

# Informatique Décisionnelle

Michel de Rougemont  
Université Paris II

## Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Les données et les algorithmes</b>	<b>5</b>
<b>3</b>	<b>Modèle de calcul sur les mots</b>	<b>6</b>
3.1	Les automates sur les mots . . . . .	6
3.2	Expressions régulières . . . . .	7
3.3	Tester un mot . . . . .	7
3.4	Text Mining . . . . .	8
3.4.1	Classifier pour la distance d'édition . . . . .	8
3.4.2	Apprentissage supervisé . . . . .	10
3.4.3	Apprentissage non-supervisé . . . . .	10
<b>4</b>	<b>Modèle XML</b>	<b>11</b>
4.1	Arbres étiquetés . . . . .	12
4.2	DTD: Data Type Definition . . . . .	13
4.2.1	Distance à une DTD . . . . .	14
4.3	XSL et XSLT . . . . .	14
4.4	Automates sur les arbres (Tree automata) . . . . .	15
4.4.1	Automata on unranked trees . . . . .	16
<b>5</b>	<b>Modèle relationnel</b>	<b>17</b>
5.1	Entité-Relations . . . . .	17
5.2	Schéma relationnel . . . . .	18
5.3	Algèbre relationnelle et Langage SQL . . . . .	19
5.3.1	Syntaxe SQL . . . . .	19
5.4	Conception de Schémas . . . . .	20
5.4.1	Dépendances fonctionnelles . . . . .	20
5.4.2	Décomposition en schémas relationnels . . . . .	20
5.5	Formes normales . . . . .	21
5.5.1	Boyce-Codd . . . . .	21
5.5.2	3ème forme normale . . . . .	21

<b>6</b>	<b>Modèle OLAP</b>	<b>22</b>
6.1	Intégration des informations . . . . .	22
6.2	Constitution d'un entrepôt de données . . . . .	23
6.3	Caractéristiques principales des entrepôts de données . . . . .	23
6.4	Utilisation d'un entrepôt de données . . . . .	24
6.5	Algèbre fonctionnelle . . . . .	24
6.6	Expressions de chemin et langage OLAP . . . . .	25
<b>7</b>	<b>Data-Mining</b>	<b>29</b>
7.1	Arbres de décision . . . . .	31
7.1.1	Sélection parmi les Attributs dans un arbre de décision . . . . .	32
7.1.2	Approximations et arbre de décision . . . . .	33
7.2	Régression linéaire et logistique . . . . .	34
7.2.1	Régression logistique . . . . .	35
7.3	Réseaux de neurones . . . . .	36
7.4	Apprentissage non supervisé: $k$ -Means . . . . .	37
7.5	Graph Mining . . . . .	37
7.6	Qualité de l'approximation . . . . .	39
7.7	Logiciels de Fouille de Données . . . . .	40
7.7.1	Dtree . . . . .	40
7.7.2	SAS/Entreprise Miner . . . . .	40
7.7.3	Weka . . . . .	40
<b>8</b>	<b>Mécanismes</b>	<b>41</b>
8.1	Jeux à N joueurs . . . . .	41
8.1.1	Jeux à deux joueurs à somme nulle . . . . .	43
8.1.2	Jeux à deux joueurs . . . . .	43
8.2	Mécanismes . . . . .	43
8.3	Le Mécanisme Adwords . . . . .	43

# 1 Introduction

Ce cours présuppose la maîtrise des modèles informatiques suivants:

- Le modèle des fichiers auxquels on accède en utilisant le réseau, à l'aide de protocoles comme FTP ou HTTP,
- Le modèle des tableurs (Excel): calculer des informations stockées dans des cellules d'un tableur, afficher des statistiques,
- Le modèle relationnel (Access): les informations sont stockées dans des tables et on calcule des requêtes dont le résultat est aussi stocké dans des tables,
- Le modèle algorithmique simple: calcul d'une fonction (par exemple le tri) à l'aide d'un algorithme (par exemple, Quicksort), représenté dans un langage informatique (VBA, C, Java,..).

Le but du cours est de développer la Science des Données (*Data Science*), qui comporte l'analyse des données, la prédiction, l'intégration et la valorisation.

- l'analyse des données, aussi appelée analyse OLAP, consiste à étudier la répartition de certaines mesures de ces données selon certaines dimensions, spécifiées par un schéma OLAP,
- la prédiction, aussi appelée apprentissage (*machine learning*), consiste à trouver à partir de données d'apprentissage  $(x_1, f(x_1)), (x_2, f(x_2)), \dots (x_k, f(x_k))$ , une fonction  $g$  proche d'une fonction inconnue  $f$ . Pour prédire  $f(x)$  pour un  $x$  arbitraire, on utilise  $g(x)$ ,
- l'intégration des données est la possibilité de combiner plusieurs sources de données pour en faire l'analyse et la prédiction globale,
- la valorisation est l'étape qui permet de donner une valeur économique aux opérations précédentes.

Les techniques que nous présentons ont une robustesse par rapport à l'incertitude des données. Les données réelles sont en effet souvent *bruitées*, c'est-à-dire inexactes et approximatives, pour de nombreuses raisons:

- elles sont produites par des agents imparfaits,
- elles sont traitées par des processus physiques,
- elles reflètent des mesures qui sont toujours approximatives,
- elles peuvent être manipulées.

Il est donc fondamental d'isoler des procédures informatiques robustes au bruit, c'est-à-dire qui produisent des résultats  $\varepsilon$ -approchés lorsque les données sont  $\varepsilon$ -bruitées. Différentes distances entre objets et différentes notions d'approximation sont introduites et associées à la notion de robustesse.

Les principaux modèles informatiques étudiés dans ce cours sont:

- Le modèle d'automates sur les mots
- Le modèle d'arbres ordonnés, aussi appelé XML

- Le modèle relationnel: MySQL
- Le modèle OLAP: Mondrian
- Le Data-Mining: Dtree et SAS/Entreprise Miner
- Les mécanismes pour la valorisation des données

Les automates sur les mots se généralisent aux arbres et sont le fondement du langage XML. Le modèle relationnel se généralise au modèle OLAP et constitue le coeur des systèmes d'Information. Le Data-Mining s'applique à tous les modèles, mais le cours décrit son utilisation à partir du modèle OLAP, qui constitue les fondements du *Business Intelligence*.

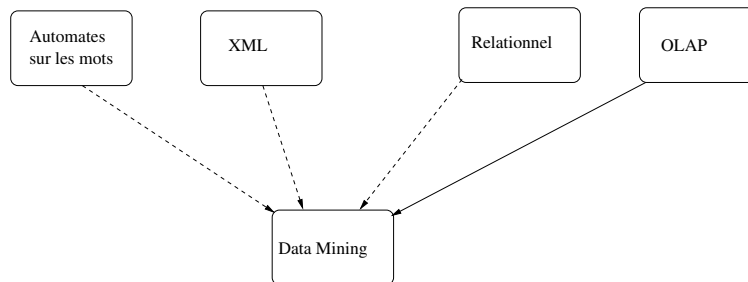


Figure 1: Les principaux liens entre les parties du cours.

Le cours est prolongé par des travaux dirigés qui illustrent les différentes techniques numériques introduites.

La section 2 décrit les modèles de données et la notion d'algorithme. La section 3 décrit les modèles d'automates, modèle simplifié de calcul sur les données de texte. La section 4 décrit le modèle XML d'arbres. La section 5 décrit le modèle relationnel, et la section 6 le modèle OLAP. La section 7 décrit le Data-Mining et la section 8 les Mécanismes.

## 2 Les données et les algorithmes

Les données envahissent la vie quotidienne: elles se trouvent sur des disques durs d'ordinateurs ou de serveurs ou tout simplement sur des tablettes et téléphones portables et prennent différentes formes. Cela peut être des fichiers de texte, des fichiers HTML ou XML, des tableaux, des images, de la vidéo.

- Dans le cas du texte et on s'intéresse alors à la recherche d'information comme le fait Google. Etant donné des mots clés, trouver parmi  $10^{11}$  fichiers (100 milliards), les fichiers qui contiennent ces mots ou en sont les plus proches.
- Il est important de considérer les fichiers HTML ou XML comme des arbres et on peut généraliser la recherche de sous-arbres dans des arbres.
- Dans le cas des tableaux aussi appelés *relations*, on recherche des sous-tableaux de manière similaire, à l'aide d'un langage informatique très important: SQL (Sequel Query Language).

Pour chacun de ces modèles, nous allons présenter des *modèles de calcul* définis à partir d'opérations élémentaires, puis des *algorithmes* qui composeront ces opérations élémentaires pour obtenir la solution aux problèmes de recherche.

La notion d'algorithme est centrale en informatique. Euclide (-300 avant J.C.) avait bien compris l'importance de procédures qui permettaient de trouver une solution à un problème, par exemple *l'algorithme d'Euclide* qui permet de trouver le plus petit commun multiple de deux nombres. Al-Khwarizm (environ 800 après J.C.) a permis de mieux cerner le problème des procédures en posant la question centrale suivante:

*Pour une fonction  $y = f(x)$  et plus généralement pour une fonction  $y = f(x_1, \dots, x_k)$  sur les entiers, existe-il toujours une méthode qui étant donné  $x$  ou plus généralement  $x_1, \dots, x_k$ , trouve  $y$ ?*

Cette question n'a eu de réponse que vers les années 1936, avec plusieurs réponses équivalentes (la réponse à la question précédente est NON), données par trois mathématiciens Church, Gödel et Turing. Le plus proche de l'Informatique est Alan Turing, considéré aujourd'hui comme un des pères de l'Informatique. Cette question a figé le mot *Algorithme* en hommage à Al-Khwarizm.

La notion de procédure suppose des opérations élémentaires que l'on va composer. Dans le cas des entiers, les opérations élémentaires peuvent être les opérations arithmétiques  $+$ ,  $*$ ,  $/$  mais aussi on peut aussi essayer des les réduire au minimum. Par exemple les seules opération  $+1$  et  $-1$  (successeur et prédécesseur) pourraient suffire.

L'informatique permet de traiter toute sorte de données: du texte, des arbres, des tables, des images, du son, .... Dans ce cas il faut tout d'abord définir les opérations élémentaires et ce n'est pas aussi simple que l'on pourrait penser.

C'est la raison pour laquelle chaque chapitre va décrire les opérations élémentaires retenues, puis les algorithmes dans ce cadre.

### 3 Modèle de calcul sur les mots

Un mot suppose un alphabet  $\Sigma$  décrivant des symboles et consiste en une succession de ces symboles. Ainsi le mot *abba* suppose que  $\Sigma = \{a, b\}$  et consiste en 4 symboles concaténés, c'est-à-dire écrits à la suite.

Supposons que nous lisons le mot de gauche à droite avec une seule opération possible: *lire la lettre suivante*. Quels algorithmes pouvons-nous décrire ainsi?

Le modèle de calcul le plus simple avec cette opération élémentaire est celui des *automates finis* que nous allons décrire. Ce modèle admet des variantes sur les arbres ainsi que sur d'autres classes de structures.

Ces modèles de machines sont des machines de Turing *sans mémoire* et jouent un rôle important en Logique et en Informatique.

#### 3.1 Les automates sur les mots

Soit  $\Sigma$  un alphabet fini et  $w$  un mot sur  $\Sigma$ . Un *Langage* est un ensemble de mots.

**Dnition 1.** Un *automate fini* est défini par  $(Q, q_0, F, \delta)$ , tel que:

- $Q$  est un ensemble fini d'états,
- $q_0 \in Q$  est l'état initial,
- $F \subseteq Q$  est un ensemble d'états finaux,
- $\delta \subseteq Q \times \Sigma \times Q$ .

Un automate est *déterministe* si  $\delta$  est une fonction  $Q \times \Sigma \rightarrow Q$ . Dans ce cas, on écrit:  $\delta(q, s) = q'$ . Dans le cas où  $\delta$  n'est pas une fonction, l'automate est dit *non déterministe*.

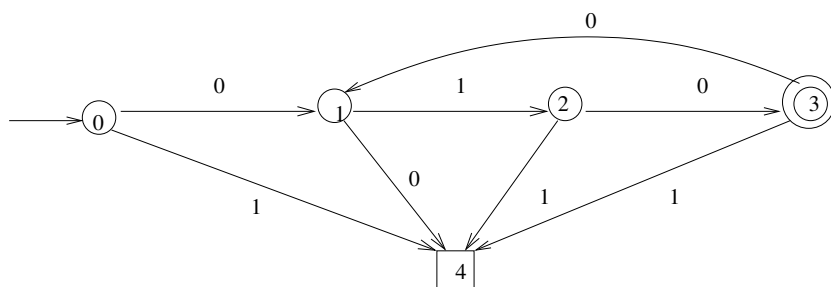


Figure 2: Un automate fini à 4 états et sa complétion à 5 états.

On représente souvent un automate par un graphe où les noeuds sont les états et les arêtes sont des transitions possibles lorsque l'automate lit un symbole  $s$ . Chaque arête est étiquetée par un tel symbole  $s$  tel que  $q.s.q'$  soit une transition. L'exemple précédent a 4 états  $\{0, 1, 2, 3\}$  et l'état 4 est l'état qui permet de compléter l'automate pour que la fonction  $\delta$  soit toujours définie. On obtient

alors la *complétion* de l'automate.

Un mot  $w = w_1, \dots, w_n$  est accepté par l'automate s'il existe une séquence d'états  $q_1, \dots, q_n \in Q$  tels que:

- $\delta(q_0, w_1) = q_1$ ,
- $\delta(q_i, w_{i+1}) = q_{i+1}$ , pour  $i = 1 \dots n - 1$ ,
- $q_n \in F$ .

Dans l'exemple ci-dessus, l'automate est déterministe. Si  $q_2$  est le seul état final, le langage accepté est :

$$L = \{1\} \cup \{01^n 0 \quad : \quad n = 0, 1, 2, \dots\}$$

Soit  $\Lambda$  le mot vide.

**Dnition 2.** Si  $w$  est un mot, alors  $w^*$  est l'ensemble des mots  $\{\Lambda, w, w^2, w^3, \dots, w^n \dots\}$ . Si  $L$  est un langage (ensemble de mots),  $L^*$  est l'ensemble des mots  $\{\Lambda, w_1, w_2, w_3, \dots, w_n \dots\}$  tels que  $w_i \in L^i$ .

### 3.2 Expressions régulières

Une expression régulière est un ensemble de mots défini par:

- un mot fini est une expression régulière.
- l'union, la concaténation et l'opérateur  $*$  appliqués à des expressions régulières, définissent une nouvelle expression régulière.

Les expressions:  $0^*11^*$ , ou  $(0^*11^* + (O(11)^*))^*$  sont des expressions régulières.

Les langages acceptés par des automates finis sont dits *réguliers*, car ils correspondent précisément à des expressions régulières.

Etant donné une expression régulière, on peut construire un automate équivalent et réciproquement étant donné un automate on peut construire une expression régulière équivalente.

Deux constructions importantes existent pour les automates:

- la déterminisation, qui consiste à transformer un automate non déterministe en automate déterministe équivalent.
- la minimisation, qui consiste à transformer un automate déterministe en automate déterministe minimum, avec un nombre minimum d'états..

### 3.3 Tester un mot

Soit  $w$  un mot de longueur  $n$  et  $r$  une expression régulière. L'automate lit le mot et décide si  $w \in r$  ou  $w \notin r$  en  $O(n)$  étapes.

Peut-on décider approximativement si  $w \in r$  en  $O(\log n)$  étapes ou même  $O(1)$  étapes, soit un nombre d'étapes constant, indépendant de  $n$ ?

Pour la distance d'édition et même pour la distance de Hamming, la réponse est positive. Il existe un algorithme  $A$  très simple tel que:

- Si  $x \in r$ , alors  $A(x) = 1$ ,
- Si  $x$  est  $\varepsilon$ -loin de  $r$ , alors  $Prob[A(x) = 0] > 2/3$ ,

Exemple: soit  $r = (ab)^* b^*$  et un mot  $w$  sur l'alphabet  $\Sigma = \{a, b\}$  et  $C$ . Soit  $\varepsilon = 10\%$ . Un échantillon est un sous-mot ou facteur de longueur 10, d'une position aléatoire  $1 \leq i \leq n$ . Le Testeur procède ainsi:

On repète 100 fois la procédure:

- tirer deux facteurs:  $u_1$  qui précède  $u_2$ .
- décider s'ils sont compatibles avec  $(ab)^* b^*$ . Rejeter s'ils ne le sont pas.
- accepter si on n'a pas rejeté au cours des tests.

Un facteur  $u$  est compatible avec  $(ab)^*$  s'il est du type  $abab\dots ab$  ou  $baba\dots ba$ . Il est compatible avec  $b^*$  s'il c'est  $bbb\dots b$ . On dit que  $(u_1, u_2)$  est compatible avec  $(ab)^* b^*$  si soit  $u_1$  et  $u_2$  sont compatibles avec  $(ab)^*$  ou avec  $b^*$ , ou  $u_1$  est compatible avec  $(ab)^*$  et  $u_2$  est compatible avec  $b^*$ , ou  $u_1$  a un préfixe compatible avec  $(ab)^*$  et un suffixe compatible avec  $b^*$  et  $u_2$  est compatible avec  $b^*$ .

### 3.4 Text Mining

Le but de la fouille de texte (Text Mining) est de pouvoir classifier des mots à partir d'exemples. On parle alors d'apprentissage et on distingue:

- l'apprentissage supervisé où les exemples sont étiquetés par des classes,
- l'apprentissage non supervisé où les exemples n'ont pas d'étiquetés et il faut donc trouver des classes.

#### 3.4.1 Classer pour la distance d'édition

Soit  $w$  un mot et deux expressions régulières représentant des classes différentes. Par exemple  $r_1 = (ab)^* b^*$  et  $r_2 = (aba)^*$ . On demande alors de quelle expression  $w$  est-il le plus proche. Si on prend la distance d'édition avec déplacement, on peut obtenir un testeur encore plus simple que le testeur précédent.

Pour un mot donné  $w$ ,  $ustat_k(w)$  est la densité des sous-mots de longueur  $k$ . Par exemple si  $k = 2$  et  $\Sigma = \{a, b\}$ , on a quatre sous-mots possibles:  $aa, ab, ba, bb$ . Il suffit alors de compter le nombre de chaque sous-mot et de normaliser. Si  $w = aaabb$  alors

$$ustat_2(w) = \frac{1}{4} \begin{bmatrix} 2 \\ 1 \\ 0 \\ 1 \end{bmatrix} \quad (1)$$



Il y a 4 sous-mots possibles, 2 fois  $aa$ , 1 fois  $ab$ , 0 fois  $ba$  et 1 fois  $bb$ .

Par extension  $\text{ustat}_k((ab)^*) = \lim_{n \rightarrow \infty} \text{ustat}_k((ab)^n)$ . On obtient alors:

$$\text{ustat}_2((ab)^*) = \begin{bmatrix} 0 \\ \frac{1}{2} \\ \frac{1}{2} \\ 0 \end{bmatrix} \quad (2)$$

On introduit alors  $s_1 = \text{ustat}_2((ab)^*)$ ,  $s_2 = \text{ustat}_2(a^*)$ ,  $s_3 = \text{ustat}_2((aba)^*)$ . On note  $Hull(s_1, s_2, s_3)$  la fermeture convexe des points  $s_1, s_2, s_3$ , c'est-à-dire tous les points  $s = a_1.s_1 + a_2.s_2 + a_3.s_3$ , où  $a_i \geq 0$  et  $\sum a_i = 1$ . Soit  $H_1 = Hull(s_1, s_2)$  et  $H_2 = Hull(s_3)$ . Ce sont les représentations statistiques de  $r_1$  et  $r_2$ , et cette méthode se généralise à toute expression régulière.

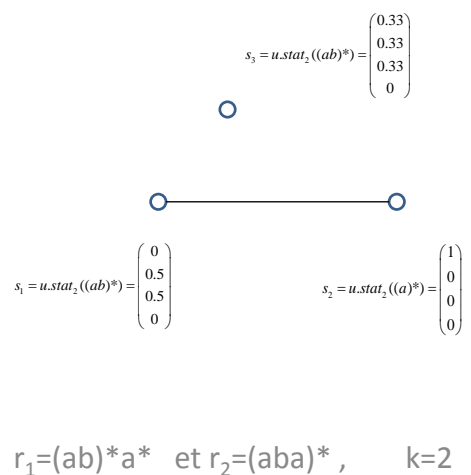


Figure 3: Représentation statistique:  $H_1 = Hull(s_1, s_2)$  et  $H_2 = Hull(s_3)$ .

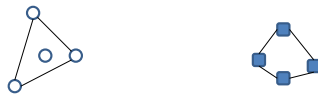
Le test d'appartenance pour décider  $\varepsilon$ -approximativement si  $w \in r$  est alors: construire  $H_r$  pour  $k = 1/\varepsilon$ . Estimer  $y = \text{ustat}_k(w)$  et calculer la distance géométrique entre  $y$  et  $H_r$ . Accepter si cette distance est inférieure à  $\varepsilon$ .

Pour classifier  $w$ , on choisit une séquence géométrique de  $\varepsilon = 0.5, 0.25, 0.125, \dots$  et on applique le test de  $w \in r_1$  et de  $w \in r_2$ . On finira par trouver un  $\varepsilon$  tel que les deux tests diffèrent et on saura alors si  $w$  doit être classifié par  $r_1$  ou par  $r_2$ . On peut aussi calculer directement la distance de  $y$  à  $H_1$  et à  $H_2$  et prendre la plus petite distance.

### 3.4.2 Apprentissage supervisé

Supposons que nous ayons des mots étiquetés par deux classes 0 et 1, qui servent d'exemples pour des classes inconnues que l'on modélise par des expressions régulières. On représente les mots par leurs vecteurs  $ustat_k$  et on peut alors construire les plus grandes fermetures convexes consistantes. On groupe ainsi les mots par des fermetures qui contiennent uniquement des mots de la même étiquette.

On approxime ainsi chaque classe par l'union de fermetures convexes. Chaque fermeture convexe



Statistiques des exemples étiquetés

Figure 4: Fermeture convexe des statistiques d'exemples

peut elle-même être approximée par une expression régulière. Etant donné un nouveau mot,  $w$  comment le classifier? On applique simplement le testeur précédent, ou on calcule la distance du point  $y = ustat_k(w)$  aux deux polytopes  $H_0$  et  $H_1$ , et on choisit la classe qui est la plus proche.

### 3.4.3 Apprentissage non-supervisé

Dans ce cas, nous avons des exemples de mots mais aucune classe ne les étiquette au départ. Il faut donc trouver les meilleures classes qui structurent ces mots.

Un exemple est *Google news*. Un programme parcourt toutes les nouvelles publications des journaux et les classe sans connaître au départ les différentes classes (Politique, Business, Sport, etc...).

A un instant donné, on peut utiliser les classifications précédentes ou rechercher les partitions de fermetures convexes, c'est-à-dire les fermetures convexes qui n'ont pas d'intersection. Chacune représentera une classe.

En pratique les deux approches sont complémentaires. On classe selon la classification en usage à un instant donné et on peut aussi décider de modifier la classification, donc de créer de nouvelles classes ou de supprimer des classes existantes.

## 4 Modèle XML

XML est un langage générique qui permet d'unifier la manipulation de données sur des serveurs différents. Les fonctions essentielles sont:

- la transmission de données semi-structurées entre Client et Serveur. C'est la fonction des Web services qui utilisent le protocole SOAP.
- l'interrogation de données semi-structurées (XPATH, XQUERY)
- la transformation de données (XSL, XSLT),
- l'intégration de données.

Ce langage est important dans l'usage du Web, car XML (eXtensible Markup Language) est une recommandation du W3C, l'organisme de normalisation de logiciels pour le Web.

Un fichier XML distingue clairement trois parties indépendantes:

- la structure du document, aussi appelée DTD (Document Type Definition).
- la feuille de style XSL (eXtensible Style Sheet) pour afficher le document,
- le document XML lui-même.

Considérons le fichier s2.xml ci-dessous.

```
<?xml version='1.0' ?>
<!DOCTYPE livre SYSTEM 'livre.dtd' [
  <!ENTITY ionesco SYSTEM 'ionesco.xml'>]>

<?xml:stylesheet
  type="text/xsl"
  href="s2.xsl"?>

<livre>
&ionesco;
</livre>
```

Ce fichier utilise la DTD livre.dtd ci dessous:

```
<?xml version='1.0' ?>

<!ELEMENT livre (chapitre*,titre,auteur)>
<!ELEMENT chapitre (titre,para*)>
<!ELEMENT titre (#PCDATA)>
<!ELEMENT para (#PCDATA)>
<!ELEMENT auteur (#PCDATA)>
```

qui organise la structure des balises.

La feuille de style s2.xsl ci-dessous:

```

<?xml version="1.0" ?>
- <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
- <xsl:template match="/">
  <xsl:apply-templates />
</xsl:template>
- <xsl:template match="chapitre/titre">
- <h1>
  <xsl:value-of select="." />
</h1>
</xsl:template>
</xsl:stylesheet>

```

décrit comment afficher le fichier ionesco.xml qui peut varier selon le type de terminal (écran ordinateur, WAP, téléphone mobile).

Pour un économiste, il permet l'analyse de données à grande échelle. Une norme XMLA généralise ainsi l'approche OLAP pour intégrer des fichiers d'origines différentes. Il permet aussi l'introduction de modèles de valeur de l'information, basée sur les distances entre documents et entre un document et un langage (ensemble de documents), aussi appelé DTD.

## 4.1 Arbres étiquetés

Un fichier XML est une séquence de balises imbriquées, définies par l'utilisateur. Chaque balise peut avoir des attributs.

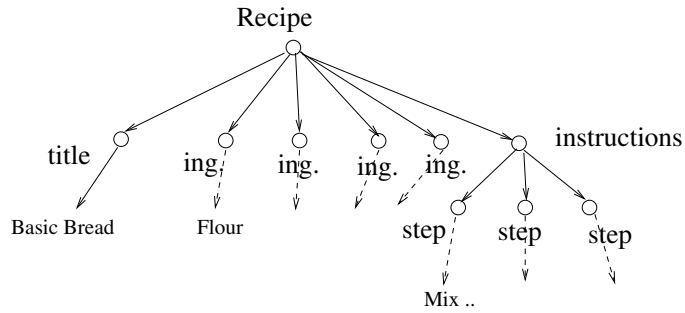
```

<?xml version="1.0" encoding="UTF-8"?>

<Recipe name="bread" prep_time="5 mins" cook_time="3 hours">
  <title>Basic bread</title>
  <ingredient amount="3" unit="cups">Flour</ingredient>
  <ingredient amount="0.25" unit="ounce">Yeast</ingredient>
  <ingredient amount="1.5" unit="cups">Warm Water</ingredient>
  <ingredient amount="1" unit="teaspoon">Salt</ingredient>
  <Instructions>
    <step>Mix all ingredients together, and knead thoroughly.</step>
    <step>Cover with a cloth, and leave for one hour in warm room.</step>
    <step>Knead again, place in a tin, and then bake in the oven.</step>
  </Instructions>
</Recipe>

```

Dans cet exemple la balise `< Recipe >` a les attributs `name`, `prep.time` et `cook.time`, qui ont les valeurs *bread*, *5 mins*, *3 hours*. Le texte entre deux balises conjuguées (ouvrante et fermante) est appelé *PCDATA*. On peut représenter le fichier par un arbre étiqueté par les balises (tags), les attributs et les *PCDATA*.



Un arbre XML.

Cette représentation est le *DOM* (Document Object Model), associé à un fichier XML *F.xml*. La sérialisation d'un arbre étiqueté est l'opération inverse qui associe un fichier XML à un arbre.

## 4.2 DTD: Data Type Definition

La DTD est l'élément structurel qui définit une classe d'arbres organisés selon une structure. La DTD ci-dessous décrit comment est organisé un livre et prend la forme d'une séquence de règles de deux types:

- $t : r$  où  $r$  est une expression régulière sur les tags,
- $t : PCDATA$  indiquant que  $t$  peut se terminer sur une feuille de l'arbre.

Un fichier est *bien formé* si la séquence de tags ouvrant/fermant est bien imbriquée, c'est-à-dire qu'un tag se ferme lorsque tous les tags internes sont bien imbriqués. La séquence suivante  $\langle a \rangle \langle b \rangle \langle /a \rangle \langle /b \rangle$  n'est pas bien formée car la séquence interne au tag  $\langle a \rangle$  n'est pas bien formée.

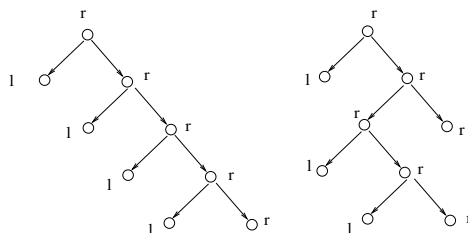
Un fichier est *valide* lorsque l'arbre associé suit la DTD, c'est-à-dire que chaque noeud de tag  $t$  suit les deux conditions suivantes:

- si le noeud a des fils dont la séquence de tags est  $l_1, l_2, \dots, l_k$ , il existe une règle  $t : r$  telle que  $l_1, l_2, \dots, l_k \in r$ ,
- si le noeud est une feuille, il existe une règle  $t : PCDATA$ .

Un automate d'arbre  $\mathcal{A}$  est directement associé à une DTD. Il procède bottom-up est décide si un fichier  $F$  est accepté. Un fichier  $F$  est valide ssi il est accepté par l'automate.

```
<?xml version='1.0' ?>
```

```
<!ELEMENT r (1,r) >
<!ELEMENT r (#PCDATA)>
<!ELEMENT l (#PCDATA)>
```



### 4.2.1 Distance à une DTD

La distance d'Édition avec déplacements comprend les opérations élémentaires suivantes:

- la modification de tags et attributs,
- l'insertion et la suppression de noeuds et d'arêtes,
- le déplacements de sous-arbres entiers.

La distance entre deux documents XML  $\text{dist}(F, F')$  est le nombre minimum d'opérations élémentaires qu'il faut appliquer pour faire correspondre  $F$  et  $F'$ . La distance d'un fichier  $F$  à une DTD  $\text{dist}(F, DTD)$  est la distance minimum  $\text{dist}(F, F')$  pour  $F'$  valide, c'est-à-dire  $F' \in DTD$ .

### 4.3 XSL et XSLT

XSL (eXtended Style Sheet) est une feuille de style, adaptée à XML. Son extension XSLT (Transformation) permet de transformer un fichier XML dans un format cible qui peut être HTML ou XML.

L'exemple suivant donne un fichier XML source, puis un programme XSLT, qui transforme le fichier source en fichier cible.

Fichier Source:

```
<persons>
  <person username="MP123456">
    <name>John</name>
    <family_name>Smith</family_name>
  </person>
  <person username="PK123456">
    <name>Morka</name>
    <family_name>Ismincius</family_name>
  </person>
</persons>
```

Programme XSLT de transformation:

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:output method="xml" indent="yes"/>
<xsl:template match="/">
  <transform>
    <xsl:apply-templates/>
  </transform>
</xsl:template>
<xsl:template match="person">
  <record>
    <username>
```

```

        <xsl:value-of select="@username" />
    </username>
    <name>
        <xsl:value-of select="name" />
    </name>
</record>
</xsl:template>
</xsl:stylesheet>

```

Fichier Cible résultant:

```

<?xml version="1.0" encoding="UTF-8"?>
<transform>
  <record>
    <username>MP123456</username>
    <name>John</name>
  </record>
  <record>
    <username>PK123456</username>
    <name>Morka</name>
  </record>
</transform>

```

#### 4.4 Automates sur les arbres (Tree automata)

(facultatif)

A ranked ordered tree with  $n$  nodes is a structure  $T_m = (D_n, \{Child_i\}_{i \leq m}, root)$  where the domain  $D_n = \{1, \dots, n\}$  is the set of nodes,  $Child_i$  is a binary relation such that  $Child_i(u, v)$  if  $u$  is the  $i$ th child of  $v$  for  $i \leq m$  and a fixed  $m$  and  $root$  is a distinguished element of  $D_n$  with no predecessors. For each  $i \neq i' \leq m$  such that  $Child_i(u, v)$  and  $Child_{i'}(u, v')$  then  $v \neq v'$  and for each  $i$  there is at most one  $v$  such that  $Child_i(u, v)$ . In addition, the graph whose edges are in  $Child_i$  for  $i \leq m$  backwards or forwards does not contain any circuits. A *Binary Tree* is a ranked ordered tree with  $m = 2$  where each node which is not a leaf has a left and a right child.

**Dnition 3.** A tree-automaton on binary trees, is a 4-tuple  $A = (Q, \delta, q_0, F)$  where  $Q$  is the finite set of states,  $q_0$  the initial state,  $F$  the set of accepting states and  $\delta$  a finite set of transition  $(q_i, q_j) \rightarrow q_l$ .

The transition function allows one to assign the state  $q_l$  to a node  $v$  whose left child is in state  $q_i$  and right child in state  $q_j$ . A run starts on the leaves of the tree in state  $q_0$ , proceeds bottom-up assigning states to the nodes of the tree and accepts if the root is in an accepting state. The language defined by a tree automaton is the set of accepted trees.

**Exemple.** The automaton  $A = (\{q_0, q_1\}, \delta, \{q_1\})$  with initial state  $q_0$ , accepting state  $q_1$  and whose transitions  $\delta$  are :  $(q_0, q_0) \rightarrow q_1$  and  $(q_0, q_1) \rightarrow q_1$  accepts the language  $L_r$  of right-branch trees. In the completed automaton there is always a sink state  $q_s$  such that  $(q_1, q_0) \rightarrow q_s$  and  $(q_1, q_1) \rightarrow q_s$ , although we do not represent it. We draw the transitions with dashed lines for the left branch and solid lines for the right branch.



Figure 5: A tree-automaton for  $L_r$  the language of right-branch trees and a tree  $t \in L_r$ .

Tree automata can be non-deterministic and define the class of *regular trees*. They have a deterministic equivalent automaton. In the case of *top-down* automata, the non-deterministic and deterministic versions are different.

#### 4.4.1 Automata on unranked trees

An unranked tree is a tree where each node has an arbitrary number of successors, which are ordered.

Every unranked tree  $T$  can be coded as a binary tree  $e(T)$  but many encodings are possible. Consider the classical encoding where each node  $v$  of the unranked tree is a node  $v$  in the binary encoding, the left successor of  $v$  in the binary tree is its first successor in the unranked tree, the right successor of  $v$  in the binary tree is its first sibling in the unranked tree. New nodes with labels  $\perp$  are added to complete the binary tree when there are no successor or no sibling in the unranked tree.

An *unranked tree automaton* generalizes the transition function to  $\delta : Q \times \Sigma \rightarrow 2^{Q^*}$  such that  $\delta(q, a)$  is a regular language on  $Q$ .

A run  $\lambda$  is generalized such that if  $u$  is a node with successors  $v_1, \dots, v_l$  in states  $\lambda(v_1), \dots, \lambda(v_l)$  and there is a  $q$  such that  $\lambda(v_1), \dots, \lambda(v_l) \in \delta(q, l(u))$ , then  $\lambda(u) = q$ .

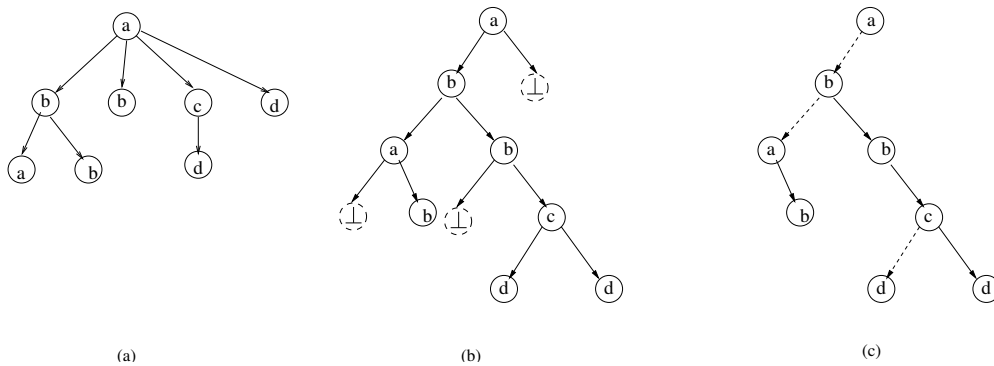


Figure 6: An unranked tree, coded as a binary tree and as an extended binary tree.

A DTD is a special case of a tree automaton when  $Q = \Sigma$ . It is defined as a set of rules  $a : r$ , where  $a \in \Sigma$  and  $r$  is a regular expression.



```

<?xml version='1.0' ?>

<!ELEMENT livre (chapitre*,titre,auteur)>
<!ELEMENT chapitre (titre,para*)>
<!ELEMENT titre (#PCDATA)>
<!ELEMENT para (#PCDATA)>
<!ELEMENT auteur (#PCDATA)>

```

In this case  $\Sigma = \{livre, titre, chapitre, auteur, para\}$ . The first rule is:

$$livre : chapitre^* titre auteur$$

It specifies that a node labelled *livre* is followed by many nodes labelled *chapitre*, followed by a node labelled *titre* and a node labelled *auteur*.

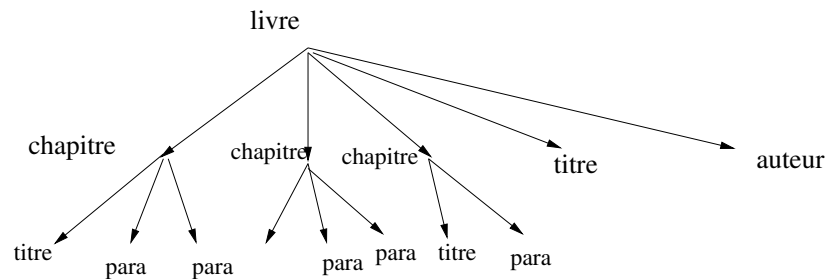


Figure 7: A valid tree for the livre DTD.

## 5 Modèle relationnel

Un des modèles de données les plus importants est le *modèle relationnel* où les données sont représentées par des tables. Une table est constituée de plusieurs colonnes appelées *attributs* et de lignes appelées *enregistrements* ou *tuples*.

Une *requête* est une fonction qui associe à un ensemble de tables une nouvelle table. C'est la représentation mathématique d'une question posée à une base de donnée.

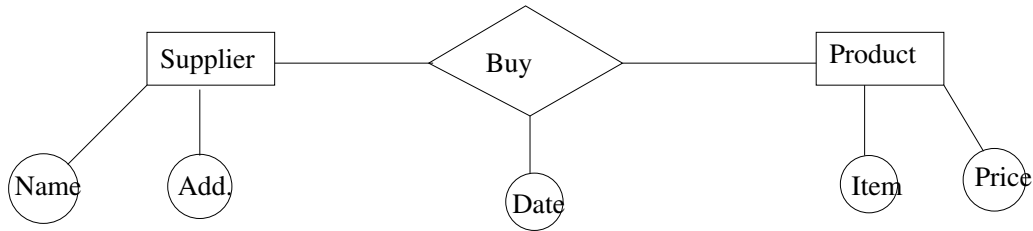
La définition des différentes tables constitue le *Schéma*. Pour concevoir un schéma, il est utile d'isoler des entités et des relations entre entités. Une entité décrit un objet muni d'attributs.

### 5.1 Entité-Relations

La conception d'un schéma est facilitée par un *diagramme Entité-Relations*, qui décrit de manière graphique des entités munies d'attributs et des relations entre entités. On distingue:

- les entités dans des rectangles,
- les attributs des entités dans des cercles,

- les relations dans des losanges.



Un diagramme Entité Relations.

## 5.2 Schéma relationnel

Soit  $\mathbf{U} = (D, R_1, \dots, R_k, c_1, \dots, c_m)$  une structure relationnelle, où:

- $D$  est l'union de différents domaines  $\bigcup_i D_i$ . Chaque  $D_i$  peut être un domaine à  $n$  éléments et on écrit  $D_i = \{a_1, \dots, a_n\}$ , ou un ensemble comme les entiers  $N$  ou les réels  $R$ .
- chaque  $R_i$  pour  $i = 1, \dots, k$  est une relation d'arité  $r_i$  sur  $D$ , i.e.  $R_i \subseteq D^{r_i}$ . Dans le cas où  $r_i = 1$ , on parle de relation unaire  $R_i \subseteq D$ , et où  $r_i = 2$ , on parle de relation binaire  $R_i \subseteq D^2$ ,
- chaque  $c_j$  pour  $j = 1, \dots, m$  est un élément distingué.

Chaque relation peut être considérée comme une table avec un nombre de colonnes égal à l'arité de la relation. Un *attribut* est le nom d'une colonne dont les valeurs sont dans un domaine  $D_i$ . La *signature* de la structure est la séquence  $r_1, \dots, r_k, m$  ainsi que l'information spécifiant le domaine de chaque attribut définit le schéma.

Soit  $\mathbf{K}$  l'ensemble de toutes les structures  $\mathbf{U}$ .

Exemple d'un mauvais schéma :  $Suppliers(Sname, Saddress, Item, Price)$ .

Sname	Saddress	Item	Price
IBM	New-York	PC	1000
IBM	New-York	Mainframe	1 000 000
IBM	New-York	Service	100 000
Microsoft	Seattle	Windows	100
Microsoft	Seattle	Office	100

Supposons que certains tuples soient modifiés, effacés ou insérés. On observera alors des problèmes de:

- Redondance
- Inconsistance
- Anomalies insertion/Effacement

Des exemples de requêtes sont:

- Quelle est l'adresse d'IBM ?

- Est-ce que Microsoft est un fournisseur?
- Quel est le produit le plus vendu?
- Quel sont les produits et leur fournisseur des produits les plus vendus en 2000 ?

Exemple d'un bon schéma :  $Sa(Sname, Saddress)$ ,  $Sip(Sname, Item, Price)$ . Chaque relation décrit les entités. Par contre, certaines requêtes peuvent nécessiter plus de calculs.

### 5.3 Algèbre relationnelle et Langage SQL

Considérons les opérateurs suivants, c'est-à-dire des fonctions qui prennent des relations comme arguments et définissent une nouvelle relation.

- Union:  $S \cup R$ , qui définit l'ensemble des tuples de  $R$  et de  $S$ ,
- Différence:  $S - R$ , qui définit l'ensemble des tuples de  $S$  qui ne sont pas dans  $R$ ,
- Produit cartésien:  $S \times R$  qui définit l'ensemble des paires  $(s_i, t_j)$  où  $s_i \in S$  et  $t_j \in R$ ,
- Projection:  $\pi_{A_1, \dots, A_k}(R)$  qui définit la projection des tuples de  $R$  sur les attributs  $A_1, \dots, A_k$ ,
- Sélection:  $\sigma_{\Theta(A_1, \dots, A_k)}(R)$  qui définit la sélection des tuples de  $R$  sur la condition  $\Theta(A_1, \dots, A_k)$ , composition booléenne de formules atomiques sur les attributs  $A_1, \dots, A_k$ .

Un autre opérateur est la  $\Theta$ -jointure,  $R_{\Theta}S$  et la jointure simple  $RS$ . Dans le cas de la jointure simple, on prend le produit cartésien, puis l'égalité des attributs communs et enfin la sélection pour éliminer les colonnes identiques.

**Dnition 4.** *L'algèbre relationnelle est l'ensemble des expressions obtenues par composition des opérateurs d'union, de différence, de produit, de projection et de sélection.*

Une formule logique sur un schéma est composée de:

- de formules atomiques  $R(x_1, \dots, x_k)$ ,  $x_i = x_j$ ,  $x_i = a$ , et  $x_i \leq a$ ,
- de formules  $\neg\psi(x)$ ,  $\psi_1(x) \vee \psi_2(x)$ ,  $\psi_1(x) \wedge \psi_2(x)$ ,
- de formule  $\exists y\psi(x, y)$ , et  $\forall y\psi(x, y)$ ,

**Dnition 5.** *Une formule du 1er ordre sur un schéma est la composition de formules atomiques à l'aide des opérateurs booléens et des quantificateurs.*

**Th 1.** *Une requête est définissable par une expression de l'algèbre universelle ssi elle est définissable par une formule du 1er ordre.*

#### 5.3.1 Syntaxe SQL

Le langage SQL permet trois types d'opérations:

- Manipulation de tables: création, effacement, modification,
- Transactions: insertion, effacement, modifications de tuples,
- Requêtes: création de nouvelles tables ou de vues.

**Syntaxe du SELECT en SQL** L'expression standard est:

*Select A,B from R where conditions*

Exemples:

- Select Saddress from Suppliers where Sname='IBM' ;
- Select Sname from Suppliers where Sname='Microsoft' ;
- Select Item, Sname from Suppliers where date='2000' ; en supposant un nouvel attribut date sur le schéma précédent.

### Correspondances entre formule logique, expression SQL et algèbre relationnelle

- $\exists x, z, t \text{ Suppliers}(x, y, z, t) \wedge x = 'IBM'$ ; Select Saddress from Suppliers where Sname='IBM' ;  $\pi_{Saddress}(\sigma_{Sname='IBM'})$ .
- $\exists y, z, t \text{ Suppliers}(x, y, z, t) \wedge x = 'Microsoft'$ ; Select Sname from Suppliers where Sname='Microsoft' ;  $\pi_{Sname}(\sigma_{Sname='IBM'})$ .
- $\exists x, y, t, d \text{ Suppliers}(x, y, z, t, d) \wedge d = '2000'$ ; Select Item, Sname from Suppliers where date='2000' ;  $\pi_{Item, Sname}(\sigma_{date='2000'})$ .

## 5.4 Conception de Schémas

### 5.4.1 Dépendances fonctionnelles

Une dépendance fonctionnelle entre un ensemble d'attributs  $\{A_1, \dots, A_n\}$  et un attribut  $B$  indique que si les valeurs des attributs  $A_1, A_2, \dots, A_n$  sont fixées, alors la valeur de  $B$  est fixée. On écrit:

$$A_1 A_2, \dots, A_n \rightarrow B$$

Certaines dépendances fonctionnelles impliquent d'autres dépendances: par exemple  $A \rightarrow B$  et  $B \rightarrow C$  impliquent  $A \rightarrow C$ . Si  $F$  est un ensemble de dépendances,  $F^+$  est l'ensemble des conséquences logiques.

**Définition 6.** Soit  $R(A_1, A_2, \dots, A_n)$  une relation et  $F$  un ensemble de dépendances fonctionnelles. Un sous ensemble  $X$  de  $A_1, A_2, \dots, A_n$  est une clé si c'est le plus petit ensemble tel que  $X \rightarrow A_1, A_2, \dots, A_n$  est dans  $F^+$ .

### 5.4.2 Décomposition en schémas relationnels

Il existe de nombreuses décompositions possibles et il est important de comprendre les avantages et inconvénients de chaque décomposition.

**Décomposition sans perte d'information.** Soit  $r$  est une instance de

*Suppliers(Sname, Saddress, Item, Price)*

$rSA(r)$  et  $rSIP(r)$  les instances projection de  $r$ . Alors  $r = join(rSA(r), rSIP)$ . Si deux relations  $rSA$  et  $rSIP$  sont données, alors si  $s = (rSA(r), rSIP)$ , la décomposition est sans perte si  $s = r$ . Sinon on ne peut pas retrouver  $r$  de manière unique.

**Préservation des dépendances fonctionnelles.** Si  $F$  est l'ensemble des dépendances fonctionnelles, alors  $pSA(F)$  est l'ensemble des dépendances concernant les attributs S et A. La décomposition préserve les dépendances fonctionnelles si l'union des  $pSA(F)$  pour toutes les projections implique logiquement  $F$ .

Exemple : City (C), Street (S), Zip (Z). Dépendances sont:  $CS \rightarrow Z$  et  $Z \rightarrow C$ . Les clés sont  $\{C, S\}$  et  $\{Z, S\}$ . Décomposition en SZ et CZ est sans perte mais ne préserve pas la dépendance  $CS \rightarrow Z$ .

S	Z
a	b
a	b'

C	Z
c	b
c	b'

La jointure ne respecte pas la dépendance fonctionnelle  $CS \rightarrow Z$ .

C	S	Z
c	a	b
c	a	b'

## 5.5 Formes normales

Les formes normales caractérisent les bons schémas relationnels, du point de vue des dépendances fonctionnelles.

### 5.5.1 Boyce-Codd

Si  $X \rightarrow A$  est une dépendance fonctionnelle de  $R$ ,  $A$  n'étant pas dans  $X$ , alors  $X$  contient une clé. Le schéma  $(CSZ, CS \rightarrow Z \text{ et } Z \rightarrow C)$  n'est pas en forme normale de Boyce-Codd. En effet  $Z$  ne contient pas de clé !

### 5.5.2 3ème forme normale

Un attribut est premier s'il appartient à une clé. Dans l'exemple, tous les attributs sont premiers. Un schéma est en 3ème forme normale si pour toute dépendance fonctionnelle  $X \rightarrow A$ ,  $A$  n'étant pas dans  $X$ , alors  $X$  contient une clé ou  $A$  est premier. Si le schéma ne satisfait pas cette forme normale, alors soit :

- $X$  est un sous-ensemble strict d'une clé (dépendance partielle) ou
- $X$  un sous-ensemble strict d'aucune clé (dépendance transitive).

## 6 Modèle OLAP

Le modèle OLAP permet de répondre à des questions de gestionnaires et d'économistes, pour analyser des données. Par exemple:

- Quels sont les produits les plus vendus en 2005?
- Quels sont les magasins et les produits les plus vendus entre 2004 et 2005 en Europe?

Ces requêtes généralisent le GROUP BY de SQL mais en l'intégrant à plusieurs attributs. Les requêtes OLAP étendent SQL dans ce sens. Le type de question posée sous-entend des bases de données beaucoup plus grandes qui rassemblent plusieurs bases de données. Se pose alors l'intégration de ces données, au sein d'une même structure.

L'intégration de données est un problème plus vaste qui concerne les économistes. Comment en effet passer de l'analyse de données réalisée à partir d'une source comme l'INSEE ou l'Eurostat à une analyse concernant toutes les sources du Web sur un même sujet?

### 6.1 Intégration des informations

Le besoin : collecter des informations à partir des plusieurs sources dans le but d'exploiter leur synthèse.

Le problème : les sources peuvent être autonomes et hétérogènes autorisations nécessaires pour l'extraction utilisation de plusieurs langages pour la définition de l'information à extraire.

La solution : utiliser un schéma fédérateur, avec ou sans données, que les utilisateurs manipulent comme s'il s'agissait d'une base de données habituelle.

- avec données courantes provenant d'une seule source (vues matérialisées) ou de plusieurs sources autonomes et éventuellement hétérogènes (base de données intégrée)
- besoin d'outils pour la conception du schéma, transformation/chargement de données et propagation de changements
- avec données historiques (entrepôt de données, données en avance), c'est notre cas
- besoin d'outils pour la conception du schéma, transformation/chargement de données, propagation de changements et rafraîchissement périodique
- sans données, au sein d'une seule source (vues virtuelles) ou lié à plusieurs sources autonomes et éventuellement hétérogènes (médiateurs, données à la demande)
- besoin d'outils pour la réécriture/évaluation des requêtes et la fusion des résultats dans tous les cas, l'accès aux données se fait presque exclusivement en lecture

## 6.2 Constitution d'un entrepôt de données

Architecture à trois niveaux : sources - entrepôt - data mart. Un data mart est un 'petit' entrepôt orienté sujet, dont les données sont dérivées de l'entrepôt.

Extraction, Transformation, Chargement de données sont réalisés à l'aide d'outils ETL.

Propriétés d'un Extracteur :

- traduction vers le langage source, évaluation, traduction vers le langage de l'entrepôt
- détection de changements aux sources

Propriétés d'un Intégrateur :

- réconciliation / correction d'erreurs / filtrage / estampillage pour conformer au schéma de l'entrepôt,
- chargement de données, rafraîchissement,

## 6.3 Caractéristiques principales des entrepôts de données

Un entrepôt de données est l'endroit où les données intégrées sont stockées et exploitées à l'aide d'un système de gestion de bases de données. Par conséquent, un entrepôt de données est avant tout une base de données, même si les caractéristiques suivantes le distinguent clairement des bases de données transactionnelles habituelles.

Utilisation : les utilisateurs principaux sont les décideurs de l'entreprise qui ont besoin des schémas faciles à lire (pas de schémas normalisés) (schéma dimensionnel).

Mode d'accès : souvent à travers un data mart, en lecture uniquement. Certains index qui ne sont pas efficaces pour le transactionnel le deviennent pour les entrepôts.

Volume : de l'ordre de téra octets d'où un besoin d'algorithmes efficaces pour le chargement de données et l'évaluation des requêtes.

Maintenance : les mises à jour sont propagées des sources vers l'entrepôt:

- immédiatement ou périodiquement (suivant la nature de l'application)
- par reconstruction ou de manière incrémentale.

Rafraîchissement : les données qui deviennent 'obsolètes' sont supprimées.

Metadonnées : Les métadonnées d'un entrepôt sont plus complexes et plus volumineuses que celles d'une base de données transactionnelle, et sont souvent gérées en dehors de l'entrepôt.

## 6.4 Utilisation d'un entrepôt de données

OLAP : Online Analytic Processing permet de définir des requêtes et d'effectuer résumés/agrégations selon plusieurs critères et sur plusieurs dimensions, et de générer des rapports.

Extraction de connaissances à partir de données (Fouille de données, 'Data Mining') est une autre application.

## 6.5 Algèbre fonctionnelle

Rappelons d'abord qu'une fonction est désignée par  $f : X \rightarrow Y$ , où  $f$  est le nom de la fonction,  $X$  son ensemble de départ et  $Y$  son ensemble d'arrivée. Nous noterons  $\text{def}(f)$  l'ensemble des éléments de  $X$  pour lesquels  $f$  est définie et  $\text{range}(f)$  l'ensemble des valeurs prises par  $f$ . Une fonction  $f$  est appelée :

- fonction totale (ou application) si  $\text{def}(f) = X$  (sinon,  $f$  est appelée fonction partielle)
- fonction injective (ou injection) si  $x \neq x'$  implique  $f(x) \neq f(x')$ , pour tous  $x, x'$  dans  $\text{def}(f)$
- fonction surjective (ou surjection) si  $\text{range}(f) = Y$
- fonction bijective si elle est à la fois injective et surjective.

Dans la suite, nous nous intéresserons uniquement aux fonctions totales, et l'algèbre fonctionnelle dont nous aurons besoin comportera quatre opérations :

- Composition : prend en argument deux fonctions,  $f$  et  $g$ , telles que  $\text{range}(f) \subseteq \text{def}(g)$ , et renvoie une fonction  $g \circ f : \text{def}(f) \rightarrow \text{range}(g)$  définie par  $(g \circ f)(x) = g(f(x))$ ,  $x \in \text{def}(f)$
- Couplage : prend en argument deux fonctions,  $f$  et  $g$ , telles que  $\text{def}(f) = \text{def}(g)$ , et renvoie une fonction  $f \wedge g : \text{def}(f) \rightarrow \text{range}(f) \times \text{range}(g)$  définie par  $(f \wedge g)(x) = \langle f(x), g(x) \rangle$ ,  $x \in \text{def}(f)$
- Projection : l'opération habituelle sur le produit de deux ou plusieurs ensembles
- Restriction : prend en argument une fonction  $f : X \rightarrow Y$  et un ensemble  $E \subseteq \text{def}(f)$ , et renvoie une fonction  $f/E : E \rightarrow Y$  définie par  $(f/E)(x) = f(x)$  pour tout  $x$  dans  $E$

Propriétés des inverses. Rappelons d'abord que l'inverse (ou réciproque) d'une fonction  $f : X \rightarrow Y$ , noté  $f^{-1}$ , est définie comme suit :  $f^{-1}(y) = \{x / f(x) = y\}$  ; la fonction inverse associe chaque élément  $y$  de  $\text{range}(f)$  à l'ensemble des éléments  $x$  de  $\text{def}(f)$  ayant  $y$  comme image (et donc chaque élément  $y$  n'appartenant pas à  $\text{range}(f)$  à l'ensemble vide).

Voici quelques propriétés des inverses :

composition :  $(g \circ f)^{-1}(z) = \{f^{-1}(y) / y \in g^{-1}(z)\}$ , pour tout  $z \in \text{range}(g \circ f)$ ,

couplage :  $(f \wedge g)^{-1}((y, z)) = f^{-1}(y) \cap g^{-1}(z)$ ,

restriction :  $(f/E)^{-1}(y) = E \cap f^{-1}(y)$ , pour tout  $y \in \text{range}(f/E)$ ,



## 6.6 Expressions de chemin et langage OLAP

Un schéma dimensionnel aussi appelé *Schéma Etoile* est un graphe connexe, orienté, acyclique, étiqueté tel que : une seule racine, appelé l'origine et notée O. Chaque noeud a une étiquette (dimension) et chaque arête (flèche) a une étiquette. Toutes les flèches ont des étiquettes distinctes. Les flèches de source O sont d'un de deux types, dimension ou mesure.

Terminologie : la notation  $f : X \rightarrow Y$  indiquera une flèche avec étiquette f, source X et cible Y dans le schéma; de même, on parlera de la source et de la cible d'un chemin dans le schéma.

- Chaque sommet autre que l'origine est appelé attribut,
- les valeurs du domaine de O sont appelées des objets,
- la distinction entre dimensions et mesures est faite par le concepteur.
- Tout schéma dimensionnel doit contenir au moins une mesure. Un *chemin dimensionnel* est un chemin de source O finissant par une dimension. Tout attribut d'un tel chemin est appelé un niveau d'agrégation.
- Un *chemin de mesure* : tout chemin de source O finissant par une mesure
- Tout attribut d'un tel chemin est appelé un niveau de mesure.

**Base de données dimensionnelle (bdd)** Etant donné un schéma dimensionnel S, une bdd sur S est une fonction  $\delta$  qui associe chaque sommet A de S à un sous-ensemble fini  $\delta(A)$  du domaine de A, chaque autre flèche  $f : X \rightarrow Y$  de S à une fonction totale  $\delta(f) : \delta(X) \rightarrow \delta(Y)$  telle que:

- les fonctions d'une bdd peuvent être données de manière explicite ou implicite
- dans la suite, nous omettrons le symbole (, le contexte indiquant s'il s'agit d'une flèche ou d'une fonction
- le fait que les fonctions d'une bdd soient totales entraîne la contrainte suivante :  
contrainte référentielle :  $\text{range}(f) \subseteq \text{def}(g)$  pour toutes fonctions  $f : X \rightarrow Y$  et  $g : Y \rightarrow Z$
- le passage à un niveau supérieur induit un regroupement ou "agrégation" au niveau inférieur

**Expression de chemin** Etant donné un schéma dimensionnel S, une *expression de chemin* sur S est une expression bien formée dont les opérandes sont des flèches de S et dont les opérateurs sont ceux de l'algèbre fonctionnelle.

Une *expression dimensionnelle sur S* : toute expression de chemin constituée uniquement de flèches figurant sur des chemins dimensionnels. Une *expression de mesure sur S* : toute expression de chemin constituée uniquement de flèches figurant sur un chemin de mesure.

Une *évaluation* d'une expression de chemin e par rapport a une base (bdd)  $\delta$  sur S : se fait en remplaçant chaque flèche f figurant dans e par la fonction  $\delta(f)$  qui lui est associée par  $\delta$ , et

en effectuant les opérations de l'algèbre fonctionnelle. Le résultat étant toujours une fonction, par propriété de fermeture, comme en relationnel.

Une *vue sur S* : schéma dimensionnel S' dont chaque sommet est un sommet de S et donc chaque flèche est une expression de chemin sur S. Un *data mart* est défini comme une vue sur S, pouvant être virtuelle ou matérialisée suivant les besoins de l'application.

**Requête OLAP** Etant donné un schéma dimensionnel S, un 'OLAP Pattern' sur S est un couple  $P = (u, v)$ , où u est une expression dimensionnelle, v une expression de mesure et  $\text{source}(u) = \text{source}(v) = O$ . La cible de u est appelée le niveau d'agrégation de P et la cible de v le niveau de mesure de P.

Notons que, si u comporte des couplages, alors A peut être le produit de plusieurs attributs, c'est-à-dire, le niveau d'agrégation peut être composé de plusieurs autres niveaux 'simples' (et de même pour M).

Une requête OLAP sur S est alors un couple  $Q = \langle P, op \rangle$ , où P est un OLAP pattern et op est une opération applicable sur le domaine du niveau de mesure de P. Si A est le niveau d'agrégation de P et M son niveau de mesure, alors la réponse à Q par rapport à une base  $\delta$  sur S est définie comme une fonction *ans* qui associe à chaque élément a de A un ensemble de mesures  $op(M_A)$  où  $M_a$  est l'ensemble des mesures associés aux objets ayant comme valeur a sur les attributs A.

Ce calcul est assez complexe et suit plusieurs procédures qui selon l'algèbre fonctionnelle.

**Exemple OLAP** Considérons un entrepôt qui regroupe toutes factures émises par une chaîne de magasin. Le schéma pourrait être défini par l'arbre suivant:

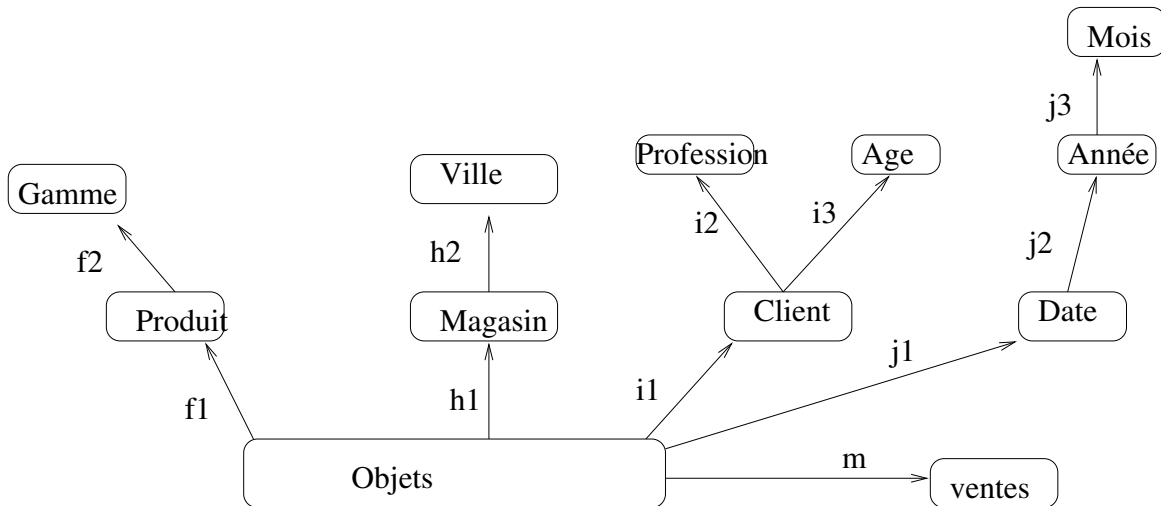


Figure 8: Un schéma Etoile.

On peut analyser le nombre de ventes par gamme de produits et par ville. Les dimensions de cette requête sont Gamme, Ville. On peut exprimer cette requête par les chemins:  $f2of1$  pour

Gamme et *h2oh1* pour Ville. La mesure est Ventes et l'opérateur est la somme.

On peut aussi donner des conditions de sélection: par exemple Année=2004, Région=Europe.

L'outil Mondrian (<http://dup2.dyndns.org:8080/mondrian/> ) permet de décrire des requêtes OLAP en procédant comme suit:

- Choix des conditions, ou du filtre,
- Choix des dimensions,
- Choix des mesures.

On affiche alors un tableau ordonné selon les valeurs des attributs. Dans le cas de l'analyse Gamme, Ville, où Gamme a les valeurs (A,B,C) et Ville les valeurs (NY, Paris, Berlin) on aura par exemple un tableau:

Résultat T:  
A, NY, 100  
A, Paris, 200  
A, Berlin, 300  
B, NY, 100  
B, Paris, 400  
B, Berlin, 500  
C, NY, 10  
C, Paris, 200  
C, Berlin, 30

Une représentation graphique par histogramme est aussi fournie.

**Exercice** Un distributeur (grossiste) approvisionne plusieurs magasins en produits, en effectuant au plus une livraison par jour et par magasin. Les informations qui figurent sur chaque bon de livraison sont les suivantes : le numéro du bon de livraison, la date de livraison, la référence du magasin, et pour chaque type de produit livré sa référence et la quantité livrée (le nombre d'articles). Ces informations sont stockées chez le distributeur, et accumulées pendant des longues périodes afin de les analyser pour améliorer le service de distribution.

Les analyses se font suivant plusieurs axes, et à plusieurs niveaux, en analysant les mouvements des produits par jour et par mois, par ville et par région, par fournisseur et par catégorie de produit.

On supposera qu'un fournisseur peut fournir au distributeur des produits dans plusieurs catégories et qu'une catégorie de produit peut être fournie par plusieurs fournisseurs.

Nota : Souvent, la référence du magasin contient l'information sur la ville et la région où se trouve le magasin, et de même, la référence du produit contient l'information sur le fournisseur et la catégorie du produit.

1/ Se convaincre que le schéma dimensionnel S vu en cours (voir figure 2) convient comme interface pour les analystes.

2/ Pour chacune des expressions de chemin suivantes sur S, dire si elle est bien formée (on dit aussi bien typée),

et si oui, donner la source, la cible, et le résultat de son évaluation sur la base dimensionnelle:

$$e1 = f \wedge g \wedge h$$

$$e2 = g1o(g)$$

$$e3 = (g1og)$$

$$e4 = (g1og) \wedge (f1of) \wedge (ho(h12))$$

3/ Définir et évaluer les requêtes OLAP suivantes (pour le schéma de la figure 2 et la base de la table 1) :

Q1 : Les totaux de quantités livrées par type de produit

Q2 : Les totaux de quantités livrées par catégorie

Q3 : Les minimaux de quantités livrées par ville

Q4 : Les moyennes de quantités livrées par ville et type de produit

Q5 : Les totaux de quantités livrées par catégorie et ville

Q6 : Les maximaux de quantités livrées par date et catégorie

Q7 : Les totaux de quantités livrées par région et par mois

Q8 : Les totaux de quantités livrées par fournisseur, magasin et mois

Q9 : La somme de toutes les quantités livrées

4/ Définir le schéma d'un data mart permettant des analyses mensuelles par ville et catégorie de produits,

et concernant uniquement les villes de Paris et de Lyon. Dans le contexte de ce data mart, définir la requête suivante

et donner sa traduction en une requête sur l'entrepôt.

Q10 : Les totaux de quantités livrés par catégorie de produits, pendant le premier trimestre

## 7 Data-Mining

Le terme Data-Mining est utilisé dans de nombreux contextes avec un sens différent. Dans son interprétation la plus simple, il s'agit de trouver une fonction  $f$  sur la base d'échantillons de valeurs  $x_i, y_i = f(x_i)$ .

Considérons les exemples suivants:

- $f$  est une fonction booléenne:  $\{0, 1\}^n \rightarrow \{0, 1\}$ .
- $f$  est une fonction de  $D^2 \rightarrow \{0, 1\}$  où  $D = [1, \dots, N]$  est un segment réel.
- $f$  est une fonction de  $\Sigma^* \rightarrow \{0, 1\}$  où  $\Sigma$  est un alphabet fini.
- $f$  est une fonction de  $D_1 \times D_2 \times \dots \times D_k \rightarrow \{0, 1\}$  où chaque  $D_i$  est un domaine discret de valeurs.

Ces exemples sont représentatifs de problèmes théoriques et pratiques importants.

- Dans le premier cas, on peut imaginer un circuit booléen composé de portes logiques qui prend  $n$  bits en entrée et fournit une sortie  $y$  de 1 bit. Si l'on considère le circuit  $C$  comme une boîte noire, on cherchera alors à connaître  $f$  à partir de mesures. Une mesure est l'observation d'un  $x_i \in \{0, 1\}^n$  et de la valeur de sortie  $y_i$ .

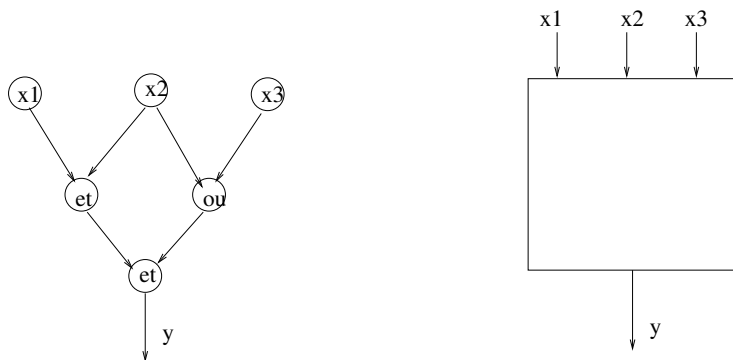
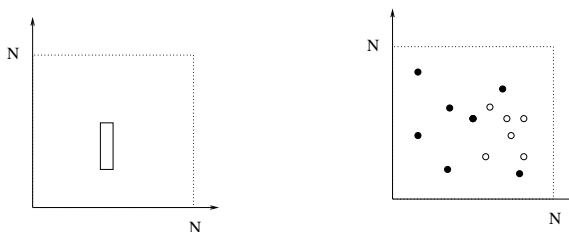


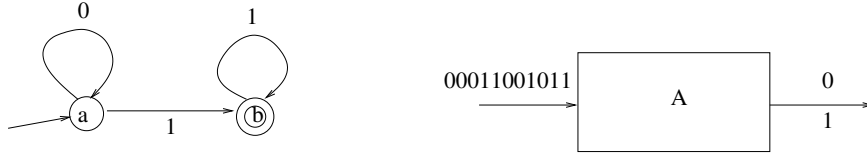
Figure 9: Un circuit décrivant la fonction  $f = (x_1 \wedge x_2) \wedge (x_2 \vee x_3)$  et un circuit décrivant une fonction inconnue.

- Si  $f$  est une fonction de  $D^2 \rightarrow \{0, 1\}$  où  $D = [1, \dots, N]$  est un segment réel, alors  $f$  décrit une figure géométrique dans l'espace à deux dimensions. On peut chercher à connaître cette figure sur la base d'échantillons.



Une fonction  $f : D^2 \rightarrow \{0, 1\}$  où  $D = [1, \dots, N]$  est un segment réel et une fonction  $g$  inconnue dont on ne connaît que des échantillons.

- Si  $f$  est une fonction de  $\Sigma^* \rightarrow \{0, 1\}$  où  $\Sigma$  est un alphabet fini, alors  $f$  décrit un langage, i.e.  $L = \{x \in \Sigma^*, f(x) = 1\} \subseteq \Sigma^*$ .



Une fonction  $f : D^2 \rightarrow \{0, 1\}$  où  $D = [1, \dots, N]$  décrit une figure (rectangle) et une fonction  $g$  inconnue dont on ne connaît que des échantillons.

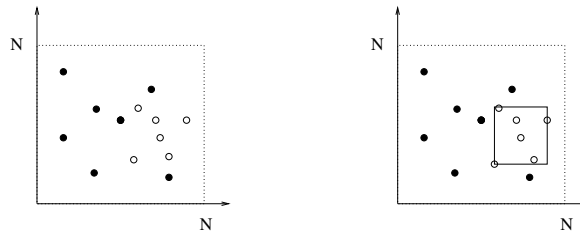
- Si  $f$  est une fonction de  $D_1 \times D_2 \times \dots \times D_k \rightarrow \{0, 1\}$  où chaque  $D_i$  est un domaine discret de valeurs, on peut imaginer  $f$  comme une table:

$D_1$	$D_2$	...	$D_k$	Val
a	b	..	h	1
a	b'	..	h'	0
a	b''	..	h'	1

Par contre, on peut aussi imaginer les lignes de la table comme des échantillons d'une fonction  $f$  inconnue que l'on cherche à apprendre.

Une caractéristique générale est que l'on va proposer une fonction  $g$  approximativement proche de  $f$  sur la base d'échantillons. Une question fondamentale est de trouver la bonne distance entre deux fonctions. La distance semble assez naturelle dans le cas de l'exemple géométrique où on dira que  $f$  est  $\varepsilon$  proche de  $g$  si tout point  $x$  tel que  $f(x) = 1$  est à distance euclidienne inférieure à  $\varepsilon.N$  de  $x'$  tel que  $g(x') = 1$  et symétriquement si tout point  $x$  tel que  $g(x) = 1$  est à distance euclidienne inférieure à  $\varepsilon.N$  de  $x''$  tel que  $f(x'') = 1$ .

Si on tire  $M$  échantillons  $x_i$  au hasard tels que  $f(x_i) = 1$  et tels que  $f(x_i) = 0$  soient obtenus avec une probabilité constante, alors certaines figures géométriques peuvent être apprises très efficacement.



Si  $f : D^2 \rightarrow \{0, 1\}$  est un rectangle, alors la fonction  $g$  définie par les points extrêmes sera une bonne approximation de  $f$ .

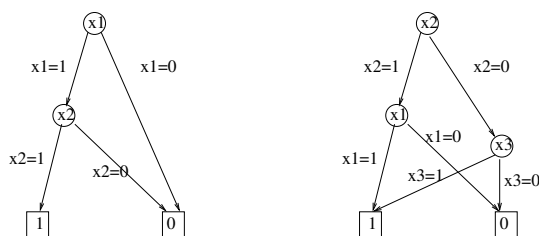
Dans le cas d'un langage régulier, la distance entre deux mots peut être la distance d'Editions avec déplacements. Dans ce cas tout langage régulier est facilement apprenable.

Dans la suite, on considère un tableau qui peut être un entrepôt de données, comme on l'a vu dans le modèle OLAP. On cherche alors à prédire la valeur d'un champs appelé TARGET, à partir d'une phase d'apprentissage et donc à apprendre la fonction  $f$  sur la base d'échantillons. On pourra alors prédire la valeur de ce champs, connaissant les valeurs des autres champs. Cette procédure est la base du Data-Mining et son utilisation est au coeur du *Business Intelligence*.

## 7.1 Arbres de décision

Un arbre de décision est une représentation succincte d'une fonction. Dans le cas d'une fonction booléenne, on imagine un arbre dont les noeuds sont les variables et dont les arêtes représentent les valeurs des variables.

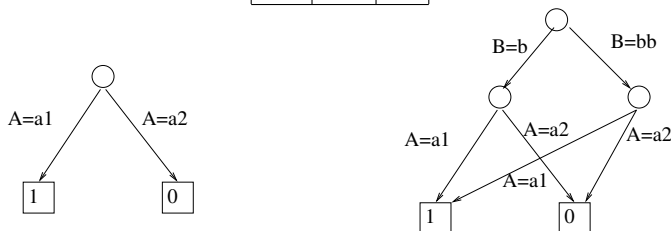
Deux noeuds terminaux correspondent aux valeurs de la fonction. L'arbre de décision est déterminé par l'ordre des variables.



Deux arbres de décision pour la fonction  $f = (x_1 \wedge x_2) \wedge (x_2 \vee x_3)$ , selon deux ordres différents:  $x_1, x_2, x_3$  et  $x_2, x_1, x_3$ .

Dans le cas d'une table  $T$  avec les attributs  $A, B, C$ , supposons que la 3ème colonne soit la valeur de la fonction. On peut donc représenter la table suivante par plusieurs arbres possibles:

A	B	C
a1	b	1
a1	bb	1
a2	b	0
a2	bb	0



Deux arbres de décision pour la fonction  $f$  définie par la table précédente.

La réalité d'un entrepôt de données est que les échantillons sont bruités, c'est-à-dire que l'on peut avoir à la fois  $(a1, b, 1)$  et  $(a1, b, 0)$ . D'autre part le domaine de certains attributs peut être continu et non plus discret et la valeur de la fonction  $f$  est continue et non plus 0 ou 1. Il s'agit donc de trouver un arbre de décision qui va approximer une fonction inconnue dont on observe des échantillons bruités.

**Dnition 7.** *Un arbre de décision associé à une table  $R$  ayant un attribut target  $T$  est un arbre dont les noeuds sont étiquetés par des attributs, les arêtes sortantes par les valeurs de ces attributs, et les feuilles par les valeurs de la fonction ou par un symbole indéterminé.*

On souhaite définir précisément quand un arbre de décision approxime une table  $R$ .

### 7.1.1 Sélection parmi les Attributs dans un arbre de décision

Il est important d'obtenir très rapidement un arbre qui soit une bonne approximation. Un algorithme standard (ID3) implémenté dans tous les outils de Business Intelligence procède par sélection d'attributs selon le Gain d'Information de chaque attribut.

L'entropie  $Ent(S) = -\sum_i p_i \log p_i$  d'une distribution mesure le degré d'incertitude. Le minimum 0 est atteint pour une distribution de Dirac, pour laquelle la situation est certaine. Une valeur  $p_i = 1$  et les autres  $p_j = 0$  pour  $j$ . Le maximum  $\log n$  est atteint pour la distribution uniforme où chaque  $p_i = 1/n$ .

Soit  $S$  la distribution de la table  $T$  précédente, correspondant au fichier test.txt ci-dessous, c'est-à-dire la distribution où un tuple est tiré au hasard selon la distribution uniforme. Dans ce cas, il y a 4 tuples et chacun est tiré avec la probabilité  $p_i = \frac{1}{4}$ .

T: A,B,C

a1,b,1;

a1,bb,1;

a2,b,0;

a2,bb,00;

La Source  $S$  associée à cette table génère un tuple aléatoirement et révèle la valeur  $C$ , soit 1, soit 0. Le nombre d'enregistrements où  $C = 1$  est 2 et le nombre d'enregistrements où  $C = 0$  est également 2. On a donc deux observations, chacune avec probabilité  $\frac{1}{2}$ .  $Ent(S) = -\sum_i p_i \log p_i = \sum_i \frac{1}{2} \log 2 = \log 2$ .

Soit  $S_{a1}$  la distribution selon  $A = a1$ , c'est-à-dire les enregistrements où la valeur de  $A$  est  $a1$ .

Le *Gain d'Information*, aussi appelé Gain d'Entropie, est une mesure de l'information d'un attribut pour la source  $S$ .

$$Gain(S, A) = Ent(S) - \sum_{\nu \in A} \frac{|S_\nu|}{|S|} \cdot Ent(S_\nu)$$

On soustrait à  $Ent(S)$ , l'Entropie conditionnelle  $Ent(S|A) = \sum_{\nu \in A} \frac{|S_\nu|}{|S|} \cdot Ent(S_\nu)$ .

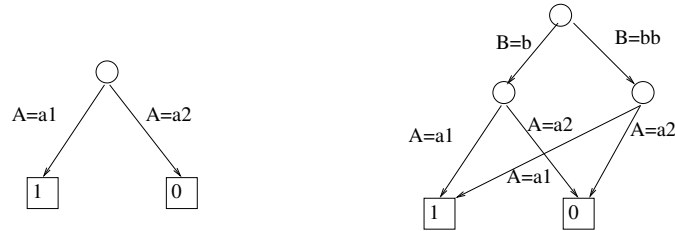
Dans l'exemple précédent, il y a deux valeurs possibles pour l'attribut  $A$ , soit  $A = a1$ , soit  $A = a2$ .  $Ent(S_{a1}) = 0$  car la source vaut 1 avec probabilité 1 et  $Ent(S_{a2}) = 0$  car la source vaut 0 avec probabilité 1. Dans les deux cas la source est certaine, car  $A \rightarrow C$ , il existe une dépendance fonctionnelle entre  $A$  et  $C$ . On a donc  $Ent(S|A) = \log 2$ .

Similairement  $Gain(S, B) = Ent(S) - \sum_{\nu \in B} \frac{|S_\nu|}{|S|}$  et  $Ent(S_b) = Ent(S_{bb}) = \log 2$  car chaque source prend la valeur 1 ou 0 avec probabilité  $\frac{1}{2}$ . On a aussi  $\frac{|S_b|}{|S|} = \frac{|S_{bb}|}{|S|} = \frac{1}{2}$  et  $Ent(S|B) = \sum_{\nu \in B} \frac{|S_\nu|}{|S|} \cdot \log 2 = \frac{1}{2} \log 2 + \frac{1}{2} \log 2 = \log 2$  et finalement  $Gain(S, B) = Ent(S) - Ent(S|B) =$



$$\log 2 - \log 2 = 0$$

L'algorithme ID3 calcule pour chaque noeud le gain pour chaque attribut  $A_i$ . Dans cet exemple, on choisit  $A$  qui maximise le gain et on décompose le noeud avec autant de fils que de valeurs de  $A$ . D'autres critères peuvent être choisis: le *rapport de gain* qui compense le gain pour tenir compte du nombre de valeurs, le *coefficient de Gini* ou le coefficient du  $\chi^2$ .

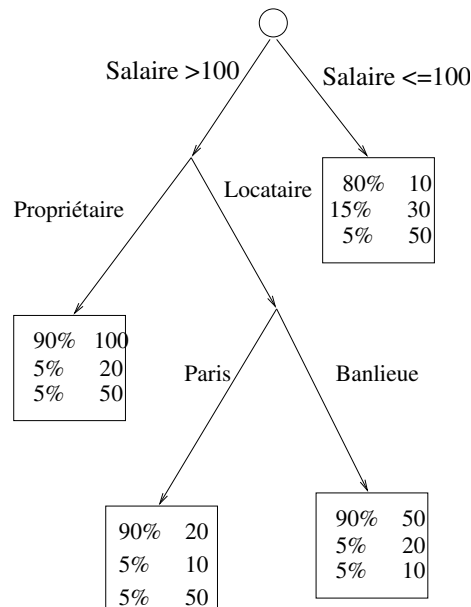


Deux arbres de décision obtenus pour le fichier test.txt.

### 7.1.2 Approximations et arbre de décision

Dans un cas réel, on peut imaginer 30 attributs, des valeurs cibles discrètes, par exemple 10, 20, 50, 100 au lieu de 0 et 1 et  $10^6$  tuples. On recherche alors un arbre de profondeur minimum et chaque feuille correspond à une valeur prépondérante. Par exemple, la feuille gauche correspond à la valeur prépondérante de 100, à 90%.

Chaque feuille est une distribution de valeurs, qui permet de prédire une valeur et une confiance. Ainsi la feuille gauche correspondra à la prédiction 100 avec la confiance 90% ou la valeur 20 avec la confiance 10%, ou la valeur 50 avec la confiance 10%.



Un arbre de décision pour le fichier test1.txt.

Chaque arbre va donc approximer la fonction  $f$  en prédisant pour chaque tuple  $a_1, \dots, a_n$ , la valeur majoritaire de la feuille obtenue en suivant l'arbre. L'erreur est toujours la proportion de mauvaises prédictions.

## 7.2 Régression linéaire et logistique

Une régression linéaire est une interpolation de  $N$  points en dimension  $n$  associés à une fonction  $y = f(x_1, \dots, x_n)$  à l'aide d'une équation linéaire  $a_0 + a_1.x_1 + a_2.x_2 + \dots + a_n.x_n = 0$ . Cette situation est adaptée aux attributs numériques, par opposition aux attributs discrets.

Etant donnés  $N$  points que l'on peut interpréter comme des tuples  $(t_1, \dots, t_N)$  on recherche les coefficients  $a_0, a_1, \dots, a_n$  qui minimisent un critère général, qui peut être pris comme la somme des distances de chaque point à l'hyperplan défini par l'équation linéaire.

Soit  $A$  matrice représentant la table qui donne la liste des tuples, chaque tuple étant une ligne de dimension  $n + 1$ , car la 1ère valeur est un 1.  $A_i$  est la ligne  $i$ . On a donc  $N$  lignes et  $n + 1$  colonnes. Soit  $b$  le vecteur de dimension  $N$  dont les valeurs sont celles de la fonction  $f$  et  $\alpha$  un vecteur dont la valeur sur la ligne  $i$  est  $a_i$ . On recherche alors un vecteur  $\alpha$  de dimension  $n + 1$  tels que:

$$A.\alpha \approx_{\varepsilon} b$$

L'erreur considérée est:

$$\sum_{i=1, \dots, N} \frac{(A_i.\alpha - b_i)^2}{2.N}$$

soit la somme pondérée pour chaque tuple des écarts au carré, entre la valeur prédite  $A_i.\alpha$  et la valeur réelle  $b_i$ . Dire que  $A.\alpha \approx_{\varepsilon} b$ , c'est garantir que l'erreur relative est alors inférieure à  $\varepsilon$ . Le vecteur  $\alpha$  détermine l'hyperplan recherché.

Exemple: Considérons une fonction qui donne le prix d'appartements (1 unité=10<sup>6</sup> Euros) en fonction de critères comme les m<sup>2</sup>, le nombre de pièces, les quartiers, etc... Pour  $n = 1, N = 3$ , des données possibles seraient:

100, 1  
200, 2  
300, 2.2

Le premier tuple indique qu'un appartement de 100m<sup>2</sup> vaut 1 million d'Euros. La matrice  $A$  est:

$$A = \begin{pmatrix} 1 & 100 \\ 1 & 200 \\ 1 & 300 \end{pmatrix}$$

La 1ère colonne est toujours une colonne de 1. Le vecteur  $b$  est:

$$b = \begin{pmatrix} 1 \\ 2 \\ 2.2 \end{pmatrix}$$

Une solution possible est  $0.3 + 0.009.x_1$  et l'erreur serait de 0.04 pour le premier tuple, 0.01 pour le second tuple et 0.64 pour le troisième tuple. L'erreur globale serait de 0.106 .

Pour  $n = 2, N = 3$ , des données possibles seraient:

100, 5, 1

200, 7, 2

300, 10, 2.2

Le premier tuple indique qu'un appartement de 100m<sup>2</sup> et de 5 pièces vaut 1 million d'Euros. Les valeurs numériques du 1er attribut sont bien supérieures à celles du 2ème. Il est alors utile de normaliser toutes les valeurs numériques pour obtenir des valeurs entre 0 et 1. Une solution possible serait alors

$$a_0 + a_1.x_1 + a_2.x_2$$

Comment obtenir la solution optimale? Le vecteur  $\alpha$  est solution de l'équation:

$$\alpha = (A^t.A)^{-1}.A^t.b$$

On appliquant cette formule, on trouve:

$$\alpha = \begin{pmatrix} 0.53 \\ 0.006 \end{pmatrix}$$

L'équation optimale est donc  $0.53 + 0.006.x_1$ . L'erreur est alors de 0.015.

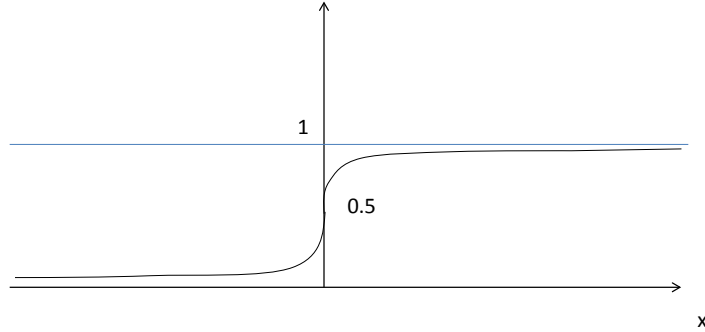
Il est souvent plus efficace d'appliquer un algorithme de gradient, qui part d'une solution et la modifie pour l'améliorer à chaque étape, dès que les valeurs de  $N$  ou  $n$  dépassent  $10^4$ . En effet l'inverse d'une matrice de cette taille pose des problèmes de précision et de temps de calcul.

### 7.2.1 Régression logistique

Une régression logistique suit une approche similaire pour *classifier* des objets. Dans le cas le plus simple, on suppose deux classes différentes et les échantillons indiquent s'ils sont de la classe 1 ou de la classe 0. On cherche alors une fonction linéaire  $\alpha^t.x = a_0 + a_1.x_1 + a_2.x_2 + \dots.a_n.x_n = 0$  qui sépare au mieux les échantillons de la classe 1 de ceux de la classe 0. Plus généralement on peut aussi rechercher une fonction polynomiale, par exemple  $2 + 3x_1 - 4x_2 + 5x_1.x_2 + 6x_1^2 - 7x_2^3$ , dans le cas où la frontière entre les points n'est pas un hyperplan.

On utilise une fonction non linéaire spécifique, appelée sigmoïde, pour ramener entre 0 et 1 les grandes valeurs possibles positives ou négatives de  $\alpha^t.x$ . La définition d'une sigmoïde est:

$$g(z) = \frac{1}{1 + e^{-z}}$$



Une sigmoïde  $f(x) = \frac{1}{1+e^{-x}}$

La classification est alors basée sur le test  $g(\alpha^t \cdot x) \geq 1/2$  ce qui est équivalent à  $\alpha^t \cdot x \geq 0$ . La fonction de potentiel associée est:

$$J(\alpha) = 1/m \sum_{i=1 \dots m} -y^i \cdot \log(g(\alpha^t \cdot x)) - (1 - y^i) \cdot \log(1 - g(\alpha^t \cdot x))$$

Un algorithme de gradient cherche à minimiser cette fonction, en réalisant des petites variations à partir d'une solution  $\alpha$ . Cette fonction admet des variantes, en particulier *la régularisation* qui rajoute un terme

$$\lambda \cdot \sum_{i=1 \dots n} a_i^2$$

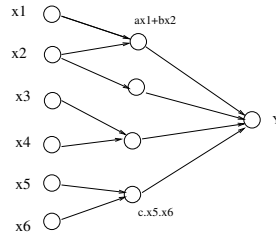
afin de privilégier des petits coefficients des  $a_i$ .

### 7.3 Réseaux de neurones

Un réseau de neurones généralise la régression logistique en considérant plusieurs niveaux de variables  $y_i$ , associées à des régressions logistiques. On représente cette fonction par un circuit dont les portes élémentaires sont:

- des additions pondérées :  $a \cdot x_i + b \cdot x_j$ ,
- des sigmoïdes:  $f(z) = \frac{1}{1+e^{-z}}$
- des fonctions de seuil :  $1$  si  $x_i \geq d$ ,  $0$  sinon.

La sortie du circuit est la valeur de la fonction. On recherche alors tous les coefficients du circuit qui minimisent l'erreur.



Un réseau de neurones.

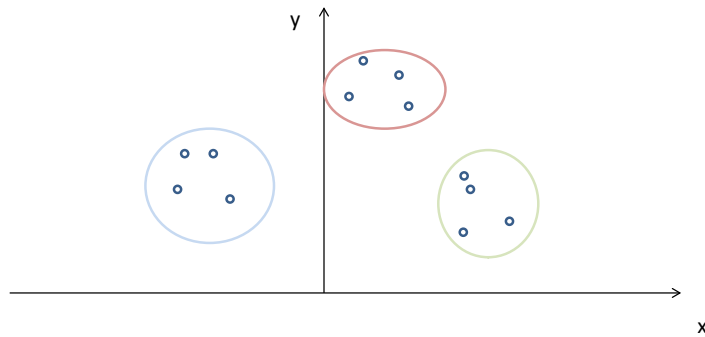
## 7.4 Apprentissage non supervisé: $k$ -Means

Dans ce cas, les échantillons sont des points dans un espace de dimension  $d$  si le tableau de données a  $d$  colonnes.

L'algorithme  $k$ -Means fixe un nombre de classes (clusters)  $k$  à l'avance. Il procède en itérant deux étapes à partir de  $k$  points aléatoires  $c_1, \dots, c_k$ , pris parmi les échantillons:

- Colorer chaque point  $c$  par l'index  $i$  du  $c_i$  le plus proche.
- Définir les barycentres des points de la même couleur,  $c_1^1, \dots, c_k^1$

En itérant ces deux opérations, on obtient une séquence  $c_1^1, \dots, c_k^1$ , puis  $c_1^2, \dots, c_k^2$ , puis  $c_1^3, \dots, c_k^3, \dots$  puis  $c_1^j, \dots, c_k^j$  après  $j$  itérations. On arrête l'algorithme quand les distances entre  $c_i^j$  et  $c_i^{j+1}$  sont inférieures à some  $\varepsilon$  pour  $i = 1, \dots, k$ .



Clusters trouvés par 3-Means pour  $d = 2$ .

## 7.5 Graph Mining

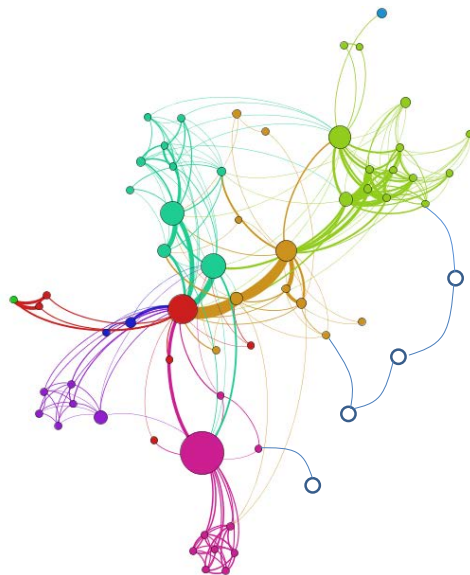
Les réseaux sociaux comme Facebook ou Twitter génèrent des graphes  $G = (V, E)$  où  $E \subseteq V.V$  en identifiant des noeuds (utilisateurs pour FB, tags pour Twitter) et des arêtes ( amis pour FB)

et liens par message pour Twitter. Si un tweet est envoyé par @x et contient @y et z (parmi les 140 caractères), alors on a deux arêtes: @x E @y et @x E z. Ces graphes ont des caractéristiques remarquables:

- les graphes sont connexes et de petit diamètre (Environ 6 pour Facebook, comme dans l'expérience de Milgram),
- la distribution des degrés est une loi  $1/n^k$  (power law).

Beaucoup d'autres propriétés sont aussi importantes. Un problème des réseaux sociaux est la détection de communauté, c'est-à-dire le regroupement des noeuds en classes (par couleur). Pour cela on utilise des variantes du problème Mincut, la coupure minimale pour deux classes.

On peut utiliser la programmation linéaire et la méthode de Ford-Fulkerson pour trouver la coupe Minimum entre deux points  $s, t$ . On peut aussi utiliser l'algorithme randomisé de Karger qui réduit le graphe par étape en éliminant des arêtes aléatoires.



Réseau social avec ses communautés

Le logiciel Gephi est utilisé pour visualiser des graphes sociaux en distinguant 3 étapes:

- spatialisation: placer les noeuds dans l'espace,
- le ranking: la taille des noeuds est lié à leur degré
- la détection de communauté.

Un expérience classique est de se brancher au flux Twitter à partir de mots-clés (Naoyun) puis d'observer le graphe. La description des noeuds capture le contenu des tweets et toutes les informations sur leurs auteurs.

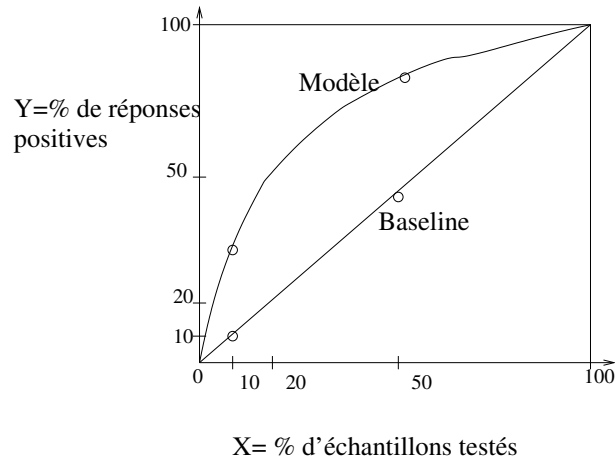
## 7.6 Qualité de l'approximation

Le principal critère d'une technique prédictive est le taux de bonne réponse, d'erreur et de non prédiction.

Des critères plus précis distinguent une qualité de prédiction. En effet dans le cas d'un arbre de décision partiel, c'est-à-dire dont les noeuds ne sont:

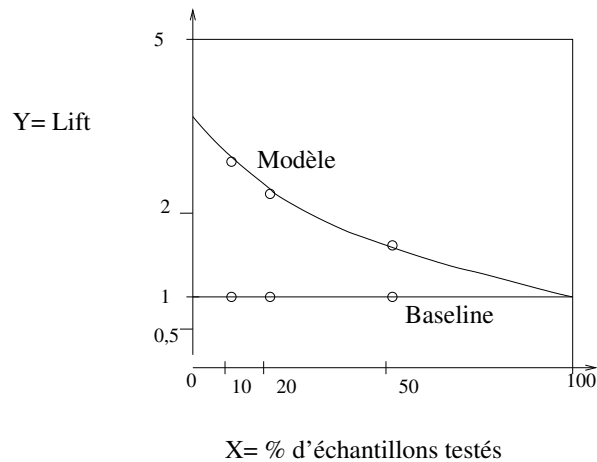
- la courbe de Gains (Cumulative Gains Chart)
- la courbe de Lift (Lift Chart)

La courbe de Gains donne le pourcentage  $Y$  de bonnes réponses pour les  $X\%$  meilleures prédictions, lorsque  $X = 10, 20, \dots, 100$ . On compare  $Y$  à la valeur  $Y'$  correspondant à une décision aléatoire, appelée *Baseline*.



Une courbe de Gains.

Le Lift est défini comme le rapport des pourcentages de bonne réponse entre le modèle et la Baseline.



Une courbe de Lift.

## 7.7 Logiciels de Fouille de Données

Nous décrivons trois logiciels qui implémentent les techniques de fouilles de données.

### 7.7.1 Dtree

Ce logiciel très simple construit des arbres de décision, à partir d'un fichier de données, sur des valeurs discrètes.

### 7.7.2 SAS/Entreprise Miner

Ce module de SAS intègre les trois principaux modèles et permet de les comparer, en sortant des courbes de lift.

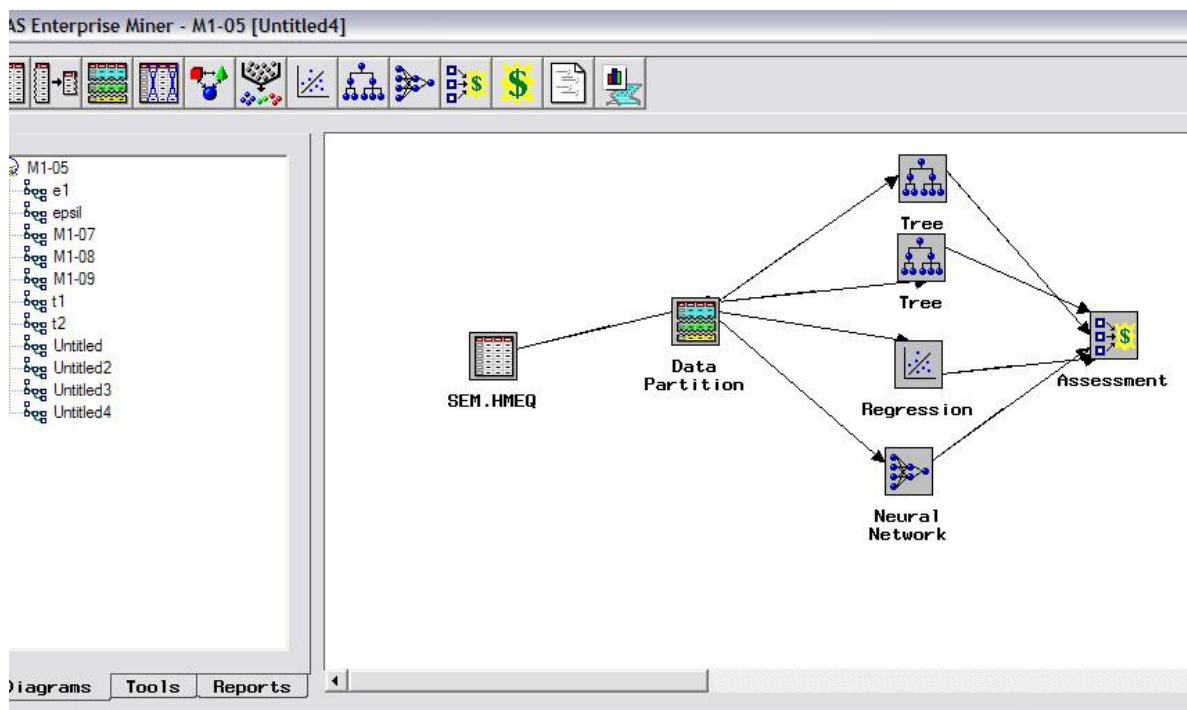


Figure 10: La construction de modèles dans SAS/Entreprise Miner

### 7.7.3 Weka

Ce logiciel libre a une interface, largement inspirée de celle de SAS/Entreprise Miner. De très nombreux modèles sont disponibles, ce qui rend le logiciel plus difficile à utiliser.



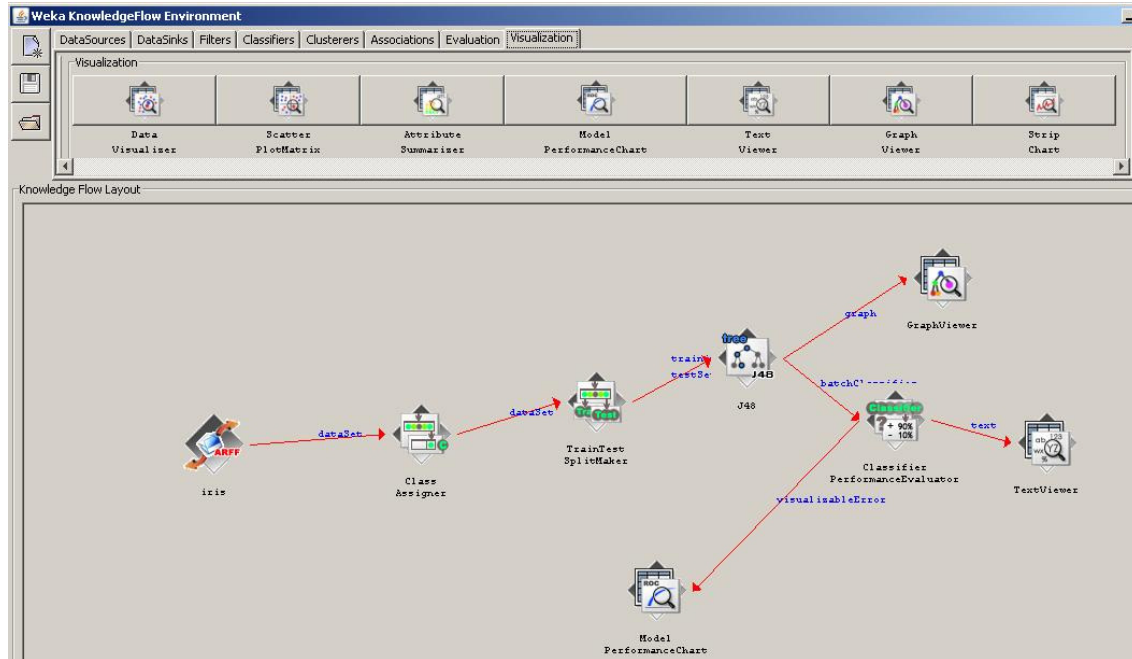


Figure 11: La construction de modèles dans Weka

## 8 Mécanismes

Le modèle OLAP considéré suppose des données sur un serveur spécifique. On souhaite généraliser ce modèle lorsque les données se trouvent sur des serveurs distincts, générées par des clients distincts.

Le standard d'échange est XML, les données sont décrites par une DTD et stockées sur un serveur. Supposons plusieurs entrepôts de données stockant des enregistrements proches sur des serveurs différents. Il faut donc trouver une représentation statistique qui fait l'union de différentes représentations statistiques sur chaque serveur afin de répondre à des analyses OLAP. C'est un domaine de la recherche actuelle sur *l'Intégration de Données*. La conception de modules ETL (Extract Transform Load) présents dans les principaux outils de Business Intelligence (SAS, SPSS, Business Object) s'inspire directement des recherches actuelles. Certains outils du logiciel libre commencent à apparaître, tels Pentaho sur <http://www.pentaho.org>, à partir de Mysql et Mondrian.

On peut supposer que  $N$  sources de données doivent être intégrées sur un serveur cible (target), et le problème est alors de comprendre l'interaction possible entre  $N$  acteurs. La théorie des jeux et des mécanismes présentent un point de vue important pour des économistes, car il permet d'introduire des notions d'utilité, de prix au sein des systèmes informatiques.

### 8.1 Jeux à $N$ joueurs

Différents clients qui interagissent avec des serveurs peuvent être considérés comme des agents indépendants qui cherchent à réaliser des tâches particulières. Le modèle de jeu définit une *utilité* pour chaque décision élémentaire des  $N$  joueurs et un des problèmes classiques est de trouver des équilibres de Nash, stables. Si chaque client souhaite maximiser sa bande passante, c'est-à-dire la

quantité d'information qu'il peut échanger, sur le réseau Internet, un équilibre de Nash déterminera le trafic possible pour chaque client.

Les réseaux informatiques sont donc de bons exemples de jeux où le nombre de joueurs est grand et où la possibilité de trouver de bons équilibres devient un enjeu économique.

For a natural number  $n$ , we denote by  $[n]$  the set  $\{1, \dots, n\}$ . For an integer  $r \geq 2$ , an  $r$ -players game in normal form is specified by a set of pure strategies  $S_p$ , and a utility or payoff function  $u_p : S \rightarrow R$ , for each player  $p \in [r]$ , where  $S = S_1 \times \dots \times S_r$  is the set of pure strategy profiles. For  $s \in S$  the value  $u_p(s)$  is the payoff of player  $p$  for pure strategy profile  $s$ . Let  $S_{-p} = S_1 \times \dots \times S_{p-1} \times S_{p+1} \times \dots \times S_r$ , the set of all pure strategy profiles of players other than  $p$ . For  $s \in S$ , we set the partial pure strategy profile  $s_{-p}$  to be  $(s_1, \dots, s_{p-1}, s_{p+1}, \dots, s_r)$ , and for  $s'$  in  $S_{-p}$ , and  $t_p \in S_p$ , we denote by  $(s'_{-p}, t_p)$  the combined pure strategy profile  $(s'_1, \dots, s'_{p-1}, t_p, s'_{p+1}, \dots, s'_r) \in S$ . We will suppose that  $S_p = [n]$  for all  $p \in [r]$ .

A mixed strategy for player  $p$  is a probability distribution over  $S_p$ , that is a vector  $x_p = (x_p^1, \dots, x_p^n)$  such that  $x_p^i \geq 0$ , for all  $i \in [n]$ , and  $\sum_{i \in [n]} x_p^i = 1$ . We define  $\text{support}(x_p)$ , the support of the mixed strategy  $x_p$  as the set of indices  $i$  for which  $x_p^i > 0$ . We denote by  $\Delta_p$  the set of mixed strategies for  $p$ , and we call  $\Delta = \Delta_1 \times \dots \times \Delta_r$  the set of mixed strategy profiles. For a mixed strategy profile  $x = (x_1, \dots, x_r)$  and pure strategy profile  $s \in S$ , the product  $x^s = x_1^{s_1} x_2^{s_2} \dots x_r^{s_r}$  denotes the probability of  $s$  in  $x$ . The payoff functions naturally extend from  $S$  to  $\Delta$  by setting  $u_p(x) = \sum_{s \in S} x^s u_p(s)$ . The set  $\Delta_{-p}$ , the partial mixed strategy profile  $x_{-p}$  for  $x \in \Delta$ , and the combined mixed strategy profile  $(x', x_p)$  for  $x' \in \Delta_{-p}$  and  $x_p \in \Delta_p$  are defined analogously to the pure case. The pure strategy  $s_p$  is a best response for player  $p$  against the partial mixed strategy profile  $x_{-p}$  if it maximizes  $u_p(x_{-p}, \cdot)$ . We will denote by  $\text{best}(x_{-p})$  the set of best responses against  $x_{-p}$ .

A Nash equilibrium is a mixed strategy profile  $x^*$  such that for all  $p \in [r]$ , and for all  $x_p \in \Delta_p$ ,

$$u_p(x^*_{-p}, x_p) \leq u_p(x^*).$$

An equivalent condition is  $u_p(x^*_{-p}, s_p) \leq u_p(x^*)$  for every  $s_p \in \text{best}(x^*_{-p})$ . Nash has shown in the 1950s that for games with a finite number of players there exists always an equilibrium. It is immediate that the set of Nash equilibria is invariant by translation and positive scaling of the utility functions. Therefore we will suppose that they take values in the interval  $[0, 1]$ .

Computing a Nash equilibrium is not an easy task. It was proven recently in a series of papers [?], that this computation is complete for the class PPA, even when the number of players is restricted to two, and therefore it is unlikely to be feasible in polynomial time.

Several relaxations of the notion of equilibrium have been considered in the form of additive and multiplicative approximations. Let  $\varepsilon > 0$ . An additive  $\varepsilon$ -approximate equilibrium is a mixed strategy profile  $x^*$  such that for all  $p \in [r]$ , and for all  $x^p \in \Delta_p$ ,

$$u_p(x^*_{-p}, x_p) \leq u_p(x^*) + \varepsilon.$$

A multiplicative  $\varepsilon$ -approximate equilibrium is a mixed strategy profile  $x^*$  such that for all  $p \in [r]$ , and for all  $x^p \in \Delta_p$ ,

$$u_p(x^*_{-p}, x_p) \leq (1 + \varepsilon)u_p(x^*).$$

Since by our convention  $0 \leq u_p(x^*) \leq 1$ , a multiplicative  $\varepsilon$ -approximate equilibrium is always an additive  $\varepsilon$ -approximate equilibrium, but the converse is not necessarily true.

The input size of a  $r$ -players game is the length of the description of  $rn^r$  rational numbers. Here we will consider the computational model where arithmetic operations and comparisons have unit cost.

### 8.1.1 Jeux à deux joueurs à somme nulle

Dans ce cas, on note  $A$  la matrice de gain du joueur I et  $B = -A$  la matrice de gain du joueur II. Il existe un algorithme polynomial, utilisant la programmation linéaire, pour trouver un équilibre de Nash, qui correspond aux meilleures stratégies de chaque joueur.

### 8.1.2 Jeux à deux joueurs

Dans le cas général, il n'existe pas d'algorithme polynomial pour trouver un équilibre de Nash.

## 8.2 Mécanismes

Dans l'approche des mécanismes, on observe un équilibre et on cherche un jeu dont l'équilibre est l'équilibre recherché. Ce jeu décrit l'interaction d'un joueur avec ses voisins et une fonction d'utilité. L'informatique distribuée s'intéresse aux interactions mais ne donne pas de vision économique des situations possibles.

## 8.3 Le Mécanisme Adwords

Un des mécanismes récents les plus célèbres est *Adwords* introduit par Google, pour gérer la publicité ciblée, c'est-à-dire les encarts publicitaires attachés aux réponses du moteur de recherche. Une nouvelle industrie est née sur ce modèle et de nouveaux services dont le modèle économique sera inspiré de celui de Google vont sans-doute se développer.

Des annonceurs paient sur des ensembles de mots-clés en fixant un prix pour chaque ensemble de mots clés et un budget journalier. Par exemple 1\$ pour *hôtel, Paris*, 2\$ pour *hôtel, Paris, centre*, 1/2\$ pour *hôtel, Marseille* et un budget de 50\$ par jour, pour une chaîne hôtelière qui auraient des offres sur Paris et Marseille. L'aspect innovant est le côté Multilingues et Multimarché. On peut mélanger des mots de plusieurs langues (anglais et chinois par exemple) et viser des marchés particuliers. Le marché chinois sur <http://www.google.com/intl/zh-CN/> ou le marché indien sur <http://www.google.co.in/>.

Le moteur de recherche répond à des milliers de demande par seconde et propose pour chaque ensemble de mots  $A, B, C$  un ensemble d'encarts ordonnés de 1 à 10 (de haut en bas) et un prix pour un Click potentiel sur l'encart, appelé CPC (Cost-Per-Click).

L'algorithme utilisé a les caractéristiques suivantes:

- Il est en-ligne (Online), c'est-à-dire qu'il répond immédiatement à partir de l'état du système,
- Il cherche à maximiser le profit de Google.

On peut imaginer une enchère de Vickrey, qui allouerait l'espace aux meilleurs annonceurs pour le prix de la seconde meilleure annonce. Le système Adwords s'inspire de ce mécanisme mais le raffine.

Chaque encart a en effet une probabilité de Click qui lui est propre et qui est estimée à l'aide des statistiques de Click. Celles-ci sont très précises et permettent de prédire le trafic d'un ensemble de mots clé tels que  $A, B, C$ . Si l'encart  $E_1$  a une probabilité de click  $p_1 = 0,1$  et un pari de 2\$, l'encart  $E_2$  peut avoir une probabilité de click  $p_2 = 0,5$  et un pari de 1\$, et  $E_2$  est alors plus intéressant que  $E_1$ . Il faut donc remplacer le pari d'un encart  $E_i$  de probabilité de click  $p_i$  et de pari de  $v_i$ , par son espérance,  $p_i.v_i$ .

Il faut aussi choisir les encarts et en utilisant aussi le budget journalier des clients. Supposons que deux clients A et B paient 1 pour deux ensembles de mots  $x$  et  $y$  avec un budget journalier

de 4. Si B a une meilleure probabilité de Click, une allocation possible gloutonne (*greedy*) à la séquence de requêtes  $xxxxyyyy$  serait  $BBBB$  et rapporterait 4. Pourtant l'allocation  $AAAABBBB$  rapporterait 8, soit deux fois plus. Le rapport d'approximation de la technique gloutonne est alors de  $1/2$  et c'est le pire cas.

Une autre technique, appelée *Balance* analysée en 2004 alloue en priorité le client qui a le moins dépensé son budget. Dans ce cas l'allocation serait:  $ABABBB$  et le rapport d'approximation est supérieur à  $3/4$ . En général le rapport d'approximation est supérieur à  $1 - 1/e$  mais est optimal parmi tous les algorithmes en ligne.