurs types de contraintes. Chacun de ces résolveurs est basé sur un comparateur bien déterminé.

Pour résoudre une hiérarchie de contraintes, cet algorithme prend en compte le fait que certaines des contraintes de cette hiérarchie doivent être résolues ensemble pour former des cellules de contraintes. Dans ce cas, on parle aussi de subdivision de l'ensemble des contraintes de la hiérarchie en des sous ensembles de contraintes qui doivent être résolus séparément. On pourrait tout résoudre par une optimisation globale, mais ici, on espère qu'en traitant séparément chaque cellule, on arrivera à réduire la complexité de la résolution.

L'ensemble des cellules formé par cet algorithme ne doit pas contenir de dépendance cyclique entre les cellules. Donc, on doit définir un ordre partiel entre les cellules de cet ensemble qui permette de propager les valeurs des variables d'une cellule à une autre.

Chacune de ces cellules formées est résolue par un ou plusieurs résolveurs selon les types de contraintes qui la composent. Les valeurs des variables ainsi obtenues à la suite de cette résolution sont propagées à d'autres cellules de contraintes non encore résolues. Cette propagation se fait par l'algorithme de propagation locale.

Il s'agit d'un résolveur incrémental : les contraintes peuvent être ajoutées ou retranchées du système. A chaque ajout ou retrait de contrainte, un nouveau graphe solution de cellules de contraintes est formé et la résolution locale de ces cellules est exécutée. La formation d'une cellule est soumise à des conditions qui favorisent la planification d'un graphe admissible. Des conditions supplémentaires sont ajoutées pour générer un graphe solution à partir du graphe admissible. Ce graphe solution obtenu permet de trouver les solutions *S* après résolution des cellules qui le composent.

Les graphes admissibles et les graphes solutions créés par cet algorithme ont une sémantique légèrement différente de celle des graphes admissibles et des graphes solutions obtenus par les autres résolveurs présentés dans les autres chapitres (il est à rappeler que ces résolveurs sont des algorithmes de propagation locale et que chacun d'eux implémente un critère local ou global utilisant l'erreur prédicat). Cette différence se concrétise par les faits suivants :

un graphe admissible créé par les algorithmes de propagation locale est un graphe qui contient toutes les contraintes requises actives (satisfaites), tandis qu'un graphe admissible créé par Latif est un graphe de cellules contenant toutes les contraintes de la hiérarchie.

un graphe solution créé par les autres résolveurs est un graphe, qui respecte le critère local ou le critère global utilisé, tandis qu'un graphe solution créé par Latif est un graphe qui respecte, le critère global utilisé dans le dernier niveau de la hiérarchie, et le critère local utilisé pour les niveaux supérieurs de la hiérarchie.

Il faut signaler que cet algorithme suppose que chaque contrainte portant sur un ensemble de variables de taille n et ayant m variables en sortie ($n>m$) possède un ensemble de méthodes de taille C_n^m (c.à.d. n'importe quelles m variables de cet ensemble peuvent être déterminées par les n-m variables qui restent[1]).

La résolution d'une cellule de contraintes peut nécessiter un appel à un ou plusieurs résolveurs si cette dernière contient plusieurs types de contraintes. Par exemple, si l'on considère une hiérarchie à trois niveaux *(fort, moyenne, faible)* de contraintes, que les deux premiers niveaux sont associés au comparateur *Localement-Prédicat-Meilleur* (τ_{LPM}), et que le troisième niveau est associé au comparateur *Pire-Cas-Métrique* (τ_{PCM}), alors une cellule du graphe solution peut contenir ces deux types de contraintes. La résolution de cette cellule consiste à appeler un résolveur spécifique sachant résoudre les deux types de contraintes.

8.2 Cellules de contraintes et graphe admissible

Un algorithme de propagation locale échoue à résoudre un graphe de méthodes de contraintes contenant des circuits (ou des conflits si l'erreur métrique est utilisée), puisque tout simplement la technique de propagation locale n'est pas conçue pour cet effet. Pour considérer ce problème, une nouvelle définition de graphe admissible de contraintes uni-sortie a été proposée dans [HKS+94]. Avant de présenter cette définition, on présente d'abord celles qui modélisent un graphe de contraintes et une cellule de ce graphe de contraintes.

8.2.1 Graphe

Définition 8.1

Soit une hiérarchie de contraintes $H=(V,C)$. Le graphe biparti $G=(V, C, E)$ (V et C étant des ensembles de sommet et E est un ensemble d'arêtes) est un graphe de contraintes de H si et seulement si : $E=\{(v, c) \in V \times C / v$ est contrainte par $c\}$.

8.2.2 Cellules de contraintes

Dans [HKS+94] la notion de cellule de contraintes à été introduite. Cette notion est modélisée par la définition suivante :

1. Dans le cas particulier où $m=1$, cette hypothèse est équivalente à celle de l'algorithme écrit dans [18].

Définition 8.2[1]

Soit une hiérarchie de contraintes $H=(V,C)$ et soit $G=(V, C, E)$ un graphe de contraintes de H. pour $X \subseteq V$, Γ est défini par :

$$\Gamma(X) = \{ c \in C / \exists v \in X \wedge (v,c) \in E \}$$

La paire $p=(V_p, C_p)$ est une cellule de contraintes dans G si et seulement si l'une des deux conditions suivantes est vraie :

1. $V_p \subseteq V$, $C_p = \varnothing$ et $Card(V_p) = 1$.
2. $V_p \subseteq V$, $C_p \subseteq C$, *le sous graphe induit par* V_p *et* C_p *est connexe et*

$\forall X \subseteq V_p \; Card(X) \leq Card(\Gamma_p(X))$ *avec* $\Gamma_p(X) = \Gamma(X) \cap C_p$.

La paire $p=(V_p, C_p)$ est une cellule sur-contrainte dans G si et seulement si : $\text{Card}(V_p) < \text{Card}(C_p)$.

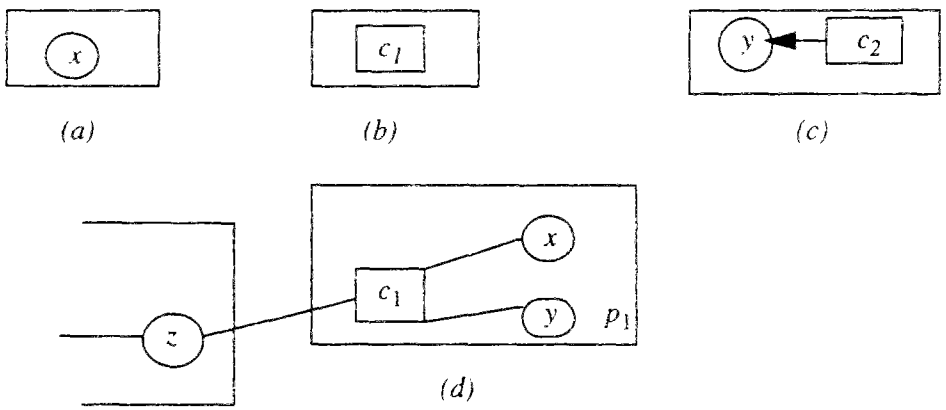
Explicitement, cette définition veut dire qu'on possède une cellule de contraintes dans l'un des deux cas suivants :

1. Si on a une seule variable.
2. Le cardinal de chaque sous-ensemble de variables dans la cellule est inférieur ou égal au nombre de contraintes attachées à ce sous-ensemble de variables dans cette cellule[2].

Dans le cas où le nombre de contraintes dans la cellule p est supérieur au nombre de variables dans p alors, cette cellule est dite sur-contrainte (c.à.d. qu'il existe une variable qui est déterminée par plus d'une contrainte).

La figure 52 montre quelques exemples de cellules de contraintes. Le rectangle contenant le cercle dans la partie (a) de la figure montre une cellule contenant une seule variable. La partie (b) montre une cellule contenant une seule contrainte.

FIGURE 52 : Cellules de contraintes



Les valeurs des variables d'une cellule de contrainte sont obtenues par la résolution des contraintes dans la cellule. Par exemple, dans la cellule de la partie (c) de la figure 41, la valeur de la variable y est déterminée par la contrainte c_2 puisque d'après à la définition 8.2, ceci est possible pour les contraintes uni-sortie.

1. La sémantique de cette définition reste identique à celle donnée dans [HKS+94], mais pour des raisons d'homogénéité nous l'avons traduite dans notre formalisme.
2. Intuitivement, ce cas veut dire : la valeur de chaque variable dans X est déterminée par au moins une contrainte dans C_p.

Le deuxième cas de la définition 8.2 ne permet pas de conclure sur l'existence d'une cellule de contraintes lorsque la hiérarchie contient des contraintes multi-sorties. Par exemple, dans la partie *(d)* de la figure 52, si la contrainte c_1 est à 2 sorties alors on n'est ni dans le cas 1 de la définition 8.2 (puisque p_1 n'est pas réduit à une variable) ni dans le cas 2 de la définition 8.2 (puisque le nombre de variables dans p_1 n'est pas inférieur au nombre de contraintes dans p_1) pour pouvoir dire que p_1 est une cellule et qu'elle peut être résolue localement[1]. Partant de là, et dans le but de pouvoir manipuler des contraintes multi-sorties, on présente par la définition 8.3 une nouvelle modélisation de la notion de cellule de contraintes. Cette dernière peut être vue comme une généralisation de la définition précédente. Juste après la présentation de la nouvelle définition d'une cellule, on donnera plusieurs exemples pour bien montrer son intérêt.

Notation et hypothèses :

Pour $c \in C$, on dénote par $m(c)$ le nombre de variables de sortie de la contrainte c et par $n(c)$ le nombre de variables contraintes par c. $(\forall\, c \in C,\ n(c) - m(c) >= 1)$.On suppose que chaque méthode d'une contrainte c du système possède $n(c)$ variables dont $m(c)$ sont des variables de sortie.

Définition 8.3

Soit une hiérarchie de contraintes $H=(V,C)$ et soit $G=(V, C, E)$ un graphe de contraintes de H. Soit $p=(V_p, C_p)$ avec $V_p \subset V$ et $C_p \subset C$.

pour $c \in \Gamma_p(X)$, on dénote par $n_X(c) = Card\ \{v \in X, (v, c) \in G\}$ *(il est à noter que :* $n_X(c) \leq n(c)$*)*.

La paire p est une cellule de contraintes de G si et seulement si l'une des deux conditions suivantes est vraie :

1. $C_p = \varnothing$ et $Card(V_p) = 1$,
2. *le sous graphe de G induit par* V_p *et* C_p *est connexe* et

$\forall\, X \subseteq V_p\ Card(X) \leq \mu(\Gamma_p(X))$ avec $\mu(\Gamma_p(X)) = \sum_{c \in \Gamma p(X)} Inf(m(c), n_X(c))$.

Explicitement, cette définition généralisée veut dire qu'on possède une cellule de contraintes dans l'une des deux cas suivant :

1. Si on a une seule variable.

2. Le cardinal de chaque sous ensemble de variables dans la cellule est inférieur ou égal à la somme des nombres minimaux de variables liés à ce sous ensemble. Ces nombres sont obtenus à partir des contraintes attachées aux variables de ce sous ensemble, en considérant pour chacune de ces contraintes, ses variables de sortie et ses variables dans ce sous-ensemble.

Exemples de cellules :

La partie (a) de la figure 53 montre le cas où p_1 est une cellule contenant une contrainte uni-sortie selon les deux définitions 8.2 et 8.3, puisque :

Si on considère la définition 8.2, on a la condition du deuxième cas vérifiée car :

$\forall\, X \subseteq V_p\ Card(X) \leq Card(\Gamma(X) \cap C_p) \Leftrightarrow Card\{x\} \leq Card\{c_1\}$

$Card\{x\} \leq Card\{c_1\} \Leftrightarrow 1 \leq 1 \Leftrightarrow$ *vraie.*

Si on considère la définition 8.3, on a la condition du deuxième cas vérifiée car :

$\forall\, X \subseteq V_p\ Card(X) \leq \sum_{c \in \Gamma p(X)} Inf(m(c), n_X(c)) \Leftrightarrow Card\{x\} \leq Inf(m(c_1), 1)$

$Card\{x\} \leq Inf(m(c_1), 1) \Leftrightarrow 1 \leq 1 \Leftrightarrow$ *vraie.*

1 Les variables de sortie d'une contrainte dans la cellule doivent être déterminées au sein de cette cellule.

La partie (b) de la figure 42 montre le cas où p_1 peut être une cellule par la définition 8.3, tandis qu'avec la définition 8.2 elle ne l'est pas.

Si on considère la définition 8.2, p_1 n'est pas considérée comme cellule puisque :

$$\forall X \subseteq V_p \; Card(X) \leq Card(\Gamma(X) \cap C_p) \Leftrightarrow Card\{x,y,z\} \leq Card\{c_1\} \Leftrightarrow 3 \leq 1 \Leftrightarrow faux.$$

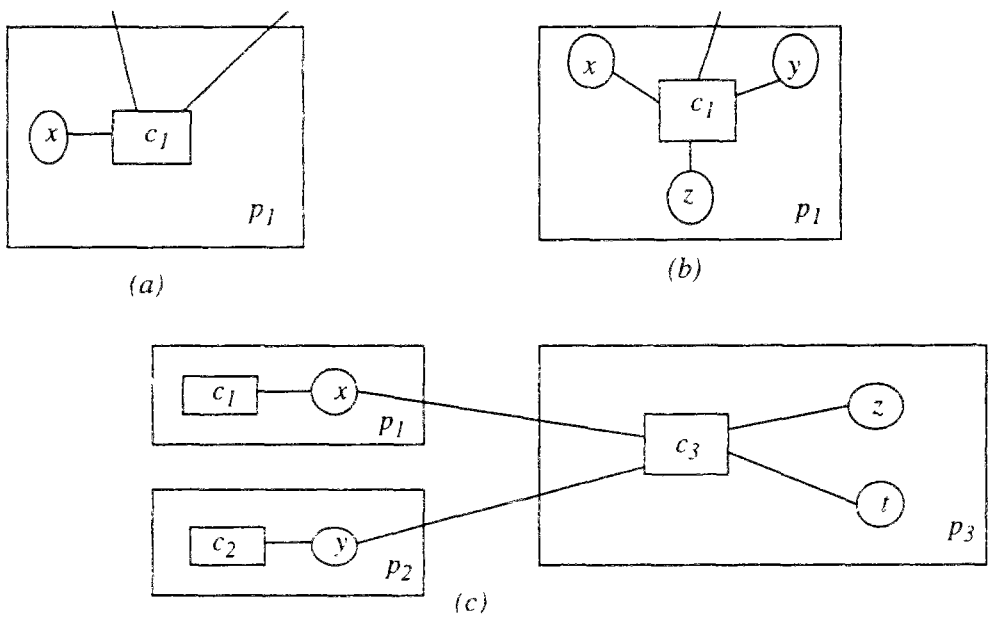
Si on considère la définition 8.3 on a la condition du deuxième cas vérifiée lorsque la contrainte c_1 est à 3 sorties[1]. Si c_1 a deux ou une variable en sortie, alors p_1 ne sera pas considéré comme une cellule.

$$\forall X \subseteq V_p \; Card(X) \leq \sum_{c \in \Gamma p(X)} Inf(m(c), n_X(c)) \Leftrightarrow Card\{x,y,z\} \leq Inf(m(c_1), 3)$$

$$Card\{x,y,z\} \leq Inf(m(c_1), 3) \Leftrightarrow 3 \leq Inf(3,3) \Leftrightarrow vraie.$$

La partie (c) de la figure 42 montre le cas d'un graphe contenant p_1, p_2 et p_3. Les contraintes c_1 et c_2 sont uni-sortie et la contrainte c_3 est à 2 sorties. p_1 et p_2 sont considérées comme des cellules par les deux formalismes des deux définitions 8.2 et 8.3. Seul le formalisme de la définition 8.3 considère que p_3 est une cellule.

FIGURE 53 : Cellules de contraintes selon les formalismes définis.



La figure 54 montre le cas d'une cellule contenant plus d'une contrainte selon le formalisme de la définition 8.3. On montre par cet exemple le rôle de la somme (c.à.d $\sum_{c \in \Gamma p(X)} Inf(m(c), n_X(c))$) dans la détermination des variables de sorties des contraintes dans une cellule[2]. On suppose que les deux contraintes c_1 et c_2 sont à 2 sorties. La contrainte c_1 et les variables sur lesquelles elle porte ne constituent pas une cellule (puisque $Card\{x,y,z\} > Inf(2, 3)$). La contrainte c_2 et le sous ensemble de variables $\{y,t\}$ sur lequel elle porte, constituent une cellule de contraintes (puisque $Card\{y,t\} \leq Inf(2, 2)$). Le fait de considérer ces deux contraintes ensemble avec les variables $\{x,y,z,t\}$ constitue une cellule bien formée. Ceci provient de la variable y qui est une variable de sortie de la contrainte c_2 et aussi utilisée comme variable d'entrée de la contrainte c_1. Par conséquent, l'utilisation de la somme est justifiée. Lorsque les variables e et f seront déterminées par d'autres contraintes dans d'autres cellules, leurs valeurs seront propagées à la cellule p_1.

1. Dans la phase de résolution, les trois variables seront déterminées par la contrainte c_1 dans la cellule.
2. La définition 8.3 telle qu'elle est conçue, permet la détermination des variables de sortie localement dans la cellule.

La contrainte c_2 pourra ainsi déterminer les valeurs de ses deux variables de sortie t et y et ensuite la contrainte c_1 déterminera les valeurs de ses variables de sortie x et z.

La condition $\forall X \subseteq V_p$ $Card(X) \leq \Sigma_{c \in \{c1,c2\}} Inf(m(c), n_X(c))$ est vraie puisque :

Si $X=V_p$ on aboutit à $4 \leq (Inf(2,3) + Inf(2,2))$ ce qui est vrai.

Si $X=\{x,y\}$ on aboutit à $2 \leq (Inf(2,2) + Inf(2,1))$ ce qui est vrai.

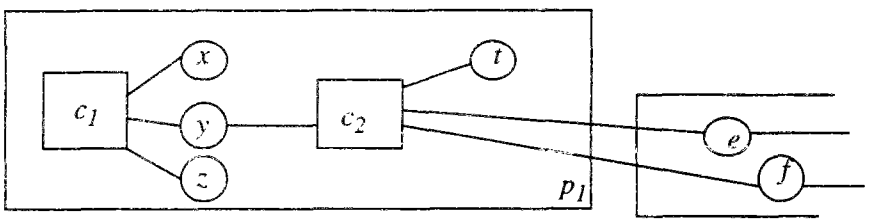
Si $X=\{x\}$ on aboutit à $1 \leq (Inf(2,1) + Inf(2,0))$ ce qui est vrai.

...

pour tout ensemble dans l'ensemble des parties de $\{x,y,z,t\}$ on obtient vrai.

D'une façon générale, lorsqu'un sous graphe induit par n contraintes et leurs variables est connexe et la condition $\forall X \subseteq V_p$ $Card(X) \leq \Sigma_{c \in \Gamma p(X)} Inf(m(c), n_X(c))$ est réalisée alors on a une cellule contenant ces n contraintes.

FIGURE 54 : Cellule contenant plus d'une contrainte



Remarque 8.1:

Si toutes les contraintes dans C sont à une sortie $(c.à.d.\ m(c)=1)$ alors la définition 8.3 est équivalente à la définition 8.2 du fait que $\mu(\Gamma_p(X)) = Card(\Gamma_p(X))$ puisque :

$\mu(\Gamma_p(X)) = \Sigma_{c \in \Gamma p(X)} Inf(1, n_X(c))$

$= \Sigma_{c \in \Gamma p(X)} 1$ puisque $n_X(c)$ est toujours supérieur ou égal à 1

$= Card(\Gamma_p(X))$

Propriété 8.1:

$\Gamma_p(V_p) = C_p$ avec $(p \neq (V_p, \varnothing))$

Preuve 8.1:

- $\Gamma(V_p) = \{ c \in C / \exists v \in V_p$ et $(v,c) \in E \}$

- p est connexe $\Rightarrow (V_p=\varnothing$ et $Card(C_p)=1)$ ou $(C_p \subset \Gamma(V_p))$

par définition : $\Gamma_p(V_p) = \Gamma(V_p) \cap C_p$, et donc, puisque $C_p \subset \Gamma(V_p)$ alors $\Gamma_p(V_p) = C_p$.

Propriété 8.2 :

La condition $Card(V_p) \leq \mu(C_p)$ (c.à.d $Card(X) \leq \mu(\Gamma_p(X))$) est nécessaire pour avoir une cellule mais elle n'est pas suffisante. Il faut considérer aussi tous les éléments de l'ensemble des parties de V_p (comme il est indiqué dans la définition 8.3).

Preuve 8.2:

Considérons la figure 55, on suppose[1] que toutes les contraintes sont uni-sortie ($m(c_i) = 1$ $\forall i \in [1..5]$).. Soit $p = (V_p, C_p)$ avec $V_p = \{x_1, x_2, x_3, x_4\}$, $Card(V_p) = 4$ et $C_p = \{c_1, c_2, c_3, c_4, c_5\}$.

$\mu(C_p) = \Sigma_{c \in Cp} Inf(1, n_{Vp}(c))$

$= \Sigma^5_{i=1} Inf(1, n_{Vp}(c_i))$

$= \Sigma^5_1 1 = 5$ puisque $n_{Vp}(c_1) = 1$, $n_{Vp}(c_2) = 1$, $n_{Vp}(c_3) = 2$, $n_{Vp}(c_4) = 1$, $n_{Vp}(c_5) = 3$.

La condition $Card(V_p) \leq \mu(C_p)$ est vérifiée (puisque $4 \leq 5$) et pourtant p n'est pas une cellule puisque la condition : $\forall X \subseteq V_p$ $Card(X) \leq \mu(\Gamma_p(X))$ de la définition 8.3 n'est pas vérifiée. Il suffit de prendre :

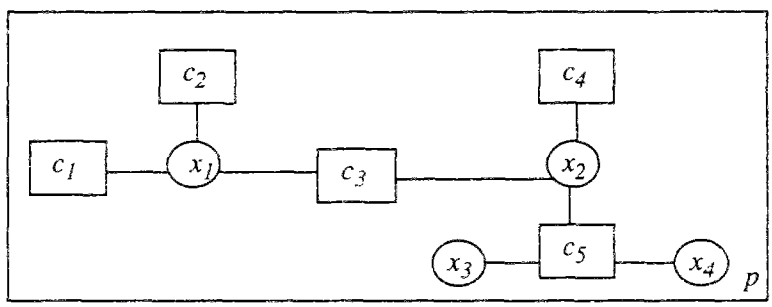
$X = \{x_3, x_4\}$ et $\Gamma_p(X) = \{c_5\}$ et donc on a :

$\mu(\Gamma_p(X)) = Inf(n_X(c_5), m(c_5))$

$= Inf(n_X(c_5), 1)$

$= Inf(2, 1) = 1$. On aboutit donc à : $Card(X) > \mu(\Gamma_p(X))$ (car $2 > 1$).

FIGURE 55 : Graphe connexe ne constituant pas une cellule.



8.2.3 Cellule sur-contrainte

On définit maintenant la notion de cellule sur-contrainte par la définition suivante :

Définition 8.4:

Soit $p = (V_p, C_p)$ une cellule de contraintes telle que $C_p \neq \emptyset$. p est une cellule sur-contrainte si et seulement si : $Card(V_p) < \mu(\Gamma_p(V_p))$.

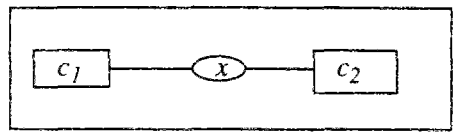
En utilisant la propriété 8.1, la cellule de contrainte p ($\neq (V_p, \emptyset)$) est une cellule sur-contrainte si et seulement si : $Card(V_p) < \mu(C_p)$. Nous verrons par la suite l'utilisation de ce type de cellule.

Exemple :

La cellule de la figure 56 montre une cellule sur-contrainte puisque $Card(Vp) = 1$ et $\mu(C_p) = 1 + 1 = 2$.

1. On peut faire cette supposition puisqu'on a montré que les deux définitions 8.2 et 8 3 sont équivalentes lorsqu'il s'agit de contraintes uni-sortie.

FIGURE 56 : Cellule sur-contrainte



8.2.4 Graphe admissible

Définition 8.5[1]:

Etant donné un graphe de contraintes $G=(V,C,E)$ et un ensemble P de cellules de contraintes dans B. Le quadruplet $G_A=(V,C,E,P)$ est un graphe admissible pour G si et seulement si :

Chaque variable dans V se trouve dans une et une seule des cellules de P.

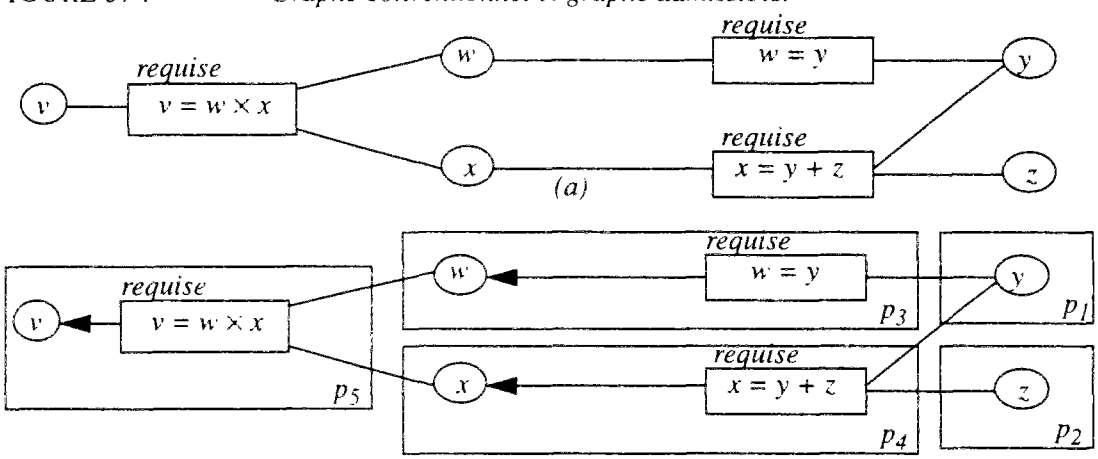
Chaque contrainte dans C se trouve dans une et une seule des cellules de P.

Il n' y a pas de dépendance cyclique entre les cellules de P.

Exemple :

La partie (b) de la figure 57 montre un graphe admissible équivalent à celui de la partie (a) de cette figure. Une valuation qui satisfait les contraintes de cet exemple peut être obtenue en résolvant les cellules: p_1, p_2, p_3, p_4 et p_5 dans cet ordre et en utilisant l'algorithme de la propagation locale pour propager les valeurs des variables entres les cellules.

FIGURE 57 : Graphe conventionnel et graphe admissible.



L'intérêt principal d'un graphe composé de cellules de contraintes est de pouvoir modéliser la prise en compte des circuits entre les méthodes des contraintes du système, et aussi, les conflits des méthodes des contraintes associées à un comparateur utilisant l'erreur métrique. Dans la section suivante, on présentera le mécanisme à utiliser pour obtenir un graphe solution à partir d'un graphe admissible. Les graphes solutions que l'on vise à obtenir ici sont des graphes solutions à des hiérarchies possédant à leurs derniers niveaux des contraintes associées au comparateur global *Moindre-Carrés-Métrique* ou au comparateur global *Pire-Cas-Métrique* (ou encore un autre comparateur qui possède les mêmes propriétés que ces deux derniers).

1. Cette définition est équivalente à celle donnée dans [HKS94+], mais pour ce qui nous concerne ces fonctionnalités sont différentes puisqu'elle repose sur la notion de cellule.

8.3 Graphe solution

Dans la première partie de cette section, on présente un exemple qui montre le déroulement de la procédure visant à l'obtention d'un graphe solution pour un des types des hiérarchies cité auparavant. En deuxième partie de cette section, nous présentons quelques définitions qui permettent d'avoir des outils. Ces derniers serviront à définir d'une manière générale un graphe solution afin de produire une solution à la hiérarchie. Nous rappelons ici qu'une hiérarchie peut contenir plusieurs types de contraintes (c.à.d les niveaux de la hiérarchie excepté le dernier, ne sont pas forcement associés au même comparateur local) avec, en dernier niveau, les contraintes sont associées à l'un des comparateurs *Pire-Cas-Métrique* (type de solution τ_{PCM}), *Moindre-Carrés-Métrique* (type de solution τ_{MCM}).

8.3.1 Exemple d'un graphe solution

Les graphes conventionnels ne peuvent pas produire de solution à une hiérarchie du type décrit auparavant. Par contre les graphes composés de cellules de contraintes sont un moyen pour la résolution de ce type de hiérarchies. Par exemple, considérons la hiérarchie à trois niveaux de préférence décrite dans la figure 58. Les deux niveaux *fort* et *moyen* sont associés au comparateur *Localement-Prédicat-Meilleur* (τ_{LPM}) et le niveau *faible* est associé au comparateur *Moindres-Carrés-Métrique*. Alternativement, on supposera aussi que ce niveau peut être associé au comparateur *Pire-Cas-Métrique*. Les contraintes de cette hiérarchie sont toutes supposées pondérées à un poids de 1. Toutes les contraintes ici sont uni-sortie, exceptée la contrainte c_4 qui est une contrainte à 2 sorties.

FIGURE 58 : Hiérarchie à trois niveaux intégrant deux types de contraintes.

requise

$c_1 : v=0$

$c_7 : t-s=1$

forte

$c_4 : (z, y) = (v-y, x+v)$

$c_8 : t+s=r$

moyenne

$c_6 : r=2$

faible

$c_2 : w=v$

$c_3 : x=1$

$c_5 : y=t$

La figure 59 montre le graphe de contraintes de cette hiérarchie et la figure 60 montre un graphe de méthodes pour cette hiérarchie. Ce dernier ne peut pas produire par propagation locale une solution à cette hiérarchie puisqu'il contient un conflit entre la contrainte c_1 et la contrainte c_2 sur la variable v et un autre conflit entre c_4 et c_5 en y ainsi qu'un circuit entre c_7 et c_8.

FIGURE 59 : Graphe de contraintes

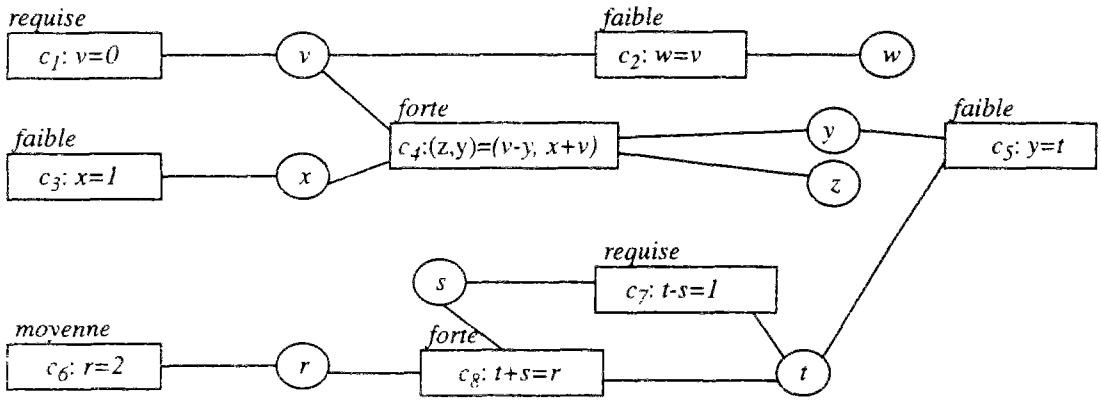
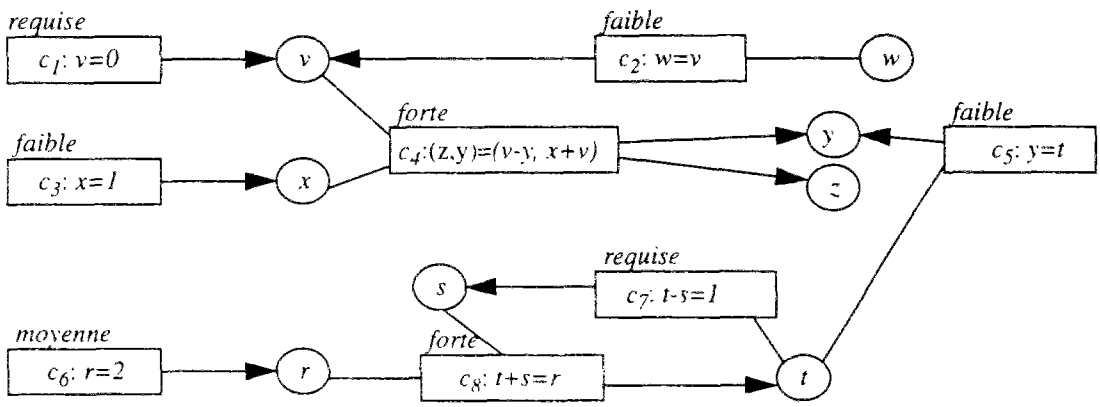


FIGURE 60 : Graphe conventionnel contenant cycle et conflits



Pour débuter, on crée un graphe admissible composé de cellules qui peuvent contenir des cycles et des conflits. Ceci est illustré par la figure 61. En satisfaisant les contraintes localement dans les cellules, on obtient la valuation θ: *{x=1, v=0, w=0, y=1, z=-1, r=2, t=3/2, s=1/2}*. Les séquences de combinaisons d'erreurs par niveau et selon le critère associé sont:

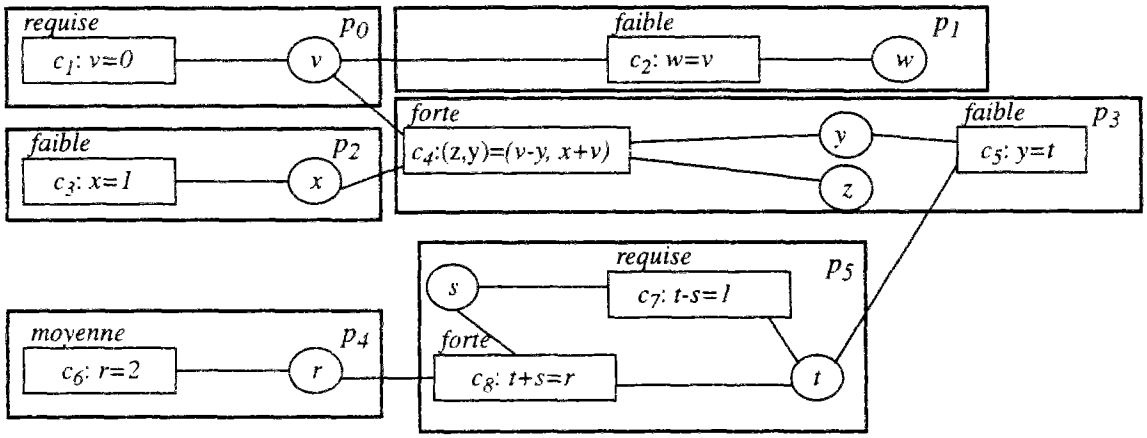
forte : $g_{\tau LPM}(E_{\tau LPM}([c_4,c_8]\ \theta)) = g_{\tau LPM}([0,0])=[0,0]$.

moyen : $g_{\tau LPB}(E_{\tau LPM}([\ c_6]\ \theta)) = g_{\tau LPM}([0])=[0]$.

faible : $g_{\tau MCM}(E_{\tau MCM}([c_2,\ c_3,\ c_5]\ \theta)) = g_{\tau MCM}([0,0,1/2]= 0^2+0^2+1/2^2=1/4$.

$g_{\tau PCM}(E_{\tau PCM}([c_2,\ c_3,\ c_5]\ \theta)) = g_{\tau PCM}([0,0,1/2]= 1/2$.

FIGURE 61 : Graphe admissible composé de cellules de contraintes



Le fusion d'une cellule sur-contrainte avec d'autres cellules non sur-contraintes crée en général[1] un meilleur graphe (c.à.d une meilleure valuation). Ceci vient du fait que la nouvelle cellule obtenue après fusion acquiert plus de liberté pour déterminer les valeurs de ces variable. La fusion des cellules repose sur les critères "cellules contenant des contraintes du dernier niveau de la hiérarchie", "cellules adjacentes" et "cellules formant une dépendance cyclique". On définira par la suite la notion de cellules adjacentes.

Par exemple, la cellule p_3 de la décomposition dans la figure 61 est une cellule sur-contrainte (d'après la définition 8.4). En fusionnant cette cellule avec la cellule p_2, on aboutit à la création de la nouvelle cellule p_{23}. Ceci donne le graphe solution de cette hiérarchie montré par la figure 62. Dans ce cas, les valeurs des variables *v, w, r, t* et *s* ne sont pas changées puisqu'elles sont déterminées par les contraintes des autres cellules ($\neq p_2$ *et* $\neq p_3$). Les valeurs des variables *x, y* et *z* sont redéterminées par les contraintes de la nouvelle cellule p_{23} telles que la contrainte c_4 soit satisfaite[2] *(i)* et que l'erreur métrique produite par les contraintes c_3 et c_5 soit minimale *(ii)*[3] *ou (iii)*[4] avec :

$(i) \Leftrightarrow z = 0 - y \wedge y = x+0$

$(ii) \Leftrightarrow (y-3/2)^2+(x-1)^2$ *est minimale.*

$(iii) \Leftrightarrow Max\ (|y-3/2|, |x-1|)$ *est minimale.*

Pour le cas où τ_{MCM} est utilisé alors :

$(i) \wedge (ii) \Leftrightarrow (x=y=-z) \wedge ((x-3/2)^2+(x-1)^2)$ *est minimale*[5].

$(i) \wedge (ii) \Leftrightarrow x=y=-z=5/4.$

Pour le cas où τ_{PCM} est utilisé alors :

$(i) \wedge (iii) \Leftrightarrow (x=y=-z) \wedge Min[Max(|x-3/2|, |x-1|)].$

$(i) \wedge (iii) \Leftrightarrow (x=y=-z) \wedge Min(|x-3/2|$ *avec* $x \in]-\infty, 5/4])$; $|x-1|$ *avec* $x \in [5/4, +\infty[).$

$(i) \wedge (iii) \Leftrightarrow x=y=-z=5/4$

Normalement, chaque calcul effectué au-dessus doit être fait par un résolveur spécifique selon le critère utilisé par les contraintes et selon la nature des contraintes (c.à.d linéaire, non linéaire ...)

1. On garantit que cette opération dans le pire des cas produit un graphe aussi bon que le graphe initial.
2. Puisque la contrainte c_4 est associée au type de solution τ_{LPM}.
3. Si les contraintes c_3 et c_5 sont associées au type de solution τ_{MCM}
4. Si les contraintes c_3 et c_5 sont associées au type de solution τ_{PCM}
5. Ici la dérivée de cette fonction s'annule lorsque $x = 5/4$.

La nouvelle valuation obtenue en utilisant le type τ_{MCM} ou le type τ_{PCM} pour les contraintes du niveau faible est : $\theta' = \{x=5/4,\ v=0,\ w=0,\ y=5/4,\ z=-5/4,\ r=2,\ t=3/2$ *et* $s=1/2\}$. Les séquences de combinaisons d'erreurs par niveau et selon le critère associé sont :

forte $g_{\tau LPM}(E_{\tau LPM}([c_4,c_8]\ \theta')) = g_{\tau LPM}([0,0])=[0,0]$.

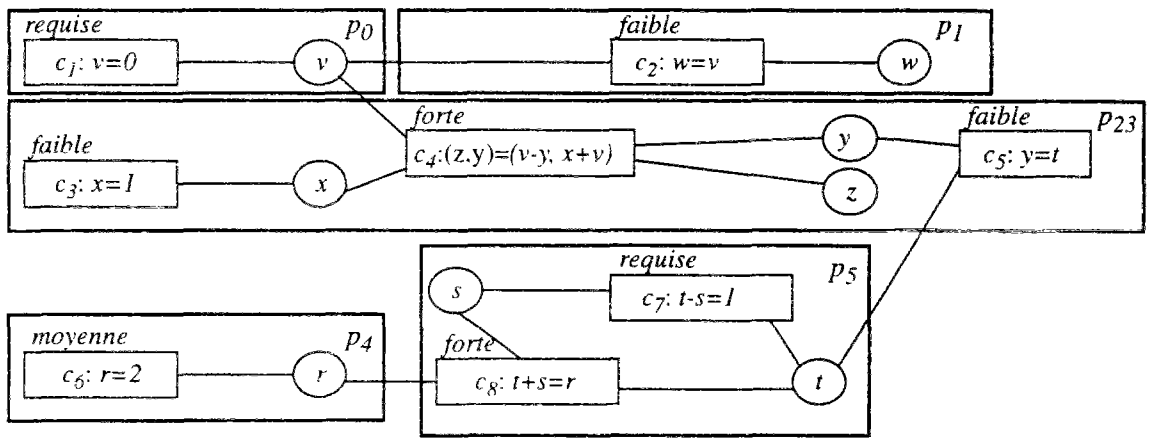
moyen $g_{\tau LPM}(E_{\tau LPM}([\ c_6]\ \theta')) = g_{\tau LPM}([0])=[0]$.

faible $g_{\tau MCM}(E_{\tau MCM}([c_2,\ c_3,\ c_5]\ \theta')) = g_{\tau MCM}([0,1/4,1/4]= 0^2+1/4^2+1/4^2=1/8$.

$g_{\tau PCM}(E_{\tau PCM}([c_2,\ c_3,\ c_5]\ \theta')) = g_{\tau PCM}([0,1/4,1/4]= 1/4$.

Ces séquences montrent bien que θ' est meilleur que θ puisqu'au niveau faible de la hiérarchie, l'erreur après application de θ' est inférieure à celle obtenue après application de θ.

FIGURE 62 : Graphe solution de hiérarchie de contraintes



8.3.2 Comment obtenir un graphe solution

Dans ce paragraphe, on définit comment obtenir un graphe de cellules solution pour une hiérarchie. Ce graphe solution produira une solution à la hiérarchie de contraintes. Cette solution sera obtenue en résolvant localement chaque cellule et en utilisant la propagation locale pour propager les valeurs des variables d'une cellule aux cellules voisines.

Avant de définir un graphe solution, on présente successivement les définitions de cellules adjacentes, de l'*étiquette-interne* d'une cellule et enfin celle de l'*étiquette-voyageuse* d'une cellule. Ces trois définitions sont équivalentes à celles données dans [HKS+94] puisque leurs sémantiques restent valides pour notre problématique, cependant elles sont traduites dans notre formalisme pour une question d'homogénité. Ces trois définitions participent à la définition d'un graphe solution d'une hiérarchie de contraintes que nous allons concevoir.

La définition de l'*étiquette-voyageuse* d'une cellule est inspirée de celle introduite dans les formalismes DeltaBlue et SkyBlue pour les variables (voir chapitre 4).

Cellules adjacentes

Définition 8.6:

Soient p et p' deux cellules de contraintes telles que $p=(V_p, C_p)$ et $p'=(V_{p'}, C_{p'})$. *p' est adjacente à p* si et seulement si $\exists l>=1\ \exists v_1,...,v_l \in V_{p'}\ \exists c \in C_p$ avec : $(v_1, c) \in E,....,(v_l, c) \in E$.

Explicitement, une cellule est adjacente à une autre cellule si elle possède au moins une variable reliée à une contrainte dans cette autre cellule. Intuitivement, le but de cette définition est : lorsqu'on fusionne une cellule sur-contrainte avec certaines cellules voisines adjacentes à cette cellule[1], on est certain que la nouvelle cellule obtenue produira une solution meilleure (ou au moins aussi bonne) que la solution obtenue par résolution de ces cellules sans les fusionner.

Exemple :

Reprenant l'exemple de la figure 61, les cellules p_0 , p_2 et p_5 sont adjacentes à la cellule p_3. Ici, seule la cellule p_2 est fusionnée à la cellule p_3 (puisqu'il est impossible de créer une solution meilleure si l'on fusionne aussi p_0 et/ou p_5).

On verra *via* les définitions de l'*étiquette-interne* et de l'*étiquette-voyageuse* d'une cellule, comment on peut prédire (ou déterminer) l'ensemble de cellules adjacentes à une cellule sur-contrainte dans un graphe admissible qu'il faut fusionner pour obtenir un graphe solution.

Etiquette-interne d'une cellule de contraintes

Définition 8.7:

Soit $p=(V_p, C_p)$ une cellule de contraintes. Si $C_p = \varnothing$ alors l'*étiquette-interne* d'une cellule est égale à *très-faible*[2] sinon elle est égale à la plus faible valeur de l'ensemble des valeurs des étiquettes associées aux contraintes dans C_p.

Exemple :

Reprenons l'exemple de la figure 61. L'*étiquette-interne* de la cellule p_3 est égale à *faible* (puisque *Inf(faible, forte) = faible*).

Etiquette-voyageuse d'une cellule de contraintes

Définition 8.8 :

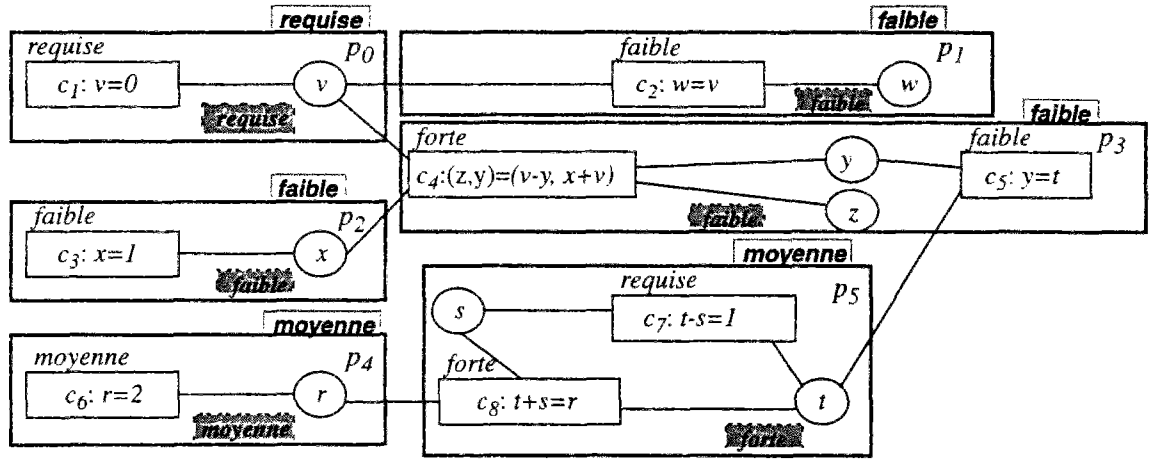
Soit $p=(V_p, C_p)$ une cellule de contrainte. L'*étiquette-voyageuse* de p est égale à l'étiquette minimale dans l'ensemble formé par l'*étiquette-interne* de p et les *étiquettes-voyageuses* des cellules adjacentes à p (cette définition est récursive, mais on garantit toujours l'existence d'un point fixe pour n'importe quel graphe de cellules).

Exemple :

Si l'on considère l'exemple de la figure 61, si on calcule les *étiquettes-voyageuses*[3] des cellules de ce graphe alors on aboutit au graphe de la figure 53.

1. Nous verrons après pourquoi on a dit : "certaines cellules voisines" et pas "toutes les cellules voisines".
2. On suppose que *tres-faible* est la plus faible étiquette de la hiérarchie.
3. On calcule d'abord les *étiquettes-internes* des cellules afin de pouvoir calculer les *étiquettes-voyageuses*.

FIGURE 63 : *Etiquettes-voyageuses et étiquettes-internes* des cellules d'un graphe



Théorème 8.1

Soit p une cellule sur-contrainte telle que :

> tout couple formé par les contraintes ayant des étiquettes différentes de l'étiquette-interne de cette cellule et les variables de ces contraintes dans cette cellule ne constitue pas une cellule.

Soit p' une cellule adjacente à p.

1. Si (l'*étiquette-interne* de p = l'étiquette du dernier niveau de la hiérarchie = l'*étiquette-voyageuse* de p') alors la résolution de la cellule résultante de la fusion de p et p' produira une valuation meilleure (au moins aussi bonne) que celle obtenue par la résolution séparée de p et de p'.

2. Si (l'*étiquette-interne* de p = l'étiquette du dernier niveau de la hiérarchie < l'*étiquette-voyageuse* de p') alors la résolution de la cellule résultante de la fusion de p et p' ne produira pas de valuation meilleure que celle obtenue par la résolution séparée de p et de p'.

Preuve :

On dénote par i_p l'*étiquette-interne* de p (qui est aussi l'étiquette associée au dernier niveau de la hiérarchie) et par $v_{p'}$ l'*étiquette-voyageuse* de p'. D'après la définition de l'*étiquette-voyageuse*, on affirme que les variables de la cellule p' sont déterminées par des contraintes ayant des étiquettes plus fortes ou égales à $v_{p'}$ *(1)*. D'après l'hypothèse de ce théorème et les définitions d'une cellule sur-contrainte et de l'*étiquette-interne*, on affirme que les contraintes étiquetées par i_p contribuent à déterminer les variables de la cellule p *(2)*.

1. Partant des affirmations *(1)* et *(2)* et puisque i_p est égale à $v_{p'}$, le fait de fusionner p et p' ne peut que faire décroître l'erreur globale des contraintes du dernier niveau de la hiérarchie (c.à.d celle associées à l'un des comparateurs *MCM* ou *PCM*) puisque ces contraintes auront plus de liberté pour déterminer les variables, et de plus , cette fusion n'aura pas d'effet sur un niveau plus haut associé à un critère local. Par conséquent, il y aurait alors création d'une valuation meilleure que celle produite par une résolution séparée des deux cellules p et p'.

2. Partant des affirmations *(1)* et *(2)* et puisque i_p est plus faible que $v_{p'}$ alors aucune contrainte du dernier niveau n'est dans p' (ni dans la cellule adjacente à p', ni dans l'adjacente à l'adjacente à p', etc). Par conséquent, il est impossible d'obtenir une valuation meilleure que celle produite par une résolution séparée des deux cellules p et p'.

Dans la figure 63, les cellules p_0, p_2 et p_5 sont toutes adjacentes à la cellule sur-contrainte p_3. Seule p_2 est fusionnée à p_3 puisque son *étiquette-voyageuse (= faible)* est égale à l'*étiquette-interne* de p_3 *(= faible)*. Le résultat issu de cette fusion est dans la figure 62.

Arrivé à ce stade, on dispose maintenant des outils nécessaires pour définir un graphe solution d'une hiérarchie qui, utilise à son niveau le moins important, un des critères globaux cités auparavant. La définition de graphe solution est présentée dans la section suivante.

Graphe solution d'une hiérarchie

Définition 8.9:

Un graphe admissible est un graphe solution s'il satisfait les deux conditions suivantes :

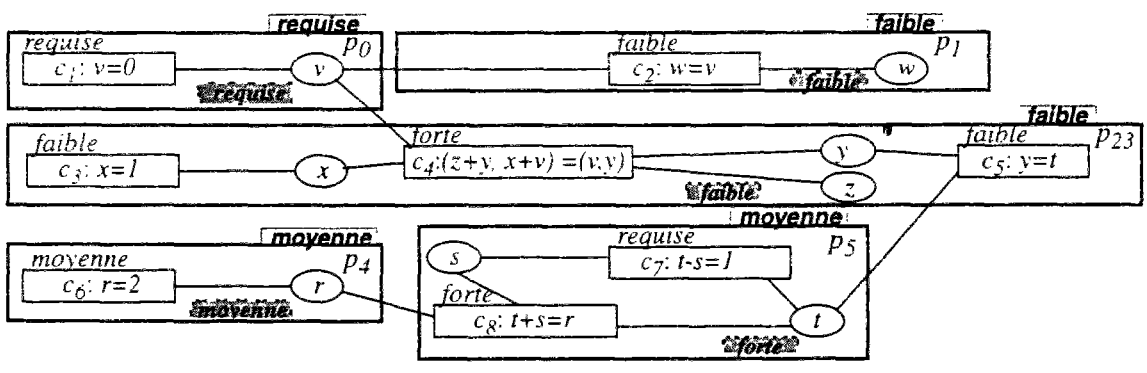
Pour chacune des cellules dans le graphe ayant plusieurs contraintes, le couple formé par les contraintes ayant des étiquettes différentes de l'*étiquette-interne* de cette cellule et les variables de ces contraintes dans cette cellule ne constitue pas une cellule de contraintes.

Pour chacune des cellules *sur-contraintes* contenant des contraintes du dernier niveau de la hiérarchie, son *étiquette-interne* est plus faible que chacune des *étiquettes-voyageuses* des cellules adjacentes à cette cellule sur-contrainte.

La première condition veut dire qu'une cellule de contraintes doit utiliser ses contraintes faiblement étiquetées pour la détermination des valeurs de ses variables. Par conséquent on obtient une valuation produisant une erreur minimale. Par exemple, dans la cellule p_3 de la figure 63, le couple $(c_4, \{y,z\})$ constitue une cellule de contraintes selon la définition 8.3 et par conséquent la première condition de la définition d'un graphe solution n'est pas réalisée. La fusion de p_2 et p_3 en la cellule p_{23} de la figure 64 crée une cellule qui satisfait cette condition (puisque le couple $(c_4, \{x,y,z\})$ ne constitue pas une cellule[1]).

La seconde condition est basée sur le théorème 8.1. Cette condition, si elle est vérifiée, exprimera l'impossibilité de créer un graphe (en fusionnant une cellule sur-contrainte avec d'autres cellules non sur-contraintes) qui produira une valuation meilleure. Les cellules du graphe de la figure 64 obéissent toutes à cette condition puisque la seule cellule sur-contrainte est p_{23} et son étiquette interne *(= faible)* est plus faible que les *étiquettes-voyageuses* des cellules p_0 (= *requise*) et p_5 (= *moyenne*) qui lui sont adjacentes.

FIGURE 64 : Graphe solution de cellules étiquetées



1 Il suffit de prendre $X=\{x,y,z\}$ pour falsifier la condition $\forall X \subseteq \{x,y,z\}\ Card(X) \leq \sum_{c \in \{c4\}} Inf(m(c), n_X(c))$ (car on aboutit à $3 \leq 2$).

8.4 Algorithmes

L'algorithme décrit dans [HKS+94] manipulant uniquement des hiérarchies de contraintes unisortie est conçu en deux couches : un résolveur général et des résolveurs spécifiques. Ici, nous gardons cette structure pour présenter la généralisation que nous avons conçue et qui consiste à considérer les contraintes multi-sorties et de considérer aussi le cas où le comparateur *Pire-Cas-Métrique* est associé au dernier niveau de la hiérarchie.

Le résolveur général planifie un graphe solution et les résolveurs-spécifiques permettent d'obtenir les valeurs des variables en résolvant les contraintes localement dans chaque cellule.

Comme tout algorithme incrémental, Latif est composé de deux procédures *Ajout-contrainte* et *Retrait-contrainte*. La structure utilisée par ces deux procédures est presque la même que celle donnée par l'algorithme initial, cependant les fonctionnalités changent ici puisque les définitions présentées avant opèrent sur un champ un peu plus large du monde des contraintes. La procédure *Ajout-contrainte* invoque deux opérations qui sont : *ajout-variable* et *propagation-locale*. La procédure *Retrait-contrainte* invoque elle aussi deux opérations qui sont : *retrait-variable*, *propagation-locale*. Cette dernière opération utilise l'algorithme de la propagation locale tandis que les autres modifient l'architecture du graphe.

8.4.1 Ajout-contrainte

Initialement, on suppose qu'il existe un graphe solution. Lorsqu'une nouvelle contrainte c est ajoutée à la hiérarchie, la procédure *Ajout-contrainte* met cette nouvelle contrainte dans une nouvelle cellule (ligne 1; figure 65), et cherche la cellule adjacente ayant la plus faible *étiquette-voyageuse* (ligne 2; figure 65).

Si cette *étiquette-voyageuse* est plus faible que l'étiquette de la contrainte c(on est dans le cas où il y a existence d'une ou plusieurs contraintes étiquetées par des étiquettes plus faibles ou égales à l'étiquette de la contrainte ajoutée. Ces contraintes sont appelées contraintes victimes et peuvent être visées afin de garder la sémantique de la hiérarchie. La procédure cherche le chemin dans le graphe des cellules partant de la cellule contenant c et arrivant aux cellules contenant les contraintes victimes en inversant les dépendances entre les cellules dans le chemin[1] (lignes 3,4; figure 65). Après cette opération, la contrainte c devient active.

La procédure élimine les dépendances cycliques entre les cellules générées par la précédente opération en fusionnant les cellules dépendant de la contrainte c (ligne 5; figure 65). Elle met aussi à jour les valeurs des *étiquettes-voyageuses* des cellules du graphe (ligne 6; figure 65).

Si le type de contraintes est l'un des types suivants : τ_{MCM}, τ_{PCM} (ou autre type de comparateur qui se comporte de la même façon que ces deux comparateurs) alors la procédure fusionne la cellule sur-contrainte avec les cellules qui lui sont adjacentes et qui possèdent des *étiquettes-voyageuses* égales à celle de cette cellule sur-contrainte. On rappelle que cela a pour but de minimiser l'erreur des contraintes (cf théorème 8.1) (ligne 7; figure 65).

Finalement, la procédure résout chaque cellule localement en appelant un ou plusieurs résolveurs-types selon les types de contraintes de la cellule et utilise l'algorithme de la propagation locale pour propager les valeurs de cellule en cellule dans le graphe (ligne 8; figure 65).

1. Un pas de cette opération est entièrement décrit dans la figure 66. L'exploration du chemin en entier est réalisée par la boucle tant que.

FIGURE 65 : *Procédure d'ajout d'une contrainte*

Ajout-contrainte *(c , ins-comp-global)*

1 *cl* ← *Cree-cellule(c)*
2 *evm* ← *Min(étiquettes-voyageuses* des cellules adjacentes à *cl)*
3 Si *evm* est plus faible que l'étiquette de *c* alors *eti* ← l'étiquette de *c* Fin-Si
4 Tant que *eti* est plus forte que *evm* faire : ***Chercher-et-décomposer****(cl, eti, evm)* Fin-Tantque
5 Fusionner les contraintes dépendantes de *c* et qui forment un cycle
6 Met à jour les *étiquettes-voyageuses* des cellules dépendantes de *c*
7 Si *((evm* est la plus faible étiquette) ou *(*les contraintes étiquetées par *evm* sont associées à un comparateur dans *ins-comp-global))* alors fusionner les cellules adjacentes à *c* qui possèdent des *étiquettes-voyageuses* égales à celle contenant *c*
8 Résoudre les cellules en utilisant les résolveurs-specifiques et l'opération *propagation-locale*

La figure 66 montre un pas du processus qui inverse la dépendance entre les cellules du chemin partant de la cellule contenant *c* et arrivant à la cellule contenant les contraintes étiquetée par *evm*. La procédure *Chercher-et-décomposer* réalise ce pas. Cette procédure est paramétrée par la cellule courante *cl*, l'*étiquette-interne* de cette cellule courante *eti* et enfin l'*étiquette-voyageuse evm* qui représente l'étiquette des contraintes visées à atteindre.

La procédure repère la cellule adjacente à la cellule *cl* et ayant pour *étiquette-voyageuse evm* (ligne 1; figure 66), et enlève l'ensemble des variables adjacentes à *cl* de la cellule repérée. L'ensemble de ces variables enlevées est ajouté à la cellule *cl* (ligne 3; figure 66)(c.à.d les valeurs de ces variables dépendent maintenant de la contrainte *c*). Après cette opération, la cellule adjacente à *cl* peut devenir sur-contrainte (puisqu'on vient de lui retirer un sous ensemble de l'ensemble de ces variables).

La procédure examine la cellule *adj*, si cette dernière est vide (si elle ne contenait que les variables ôtées) alors l'étiquette *très-faible*[1] est affectée à la variable *eti* afin de pouvoir falsifier la condition de la boucle Tant-que dans la procédure appelante (cf. figure 65).

La procédure compare l'*étiquette-interne* de la cellule *cl* avec l'*étiquette-voyageuse evm* de la cellule adjacente. Si ces deux étiquettes sont égales *(c.à.d* on a atteint l'extrémité du chemin*)* (ligne 5; figure 66) alors la procédure fait appel à une autre procédure pour décomposer[2] cette cellule en un ensemble de cellules (ligne 6; figure 66). La procédure détecte la cellule sur-contrainte dans cet ensemble de cellules obtenu à l'issu de la décomposition (ligne 7; figure 66), et détermine ensuite son *étiquette-interne* (ligne 8; figure 66). Dans le cas où on n'a pas encore atteint l'extrémité du chemin[3] (c.à.d l'*étiquette-interne* de la cellule *cl* n'est pas égale à *evm*) (ligne 9; figure 66) la procédure enlève alors la contrainte de la cellule *adj* (ligne 10,11; figure 66) (Il s'agit ici de la contrainte qui est adjacente à la cellule ayant pour *étiquette-voyageuse evm*).

La procédure décompose la cellule *adj* (ligne 12; figure 66), crée une nouvelle cellule contenant la contrainte enlevée[4] (ligne 13; figure 66) et détermine l'*étiquette-interne*[5] de cette nouvelle cellule crée (ligne 14; figure 66). Enfin, la procédure retourne la position courante du chemin où elle est restée (ligne 15; figure 66).

1. On suppose que *très-faible* est la plus faible étiquette de la hiérarchie.
2. On verra par la suite l'intérêt de cette décomposition.
3. Alors il faut avancer
4. Bien évidemment, cette nouvelle cellule créée a pour cellule adjacente celle qui a l'*étiquette-voyageuse evm*
5. Dans ce cas c'est l'etiquette de la contrainte *cn* de cette cellule

FIGURE 66 : Procédure Chercher-et-décomposer

```
Chercher-et-décomposer(cl, eti, evm)
 1 adj ← la cellule adjacente à cl ayant pour étiquette-voyageuse evm
 2 vars ← l'ensemble des variables dans adj adjacentes à cl
 3 Enlever vars de adj et les remettre dans cl
 4 Si adj = ∅ alors eti ← très-faible
     Sinon
 5     Si l'étiquette-interne de adj est égale à evm alors
 6       cls ← Décomposer (adj, evm)
 7       cl ← une cellule sur-contrainte dans cls
 8       eti ← l'étiquette-interne de cl
       Sinon
10       cn ← la contrainte dans adj telle que cette contrainte est adjacente à une cellule
               ayant pour étiquette-voyageuse evm
11       adj ← adj - cn
12       Décomposer (adj, evm)
13       cl ← une nouvelle cellule contenant la contrainte cn
14       eti ← l'étiquette-interne de cl
       Fin-Si
   Fin-Si
15 Retourner(cl, eti)
```

La procédure *Décomposer* est utilisée pour la véracité de la première condition de la définition d'un graphe solution. Cette procédure décompose une cellule contenant plusieurs contraintes en un ensemble de cellules. Par exemple, considérons le graphe de la figure 67 où les contraintes sont uni-sortie. Ici, il s'agit d'un graphe solution qui est composé d'une seule cellule puisque cette dernière satisfait la condition imposée par la définition d'un graphe solution (la paire $(\{c1,c2,c4\}, \{t,x,y,z\})$ ne forme pas une cellule de contrainte puisque la deuxième condition de la définition 8.3 qui modélise une cellule n'est pas satisfaite du fait que la condition $X=\{t,x,y,z\}$ $Card(X) \leq \sum_{c \in \{c1,c2,c4\}} Inf(m(c), n_X(c))$ est fausse puisqu'on obtient 4≤3).

La figure 68 montre le premier pas réalisé[1] à la suite de l'introduction de la contrainte c_6 (possédant deux variables de sortie t et e). Le fait d'enlever les variables t et e de la cellule p_0 et de les mettre dans la cellule p_1 (figure 69) ne suffit pas pour obtenir un graphe solution, puisque maintenant p_0 ne satisfait plus la première condition imposée par la définition d'un graphe solution (car la paire $(\{c1,c2,c4\}, \{x,y,z\})$ forme une cellule de contrainte selon la définition 8.3, puisque on a la condition $\forall X \subseteq \{x,y,z\}$ $Card(X) \leq \sum_{c \in \Gamma p(X)} Inf(m(c), n_X(c))$ vrai).

Le graphe solution de cette hiérarchie est obtenu en décomposant p_0 en un ensemble de cellules comme cela est montré dans la figure 70. L'idée clé utilisée pour réaliser cette décomposition est de trouver une distribution parfaite des variables sur les contraintes dans la cellule. Cette distribution doit considérer en priorité les contraintes étiquetées par des étiquettes plus fortes que l'*étiquette-voyageuse* de cette cellule[2] (voir exemple plus bas). Par conséquent, on aboutit à la formation de cellules[3] à partir de ces contraintes et ces variables. Comme dans la figure 70, les contraintes étiquetées par *forte* (c.à.d c_1, c_2 et c_4) sont considérées en premier une par une.

1. Création d'une nouvelle cellule contenant cette contrainte et calcul de l'*étiquette-interne* de cette cellule créée
2. Quitte à laisser les contraintes ayant des étiquettes égales à l'*étiquette-voyageuse* non satisfaites.
3. Selon la définition 8.3.

Ensuite, chacune de ces contraintes est associée[1] à une variable[2] liée par cette contrainte et une cellule est formée. On obtient ainsi les trois cellules p_4, p_5 et p_2. Les contraintes restantes c_3 et c_5 forment chacune une cellule qualifiée de sur-contrainte. Ces dernières seront peut-être par la suite fusionnées à d'autres cellules du graphe par la procédure *Ajout-contrainte*.

FIGURE 67 : Graphe solution ayant une seule cellule

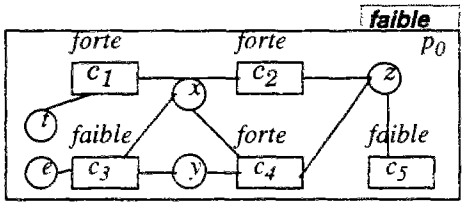


FIGURE 68 : Ajout d'une contrainte à 2-sorties

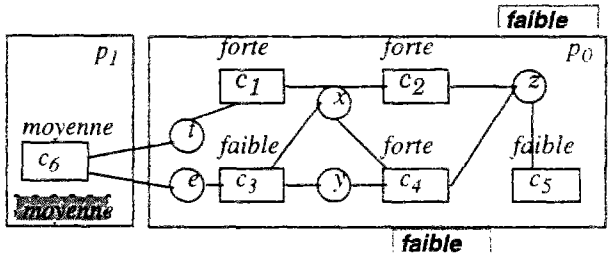


FIGURE 69 : Graphe non solution

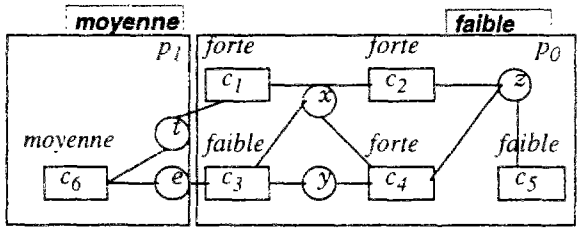
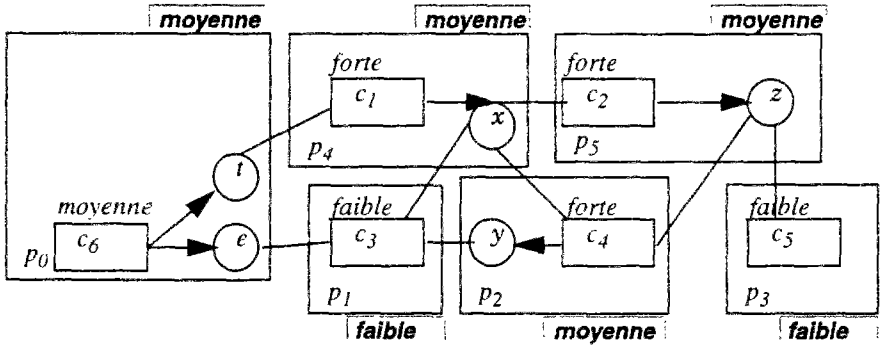


FIGURE 70 : Décomposition d'une cellule et aboutissement au graphe solution



1. Cette association est faite de telle sorte qu'elle soit parfaite. On aurait pu associer la variable x à la contrainte c_3 et la variable z à la contrainte c_5, mais ceci ne constituerait pas une association parfaite puisque la contrainte c_2 n'aurait pas de variable qui lui serait associée. Ou encore, on aurait pu associer les variable x et z à la contrainte c_3 et la variable y à la contrainte c_5, mais ceci ne serait pas bon puisque la paire $(\{x,z\},\{c_3\})$ ne formerait pas une cellule d'après la définition 8.3. etc .
2. Puisqu'on a supposé que ces contraintes sont uni-sortie.

La figure 71 décrit la procédure qui décompose une cellule contenant plusieurs contraintes en un ensemble de petites cellules[1]. Cette procédure est paramétrée par la cellule à décomposer *cl* et par l'*étiquette-voyageuse evm* de cette cellule. Au départ, on isole l'ensemble des contraintes de *cl* telles que chacune de ces contraintes est étiquetée par une étiquette plus forte que *evm* (ligne 1; figure 71). Après cette opération, on construit un ensemble de cellules minimales à partir de l'ensemble des contraintes isolées et l'ensemble des variables dans *cl* (ligne 2; figure 71). Pour chacune des cellules construites, on inverse la dépendance entre contraintes et variables dans cette cellule (ligne 3; figure 71).

La construction de cet ensemble de cellules minimales repose sur une distribution parfaite de l'ensemble des variables[2] dans *cl* à l'ensemble des contraintes isolées et sur la définition 8.3 qui modélise une cellule. La procédure considère ensuite les contraintes restantes dans *cl* (c.à.d ayant des étiquettes équivalentes à *evm*) (ligne 4; figure 71). Pour chacune de ces contraintes, si les variables auxquelles elle est attachée dans *cl* ont été utilisées dans la construction de l'ensemble des cellules minimales, alors une nouvelle cellule serait formée et comporterait cette contrainte (ligne 9; figure 71). Dans le cas contraire, cette contrainte serait considérée avec les variables restantes pour la formation d'une cellule (lignes 7,8; figure 71).

La définition 8.3 qui modélise une cellule de contraintes garantit qu'il ne reste pas de variables indéterminées après la décomposition d'une cellule contenant plusieurs contraintes. Dans le cas où une cellule ne satisfait pas la troisième condition de la définition 8.5 (c.à.d. pas de dépendance cyclique entre les cellules) ou elle ne satisfait pas la deuxième condition de la définition 8.9 (c.à.d. cellule sur-contrainte ayant une cellule adjacente dont l'*étiquette-voyageuse* est plus faible que l'*étiquette-interne* de cette cellule sur-contrainte) cette cellule sera fusionnée avec une cellule du graphe. On prouve que le résultat de cette fusion est une cellule (voir preuve de terminaison ci-dessous).

FIGURE 71 : Décomposition d'une cellule

```
Décomposer(cl, evm)
1 cs ← {c / c ∈ C_cl ∧ étiquette de c est plus forte que evm} .
2 Construire un ensemble de cellules minimales à partir de cs et de V_cl
3 Inverser la dépendance entre contrainte et variable pour chacune des cellules construites
4 cs' ← {c / c ∈ C_cl ∧ étiquette de c est égale à evm}
5 Pour chaque c ∈ cs' faire :
6   S'il reste des variables contraintes par c non utilisées dans le pas 2 alors
7     Crée une cellule à partir de ces variables et de c
8     Inverser la dépendance entre c et les variables dans la cellule crée
    Sinon
9     Crée une cellule contenant c
  Fin-Pour
```

Preuve de terminaison

On peut se poser la question de la terminaison de cet algorithme, c'est-à-dire lors de la fusion de deux cellules formant une dépendance cyclique, est ce que le résultat est une cellule ? La réponse est oui. Dans ce qui suit, on prouvera d'une manière générale que la fusion de deux cellules dont une est adjacente à l'autre résulte en une cellule.

1. Ceci permet aussi une efficacité remarquable lors de la résolution du graphe solution obtenu.
2. Il s'agit de l'ensemble des variables dans *cl* qui sont attachées à ces contraintes.

Soient p et p' deux cellules de contraintes telles que : $p = (V_p, C_p)$ et $p = (V_{p'}, C_{p'})$.

On suppose que $C_p \neq \emptyset$ et $C_{p'} \neq \emptyset$ et que p et p' sont connexes et disjointes entre elles.

Montrons que $p \cup p'$ est une cellule.

Soit $z \subseteq V_p \cup V_{p'}$, on note par $z_p = z \cap V_p$ et $z_{p'} = z \cap V_{p'}$.

On rappelle que $\Gamma(X) = \{c \in C / \exists v \in X \wedge (v,c) \in E\}$ et que $\Gamma_p(X) = \Gamma(X) \cap C_p$.

Il est facile de voir que :

$$\Gamma_{p \cup p'}(z) = \Gamma_p(z_p) \cup \Gamma_{p'}(z_{p'}) \cup ((\Gamma(z_p) \cap C_{p'}) \setminus \Gamma_{p'}(z_{p'})) \cup ((\Gamma(z_{p'}) \cap C_p) \setminus \Gamma_p(z_p)).$$

Et donc :

$$(*)\ \mu(\Gamma_{p \cup p'}(z)) = \sum_{c \in \Gamma p(zp)} Inf(m(c), n_z(c)) + \sum_{c \in \Gamma p'(zp')} Inf(m(c), n_z(c)) + \sum_{c \in ((\Gamma(zp) \cap Cp') \setminus \Gamma p'(zp'))} Inf(m(c), n_z(c)) + \sum_{c \in ((\Gamma(zp') \cap Cp) \setminus \Gamma p(zp))} Inf(m(c), n_z(c)).$$

(dans les formules $Inf(m(c), n_z(c))$ de cette somme on examine $n_z(c)$ car on se place par rapport à $p \cup p'$).

Maintenant il est intéressant de faire les trois remarques suivantes :

(1) $n_{zp}(c) \leq n_z(c)$ puisque $z_p \subset z$ et par conséquent $Inf(m(c), n_{zp}(c)) \leq Inf(m(c), n_z(c))$.

$n_{zp'}(c) \leq n_z(c)$ puisque $z_{p'} \subset z$ et par conséquent $Inf(m(c), n_{zp'}(c)) \leq Inf(m(c), n_z(c))$.

(2) $Card(z_p) \leq \sum_{c \in \Gamma p(zp)} Inf(m(c), n_{zp}(c))$ puisque p est une cellule.

$Card(z_{p'}) \leq \sum_{c \in \Gamma p'(zp')} Inf(m(c), n_{zp'}(c))$ puisque p' est une cellule.

(3) $Card(z) = Card(z_p) + Card(z_{p'})$.

D'après (3) et (2) on aboutit à :

(4) $Card(z) \leq \mu(\Gamma_p(z_p)) + \mu(\Gamma_{p'}(z_{p'}))$

D'après (1) on aboutit à :

(5) $\mu(\Gamma_p(z_p)) + \mu(\Gamma_{p'}(z_{p'})) \leq \sum_{c \in \Gamma p(zp)} Inf(m(c), n_z(c)) + \sum_{c \in \Gamma p'(zp')} Inf(m(c), n_z(c))$.

D'après (*) on a:

(6) $\sum_{c \in \Gamma p(zp)} Inf(m(c), n_z(c)) + \sum_{c \in \Gamma p'(zp')} Inf(m(c), n_z(c)) \leq \mu(\Gamma_{p \cup p'}(z))$.

Par transitivité entre (4) (5) et (6) on aboutit à :

$Card(z) \leq \mu(\Gamma_{p \cup p'}(z))$. *Ceci pour tout* $z \subseteq V_p \cup V_{p'}$. D'où on a bien $p \cup p'$ *une cellule.*

Exemple d'ajout d'une contrainte:

L'exemple de la figure 72 montre le déroulement de la procédure d'ajout d'une contrainte. Initialement, on suppose l'existence d'un graphe solution montré par la figure 72.a. Lorsque la contrainte c_8 est ajoutée à la hiérarchie, la procédure d'ajout décrite avant procède comme suit:

- La cellule p_7 est créée contenant la contrainte c_8 (figure 72.b) et l'étiquette de la contrainte victime est déterminée.

- La variable z est enlevée de la cellule p_6 et ajoutée à la cellule p_7 (figure 72.c).

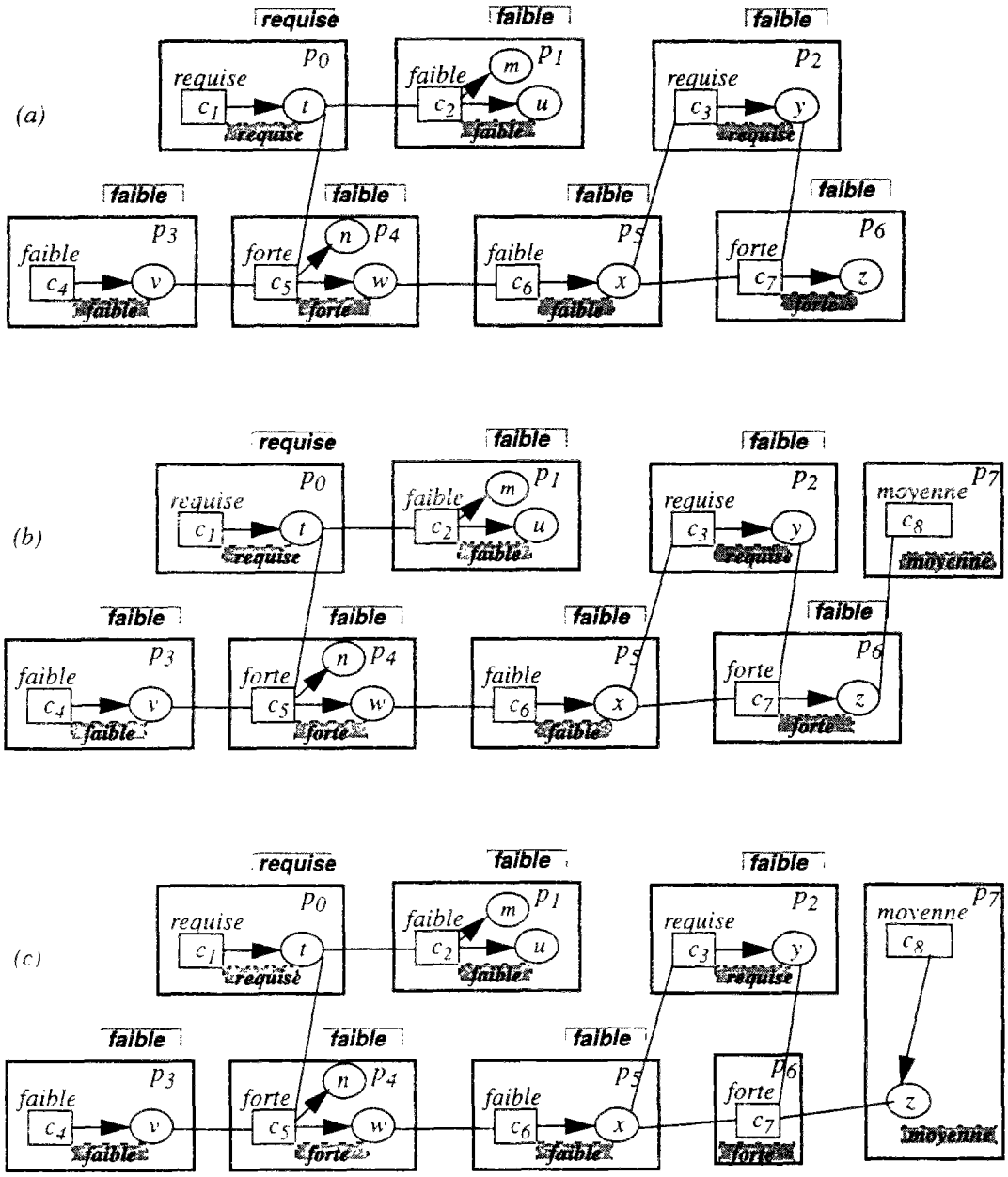
- La variable x est enlevée de la cellule p_5 et ajoutée à la cellule p_6 (figure 72.d). La contrainte c_6 est déterminée à être la contrainte victime.

- Les cellules p_2 et p_6 sont fusionnées puisqu'elles forment une dépendance cyclique (figure 72.e).

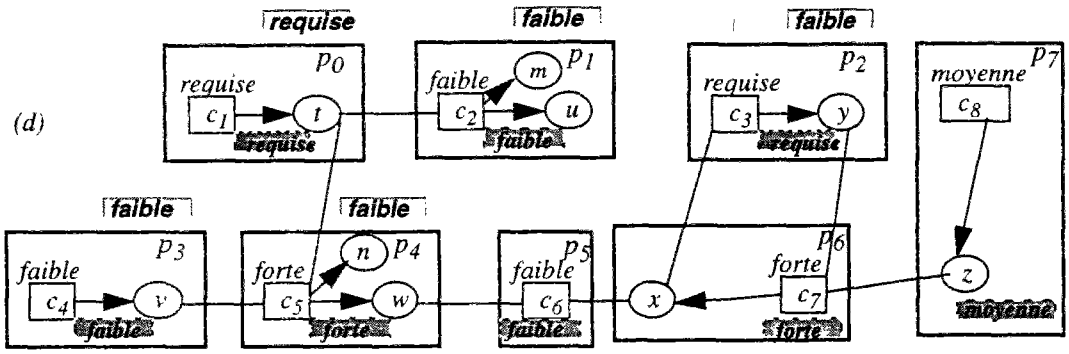
- Les étiquettes-voyageuses des cellules sont mises à jour (figure 72.f).

- Puisque la cellule p_5 est sur-contrainte, elle est fusionnée avec les cellules p_4 et p_3 qui possèdent les mêmes *étiquettes-voyageuses* (figure 72.g).

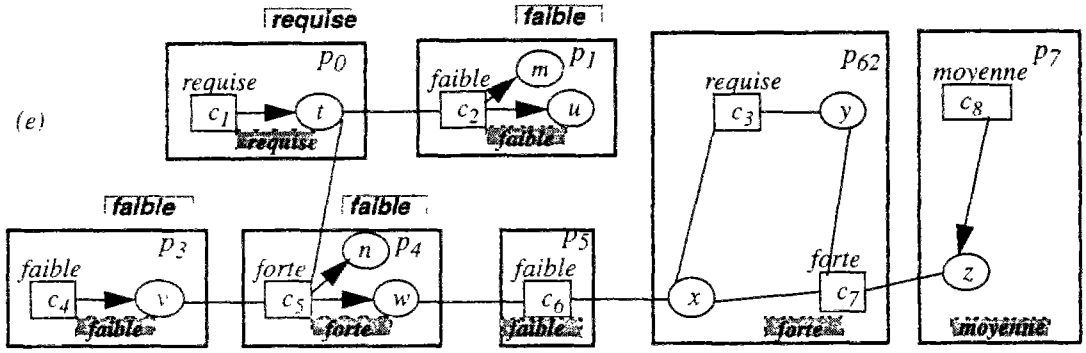
FIGURE 72 : Ajout d'une contrainte à un graphe solution



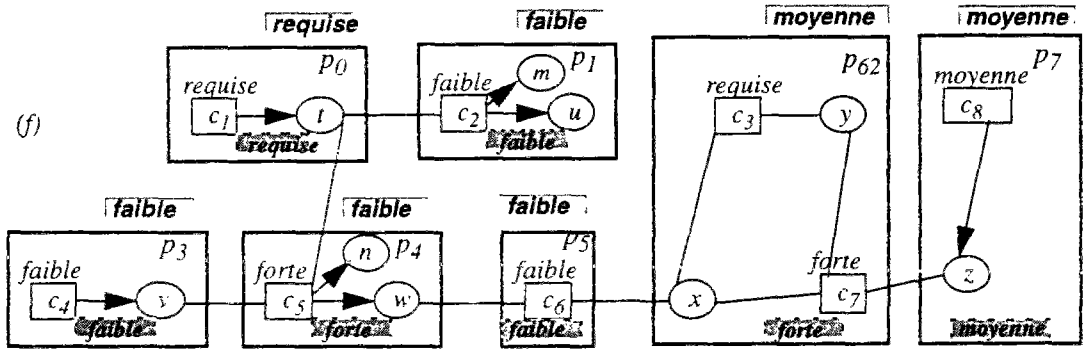
(d)


requise
faible
faible
p0
p1
p2
p7
requise
faible
requise
moyenne
c1
t
c2
m
u
c3
y
c8
faible
faible
p3
p4
p5
p6
faible
forte
faible
forte
c4
v
c5
n
w
c6
x
c7
z
faible
forte
faible
forte
moyenne

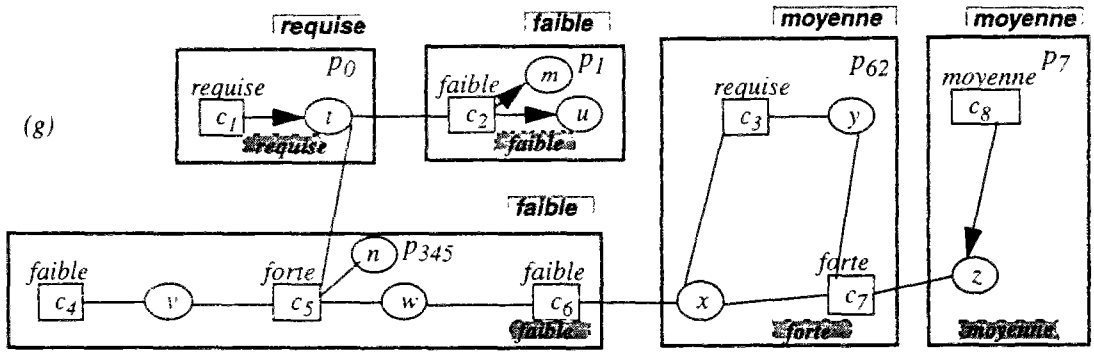
(e)


requise
faible
p0
p1
p62
p7
requise
faible
requise
moyenne
c1
t
c2
m
u
c3
y
c8
faible
faible
p3
p4
p5
faible
forte
faible
forte
c4
v
c5
n
w
c6
x
c7
z
faible
forte
faible
forte
moyenne

(f)


requise
faible
moyenne
moyenne
p0
p1
p62
p7
requise
faible
requise
moyenne
c1
t
c2
m
u
c3
y
c8
faible
faible
faible
p3
p4
p5
faible
forte
faible
forte
c4
v
c5
n
w
c6
x
c7
z
faible
forte
faible
forte
moyenne

(g)


requise
faible
moyenne
moyenne
p0
p1
p62
p7
requise
faible
requise
moyenne
c1
t
c2
m
u
c3
y
c8
faible
p345
faible
forte
faible
c4
v
c5
n
w
c6
x
c7
z
faible
forte
moyenne

8.4.2 Retrait-contrainte

La procédure retrait-contrainte se déroule de la même manière que la procédure ajout-contrainte. Le retrait d'une contrainte de la hiérarchie est réalisé par le retrait de cette contrainte de la cellule où elle se trouve. Ceci peut créer un changement de l'*étiquette-interne* de la cellule et par conséquent un changement de l'*étiquette-voyageuse* de cette cellule (c.à.d que l'on peut être amené à faire décroître l'erreur des contraintes ayant des étiquettes équivalentes, ceci en inversant les dépendances entre la cellule contenant ces contraintes et la cellule qui contenait la contrainte retirée).

L'exemple de la figure 73 montre le déroulement de la procédure du retrait de la contrainte c_8. Lorsque la contrainte c_8 est retirée de la hiérarchie, la procédure de retrait procède comme suit:

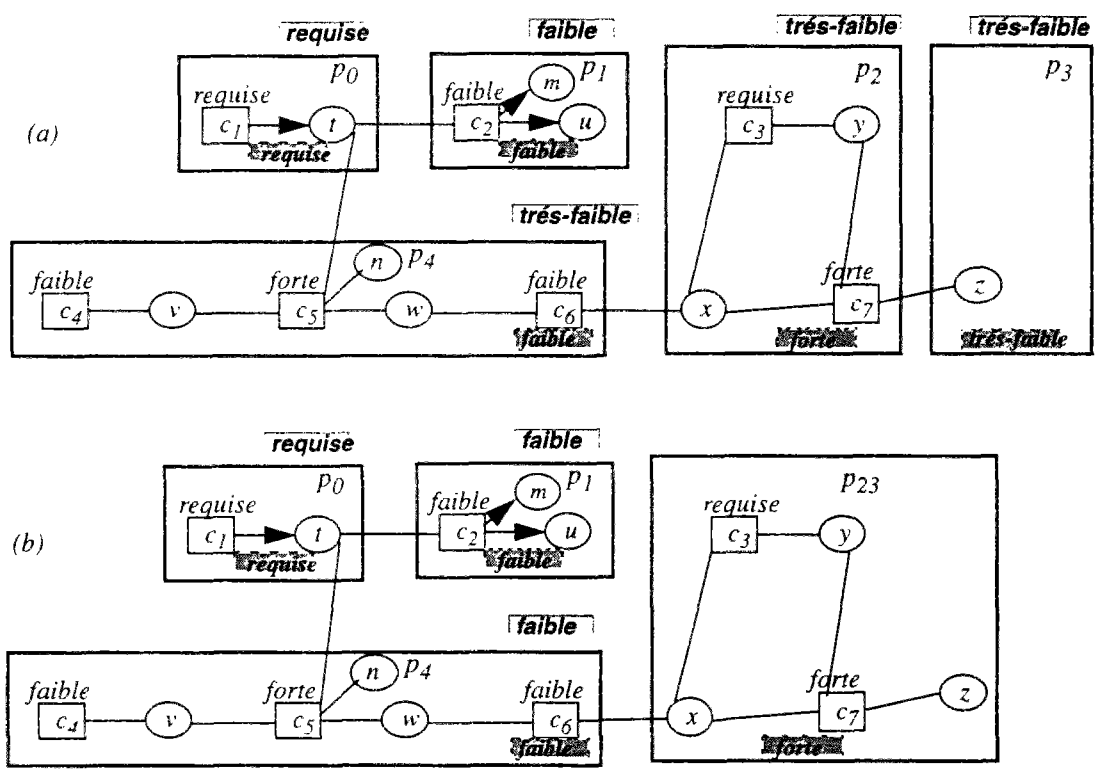
- La contrainte c_8 est retirée de la cellule p_3 et l'*étiquette-interne* de cette cellule est redéterminée (d'après la définition de l'étiquette-interne d'une cellule, cette étiquette a pour valeur *très-faible)*. Les étiquettes-voyageuses des autres cellules sont mises à jour (figure 73.a).

- La cellule p_2 est fusionnée à la cellule p_3 et donc il y a formation de p_{23}. Cette dernière ne constitue pas une cellule d'après la définition 8.3 (figure 73.b).

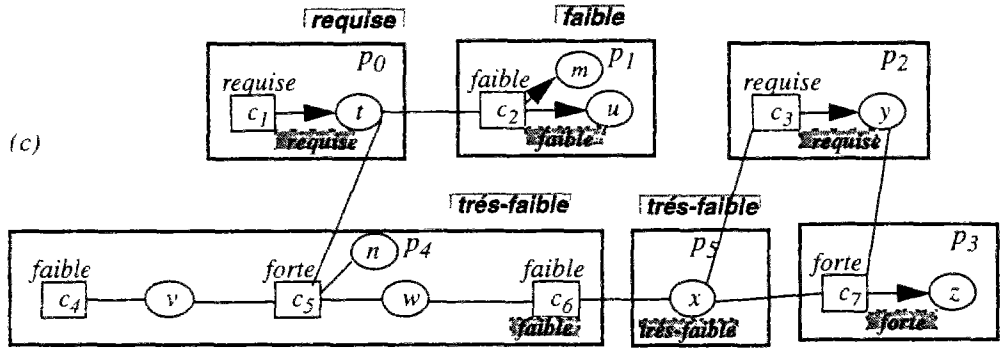
- Dans le but d'obtenir un graphe-solution, la cellule p_{23} est décomposée en trois cellules: p_2, p_3 et p_5. les *étiquettes-voyageuses* des cellules sont recalculées (figure 73.c).

- La cellule sur-contrainte p_4 est fusionnée à la cellule p_5 (puisqu'elles possèdent des étiquettes-voyageuses équivalentes) (figure 73.d). Le graphe obtenu satisfait les conditions d'un graphe solution.

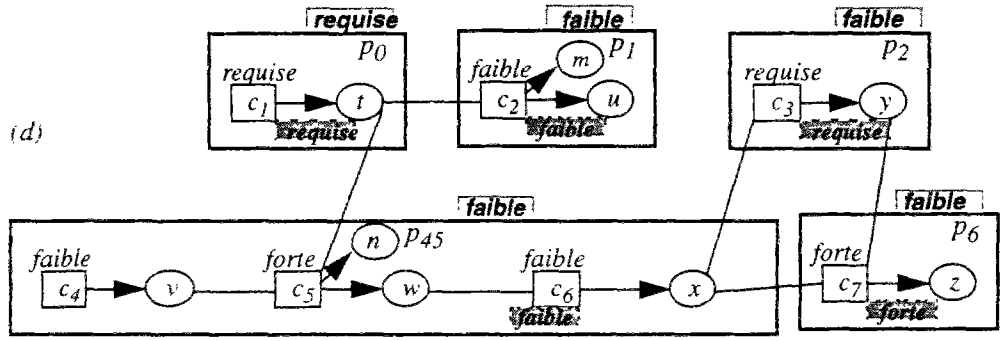
FIGURE 73 : Retrait d'une contrainte d'un graphe solution



(c)


requise
faible
requise
faible
requise
p_0
p_1
p_2
c_1
t
c_2
m
u
c_3
y
très-faible
très-faible
faible
forte
faible
p_4
p_5
p_3
c_4
v
c_5
n
w
c_6
x
c_7
z

(d)


requise
faible
faible
requise
faible
p_0
p_1
p_2
c_1
t
c_2
m
u
c_3
y
faible
faible
p_{45}
p_6
faible
forte
faible
c_4
v
c_5
n
w
c_6
x
c_7
z

Synthèse du chapitre

Comme dans les chapitres précédents dans ce mémoire, l'objet de ce chapitre a été de faire une analyse plus poussée de la théorie des hiérarchies de contraintes. Ici on considère des hiérarchies où les modes de combinaison peuvent varier selon les niveaux. Ce chapitre présente également une autre vue sur les graphes solutions à une hiérarchie de contraintes. La résolution de ces graphes solutions permet l'obtention des valuations de l'ensemble S qui sont des solutions à la hiérarchie de contraintes considérée.

Certains sous-ensembles de contraintes ne peuvent pas être résolus par une simple utilisation de l'algorithme de la propagation locale. Ces sous-ensembles de contraintes doivent être résolus ensemble. Par exemple, un sous-ensemble de contraintes dont les méthodes forment un circuit ou encore, un sous-ensemble de contraintes associées à un des deux critères globaux de comparaison (*MCM* ou *PCM*) ayant leurs méthodes en conflit. Le fait de considérer des cellules de contraintes permet de localiser ces sous-ensembles et donc de pouvoir les résoudre localement en utilisant des résolveurs spécifiques.

Un autre intérêt de la décomposition du graphe en cellules de contraintes est de ne pas être obligé de résoudre toutes les contraintes du système lors d'une perturbation. En moyenne, avec la décomposition, seul un sous-ensemble de cellules du graphe est réexaminé. Cependant, il n'est pas très intéressant d'utiliser cette modélisation pour des applications non incrémentales où l'utilisateur a besoin d'avoir une solution à son système de contraintes et non une solution à l'ajout de chaque contrainte. Puisque ici, pour chaque contrainte introduite, il y a une réorganisation d'un ensemble de cellules afin d'aboutir à un graphe-solution.

Ce chapitre présente une nouvelle modélisation des cellules de contraintes. Cette modélisation a été conçue dans le but de pouvoir manipuler des contraintes multi-sorties. Outre leur intérêt dans certains applications réelles [Ros94, San94, Hil93], elles permettent de donner un aspect optimal et élégant pour l'expressivité des contraintes du problème traité.

L'obtention d'un graphe solution après ajout ou retrait d'une contrainte est réalisée en considérant un graphe admissible de cellules de contraintes ainsi que des techniques d'optimisation qui sont basées sur la propagation locale (c.à.d *étiquette-voyageuse* et *étiquette-interne* d'une cellule). Ces dernières permettent de localiser les cellules à modifier en répercutant les effets de l'ajout ou du retrait d'une contrainte.

La résolution globale de la hiérarchie consiste en la résolution locale de chacune des cellules du graphe solution de cette hiérarchie et en la propagation des valeurs obtenues de cellule en cellule dans le graphe.

On a prouvé la correction et la terminaison de l'algorithme présenté dans ce chapitre. Cet algorithme constitue une simple modélisation afin de faire coopérer des résolveurs spécifiques. Ces résolveurs spécifiques doivent être conçus selon les instances de comparateurs (les critères) utilisés dans l'application réelle traitée.

L'utilisation de ce résolveur sur un graphe contenant des gros cycles n'est pas non plus très souhaitable puisque le graphe solution est réduit à un petit nombre de cellules contenant beaucoup de contraintes (du fait qu'on ne se permet pas d'avoir des dépendances cycliques entres les cellules) et donc il y a une perte d'efficacité au niveau de la résolution globale du graphe.

Le tableau de la figure 74 situe cet algorithme par rapport aux autres algorithmes vus dans les chapitres précédents:

FIGURE 74 : Comparaison entre résolveurs basés sur la propagation locale.

utilise / résolveurs	contraintes ont plusieurs variables en sortie	contraintes sont multi-directionnelles	contraintes sont dans une hiérarchie	compatible avec résolveur de cycle	utilise un critère global	unifie plusieurs types de comparateurs
Blue	–	+	–	–	–	–
DeltaBlue	+ –	– +	+	–	–	–
SkyBlue	+	+	+	+	–	–
QuickPlan	+	+	+	+	–	–
Houria	+	+	+	– +	+	–
LSCS	–	+	+	+	+	+
Latif	+	+	+	+	+	+