Recherche Opérationnelle
Programmation Par Contraintes
Équivalences PPC - BIP

Optimisation Combinatoire
(Version Élèves)

Alexandre Saidi
École Centrale de Lyon
Département Mathématiques-Informatique
UMR LIRIS - CNRS

Novembre 2015

CSP

- La Programmation Par (avec) Contraintes : PPC (Constraint Satisfaction Programming : CSP).

		5			6	7		1
	6		7	4			8	
1						3	2	
	4		1	6				
8								5
3		6					1	
6						1		
	1		8	2	5		7	3
7	3							

CSP (suite)

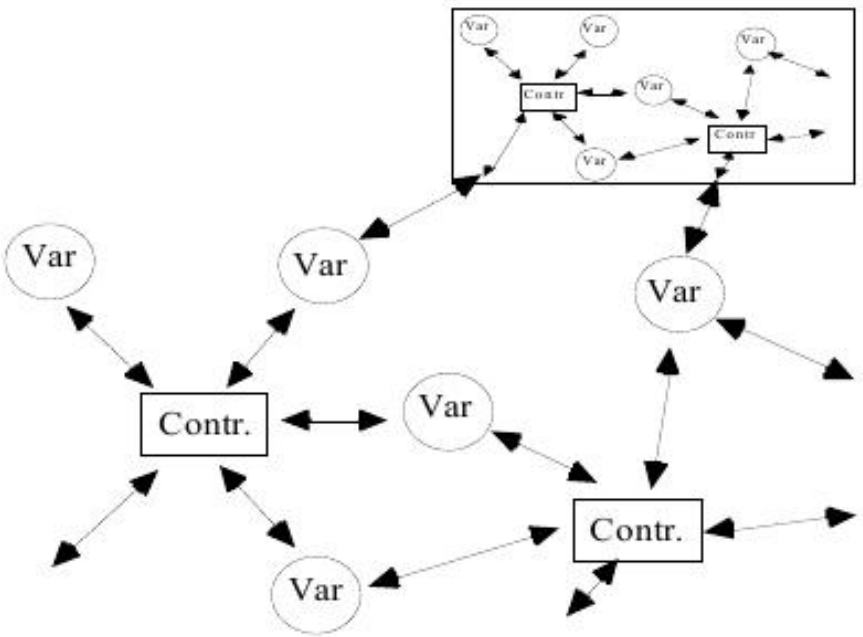
Principes de la programmation par contraintes :

- Un modèle : (V, D, C)
- Raisonnement par élimination
  ➙ on ne cherche pas directement une valeur pour les variables mais plutôt les valeurs qui peuvent être prises par une variable :
  ➙ Réduction du domaine par **propagation**
- Raisonnement local : contraintes considérées indépendamment
  ➙ vérifier choix cohérents : tests satisfiabilité
  ➙ éliminer valeurs impossibles : filtrage
- Transmission des déductions aux autres régions
  ➙ les contraintes communiquent et interagissent par l'intermédiaire des variables,
  ➙ lorsque le domaine courant d'une variable change, toutes les contraintes dans lesquelles elles apparaissent sont réveillées : propagation

CSP (suite)

Un exemple simple (variables, domaines, contraintes) :

- Domaine des variables
  - ➙ $x, y \in \{0, 1, ...7\}$
- Contraintes
  - ➙ $x \times y = 6$
  - ➙ $x + y = 5$
  - ➙ $x > y$



CSP (suite)

Qu'est ce que la PPC :

- Outil informatique pour résoudre des problèmes combinatoires

- Minimise l'étape de conception :
  modélisation facile, langage proche des concepts des utilisateurs

- Contraindre = maintenir dans les limites
  ➙ permet d'éliminer rapidement des solutions non réalisables

- Bien séparer modélisation et résolution

- Notion naturelle de contraintes :
  ➙ affecter stages étudiants
  ➙ ranger des objets dans une boite
  ➙ planifier le trafic aérien
  ➙ établir un menu équilibré

- Le **Quoi**, le **Comment** : indépendance vs. la résolution (Algos)

CSP (suite)

Plus formellement :

- **CSP** : ensemble de problèmes définis par des contraintes
  ➙ CSP consiste à chercher une solution respectant ces contraintes

- **Résolution d'un CSP combinatoire :**
  ➙ On envisage un très grand nombre de combinaisons avant d'en trouver une qui satisfait toutes les contraintes
  ➙ Peut être Trop long (selon la complexité du problème)
  ➙ Introduire des "raisonnements" ou "heuristiques" pour réduire la combinatoire et orienter la recherche

- **Solveur de contraintes :**
  ➙ on décrit les contraintes ;
  ➙ le solveur prend en charge automatiquement la résolution

CSP (suite)

Quelques exemple d'Applications de PPC :

- Modélisation de gestion de projet
- Puzzle combinatoire
- Conception de matériels informatique
  - ➙ vérification/simplification de circuits, connexions des couches de circuits
  - ➙ moins efficace que le code dédié, mais plus flexible
- Placement d'objets
  - ➙ placement des containers dans un port
  - ➙ remplissage des containers
- Problèmes de découpage
  - ➙ Minimiser les pertes lors de la découpe du papier, du verre, du bois, du métal, etc.
- Allocation d'espace
  - ➙ Portes pour les avions
  - ➙ Quais pour les trains ou les bateaux

CSP (suite)

- Allocation de fréquences
  - ➙ Trouver des fréquences radio pour les cellulaires, les communications radios, l'armée, etc.
- Ordonnancement de la production
  - ➙ Planifier des taches sur des machines dans une usine
  - ➙ Plus important succès de la PPC
  - ➙ Librairies dédiées à l'ordonnancement (ex. ILOG Scheduler)
- Conception d'horaires académiques
  - ➙ Planifier l'horaire des cours ou des examens, en tenant compte des différentes ressources (étudiants, professeurs, locaux)
- Tournée de véhicules
  - ➙ Confection de routes sujet à beaucoup de contraintes.
  - ➙ Librairies spécialisées, contraintes dédiées
- Construction d'Emploi du Temps du personnel
  - ➙ Affecter les gens à des tâches précises
  - ➙ Santé, commerce de détail, usine, etc.
  - ➙ Permet de modéliser les contraintes complexes

CSP (suite)

Environnements PPC

CSP(X) : avec $X \in \{\mathbb{N}, \mathbb{Z}, \mathbb{Q}, \mathbb{R}, \mathbb{B}, \mathbb{S}, ...\}$

- Les contraintes sont déclaratives et définissent des relations sur les variables

  $x - 3y = z \qquad (c = 1) \vee z = 2x + y \qquad (a \implies b) \Leftrightarrow (c \in E)$

  ➙ Une **relation** (équation, fonction, valeur et variable, ...)
  ➙ On ne s'occupe pas de l'algorithme de résolution.
  ➙ L'ordre des contraintes n'a (normalement) pas d'importance.
  ➙ Différentes consistances, propagation, ...

- Les contraintes sont unaires, binaires ..... n-aires
  ➙ Peuvent être définies par l'utilisateur

☞ Il est possible d'utiliser LP/MIP dans CSP.
☞ Il est possible d'utiliser les techniques venant de LP en CSP
  ➙ on peut toujours ajouter un plan de coupe, *B&B*, etc (ad hoc).

Résolutions dans les environnements PPC

I) Traduction en contraintes plus basiques :
2*X + 3*Y + 2 < Z, est traduit en contraintes plus simples.

• A base de ***X in R*** **:** *étant donné les domaines de* ***X, Y et Z,*** *procéder à une réduction des domaine*

• Réduction des bornes des domaines et / ou des valeurs des domaines :

```
'X=Y+C'(X,Y,C) :-      % LookAhead partiel
    X in min(Y)+C .. max(Y)+C, Y in min(X)-C .. max(X)-C.
```

Exemple : si dom(Y)= {1,3,4,7,9} et C=2, alors X peut prendre une valeur dans 3..12 (extrémums)

```
'X=Y+C'(X,Y,C) :-      % LookAhead complet
    X in dom(Y)+C ..Y in dom(X)-C.
```

Exemple : si dom(Y)= {1,3,4,7,9} et C=2, alors X peut prendre une valeur dans {3,5,6,9,11}

Résolutions dans les environnements PPC (suite)

- **Retour arrière** : *backtrack*.

- Résolution : les 3 tests de **consistances** + moteur **BT**

II) Résolution directe (sans simplification), par exemple à l'aide d'une version modifiée du Simplex.

➙ En particulier pour LP.

III) Résolution à base d'intervalles (Réels)

Résolutions dans les environnements PPC (suite)

Techniques de Consistance (Nœud, Arc et Chemin) :

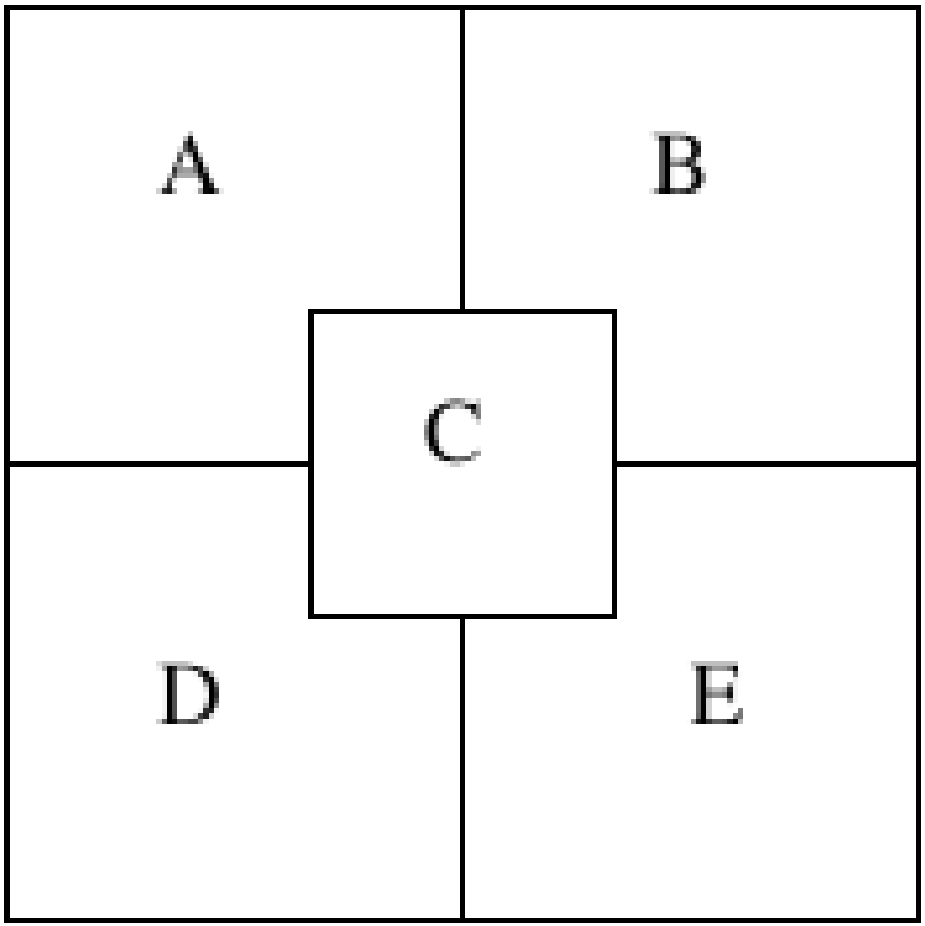
• **Il est possible de construire un solveur (complet) avec ces consistances + BT.**

• Soit X = {X1,X2,X3}, D = {D1, D2, D3}, D = {1,2,3}, C = {X1 > X2 > X3}

- **Node Consistance** :
  - ➙ Aucune simplification, les variables gardent leurs domaines.
  - ➙ Les contraintes des domaines sont vérifiées.
    Exemple : **X in 1..3, X>3** vérifiée n'est pas valide.
- **Arc Consistance :**
  - ➙ Entre X1 et X2 ⇒ X1={2,3}, X2={1,2}.
    Puis indépendamment pour X2 et X3.
- **Path Consistance : (couteux)**
  - ➙ On peut simplifier en : X1=3, X2=2, X3=1

Résolutions dans les environnements PPC (suite)

- Schéma BT / propagation



Quelques Contraintes globales PPC

Exemples d'outils caractéristiques de PPC

➙ Opérateurs Logiques :
ET, OU, NOT, EXISTS, FORALL

➙ Opérateurs Conditionnels :
if-then, if-then-else, $\Longrightarrow$, $\Leftrightarrow$,

➙ Opérateurs de Comptage :
count, atmost, atleast, exactly,...

➙ Opérateurs mutuels :
alldiff, element, ...

➙ Contraintes globales (Minmax, Cumulative, ...)

Quelques Contraintes globales PPC (suite)

Une contrainte globale importante (en PPC) : **element**

• $element(I, Liste, X)$ contraint la variable X à être égale à la I^{eme} variable (depuis 1) de Liste.
  ➙ $I \in 1..n$ peut donc être un indice

Exemples :

• Une contrainte comme $C_Y \leq 5$ peut être implantée par :
  $Z \leq 5$
  $element(Y, (C_1, \cdots C_n), Z)$
  ➙ Qui affectera (équationnel) à Z la Y^{eme} valeur dans la liste $(C_1, \cdots C_n)$.

• De même, la contrainte $X_Y \leq 5$ peut être implantée par :
  $Z \leq 5$
  $element(Y, (X_1, \cdots X_n), Z)$
  ➙ *element* ajoute ici la contrainte $Z = X_Y$.
  ➙ Son utilisation est légèrement différente de la précédente.

Quelques Contraintes globales PPC (suite)

La contrainte Cumulative :

• Utilisée pour la planification de toutes sortes.
• En particulier avec des contraintes de ressources.

• Exemple : $s_1, .., s_n$ pour n tâches à planifier (s_i : date de début de t_i) avec des durées respectives $d_1, ..., d_n$ lesquelles tâches réclament des ressources respectives.

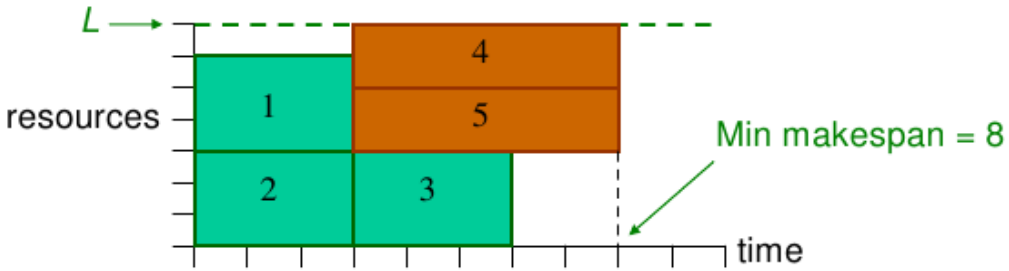
➙ Le total des ressources consommées par ces tâches ne doit pas dépasser à aucun moment la limite L.

$$cumulative((s_1, .., s_n), (d_1, ..., d_n), (r_1, ..., r_n), L)$$

Exemple 1 :

- Minimiser le temps total (*makespan*) de réalisation des taches sans deadline.

$$
\begin{array}{ll}
min & Z \\
s.t. & cumulative((t_1, .., t_5 \\
& (3,3,3,5,5), (3,3,3,2,2) \\
& Z \geq t_1 + 3 \\
& \ldots\ldots \\
& Z \geq t_5 + 2
\end{array}
$$



Avec :

$(t_1, .., t_5)$: date de début des tâches

$(3,3,3,5,5)$: durées des tâches

$(3,3,3,2,2)$ ressources utilisées par les tâches

➞ Ne jamais dépasser la limite des ressources 7

➞ On a le total des durées = 8 (vs la somme des durées = 19)

Exemple 1 : (suite)

Exemple 2 (du manuel OPL):

Les données du problème :

- Planifier 34 containers sur un bateau en un temps minimum (min *makespan*).
- Le chargement de chaque container nécessite un certain temps et un certains nombre d'ouvriers (cf. tableau).
- On dispose de 8 ouvriers.

Container	Durée	Nb. Ouvriers
1	3	4
2	4	4
3	4	3
4	6	4
5	5	5
6	2	5
7	3	4
8	4	3
9	3	4
10	2	8
11	3	4
12	2	5
13	1	4
14	5	3
15	2	3
16	3	3
17	2	6

Container	Durée	Nb. Ouvriers
18	2	7
19	1	4
20	1	4
21	1	4
22	2	4
23	4	7
24	5	8
25	2	8
26	1	3
27	1	3
28	2	6
29	1	8
30	3	3
31	2	3
32	1	3
33	2	3
34	2	3

Exemple 1 : (suite)

- Les contraintes de précédence :

1 → 2,4	11 →13	22 →23
2 →3	12 →13	23 →24
3 →5,7	13 →15,16	24 →25
4 →5	14 →15	25 →26,30,31,32
5 →6	15 →18	26 → 27
6 →8	16 →17	27 → 28
7 →8	17 →18	28 → 29
8 →9	18 →19	30 → 28
9 →10	18 →20,21	31 → 28
9 →14	19 →23	32 → 33
10 →11	20 → 23	33 → 34
10 →12	21 → 22	

$t_i \rightarrow t_j$: t_j commence après la fin de t_i.

- Le modèle CP :

$$
\begin{array}{ll}
min & Z \\
s.t. & cumulative((t_1, .., t_{34}), \\
 & (3, 4, 2, \cdots, 2), (4, 4, \cdots, 3), 8) \\
 & Z \geq t_1 + 3 \\
 & Z \geq t_2 + 4 \\
 & \cdots\cdots \\
 & t_2 \geq t_1 + 3 \\
 & t_4 \geq t_1 + 3 \\
 & \cdots\cdots
\end{array}
$$

Exemple de C définie par l'utilisateur

Dans un problème de tournée :

- *Un contrôleur visite des sites : il visite le site* ***S*** *le jour* ***J****.*
- *Il ne visite chaque site qu'une fois;*
- *Il peut visiter 2 sites en deux jours consécutifs ou espacés d'au plus 1 jour.*

Modélisation :

- Une variable par site S_i, l'ensemble donne Z=***Liste_des_sites***
- Domaine D de chaque élément de *Liste_des_sites*:
  ➙ les jours (1..7). ➙ ***Liste_des_sites in 1..7***

N.B. : certains environnements permettent des valeurs nominales (lundi, ...).

- Visiter chaque site une seule fois : ***alldifferent(Liste_des_sites)***
- Pour toute paire de sites S_1 et S_2 :

  S1 = S2 +X, I in 1..4, element(I, [-1,-2,1,2], X).

  Dès que S_1 prend une valeur, on élimine les valeurs incompatibles restantes pour S_2

Exemple de C définie par l'utilisateur (suite)

Rappel : $S_1 = S_2 + X,\ I :: 1..4,\ element(I, [-1, -2, 1, 2], X)$.

- ➙ On peut exprimer la même contraintes de différentes manières :
- ➙ Par $S_1 = S_2 + X, X \in \{-1, -2, 1, 2\}$
- ➙ Par $(S_1 = S_2 + X) \vee (S_1 = S_2 - X), X :: 1..2$
- ➙ Par $(S_1 = S_2 + 1) \vee (S_1 = S_2 + 2) \vee (S_1 = S_2 - 1) \vee (S_1 = S_2 - 2)$
- ➙ ...

• Il suffit ensuite de générer des valeurs pour les éléments de *Liste_des_site.*

```
visite(L) :- L :: 1..7, alldifferent(L), delais(L).

% les visites en jours consécutifs ou espacés d'un jour (autre solution).}
delais([]).
delais([X]):- !.
delais([X,Y{\textbar}L]) :-
        D :: [-2, -1, 1, 2],
        X #= Y + D,
        delais([Y | L]).

Ex. de solution : L = [1, 2, 3, 4, 5], L = [1, 2, 3, 4, 6]  ...
```

☞ $NbJours = NbSite$: pas besoin d'énumérer : la propagation fait le travail.

Modèle et Complexité

- Le choix du modèle influe sur la complexité de la solution.
- **Exemple N-reines avec N=8** (solution ci-contre)
- Les contraintes à satisfaire : ...

Choix de la représentation :

1- un ensemble S de 8 *variables* {X1..X8} chacune prenant une valeur dans l'intervalle D={A..H}.

➙ 8^8

2- un ensemble S de 8 variables {A..H} chacune prenant une valeur dans l'intervalle D={1..8}.

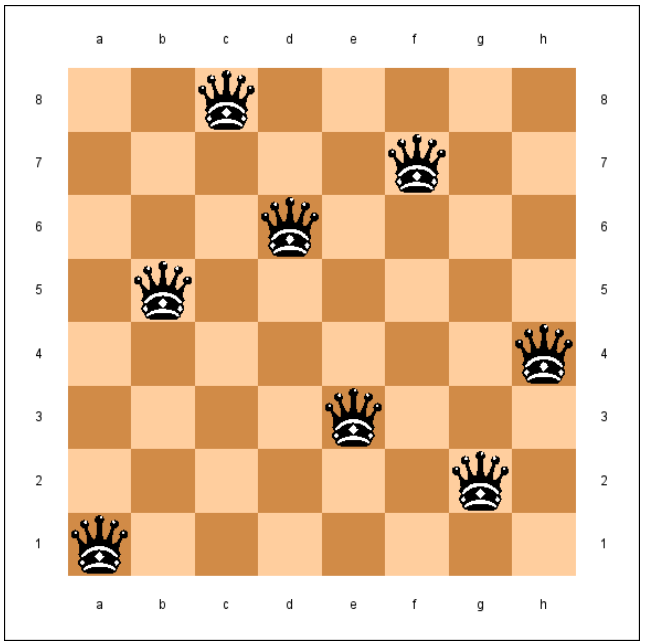
➙ 8^8

3- un ensemble S de 8 variables {V1..V8} chacune prenant une valeur dans l'intervalle D={1..64}.

➙ $64^8 = 8^{16}$

4- un ensemble S de 64 variables {E1..E64} chacune prenant une valeur dans l'intervalle D={0,1}.

➙ $2^{64} > 8^{21}$ ...



Choix selon la complexité $|D|^{|V|}$

Algèbre de Bool et LP/MIP

- OR ($\vee$) : modélise le choix entre plusieurs décisions :
  $Y_1 \vee Y_2 \vee ... \vee Y_j \vee ... \vee Y_n$
  ➟ et correspond à la relation algébrique
  $y_1 + y_2 + ...y_j + ... + y_n \geq 1$

- EXCLUSIVE OR ($\veebar$) : l'un ou l'autre (pas les deux : exclusif)
  $Y_1 \veebar Y_2 \veebar ... \veebar Y_j \veebar ... \veebar Yn$
  ➟ et correspond à la relation algébrique
  $y_1 + y_2 + ... + y_n = 1$

- AND ($\wedge$) : conjonction
  $Y_1 \wedge Y_2$ ➟ Devient $y_1 \geq 1, y_2 \geq 1$

- NEGATION ($\neg$) : nie une variable ou expression booléenne
  $\neg Y_1$ ou $\neg(Y_1 \vee Y_2)$
  ➟ Pas d'équivalence. Il suffit de nier
  ➟ Si le booléen Y (et donc $\neg Y$) existe, $\neg Y$ devient $(1 - y)$

Algèbre de Bool et LP/MIP (suite)

- UNE CLAUSE: doit être vraie par elle même
  $Y_1 \vee Y_2$ est une clause

- IMPLICATION ($\implies$) :
  $Y_1 \implies Y_2$
  1. écrire $Y_1 \implies Y_2$ sous forme disjonctive (avec $\vee$) :
     ➡ $\neg Y_1 \vee Y_2$
  2. remplacer les négations par -1, en remplaçant $\vee$ par un $+$
  (et y algébrique pour Y logique):
     ➡ $1 - y_1 + y_2$
  3. ajouter en partie droite un "vrai" :
     ➡ $1 - y_1 + y_2 \geq 1$
  4. Simplifier éventuellement :
     ➡ $y_1 - y_2 \leq 0$

☞ $Y_1 \implies Y_2$ devient donc $y_1 \leq y_2$

Algèbre de Bool et LP/MIP (suite)

- EQUIVALENCE ($\Leftrightarrow$)= double implications

  $Y1 \Leftrightarrow Y2$

  1. écrire les deux implications connectées avec un ($\wedge$) :

     $(\neg Y1 \wedge Y2) \wedge (\neg Y2 \wedge Y1)$

  2. remplacer les négations par -1, en remplaçant $\vee$ par un $+$

  (et y algébrique pour Y logique):

     $(1 - y_1 + y_2) \wedge (1 - y_2 + y_1)$

  3. Traiter le AND ($\wedge$) :

     $1 - y_1 + y_2$

     $1 - y_2 + y_1$

  4. Ajouter en RHS la valeur de vérité :

     $1 - y_1 + y_2 \geq 1$

     $1 - y_2 + y_1 \geq 1$

  5. Simplifier

     $y_1 - y_2 \leq 0$

     $y_2 - y_1 \leq 0$

➙ La seule solution faisable est $y_1 = y_2$

Algèbre de Bool et LP/MIP (suite)

- **if Cond Then E1 else E2**

est équivalent à :
Cond*E1+(1-Cond)*E2.

Remarques

- DISTRIBUTION DE OU ($\vee$) SUR ET ($\wedge$) :
  $(Y_1 \wedge Y_2) \vee Y_3$
  $(Y_1 \vee Y_3) \wedge (Y_2 \vee Y_3)$

- THEOREME de DeMORGAN :
  $\neg(Y_1 \wedge Y_2) \Leftrightarrow \neg Y_1 \vee \neg Y_2$
  $\neg(Y_1 \vee Y_2) \Leftrightarrow \neg Y_1 \wedge \neg Y_2$

Exemples de conversion

Exemple :
Soit la proposition $Y_1 \Leftrightarrow Y_2 \vee Y_3$.
Proposer une contraintes algébrique qui y correspond.

Solution :

- Pour $\Rightarrow$:
  $\neg Y_1 \vee Y_2 \vee Y_3$
  $1 - y_1 + y_2 + y_3 \geq 1$
  $-y_1 + y_2 + y_3 \geq 0$

- Pour $\Leftarrow$:
  $\neg(Y_2 \vee Y_3) \vee Y_1$ Demorgan
  $(\neg Y_2 \wedge \neg Y_3) \vee Y_1$ distribution de OU sur ET
  $(\neg Y_2 \vee Y_1) \wedge (\neg Y_3 \wedge Y_1)$ ensuite, un ET détache les expressions :
  $1 - y_2 + y_1 \geq 1$ et
  $1 - y_3 + y_1 \geq 1$
- Simplifications :
  $y_1 - y_2 \geq 0$
  $y_1 - y_3 \geq 0$

- On regroupe les 2 groupes :

$$\boxed{\begin{array}{l} -y_1 + y_2 + y_3 \geq 0 \\ y_1 - y_2 \geq 0 \\ y_1 - y_3 \geq 0 \end{array}}$$

Exemples de conversion (suite)

Un autre exemple :

$$(Y_1 \wedge Y_2) \implies (Y_3 \wedge Y_4)$$

➙ Donnera :

$$y_1 + y_2 - y_3 \leq 1 \qquad \text{et}$$
$$y_1 + y_2 - y_4 \leq 4$$

Traitement de la Disjonction

Les disjonctions sont très utiles à la modélisation des expressions conditionnelles dans les cas suivants :

• Relation entre des variables entiers et des variables continues ou des contraintes
• Relation entre des variables continues et d'autres variables continues ou des contraintes

Modèle conditionnel : modélise la validité d'une fonction sous différentes conditions.

Par exemple (une seule des contraintes vraie) :

Soit	$3x1 + 2x2 = 18$	si $0 \leq x1 \leq 3$
Ou	$x1 + 4x2 = 16$	si $3 < x1 \leq 8$

La transformation : ../..

Traitement de la Disjonction (suite)

La méthode d'**Enveloppe Convexe**

- On pose la disjonction en introduisant la booléenne Y

$$\begin{pmatrix} Y \\ 3x_1 + 2x_2 = 18 \\ 0 \leq x_1 \leq 3 \end{pmatrix} \quad \vee \quad \begin{pmatrix} \neg\, Y \\ x_1 + 4x_2 = 16 \\ 3 < x_1 \leq 8 \end{pmatrix}$$

Pour convertir cette expression via la méthode **enveloppe convexe** [Turkay &l al 1996] :
- Séparer les variables continues pour chaque disjonction (x_1 désagrégée en x_1^1 et x_1^2)
- Remplacer Y_i par la variable binaire y_i (et $\neg\, Y_i$ par $(1 - y_i)$)
- Répéter les contraintes avec les variables désagrégées
- Simplifier éventuellement.

On obtient l'ensemble de contraintes suivant (conjonction entre toutes) :

$$\begin{aligned} 3x_1^1 + 2x_2^1 &= 18y, & x_1^2 + 4x_2^2 &= 16(1-y), \\ x_1^1 &\leq 3y, & x_1^2 &> 3(1-y), \\ x_1^1 &\geq 0y, & x_1^2 &\leq 8(1-y), \\ x_1 &= x_1^1 + x_1^2, \\ x_2 &= x_2^1 + x_2^2, \\ x_1, x_2, x_1^1, x_1^2, x_2^1, x_2^2 &\geq 0, \\ y &\in \{0,1\} \end{aligned}$$

➞ La désagrégation des variables : exposant = 2 : 2 branches de la conjonction

➞ x_1 remplacée par x_1^1, x_2 par x_2^1 dans la 1e C

➞ x_1 remplacée par x_1^2, x_2 par x_2^2 dans la 2e C

Traitement de la Disjonction (suite)

La méthode Big-M (plus simple) :

En appliquant la méthode big-M on obtient la version MILP du problème :

- On pose la disjonction en introduisant la booléenne Y

$$\begin{pmatrix} \neg Y \\ 3x_1 + 2x_2 = 18 \\ 0 \leq x_1 \leq 3 \end{pmatrix} \quad \vee \quad \begin{pmatrix} Y \\ x_1 + 4x_2 = 16 \\ 3 < x_1 \leq 8 \end{pmatrix}$$

$3x1 + 2x2 \leq 18 + M(1-y)$,
$x1 \leq 3 + M(1-y)$,

$x1, x2 \geq 0, \quad y \in \{0,1\}$

$x1 + 4x2 \leq 16 + My$,
$x1 > 3 - My$, ☞ On note : $x1 > 3$ impose $-My$
$x1 \leq 8 + My$,

Les différences majeurs entre les méthodes **Enveloppe Convexe** et **Big-M** sont les suivantes :
- La méthode Enveloppe Convexe nécessite plus de variables que Big-M,
- Big-M nécessite la relaxation sur la contrainte d'égalité pour conserver les solutions,
- Big-M nécessite l'utilisation des paramètres supplémentaires M. Ces paramètres doivent être définis selon les données et il se peut que plusieurs M avec des valeurs différentes soient nécessaires.

☞ On n'ajoute pas $x \geq 0 + M(1-y)$ car si $x \geq 0$, on ne peut rien dire sur $0 + M(1-y)$:

pour $y = 0$, on ne peut imposer $x \geq M$.

Variables 0/1 et Big-M

A propos de Big-M dans un Modèle conditionnel :

Soit $3x1 + 2x2 \leq 18$ Seule une des deux contraintes à satisfaire
Ou $x1 + 4x2 \leq 16$ (selon certaines conditions)

- Le paramètre M doit être assez large (conjointement avec le booléen y)

$3x1 + 2x2 \leq 18 + My$
$x1 + 4x2 \leq 16 + M(1 - y)$

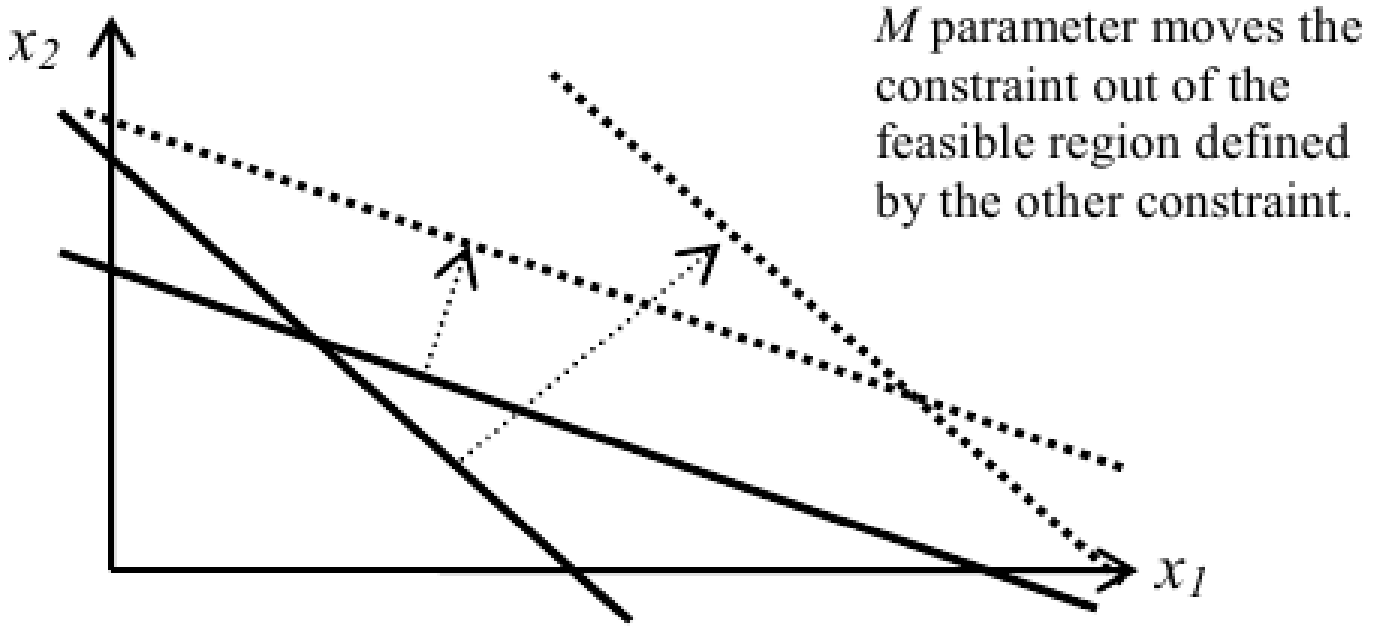
→ si $y = 0$ ➡ $3x1 + 2x2 \leq 18$ la 1e contrainte tient
$x1 + 4x2 \leq 16 + M$ un M grand rend la 2e contrainte redondante

→ si $y = 1$ ➡ $3x1 + 2x2 \leq 18 + M$ un M grand rend la 1e contrainte redondante
x1 + 4x2 ≤ 16 la 2e contrainte tient

M doit être tel que la région faisable ne coupera pas la seconde contrainte lorsque la seconde doit être vérifiée. ../..

Variables 0/1 et Big-M (suite)

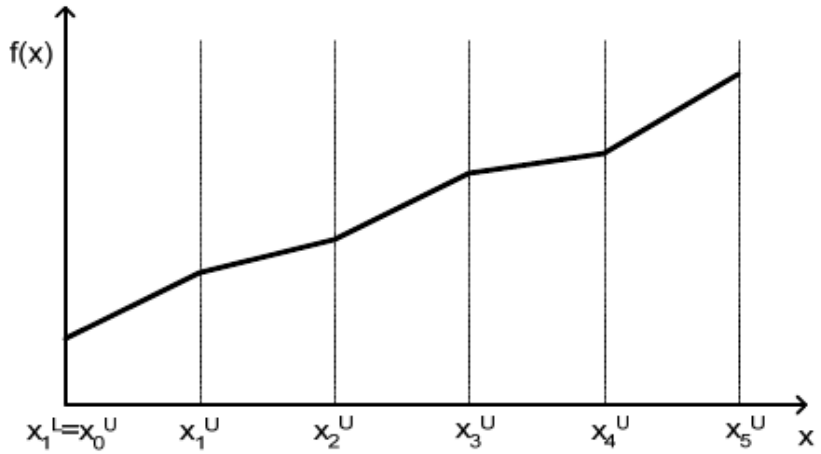
- Ce qui peut se comprendre avec la figure suivante :



Variables 0/1 : Généralisation

Généralisation aux fonctions linéaire mutuellement exclusives (Piecewise) :

$$
f(x) = \begin{array}{ll}
a_1 x \leq b_1 & si\ x_0^U \leq x < x_1^U \\
a_2 x \leq b_2 & si\ x_1^U \leq x < x_2^U \\
a_3 x \leq b_3 & si\ x_2^U \leq x < x_3^U \\
a_4 x \leq b_4 & si\ x_3^U \leq x < x_4^U \\
a_5 x \leq b_5 & si\ x_4^U \leq x < x_4^U
\end{array}
$$



Variables 0/1 : Généralisation (suite)

Big-M

On introduit un M assez large ainsi que y_j :

$$
\begin{array}{l}
f(x) = \\
\quad a_1 x \leq b_1 \quad si\ x_0^U \leq x < x_1^U \\
\quad a_2 x \leq b_2 \quad si\ x_1^U \leq x < x_2^U \\
\quad a_3 x \leq b_3 \quad si\ x_2^U \leq x < x_3^U \\
\quad a_4 x \leq b_4 \quad si\ x_3^U \leq x < x_4^U \\
\quad a_5 x \leq b_5 \quad si\ x_4^U \leq x < x_4^U
\end{array}
$$

$$
\begin{array}{l}
a_1 x \leq b_1 + M(1 - y_1) \\
a_2 x \leq b_2 + M(1 - y_2) \\
a_3 x \leq b_3 + M(1 - y_3) \\
a_4 x \leq b_4 + M(1 - y_4) \\
a_5 x \leq b_5 + M(1 - y_5) \\
\displaystyle\sum_{j=1}^{5} x_j^L y_j \leq x \leq \sum_{j=1}^{5} x_j^U y_j \\
\displaystyle\sum_{j=1}^{5} y_j = 1
\end{array}
$$

Application à un pb. de charge Fixe

Problème de charge fixe (les 2 méthodes) :

Le cout d'une activité est représenté par :

$$\begin{cases} k_j + c_j x_j & si\ 0 < x_j \leq x_j^U \\ 0 & si\ x_j = 0 \end{cases}$$

Le modèle de programmation disjonctive généralisé :

$$min \quad Z = g(x) + \sum_j cs_j$$

$$s.t. \qquad h(x) \geq 0 \qquad \leftarrow terme\ générique$$

$$\begin{pmatrix} Y_j \\ cs_j = k_j + c_j x_j \\ 0 < x_j \leq x_j^U \end{pmatrix} \quad \vee \quad \begin{pmatrix} \neg\, Y_j \\ cs_j = 0 \\ x_j = 0 \end{pmatrix}$$

$$x_j, cs_j \geq 0,\ Y_j \in \{True, false\}$$

Application à un pb. de charge Fixe (suite)

$$
\begin{aligned}
min \quad & Z = g(x) + \sum_j cs_j \\
s.t. \quad & h(x) \geq 0 \\
& \begin{pmatrix} Y_j \\ cs_j = k_j + c_j x_j \\ 0 < x_j \leq x_j^U \end{pmatrix} \quad \vee \quad \begin{pmatrix} \neg Y_j \\ cs_j = 0 \\ x_j = 0 \end{pmatrix} \\
& x_j, cs_j \geq 0, Y_j \in \{True, false\}
\end{aligned}
$$

L'**enveloppe convexe** et la formulation **big-M** de ce problème sera :

$$
\begin{aligned}
min \quad & Z = g(x) + \sum_j cs_j \\
s.t. \quad & \\
& h(x) \geq 0 \\
& cs_j = c_j x_j + k_j y_j \\
& x_j \leq x_j^U y_j \\
& x_j, cs_j \geq 0, y_j \in \{0,1\}
\end{aligned}
\qquad
\begin{aligned}
min \quad & Z = g(x) + \textstyle\sum_j cs_j \\
s.t. \quad & \\
& h(x) \geq 0 \\
& cs_j \geq c_j x_j + k_j - M(1 - y_j) \\
& x_j \leq x_j^U - M(1 - y_j) \\
& x_j, cs_j \geq 0, y_j \in \{0,1\}
\end{aligned}
$$

→ Pour le cas de Big-M, on remarque qu'on retranche $M(1 - y_j)$.
Or, pour $y_j = 1$, $1 - y_j = 0$ et la contrainte d'origine est respectée.

Variables 0/1 et K parmi N

K parmi N contraintes doivent être vérifiées :

$f_1(x_1, x_2, ..., x_n) \leq d_1$
$f_2(x_1, x_2, ..., x_n) \leq d_2$
...
$f_n(x_1, x_2, ..., x_n) \leq d_n$

K parmi N contraintes doivent être vérifiées.

➙ Voir le cas suivant.

Variables 0/1 et K parmi N (suite)

Fonctions avec N valeurs possibles :

Soit une fonction qui a des valeurs discrètes en partie droite :

$$f(x_1, x_2, ..., x_n) = d_1 \ ou \ d_2, \ ou \ ..., \ ou \ d_n$$

On introduit une variable binaire pour la partie droite :

$$f(x_1, x_2, ..., x_n) = \sum_{i=1}^{N} d_j \, y_i$$

$$\sum_{i=1}^{N} y_i = 1$$

$$y_i \in \{0, 1\} \quad i = 1, 2, ..., N$$

Généralisation de la disjonction

Plus généralement (Enveloppe Convexe : *convex hull*):

$$A^1 X \leq a^1 \vee ... \vee A^k X \leq a^k$$

Devient :

$$A^i X_i \leq a^i Y_i, i = 1, ..., k$$
$$X = X_1 + ... + X_k$$
$$Y_i \in \{0, 1\}$$

Exemple : Soit $X \in \{1..5\}, (3x \leq 7) \vee (2x \leq 5)$

```
| ?- X::0..5, (3*X #=< 7) #\/ (2*X #=<5), labeling(X).
X = 0 ?;
X = 1 ?;
X = 2 ?;
```

Version traduite :

```
| ?- X::0..5, 3*X1 #=< 7*Y, 2*X2 #=< 5*(1-Y), X #= X1+X2, Y::0..1, X1 #=< 5*Y, labeling(X).
X = 0       avec des valeurs pour X1 et X2 (peu important, c'est X qui compte !)
X = 1       idem
X = 2       idem
```

Généralisation de la disjonction (suite)

Quelques cas de figures où ces conversions (en BIP) peuvent être appliquées :

(i). Cas d'investissement avec :
$x_j = 1$ investir si l'alternative j
0 sinon
(ii). Sélection de sites :
$x_j = 1$ sélectionner le site x_j pour une activité
0 sinon
(iii). Conception, production et distribution :
$x_j = 1$ utiliser le centre de distribution i pour satisfaire le marché j
0 sinon
(iv). Expédition de marchandises
(v). Organisation d'activités liés
etc....

Représentation binaire des entiers

Soit la variable du type entier $0 \leq x < u$

➞ N est un entier telle que $2^N \leq u < 2^{N+1}$

Et la représentation binaire de x sera :

$$x = \sum_{i=0}^{N} 2^i y_i \qquad y_i \in \{0, 1\}$$

Exemple :

$$x \leq 5$$
$$2x_1 + 3x_2 \leq 30$$

Soit x_1 : $u = 5, \quad 2^2 < 5 < 2^3 \Rightarrow N = 2$

x_1 : $u = 10, \quad 2^3 < 10 < 2^4 \Rightarrow N = 3$ 10 car x_2 sera ≤ 10

Ce qui donne :

$$x_1 = y_0 + 2y_1 + 4y_2$$
$$x_2 = y_3 + 2y_4 + 4y_5 + 8y_6$$

Et on remplace les expressions dans les contraintes :

$$y_0 + 2y_1 + 4y_2 \leq 5$$
$$2(y_0 + 2y_1 + 4y_2) + 3(y_3 + 2y_4 + 4y_5 + 8y_6) \leq 30$$

Représentation binaire des entiers (suite)

• De même, le domaine discret d'une variable peut être représenté par des variable 0/1 :

Exemple : $X \in \{1, 5, 7, 9, 23\}$

$$X = 1y_1 + 5y_2 + 7y_3 + 9y_4 + 23y_5$$
$$y_1 + y_2 + y_3 + y_4 + y_5 = 1$$
$$y_i \in \{0, 1\}$$

Résolution et MIP

Soit les clauses (X_i : *bool*):

$X1 \vee X2 \vee \neg X3$
$\neg X1 \vee X2 \vee X3$

On obtient le résolvant : $\boldsymbol{X}\, 2 \vee \neg X3 \vee X4$

Les 2 clauses écrites en MIP :

$X1 + X2 + (1 - X3) \geq 1$
$(1 - X1) + X2 + X4$

On élimine $X1$ et on obtient $X2 + 1/2(1 - X3) + 1/2X4 \geq 1/2$
Cette expression "domine" $X2 + (1 - X3) + X4 \geq 1/2$
Ensuite, on peut arrondir le $1/2$ à droite
➙ car les variables sont booléennes et $\geq 1/2$ est forcement ≥ 1).

☞ Le résolvant correspond à un *Cutting Plan* du niveau 1 (de *Chvatal*).
➙ Les cutting plans sont présentés ici par l'élimination des variables.

Exemple 1 de modélisation

1- Choix discrets entre les variables continues.

	Temps de prod. par unité de Produit			Temps Machine dispo par semaine
	P1	P2	P3	
Plan 1	3	4	2	30
Plan 2	4	6	2	40
Profit unitaire	5	7	3	
Demande	7	5	9	

Etant donné cette table, proposer un modèle qui maximise le profit en respectant les 2 **contraintes supplémentaires** suivantes :

1. Parmi les produits possibles, il faut choisir **au plus deux** à fabriquer,
2. Seulement un des 2 plans doit être choisi pour fabriquer les produits choisis (à fabriquer).

Exemple 1 de modélisation (suite)

Une modélisation PPC :

	Temps de prod. par unité de Produit			Temps Machine dispo par semaine
	P1	P2	P3	
Plan 1	3	4	2	30
Plan 2	4	6	2	40
Profit unitaire	5	7	3	
Demande	7	5	9	

- Un booléen B_{ij} par case P_{ij} : $Produit_i \times Plan_j$
- Pour une ligne, $\sum_{1}^{3} B_{ij} \leq 2, \quad j = 1, 2$
- Seul un des plans doit être utilisé pour fabriquer ces produits :

$$\sum_{1}^{3} B_{i1} > 0 \Leftrightarrow \sum_{1}^{3} B_{i2} = 0 \quad \text{ET} \quad \sum_{1}^{3} B_{i2} > 0 \Leftrightarrow \sum_{1}^{3} B_{i1} = 0$$

- Si $B_{ij} = 1$ alors le produit P_{ij} est fabriqué : $(B_{ij} = 1) \Leftrightarrow (P_{ij} > 0) \quad i, j \in 1..3$
- Éventuellement (redondant) : la somme de chaque colonne < 2
- Les contraintes sur les quantités sont simplement installées.

☞ Avec PPC, l'installation des contraintes peut se faire par un prédicat.

Exemple 1 de modélisation (suite)

Modèle CLP (solution BProlog)

```
produits1(L_production, Total_Profits) :-
  L_production = [Nb_Prod1_Plan1, Nb_Prod2_Plan1,Nb_Prod3_Plan1,
                  Nb_Prod1_Plan2, Nb_Prod2_Plan2, Nb_Prod3_Plan2],

  [Bool_Prod1_Plan1,Bool_Prod2_Plan1,Bool_Prod3_Plan1,  % savoir quels produits fabriquer
    Bool_Prod1_Plan2,Bool_Prod2_Plan2,Bool_Prod3_Plan2] :: 0..1,

  % au plus  deux des 3 produits doivent être fabriqués. On désigne leur somme
  Somme_Bools_Produits_Plan1 #= Bool_Prod1_Plan1+Bool_Prod2_Plan1+Bool_Prod3_Plan1,
  Somme_Bools_Produits_Plan1 #=<2,
  Somme_Bools_Produits_Plan2#=Bool_Prod1_Plan2+Bool_Prod2_Plan2+Bool_Prod3_Plan2,
  Somme_Bools_Produits_Plan2#=<2,

  % Seul un des plans doit être urilisé pour faire ces produits
  (Somme_Bools_Produits_Plan1 #> 0 ) #<=> (Somme_Bools_Produits_Plan2 #= 0 ),
  (Somme_Bools_Produits_Plan2 #> 0 ) #<=> (Somme_Bools_Produits_Plan1 #= 0 ),

  % Important : liens entre les bools et les produits (sans quoi les mauvaises réponses)
  (Bool_Prod1_Plan1#=1)  #<=> (Nb_Prod1_Plan1 #> 0),
  (Bool_Prod2_Plan1#=1)  #<=> (Nb_Prod2_Plan1 #> 0),
  (Bool_Prod3_Plan1#=1)  #<=> (Nb_Prod3_Plan1 #> 0),
  (Bool_Prod1_Plan2#=1)  #<=> (Nb_Prod1_Plan2 #> 0),
  (Bool_Prod2_Plan2#=1)  #<=> (Nb_Prod2_Plan2 #> 0),
  (Bool_Prod3_Plan2#=1)  #<=> (Nb_Prod3_Plan2 #> 0),

  [Nb_Prod1_Plan1, Nb_Prod2_Plan1,Nb_Prod3_Plan1,
    Nb_Prod1_Plan2, Nb_Prod2_Plan2,Nb_Prod3_Plan2] :: 0..20,    % la max d'un produite est 20 (40/2)

  % Les temps sous contrainte hébdo
  Nb_Prod1_Plan1*3+Nb_Prod2_Plan1*4+Nb_Prod3_Plan1*2 #=<30,
  Nb_Prod1_Plan2*4+Nb_Prod2_Plan2*6+Nb_Prod3_Plan2*2 #=<40,
```

Exemple 1 de modélisation (suite)

```
  % Les demandes limitent les productions
  Nb_Prod1_Plan1+Nb_Prod1_Plan2 #=< 7,
  Nb_Prod2_Plan1+Nb_Prod2_Plan2 #=< 5,
  Nb_Prod3_Plan1+Nb_Prod3_Plan2 #=< 9,

  % Les profits
  Total_Profits #= (Nb_Prod1_Plan1+Nb_Prod1_Plan2)*5+(Nb_Prod2_Plan1+Nb_Prod2_Plan2)*7+
                                        (Nb_Prod3_Plan1+Nb_Prod3_Plan2)*3,
  maxof(labeling(L_production),Total_Profits).


Test :
    ? produits(L,S).
    L = [0,0,0,6,0,8]
    S = 54
```

Exemple 1 de modélisation (suite)

Modèle MIP (sans les 2 contraintes de la fin du sujet) :

- Variables de décision : x_i nbr de produit i fabriqué
- Objectif : $max\ z = 5x1 + 7x2 + 3x3$ (maximiser le profit).

Les contraintes :

- Temps disponible pour les plans :
  $3x1 + 4x2 + 2x3 \leq 30$
  $4x1 + 6x2 + 2x3 \leq 40$

- La demande :
  $x1 \leq 7$
  $x2 \leq 5$
  $x3 \leq 9$

- La non négativité : $x_j \geq 0$

Exemple 1 de modélisation (suite)

La prise en compte des 2 contraintes supplémentaires :

(1) Au plus deux des 3 produits d'un plan seront fabriqués.
On introduit une variable binaire par produit :
$y_j = 1 \quad si \ x_j > 0$ ce qui est exprimé par une implication dans PPC.
$y_j = 0 \quad si \ x_j = 0$ (ici, $x_j > 0 \Leftrightarrow y_j = 1$)

Pour cela, on ajoute les contraintes suivantes (les $x_j > 0$ n'apportent rien !):
$x_j \leq M.y_j$ où M est un nombre grand (voir ci-dessous)
➙ $y_j = 0$ $x_j \leq M.0 \rightarrow x_j \leq 0 \rightarrow x_j = 0$
➙ $y_j = 1$ $x_j \leq M.1 \rightarrow x_j \leq M$
Pour les y_j, on ajoute $\boxed{y_1 + y_2 + y_3 \leq 2}$

A propos de M :
En général, M est un grand nombre mais habituellement, il représente la borne supérieure du domaine de x_i (lorsqu'il est connu = la demande).
➙ Pour ce cas, cette borne est connue :
$\boxed{x_1 \leq 7y_1, \quad x_2 \leq 5y_2, \quad x_3 \leq 9y_3}$

Exemple 1 de modélisation (suite)

☞ Cette technique est inspirée d'enveloppe convexe.

➙ Pour ce cas de figure, elle est plus simple que celle venant de Big-M.

➙ Elle est en particulier employée lorsque l'énoncé même fait figurer un booléen (comme dans le cas $d = 1 \Leftrightarrow x > 0$)

• La technique Big-M concernant (uniquement) la 1ère contrainte donnerait :

```
.....
% Uniquement la  1ère contrainte traitée à la Big-M
X1  #=< 7+M1*(1-Y1),
X1 #> M1*(1-Y1), % cas Xi > 0.  % cette 2e C est redondante.
                 % De plus la technique utilisée ci-dessus est plus simple (à la "Enveloppe Convexe")
X1 #=0+M1*Y1,    % cas X_i = 0

% on répète la même chose pour les autres variables
X2  #=<  5+M2*(1-Y2), X2 #> M2*(1-Y2), X2 #=0+M2*Y2,

X3 #=<  9+M3*(1-Y3), X3#> M3*(1-Y3), X3 #=0+M3*Y3,

Y1 + Y2 +Y3 #=<  2,          % Ne pas écrire (1-Y1)+(1-Y2)+(1-Y3) #=< 2  !!!!
....
```

Exemple 1 de modélisation (suite)

(2) Utiliser seulement un des deux plans :

➙ On introduit la variable binaire y_4 :

$y_4 = 1$: $\quad 3x1 + 4x2 + 2x3 \leq 30$ $\qquad$ choix Plan 1

$y_4 = 0$: $\quad 4x1 + 6x2 + 2x3 \leq 40$ $\qquad$ choix Plan 2

Ce qui donne (méthode Big-M) :

$3x1 + 4x2 + 2x3 \leq 30 + M(1 - y_4)$

$4x1 + 6x2 + 2x3 \leq 40 + My_4$

Si $y_4 = 1 \rightarrow 3x1 + 4x2 + 2x3 \leq 30 + 0$ $\qquad$ Plan1 actif

$4x1 + 6x2 + 2x3 \leq 40 + M$ $\qquad$ la contrainte pour Plan 2 est **relaxée**

☞ A propos de y_j dans la contrainte (1) de la page précédente :

➙ A la place de $x_j \leq M.y_j$

On peut aussi bien utiliser $x_j \leq M.(1 - y_j)$

Exemple 1 de modélisation (suite)

Ce qui donne la version MIP :

```
produits_mip_BigM(L_production, Total_Profits) :-
  L_production=[X1,X2,X3],
  L_production :: 0..20,
  [Y1, Y2, Y3, Y4] :: 0..1,
  X1  #=< 7*Y1,
  X2  #=<  5*Y2,
  X3 #=<  9*Y3,
  Y1 + Y2 +Y3 #=<  2,
  X1#>= 0, X2#>= 0, X3 #>= 0,
  M=100,
  Total_Profits #= 5*X1 + 7*X2 + 3*X3,
  3*X1 + 4*X2 + 2*X3 #=< 30 + M*(1-Y4),
  4*X1 + 6*X2 + 2*X3  #=< 40 + M*Y4,
  maxof(labeling(L_production),Total_Profits).

Test :
| ?- produits_mip(L,S).
L = [6,0,8]
S = 54
```

Exemple 1 de modélisation (suite)

La version Enveloppe Convexe

Rappel des contraintes :

- Temps disponible pour les plans :
  $3x1 + 4x2 + 2x3 \leq 30$
  $4x1 + 6x2 + 2x3 \leq 40$

- La demande :
  $x1 \leq 7 \qquad x2 \leq 5 \qquad x3 \leq 9$

- La non négativité : $x_j \geq 0$

- On utilise 4 variables 0/1 : y_1, y_2, y_3 pour les produits dans un plan donné et y_4 pour le temps disponible :
  $3x_1^1 + 4x_2^1 + 2x_3^1 \leq 30y_4$,
  $4x_1^2 + 6x_2^2 + 2x_3^2 \leq 40(1 - y_4)$,
  $x_1 = x_1^1 + x_1^2, \qquad x2 = x_2^1 + x_2^2, \qquad x3 = x_3^1 + x_3^2,$
  $x1 \leq 7y_1, \qquad x_2 \leq 5y_2, \qquad x_3 \leq 9y_3,$ (voir les redondantes ci-dessous)
  $y_1 + y_2 + y_3 \leq 2,$
  $0 \leq x_i \leq 20, y_j \in \{0, 1\}$

- Les contraintes redondantes (couvertes par celles sur x_i ci-dessus):
  $x_1^1 \leq 7y_4, x_2^1 \leq 5y_4, x_3^1 \leq 9y_4,$
  $x_1^2 \leq 7(1 - y_4), x_2^2 \leq 5(1 - y_4), x_3^2 \leq 9(1 - y_4),$

Exemple 1 de modélisation (suite)

```
produits_mip_convexe(L_production, Total_Profits) :-
  L_production=[X1,X2,X3],
  L_production :: 0..20,
  [Y1, Y2, Y3, Y4] :: 0..1,
  X1  #=< 7*Y1,
  X2  #=<  5*Y2,
  X3 #=<  9*Y3,
  Y1 + Y2 +Y3 #=<  2,
  X1#>= 0, X2#>= 0, X3 #>= 0,

  Total_Profits #= 5*X1 + 7*X2 + 3*X3,

  3*X11 + 4*X21 + 2*X31 #=< 30*Y4,
  4*X12 + 6*X22 + 2*X32  #=< 40 * (1-Y4),
  X1 #= X11+X12,  X2 #= X21+X22,  X3 #= X31+X32,

  X11 #=< 7*Y4,
  X21  #=<  5*Y4,
  X31 #=<  9*Y4,

  X12 #=< 7*(1-Y4),
  X22  #=<  5*(1-Y4),
  X32 #=<  9*(1-Y4),
  Y4 :: 0..1,
  [X11,X21,X31,X12,X22,X32] :: 0..20,

  maxof(labeling([X1,X2,X3,Y4]),Total_Profits). % il faut un labeling de Y4 et Xi

Test :
?- produits_mip_bis(L_production, Total_Profits).
L_production = [6,0,8]
Total_Profits = 54
```

Exemple 2 : Cout / profit discrets

2 - Sélection de campagne de pub.

- 3 produits
- 5 spots télés disponibles au total
- Maximum 3 spots pour un produit
- Profits attendus (à maximiser)

➙ Il se peut que l'on décide de ne pas faire de pub pour un des produits.

➙ Ce produit là prendra la ligne "0 spots" de la matrice.

	Profits par Produit P_i		
Nb. Spots TV	P1	P2	P3
0	0	0	0
1	1	0	-1
2	2	2	2
3	3	3	4

Exemple 2 : Cout / profit discrets (suite)

Modélisation :

Variable de décision : $y_{jk} = 1$ Si le produit P_j a k spots
$y_{jk} = 0$ Sinon

Paramètres : c_{jk} = le profit généré par le produit P_j avec k spots

Fonction Objective :

$$\text{Maximiser } z = \sum_{j=1}^{3}\sum_{k=1}^{3} c_{jk}\, y_{jk}$$

s.t.

$\sum_{k=0}^{3} y_{jk} = 1$ pour $\forall j$ (un produit a un seul nbr de spot : somme de colonne par produit=1)

$\sum_{j=1}^{3}\sum_{k=1}^{3} y_{jk} = 5$ (il faut 5 "true" dans la zone non nulle de la matrice)

$y_{jk} \in \{0,1\}, j = 1,2,3, k = 0,1,2,3$

➙ Si une case jk de la matrice contient 1, le produit j aura k spots et le profit correspondant (c_{jk}) est comptabilisé.

Variables 0/1 et les bornes

- D'après la méthode de transformation d'une disjonction vue plus haut :

$$[d = 1, M \geq X > 0] \quad \vee \quad [d = 0, x = 0]$$

- On utilise $M = bsup(X)$, sinon on n"avance pas !
- On devrait introduire les bool y et $(1 - y)$ pour l'autre cas de $\vee$.
  ➙ Or, la variable d ici joue le même rôle.

☞ Il faut repérer la condition sur la ou les variables et les y sur les C.

- Donc, on a pour la branche gauche :
  $M \geq X > 0$ devient $y = d, M.y \geq X > 0.y$
- La branche de droite $d = 0, x = 0$ devient $y = d, x = 0.y$
- Résultat : $M.d \geq X > 0$
  $0 \geq X - Md$

☞ NB: dans la méthode enveloppe convexe, on ne multiplie pas les variables par les y, seulement les autres constantes.

Différence (X != Y)

- Différence : $A \neq B$ devient $(A > B) \vee (A <) B$

Méthode Big-M (seul Big-M est simple à implanter, voir plus loin) :

$$\begin{pmatrix} \neg Y \\ X1 < X2 \end{pmatrix} \quad \vee \quad \begin{pmatrix} Y \\ X1 > X2 \end{pmatrix}$$

Ce qui donne :

$$X1 < X2 + M(1 - y) \quad , \quad X1 > X2 + My$$

Test (CLP):

$Sup = 2, [X, Z] :: 1..Sup, X\# < Z + M * (1 - Y), X\# > Z + M * Y,$
$Y :: 0..1, labeling([X, Z, Y]).$

☞ la bsup (Sup) de X, Z n'est pas forcément M (ne marcherait pas ici).
➙ L'égalité de M et de U est utilisable dans le cas d'enveloppe convexe.

Différence (X != Y) (suite)

Tests exhaustifs (Big-M)

```
Inf=1, Sup=2, [X1,X2]::Inf..Sup, X1-X2#<M*(1-Y), X1-X2#>M*Y, Y::0..1, labeling([X1,X2,Y]).
Et TOUT EST OK :
Inf = 1
Sup = 2
X1 = 1
X2 = 2
M = _01530::[-268435455..-2]
Y = 1 ?;
Inf = 1
Sup = 2
X1 = 2
X2 = 1
M = _01530::[2..268435455]
Y = 0 ?;
```

➙ Idem pour différentes autres valeurs de Sup.

☞ Un moyen simple à utiliser pour vérifier les réponses :

➙ Poser les contraintes et *labeling* (de toutes vars), puis l'inverse de la contrainte à vérifier (plus simple que de vérifier toutes les réponses !):

```
Inf=1, Sup=100, [X1,X2]::Inf..Sup, X1-X2#<M*(1-Y), X1-X2#>M*Y, Y::0..1, labeling([X1,X2,Y]), X1 #= X2.
-> rép = NO.
```

☞ Il faut toujours énumérer Y.

Différence (X != Y) (suite)

Une solution **Big-M** exhaustive avec l'étude des bornes. :

☞ La solution Big-M reste plus intéressante.

$$X1 < X2 \quad \vee \quad X1 > X2$$
$$X1 - X2 < 0 \quad \vee \quad X1 - X2 > 0$$

A priori, pour $X \in [inf..sup]$:
$X1 - X2 :: [inf - Sup..Sup - inf]$

Mais, détallons les signes de $(X1 - X2 < 0)$ et $(X1 - X2 > 0)$ pour obtenir :

$X1 - X2 < 0$	$\vee$	$X1 - X2 > 0$
$X1 - X2 :: [Inf - Sup.. - 1]$	$\vee$	$X1 - X2 :: [1..Sup - Inf]$
$(car < 0)$		$(car > 0)$, qq soit Inf et Sup

On peut maintenant traduire en 2 colonnes (Big-M) avec $Y :: 0..1$:

$$X1 - X2 >= (Inf - Sup) + M(1 - Y), \qquad X1 - X2 >= 1 + M * Y,$$
$$X1 - X2 =< -1 + M * (1 - Y), \qquad X1 - X2 =< (Sup - Inf) + M * Y.$$

☞ On a besoin d'utiliser M (autre que Sup) dans cette formulation.

Différence (X != Y) (suite)

Test :

```
Inf=1, Sup=2, [X1,X2]::Inf..Sup,
X1-X2#>= (Inf-Sup)+M*(1-Y), X1-X2#=< -1+M*(1-Y),
X1-X2#>=1+M*Y, X1-X2 #=<(Sup-Inf)+M*Y, Y::0..1, labeling(Y).

Donne les deux solutions :
Inf = 1
Sup = 2
    X1 = 2
    X2 = 1
M = 2, Y = 0
--------------------
Inf = 1
Sup = 2
    X1 = 1
    X2 = 2
M = -2, Y = 1 ?;
```

Autres opérateurs

- Si on n'a pas l'égalité "="
  Alors $A = B$ devient $A \geq B \wedge A \leq B$
  ➟ Avec un ET entre les deux

- Si on n'a pas strictement $>$ (mais on a $\geq$) :
  ➟ $A > B$ *devient* $A \geq B - 1$
  ➟ $A < B$ *devient* $-A > -B$

  → en fait, c'est le coefficient de A qui reçoit le '-' (y compris le coefficient 1).

Exemple 1

Remarque : La modélisation par Big-M est bien plus simple.

Exemple : $S = a_1x_1 + a_2x_2 + a_3x_3 + a_4x_4, (D = 1 \Leftrightarrow S \leq B)$,

est convertie en (Big-M) :

$$S = a_1x_1 + a_2x_2 + a_3x_3 + a_4x_4, S \leq B + M(1 - D), S > B + MD, D \in \{0, 1\},$$

Vérification PPC et BIP :

```
Version PPC :

        A=[A1,A2,A3,A4], X=[X1,X2,X3,X4], A::1..4, X::1..3, D::0..1,Sum#=A1*X1+A2*X2+A3*X3+A4*X4,
        B#=4,((D#=1) #<=> Sum #=<B), labeling([A1,A2,A3,A4, X1,X2,X3,X4,D]).

--> Pour D=1, la seule réponse :
A = [1,1,1,1]
X = [1,1,1,1]
Sum = 4,        B = 4
----------------------------------------
Traduction (BIP)

        A=[A1,A2,A3,A4], X=[X1,X2,X3,X4], A::1..4, X::1..3, D::0..1,Sum#=A1*X1+A2*X2+A3*X3+A4*X4,
        B#=4, Sum #=< B+M*(1-D), Sum #> B+M*D,labeling([A1,A2,A3,A4, X1,X2,X3,X4]).

--> Pour D=1, la seule réponse :
A = [1,1,1,1]
X = [1,1,1,1]
Sum = 4,        B = 4
```

Exemple 2

- Énoncé :

$$f(x) = 10x \quad si\ 0 \leq x \leq 100$$
$$f(x) = 100 + 9x \quad si\ 100 < x \leq 300$$
$$f(x) = 1000 + 6x \quad si\ 300 < x \leq 500$$

☞ Bien faire attention aux sens des inégalités et l'expression de M_i, Yi ajoutées :

➙ Par ex, si on a $X = K$ et qu'on ajoute $M * (1 - Y)$,
on écrit $X \leq K + M(1 - Y)$
car on vient d'ajouter un terme et l'égalité change.

☞ Dans la pratique, les résultats avec $M_i * (1 - Y_i)$ sont préférées à $M_i * Y$ tout en imposant $\sum_j y_i = 1$ (et non $\sum_j (1 - y_i) = 1$).

- Vérification : ../..

Exemple 2 (suite)

- On avait :
  $f(x) = 10x \quad si\ 0 \leq x \leq 100$
  $f(x) = 100 + 9x \quad si\ 100 < x \leq 300$
  $f(x) = 1000 + 6x \quad si\ 300 < x \leq 500$

Conversion :

$Fx_1 + M1(1 - Y1) \geq 10X1, X1 \leq 100 + M1(1 - Y1), X1 \geq 0 + M1(1 - Y1),$
$Fx_2 + M2(1 - Y2) \geq 100 + 9X2, X2 \leq 300 + M2(1 - Y2), X2 > 100 - M2(1 - Y2),$
$Fx_3 + M3(1 - Y3) \geq 1000 + 6X3, X3 \leq 500 + M3(1 - Y3), X3 > 300 - M3(1 - Y3),$
$Fx_= Fx_1(1 - Y1) + Fx_2(1 - Y2) + Fx_3(1 - Y3),$
$Res = 10X1 + 100 + 9X2 + 1000 + 6X3,$
% Pour avoir le résultat car en l'absence des Mi, on ne l'a pas via Fx_i

```
p_bigM([Y1,Y2,Y3],[X1,X2,X3],[Z1,Z2,Z3,Z],Res) :-
  [Y1,Y2,Y3]::0..1, Y1+Y2+Y3#=1,
  [X1,X2,X3]::0..500,
  Z1+M1*(1-Y1)#>=10*X1, X1 #=< 100+M1*(1-Y1), X1 #>=0+M1*(1-Y1),
  Z2+M2*(1-Y2)#>=100+9*X2, X2 #=<300+M2*(1-Y2), X2 #>100-M2*(1-Y2), %  car > et
  Z3+M3*(1-Y3)#>=1000+6*X3, X3 #=<500+M3*(1-Y3), X3 #>300-M3*(1-Y3),
  Z #=Z1*(1-Y1)+Z2*(1-Y2)+Z3*(1-Y3),
  Res#=10*X1+100+9*X2+1000+6*X3, % Pour avoir le résultat car en l'absence des
  labeling([Y1,Y2,Y3,X1,X2,X3]).

Test :  [Y1,Y2,Y3] = [0,0,1]    [X1,X2,X3] = [0,0,301]  Res = 2906
                        ...
```

RO : un exemple BIP

☞ Exemple traité dans l'Addendum Cours-1 sur B&B.

• Soit un problème de type entrepôt (Modélisation BIP) :
Une entreprise souhaite construire une nouvelle usine à Lyon ou à Grenoble (ou les deux).

➙ AU plus un seul entrepôt mais là où on aura construit une usine.
➙ Le bénéfice et le cout de chaque configuration est donné dans la table ci-dessous.
➙ L'objectif est de maximiser les bénéfices.

Question oui/non	variable de décision	Bénéfices	couts
usine à Lyon	X_1	9 millions	6 millions
usine à Grenoble	X_2	5	3
entrepôt à Lyon	X_3	6	5
entrepôt à Grenoble	X_4	4	2
Capitaux disponibles max			10 millions.

RO : un exemple BIP (suite)

Modélisation :

- Variables de décision binaires : $\forall j = 1..4, x_j \ \textit{binaire}$
- Les contraintes de cout et des bénéfices donnent :
  $6x1 + 3x2 + 5x3 + 2x4 \leq 10$ et maximiser $9x1 + 5x2 + 6x3 + 4x4$
- Un seul entrepôt : $x3 + x4 \leq 1$
- Entrepôt si Usine : $x3 \leq x1$ et $x4 \leq x2$

RO : un exemple BIP (suite)

- On obtient le système BIP suivant :

$$
\begin{aligned}
maximize \quad & Z = 9x1 + 5x2 + 6x3 + 4x4 \\
s.t. \quad & 6x1 + 3x2 + 5x3 + 2x4 \leq 10 \\
& x3 + x4 \leq 1 \\
& -x1 + x3 \leq 0 \\
& -x2 + x4 \leq 0 \\
given \quad & x_j \ binaire \ \forall j = 1..4
\end{aligned}
$$

☞ Nous avons traduit "entrepôt si usine" par $X3 \implies X1$ convertie ensuite en $X3 \leq X1$. Que se passera-t-il si on impose plutôt "usine si entrepôt"?

➙ Réponse : on aura toujours une usine et un entrepôt, quelque soit le Capital initial (qui permettrait d'ouvrir 2 usines et un entrepôt).

Solution PPC : la logique des booléennes ($y_{1..4}$) est la même et on utilise 2 implications $y_3 \implies y_1, y_4 \implies y_2$ et $\neg(y_3 \wedge y_4)$ pour l'entrepôt.

RO : un exemple BIP (suite)

Solutions PPC et BIP (testé avec Capital=10 puis 15) :

```
entrepot_CPP(Capital, Ly, Benefs) :-
  Ly = [Y1,Y2,Y3,Y4], Ly :: 0..1,
  6*Y1+3*Y2+5*Y3+2*Y4 #=< Capital,
  #\(Y3 #/\ Y4),          % Au plus un (= la négation de AND)
  Y3 #=> Y1,  Y4 #=> Y2,
  Benefs #= 9*Y1+5*Y2+6*Y3+4*Y4,
  maxof(labeling(Ly),Benefs).

-------------------------------------------------------

entrepot_BIP(Capital, Lx, Benefs) :-
  Lx = [X1,X2,X3,X4], Lx :: 0..1,
  6*X1+3*X2+5*X3+2*X4 #=< Capital,
  X3+X4 #=<1,
  X3 #=< X1,  X4 #=< X2,
  Benefs #= 9*X1+5*X2+6*X3+4*X4,
  maxof(labeling(Lx),Benefs).

Tests :
| ?- entrepot_CPP(10,L,B).              L = [1,1,0,0], B = 14
| ?- entrepot_CPP(15,L,B).              L = [1,1,1,0], B = 20
| ?- entrepot_BIP(10,L,B).              L = [1,1,0,0], B = 14
| ?- entrepot_BIP(15,L,B).              L = [1,1,1,0], B = 20
```

Exemple Tournée

• Tournée : un exemple BIP avec modélisation OR/LP_Solve et PPC.

• Pour une prospection, un étudiant veut visiter les campus de trois universités en Rhône-Alpes pendant un voyage à partir de "Lyon" et retour.
Les trois Universités sont situées dans les villes "St-Étienne", "Valence" et "Grenoble" et l'étudiant veut visiter chaque ville universitaire une seule fois tout en faisant l'aller-retour le plus court possible.

• La table suivante donne les distances entre les villes :

Villes	Ville 1	Ville 2	Ville 3	Ville 4
	Lyon	St-Etienne	Valence	Grenoble
Lyon	0	26	34	78
St-Etienne	26	0	18	52
Valence	34	18	0	51
Grenoble	78	52	51	0

Exemple Tournée (suite)

Modélisation OR :

1. **Choix des variables** : une phase importante (clarté, simplicité, complexité).
   • Les étapes sont entre 2 villes.
   • On peut décider de représenter chaque étape par une variable $x_{dep,arr}$ dont la valeur finale reflète le fait de faire ou non une étape entre 2 villes.

2. **Les domaines** : booléen (0, 1).
   ➙ E.g. si $x_{1,2} = 1$ alors on fera l'étape entre ville1 et ville2; 0 sinon.

$$x_{i,j} \in \{0,1\} \quad \forall\ i,j = 1,2,3,4 \qquad \text{(C1)}$$

☞ Les domaines des variables peut être considéré comme une contrainte. Le choix des variables et leurs domaines est en rapport direct avec la complexité.

☞ Ne pas hésiter à changer de variable si l'on estime que le choix est pas bon !

Exemple Tournée (suite)

3. **Modélisation et contraintes :**
   1. Il n'y aura pas d'étape d'une ville à elle même, donc :

$$x_{i,i} = 0 \quad \forall\ i = 1, 2, 3, 4 \qquad \text{(C2)}$$

   2. Chaque ville est visitée une seule fois.
   Par exemple, ville-2 (St-Etienne) ne doit apparaître comme étape d'arrivée qu'une seule fois et donc $x_{i,2} \neq 0$ pour exactement un i.
   ➙ Ce qui donne, pour cette ville : $x_{1,2} + x_{2,2} + x_{3,2} + x_{4,2} = 1$.
   • Généralisation : toutes les variables sont binaires (et on a la contrainte C2 qui ne gène pas) :

$$\sum_{i=1}^{4} x_{i,j} = 1,\ \forall j\ \textit{fixé} \in 1..4 \qquad \text{(C3)}$$

   ➙ Ex : on fixe $j = 2$, on fait varier $i \in 1..4$ et on obtient (sachant que $x_{1,1} = 0$) $\quad x_{1,2} + x_{2,2} + x_{3,2} + x_{4,2} = 1.$
   ➙ Faire de même pour $j = 1, 3, 4$

Exemple Tournée (suite)

3. Les contraintes précédentes assurent que chaque ville est visitée une seule fois (comme destination) mais elles ne garantissent pas une solution correcte.
   ➞ Par exemple, la (mauvaise) solution $x_{1,2} = 1, x_{1,3} = 1, x_{1,4} = 1, x_{2,1} = 1$ et 0 pour toutes les autres variables satisfera les contraintes ci-dessus et pourtant, ce parcours-là est improbable !

   • Pour régler ce problème, il nous faut éviter qu'une ville soit plus d'une fois l'origine d'une étape et ajouter la contrainte :

$$\sum_{j=1}^{4} x_{i,j} = 1, \ \forall i \ \textit{fixé} \in 1..4 \qquad \text{(C4)}$$

   ➞ Par exemple, pour i fixé à 2, on impose que $x_{2,1} + x_{2,2} + x_{2,3} + x_{2,4} = 1$, c'est à dire : une seule de ces 4 variables vaudra 1.

Exemple Tournée (suite)

4. Les étapes obtenues ne doivent pas être déconnectées les unes des autres.
   ➙ Par exemple, une solution telle que $x_{1,2} = 1, x_{2,1} = 1, x_{3,4} = 1, x_{4,3} = 1$ (et toutes les autres=0) ne constitue pas un voyage aller-retour.
   ➙ Nous pouvons utiliser (appelée en PPC une étape *vérification de la solution*)

$$x_{i,j} + x_{j,i} \leq 1, \ \forall i = 1..4, j = 1..4 \qquad \text{(C5)}$$

   ☞ Pourquoi ne pas utiliser ici $x_{i,j} + x_{j,i} = 1, \ \forall ...$?
   ➥ Car par ex. pour $i = j = 2$, on aura $x_{2,2} + x_{2,2} = 0$.
   ➥ Aussi, imposer $x_{i,j} + x_{j,i} = 1$ (au lieu de ≤ 1) exigera qu'au moins l'une des deux variables ($x_{i,j}$ ou $x_{j,i}$) soit $= 1$. Or, il se peut que dans certaines combinaisons i, j, les deux variables soient nulles dans la solution finale.

• L'analyse OR ci-dessus est complète. Il suffira d'écrire les contraintes BIP (telles que indiquées ci-dessus pour obtenir une solution.
➙ Il y aura environ 30 expressions.

Exemple Tournée (suite)

- **Solution OR (lp_solve) :**

```
min : ;

/* Contrainte C1 */
X11=0; X22=0;X33=0;X44=0;

/* Contrainte C3 (départs) */
X12+X11+X13+X14=1;
X11+X13+X14+X12=1;
X13+X14+X12+X11=1;
X14+X12+X11+X13=1;
X22+X21+X23+X24=1;
X21+X23+X24+X22=1;
X23+X24+X22+X21=1;
X24+X22+X21+X23=1;
X31+X32+X33+X34=1;
X32+X33+X34+X31=1;
X33+X34+X31+X32=1;
X34+X31+X32+X33=1;
X41+X42+X43+X44=1;
X42+X43+X44+X41=1;
X43+X44+X41+X42=1;
X44+X41+X42+X43=1;


/* Contrainte C3 (arrivées) */
X22+X12+X32+X42=1;
X21+X11+X31+X41=1;
X13+X23+X33+X43=1;
X14+X24+X34+X44=1;
X12+X22+X32+X42=1;
X11+X21+X31+X41=1;
X23+X13+X33+X43=1;
```

Exemple Tournée (suite)

```
 X24+X14+X34+X44=1;
 X32+X22+X12+X42=1;
 X31+X21+X11+X41=1;
 X33+X13+X23+X43=1;
 X34+X14+X24+X44=1;
 X42+X22+X12+X32=1;
 X41+X21+X11+X31=1;
 X43+X13+X23+X33=1;
 X44+X14+X24+X34=1;

 /* faire une tournée  */
 X12+X21 <= 1;
 X13+X31 <= 1;
 X14+X41 <= 1;
 X23+X32 <= 1;
 X24+X42 <= 1;
 X34+X43 <= 1;

int X12,X13,X14,X21,X22,X23,X24,X31,X32,X33,X34,X41,X42,X43,X44;
```

- Solution lp_solve :

```
X12                             1
X24                             1
X31                             1
X43                             1

Toutes les autres variable = 0
```

Exemple Tournée (suite)

Solution PPC :

- En PPC, on peut coder cet exemple exactement comme avec OR.
  ➙ Les systèmes PPC peuvent utiliser un solveur Simplex à la demande.
- Une solution sans passer par un MIP.

➙ **Choix des variables :**

```
Liste_etapes=[X11,X12,X13,X14,X21,X22,X23,X24,X31,X32,X33,X34,X41,X42,X43,X44],
```

➙ **Domaine des variables :**

```
Liste_etapes   ::  0..1,
```

➙ **Contrainte C1 : on va pas d'une ville à elle même**

```
X11 = 0, X22 = 0, X33 = 0, X44 = 0,

Liste_Depart_ville_1=[X11,X12,X13,X14],        Liste_Depart_ville_2=[X21,X22,X23,X24],
Liste_Depart_ville_3=[X31,X32,X33,X34],        Liste_Depart_ville_4=[X41,X42,X43,X44],
```

➙ **Contrainte C3 : une ville est seulement une fois le départ d'une étape :**

```
only_one_true (Liste_Depart_ville_1),
only_one_true(Liste_Depart_ville_2),
only_one_true(Liste_Depart_ville_3),
only_one_true(Liste_Depart_ville_4),

Liste_Arrivee_ville__1=[X11,X21,X31,X41],        Liste_Arrivee_ville__2=[X12,X22,X32,X42],
Liste_Arrivee_ville__3=[X13,X23,X33,X43],        Liste_Arrivee_ville__4=[X14,X24,X34,X44],
```

➙ **Contrainte C2 : une ville est seulement une fois l'arrivée d'une étape :**

```
only_one_true(Liste_Arrivee_ville__1),
only_one_true(Liste_Arrivee_ville__2),
only_one_true(Liste_Arrivee_ville__3),
only_one_true(Liste_Arrivee_ville__4),
```

Exemple Tournée (suite)

➙ **Contrainte C4 : que_ce_soit_une_tourne**

```
X12   NAND  X21,  X13 NAND  X31,  X14 NAND  X41,
X23 NAND  X32,  X24 NAND  X42,
X34 NAND  X43.

enumerer (Liste_etapes).
```

- Solutions : notons qu'en PPC, on obtient toutes les solutions.

➙ X14=1, X23=1, X31=1, X42=1 (V1 → V4 → V2 → V3 → V1)

➙ X14=1, X21=1, X32=1, X43=1.

➙ X13=1, X24=1, X32=1, X41=1.

➙ X13=1 ,X21=1, X34=1, X42=1.

➙ X12=1, X24=1, X31=1, X43=1.

➙ X12=1, X23=1, X34=1, X41=1.

Tournoi

Un organisateur sportif prévoit un tournoi avec 8 équipes [Helmut Simonis-2009] :

- chaque équipe joue contre toutes les autres équipes une seule fois.
- le tournoi se joue sur 7 jours,
- chaque équipe joue tous les jours (des 7),
- les matchs sont prévus sur 7 sites, et
- chaque équipe doit jouer dans chaque site exactement une fois.

• Dans le cadre d'un accord avec la télévision, certains matchs sont pré-organisés.
➙ On peut soit fixer le match entre deux équipes particulières à une date déterminée et un site déterminé,
➙ Ou seulement décider par avance qu'une certaine équipe doit jouer un jour donné en un lieu donné.

• L'objectif est de déterminer le calendrier, de sorte que toutes les contraintes soient satisfaites.

Tournoi (suite)

Expressions des exigences (contraintes) :

➙ permettent d'envisager différentes contraintes / solutions :

- Il y a 8 équipes, 7 jours et 7 sites
- Chaque équipe joue chaque autre équipe une fois exactement
- Chaque équipe joue 7 matchs (redondant)
- Chaque équipe joue exactement une fois dans chaque site
- Chaque équipe joue chaque jour exactement une fois
- Un match se compose de 2 différentes équipes
- Il y a 4 matchs chaque jour (redondant)
- Il y a 4 matchs à chaque emplacement (redondants)
- Dans n'importe quel site, il y a au plus un match à la fois

☞ Selon les choix, on peut proposer différents idées de solution ../..

Tournoi (suite)

- **Idée 1 :** utiliser une matrice $Jour \times match$ (7×4)
  - ➙ Chaque cellule contient 2 variables (deux équipes)
  - ➙ Contrainte : toute équipe joue une seule fois chaque jour (*alldifferent*)
  - ➙ Les colonnes n'auront pas de signification
  - ➙ Les sites non représentés : comment faire ?
  - ➙ Comment dire : chaque équipe joue avec une autre une seule fois.

- **Idée 2 :** variables binaires pour exprimer : équipe i joue en site j le jour k.
  - ➙ Matrice à 3 dimensions
  - ➙ Chaque équipe joue une seule fois chaque jour
  - ➙ Chaque équipe joue une seule fois dans chaque site
  - ➙ Un match a 2 différentes équipes ? (variables auxiliaires nécessaires).
  - ➙ Chaque pair d'équipe se rencontrent une seule fois ? (vars auxiliaires).

- **Idée 3 :** variables booléennes pour exprimer : équipe i rencontre équipe j en site k le jour l.
  - ➙ $3136 = 8 * 8 * 7 * 7$ variables
  - ➙ Toutes les contraintes sont linéaires

Tournoi (suite)

- Idée 4 : chaque équipe joue contre une autre exactement une fois :
  - ➙ $7 * 4$ matchs à organiser (28 variables)
  - ➙ Toutes les variables différentes (pas de match le même jour, le même lieu)
  - ➙ Par construction, chaque équipe jouera 7 matchs
  - ➙ Comment dire : chaque équipe jouera une fois par jour ?
  - ➙ Comment dire : chaque équipe jouera une fois par site ?

- Cette idée donnera :

	City 1	City 2	City 3	City 4	City 5	City 6	City 7
Day 1	1	2	3	4	5	6	7
Day 2	8	9	10	11	12	13	14
Day 3	15	16	17	18	19	20	21
Day 4	22	23	24	25	26	27	28
Day 5	29	30	31	32	33	34	35
Day 6	36	37	38	39	40	41	42
Day 7	43	44	45	46	47	48	49

Tournoi (suite)

- Jour 1 : valeurs 1..7
- 4 variables prendront une des ces valeurs (1..7)
- Jour 2 : 8..15, &c.
- Une contrainte par jour

➞ Exactement 4 variables prendront leur valeurs dans la ligne correspondantes
➞ 7 de ces contraintes (car 7 lignes).

- Le site 1 correspond aux valeurs 1,8,15, 22, ...
- 4 variables prendront une des ces valeurs
- Site 2 correspond à 2, 9,16, ...
- Une contrainte par site

➞ Exactement 4 variables prendront leur valeurs dans l'ensemble correspondant
➞ 7 de ces contraintes sur chacune des 28 variables.

- Choisir les variables qui correspondent à l'équipe i

  ➞ Exactement une de ces variables prendront un evaleur dans 1..7
- De même pour les autres jours
- De même pour les autres matchs

  ➞ 56 contraintes sur chacune des7 variables
- De même pour les équipes et sites :

  ➞ 56 autres contraintes

	City 1	City 2	City 3	City 4	City 5	City 6	City 7
Day 1	1	2	3	4	5	6	7
Day 2	8	9	10	11	12	13	14
Day 3	15	16	17	18	19	20	21
Day 4	22	23	24	25	26	27	28
Day 5	29	30	31	32	33	34	35
Day 6	36	37	38	39	40	41	42
Day 7	43	44	45	46	47	48	49

Bilan de l'idée 4 :

- 28 variables,
- Un *alldifferent*
- 7 contraintes sur toutes les variables (pour les jours)
- 7 contraintes sur toutes les variables (pour les sites)
- 56 contraintes sur 7 variables (pour les jours)
- 56 contraintes sur 7 variables (pour les sites)

- . . . et on n'a pas encore fini !

Tournoi (suite)

- Idée 5 (reprise en PPC) :
  une matrice carré de $Jour \times Site = 49$ de matchs (couples d'équipes [A,B])
  - ➙ Chaque cellule contient un match (couples d'équipes)
  - ➙ Comment éviter les symétries ([A,B] vs [B,A])
  - ➙ Utiliser [0,0] pour l'absence de match
  - ➙ Une cellule contient $[0, 0]$ ou $[A, B], A \neq B \neq 0$.
  - ➙ Plus facile : chaque ligne / colonne contient un match une seule fois.
  - ➙ Attention à *alldifferent* (pour les zéros)
  - ➙ Comment exprimer : chaque paire ordonnée apparaît une seule fois ?
  - ➙ Comment exprimer : chaque équipe joue une fois par jour / site ?
- Et du coté des contraintes non-domaine-fini ?

☞ Mettre mon ex de choix SD n-reines / complexité

Une solution PPC

- Le tableau initial (avec les matchs pré organisés), puis une solution :

	City 1	City 2	City 3	City 4	City 5	City 6	City 7
Day 1		8			7, 5		
Day 2	2	1, 5					
Day 3	7		8				
Day 4					2	5	1
Day 5	8					1	
Day 6				5, 4			
Day 7	4				1, 3		

	City 1	City 2	City 3	City 4	City 5	City 6	City 7
Day 1		6, 8		1, 2	5, 7		3, 4
Day 2	2, 3	1, 5			4, 8	6, 7	
Day 3	1, 7	2, 4	3, 8				5, 6
Day 4			4, 7		2, 6	3, 5	1, 8
Day 5	5, 8			3, 6		1, 4	2, 7
Day 6		3, 7	1, 6	4, 5		2, 8	
Day 7	4, 6		2, 5	7, 8	1, 3		

Une solution PPC (suite)

Une modélisation (applicable à cette classe de problèmes) :

Placez numéros 1 à 8 dans les cellules de la matrice de sorte que :

- dans chaque rangée et dans chaque colonne figure chaque numéro d'équipe ($\in 1..8$) exactement une fois, et
- chaque cellule contient soit pas de chiffres, soit un couple de chiffres ($\in 1..8$, = un match) différents, et
- chaque couple de chiffres (un match) apparaît dans seulement une cellule.

- On peut envisager la structure de données suivante :
  Pour 8 équipes ($E1..E8$) :
  ➙ M : une matrice 7 x 7
  ➙ Lignes : les jours ($Ji : i = 1..7$)
  ➙ Colonnes : les Sites ($Si, i = 1..7$)
  ➙ Une case : vide ou $[Eu, Ev]$: les deux équipes qui jouent

Une solution PPC (suite)

1. **Choix des variables :**
   Une matrice carré **M** de $7 \times 7 = 49$ de couples d'équipes ($Jours \times Site$)
   ➙ Chaque couple dans une cellule $Mij, i, j = 1..7$
2. **Domaines :**
   49 cellules contenant chacune un couple $[Eu, Ev], u, v \in 0..8, u \neq v$,
   ☞ Pour une cellule vide (pas de match $jour \times site$), on utilisera le couple $[0, 0]$.
3. **Contraintes :**
   Pour simplifier et éviter les solutions symétriques, on décide que dans un couple non vide (donc : $\neq [0, 0]$) $[Eu, Ev] : u < v$.
   - Cel_i est autorisée à contenir soit $[0, 0]$ soit $[Eu, Ev], Eu \neq 0, Ev \neq 0$
   - Déf : $[Eu, Ev] \neq [Eu', Ev']$ si $u \neq u'$ ou $v \neq v'$
   - Pour chaque ligne de la matrice M contenant 7 cellules $Cel_{i=1..7}$ (représentant tous les matchs d'un jour) :
     ➙ Si $Cel_i = [Eu, Ev] \neq [0, 0]$ alors $Cel_i \neq Cel_{k=1..7, i \neq k}$
     ➙ Que toutes les équipes soient impliquées dans $Cel_{i=1..7}$

Une solution PPC (suite)

- Pour chaque colonne de la matrice M contenant 7 cellules $Cel_{j=1..7}$ (représentant tous les matchs d'un site) :
  - ➙ Si $Cel_j = [Eu, Ev] \neq [0,0]$ alors $Cel_j \neq Cel_{k=1..7, k\neq j}$
  - ➙ Que toutes les équipes soient impliquées dans $Cel_{j=1..7}$

- Il y a beaucoup de solutions.
- Une solution (autre) :

```
[[0,0],[0,0],[0,0],[1,2],[3,4],[5,6],[7,8]]
[[7,8],[3,4],[5,6],[0,0],[0,0],[0,0],[1,2]]
[[5,6],[0,0],[0,0],[3,4],[8,7],[1,2],[0,0]]
[[0,0],[7,2],[1,8],[0,0],[0,0],[3,4],[5,6]]
[[3,4],[6,8],[0,0],[5,7],[1,2],[0,0],[0,0]]
[[0,0],[1,5],[2,7],[6,8],[0,0],[0,0],[3,4]]
[[1,2],[0,0],[3,4],[0,0],[5,6],[7,8],[0,0]]
```

- Aspects déclaratifs de la PPC.

Une solution PPC (suite)

```
Tournoi(M) :
    Créer_matrice(M[7,7]) contenant 49 couples [Ei,Ej]
    Fixer_Domaines : pour chaque [Ei,Ej] = 0..8, 0..8    ☞ 0 pour [0,0]
    Contraindre_lignes(M) : chaque case d'une ligne contient un couple [Ei,Ej] ou [0,0]
                ☞       [Ei, Ej], Ei/=Ej est unique mais pas les [0,0]
    Contraindre_colonnes(M) : chaque case d'une colonne contient un couple [Ei,Ej] ou [0,0]
                ☞       [Ei, Ej], Ei/=Ej est unique mais pas les [0,0]
    enumerer(M).

Contraindre_lignes(M) :
    Pour toutes les lignes L de M
        Toute_equipe_0_ou_1_fois_dans_chaque_ligne(L)

Contraindre_colonnes(M) :
    Transposer(M,M')        % pour faire de même que pour les lignes
    Pour toutes les lignes L de M'
        Toute_equipe_0_ou_1_fois_dans_chaque_ligne(L)

Toute_equipe_0_ou_1_fois_dans_une_ligne(L) :
          ☞  L = [E1,E2]_1, ... , [E1,E2]_7
          ☞  imposer à chaque [E1,E2]_i d'être soit =[0,0], soit unique dans L

    Tout_couple_contient_2_equipes_ou_0_0(L),
    Couples_tous_differents_sauf_pour_0_0(L),
    Toute_equipe_est_placee(L).      ☞ Ici, on est déconnecté des colonnes et on veut
```

Une solution PPC (suite)

```
                                                    que toutes les équipes soient présentes
Couples_tous_differents_sauf_pour_0_0(L : [E1,E2]_1, ... , [E1,E2]_7) :
    Aplatir L ➙  L'= [E11,E21, E12,E22, E13,E23, ... E77] : 14 variables
    Pour tout X dans L'
        X_egal_0_ou_pas_dans(X, reste_de(L')),
    Couples_tous_differents_sauf_pour_0_0(reste_de(L')). ☞ à ne  pas oublier


X_egal_0_ou_pas_dans(X, L) :     % L est une liste plate
    Pour tout élément Y de L     % Si L=[], ne rien faire
        (X = 0 <==> true)
         OR
        (X /=0 <==> X /= Y).

Tout_couple_contient_2_equipes_ou_0_0(L : L = [X,Y]_1, ... , [X,Y]_7) :
    Pour tout couple <X,Y> dans L :
       (X  = 0  <==> Y = 0)
       OR
       (X /= 0  <==> Y /= 0).      % pas besoin de dire X /=Y (on pourrait) car :
    ☞  De plus, la somme = 36 impose que tous les chiffres (équipes) soient présents.

Toute_equipe_est_placee(L) :    %  est appelé "vérification d'une solution en PPC"
    % par exemple :
    Somme_tous_chiffres(L,36)  %(car 1+2+...+8=36)
```

Une solution PPC (suite)

Remarques :

• La contrainte exactement_un existe pour les variables en domaine fini
  ➙ Mais on peut créer la notre (par exemple pour un match).
  ➙ cf. *alldifferent* sur les couples d'équipes (utilisé ci-dessus)

• Dans certains environnements, il est possible de transformer un prédicat en contrainte (cf. Eclipse).
  ➙ Expliquer la différence.
  ➙ Eclipse propose également *gcc : global cardinality constraint*
  ➙ *gcc(low, high, value)* : *value* $\in$ *low..high*

Exemple Bin Packing

- Un ensemble de n objets de hauteur h_i
- A ranger dans des boîtes de hauteur H
- Minimiser le nombre de boîtes utilisées

Formulation P en PLNE

- $x_{ij} \equiv 1$ si i est rangé dans la boîte j
- $y_j \equiv 1$ si la boîte j est utilisée

$$\min \sum_{j \in N} y_j$$

$$\begin{cases} \sum_j x_{ij} = 1 & \forall i \in N \\ \sum_i h_i x_{ij} \leq H y_j & \forall j \in N \\ y_j, x_{ij} \in \{0,1\} & \forall i,j \in N \end{cases}$$

Exemple Bin Packing (suite)

Suite de cet exemple :

➙ beaucoup de symétries :

Si l'optimum utilise 3 boites, autant prendre les 3 premières !

➙ Quelle contrainte ajouter ?

- Le premier PLNE est une formulation du BinPacking
- Ajouter les contraintes de symétries, n'est-ce pas redondant ?

Essayons de résoudre l'instance

- 15 objets à ranger dans des boîtes de hauteur $H = 20$
- hauteurs

6	7	8	9	10

en trois exemplaires chacun

- (très) petit exemple
- Quelle est la solution optimale ?

Exemple Bin Packing (suite)

Modélisation :

• On utilise une idée d'affectation (une matrice de booléens)

	H_1	..	H_j	..	
h_1					
...					
h_i			1		$\leftarrow \sum_j x_{ij} = 1$ (1 seule fois)
h_i			1		
...					
h_n	0	0	1	0	
			$\uparrow \sum$ des h_i dans $H_j \leq hauteur(H_j)$: $\sum x_{ij}.h_j \leq H_j.y_j$ $(si\ h_i et\ H_j = 1)$		

• On se donne une instance simple (petites boites : [6,7,8,9,10]) et volume grande boite = 10 pour les tests.

Puis, dans la version Big-M, on peut se permettre de passer à un nombre plus grand de petites boites et un volume de 20 pour la grande boite.

Exemple Bin Packing (suite)

- Solution avec une matrice de booléens (PPC)
- Cette solution PPC (utilisant l'implication) fonctionne bien pour la petite instance du problème mais pour desq données plus grandes, le temps de réponse est trop longue.

```
binpack_array_bool:-
    L_petites_boites=[10,9,8,7,6],
    Vol_big_Box=20,

    % on va résoudre le pb. On reviens ici pour les affichags.
    binpack_array_bool(L_petites_boites,Nb_Max_bigBox, Vol_big_Box, Matrice_bool),

        % Comptage du nombre de grande boites effectivement utilisées
    foreach(J in 1..N, [ac(Nb_Big_Bte_used,0)], [Col,S],
      (Col @= [Matrice_bool[I,J] : I in 1..N],
       (S is sum(Col), (((S > 0) -> Nb_Big_Bte_used^1 is Nb_Big_Bte_used^0+1
                                                    ; Nb_Big_Bte_used^1 is Nb_Big_Bte_used^0+0)))
        )),
        format("Nb␣big␣boites␣used␣=␣~2d␣\n", [Nb_Big_Bte_used]).

%-------------------------------------------------------------
binpack_array_bool(L_petites_boites, Nb_Max_bigBox, Vol_big_Box, Matrice_bool):-
    N=Nb_Max_bigBox,
    new_array(Vect_petites_btes, [N]), % Contient les petits volumes
    foreach(I in 1..N, [Ele],(nth(I,L_petites_boites,Ele), Vect_petites_btes[I]#=Ele)), % Son init

    new_array(Matrice_bool,[N,N]),      % La matrice
    Vars @= [Matrice_bool[I,J] : I in 1..N, J in 1..N],
    Vars :: 0..1,       % Matrice de bool
    foreach(I in 1..N, sum([Matrice_bool[I,J] : J in 1..N]) #= 1), % un h_i dans chaque ligne
```

Exemple Bin Packing (suite)

```
% Somme des colonnes
foreach(J in 1..N,  sum([Vect_petites_btes[I]* Matrice_bool[I,J] : I in 1..N]) #=<Vol_big_Box),

  Nb_Big_Bte_used :: 0..Nb_Max_bigBox,


foreach(J in 1..N,  [ac(Nb_Big_Bte_used,0)],[SS],
  (SS #=sum([Matrice_bool[I,J] : I in 1..N]), % Il faut passer par SS
                  (SS #> 0) #=> (Nb_Big_Bte_used^1#=Nb_Big_Bte_used^0+1), % Il faut bien les deux
                  (SS #= 0) #=> (Nb_Big_Bte_used^1#=Nb_Big_Bte_used^0+0)  % (cas =0 et cas > 0)
    )),

    minof(labeling([down,ff],[Y1,Y2|Vars]), Nb_Big_Bte_used).
```

- Solution : 3 grandes boites nécessaires.
- Comme indiqué ci-dessus, cette solution PPC devient lente dès que la combinaisons dès que la combinaison de petites boites devient grande.
- Cette lenteur vient des implications (disjonctions) dans le code

Exemple Bin Packing (suite)

Version Big-M (grandement accélérée) :

- Dans cette solution, la partie (de la solution PPC):

$$\forall i : 1..n,$$
$$(\sum_{j=1}^{n} M[i,j] > 0) \implies Nb\_Big\_Bte\_used++ \quad \wedge$$
$$(\sum_{j=1}^{n} M[i,j] = 0) \implies Nb\_Big\_Bte\_used \quad inchange$$

Devient (conjonction entre toutes) :

$$\forall i : 1..n,$$
$$\sum_{j=1}^{n} M[i,j] \leq BigM_1 * (1 - y_1), Nb\_Big\_Bte\_used \geq Nb\_Big\_Bte\_used + 1 + BigM_1 * y_1,$$
$$\sum_{j=1}^{n} M[i,j] \geq 0 + BigM_2 * (1 - y_2), Nb\_Big\_Bte\_used \leq Nb\_Big\_Bte\_used + 0 + BigM_2 * y_2,$$
$$y_k \in 0..1$$

- Ce qui donne la solution (on remarque le nombre élevé des petites boites);
  ➙ Vue la rapidité de la méthode, on peut se le permettre !

Exemple Bin Packing (suite)

../..

```
binpack_array_bool:-
    L_petites_boites=[6,7,8,9,10, 2,4,5, 6,7,8,9,10,6,7,8,9,10,1,2,2,1,3,2,1,3,6,4,2,1],
    Vol_big_Box=20,

    % on va résoudre le pb. On reviens ici pour les affichags.
    binpack_array_bool(L_petites_boites,Nb_Max_bigBox, Vol_big_Box, Matrice_bool),

        % Comptage du nombre de grande boites effectivement utilisées
    foreach(J in 1..N, [ac(Nb_Big_Bte_used,0)], [Col,S],
      (Col @= [Matrice_bool[I,J] : I in 1..N],
       (S is sum(Col), (((S > 0) -> Nb_Big_Bte_used^1 is Nb_Big_Bte_used^0+1
                                                    ; Nb_Big_Bte_used^1 is Nb_Big_Bte_used^0+0)))
        )),
        format("Nb␣big␣boites␣used␣=␣~2d␣\n", [Nb_Big_Bte_used]).

%---------------------------------------------------------------
binpack_array_bool(L_petites_boites, Nb_Max_bigBox, Vol_big_Box, Matrice_bool):-
    N=Nb_Max_bigBox,
    new_array(Vect_petites_btes, [N]), % Contient les petits volumes
    foreach(I in 1..N, [Ele],(nth(I,L_petites_boites,Ele), Vect_petites_btes[I]#=Ele)), % Son init

    new_array(Matrice_bool,[N,N]),      % La matrice
    Vars @= [Matrice_bool[I,J] : I in 1..N, J in 1..N],
    Vars :: 0..1,       % Matrice de bool
    foreach(I in 1..N, sum([Matrice_bool[I,J] : J in 1..N]) #= 1), % un h_i dans chaque ligne

    % Somme des colonnes
    foreach(J in 1..N,  sum([Vect_petites_btes[I]* Matrice_bool[I,J] : I in 1..N]) #=<Vol_big_Box),

    Nb_Big_Bte_used :: 0..Nb_Max_bigBox,

    [Y1,Y2] :: 0..1,
```

Exemple Bin Packing (suite)

```
    foreach(J in 1..N,
      [ac(Nb_Big_Bte_used,0)],  % les accus
      [S, BigM1],
      (S #=sum([Matrice_bool[I,J] : I in 1..N]),
                        S #=<BigM1*(1-Y1),Nb_Big_Bte_used^1#>=Nb_Big_Bte_used^0+1+BigM1*Y1,
                        S #>= 0+BigM2*(1-Y2), Nb_Big_Bte_used^1#=<Nb_Big_Bte_used^0+0+BigM2*Y2
        )),
        )),

        minof(labeling([down,ff],[Y1,Y2|Vars]), Nb_Big_Bte_used).
```

Test :

petites	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
6	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Exemple Bin Packing (suite)

```
   2   | 0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
   1   | 0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
   3   | 0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
   2   | 0  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
   1   | 0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
   3   | 0  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
   6   | 0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
   4   | 0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
   2   | 0  0  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
   1   | 0  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
Pour un nbr 30 et un totale de 159 petites boites, Le Nb grde Boites Used = 8
(valeur Nominale 159/ 20 = 7.95)
```

Exemple Giapetto

• L'atelier de *Giapetto* fabrique deux types de jouets en bois : soldats et trains.
• Un soldat se vend pour 27 unités et emploie pour 10 unités des matières premières.
• Chaque soldat coute par ailleurs 14 unités de couts divers (amortissement, salaire, etc.).
• Un train se vend pour 21 et utilise la valeur 9 unités en matières premières.
• Chaque train coute en frais généraux 10 unités.
• La fabrication des soldats et des trains en bois exige deux types de machines pour la menuiserie et la finition.
• Un soldat a besoin de 2 heures de finition et de 1 heure de menuiserie ou. Un train a besoin de 1 heure de finition et 1 heure de menuiserie.
• L'entreprise dispose de toute la matière première. Par contre, elle ne dispose que de 100 heures de finition et 80 heures de menuiserie.
• La demande des trains est illimitée, mais au plus 40 soldats sont achetés chaque semaine.
• *Giapetto* veut maximiser ses bénéfices d'hebdomadaire.

Exemple Giapetto (suite)

- Soit x_1 : nbr de soldats en bois produits par semaine et
  x_2 : nbr de trains produits par semaine.
- Les bénéfices seront alors
$$Z = (27 - 10 - 14)x_1 + (21 - 9 - 10)x_2 = 3x_1 + 2x_2$$
- Par ailleurs, les contraintes de production sont :
  $\infty \geq x_1 \geq 0$ et $40 \geq x_2 \geq 0$ (nbr de produits fabriques)
  $x_1 + x_2 \leq 80$ (menuiseries) et $2x_1 + x_2 \leq 100$ (finition)

```
Z#=3*X1+2*X2 ,
X1::0..40,        X2 #>= 0,
X1+X2 #=< 80, 2*X1+X2 #=< 100,
maxof(labeling([X1,X2]),Z).

Test :
        Z = 180
        X1 = 20
        X2 = 60
```

Exemple Giapetto (suite)

Une solution GLPK (*giapetto.mod*)

```
# problème giapetto
/* Decision variables */
var x1 >=0; /* soldier */
var x2 >=0; /* train */
/* Objective function */
maximize z: 3*x1 + 2*x2;
/* Constraints */
s.t. Finishing : 2*x1 + x2 <= 100;
s.t. Carpentry : x1 + x2 <= 80;
s.t. Demand : x1 <= 40;
end;

Test :
glpsol -m giapetto.mod -o giapetto.soldat

Et la solution comme ci-dessus (Z=180)
```

Tabmat