

Comparaisons de motifs

fonction diverses pour le rapprochement de données



MVP SQL Server

Expert langage SQL, SGBDR, modélisation de données Enseigne à l'ISEN Toulon et aux Arts & Métiers

par [Frédéric Brouard \(SQLpro\)](#)

Date de publication : Avril 2006

Dernière mise à jour :

La distance de Levenshtein et la différence de Hamming mesurent la similarité entre deux chaînes de symboles. Ces méthodes peuvent être utilisées notamment pour recoller des données dans le cadre de l'alimentation d'une base de données servant un outil décisionnel (dimension).

Cependant, l'une couteuse, comme l'autre, trop simpliste ne donnent pas toujours satisfaction. C'est pourquoi je vous présente une troisième fonction de mon invention qui semble donner de très bons résultats. Je l'ai appelée **inférence basique...**

- I - Préambule
- II - Distance de Levenshtein
- III - L'Implémentation du Levenshtein "standard"
- IV - Un Levenshtein "limité"
- V - Différence de Hamming
- VI - Inférence basique
- VII - Comparaison des temps de calcul
- VIII - De plus amples informations vous sont nécessaires ?

I - Préambule

Les méthodes de rapprochement de données que nous allons voir se basent sur des fonctions qui mesurent le rapprochement ou l'éloignement de deux chaînes de caractères.

Ces méthodes sont destinées à trouver la bonne valeur dans une table référençant des valeurs exactes à partir de données incorrectes ou imprécises. Elles s'avèrent souvent utile à la construction de certaines solutions d'informatique décisionnelle afin de qualifier les données lors de la phase de migration (cadre de l'ETL, ou ELT).

II - Distance de Levenshtein

La distance de Levenshtein doit son nom à Vladimir Levenshtein qui l'a défini en 1965.

On appelle distance de Levenshtein entre deux mots *source* et *cible* le coût minimal pour aller de *source* à *cible* en effectuant des opérations élémentaires telles que:

- substitution d'un caractère;
- ajout dans d'un caractère;
- suppression d'un caractère;

En associant à chacune des opérations un poids on trouve le coût par sommation des poids des différentes opérations.

La distance de Levenshtein opère généralement sur des données de type chaîne de caractères, mais on peut l'employer sur toutes chaînes de symboles et en particulier sur des données binaires.

L'algorithme présenté, d'une forte complexité ($m \times n$ opérations), peut être écrit de manière simplifiée à l'aide d'une matrice que nous élaborerons à l'aide d'une table. Cela n'en reste pas moins un algorithme couteux...

C'est pourquoi nous allons voir une première implémentation basique de cette fonction, ainsi qu'une implémentation particulière limitant la profondeur de calcul afin de donner des résultats plus rapidement.

Notons d'ores et déjà qu'il n'y a pas une seule implémentation de la distance de Levenshtein, mais toute une famille d'algorithmes qui considèrent ou non telle ou telle opération et qui valent les différents poids en fonction du problème à traiter.

Par exemple en SQL on peut considérer que :

- Si le type de données est CHAR et que la substitution est évaluée à 1 alors l'ajout comme la suppression d'un caractère peut être évalué à 3 du fait du décalage nécessaire dans la chaîne.
- En revanche, si le type de données est VARCHAR et que la substitution est évaluée à 1 alors l'ajout peut être évalué à 9 du fait de la nécessité d'agrandir l'espace de stockage pour ce faire, mais pour la suppression cette évaluation peut rester à 4 car il n'est pas nécessaire de réajuster l'espace de stockage, mais juste de modifier la valeur du paramètre qui spécifie la longueur réelle de la données stockée.

Bref tout dépend du point de vu dans lequel on se place.

III - L'Implémentation du Levenshtein "standard"

Dans nos exemples nous avons considéré que toutes les opérations de l'algorithme de Levenshtein devaient être valuées à 1.

Par exemple pour aller du mot DEPORTA au mot PORTEURS, il nous faut 6 opérations unitaires :

1	suppression du D	EPORTA
2	suppression du E	PORTA
3	modification du A en E	PORTE
4	ajout du U	PORTEU
5	ajout du R	PORTEUR
6	ajout du S	PORTEURS

Voici une première implémentation de l'algorithme en Transact SQL :

```

CREATE FUNCTION F_DISTANCE_LEVENSHTTEIN (@SOURCE VARCHAR(8000),
                                          @CIBLE VARCHAR(8000))
RETURNS INT AS
BEGIN
    -- gestion des effets de bord
    -- null en entrée
    IF @SOURCE IS NULL OR @CIBLE IS NULL
        RETURN NULL

    -- identique
    IF @SOURCE = @CIBLE
        RETURN 0

    DECLARE @LN_SOURCE INT
    DECLARE @LN_CIBLE INT

    SET @LN_SOURCE = LEN(@SOURCE) + 1
    SET @LN_CIBLE = LEN(@CIBLE) + 1

    -- chaine vide avec chaine pleine
    IF @LN_SOURCE = 1 OR @LN_CIBLE = 1
    BEGIN
        RETURN @LN_SOURCE + @LN_CIBLE - 2
    END

    DECLARE @MATRICE TABLE(ID Int, Valeur INT)
    DECLARE @i INT
    DECLARE @j INT
    DECLARE @tmp Int
    DECLARE @v1 Int
    DECLARE @v2 Int
    DECLARE @v3 Int
    DECLARE @Vmin Int
    DECLARE @COUT INT

    /* Initialisation de la MATRICE */
    SET @i = 0
    WHILE (@i < @LN_SOURCE * @LN_CIBLE)
    BEGIN
        INSERT INTO @MATRICE VALUES (@i, 0)
        SET @i = @i + 1
    END

```

```
/* Initialisation de la premiere ligne */
SET @i = 0
WHILE (@i < @LN_SOURCE)
BEGIN
    SET @tmp = 0
    UPDATE @MATRICE SET Valeur = @i
        WHERE ID = @tmp
    SET @i = @i + 1
END

/* initialisation de la premiere colonne */
SET @i = 0
WHILE @i < @LN_CIBLE
BEGIN
    SET @tmp = @i * @LN_SOURCE
    UPDATE @MATRICE SET Valeur = @i
        WHERE ID = @tmp
    SET @i = @i + 1
END

/* On compare les deux chaines */

SET @i = 1

WHILE @i < @LN_SOURCE /* Colonne */
BEGIN
    SET @j = 1
    WHILE @j < @LN_CIBLE /* Ligne */
    BEGIN

        /* On regarde si les caractères correspondent */
        IF SUBSTRING(@SOURCE, @i, 1) = SUBSTRING(@CIBLE, @j, 1)
            /* EGALITE */
            SET @COUT = 0
        ELSE
            /* SUBSTITUTION, INSERTION */
            SET @COUT = 1

        /* Lecture de la case de gauche */
        SELECT @v1 = Valeur+1
        FROM @MATRICE
        WHERE ID = @j * @LN_SOURCE + @i - 1

        /* Lecture de la case d au dessus */
        SELECT @v2 = Valeur+1
        FROM @MATRICE
        WHERE ID = (@j-1) * @LN_SOURCE + @i

        /* Lecture de la case de diagonale gauche haute */
        SELECT @v3 = @COUT+ Valeur
        FROM @MATRICE
        WHERE ID = (@j-1) * @LN_SOURCE + @i-1

        /* On prend la valeur la plus petite et on l'insere dans la case actuelle */
        IF @V1 <= @V2 AND @V1 <= @V3
            SET @Vmin =@V1
        ELSE
            IF @V2 <= @V3
                SET @Vmin = @V2
            ELSE
                SET @Vmin = @V3

        SET @tmp = @j * @LN_SOURCE + @i
        UPDATE @MATRICE
        SET Valeur = @Vmin
        WHERE ID = @tmp
    END
END
```

```
        SET @j = @j +1
    END

    SET @i = @i +1
END

/* On retourne le cout de la transformation */
SELECT @tmp = Valeur
FROM   @MATRICE
WHERE  ID = (@LN_CIBLE-1) * @LN_SOURCE + @LN_SOURCE-1

RETURN @tmp

END

GO
```

Pour en tester les effets, voici le jeu d'essais que je vous propose :

Soit une table de vêtements contenant une colonne couleur et la table des couleurs cible comportant les couleurs de référence. Nous allons utiliser la fonction pour tenter de trouver la valeur la couleur de référence de chaque vêtement, la plus proche de celles saisies par des opérateurs insouciantes...

```
CREATE TABLE T_VETEMENT_VTM
(VTM_DESCRIPTION VARCHAR(16),
 VTM_COULEUR     VARCHAR(32))
GO

INSERT INTO T_VETEMENT_VTM VALUES ('Pantalon1', 'Rouge')
INSERT INTO T_VETEMENT_VTM VALUES ('Robe2', 'Vert d'eau')
INSERT INTO T_VETEMENT_VTM VALUES ('Robe3', 'Viol.')
INSERT INTO T_VETEMENT_VTM VALUES ('Pull4', 'Jaune paille')
INSERT INTO T_VETEMENT_VTM VALUES ('Robe5', 'Verte')
INSERT INTO T_VETEMENT_VTM VALUES ('Pantalon6', 'Gris bleu')
INSERT INTO T_VETEMENT_VTM VALUES ('Pantalon7', 'Vert')
INSERT INTO T_VETEMENT_VTM VALUES ('Robe8', 'Rouge')
INSERT INTO T_VETEMENT_VTM VALUES ('Pull9', 'blanc')
INSERT INTO T_VETEMENT_VTM VALUES ('Robel10', 'Blanche')
INSERT INTO T_VETEMENT_VTM VALUES ('Pantalon11', 'Gris souris')
INSERT INTO T_VETEMENT_VTM VALUES ('Robel12', 'rose')
INSERT INTO T_VETEMENT_VTM VALUES ('Pull13', 'bleue')
INSERT INTO T_VETEMENT_VTM VALUES ('Pantalon14', 'Rouge brique')
INSERT INTO T_VETEMENT_VTM VALUES ('Pull15', 'Marron')
INSERT INTO T_VETEMENT_VTM VALUES ('Chemisel16', 'Vrt')
INSERT INTO T_VETEMENT_VTM VALUES ('Chemisel17', 'Blenc')
GO

CREATE TABLE T_REF_COULEUR_RCL
(RCL_COULEUR VARCHAR(16))
GO

INSERT INTO T_REF_COULEUR_RCL VALUES ('Noir')
INSERT INTO T_REF_COULEUR_RCL VALUES ('Blanc')
INSERT INTO T_REF_COULEUR_RCL VALUES ('Vert')
INSERT INTO T_REF_COULEUR_RCL VALUES ('Jaune')
INSERT INTO T_REF_COULEUR_RCL VALUES ('Bleu')
INSERT INTO T_REF_COULEUR_RCL VALUES ('Rouge')
INSERT INTO T_REF_COULEUR_RCL VALUES ('Rose')
INSERT INTO T_REF_COULEUR_RCL VALUES ('Violet')
INSERT INTO T_REF_COULEUR_RCL VALUES ('Noir')
INSERT INTO T_REF_COULEUR_RCL VALUES ('Gris')
INSERT INTO T_REF_COULEUR_RCL VALUES ('Marron')
GO
```

La requête suivante va nous donner une approche de la solution :

```

SELECT VTM_DESCRIPTION, VTM_COULEUR, RCL_COULEUR,
       dbo.F_DISTANCE_LEVENSHTTEIN(VTM_COULEUR, RCL_COULEUR) AS DL
FROM   T_VETEMENT_VTM V
       CROSS JOIN T_REF_COULEUR_RCL C
WHERE  dbo.F_DISTANCE_LEVENSHTTEIN(VTM_COULEUR, RCL_COULEUR)
       = (SELECT MIN(dbo.F_DISTANCE_LEVENSHTTEIN(VTM_COULEUR, RCL_COULEUR))
          FROM   T_VETEMENT_VTM
               CROSS JOIN T_REF_COULEUR_RCL
               WHERE VTM_DESCRIPTION = V.VTM_DESCRIPTION)
ORDER  BY 1, 4 ASC
    
```

VTM_DESCRIPTION	VTM_COULEUR	RCL_COULEUR	DL
Chemise16	Vrt	Vert	1
Chemise17	Blenc	Blanc	1
Pantalon1	Rouge	Rouge	0
Pantalon11	Gris souris	Gris	1
Pantalon14	Rouge brique	Vert	3
Pantalon14	Rouge brique	Jaune	3
Pantalon14	Rouge brique	Bleu	3
Pantalon14	Rouge brique	Rouge	3
Pantalon14	Rouge brique	Rose	3
Pantalon6	Gris bleu	Bleu	0
Pantalon7	Vert	Vert	0
Pull13	bleue	Bleu	1
Pull15	Marron	Marron	0
Pull4	Jaune paille	Bleu	2
Pull9	blanc	Blanc	0
Robe10	Blanche	Blanc	2
Robe12	rose	Rose	0
Robe2	Vert d'eau	Vert	3
Robe2	Vert d'eau	Jaune	3
Robe2	Vert d'eau	Bleu	3
Robe3	Viol.	Violet	2
Robe5	Verte	Vert	1
Robe8	Rouge	Rouge	0

Mais le gros inconvénient de cette méthode, c'est qu'avec si peu de données, la combinatoire est telle que la masse des calculs nécessite déjà 3 secondes de calculs sur un PC haut de gamme. C'est bien évidemment inexploitable en production avec de grosses masses d'information.

IV - Un Levenshtein "limité"

L'idée est alors de limiter la profondeur de scrutation. En observant les résultats, nous nous apercevons que les bonnes approches l'ont été avec une distance de Levenshtein comprise entre 0 et 2. À trois nous avons déjà le pantalon14 et la robe2 qui nous offre trop de correspondances.

Nous pouvons introduire dans le code cette limitation en modifiant l'algorithme comme suit :

```
CREATE FUNCTION F_DISTANCE_LEVENSHTTEIN_LIMITE (@SOURCE VARCHAR(8000),
                                                @CIBLE VARCHAR(8000),
                                                @LIMITE INT)
RETURNS INT AS
BEGIN
-- gestion des effets de bord
-- null en entrée
IF @SOURCE IS NULL OR @CIBLE IS NULL OR @LIMITE IS NULL
    RETURN NULL

-- limite de 0 et différent !
IF @LIMITE = 0 AND @SOURCE <> @CIBLE RETURN NULL

-- limite inférieure à zéro :
IF @LIMITE < 0 RETURN NULL

-- identique
IF @SOURCE = @CIBLE
    RETURN 0

DECLARE @LN_SOURCE INT
DECLARE @LN_CIBLE INT

SET @LN_SOURCE = LEN(@SOURCE) + 1
SET @LN_CIBLE = LEN(@CIBLE) + 1

-- chaine vide avec chaine pleine
IF @LN_SOURCE= 1 OR @LN_CIBLE = 1
BEGIN
    RETURN @LN_SOURCE + @LN_CIBLE - 2
END

DECLARE @MATRICE TABLE(ID Int, Valeur INT)
DECLARE @i INT
DECLARE @j INT
DECLARE @tmp Int
DECLARE @v1 Int
DECLARE @v2 Int
DECLARE @v3 Int
DECLARE @Vmin Int
DECLARE @COUT INT
```

```
/* Initialisation de la MATRICE */
SET @i = 0
WHILE (@i < @LN_SOURCE*@LN_CIBLE)
BEGIN
    INSERT INTO @MATRICE VALUES (@i, 0)
    SET @i = @i + 1
END

/* Initialisation de la premiere ligne */
```

```
SET @i = 0
WHILE (@i < @LN_SOURCE)
BEGIN
    SET @tmp = 0
    UPDATE @MATRICE SET Valeur = @i
        WHERE ID = @tmp
    SET @i = @i + 1
END

/* initialisation de la premiere colonne */
SET @i = 0
WHILE @i < @LN_CIBLE
BEGIN
    SET @tmp = @i * @LN_SOURCE
    UPDATE @MATRICE SET Valeur = @i
        WHERE ID = @tmp
    SET @i = @i + 1
END

/* On compare les deux chaines */

SET @i = 1

WHILE @i < @LN_SOURCE /* Colonne */
BEGIN
    SET @j = 1
    WHILE @j < @LN_CIBLE /* Ligne */
    BEGIN

        /* On regarde si les caractères correspondent */
        IF SUBSTRING(@SOURCE, @i, 1) = SUBSTRING(@CIBLE, @j, 1)
            /* EGALITE */
            SET @COUT = 0
        ELSE
            /* SUBSTITUTION, INSERTION */
            SET @COUT = 1

        /* Lecture de la case de gauche */
        SELECT @v1 = Valeur+1
        FROM @MATRICE
        WHERE ID = @j * @LN_SOURCE + @i - 1

        /* Lecture de la case d au dessus */
        SELECT @v2 = Valeur+1
        FROM @MATRICE
        WHERE ID = (@j-1) * @LN_SOURCE + @i

        /* Lecture de la case de diagonale gauche haute */
        SELECT @v3 = @COUT+ Valeur
        FROM @MATRICE
        WHERE ID = (@j-1) * @LN_SOURCE + @i-1

        /* On prend la valeur la plus petite et on l'insere dans la case actuelle */
        IF @V1 <= @V2 AND @V1 <= @V3
            SET @Vmin = @V1
        ELSE
            IF @V2 <= @V3
                SET @Vmin = @V2
            ELSE
                SET @Vmin = @V3

        SET @tmp = @j * @LN_SOURCE + @i
        UPDATE @MATRICE
        SET Valeur = @Vmin
        WHERE ID = @tmp

    END
END

-- si la limite est dépassée, on retourne NULL !
IF EXISTS(SELECT *
```

```

        FROM @MATRICE
        WHERE ID = (@LN_CIBLE-1) * @LN_SOURCE + @LN_SOURCE-1
              AND Valeur > @LIMITE)
    RETURN NULL

    SET @j = @j +1
END

    SET @i = @i +1
END

/* On retourne le cout de la transformation */
SELECT @tmp = Valeur
FROM @MATRICE
WHERE ID = (@LN_CIBLE-1) * @LN_SOURCE + @LN_SOURCE-1

RETURN @tmp

END
    
```

Dans notre cas, en utilisant une limite de 2, nous obtenons une réponse déjà plus consistante :

```

-- les meilleurs approches avec une limite de 2
SELECT VTM_DESCRIPTION, VTM_COULEUR, RCL_COULEUR,
dbo.F_DISTANCE_LEVENSHTTEIN(VTM_COULEUR, RCL_COULEUR) AS DL
FROM T_VETEMENT_VTM V
CROSS JOIN T_REF_COULEUR_RCL C
WHERE dbo.F_DISTANCE_LEVENSHTTEIN_LIMITE(VTM_COULEUR, RCL_COULEUR, 2)
= (SELECT MIN(dbo.F_DISTANCE_LEVENSHTTEIN_LIMITE(VTM_COULEUR,
RCL_COULEUR, 2))
FROM T_VETEMENT_VTM
CROSS JOIN T_REF_COULEUR_RCL
WHERE VTM_DESCRIPTION = V.VTM_DESCRIPTION)
ORDER BY 1, 4 ASC
    
```

VTM_DESCRIPTION	VTM_COULEUR	RCL_COULEUR	DL
Chemise16	Vrt	Vert	1
Chemise17	Blenc	Blanc	1
Pantalon1	Rouge	Rouge	0
Pantalon11	Gris souris	Gris	1
Pantalon6	Gris bleu	Bleu	0
Pantalon7	Vert	Vert	0
Pull13	bleue	Bleu	1
Pull15	Marron	Marron	0
Pull4	Jaune paille	Bleu	2
Pull9	blanc	Blanc	0
Robe10	Blanche	Blanc	2
Robe12	rose	Rose	0
Robe3	Viol.	Violet	2
Robe5	Verte	Vert	1
Robe8	Rouge	Rouge	0

Vous noterez cependant qu'il existe toujours des dissemblances. Par exemple le Pantalon6 gris bleu a été trouvé plutôt bleu que gris.... Quand au Pull4 il a été trouvé bleu alors qu'il est jaune paille. En fait la distance de Levenshtein se révèle très peu crédible quand les occurrences ont des longueurs très différentes.

Il y a divers moyen de corriger cela.

- D'abord en valuant fortement les insertions et suppressions et faiblement les substitutions.
- Ensuite en prenant en compte la différence de longueur et en pondérant le résultat du Levenshtein.
- Enfin en utilisant d'autres mesures telles que la différence de HAMMING

Je vous laisse travailler à modifier savamment l'algorithme de ce Levenshtein. Quand à moi, je vais vous présenter la différence de HAMMING...

V - Différence de Hamming

C'est tout simplement le nombre de symboles différents d'une chaîne à l'autre... Il suffit de lire séquentiellement les deux chaînes en comparant chaque lettre à la position n dans les deux mots. Si la lettre est identique on compte 1.

La différence de Hamming peut être codée comme suit en Transact SQL :

```
CREATE FUNCTION F_DIFFERENCE_HAMMING (@SOURCE VARCHAR(8000),
                                      @CIBLE VARCHAR(8000))
    RETURNS INT
AS
BEGIN

    IF @SOURCE IS NULL OR @CIBLE IS NULL RETURN NULL

    IF @SOURCE = '' OR @CIBLE = '' RETURN LEN(@SOURCE) + LEN(@CIBLE)

    DECLARE @COUNT INT
    DECLARE @I INT, @L INT

    SET @I = 1
    SET @L = LEN(@SOURCE)
    IF LEN(@CIBLE) > @L
        SET @L = LEN(@CIBLE)

    SET @COUNT = 0

    WHILE @I <= @L
    BEGIN
        IF @I > LEN(@SOURCE)
            BEGIN
                SET @COUNT = @COUNT + LEN(@CIBLE) - LEN(@SOURCE) + 1
                BREAK
            END
        IF @I > LEN(@CIBLE)
            BEGIN
                SET @COUNT = @COUNT + LEN(@SOURCE) - LEN(@CIBLE) + 1
                BREAK
            END
        IF SUBSTRING(@SOURCE, @I, 1) <> SUBSTRING(@CIBLE, @I, 1)
            SET @COUNT = @COUNT + 1

        SET @I = @I + 1
    END

    RETURN @COUNT

END
GO
```

Cet algorithme est peu coûteux car linéaire et donne quelques bons résultats :

```
-- les meilleurs approches par Hamming
SELECT VTM_DESCRIPTION, VTM_COULEUR, RCL_COULEUR,
       dbo.F_DIFFERENCE_HAMMING(VTM_COULEUR, RCL_COULEUR) AS DL
FROM   T_VETEMENT_VTM V
       CROSS JOIN T_REF_COULEUR_RCL C
WHERE  dbo.F_DIFFERENCE_HAMMING(VTM_COULEUR, RCL_COULEUR)
       = (SELECT MIN(dbo.F_DIFFERENCE_HAMMING(VTM_COULEUR, RCL_COULEUR))
          FROM   T_VETEMENT_VTM
               CROSS JOIN T_REF_COULEUR_RCL
```

```

WHERE VTM_DESCRIPTION = V.VTM_DESCRIPTION)
ORDER BY 1, 4 ASC

```

VTM_DESCRIPTION	VTM_COULEUR	RCL_COULEUR	DL
Chemise16	Vrt	Vert	4
Chemise16	Vrt	Gris	4
Chemise17	Blenc	Blanc	1
Pantalon1	Rouge	Rouge	0
Pantalon11	Gris souris	Gris	8
Pantalon14	Rouge brique	Rouge	8
Pantalon6	Gris bleu	Gris	6
Pantalon7	Vert	Vert	0
Pull13	bleue	Bleu	2
Pull15	Marron	Marron	0
Pull4	Jaune paille	Jaune	8
Pull9	blanc	Blanc	0
Robe10	Blanche	Blanc	3
Robe12	rose	Rose	0
Robe2	Vert d'eau	Vert	7
Robe3	Viol.	Violet	3
Robe5	Verte	Vert	2
Robe8	Rouge	Rouge	0

Mais notre Chemise16 est vue verte et gris tandis que la Pantalon6 est vu spécifiquement gris...

Pouvons-nous encore améliorer notre score ?

J'ai pensé à travailler sur une approche plus directe, notamment en passant, non pas par la différence, mais par le rapprochement direct des lettres. J'ai appelé cet algorithme INFERENCE BASIQUE.

VI - Inférence basique

Algorithme de Frédéric Brouard.

Il s'agit ni plus ni moins de comparer les lettres d'une chaîne à une autre en avançant toujours dans le même sens :

- on avance lettre par lettre dans le mot cible;
- on se positionne à la première lettre dans le mot source;
- si une lettre est trouvée dans le mot source on compte 1 et on reste positionné à cette lettre dans le mot source;
- la comparaison n'étant pas commutative, on recommence en inversant cible et source;
- on renvoi alors le meilleur des deux comptages.

C'est un algorithme moins coûteux que Levenshtein, mais un peu plus que Hamming.

Il peut d'ailleurs être optimisé par le fait que la scrutation inverse n'est pas nécessaire si le comptage renvoie 0 à la première passe (aucune lettre commune).

Exemple :

Soit à comparer par inférence basique les mots ANNANAS et BANANE:

Première passe :

```
B A N A N E
A N N A N A S
```

Cette première passe donne 3 symboles communs

Seconde passe :

```
A N N A N A S
B A N A N E
```

Elle donne 4 symboles communs.

L'inférence directe donne donc 4.

Voici le code de cette fonction :

```
CREATE FUNCTION F_INFERENCE_BASIQUE (@SOURCE VARCHAR(8000),
                                     @CIBLE VARCHAR(8000))
RETURNS INT
AS
BEGIN

IF @SOURCE IS NULL OR @CIBLE IS NULL RETURN NULL
```

```

IF @SOURCE = '' OR @CIBLE = '' RETURN 0

DECLARE @COUNT1 INT, @COUNT2 INT
DECLARE @I INT, @J INT, @L INT
DECLARE @C CHAR(1)

SET @COUNT1 = 0
SET @COUNT2 = 0

SET @I = 1
SET @J = 1
SET @L = LEN(@SOURCE)

-- première passe source vers cible
WHILE @I <= @L
BEGIN
    SET @C = SUBSTRING(@SOURCE, @I, 1)
    IF CHARINDEX(@C, @CIBLE, @J) > 0
    BEGIN
        SET @COUNT1 = @COUNT1 + 1
        SET @J = CHARINDEX(@C, @CIBLE, @J) + 1
        SET @I = @I + 1
        CONTINUE
    END
    SET @I = @I + 1
END

IF @COUNT1 = 0 RETURN 0

SET @I = 1
SET @J = 1
SET @L = LEN(@CIBLE)

-- seconde passe cible vers source
WHILE @I <= @L
BEGIN
    SET @C = SUBSTRING(@CIBLE, @I, 1)
    IF CHARINDEX(@C, @SOURCE, @J) > 0
    BEGIN
        SET @COUNT2 = @COUNT2 + 1
        SET @J = CHARINDEX(@C, @SOURCE, @J) + 1
        SET @I = @I + 1
        CONTINUE
    END
    SET @I = @I + 1
END

-- le meilleur comptage
IF @COUNT1 < @COUNT2
    SET @COUNT1 = @COUNT2

RETURN @COUNT1

END

GO
    
```

Sur notre exemple, les résultats sont éloquentes :

VTM_DESCRIPTION	VTM_COULEUR	RCL_COULEUR	IB
Chemise16	Vrt	Vert	3
Chemise17	Blanc	Blanc	4
Pantalon1	Rouge	Rouge	5
Pantalon11	Gris souris	Gris	4

Pantalon14	Rouge brique	Rouge	5
Pantalon6	Gris bleu	Bleu	4
Pantalon6	Gris bleu	Gris	4
Pantalon7	Vert	Vert	4
Pull13	bleue	Bleu	4
Pull15	Marron	Marron	6
Pull4	Jaune paille	Jaune	5
Pull9	blanc	Blanc	5
Robe10	Blanche	Blanc	5
Robe12	rose	Rose	4
Robe2	Vert d'eau	Vert	4
Robe3	Viol.	Violet	4
Robe5	Verte	Vert	4
Robe8	Rouge	Rouge	5

La seule occurrence à ne pas avoir été tranchée est le Pantalon6 gris bleu ! C'est effectivement la seule occurrence indécidable par le simple fait d'une machine...

VII - Comparaison des temps de calcul

Les temps de calcul de ces différents algorithmes pour notre jeu d'essai s'établissent comme suit (les requêtes ont été lancées sans la clause ORDER BY) :

méthode	Temps UC (ms)	Temps coulé (ms)
Levenshtein	10 375	10 959
Levenshtein limité à 2	11 562	11 984
Hamming	407	447
Inférence basique	797	830

On s'aperçoit finalement que le Levenshtein limité ne donne pas de résultats franchement probant et que le test de limitation s'avère finalement plus coûteux que sa version "libre".

On constate que le coût de l'inférence basique est moins du double de celui de la différence de Hamming. Mais compte tenu de sa meilleure performance en matière de rapprochement de motifs il s'avère payant dans bien des cas...

VIII - De plus amples informations vous sont nécessaires ?

Venez en discuter sur le forum public français de Microsoft SQL Server :

news.msnews.microsoft.com/microsoft.public.fr.sqlserver

Ou sur le forum SQL Server de developpez.com :

<http://www.developpez.net/forums/forumdisplay.php?f=49>

