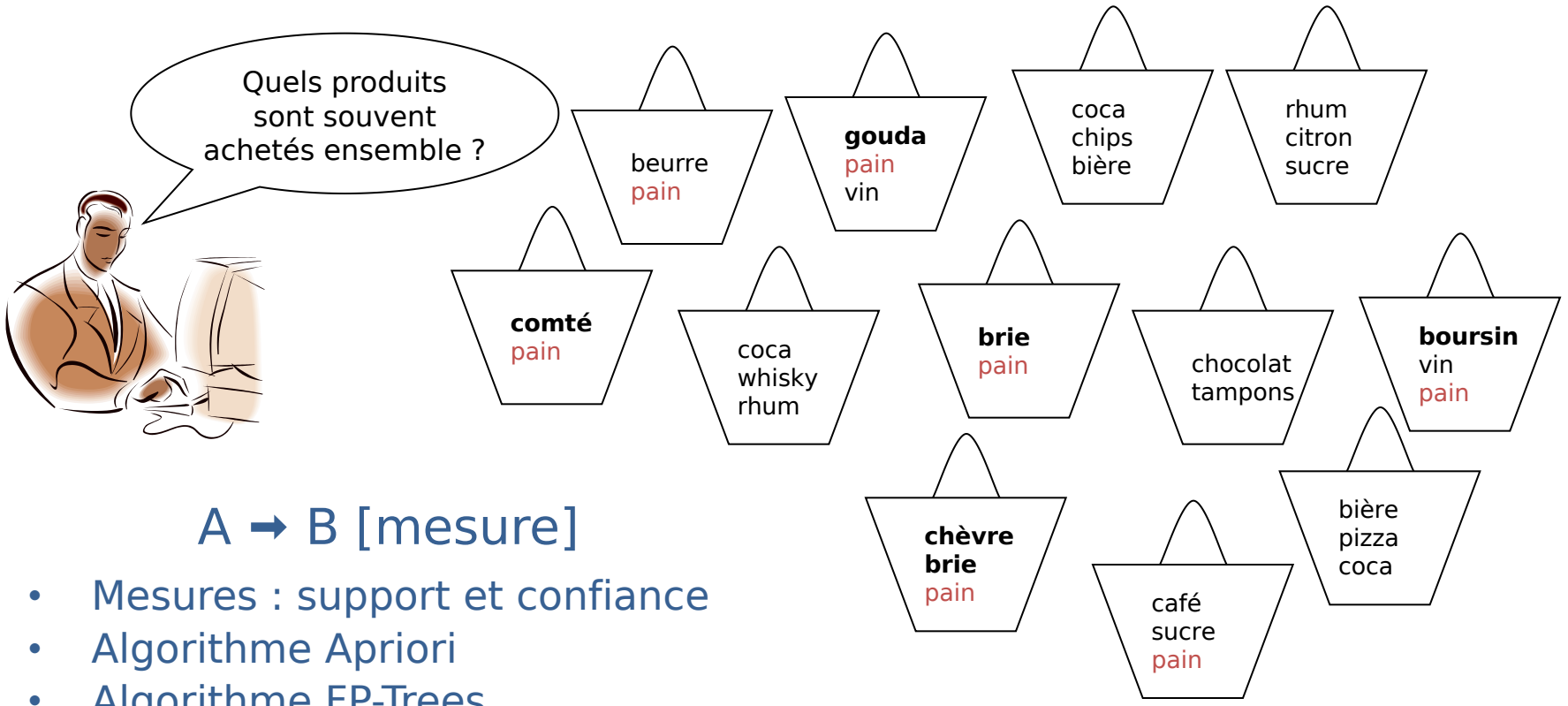
Règles d'associations



A ➡ B [mesure]

- Mesures : support et confiance
- Algorithme Apriori
- Algorithme FP-Trees
- Règles booléennes (présence/absence) / quantitatives (30<âge<40)
- Règles multi-niveaux (brie < fromage < produits laitiers)
- Associations multidimensionnelles (âge, revenus, achats)
- Autres mesures
- Exploration avec contraintes

Recherche des associations

- Règles d'association :
  - motifs de la forme : Corps ➡ Tête
  - Exemple : achète(x, "cacahuètes") ➡ achète(x, "bière")

- Étant donnés :
  - une base de transactions D, I = {i1, i2,..., in}
  - chaque transaction est décrite par un identifiant TID et une liste d'items TTID = {i1, i2,..., im} ⊆ I

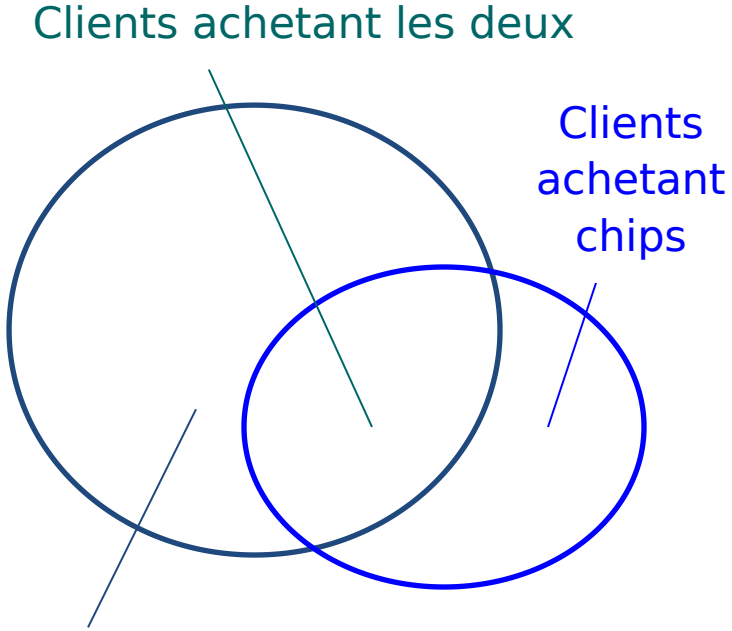
une transaction contient A ⇔ A ⊆ T

règle A ➡ B / A ⊂ I, et B ⊂ I et A ∩ B = ∅

Trouver: toutes les règles qui expriment une corrélation entre la présence d'un item avec la présence d'un ensemble d'items

Ex : 98% des personnes qui achètent des chips achètent de la bière

Mesures : support et confiance



Trouver les règles *X & Y ➡ Z*
avec un support > s et une confiance >c

support, *s*, probabilité qu'une transaction contienne {X, Y, Z}

confiance, *c*, probabilité conditionnelle qu'une transaction qui contient {X, Y} contienne aussi *Z*

Confiance=support(X,Y,Z)/support(X,Y)

ID Transaction	Items
2000	A,B,C
1000	A,C
4000	A,D
5000	B,E,F

Soit support minimum 50%, et confiance minimum 50%,

A ➡ *C* (50%, 66.6%)

C ➡ *A* (50%, 100%)

Pour $A \rightarrow C$:

support = support($\{A, C\}$) = 50%

confiance = support($\{A, C\}$)/support($\{A\}$) = 66.6%

Transaction ID	Items
2000	A,B,C
1000	A,C
4000	A,D
5000	B,E,F

Min. support 50%
Min. confiance 50%

Itemsets fréquents	Support
{A}	75%
{B}	50%
{C}	50%
{A,C}	50%

Extraction des associations : approche naïve

- D: une base de transactions
- I: ensemble de tous les items avec |I|=n

pour chaque ensemble d'items possible
  calculer son support
  si support min_support
    pour chaque règle r : $A_1,\ldots,A_{m-1} \rightarrow A_m$ tq $J=\{A_1,\ldots,A_m\}$
      si confiance(r) min_confiance
        afficher r

Bien sûr, en pratique, il y a trop de sous-ensembles de I possibles : 2^n

Illustration : nombre d'occurences de chaque paire possible

- On suppose :
  - 10^5 items (100K)
  - 10^7 transactions (10M)
  - En moyenne chaque transaction concerne 10 items.

- Pour trouver les paires d'items fréquents (2-itemsets) :

  - Pour chaque transaction, il y a C_{10}^2 paires à considérer : $45 \times 10^7 = 450$ millions

$$C_n^p = \binom{n}{p} = \frac{n!}{p!(n-p)!}$$

Extraction des assiociations : algorithme Apriori

- Principe :

  Si un ensemble est n'est pas fréquent,

  alors tous ses sur-ensembles ne sont pas fréquents

  - Si {A} n’est pas fréquent alors {A,B} ne peut pas l’être
  - si {A,B} est fréquent alors {A} et{B} le sont

- Itérativement, trouver les itemsets fréquents dont la cardinalité varie de 1 à k (k-itemset)
- Utiliser les itemsets fréquents pour générer les règles d’association

Algorithme Apriori

- Étape de jointure: C_k est généré en joignant L_{k-1}avec lui même
- Étape d'élimination: Chaque (k-1)-itemset qui n'est pas fréquent ne peut être un sous ensemble d'un k-itemset fréquent

C_k: Itemset candidat de taille k,

L_k : itemset fréquent de taille k

L_1 = {items fréquents}

pour (k = 1; L_k != ; k++)

C_{k+1} = candidats générés à partir de L_k *% jointure*

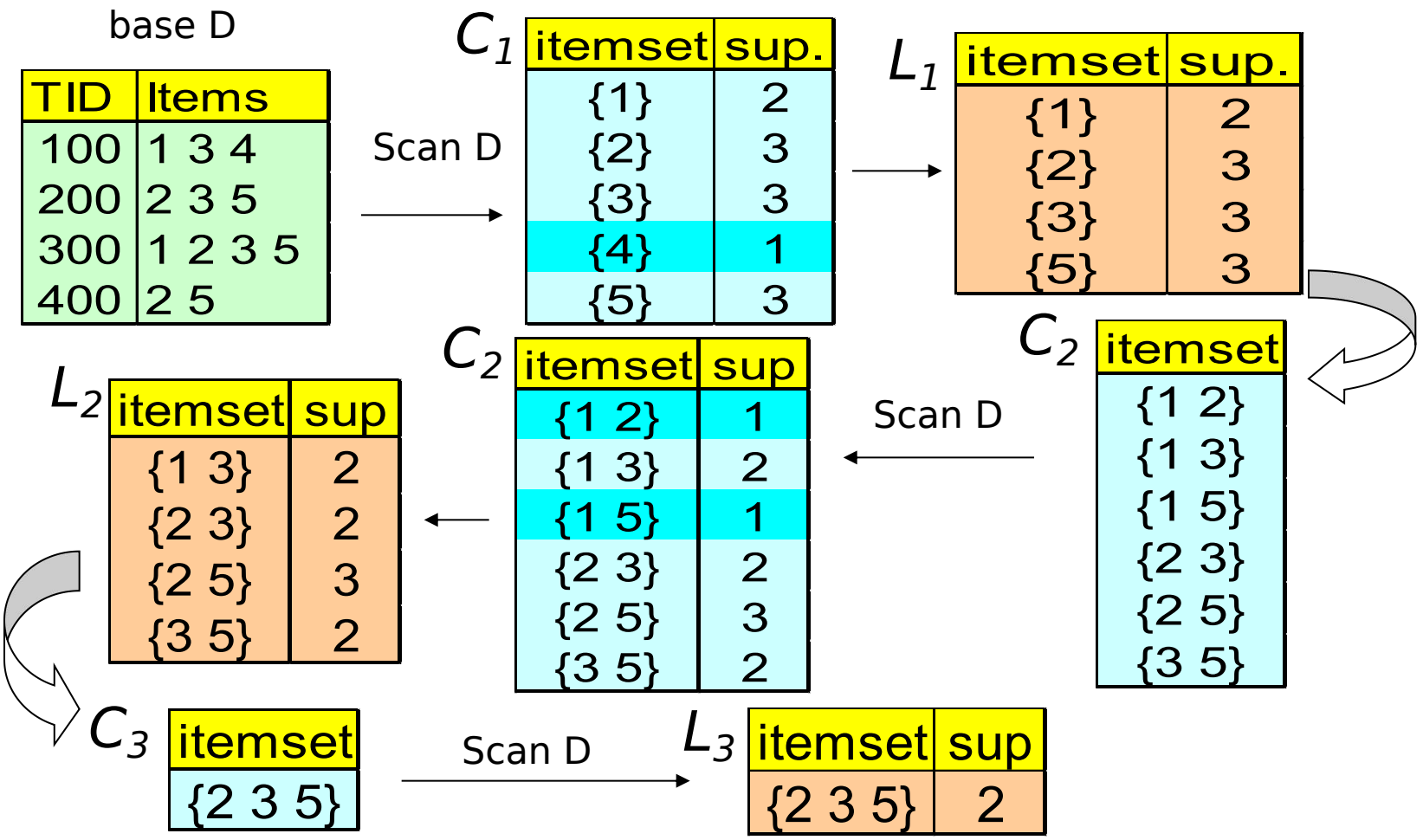
pour chaque transaction t dans la base

incrémenter le COUNT ***des candidats de C_{k+1} qui sont dans t***

L_{k+1} = candidats dans C_{k+1} dont COUNT > support_min

renvoyer $_k L_k$

avec min_support=2



Exemple de génération de candidats

- L_3={*abc, abd, acd, ace, bcd*}
- jointure : $C_4 = L_3 * L_3$
  - *abcd* à partir *de abc* et *abd*
  - *acde* à partir *acd* et *ace*
  - C_4 = {*abcd, acde*}
- Elagage :
  - *acde* est supprimé car *ade* n'est pas dans L_3
- C_4={*abcd*}

Extraction des règles

Confiance (A ➡ B)=support(AB)/support(A)

Algorithme :

pour chaque itemset fréquent f
  pour chaque sous ensemble s ⊂ f, avec s≠∅
    si confiance(s f \ s) > min_conf
    alors afficher(s f \ s)
  fin pour
fin pour

min_sup=50%
min_conf=75%

Items	count
1	2
2	3
3	3
5	3

Items	count
1, 3	2
2, 3	2
2, 5	3
3, 5	2
2, 3, 5	2

Règles	Conf.
1➡3	100%
3➡1	66%
2➡3	66%
3➡2	66%
2➡5	100%
5➡2	100%
3➡5	66%
5➡3	66%
2,3➡5	100%
2,5➡3	66%
3,5➡2	100%
2➡3,5	66%
3➡2,5	100%
5➡2,3	66%

Défauts d'Apriori

- Le principe de l'algorithme:
  - Utiliser les (k – 1)-itemsets fréquents pour générer les k-itemsets candidats
  - Scanner la base pour tester le support des candidats

- Point faible : génération des candidats
  - Beaucoup :
    - 10K 1-itemsets fréquents générant ~50M paires d'items candidates
    - Pour trouver les 100-itemsets, on doit générer 2^{100} 10^{30} candidats.
  - Plusieurs scans de la base:
    - On doit faire (n +1) scans, pour trouver les n-itemsets fréquents

Exploration sans génération de candidats

- Compresser la base, Frequent-Pattern tree (FP-tree)
  - Une représentation condensée
  - Évite les scans coûteux de la base

- Développer une méthode efficace pour l'exploration basée sur une approche
  - diviser-pour-régner: décompose le problèmes en sous-problèmes
  - Pas de génération de candidats :

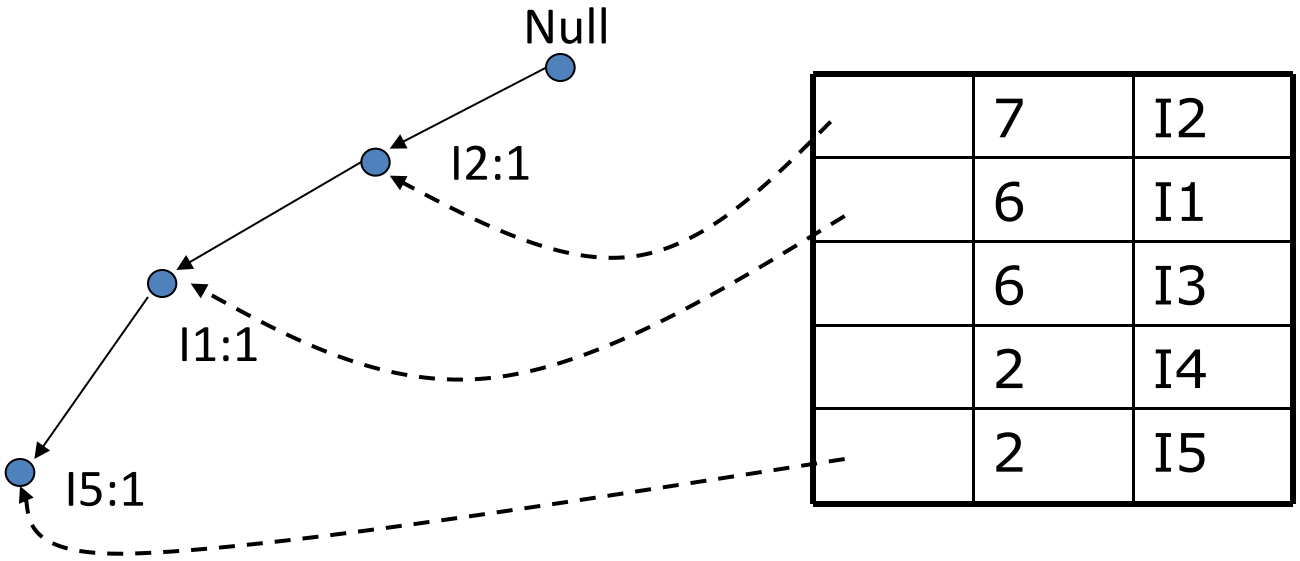
    test de la "sous-base" seulement !

FP-Trees : exemple

TID	T100	T200	T300	T400	T500	T600	T700	T800	T900
Liste items	I1, I2, I5	I2, I4, I6	I1, I3	I1, I2, I4	I2, I3, I8	I2, I3	I1, I3, I7	I1,I2,I3,I5	I1, I2, I3

Supposons que min-support=2. On construit la liste « triée »:
L = [I2:7, I1:6, I3:6, I4:2, I5:2]

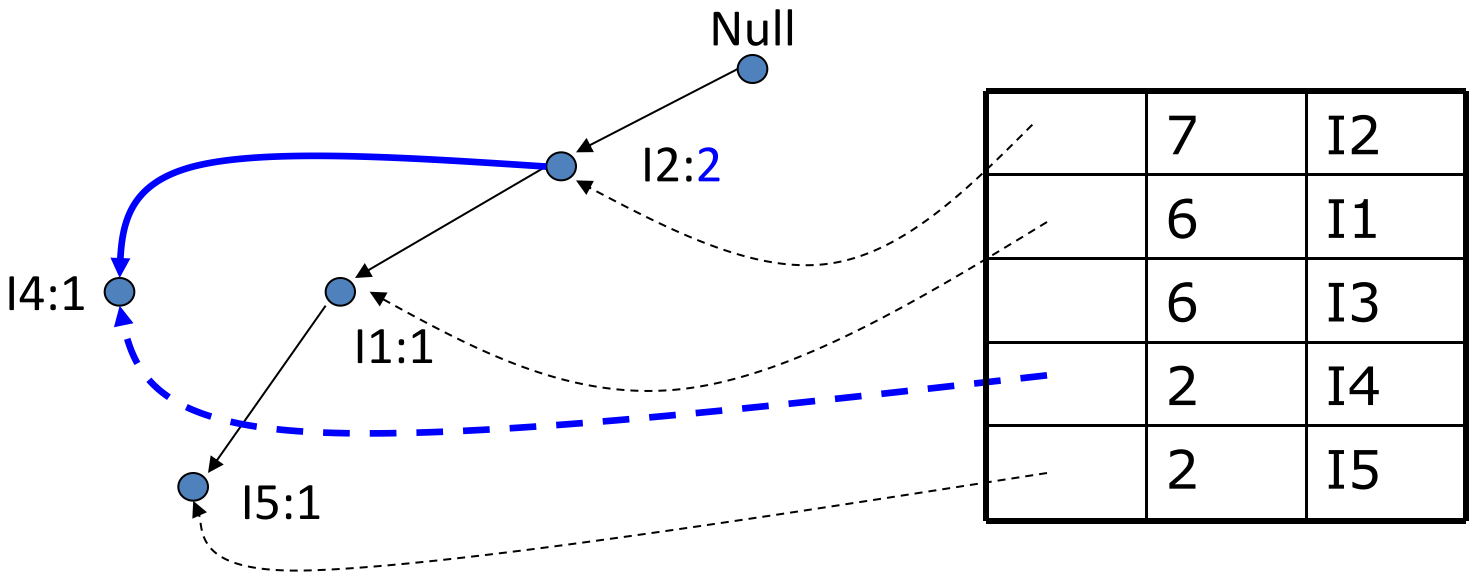
On parcourt une 2ème fois la base. On lit les transactions selon l'ordre des items dans L : pour T100 on a I2,I1,I5. La lecture de T100 donne



FP-Trees : exemple

TID	T100	T200	T300	T400	T500	T600	T700	T800	T900
Liste items	I1, I2, I5	I2, I4, I6	I1, I3	I1, I2, I4	I2, I3, I8	I2, I3	I1, I3, I7	I1,I2,I3,I5	I1, I2, I3

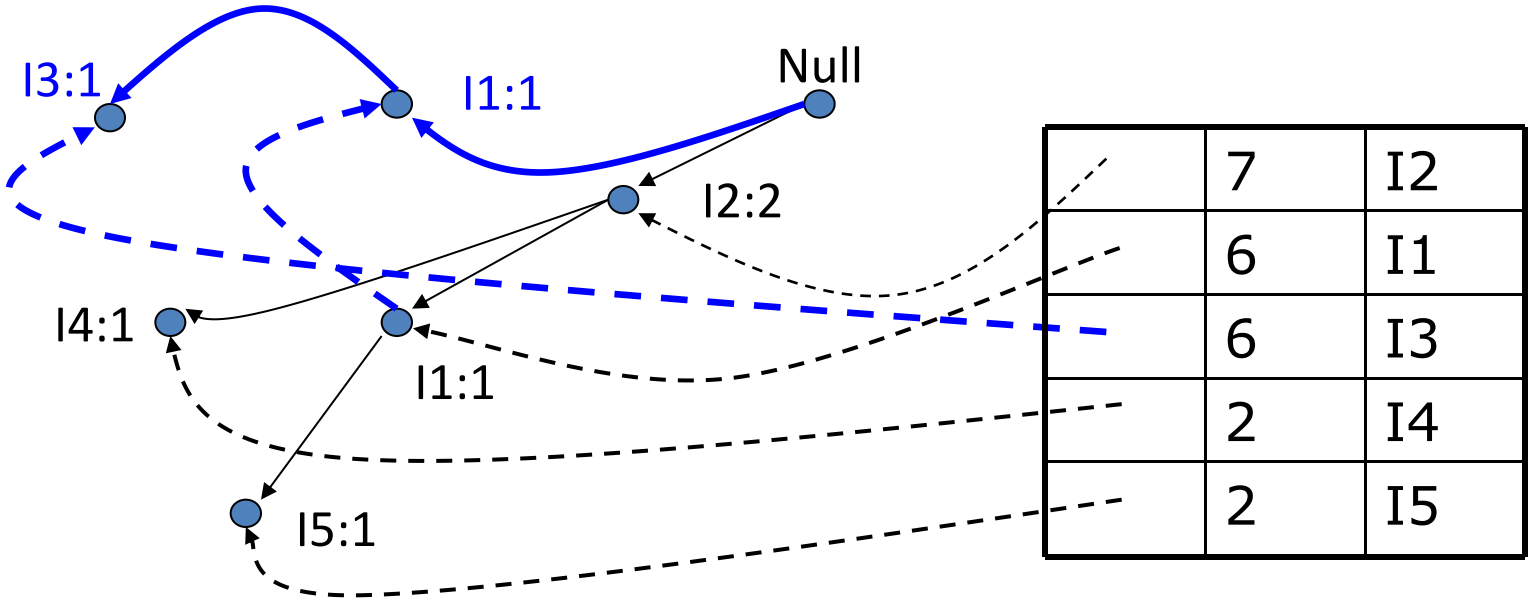
La lecture de T200 va *a priori* générer une branche qui relie la racine à I2 et I2 à I4. Or cette branche partage un préfixe (i.e I2) avec une branche qui existe déjà. L'arbre obtenu après lecture de T200 sera



FP-Trees : exemple

TID	T100	T200	T300	T400	T500	T600	T700	T800	T900
Liste items	I1, I2, I5	I2, I4, I6	I1, I3	I1, I2, I4	I2, I3, I8	I2, I3	I1, I3, I7	I1,I2,I3,I5	I1, I2, I3

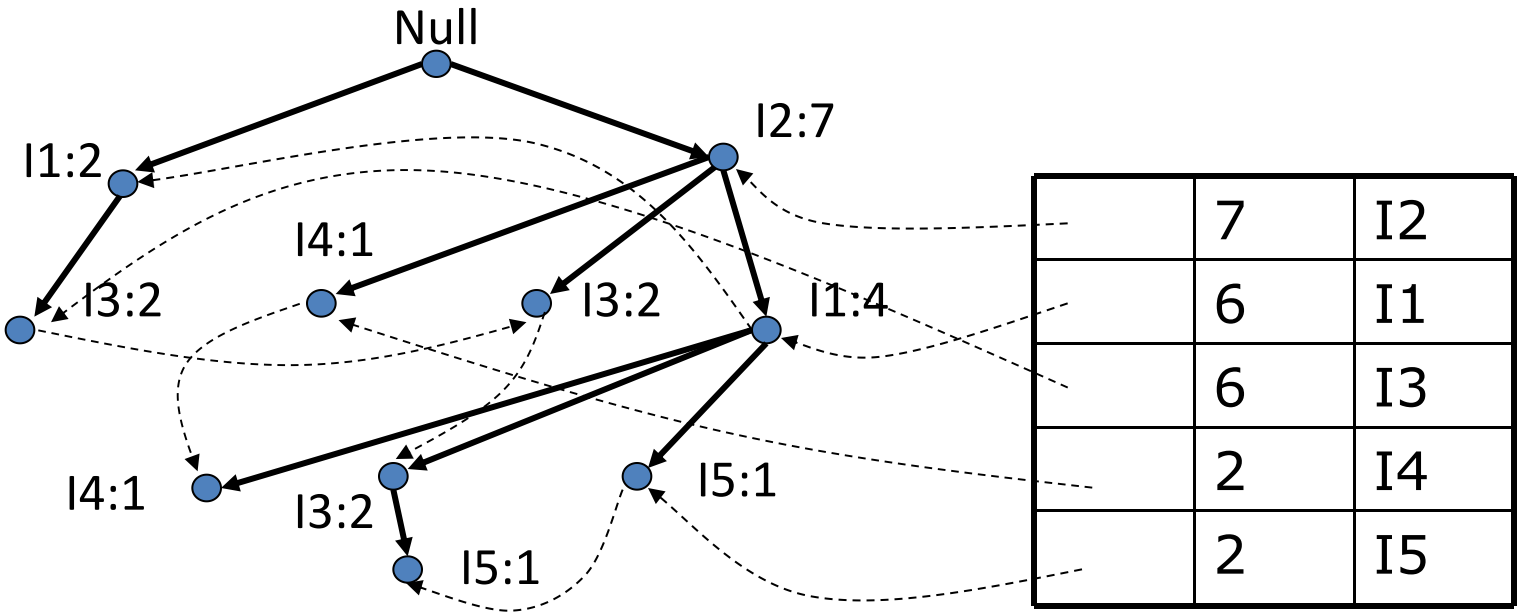
En lisant T300, l'ordre selon L est I1,I3. Ceci nous amène à ajouter une branche Null ➡ I1 ➡ I3. Noter qu'elle n'a pas de préfixe commun avec ce qui existe déjà. On obtient



FP-Trees : exemple

TID	T100	T200	T300	T400	T500	T600	T700	T800	T900
Liste items	I1, I2, I5	I2, I4, I6	I1, I3	I1, I2, I4	I2, I3, I8	I2, I3	I1, I3, I7	I1,I2,I3,I5	I1, I2, I3

Finalement, le FP-tree obtenu est



- Considérons I5. Il apparaît dans 2 branches

  I2 → I1 → I5:1 et I2 → I1 → I3 → I5:1

- Ainsi, pour le suffixe I5, on a 2 chemins préfixes: <I2,I1:1> et <I2,I1,I3:1>. Ils forment sa «table conditionnelle»

TiD	Itemset
1	I2, I1
2	I2, I3, I1

- Le «FP-tree conditionnel» de I5 contient

  une seule branche I2 → I1.

  I3 n'en fait pas partie car son support est 1 qui est <2

  (Rappel: min_support=2)

- Ce chemin unique va générer toutes les combinaisons de I5

  avec I1 et I2, i.e {I1,I5}:2, {I2,I5}:2, {I1,I2,I5}:2

Phase de l'exploration (suite)

- Considérons I4. Sa table conditionnelle est formée de <I2,I1:1> et <I2:1>

- Le FP-Tree conditionnel ne contient donc qu'un seul nœud I2

- Nous obtenons donc un itemset fréquent qui est {I2,I4}:2

Phase de l'exploration

Item	Base conditionnelle	FP-tree conditionnel	Itemsets générés
I5	<I2,I1>:1,<I2,I1,I3>:1	I2:2→I1:2	{I2, I5}:2 {I1, I5}:2 {I2,I1,I5}:2
I4	<I2,I1>:1,<I2>:1	I2:2	{I2, I4}:2
I3	<I2,I1>:2 ,<I2>:2, <I1>:2	I2:4→I1:2 I1:2	{I2, I3}:4 {I1,I3}:4 {I2, I1, I3}:2
I1	<I2>:4	I2:4	{I2,I1}:4

Ce n’est pas la peine de regarder I2 car ça va donner les combinaisons avec les autres items qui ont déjà été considérés

Application

Genome Biology 2007, 8:R3 (doi:10.1186/gb-2007-8-1-r3)

GENECODIS: a web-based tool for finding significant concurrent annotations in gene lists

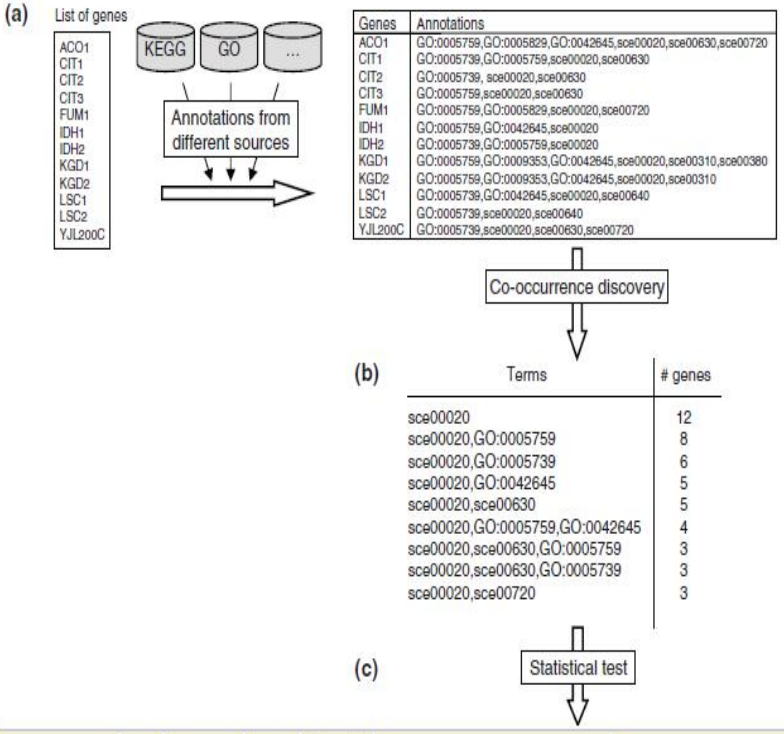
Pedro Carmona-Saez*, Monica Chagoyen*†, Francisco Tirado†, Jose M Carazo* and Alberto Pascual-Montano†

Abstract

We present GENECODIS, a web-based tool that integrates different sources of information to search for annotations that frequently co-occur in a set of genes and rank them by statistical significance. The analysis of concurrent annotations provides significant information for the biologic interpretation of high-throughput experiments and may outperform the results of standard methods for the functional analysis of gene lists. GENECODIS is publicly available at http://genecodis.dacya.ucm.es/.

Finding sets of terms that frequently appear together in a list of genes

To extract combinations of gene annotations, GENECODIS uses a modification to the methodology reported by Carmona-Saez and coworkers [6], which implements the *apriori* algorithm to extract associations among gene annotations and expression patterns.



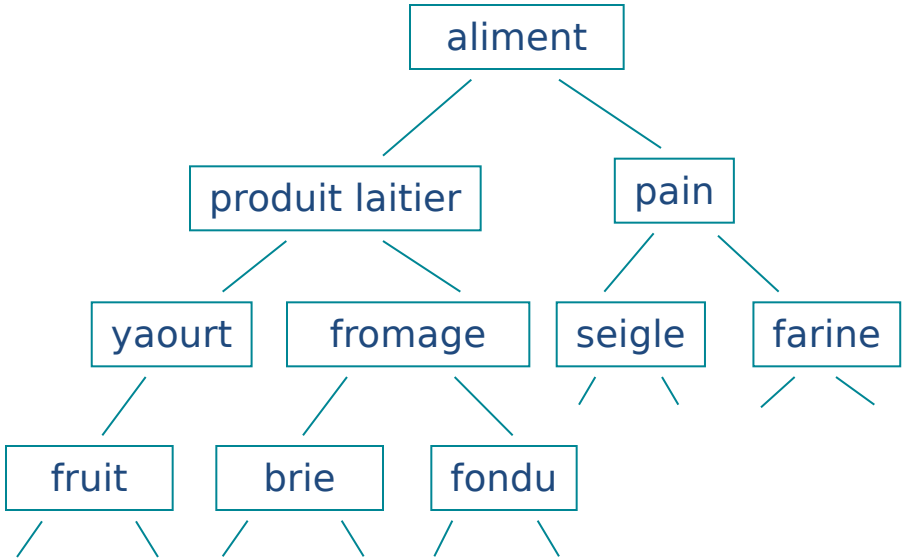
Annotation/s	# List	# Reference	*p*-value	Corrected *p*-value	Genes	Description/s
00020	12(12)	30(6194)	1.90e-28	1.90e-27	CIT2, ACO1, KGD2, LSC2, YJL200C, IDH2, LSC1, KGD1, IDH1, CIT1, FUM1, CIT3	(KEGG)Citrate cycle (TCA cycle)
00020, GO:0005759	8(12)	9(6194)	1.52e-21	1.52e-20	ACO1, KGD2, IDH2, KGD1, IDH1, CIT1, FUM1, CIT3	(KEGG)Citrate cycle (TCA cycle) | (CC)mitochondrial matrix
00020, GO:0005739	6(12)	9(6194)	5.43e-15	5.43e-14	CIT2, LSC2, YJL200C, IDH2, LSC1, CIT1	(KEGG)Citrate cycle (TCA cycle) | (CC)mitochondrion
00020, GO:0042645	5(12)	7(6194)	5.83e-13	5.83e-12	ACO1, KGD2, LSC1, KGD1, IDH1	(KEGG)Citrate cycle (TCA cycle) | (CC)mitochondrial nucleoid
00020, 00630	5(12)	8(6194)	2.62e-12	2.62e-11	CIT2, ACO1, YJL200C, CIT1, CIT3	(KEGG)Citrate cycle (TCA cycle) | (KEGG)Glyoxylate and dicarboxylate metabolism

Règles d'association multi-niveaux

Les items forment des hiérarchies.

Les items au niveau inférieur ont des supports inférieurs

Les bases de transactions peuvent prendre en compte les niveaux



Règles d'association multi-niveaux

- On dispose d’une hiérarchie de concepts
- Les niveaux sont numérotés de 0 à n en commençant par le niveau le plus général
- Les concepts (ou items) de chaque niveau i sont numérotés de 1 à m_i avec m_i qui représente le nombre de concepts dans le niveau i
- Les données dont on dispose représentent les transactions avec les items du plus bas niveau
- Avec une chaîne de n digits, on peut identifier tous les items du plus bas niveau.

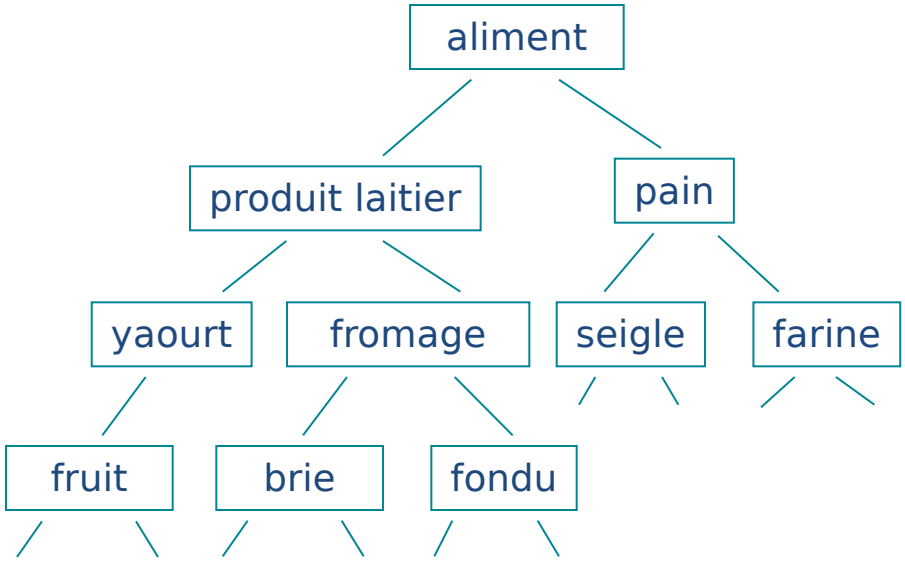
Règles d'association multi-niveaux

Les items forment des hiérarchies.

Les items au niveau inférieur ont des supports inférieurs

Les bases de transactions peuvent prendre en compte les niveaux

Item {111} représente le « yaourt fruit »



TID	Items
T1	{111, 121, 211, 221}
T2	{111, 211, 222, 323}
T3	{112, 122, 221, 411}
T4	{111, 121}
T5	{111, 122, 211, 221, 413}

Exploration multi-niveaux : support uniforme

- Approche Top-Down:
  - On utilise la propriété de non monotonie: Si un concept (ou ensemble de concepts) est non fréquent, alors tous ses fils sont non fréquents
  - Ex: Si produit_laitier est non fréquent alors yaourt ne l'est certainement pas.
  - Ex: Si {produit_laitier, pain} est non fréquent alors {yaourt, pain_au_seigle} ne l'est certainement pas
- Problème : difficulté de régler le support minimal

Exploration multi-niveaux : support décroissant

- Si niveau_i < niveau_j

  alors min-support (i) > min-support(j)

  - min-support du niveau 1 est 50% et

    min-support de niveau 2 est de 10%

- Dans ce cas, il y a différentes approches
  - Indépendance entre niveaux
  - Filtre sur un seul item
  - Filtre sur k items

Indépendance entre niveaux

- Dans ce cas, on n’utilise aucune connaissance sur les parents
  - Ce n’est pas parce qu’un parent n’est pas fréquent que le fils ne le sera pas
  - Ce n’est pas parce qu’un fils est fréquent que le parent le sera
- Le calcul peut être *Top-Down* ou *Bottom-Up*

Filtre sur k items

- Un k-itemset est vérifié au niveau i
- si les k-parents au niveau i-1 sont fréquents
  - {pain-au-seigle, yaourt} sera vérifié seulement si {pain, produit-laitier} est fréquent

- Une approche *Top-Down*

- Risque de perte de règles intéressantes aux niveaux inférieurs

Filtre sur un seul item

- Un item au niveau i sera examiné si son parent est considéré comme fréquent
  - Si Pain est fréquent

    alors on peut voir si Pain-au-seigle l’est.
  - Si produit-laitier est fréquent

    alors on peut voir si yaourt l’est.

- Approche *Top-Down*

Associations multi-niveaux : élimination des règles redondantes

- Certaines règles peuvent être redondantes à cause des relations de “parenté” entre items
- Exemple
  - Produit_laitier → pain_farine [s = 8%, c = 70%]
  - fromage → pain_farine [s = 2%, c = 72%]
- On dit que la première règle est un ancêtre de la seconde
- Une règle est redondante si son support est très proche du support prévu, en se basant sur sa règle ancêtre
  - Dans l’exemple, s’il y a 4 fils du nœud Produit_laitier dont Fromage, alors le support 2% est « prévisible »

Mesures d’intérêt

- Mesures objectives :
  - *support* et
  - *confiance*

- Mesures subjectives
  - Une règle est intéressante
  - Si elle est *inattendue* et/ou
  - *actionnable* (l’utilisateur peut faire quelque chose avec)

Critiques des notions de support et de confiance

- armi 5000 étudiants
  - 3000 jouent au basket
  - 3750 prennent des céréales
  - 2000 jouent au basket et prennent des céréales
- *Jouer au basket → prendre des céréales* [40%, 66.7%] n'est pas informative car il y a 75% d'étudiants qui prennent des céréales ce qui est plus que 66.7%.
- *jouer au basket → pas de céréales* [20%, 33.3%] est plus pertinente même avec un support et une confiance inférieurs

	basket	non basket	
céréales	2000	1750	3750
non céréale	1000	250	1250
sum(col.)	3000	2000	5000

Critiques des notions de support et de confiance

- Exemple 2:
  - X et Y: positivement corrélés,
  - X et Z, négativement corrélés
  - Les support et confiance de

    X → Z dominent

X	1	1	1	1	0	0	0	0
Y	1	1	0	0	0	0	0	0
Z	0	1	1	1	1	1	1	1

- Nous avons besoin d'une mesure de corrélation

$$corr_{A,B} = \frac{P(AB)}{P(A)P(B)}$$

- est aussi appelé le lift de

  A => B

Autres mesures

- Intérêt (corrélation) $\frac{P(A \wedge B)}{P(A)P(B)}$
  - Prendre en compte P(A) et P(B)
  - P(A ^ B)=P(B)*P(A),

    si A et B sont des événements indépendants
  - A et B négativement corrélés, si corr(A,B)<1.

X	1	1	1	1	0	0	0	0
Y	1	1	0	0	0	0	0	0
Z	0	1	1	1	1	1	1	1

Itemset	Support	Intérêt
X,Y	25%	2
X,Z	37,50%	0,9
Y,Z	12,50%	0,57

Exploration avec contraintes

- Exploration interactive où l'utilisateur pose des conditions en plus des minima de support et confiance
- Quels types de conditions ?
  - Type de connaissance recherchée: classification, association, etc.
  - Contraintes sur les données:
    - Trouver les paires de produits vendus à Toulouse en Décembre 98
  - Contraintes sur l'intérêt
    - support, confiance, corrélation
  - Contraintes sur les dimensions:
    - En rapport à région, prix, marque, catégorie client
  - Contraintes sur les règles
    - Nombres de prédicats dans le corps

Exploration avec contraintes

- Exemple, Base: (1) trans (TID, Itemset),

  (2) itemInfo (Item, Type, Prix)

- Une requête d'association contrainte (RAC) est une expression de la forme {($S1 \rightarrow S2$)|C },
  - où C est un ensemble de contraintes sur S1 et S2 incluant la contrainte de fréquence
- Une classification de contraintes (à une variable) :
  - Contraintes de classe :

    ex. S2 I1
  - Contrainte de domaine :

    ex. S2.Prix < 100
  - Contraintes d'agrégation :

    ex. avg(S2.Prix) 100

Optimisation en présence de contraintes

- Soit une RAC = { ($S1 \rightarrow S2$) | C }, l'algorithme doit être:
  - Correct : Il ne trouve que les itemsets fréquents qui satisfont C
  - Complet : Il trouve tous les itemsets fréquents qui satisfont C
- Solution naïve :
  - Appliquer A priori pour trouver les itemsets fréquents puis éliminer les itemsets ne satisfaisant pas C
- Autre approche :
  - Analyser les propriétés des contraintes pour les intégrer dans Apriori lors de la phase de l'exploration des itemsets fréquents.

Contraintes anti-monotones et monotones

- Une contrainte C_a est **anti-monotone**

  ssi pour chaque itemset S,

  si S ne satisfait pas C_a,

  alors aucun de ses sur-ensembles ne satisfait C_a

- C_m est **monotone**

  ssi pour chaque S,

  si S satisfait C_m,

  alors chacun de ses sur-ensembles satisfait C_m

Propriétés des contraintes anti-monotones

- Anti-monotonicité :

  Si S viole la contrainte

  alors chaque sur-ensemble de S viole aussi la contrainte

- Exemples :
  - *sum(S.Prix)* v est anti-monotone
  - *sum(S.Prix)* v n'est pas anti-monotone
  - *sum(S.Prix)* $= v$ est partiellement anti-monotone

- Application :
  - Pousser la condition "*sum(S.prix)* *1000*" lors des itérations du calcul des ensembles fréquents

Contraintes monotones

- Le problème avec ces contraintes est que l'on ne peut pas les intégrer lors de l'évaluation des itemsets fréquents

- Exemples
  - Sum(S.prix) 100 est monotone
  - *Sum(S.prix) v* n'est pas monotone

- Par contre, si S' S et Sum(S.prix) 100 alors ce n'est pas la peine de tester Sum(S'.prix)

Contraintes succinctes

- Une contrainte est succincte si on peut générer statiquement tous les itemsets qui la satisfont
- Exemple :
  - A S avec I={A, B, C, D}
  - L'ensemble des itemsets S qui satisfont la condition sont tous les sous-ensembles de I qui contiennent A

Contrainte convertible

- Supposons que tous les items dans les motifs soient triés selon l'ordre O

- Une contrainte C est convertible anti-monotone
  ssi un motif S satisfait C implique que chaque suffixe de S (respectivement à O) satisfait aussi C

- Une contrainte C est convertible monotone
  ssi un motif S satisfait C implique que chaque motif dont S est un suffixe (respectivement à O) satisfait aussi C

Exemple de contraintes convertibles Avg(S) V

- Soit S l'ensemble de valeurs (par ordre décroissant)

  {9, 8, 6, 4, 3, 1}

- Avg(S) v est monotone convertible respectivement à S
  - Si S est un suffixe de S_1,

    alors avg(S_1) avg(S)
    - {8, 4, 3} est un suffixe de {9, 8, 4, 3}
    - avg({9, 8, 4, 3})=6 avg({8, 4, 3})=5

  - Si S satisfait avg(S) v, alors S_1 aussi
    - {8, 4, 3} satisfait avg(S) 4, ainsi que

      {9, 8, 4, 3}