Qualité de données dans les entrepôts de données : élimination des similaires

Faouzi Boufarès*, Aïcha Ben Salem*,**, Sebastiao Correia**

*Laboratoire LIPN - UMR 7030 - CNRS, Université Paris 13
Av. J. B. Clément 93430 Villetaneuse France
{boufares, bensalem}@lipn.univ-paris13.fr,
**Société Talend, Rue Pagès 92150 Suresnes
{abensalem, scorreia}@talend.com

Résumé. Ce papier aborde la problématique de l'élimination des similaires (doublons non stricts) dans un entrepôt de données. En effet, la notion de la qualité de données présente un très grand enjeu pour une bonne gouvernance des données afin d'améliorer les interactions entre les différents collaborateurs d'une ou plusieurs organisations concernées. La présence de données en double ou similaires engendre des préoccupations importantes autour de la qualité des données. Un panorama des méthodes de calcul de distance de similarité entre les données ainsi que des algorithmes d'élimination des similaires sont exposés et comparés.

1 Introduction

Les travaux actuels sur l'extraction de connaissances à partir d'un environnement informationnel, qui se caractérise par de très grosses masses de données hétérogènes et distribuées dans les entrepôts de données (ED), se focalisent principalement sur la recherche d'information potentielle, utile et préalablement inconnue. La qualité de l'information recueillie dépend de celles des données. Prendre des décisions à partir de mauvaises informations peut nuire à l'organisation, d'où un coût de la non-qualité qui peut s'avérer très élevé. La construction d'un ED, issu de l'intégration de sources totalement hétérogènes de qualité variable, et d'outils d'aide à la décision issus de ces masses d'informations nécessite le développement de nouveaux outils d'extraction et de transformation de données (ETL - Extract Transform & Load). Ces derniers doivent, d'une part, prendre en compte l'hétérogénéité des données et leurs contraintes, et d'autre part, assurer la qualité du nouvel ensemble de données construit.
Nous avons présenté dans (Hamdoun et Boufarès, 2010) notre démarche pour intégrer des données hétérogènes. Nous avons développé un outil de construction et de maintenance d'un ED de sources hétérogènes HDIM (Heterogeneous Data Integration and Maintenance). Nous nous intéressons dans ce papier à la problématique de la qualité des données dans ces ED et plus précisément au problème d'élimination des doublons et des données similaires (Deduplication, Match and Merge problems (Hernandez et Stolfo, 1998)). Les données similaires sont des données qui ont des ressemblances au niveau de leurs valeurs. Le problème de l'élimination des données similaires (doublons non strictes) et de la fusion/intégration de données est très com-

plexe. En effet, il s'agit de localiser, fusionner et enrichir des entités qui représentent le même monde réel.

Ce papier sera présenté comme suit : Dans la deuxième section nous faisons un parcours de différentes méthodes existantes de calcul de distance de similarité selon le type de données. Nous définissons, dans la section trois, les règles de concordances entre les tuples d'une table et la manière dont on fusionne et on enrichie les données. Le principe de fonctionnement des deux fonctions de base (Match & Merge) y est présenté. Nos trois algorithmes d'élimination des similaires ainsi que des mesures de performance seront présentés, dans la quatrième section. Nos travaux futurs sont donnés en guise de conclusion.

2 Calcul de distance de similarité

Plusieurs méthodes de calcul de distance de similarité ont été présentées dans la littérature selon les types des données à comparer. Nous présentons, ci-dessous, les méthodes les plus connues. Pour plus de détails, on pourra consulter par exemple (Berti-Équille, 2006), (Koudas et al., 2006). Les types de données peuvent être résumés en deux grandes catégories (simples et complexes) :

- Il existe deux types d'algorithmes de comparaison de données de types chaînes de caractères ou textes. Le choix est basé sur deux critères : "s'écrire comme" et "se prononcer comme". Nous présentons, dans ce qui suit, cinq méthodes pour le calcul de la distance de similarité de données. Les trois premières méthodes se basent sur le concept "s'écrire comme". Alors que les deux autres se basent sur le concept "se prononcer comme".
  - La distance de Levenshtein [1] est égale au nombre minimal de caractères qu'il faut supprimer, insérer, ou remplacer pour passer d'une chaîne à une autre.
  - La distance de Jaro-Winkler est adaptée au traitement de chaînes courtes comme des noms ou des mots de passe (Winkler, 2006).
  - La distance de Jaccard [2] mesure la diversité entre des chaînes longues.

  En testant ces méthodes, Jaro-Winkler semble mieux fonctionner pour les chaînes de caractère de différents types et de longueur variable.
  - L'algorithme Soundex [3] produit une clé phonétique d'un mot. Il fonctionne sur la consonance anglo-saxonne du langage.
  - L'algorithme Metaphone [4] est un algorithme pour indexer les mots selon leur sonorité lorsqu'ils sont prononcés en anglais. La version double tient compte de la sonorité des langues étrangères comme le français, le grec et le latin.

  L'algorithme Soundex produit de nombreux faux doublons, principalement parce qu'il est basé sur les premiers caractères de la chaîne pour générer le code soundex. Une autre limitation est que c'est un algorithme phonétique, de sorte que le résultat dépend de la langue. Le soundex est adapté à l'anglais.
- Les algorithmes de comparaison de données numériques et binaires permettent de calculer la distance de similarité entre des données binaires, tels que Jaccard, Indice de Sokal

1. www.levenshtein.net
2. http ://en.wikipedia.org/wiki/Jaccard_index
3. http ://fr.wikipedia.org/wiki/Soundex
4. http ://fr.wikipedia.org/wiki/Metaphone

& Michener, Indice de Rogers et Tanimoto et Hamming, et des données numériques, tels que la distance Euclidienne, la distance Manhattan, Bennani (2006).

- On peut trouver dans la littérature plusieurs algorithmes pour comparer des données de type complexe telles que les images, les sons et les textes. On peut citer, d'une part, ceux basés sur l'approche bayésienne, Bennani (2006) ou la distance de Hausdorff pour manipuler des images, et d'autre part, le double Metaphone, Soundex, Phonex, Similar Text, Cosinus similarité, SVM pour les autres types de données (Koudas et al., 2006).

Plusieurs travaux sur l'élimination des données similaires et des doublons existent dans la littérature (Benjalloun et al., 2009), (Cohen et Richman, 2004), (Bilenko et Mooney, 2003), (Hernandez et Stolfo, 1998). Les fonction de comparaison utilisent des règles de décision portant sur les n attributs. Il existe deux types de décision : (i) dichotomique (similaires ou non) -c'est notre cas ; ou (ii) trois réponses (similaires ou non et besoin de plus de révision) (Sarawagi, 2006). La fonction de comparaison peut êre basée, soit sur les valeurs d'attributs, soit sur le concept de contexte (Köpcke, 2006). Plusieurs approches existent pour fusionner différents tuples telles que les approches basées sur les règles ou sur les workflows ou encore les approches numériques. Concernant les approches basées sur les règles, la fusion se fait selon une combinaison logique des conditions de correspondance. Alors que celles basées sur les workflows permettent des combinaisons aléatoires (Köpcke et Rahm (2009), 2006).

3 Processus de comparaison et de fusion (Match and Merge)

L'élimination de données similaires nécessite le développement de deux fonctions "Match" (comparer) et "Merge" (fusionner) (Hernandez et Stolfo, 1998). La fonction *Match* permet de définir si deux tuples dans une table sont similaires et la fonction *Merge* permet d'en générer un nouveau en fusionnant les deux supposés similaires.

Introduisons à présent quelques notations. Soit T une table et C l'ensemble de ses attributs, on note $T(C)$. On considère que $C = A \cup B$. A est l'ensemble des attributs qui servent pour éliminer les tuples similaires dans la table T. B=C-A et $A \cap B$=$\emptyset$ (B peut être vide). Soit $[[D_k]]$ l'ensemble de valeurs concrètes d'un tuple de données tels que les chaînes de caractères alphanumériques (String), les numériques (Number), les dates, les bouléens (Boolean) ou encore des listes et des intervalles de valeurs. Une table T est interprétée par l'ensemble [[T]] de tous les $\{A_1 \ldots A_n, B_1 \ldots B_m\}$-tuples définis sur $[[D_k]]_{k=1,n+m}$. [[T]] est l'ensemble des fonctions avec $t : \{A_1 \ldots A_n, B_1 \ldots B_m\} \rightarrow \bigcup [[D_k]]_{k=1,n+m}$ tel que t(A_i) noté t.A_i (respect. t.B_j) est un élément de $[[D_i]]$ (respect. $[[D_j]]$). Chaque élément t de [[T]] est un tuple de la table T et chaque t.A_i est une valeur de l'attribut dans t, que l'on notera aussi v.

Définition 1 : Similarité entre valeurs (notée$\approx$) : *Deux valeurs v et v' sont similaires, on note ($v \approx v'$), ssi la distance de similarité d, calculée entre ces deux valeurs, vérifie une condition k. Pour deux tuples t et t', pour un attribut $A_{i(i=1,n)}$, t.$A_i \approx$t'.A_i ssi la condition k_i est vérifiée. La condition k_i se base sur le calcul de la distance d_i de similarité selon le type des données. Plusieurs cas de figures peuvent être envisagés. L'utilisateur peut ainsi fixer (donner) un ou plusieurs seuils. k_i : d_i est inférieur à un seuil maximal ; (d_i<s_i) avec $s_i \in$ [0..1].*
La similarité ($\approx$) vérifier évidemment les propriétés de réflexivité, commutativité et d'associativité. C'est-à-dire $[v \approx v]$, $[(v \approx v') \Leftrightarrow (v' \approx v)]$ et $[v \approx (v' \approx v'') \Leftrightarrow (v \approx v') \approx v'']$.

Exemple 1 : Adr1←t1.Adr et Adr2←t2.Adr ; Nom1←t1.Nom et Nom2←t2.Nom ; Mail1←t1.Mail et Mail2←t2.Mail ; Tel1←t1.Tel et Tel2←t2.Tel.
Les deux adresses suivantes Adr1="133 BD MARCEL EPINAY/SEINE" et Adr2="133 BD MARCEL 93800 EPINAY-SUR-SEINE" sont similaires, selon la méthode de Jaro-Winkler car la distance calculée est de 0,057 (inférieure au seuil s=0,2).

Définition 2 : Règle de similarité : *Une règle r de similarité est une conjonction de similarités qui portent sur des attributs $A_{i\,(i=1,n)}$ de l'ensemble A de la table T : r = (t.A_1≈t'.A_1) ∧ (t.A_2 ≈ t'.A_2) ∧ (t.A_i≈t'.A_i)...∧(t.A_n≈t'.A_n).*
Exemple 2 : Deux règles de similarités r_1 et r_2 : r_1 = (Nom1≈Nom2) ∧ (Mail1≈Mail2) ∧ (Adr1≈Adr2), r_2 = (Mail1≈Mail2) ∧ (Tel1≈Tel2).

Définition 3 : Similarité entre tuples : *Deux tuples t et t' sont similaires ssi la disjonction de toutes les règles de similarité définies sur la table T est vraie. On note t≈t' ssi $r_1 \vee r_2 \vee \ldots \vee r_k \ldots \vee r_q$ est vraie avec q étant le nombre de règles de similarité.*
Exemple 3 : t1 et t2 sont similaires ssi $r_1 \vee r_2$: t1≈t2 ssi ((Nom1≈Nom2) ∧ (Mail1≈Mail2) ∧ (Adr1≈Adr2)) ∨ ((Mail1≈Mail2) ∧ (Tel1≈Tel2)).
Remarque : Ces règles peuvent présenter toutefois des incohérences ou des contradictions qui seront à vérifier.

3.1 La fonction Match

La fonction Match compare deux tuples t1 et t2 : Match(t1,t2)=Vrai ssi t1≈t2. L'étude de similarité porte sur les attributs de l'ensemble A selon les règles de similarité. La fonction Similaire($v1$,$v2$,k) permet de dire si deux valeurs v1 et v2 sont similaires selon une condition k. En effet, et selon le type des données, une des méthodes de calcul de distance de similarité, vue ci-dessus, est appliquée. Par exemple, pour les chaînes de caractères un algorithme tel que celui de Levenshtein ou Jaro-Winkler sera déclenché. La distance trouvée d est comparée au seuil s donné (exemple de condition k : $d<s$). L'algorithme est présenté ci-dessous (tab 1).

3.2 La fonction Merge

La fonction Merge traite deux tuples, supposés similaires, tel que Match(t1,t2)=Vrai. Elle retourne un nouveau tuple qui est le résultat de la fusion des deux précédents. La fonction Merge retourne un tuple qui apporte "plus" d'information. Un enrichissement des données peut être réalisé à la demande de l'utilisateur sur les attributs de l'ensemble A, c'est-à-dire ceux qui servent pour étudier la concordance (match). Le principe général de l'algorithme de fusion de deux tuples est donné ci-dessous (tab 1). La fonction fusion permet de combiner deux valeurs selon le type de donnée de l'attribut en question : (i) si l'attribut est du type chaîne de caractères, $expr$ présente la plus longue chaîne des deux, si celle-ci vérifie un ensemble d'expressions régulières proposé pour l'utilisateur. La fusion d'une chaîne quelconque avec une valeur nulle (NULL) donne la chaîne elle-même ; (ii) si l'attribut est du type numérique, la fusion peut correspondre à des calculs demandés par l'utilisateur tels que la somme, la moyenne, le minimum ou le maximum des valeurs. Des transformations peuvent êtres réalisées telles que des conversions dues à des changements d'unité. Les valeurs nulles sont considérées égales à 0 ; (iii) si l'attribut est du type date, la fusion peut permettre de garder la plus récente des deux, selon le choix de l'utilisateur, après avoir unifié les formats et vérifié leurs validités.

Algorithme Match
Input : Two tuples, t1 and t2 ∈ T ;
S (thresholds)
Output : Result := True if t1 ≈ t2
Begin
Result := True
For all Rule rj j from 1 to q Do
$Rule_j$:= True ; i :=1
While $Rule_j$ and i ≤ n **Do**
v1 :=t1.Ai ; v2 :=t2.Ai
If v1 or v2 = NULL Then Res = True
Else Result = Similar (v1,v2, si) **End If**
$Rule_j$ = $Rule_j$ and Result ; i :=i+1
End While
Result := Result or $Rule_j$
End for
End Algorithm Match

Algorithm Merge
Input : Two tuples, t1 and t2 T
Output : A tuple t
Begin
For all Attribute Ai in A i from 1 to n **Do**
v1 :=t1.Ai ; v2 :=t2.Ai
v :=fusion (expr, v1, v2)
t.Ai :=v
End for
End Algorithm Merge

TAB. 1 – *Algorithmes de Match et Merge.*

4 Elimination des similaires

Le principe de l'élimination des données similaires consiste à croiser tous les tuples d'une table. Il s'agit donc d'algorithmes très coûteux pour de gros volumes de données. L'idée est de réduire le nombre de comparaisons en fusionnant les données qui concordent. Pour entamer cette étude, nous nous sommes inspirés des trois algorithmes présentés dans (Benjalloun et al., 2009). Dans ce qui suit, nous présentons et comparons ces trois algorithmes modifiés. Le principe de partitionnement des tables dans les SGBD est utilisé afin d'améliorer leurs performances, plusieurs versions sont donc disponibles.

4.1 Algorithme G

L'algorithme G est séquentiel (tab. 3). Il permet d'éliminer les similaires (ou quasi doublons) d'un ensemble de tuples grâce à la "boite noire" composée des deux fonctions *Match* et *Merge*. L'algorithme fonctionne à l'aide de deux ensembles de tuples I et I'. I est l'ensemble des tuples initiaux qui n'ont pas encore été comparés, et I' contient le résultat final. Tout tuple t de I est comparé à tous ceux de I'. Dans le cas où t correspond à t' de I', un nouveau tuple t"(résultat de la fusion) est créé et inséré dans I. Ensuite, le tuple dominée t' est supprimé de I'. Dans le cas inverse, t est ajouté à I'. De nombreuses versions de cet algorithme sont mises en oeuvre en utilisant le concept de partitionnement du SGBD Oracle. Les améliorations apportées à l'algorithme initial se résument comme suit : (i) l'élimination des tuples dominés se fait dès qu'ils sont trouvés et non à la fin du processus, puisque les tuples dominés ne peuvent pas générer des tuples non dominés ; (ii) l'ajout de la condition t ≠ t' pour le parcours des tuples de I' afin de satisfaire les propriétés d'ICAR (Idempotence, Commutativity, Associativity, Representativity) (Benjalloun et al., 2009) (celle de l'idempotence vu qu'il n'est pas nécessaire de faire correspondre t à lui-même).

4.2 Algorithme R

Afin d'améliorer les performances de la version G, l'algorithme R effectue les modifications suivantes. Lorsque t $\in$ I correspond à t' $\in$ I', t est supprimé de I (respect. t' de I'). t"(résultat de la fusion) est ajouté à I. Le nombre de comparaisons est alors réduit. Les mesures de performance que nous avons réalisées confirment cette amélioration (tab. 5). Plusieurs versions de cet algorithme (tab. 3) sont mises en oeuvre à l'aide d'Oracle.

4.3 Algorithme F

Afin de réduire encore le nombre de comparaisons, l'algorithme F (tab. 3), inspiré de (Benjalloun et al., 2009), se base sur la notion des attributs clés ("features") pour éliminer des similaires. Cet algorithme utilise, outre les ensembles I et I', deux ensembles de données P_f (positifs) et N_f (négatifs) pour garder trace des valeurs des attributs clés déjà rencontrées et enregistrer les comparaisons redondantes positives et aussi négatives.

P_f conserve toutes les valeurs précédemment rencontrées de l'attribut clé d'élimination et associe à chaque valeur l'identifiant de tuple auquel il appartient. Le tuple t est soit le premier enregistrement pour lequel la valeur v est apparue pour l'attribut clé f_i, ou celui qui a été dérivé à partir d'une séquence de fusions des tuples. Si ce tuple n'existe pas (la valeur v, de l'attribut clé, est vue pour la première fois), $P_{fi}(v)$ retourne Null. Cependant, il ne peut y avoir au plus qu'un tuple associé à la valeur v d'un attribut f_i. Dans le cas où plusieurs tuples sont présents, ces derniers sont fusionnés dans le tuple renvoyé par $P_{fi}(v)$.

N_f est l'ensemble qui conserve les valeurs des attributs clés f_i qui ont été comparées avec toutes les valeurs des attributs pour les tuples de I' et ne correspond à aucun d'entre eux. On évite alors des comparaisons redondantes dont la fonction Match aurait renvoyé une valeur négative. Si le tuple en cours de traitement par l'algorithme, a une valeur *v* qui apparaît dans N_{fi}, cette valeur n'a pas besoin d'être davantage comparée. Quand un nouveau tuple t est traité par l'algorithme, ce dernier sauvegarde, tout d'abord dans P_{fi} toutes les valeurs de chaque nouvel attribut, puis vérifie si l'une de ces valeurs est déjà parue dans un tuple différent (en vérifiant son identifiant). Les paires de valeurs des attributs clés P(f,v) qui correspondent, sont immédiatement dirigées vers le processus de fusion des tuples, et ne sont jamais comparées de nouveau, tandis que les paires de valeurs des attributs clés qui ne correspondent pas sont ajoutés à N_{fi}, et de cette façon on garantit qu'elles ne seront jamais à nouveau comparées.

Exemple 4 : Pour deux attributs Nom et Prénom, P_Nom et P_Prenom ainsi que N_Nom et N_Prenom les structures à gérer sont les suivantes (tab. 2). Plusieurs versions de cet algorithme

P						N		
Nom	RowIdN	NbrOccN	Prenom	RowIdP	NbrOccP	Nom	Prenom	RowIdT
Le Bon	t1	4	Adam	t1	3	Le Bon	Adam	t1
Le B.	t5	1	Adem	t5	1	Dupond	Jérémy	t2
Dupond	t2	3	Jérémy	t2	2			
Dupont	t8	1	Jeremy	t8	1			

TAB. 2 – *Echantillon des ensembles de données P et N*

ont été développées selon les structures de P et N. P et N peuvent être des listes gérées en

mémoire vive ou alors des tables gérées par un SGBD. Il est aussi possible d'utiliser une table par attribut clé ou une table pour tous les attributs à la fois. Le partitionnement des tables est alors appliqué pour gérer de gros volumes de données. Nous avons apporté des modifications à la version initiale dans (Benjalloun et al., 2009) afin de traiter différents cas de figure. Parmi ces modifications, lors de la fusion de deux tuples (t, t'), les valeurs des attributs de t', existent déjà dans les ensembles de données N_f, il faut les mettre à jour de la même manière que ceux de P_f.

L'implémentation des trois algorithmes G, R et F est faite en JAVA en utilisant des listes ou des tables sous le SGBD Oracle. Plusieurs versions existent pour gérer de gros volumes de données. L'algorithme F est le plus complexe dans son implémentation à cause des ensembles de données qu'il utilise (P_f et N_f). Les performances sont relatives au nombre de comparaisons effectuées dans la fonction Match. D'après (Benjalloun et al., 2009), R fait moins de comparaison que G. Par exemple, une fois que $t1$ est fusionné dans $t12$, il n'est plus nécessaire de comparer $t1$ à aucun autre tuple, ce qui évite de générer des tuples fusionnés intermédiaires. L'algorithme R ne génère jamais $t23$, par exemple, il fusionne $t3$ directement avec $t12$ pour produire $t123$. F est plus efficace que R car il effectue encore moins de comparaisons. Une comparaison des performances de ces algorithmes est présentée dans le paragraphe suivant.

4.4 Mesures des performances

Les mesures de performance sont effectuées sur les différentes versions des trois algorithmes. Les données générées aléatoirement en utilisant GenerateData [5]. Nous avons testé une table CLIENT dont la description SQL est la suivante :*"CREATE TABLE CLIENT (PRENOM varchar2(100), NOM varchar2(100), RUE varchar2(100), CODEPOSTAL varchar2(10), VILLE varchar2(50), DATNAIS Date, TEL varchar2(100)) ;"*. Les attributs Prenom, Nom, DatNais et Tel ont été choisis pour l'élimination des similaires. Les algorithmes Jaro-winkler, Levenshtein et Jaccard ont été utilisés pour le calcul de la distance de similarité. Un seuil a été fixé pour chacun des attributs respectivement (0,2 ; 0,2 ; 0 ; 0,1). L'utilisateur peut demander l'enrichissement de ses données afin d'éliminer, d'une part, d'éventuelles valeurs nulles, et d'autre part regrouper ou corriger certaines informations. Par exemple, dans la mesure où l'on veut avoir plusieurs numéros de téléphone pour un même client, la définition de l'attribut concerné (résultat de la fusion) devrait être suffisamment élargie pour mettre x valeurs au lieu d'une seule ($Telvarchar2(100)$ au lieu de $Telvarchar2(10)$). De ce fait, dans le cas où l'utilisateur demande à enrichir ses données, il lui suffit de fixer le nombre maximal x d'occurrences des valeurs pour les attributs concernés. La variable x peut être fixée par l'utilisateur selon une règle métier. Dans notre exemple, nous avons fixé x à 3 pour l'attribut Tel. Ainsi, on pourra avoir au plus trois numéros de téléphone pour chaque client après exécution de l'algorithme d'élimination des similaires. Deux règles de similarité entre tuples ont été utilisées pour réaliser la fusion (Merge) :
R_{ules} = $r_1 \vee r_2$ avec r_1 : (d1 $\leq$ s1) $\wedge$ (d2 $\leq$ s2) $\wedge$ (d7 $\leq$ s7) ; r_2 : (d2 $\leq$ s2) $\wedge$ (d6 $\leq$ s6)

Différentes mesures (tab. 4) ont été effectuées sur des tables allant de 10^3 à 10^6 lignes. Notre jeu de données contient 10%, 40% et enfin 80% de données similaires et de doublons. Les mesures (tab. 5) montrent que l'algorithme F est toujours plus performant que R et G sauf

5. http ://www.generatedata.com/#generator

Algorithme G
input : a set I of tuples
output : a set I' of tuples
I' $\rightarrow \emptyset$
while I $\neq \emptyset$ **do**
 t $\rightarrow$ a tuple from I
 remove t from I
 If I' $\neq \emptyset$ **then**
  for all tuples t' in I' and t $\neq$ t' **do**
   if Match (t, t') **then**
    t" $\rightarrow$ Merge (t, t')
    if t" $\notin$ I U I' U t **then**
    add t" to I
    endif
   remove t' from I'
   endif **endfor**
 enfif
 add t to I'
endwhile
return I'

Algorithme R
input : a set I of tuples
output : a set I' of tuples
I' $\rightarrow \emptyset$
while I $\neq$ **do**
 t $\rightarrow$ a tuple from I
 remove t from I
 buddy $\neq$ null
 If I' $\neq \emptyset$ **then**
  for all tuples t' in I' **do**
   if Match (currentTuple , t') **then**
   buddy $\rightarrow$ t' ; Exitfor **endif**
  endfor
 endif
 if buddy = null **then**
 add currentTuple to I'
 else
  t" $\rightarrow$ Merge (currentTuple, buddy)
 remove buddy from I'
 add t" to I
 endif
endwhile
return I'

Algorithme F
input : a set I of tuples,
output : a set I' of tuples
$P_f \rightarrow$ empty hash table, for each feature f ,
$N_f \rightarrow$ empty set, for each feature f
SaveTuples $\rightarrow$ a set of tuples, I' $\rightarrow \emptyset$,
currentTuple $\rightarrow$ null
while I $\neq \emptyset$ or currentTuple $\neq$ null
 if currentTuple = null **then**
  currentTuple $\rightarrow$ a Tuple from I
  remove currentTuple from I **endif**
 buddy $\rightarrow$ null
 for all(f, v) of currentTuple **do**
  if $P_f(v)$ = null **then** $P_f(v) \rightarrow$ currentTuple
  SaveTuples $\rightarrow$ (currentTuple, rowid) **endif**
 endfor
 for all (f, v) of currentTuple **do**
  if $P_f(v) \rightarrow$ currentTuple **then** /*we compare
the value and the rowid*/
   buddy $\rightarrow$ SaveTuples($P_f(v)$) ; **exitfor**
  endif
 endfor
 if buddy = null **then**
  for all (f, v) of currentTuple **do**
   if v $\notin N_f$ **then**
    for all t' of I' **do**
     if Match(t,t') **then** buddy$\rightarrow$ t' ; exitfor
     endif
    endfor
   if buddy $\rightarrow$ null **then** exitfor **endif**
   add v to N_f **endif**
  endfor
 endif
 if buddy = null **then** add currentTuple to I' ;
  currentTuple $\rightarrow$ null
 else t" $\rightarrow$ Merge(currentTuple, buddy)
  remove buddy from I'
  for all (f, v) where $P_f(v)$ currentTuple, buddy **do**
  $P_f(v) \rightarrow$ t" **endfor**
  for all (f, v) where $N_f(v)$ currentTuple, buddy **do**
  $N_f(v) \rightarrow$ t" **endfor**
  currentTuple $\rightarrow$ t" **endif**
endwhile
return I'

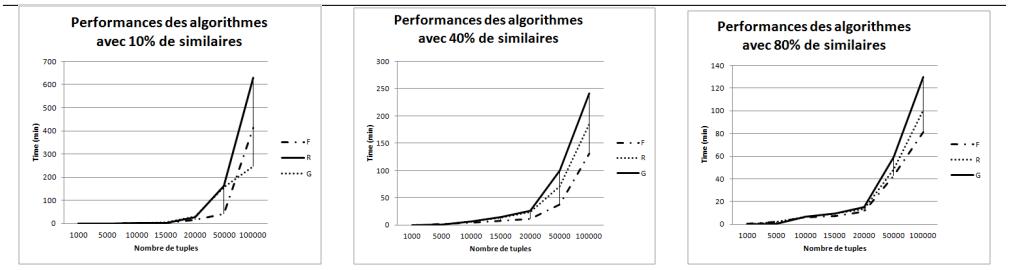
TAB. 3 – *Algorithmes d'élimination des similaires.*

le cas où le nombre de lignes est égal à 10^6 avec seulement 10% de similaires. Une étude plus détaillée est donc nécessaire pour en analyser les raisons. Signalons par ailleurs, que le taux

	10% Similaires			40% Similaires			80% Similaires		
M de lignes	20	50	100	20	50	100	20	50	100
F	17,18	43,00	414,20	11,32	37,45	130,42	11,34	43 ,00	81,49
R	28,01	161,35	630,09	24,22	71,35	185,43	13,45	48,19	100,14
G	29,53	157,45	248,36	26,03	100,20	241,06	15,36	59,32	130,05

TAB. 4 – *Mesures des performances pour les trois algorithmes (en secondes)*

d'éliminations n'est pas le même dans les trois algorithmes (tab. 6).



TAB. 5 – *Graphiques des performances des algorithmes*

	F			R			G		
Milliers de lignes	10	20	50	10	20	50	10	20	50
%des tuples éliminés	38,5	44,7	65,16	19,80	16,5	59 ,4	19,79	16,5	59 ,4
%des tuples similaires	19,79	16,5	59 ,4	19,79	16,5	59 ,4	19,79	16,5	59 ,4

TAB. 6 – *Taux d'élimination des similaires pour les trois algorithmes*

5 Conclusion

L'originalité de l'outil DQM (Data Quality Management), réside dans le fait de comparer et fusionner à la fois et de traiter l'élimination des données redondantes au sens strict et au sens large. Il traite des données de type caractère, numérique, date et logique. Le choix des règles de similarité sera automatisé. Les performances sont à améliorer pour les données très volumineuses. Il existe la stratégie de "Blocking" qui consiste à partitionner les données à comparer à l'aide d'une clé de bloc (ex $ville$). Mais cette méthode peut manquer des vrais doublons. Nous pouvons utiliser donc la stratégie qui permet de générer des clés de blocs orthogonales (i.e. qui partitionnent les données différemment) puis de faire plusieurs passes de comparaisons. Les taux d'élimination sont à étudier en détails. De nouvelles méthodes de calculs de distance de similarité pour des types de données complexes sont à développer ainsi

que l'automatisation du choix entre elles. Ceci permettrait de donner plus de confiances aux données rassemblées dans les ED. Tels sont les objectifs que nous nous sommes fixés dans nos travaux futurs. DQM est en cours de tests au Laboratoire LIPN de l'université Paris 13 et dans l'entreprise Talend, notre partenaire.

Références

Benjalloun, O., H. Garcia-Molina, D. Menestri, Q. Su, S. Whang, et J. Widom. Swoosh : A generic approach to entity resolution. *The International Journal on Very Large Data Bases (VLDB '09)*, 255–276.

Bennani, Y. (2006). *Traitement Numérique des Données*. France : Edition Hermes Science Publications.

Berti-Équille, L. Qualité des données. *Techniques de l'Ingénieur H3700-collection Technologies logicielles - Architecture des systèmes*, 1–19.

Bilenko, M. et R. Mooney. Adaptive duplicate detection using learnable string similarity measures. *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery, and Data Mining*, 39–48.

Cohen, W. et J. Richman. Iterative record linkage for cleaning and integration. *Proceedings of the 9th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery (DMKD'04)*, 11–18.

Hamdoun, S. et F. Boufarès. Un formalisme pour l'intégration de données hétérogènes. *6ème journées Francophones sur les entrepôts de données et l'Analyse en ligne (EDA'10)*, 107–119.

Hernandez, M. et S. Stolfo. Utility-based resolution of data. inconsistencies. *Proceedings of IQIS 2004, International Workshop on Information Quality in Information Systems, Maison de la Chimie*, 35–43.

Koudas, N., S. Sarawagi, et D. Srivastava. Record linkage : Similarity measures and algorithms. *Proceedings In International Conference on Management of Data, ACM SIGMOD'06*, 802–803.

Köpcke, H. et E. Rahm (2009). Frameworks for entity matching: A comparison. *Data Knowledge Engineering, DKE'09 68*, 197–210.

Winkler, W. E. Overview of record linkage and current research directions. *Research Report Series, RRS*, 1–28.

Summary

This paper handles the problem of similar data elimination (duplicates not strict) in a data warehouse. Indeed, the notion of data quality has a huge issue in a good data governance to improve the interactions between employees of one or more organizations. The presence of similar or duplicate data generates significant concerns about data quality. An overview of methods for calculating the similarity distance between data and similar data elimination algorithms are exposed and compared.