# Sujet du mémoire : Database Repairing with Respect to Functional Dependencies

## Description

Real-world databases are often inconsistent with respect to integrity constraints. The term database repairing refers to the process that takes in an inconsistent database, and returns a consistent database that is as similar as possible to the original database. Different similarity measures have been proposed in the literature, giving rise to different repair notions. The focus of this master thesis is on repairing relational databases with respect to functional dependencies, which are among the most common constraints in the relational model. In a recent article [1], the authors first formalize this problem and then study its complexity. The objective of this master thesis is to study the problem of database repairing with respect to functional dependencies, starting with [1], and to build a prototype to experimenitally validate theoretical approaches.

## References

[1] ~~Ester Livshits, Benny Kimelfeld, and Sudeepa Roy. “Computing Optimal Repairs for Functional Dependencies”. In: PODS. ACM, 2018, pp. 225–237.~~

# ~~Expérience personnelle :~~

## ~~Causes :~~

~~Mauvaise gestion au niveau applicative.~~

* ~~Plusieurs applications qui ne respectent pas toutes les mêmes règles.~~
* ~~Ajouts de tables (et donc de DF) mais oublis de gérer ces DF dans tous les applicatifs.~~

~~Insertion manuelle pour compenser une exception impossible à insérer par le biais de l’application avec soit une erreur volontaire (l’exception porte sur la DF) soit une erreur involontaire (l’exception ne porte pas sur le DF mais aucune sécurité ne signale la violation).~~

# Video & Lecture : Computing Optimal Repairs for Functional Dependencies

<https://www.youtube.com/watch?v=dXoLmFDbq7E>

~~Une base de données incohérente est une base de données qui ne respecte pas les contraintes d’intégrités.~~

~~Les contraintes d’intégrité sont par exemple des dépendances fonctionnelles (FDs).~~

~~La solution est une base de données à l’image de la base de données source après un minimum de changement et qui respecte les contraintes d’intégrités.~~

## ~~Causes :~~

~~Sources imprécises (internet, capteurs, …)~~

~~Erreurs d’encodage~~

~~Méthodes imprécises (reconnaissance vocale, analyse de signaux, d’images, …~~

~~Fusion de plusieurs bases de données avec~~

~~Des données différentes~~

~~Des formats différents~~

## ~~Solutions :~~

~~Nettoyer en supprimant les données erronées~~

~~Modifier les requêtes pour qu’elles ne renvoient que les données qui ne sont pas concernées par les lignes « inconsistante » (en erreur).~~

## ~~Motivations (de trouver cet optimal) :~~

~~Pour améliorer la qualité des données~~

~~Pour calculer la distance~~

## Modifications :

Locale : Aucun sous-ensemble strict des opérations n'atteint la cohérence.

Globale : aucun sous-ensemble plus petit (ou moins cher) ne permet d'atteindre la cohérence.

## Optimal S-repair : optimal subset repair

~~Version de la base de données obtenue par un minimum de suppression de lignes.~~

~~Algo qui ne réussit pas toujours. Pour le savoir, il est possible d’effectuer un algorithme de simplification sur les DF pour savoir si on va pouvoir vider les DF de Δ. Si oui, L’algo OptSRepair sera un succès.~~

### ~~Succès :~~

~~Polynomial.~~

~~Il peut supporter des contenus lourds et des doublons.~~

### ~~Echec :~~

~~NP difficile (et même APX-complet).~~

Avec des « denial constraint [29] »

## Optimal U-repair : optimal update repair

Version de la base de données obtenue par un minimum de mise-à-jour des valeurs.

Se base sur une dichotomie de l’algo précédent ??

## ICI ;

Modifications globales soit grâce à des suppressions soit grâce à des mise-à-jours.

## Scénario de recherche :

### Hypothèses :

#### ~~2.1 Schéma et table~~

~~D : base de données source incohérente~~

~~D’ : image de D qui est cohérente~~

~~T : table unique de notre base de données~~

~~R ou R (A1, ..., Ak) : schéma relationnel de la table T où chaque Ai est un attribut~~

~~Val : domaine infini des valeurs d’attribut~~

~~T[i] : une ligne(un tuple) de la table T~~

~~T[\*] : toutes les lignes de T~~

~~ids(T) : ?identifiant des lignes de la table T ?~~

~~Val~~~~k~~~~: valeur de la colonne k~~

~~w~~~~T~~~~(i) : poids de la ligne i de la table T~~

~~|T| = |ids(T)|: nombre de ligne dans T~~

~~t = (a1, . . . , ak ) : est une ligne de la table T~~

~~t.Aj : est la référence vers la valeur aj.~~

~~Chaque ligne à un identificateur qui ne sera jamais modifié pour pouvoir faire le lien entre la version D et D’ ainsi qu’un poids qui détermine combien ça coute de faire une modification ou de supprimer la line.~~

~~Les premières lettres de l’alphabet illustrent un seul attribut à la fois alors que les dernières lettres de l’alphabet représentent un ensemble d’attribut.~~

~~Une table peut être avec ou sans doublons et pondérée ou non (non pondérée : toutes les lignes ont le même poids)~~

#### ~~2.2 Dépendances fonctionnelles~~

~~Δ : ensemble des dépendances fonctionnelles~~

~~X -> Y : expression d’une DF où X et Y sont des séquences d’attributs.~~

~~Lhs/rhs : left/right hand side. On ne travaille qu’avec des dépendances fonctionnelles qui ont 1 attribut à gauche.~~

~~Δ |= X → Y : la DF X -> Y est ?impliquée? dans Δ. C-à-d que si une table vérifie Δ elle vérifie aussi X -> Y.~~

~~Cl(Δ) : la fermeture de Δ , regroupe toutes les DFs qui sont ?impliquée? dans Δ.~~

~~clΔ(X) : la fermeture d’un attribut (ou d’une séquence d‘attribut) X. C’est la liste de tous les attributs A tel que X -> A est impliquée dans Δ.~~

~~Δ1 = Δ2 : si leur fermeture est identique.~~

~~A common lhs : est un attribut « A » qui se trouve dans toutes les parties gauches des DF de Δ.~~

~~A chain : est une DF qui en inclus une autre (X1 inclus X2 ou inversement).~~

~~A consensus DF : est une DF dont la main gauche est vide : ∅ → Y. Cette DF est respectée si la valeur des attributs de Y est toujours identique.~~

S |= Δ : la table S satisfait les dépendances de « delta ».

#### 2.3 Réparations

distsub (S, T) : distance entre les deux tables S et T. C’est la somme des poids des lignes manquantes dans la table S pour obtenir la table T (sachant que S est un sous-ensemble de T.

distupd (U, T) : distance entre les deux tables U et T. C’est la somme des poids des attributs modifiés dans la table T pour obtenir la table U (sachant que U est un sous-ensemble de T.

H(u,t) : distance de Hamming. C’est le nombre d’attributs qui diffèrent entre une ligne de la table U et une ligne de la table T.

Un S-repair est un sous-ensemble S de T tel que S respecte Δ et qu’il n’existe pas d’ensemble S’ « plus grand » (on ne compare pas le nombre de ligne parce que supprimer 2 lignes avec un poid de 1 est mieux que de supprimer 1 ligne avec un poid de 3) qui respecte Δ.

Un U-repair est un sous-ensemble U de T tel que U respecte Δ et qu’il n’existe pas d’ensemble U’ « plus grand » qui respecte Δ.

Un S-repair optimal est un sous-ensemble S de T tel que distsub (S, T) est minimale.

Un U-repair optimal est un sous-ensemble U de T tel que distupd (U, T) est minimale.

Une α-approximation d’un S-repair optimal est un S-repair S’ dont la distance distsub (S’, T) n’est pas minimale. Elle est même : distsub (S, T) ≤ α. distsub (S’, T). On peut donc préciser que un S-repair optimal se defini par : 1-optimal S-repair.

(identique pour un U-repair)

N.B. : pour obtenir un U-repair, on n’utilise pas toujours des valeurs appartenant au domaine actif.

#### 2.4 Complexité

P : problème d’optimisation

x : entrée

y : une solution de l’espace des solutions de P pour l’entrée x.

cost(x, y) : coût nécessaire pour trouver y.

But : minimiser le coût. C’est-à-dire trouver la solution y pour laquelle cost(x, y) ≤ α. cost(x, y’), quel que soit y’.

### OptSRepair (Algorithm 1).

Les DF ont la forme : X → A

Δ – X = Δ’ : ensemble de DFs dont on a supprimé les attributs de X dans les DFs (dans les lhs et les rhs).

lhs mariage : une paire d’attribut (X1, X2) avec clΔ(X1) = clΔ(X2) (c’est-à-dire qu’il y a une réciprocité dans les DF : A -> B et B-> A) et dont toutes les lhs des DF de Δ contiennent soit X1 soit X2 :

def

ΔA↔B→C = {A → B , B → A, B →C}

wT(S) : somme des poids des lignes de S avec S un sous-ensemble de T.

π: projection

σ: selection

∪: union

#### 3.1 Algorithme :

Si l’algorithme arrive à supprimer toutes les dépendances fonctionnelles non triviales (les « common lhs », les « consensus » et les « lhs mariage ») alors il trouve un « optimal S-repair » en un temps polynomial.

N.B. : quand une DF est « Facility -> City ». Si Facility est un « common lhs » et qu’on supprime cet attribut, la DF devient : Ø -> City. L’attribut City devient alors un consensus.

##### CommonLHSRep

Avec A comme « common lhs »,

L’algorithme groupe les lignes par valeur de A et pour chaque sous-groupe, effectue un appel récursif à « OptSRepair » en supprimant la présence de A dans les DFs. Il retourne ensuite l’union des optimal S-repair.

Ici on renvoie l’union des sous-groupes.

##### ConsensusRep

Avec A comme « consensus »,

L’algorithme groupe les lignes par valeur de A et pour chaque sous-groupe, effectue un appel récursif à « ConsensusRep » en supprimant la présence de A dans les DFs. Il retourne ensuite le optimal S-repair avec le poids le plus grand.

Ici on renvoie un seul des sous-groupes (celui avec le poids le plus grand).

##### MarriageRep

Avec (X1, X2) comme « lhs mariage »,

L’algorithme doit trouver la correspondance pondérée (la plus lourde) d’un graphe bipartite. Ce graphe est défini par : G = (V1, V2, E, w). Avec :

V1 et V2 : des ensembles de nœuds disjoints.  
En [théorie des graphes](https://fr.wikipedia.org/wiki/Th%C3%A9orie_des_graphes), un graphe est dit **biparti** s'il existe une [partition](https://fr.wikipedia.org/wiki/Partition_(math%C3%A9matiques)) de son ensemble de sommets en deux sous-ensembles U {\displaystyle U} et V {\displaystyle V} telle que chaque arête ait une extrémité dans U {\displaystyle U} et l'autre dans V {\displaystyle V} .

E : un ensemble de bord qui connecte V1 à V2.

w : fonction de pondération qui attribue une pondération à chaque bord de E.

…

Il retourne ensuite l’union disjointe de l’optimal S-repair de Tv1, v2 pour (v1, v2) dans Emax.

##### Le théorème

Let Δ and T be a set of FDs and a table, respectively, over a relation schema R(A1, . . . ,Ak ). If OptSRepair(Δ,T ) succeeds,

then it returns an optimal S-repair. Moreover, OptSRepair(Δ,T ) terminates in p

olynomial time in k, |Δ|, and |T |.

#### 3.2 Dichotomie :

L’algorithme OSRSucceeds(Δ) étudie les DF et permet de savoir si l’algorithme OptSRepair(Δ,T ) sera une réussite.

##### Le théorème

Let Δ be a set of FDs.

• If OSRSucceeds(Δ) returns true, then an optimal S-repair can be computed in polynomial time by executing OptSRepair(Δ,T ) on the input T .

• If OSRSucceeds(Δ) returns false, then computing an optimal S-repair is \*not only NP-Hard but it is also\* APX-complete, and remains APX-complete on unweighted, duplicate-free tables.

Moreover, the execution of OSRSucceeds(Δ) terminates in polynomial time in |Δ|.

…..

### OptURepair

attr(∆) = liste de tous les attributs présents dans ∆.

#### Theorem 4.1.

Suppose that Δ = Δ1 ∪ Δ2 where Δ1 and Δ2 are attribute disjoint. The following are equivalent for all α ≥ 1.

(1) An α-optimal U-repair can be computed in polynomial time under Δ.

(2) An α-optimal U-repair can be computed in polynomial time under each of Δ1 and Δ2.

## Most probable database

À 18 : 00 minutes (voir video)