1.3 Extraction de règles d'association

L'extraction de règles d'association est l'une des techniques les plus populaires de la fouille de données. Ce problème, surnommé analyse du panier de la ménagère, a été introduit pour la première fois en 1993 [Agrawal et al., 1993] pour analyser des bases de données de la grande distribution. Depuis lors, ce problème a été intensément étudié pour son utilité dans de nombreux domaines d'application tels que les systèmes de recommandations, la bio-informatique ou encore les diagnostics médicaux.

L'extraction de règles d'association a pour objectif de découvrir des relations entre les variables de grandes bases de données. Une règle d'association dans la grande distribution pourrait être $Café,\ Thé \Rightarrow Sucre$ qui signifie que si le consommateur achète du café et du thé alors il y a de grandes chances qu'il achète également du sucre.

Après avoir posé le problème, nous présentons l'approche naïve ainsi que les problèmes qu'elle engendre. Nous étudions ensuite l'algorithme de référence pour l'extraction de règles d'association.

1.3.1 Définition du problème

Avant de définir le problème, nous devons expliquer le vocabulaire utilisé.

Définition 2 - Item, motif et *k*-motif :
Soit $\mathcal{I} = \{i_1, i_2, .., i_p\}$ un ensemble de p items i, où chaque item est une variable binaire de la base de données. Un ensemble d'items est appelé un motif, et plus spécifiquement on dit que X est un k-motif s'il est composé de k items : $X = \{i_1, i_2, .., i_k\}$.

Par convention, on utilise les minuscules pour représenter les items et les majuscules pour représenter les motifs. Par conséquent, x est un item et X est un motif (*qui peut être aussi un item*).

Définition 3 - Transaction :
Soit $\mathcal{D} = \{t_1, t_2, .., t_n\}$ un ensemble de n transactions t, où chaque transaction t est un ensemble d'items tel que $t \subseteq \mathcal{I}$. Une transaction t de $\mathcal{D}$ contient X, un ensemble d'items de $\mathcal{I}$, si $X \subseteq t$.

Définition 4 - Règle d'association :
Une règle d'association est une implication de la forme $X \Rightarrow Y$, où $X \subseteq \mathcal{I}$, $Y \subseteq \mathcal{I}$, et $X \cap Y = \varnothing$.

Une règle $X \Rightarrow Y$ indique que les transactions possédant le motif X ont tendance à posséder le motif Y. Cependant, il n'existe aucune relation de causalité entre X et Y : la présence de X ne cause pas la présence de Y.

Définition 5 - Prémisse, antécédent, conclusion et conséquent :
La partie gauche de la règle est appelée la prémisse ou l'antécédent et la partie droite est la conclusion ou le conséquent.

Pour une règle $X \Rightarrow Y$, X est donc la prémisse ou l'antécédent et Y est donc la conclusion ou le conséquent.

Définition 6 - Support, support relatif, support absolu, support minimum min_{sup} et motif fréquent :
Le support représente la fréquence de la règle ou la portée de la règle. Par extrapolation, on utilise la probabilité que la règle soit présente que l'on nomme support relatif par rapport au support absolu qui correspond aux nombres d'occurrences. La règle $X \Rightarrow Y$ a donc un support s si $s\%$ des transactions de $\mathcal{D}$ contiennent $X \cup Y$ également noté par XY simplification d'écriture. Autrement dit, le support correspond à la probabilité que la prémisse X et la conclusion Y soient vraies. Par $\mathcal{D}_{X \cup Y}$, nous indiquons l'ensemble de toutes les transactions qui contiennent $X \cup Y$, $\mathcal{D}_{X \cup Y} = \{t \in \mathcal{D} \mid X \cup Y \subseteq t\}$. Le support s de $X \Rightarrow Y$ est calculé comme $s = sup(X \Rightarrow Y) = sup(X \cup Y) = sup(XY) = P(XY) = \frac{|\mathcal{D}_{X \cup Y}|}{n}$. Un motif X qui respecte le support minimum (i.e. $sup(X) \geqslant min_{sup}$) est dit fréquent.

Définition 7 - Confiance et confiance minimum min_{conf} :
La confiance représente la force de la règle. La règle $X \Rightarrow Y$ a une confiance c si $c\%$ des transactions de $\mathcal{D}$ qui contiennent X contiennent également Y. Autrement dit, la confiance est la probabilité conditionnelle que la conclusion Y soit vraie sachant que la prémisse X est vraie : c'est-à-dire $P(Y|X)$. La confiance c de $X \Rightarrow Y$ est calculée comme $c = conf(X \Rightarrow Y) = \frac{sup(XY)}{sup(X)}$. Il existe également un seuil pour la confiance minimum min_{conf} afin de permettre aux utilisateurs de ne sélectionner que les règles plus pertinentes (i.e. $conf(X \Rightarrow Y) \geqslant min_{conf}$).

Définition 8 - Règle d'association valide :
Une règle d'association est dite valide si ses valeurs pour le support et pour la confiance sont supérieures aux seuils minimaux fixés par l'utilisateur : min_{sup} et min_{conf}.

Ainsi, si la règle $Café \Rightarrow Sucre$ a un support de 0,01 et une confiance de 0,90, alors cela signifie que 1% des consommateurs ont acheté du café et du sucre en même temps, et que parmi ceux qui ont acheté du café, 90% d'entre eux ont également acheté du sucre.

Le problème d'extraction de règles d'association à partir de $\mathcal{D}$ consiste à générer toutes les règles d'association valides. Cette approche est connue sous le nom d'approche support/confiance. Le tableau 1.1 récapitule les contraintes qu'une règle $X \Rightarrow Y$ doit respecter afin d'être considérée comme valide.

$X \Rightarrow Y$
$sup(X \Rightarrow Y) \geq min_{sup}$
$conf(X \Rightarrow Y) \geq min_{conf}$

Tableau 1.1 – Règles valides

La découverte de ces règles va avoir différents objectifs en fonction des données analysées. Dans l'exemple de la grande distribution donné précédemment, cela permet d'étudier le comportement des clients, et de mettre en place une meilleure stratégie marketing. En médecine, cela permet par exemple de détecter les patients à risque pour une maladie donnée. C'est pourquoi la recherche d'algorithmes efficaces de telles règles a été un problème majeur de cette communauté. Nous allons maintenant expliquer un algorithme naïf qui peut être employé pour l'extraction des règles d'association positives.

1.3.2 Algorithme naïf

Le problème de génération des règles d'association peut être décomposé en deux sous-problèmes :

1. Générer tous les motifs fréquents.
2. Générer toutes les règles d'association valides à partir des motifs fréquents.

Une approche naïve serait d'énumérer l'ensemble des motifs et de vérifier un à un le support de chaque motif, puis de générer l'ensemble des règles pour chaque motif fréquent et de vérifier une à une la confiance de chaque règle. Sur la figure 1.2, nous pouvons voir le treillis de l'ensemble des motifs que nous aurions à étudier pour une base comportant quatre items : A, B, C et D. Dans ce cas là, nous aurions donc 15 motifs à étudier puisqu'il faut vérifier le support de tous les motifs possédant au moins un item.

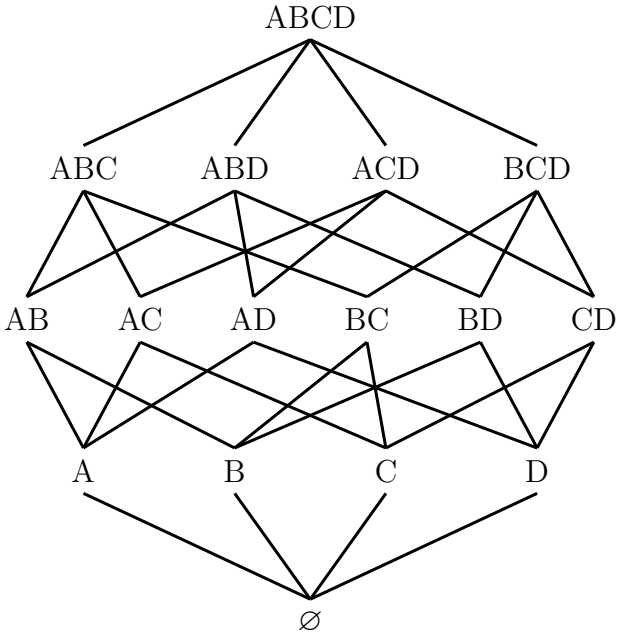

FIGURE 1.2 – Treillis des motifs

Cette méthode est cependant impossible à appliquer à cause des problèmes d'explosion combinatoire qu'elle engendre. En effet, pour une base de données comportant p items, le nombre maximal de motifs possibles est $\sum_{k=1}^{p} \binom{p}{k}$ ou plus simplement $2^p - 1$. Pour une base de données comportant 100 items, cela correspond à environ $1,27 \times 10^{30}$ motifs. L'identification des motifs fréquents est donc coûteuse en calcul et l'espace de recherche grandit exponentiellement en même temps que p. Le coût de calcul sera donc très important lorsque l'on s'intéressera aux bases de données comportant des milliers d'items, comme celles de la grande distribution.

Quant à la phase de génération des règles, le nombre de règles potentielles à analyser dépend du nombre de motifs fréquents mais également de leur taille. Ainsi pour chaque k-motif fréquent de taille supérieure ou égale à 2, il existe $2^k - 2$ règles potentielles. Dans le pire des cas, c'est-à-dire si tous les motifs sont fréquents, le nombre de règles à étudier représente : $\sum_{k=2}^{p} \binom{p}{k} \times (2^k - 2)$. En reprenant la base de l'exemple précédent comportant 100 items, le nombre de règles potentielles correspond à environ $5,15 \times 10^{47}$.

Même si la phase de génération des règles est importante, le principal goulot d'étranglement d'une telle approche provient de la génération des motifs candidats. Pas-

sons maintenant à l'algorithme de référence pour l'extraction de règles d'association positives qui comble en partie ces deux problèmes.

1.3.3 Algorithme Apriori

Comme la méthode naïve, l'algorithme `Apriori` [Agrawal and Srikant, 1994], va se décomposer en deux phases. Des optimisations vont être apportées dans les deux parties de l'algorithme, à savoir dans la recherche des motifs fréquents et dans la recherche des règles. Commençons par analyser ces optimisations et pour cela étudions les méthodes utilisées dans les deux étapes du processus. Nous déroulerons ensuite `Apriori` sur un exemple.

➤ Génération des motifs fréquents

L'algorithme `Apriori` utilise une propriété du support qui va permettre de ne pas parcourir tout l'espace de recherche et par conséquent va accélérer le processus d'extraction des motifs fréquents. Le support est une mesure anti-monotone.

Définition 9 - Mesure anti-monotone :
Une mesure $\mathcal{M}$ est dite anti-monotone si et seulement si : $\forall X, Y \subseteq \mathcal{I}$, *si* $X \subsetneq Y$ *et* $\mathcal{M}(Y)$ *alors* $\mathcal{M}(X)$.

Autrement dit, une mesure est anti-monotone si lorsqu'elle est vérifiée pour un motif, elle est forcément vérifiée pour un sous-ensemble englobant ce motif. Il existe également des mesures monotones comme nous le verrons par la suite. Profitons-en pour donner la définition.

Définition 10 - Mesure monotone :
Une mesure $\mathcal{M}$ est dite monotone si et seulement si : $\forall X, Y \subseteq \mathcal{I}$, *si* $X \subsetneq Y$ *et* $\mathcal{M}(X)$ *alors* $\mathcal{M}(Y)$.

Autrement dit, une mesure est monotone si lorsqu'elle est vérifiée pour un motif, elle est forcément vérifiée pour un sur-ensemble englobant ce motif.

Cette propriété définit donc que le support de tout sur-ensemble Y d'un motif X est inférieur ou égal au support de X, c'est-à-dire que $\forall\ Y \supsetneq X$, $sup(Y) \leqslant sup(X)$. Par conséquent, tous les sur-ensembles d'un motif non fréquent sont non fréquents. Par exemple, si C est non fréquent, aucun sur-ensemble de C ne peut être fréquent comme par exemple AC ou BC. Cette propriété va permettre d'élaguer un k-motif lorsqu'au moins un de ses sous-ensembles de taille *(k-1)* n'est pas fréquent. La figure 1.3 représente le treillis de l'ensemble des motifs à étudier si C n'est pas fréquent.

La vérification du support de C nous permet d'élaguer 7 motifs dans l'étude et qui sont les suivants : $\{AC, BC, CD, ABC, ACD, BCD, ABCD\}$. Cette propriété va donc avoir une incidence sur l'ordre dans lequel on génère les motifs. Pour éviter la vérification du support sur un maximum de motifs, il faut donc générer les motifs par ordre croissant de taille.

Nous venons de présenter la propriété du support qui permet d'accélérer le processus d'extraction des motifs fréquents. regardons maintenant comme elle est utilisée par `Apriori`. Précisons tout d'abord le prérequis que nécessite `Apriori` : les motifs doivent être écrits dans l'ordre lexicographique afin d'éviter de considérer plusieurs fois le même

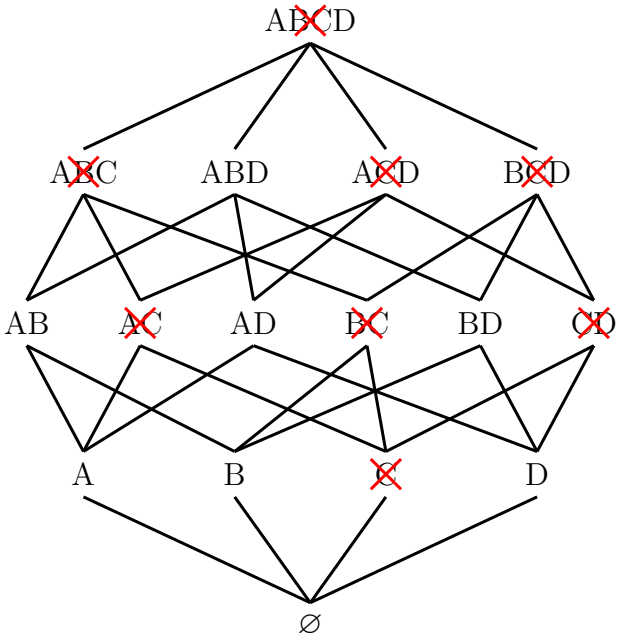

FIGURE 1.3 – Treillis des motifs potentiellement fréquents quand C n'est pas fréquent

motif. AB et BA représentent donc le même motif. Pour générer l'ensemble des motifs fréquents l'*algorithme 1*, nommé *fréquents* va être utilisé.

Algorithme 1 : *fréquents* - Génération des motifs fréquents

Entrées : base de données $\mathcal{D}$, support minimum min_{sup}
Sortie : ensemble des motifs fréquents F

```
1  F = ∅
2  C_1 = {i ∈ I}
3  pour (k = 1; C_k ≠ ∅; k++) faire
4      F_k = ∅
5      pour tout motif candidat X ∈ C_k faire
6          s = support(D, X)
7          si s ⩾ min_sup alors
8              F_k = F_k ∪ {X}
9      C_{k+1} = candidats(F_k)
10     F = F ∪ F_k
11 retourner F
```

Cet algorithme commence (*ligne 1*) par initialiser l'ensemble C_1 des *1*-motifs candidats à l'ensemble de tous les items i de la base de données $\mathcal{D}$ passée en paramètre. Le processus suivant (*lignes 2 à 9*) va être réitéré jusqu'à ce que l'on n'obtienne plus de candidat ($C_k \neq \varnothing$) à partir de l'ensemble F_{k-1} des motifs fréquents X de niveau inférieur (*ligne 8*). En effet, la génération des candidats effectuée par la fonction *candidats* (*cf. algorithme 2*), repose sur la propriété anti-monotone du support et va donc s'effectuer uniquement à partir des motifs fréquents afin d'éviter de vérifier le support de certains candidats que l'on sait par avance trop faible. Pour un niveau k donné, on commence par initialiser l'ensemble des k-motifs fréquents F_k à l'ensemble vide (*ligne 3*). Ensuite

pour tous les candidats X de C_k (*ligne 4*), on calcule le support avec la fonction *support* (*ligne 5*) qui interroge la base de données. Puis, si le motif X est fréquent (*ligne 6*), on le stocke dans l'ensemble F_k qui servira, une fois l'ensemble des motifs X de C_k parcouru, à générer les candidats de niveau supérieur (*ligne 8*). La dernière étape (*ligne 9*) consiste à ajouter l'ensemble des k-motifs fréquents F_k à l'ensemble des motifs fréquents F. Lorsqu'il n'existe plus de candidats à analyser, on retourne l'ensemble des fréquents F (*ligne 10*).

Après avoir exposé la recherche des motifs fréquents, nous expliquons la fonction *candidats*, correspondant à l'*algorithme 2*, et qui permet de générer le prochain niveau de candidats.

Algorithme 2 : *candidats* - Génération des *(k+1)*-motifs candidats

Entrée : ensemble des k-motifs fréquents F_k
Sortie : ensemble des *(k+1)*-motifs candidats C_{k+1}

```
1 C_{k+1} = F_k ⋈ F_k
2 pour tout motif candidat potentiel X ∈ C_{k+1} faire
3     pour tout k-motif Y ⊂ X faire
4         si Y ∉ F_k alors
5             C_{k+1} = C_{k+1} \ X
6 retourner C_{k+1}
```

Cette fonction va prendre en paramètre l'ensemble des k-motifs fréquents F_k et va retourner les candidats à analyser à la prochaine itération. L'algorithme commence par générer l'ensemble C_{k+1} des candidats potentiels (*ligne 1*) en combinant l'ensemble F_k avec lui même. Deux k-motifs peuvent être combinés pour créer un nouvel *(k+1)*-motif candidat si et seulement s'ils ont en commun les *(k-1)* premiers items. Par exemple, AB et AC ont leur premier item A en commun et peuvent donc se combiner pour créer le motif ABC. Cependant, les motifs AC et BC ne peuvent pas se combiner car même si C est commun, il n'est pas le premier item. La seconde étape (*lignes 2 à 5*) est l'élagage des candidats dont tous les sous-ensembles ne sont pas fréquents. Pour se faire, il faut vérifier pour chaque *(k+1)*-motif X du nouvel ensemble C_{k+1} (*ligne 2*), que chaque sous-motif Y de taille k (*ligne 3*) composant X est bien fréquent. Si le sous-motif Y n'est pas fréquent (*ligne 4*) alors le motif X analysé doit être retiré de l'ensemble C_{k+1} (*ligne 5*). Une fois que tous les motifs X sont analysés, on retourne l'ensemble des *(k+1)*-motifs candidats C_{k+1} (*ligne 6*).

La première étape du processus étant finie, passons à la seconde étape qui consiste à générer les règles à partir des motifs fréquents.

➤ Génération des règles valides

Pour la génération des règles, on utilise une propriété de la confiance pour élaguer certaines règles sans avoir à calculer leur confiance.

Propriété 1 - Propriété de la confiance :
Cette propriété est définie comme suit :
$\forall (X, Y, Z)$ *tel que* $Z \subsetneq Y \subsetneq X$, $conf(Z \Rightarrow X \backslash Z) \leqslant conf(Y \Rightarrow X \backslash Y)$.

Preuve :

$$\begin{array}{lrcl} & conf(Z \Rightarrow X\backslash Z) & \leqslant & conf(Y \Rightarrow X\backslash Y) \\ \Leftrightarrow & \frac{sup(Z \Rightarrow X\backslash Z)}{sup(Z)} & \leqslant & \frac{sup(Y \Rightarrow X\backslash Y)}{sup(Y)} \\ \Leftrightarrow & \frac{sup(X)}{sup(Z)} & \leqslant & \frac{sup(X)}{sup(Y)}, \text{ puisque } Z \subsetneq Y. \end{array}$$

Cette propriété de la confiance nous permet donc de déduire deux choses :

1. si la confiance de la règle $Z \Rightarrow X \backslash Z$ n'est pas valide alors la confiance de la règle $Y \Rightarrow X \backslash Y$ ne le sera pas non plus.
2. la proposition contraposée, si la confiance de la règle $Z \Rightarrow X \backslash Z$ est valide alors la confiance de la règle $Y \Rightarrow X \backslash Y$ le sera également.

Exemple : si A a un support absolu de 2, le support de AB est inférieur ou égal à 2. Par conséquent, si la confiance de la règle $AB \Rightarrow C$ ne respecte pas le seuil minimum de confiance alors on sait par avance que la confiance de la règle $A \Rightarrow BC$ ne le sera pas non plus. En effet, pour calculer la confiance de ces deux règles, il faut diviser le support du motif ABC par le support de la prémisse. Le numérateur est donc commun pour les deux calculs et seul le dénominateur change, or le support de AB apparaissant au dénominateur pour le calcul de la confiance de la règle $AB \Rightarrow C$ ne peut être que plus petit ou égal à celui du support de A apparaissant au dénominateur pour le calcul de la confiance de la règle $A \Rightarrow BC$.

Cette propriété va avoir une incidence sur l'ordre d'étude des différentes règles pour un même motif. Pour éviter la vérification de la confiance sur un maximum de règles, il faut donc commencer par analyser les règles qui possèdent un motif en conclusion le plus petit possible. Regardons maintenant comment ces optimisations ont été utilisées dans l'algorithme `Apriori`.

Pour générer l'ensemble des règles, Agrawal *et al.* vont utiliser l'algorithme *règles* (*cf. algorithme* 3) qui recherche les règles possédant un seul item en conséquence, puis qui fait appel à l'algorithme *autresRègles* (*cf. algorithme* 4) pour générer les autres règles, c'est-à-dire celles possédant plusieurs items en conclusion.

L'algorithme *règles* (*cf. algorithme* 3) prend en paramètre l'ensemble des fréquents F ainsi que le seuil minimum de la confiance min_{conf}. Cet algorithme se déroule en deux phases. La première phase (*lignes 1 à 9*) consiste à générer, à partir des motifs fréquents, les règles possédant un seul item en conclusion. La seconde phase (*ligne 10*) fait appel au second algorithme afin de générer, pour chaque motif, les règles possédant plusieurs items en conclusion. Concernant la première phase, on commence par parcourir tous

Algorithme 3 : *règles* - Génération des règles d'association

Entrées : ensemble des motifs fréquents F, confiance minimum min_{conf}
Sortie : ensemble des règles d'association valides R

```
1  R = ∅
2  pour tout k-motif X ∈ F tel que k > 1 faire
3      E_1 = ensemble des 1-motifs ⊂ X
4      pour tout Y ∈ E_1 faire
5          c = conf(X\Y ⇒ Y)
6          si c ⩾ min_conf alors
7              R = R ∪ {X\Y ⇒ Y}
8          sinon
9              E_1 = E_1\Y
10     R = R ∪ autresRègles(X, E_1, min_conf)
11 retourner R
```

les motifs fréquents de taille strictement supérieure à 1 (*ligne 2*) puisque l'on ne peut pas générer de règles comportant un seul item. On récupère dans l'ensemble E_1 tous les items qui composent le motif X en cours d'étude (*ligne 3*). Ensuite, pour chaque item Y de l'ensemble E_1 (*ligne 4*), on calcule la confiance de la règle $X \backslash Y \Rightarrow Y$ (*ligne 5*). Si la confiance de la règle est supérieure ou égale au seuil min_{conf} (*ligne 6*), alors la règle est ajoutée à l'ensemble des règles valides R (*ligne 7*). Si la règle n'est pas valide (*ligne 8*), le motif Y va être retiré de l'ensemble E_1 (*ligne 9*). Une fois tous les items Y parcourus, l'ensemble des règles possibles possédant un seul item en conclusion sera généré pour un motif donné et E_1 contiendra uniquement les conclusions qui ont permis de générer les règles valides. Ce nouvel ensemble E_1 sera utilisé dans la fonction *autresRègles* (*ligne 10*) pour générer les autres règles, c'est-à-dire celles possédant plusieurs items en conclusion. La fonction *autresRègles* repose sur la propriété anti-monotone du support et va nous éviter de calculer la confiance de certaines règles que l'on sait par avance trop faible.

La fonction récursive *autresRègles* (*cf. algorithme* 4) va retourner l'ensemble des règles valides possédant plusieurs items en conclusion pour chaque motif X passé en paramètre. Le second paramètre de la fonction est l'ensemble E_m des m-motifs conclusion Y pour lesquels la règle $X \backslash Y \Rightarrow Y$ est valide. L'algorithme commence par vérifier s'il est possible de générer d'autres règles à partir du motif X (*ligne 2*). En effet, il faut vérifier que la taille k du motif X passé en paramètre est strictement supérieure aux tailles $(m+1)$ des futures conclusions. Puis, on appelle la fonction *candidats* (*ligne 3*) afin de générer les conclusions de taille $(m+1)$ à partir des conclusions de taille m qui ont mené à des règles valides à l'itération précédente. Ce nouvel ensemble de conclusions est stocké dans l'ensemble E_{m+1}. Ensuite, pour chaque item Y de l'ensemble E_{m+1} (*ligne 4*) on calcule la confiance de la règle $X \backslash Y \Rightarrow Y$ (*ligne 5*). Si la confiance de la règle est supérieure ou égale au seuil min_{conf} (*ligne 6*) alors la règle est ajoutée à l'ensemble des règles valides R' (*ligne 7*). Si la règle n'est pas valide (*ligne 8*), le motif Y va être retiré de l'ensemble E_{m+1} (*ligne 9*). Une fois tous les items Y parcourus, l'ensemble des règles possibles possédant $(m+1)$ items en conclusion est généré pour un motif X donné et E_{m+1} contiendra uniquement les

conclusions qui ont permis de générer les règles valides. Ce nouvel ensemble E_{m+1} sera à son tour utilisé dans la fonction récursive *autresRègles* (*ligne 10*) pour générer les règles possédant (*m+2*) items en conclusion.

Algorithme 4 : *autresRègles* - Génération des règles d'association composées de plus d'un item en conclusion

Entrées : k-motif fréquent X, ensemble des m-motifs en conclusion E_m, confiance minimum min_{conf}
Sortie : ensemble des règles d'association valides R' possédant un *(m+1)*-motif en conclusion pour le motif X

```
1  R' = ∅
2  si k > m + 1 alors
3      E_{m+1} = candidats(E_m)
4      pour tout Y ∈ E_{m+1} faire
5          c = conf(X\Y ⇒ Y)
6          si c ⩾ min_conf alors
7              R' = R' ∪ {X\Y ⇒ Y}
8          sinon
9              E_{m+1} = E_{m+1}\Y
10     R' = R' ∪ autresRègles(X, E_{m+1}, min_conf)
11 retourner R'
```

➤ Discussion

`Apriori` est l'algorithme de référence pour l'extraction de règles d'association mais il possède cependant certaines faiblesses.

Une première faiblesse provient de la nécessité de récupérer le support des motifs dans la base de données. En effet, un passage est nécessaire pour chaque taille de motif. Par exemple, si le motif de plus grande taille possède 1000 items alors l'algorithme requiert 1000 balayages complets de la base de données. Ces balayages sont assez coûteux en terme d'entrées/sorties si la base de données ne peut pas être chargée en mémoire.

Une seconde faiblesse provient du dilemme pour choisir le seuil du support. En effet, l'extraction de règles avec un support élevé peut entraîner la perte des pépites de connaissances. Les pépites sont des règles avec un support faible mais possédant une confiance élevée. Ces règles sont intéressantes puisqu'elles apportent en général de l'information inattendue et surprenante à l'utilisateur. Néanmoins, le choix d'un support faible peut entraîner un nombre prohibitif de règles, dont la plupart sont inintéressantes et redondantes.

Depuis le célèbre algorithme `Apriori` [Agrawal and Srikant, 1994], il y a eu de nombreuses variantes et améliorations et notamment les algorithmes `Eclat` [Zaki et al., 1997] et `FP-Growth` [Han et al., 2000] qui comblent en partie ces faiblesses. `Eclat` va stocker pour chaque item la liste des transactions qui le contiennent dans un ensemble. Le support des motifs est ensuite calculé en utilisant les intersections d'ensemble. `FP-Growth` utilise une structure compacte appelée `FP-Tree` qui va permettre d'extraire les motifs fréquents sans générer de candidats. De plus, alors qu'`Apriori` interroge la base pour chaque niveau

de motifs candidats générés, `FP-Growth` nécessite seulement deux passages, ce qui accélère encore les traitements.

➤ Exemple

Déroulons maintenant l'algorithme `Apriori` sur un petit exemple (*cf. tableau* 1.2) afin d'éclaircir son fonctionnement. Cet exemple comporte 5 items : A, B, C, D et E ; et 4 transactions. Les 0 et les 1 représentent respectivement l'absence ou la présence d'un item dans la transaction. La première transaction contient donc les items A, C et D alors que les items B et E sont absents. Nous allons prendre les paramètres suivants : 0,25 pour le support minimum et 0,80 pour la confiance minimum.

A	B	C	D	E
1	0	1	1	0
0	1	1	0	1
1	1	1	0	0
0	1	0	0	1

Tableau 1.2 – Exemple de base de données

La première étape consiste à rechercher les motifs fréquents.

1) Extraction des motifs fréquents

1-Motifs fréquents :

Pour rechercher les motifs fréquents, la première étape consiste à récupérer l'ensemble des motifs de taille 1. Une fois les items A, B, C, D et E récupérés, la fonction *support* est ensuite utilisée afin de calculer les différents supports dans la base de données. Le support de chaque item est ensuite comparé à la valeur du support minimum afin de vérifier si c'est un motif fréquent. Pour un support de 0,25, un motif sera fréquent s'il apparaît au moins une fois puisque $0{,}25 \times 4 = 1$ (*support minimum* $\times$ *nombre de transactions*). Les items A, B, C, D et E sont donc fréquents. Le tableau 1.3 récapitule les 5 items fréquents avec leur support.

Item	Support	Item	Support	Item	Support
A	0,50	B	0,75	C	0,75
D	0,25	E	0,50		

Tableau 1.3 – Items fréquents de taille 1 accompagnés de leur support

2-Motifs candidats :

La prochaine étape consiste à générer les motifs candidats de taille 2 à partir des motifs fréquents de taille 1. Pour se faire il suffit de les combiner. Deux k-motifs fréquents

peuvent être combinés pour créer un nouvel *(k+1)*-motif candidat si et seulement ils ont en commun les *(k-1)* premiers items. Par conséquent A et B peuvent se combiner pour former AB car ils n'ont pas besoin d'avoir d'items en communs. Les motifs candidats de taille 2 sont référencés dans le tableau 1.4.

2-Motif					
AB	AC	AD	AE	BC	BD
BE	CD	CE	DE		

Tableau 1.4 – Motifs candidats de taille 2

2-Motifs fréquents :

Le support est ensuite calculé pour chaque motif candidat puis comparé au support minimum. Le tableau 1.5 restitue les motifs fréquents parmi les 2-motifs candidats du tableau 1.4.

2-Motif	Support	2-Motif	Support	2-Motif	Support
AB	0,25	AC	0,50	AD	0,25
BC	0,50	BE	0,50	CD	0,25
CE	0,25				

Tableau 1.5 – Motifs fréquents de taille 2 accompagnés de leur support

3-Motifs candidats :

Le tableau 1.6 donne les candidats potentiels de taille 3 générés à partir des 2-motifs fréquents du tableau 1.5. Les motifs candidats potentiels sont combinés que s'ils ont leur premier item en commun. AB et AC ont donc été combinés car ils ont A en commun.

3-Motif		
ABC	ABD	ACD
BCE	CDE	

Tableau 1.6 – Motifs candidats potentiels de taille 3

Le tableau 1.7 donne les candidats de taille 3 conservés à partir des 3-motifs candidats potentiels du tableau 1.6. ABC sera ensuite conservé car BC fait également partie des fréquents. En effet il faut vérifier que tous les sous-motifs soient fréquents. Lorsque AB et AD vont être combinés, ABD ne sera pas conservé car BD n'est pas fréquent.

3-Motif		
ABC	ACD	BCE

Tableau 1.7 – Motifs candidats de taille 3

3-Motifs fréquents :

On recherche ensuite les motifs fréquents parmi les motifs candidats. Les motifs fréquents de taille 3 sont renseignés dans le tableau 1.8.

3-Motif	Support	3-Motif	Support	3-Motif	Support
ABC	0,25	ACD	0,25	BCE	0,25

TABLEAU 1.8 – Motifs fréquents de taille 3 accompagnés de leur support

4-Motifs candidats :

L'ensemble des motifs fréquents n'est toujours pas vide, donc nous pouvons continuer la génération des candidats du prochain niveau. Cependant, l'algorithme de génération des motifs fréquents s'arrêtent puisque les motifs fréquents de taille 3 ne sont plus combinables.

2) Génération des règles valides

La prochaine étape va être la génération des règles à partir des motifs fréquents. Prenons à titre d'exemple, le motif fréquent ACD et cherchons les règles valides.

Règles possédant un seul item en conclusion :

La première étape de la recherche des règles va être de rechercher les règles possédant un seul item en conclusion. Le tableau 1.9 expose les règles possédant un seul item en conclusion accompagnées du calcul de la confiance.

Règle	Confiance
$AC \Rightarrow D$	$\frac{sup(ACD)}{sup(AC)} = \frac{0,25}{0,50} = 0,50$
$AD \Rightarrow C$	$\frac{sup(ACD)}{sup(AD)} = \frac{0,25}{0,25} = 1$
$CD \Rightarrow A$	$\frac{sup(ACD)}{sup(CD)} = \frac{0,25}{0,25} = 1$

TABLEAU 1.9 – Règles possédant un seul item en conclusion pour le motif ACD

Après l'étude des règles possédant un seul item en conclusion pour le motif ACD, les règles $AD \Rightarrow C$ et $CD \Rightarrow A$ respectent le seuil de confiance minimum ($min_{conf} = 0,80$) et vont être conservées.

Règles possédant plusieurs items en conclusion :

La seconde étape permet de générer les règles possédant plusieurs items en conclusion. Les conclusions A et C ont permis de générer deux règles valides. Par conséquent, pour le prochain niveau de recherche des règles, les deux conclusions vont être combinées afin de créer une nouvelle conclusion AC. La combinaison des conclusions se déroule comme

pour la combinaison des motifs lors de la recherche des motifs fréquents. La suite de l'étude est présentée dans le tableau 1.10.

Règle	Confiance
$D \Rightarrow AC$	$\frac{sup(ACD)}{sup(D)} = \frac{0,25}{0,25} = 1$

TABLEAU 1.10 – Règle possédant deux items en conclusion pour le motif ACD

La règle $D \Rightarrow AC$ possède une confiance supérieure au seuil minimum de confiance ($min_{conf} = 0,80$) et va par conséquent être conservée. Le même processus est effectué pour tous les autres motifs fréquents. Les résultats obtenus sont résumés dans le tableau 1.11.

Motif fréquent	Règle	Confiance	Motif fréquent	Règle	Confiance
AC	$A \Rightarrow C$	1	AD	$D \Rightarrow A$	1
BE	$E \Rightarrow B$	1	CD	$D \Rightarrow C$	1
ABC	$AB \Rightarrow C$	1	ACD	$AD \Rightarrow C$	1
ACD	$CD \Rightarrow A$	1	ACD	$D \Rightarrow AC$	1
BCE	$CE \Rightarrow B$	1			

TABLEAU 1.11 – Règles extraites sur la base d'exemple

En conclusion, `Apriori` génère 9 règles $X \Rightarrow Y$ que l'on nomme règles positives par opposition aux règles négatives que l'on cherche à extraire lorsque l'on s'intéresse à l'extraction des règles utilisant également l'absence de variables comme motif. Ces règles négatives sont présentées dans la section suivante.

1.4 Extraction de règles d'association négatives

Avant de justifier l'intérêt de l'extraction de ces règles négatives, nous définissons ce qu'est une règle négative. Nous présentons par la suite un algorithme naïf ainsi que les différentes méthodes existantes dans la littérature permettant de les extraire.

1.4.1 Définition du problème

L'extraction de règles d'association négatives permet donc d'extraire des règles dans lesquelles la présence ainsi que l'absence d'un item peuvent être utilisées. Ainsi au lieu de s'intéresser uniquement à la présence des items ou motifs X dans la règle, il faut également analyser l'absence de ces mêmes motifs, que l'on va noter $\overline{X}$. Un exemple de règle négative dans la grande distribution pourrait être que les clients qui achètent de la bière achètent généralement des chips mais pas de vin : $Bière \Rightarrow Chips$, $\overline{Vin}$ ou encore que les clients qui achètent de la bière mais pas de pizza, achètent généralement des chips : $Bière$, $\overline{Pizza} \Rightarrow Chips$.