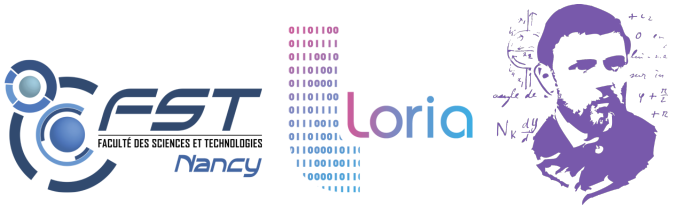
UNIVERSITÉ DE LORRAINE

Faculté des Sciences et Technologies

Rapport d'initiation à la recherche

Estimation des coûts de transformation d'une requête



Étudiants :
Adèle BARBIER
Clément COLNÉ
Corentin ROBERGE-MENTEC

Encadrants :
Olivier BRUNEAU
Nicolas LASOLLE
Jean LIEBER

Année universitaire 2020-2021

Décharge de responsabilité

L'Université de Lorraine n'entend donner ni approbation ni improbation aux opinions émises dans ce rapport, ces opinions devant être considérées comme propres à leur auteur.

Remerciements

Nous tenons à remercier toutes les personnes qui ont contribué au bon déroulement de notre travail et qui nous ont aidés lors de la rédaction de ce rapport.

Tout d'abord, nous adressons nos remerciements à nos encadrants : Docteur Jean Lieber ainsi qu'à Monsieur Nicolas Lasolle qui nous ont beaucoup aidés. Leur écoute et leurs conseils nous ont permis de progresser dans notre démarche.

Enfin, nous tenons à remercier toutes les personnes qui nous ont conseillés et relus lors de la rédaction de ce rapport : nos familles, nos enseignants encadrants ainsi que Madame Marie-Laure Alves, notre enseignante de communication.

Table des matières

1 Introduction

Une **base de données** est un outil utilisé dans de nombreux domaines informatiques. Pour récupérer des informations d'une base, un utilisateur peut formuler sa demande sous la forme d'une **requête** qui comprend toutes les contraintes souhaitées. Par ce biais, on traduit une exigence informelle dans un langage compréhensible par une machine. Le résultat retourné par la base de données est donc un ou plusieurs éléments, qui correspondent exactement aux contraintes formulées par les critères de la requête. Or, dans certains cas, il serait avantageux de récupérer plus de résultats, correspondant approximativement aux contraintes de la requête.

Plaçons nous dans le contexte de la correspondances d'Henri Poincaré. Henri Poincaré est un mathématicien, physicien, philosophe et ingénieur français $XX^{ème}$ siècle. Il est considéré comme un des derniers grands savants universels, maîtrisant en particulier l'ensemble des branches des mathématiques de son époque. Il est à noter que dans la suite de ce document, nous nous détacherons de ce contexte. Henri Poincaré a échangé durant sa vie avec plus de 300 interlocuteurs différents, pour un total de plus de 2 100 lettres. Ces différentes lettres proviennent du domaine privé, administratif ou encore scientifique. Parmi ces lettres, la majorité sont écrites par Henri Poincaré, le reste lui étaient destinées. Toutes ces lettres sont regroupées sur un site internet [1], ou elles sont classées selon leur date, leur interlocuteur ou encore leurs sujets.

Cite permet notamment à un utilisateur de les consulter via un moteur de recherche. Celui-ci s'appuie sur une extension du Web, utilisant des bases de données au format RDF, standard du **Web sémantique** [2]. Il y est utilisé les requêtes **SPARQL**, langage de requête requis pour cette extension.

Prenons pour exemple la requête suivante :

$$\mathcal{Q} = \left| \begin{array}{l} \text{Donne-moi les lettres envoyées entre 1890 et 1895 par Paul Appell à Henri Poincaré} \\ \text{et qui mentionnent des travaux en mécanique.} \end{array} \right.$$

La base de données interrogée, comme nous l'avons vu, ne retournera que les lettres correspondant aux quatre critères émis (l'expéditeur, le destinataire, la période et le thème). Imaginons alors qu'un utilisateur ne trouve pas ce qu'il désire car la lettre recherchée n'a pas été écrite dans la période entrée. Ce cas met en lumière l'utilité d'obtenir plus de résultats qui sont moins précis, mais qui sont toujours cohérents avec la requête. Prenons par exemple une lettre qui correspond à tous les critères excepté la période ; supposons qu'elle a été écrite en 1889, elle est susceptible d'intéresser l'utilisateur.

Afin de lui proposer de nouveaux résultats pouvant correspondre au mieux à ses attentes, il faut savoir comment modifier la requête initiale. Cette requête modifiée ne doit pas donner les mêmes résultats que précédemment, mais tout de même rester proche sémantiquement parlant de celle de départ. Une idée pour proposer des requêtes alternatives est de définir et d'appliquer des **règles de transformation**. Elles permettront de modifier en partie la requête afin d'élargir les résultats. Prenons l'exemple de la requête $\mathcal{Q}$, les quatre critères que nous avons évoqués pourraient être modifiés par le biais de règles de transformation. Si nous avons les règles :

$$\mathtt{r}_1 = \text{Étendre les bornes de la période de 5 ans.}$$
$$\mathtt{r}_2 = \text{Échanger le destinataire et l'expéditeur.}$$
$$\mathtt{r}_3 = \text{Généraliser l'un des thèmes abordés.}$$

1. `http://henripoincare.fr/s/correspondance/page/accueil`
2. `http://henripoincare.fr/s/correspondance/page/sparql`

Si on applique ces règles à la requête $\mathcal{Q}$, on pourrait obtenir respectivement :

$$\mathcal{Q}_1 = \left| \begin{array}{l} \text{Donne-moi les lettres envoyées entre 1885 et 1900 par Paul Appell} \\ \text{à Henri Poincaré et qui mentionnent des travaux en mécanique.} \end{array} \right.$$

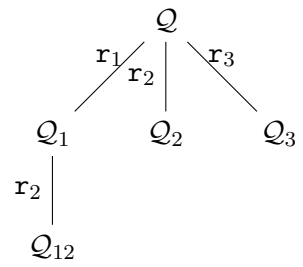
$$\mathcal{Q}_2 = \left| \begin{array}{l} \text{Donne-moi les lettres envoyées entre 1890 et 1895 par Henri Poincaré} \\ \text{à Paul Appell et qui mentionnent des travaux en mécanique.} \end{array} \right.$$

$$\mathcal{Q}_3 = \left| \begin{array}{l} \text{Donne-moi les lettres envoyées entre 1890 et 1895 par Paul Appell} \\ \text{à Henri Poincaré et qui mentionnent des travaux en physique.} \end{array} \right.$$

On peut facilement voir qu'il est nécessaire de donner un ordre d'importance relatif à l'application de ces règles de transformation. Pour ce faire, on leur affecte un **coût**. Il s'agit d'une valeur utilisée pour définir des priorités liées à l'application des règles. Ainsi plus un coût est petit, plus une règle est susceptible de fournir des résultats intéressants pour l'utilisateur.

Cependant, cette organisation des règles n'est pas intuitive car elle doit se baser sur ce que l'utilisateur désire trouver. Lorsqu'il signale qu'il veut plus de résultats à partir d'une requête, celle-ci est modifiée par la règle ayant le coût le plus faible. Mais s'il n'est pas satisfait des nouveaux résultats fournis, il faut remettre en question l'attribution de ce coût. Le problème est donc de décider comment réaffecter les coûts aux différentes règles pour s'assurer de prioritairement proposer les transformations les plus susceptibles d'intéresser l'utilisateur. Il s'agit de savoir si la fonction qui à une règle de transformation associe un coût (appelée **fonction de coût**), donne des résultats satisfaisants. Par le refus ou l'approbation des modifications apportées à sa requête, l'utilisateur établit des contraintes sur les règles de transformation que la fonction de coût devra respecter. Par exemple, s'il valide les résultats obtenus après application de la règle $\mathtt{r}_2$, mais pas ceux après application de $\mathtt{r}_1$, on devra avoir $\mathtt{coût}(\mathtt{r}_2) < \mathtt{coût}(\mathtt{r}_1)$.

La figure 1 présente un exemple de l'évolution d'une requête en fonction des coûts des règles de transformation.



avec $\mathtt{coût}(\mathtt{r}_1) > \mathtt{coût}(\mathtt{r}_2)$

$\mathtt{coût}(\mathtt{r}_1) + \mathtt{coût}(\mathtt{r}_2) > \mathtt{coût}(\mathtt{r}_3)$

FIGURE 1 – Arbre des transformations d'une requête $\mathcal{Q}$ en fonction du coût des règles.

Nos travaux de recherche porteront sur le problème de trouver un ordre satisfaisant au mieux les contraintes déduites des choix de l'utilisateur. Nous présenterons d'abord les différentes pistes que nous avons posées à la présentation du problème. Le choix de notre axe d'étude et les autres facettes du problème y seront alors évoqués. Ensuite, nous verrons quels ont été nos travaux sur cette problématique, et quels résultats nous avons pu en tirer.

2 Les différents problèmes

Ce sujet qui consiste à estimer les coûts de transformation d'une requête présente deux problèmes principaux : estimer le coût des règles, et trouver une fonction de coût qui satisfait au mieux les contraintes émises par l'utilisateur. Nous verrons dans cette section le détail de ces deux sujets d'étude.

2.1 L'estimation des coûts

Nous nous sommes dans un premier temps penchés sur le premier cas. Il s'agissait d'estimer la priorité des règles de transformation avant d'avoir eu des retours utilisateurs. Cette problématique ouvre sur un vaste champ des possibilités. D'abord, il faut déterminer quelle est la modification prioritaire qui saura apporter les résultats les plus satisfaisants. Mais il faut également savoir comment appliquer cette transformation ; il faut savoir si la sémantique prime sur la ressemblance syntaxique par exemple. Pour répondre à nos interrogations nous nous sommes renseignés sur les travaux traitant des similarités sémantique entre requêtes. En effet, nous avons pensé que plus deux requêtes étaient proches sémantiquement parlant, plus le coût de transformation pour passer de l'une à l'autre devrait être faible. À l'inverse, si la requête transformée avait un sens éloigné de la requête initiale, alors le coût de transformation devrait être élevé. Une piste aurait alors été d'étudier la distance sémantique entre deux mots. Plusieurs articles, détaillant des méthodes de détermination des distances sémantiques, nous ont paru appropriés ; Comme l'article [3], où il y est recensé plusieurs de ces méthodes, ainsi que des algorithmes les mettant en pratique. L'article [2] présente quant à lui une méthode basée sur la comparaison entre concept et ontologie. Par la suite, elle est évaluée et appliquée dans des cas concrets.

Une autre approche que nous avons étudiée a été la comparaison de définitions [1] entre différents termes. Si leurs définitions contenaient des mots importants en commun, tels que fruit à noyau pour pêche et brugnon, alors nous pouvons conclure qu'il est probable que ces deux mots aient un sens proche. Cependant, ce genre de techniques est rapidement poussée à ses limites. En effet, s'il existe des définitions imprécises, incomplètes, ou omettant des caractéristiques principales, alors deux mots proches dans leur sens ne pourraient ne pas être détectés tels quels.

Il est alors apparu qu'une multitude d'études existaient pour évaluer la similarité entre deux phrases. Il aurait donc été peu intéressant de porter trop d'attention à cette partie du sujet. En outre, nos tuteurs nous ont fait remarquer que cette notion de similarité entre concepts seule n'était pas suffisante pour évaluer la priorité de différentes règles. Ce sujet demande d'explorer différentes méthodes de comparaison de requêtes et à partir de celles-ci établir quelles sont les transformations prioritaires. C'est un travail qui nous a paru pour le moins subjectif car la priorité d'une information plutôt qu'une autre dépend du contexte dans lequel la requête est effectuée, mais également de l'utilisateur lui-même.

2.2 La réestimation des coûts

La deuxième facette de ce problème consiste à réévaluer les coûts affectés aux règles de transformation grâce aux retours d'utilisateurs. Nous nous sommes alors intéressés par la suite à la manière dont il serait possible de prendre en compte ces retours. La situation présupposée est la suivante : un utilisateur envoie sa requête à une base de données qui lui retourne le résultat ou l'ensemble des résultats correspondant. S'il le désire, il peut demander à avoir plus de résultats — c'est à ce moment que la requête initiale est modifiée. Une fois les nouveaux résultats affichés, il indique si ceux-ci le satisfont ou non. À nouveau, il peut demander plus de résultats. À la suite de ses retours sur la sortie de la base de données, il faut réestimer les coûts des règles de transformation.

Cette nouvelle piste est une toute autre approche du problème, car les coûts de l'ensemble des règles sont déjà estimés. On suppose donc que l'évaluation de priorité a été faite en amont. Ce qui nous intéresse ici est de réestimer les coûts qui ont été posés une fois que l'utilisateur a exprimé un retour sur la transformation de sa requête. Nous avons posé le problème de la sorte : si une requête a été transformée grâce à la règle $\mathtt{r}_1$ plutôt que la règle $\mathtt{r}_2$, cela implique que $\mathtt{coût}(\mathtt{r}_2) > \mathtt{coût}(\mathtt{r}_1)$. Or, si l'utilisateur n'est pas satisfait, l'inéquation doit être $\mathtt{coût}(\mathtt{r}_1) > \mathtt{coût}(\mathtt{r}_2)$.

Il est donc question ici de modifier la fonction de coût pour satisfaire les contraintes posées sous la forme d'inéquations. Il s'agit alors d'un problème d'optimisation. Par la suite nous appellerons `MRU` le multi-ensemble des retours utilisateurs. Il est composé de toutes les inéquations créées suite aux avis donnés par l'utilisateur. Or les inéquations fournies jusqu'à présent sont des inégalités strictes. Nous allons donc introduire la constante ε pour permettre d'obtenir des inéquations larges. En effet les contraintes dans un programme linéaire sont représentées par des inégalités larges, il est donc indispensable de faire cette conversion. ε sera fixée arbitrairement à 10^{-3} pour commencer. Il est nécessaire d'avoir une valeur faible par rapport à l'ordre de grandeur des coûts pour ne pas influencer le résultat du problème. Nous considérerons donc que les images de la fonction `coût` appartiennent à $\mathbb{N}$. Reprenons l'exemple $\mathtt{coût}(\mathtt{r}_2) > \mathtt{coût}(\mathtt{r}_1)$ pour illustrer ; dans ce cas cette inéquation devra dans un premier temps être transformée de la sorte : $\mathtt{coût}(\mathtt{r}_2) \geqslant \mathtt{coût}(\mathtt{r}_1) + \varepsilon$.

Une fois que le multi-ensemble `MRU` nous a été fourni, ainsi que l'ensemble des règles et la fonction de coût, plusieurs situations sont possibles : la meilleure éventualité est que la fonction de coût satisfasse déjà toutes les contraintes. Il n'y a donc pas besoin d'en chercher une nouvelle, les coûts sont correctement attribués. Si ce n'est pas le cas, il faut alors dans un premier temps vérifier que l'ensemble des retours utilisateurs est cohérent. En effet, si deux inéquations se contredisent, il sera impossible de trouver une solution. Il faut donc dans un premier temps trouver une solution pour rendre l'ensemble cohérent. C'est à ce moment que la fonction de coût va pouvoir être modifiée par la résolution du problème de programmation linéaire.

2.3 La direction de notre recherche

Étudier ces deux sujets lors de ce projet d'initiation à la recherche aurait été un travail très dense. Au vu du temps qui nous a été imparti, nos tuteurs nous ont prévenus qu'il était risqué de se lancer dans l'étude des deux problèmes. L'approfondissement de chacun en aurait été affecté. Nous avons donc décidé de ne traiter qu'une seule des deux problématiques. Notre préférence est allée pour le deuxième sujet, à savoir la réestimation des coûts à la suite d'une première estimation des règles de transformation.

3 Le problème de réestimation des coûts

Une fois que nous avons choisi précisément notre axe d'attaque du problème, il nous a fallu le poser explicitement. Les retours utilisateurs exprimés sous la forme d'un ensemble d'inéquations, et la recherche de valeurs de coûts permettant de les satisfaire, nous ont évoqué un problème d'optimisation linéaire. Pour la suite de notre travail de recherche, c'est donc ainsi que nous avons décidé de poser ce problème. En entrée, nous avons l'ensemble des règles de transformation, le multi-ensemble des retours utilisateurs `MRU`, et la fonction de coût initiale. À partir de ces éléments, il nous faut dans un premier temps identifier de quel cas il s'agit (voir 2.2). En fonction de celui-ci, il nous a fallu réfléchir aux différentes méthodes de résolution.

3.1 Cas 1 : fonction de coût cohérente avec `MRU`

Nous allons dans un premier temps présenter le cas où la fonction de coût fournie respecte les contraintes de l'utilisateur. Cela signifie que l'estimation des coûts faite en amont n'a pas à être remise en question ; l'utilisateur est satisfait de l'ordre dans lequel les transformations ont été faites.

Dans un problème d'optimisation linéaire, cela se présenterait sous la forme dans un plan en deux dimensions, comme présenté dans la figure 2.

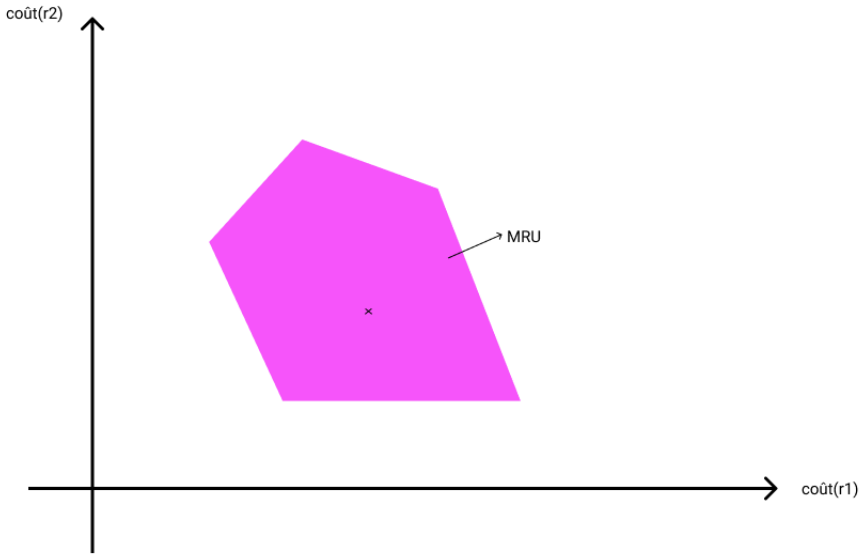


FIGURE 2 – Représentation graphique du cas 1

À chaque axe correspond la valeur de coût d'une règle (par souci de lisibilité, nous prenons un exemple avec seulement deux règles de transformation). Chaque contrainte de `MRU` est représentée par une droite. Les solutions qui satisfont toutes les inéquations correspondent aux points se trouvant à l'intérieur du polygone ainsi tracé (en rose sur le schéma). L'image de la fonction de coût correspond à l'ensemble des coûts de toutes les règles. Dans cette configuration, elle est représentée par le point $(\texttt{coût}(\texttt{r}_1), \texttt{coût}(\texttt{r}_2))$. Sur le schéma il s'agit donc du point tracé en noir à l'intérieur du polygone représentant `MRU`. Étant donné que nous sommes dans le contexte où la fonction de coût est correctement estimée, on remarque que le point représentant la fonction de coût se trouve bien à l'intérieur du polygone des contraintes.

Il s'agit du meilleur cas, car une réestimation n'est pas nécessaire. Lorsque nous rencontrons ce cas, nous avons donc décidé de ne pas changer la fonction de coût.

3.2 Cas 2 : fonction de coût incohérente avec `MRU`

Il s'agit ici du cas où la fonction de coût fournie ne satisfait pas toutes les contraintes de `MRU`. Par exemple, supposons que le retour de l'utilisateur établisse la contrainte suivante :

$$\texttt{coût}(\texttt{r}_1) > \texttt{coût}(\texttt{r}_2)$$

Maintenant supposons que la fonction de coût initialement fournie donne : $\texttt{coût}(\texttt{r}_1) = 5$ et $\texttt{coût}(\texttt{r}_2) = 10$. Pour rappel, la constante ε est fixée à une valeur très faible (10^{-3}). On voit alors bien que la fonction de coût se doit d'être évaluée de nouveau. Tout cela va dépendre de `MRU` ; c'est cet ensemble

qui va permettre de déterminer la nouvelle fonction de coût. Il est donc important de savoir si cet ensemble de contraintes est cohérent.

La cohérence de `MRU` se définit de telle sorte que toutes les contraintes qui le constituent ne se contredisent pas entre elles.

Prenons cet exemple :

$$\texttt{coût}(\texttt{r}_1) > \texttt{coût}(\texttt{r}_2)$$
$$\texttt{coût}(\texttt{r}_2) > \texttt{coût}(\texttt{r}_3)$$

Ici, aucune contrainte n'en contredit une autre, la cohérence est conservée.

Un multi-ensemble `MRU` incohérent se définit de sorte qu'il existe une ou plusieurs contraintes contredisant une autre ou d'autres. Prenons l'exemple suivant :

$$\texttt{coût}(\texttt{r}_1) > \texttt{coût}(\texttt{r}_2)$$
$$\texttt{coût}(\texttt{r}_2) > \texttt{coût}(\texttt{r}_1)$$

Nous pouvons voir que ce jeu de contraintes contient une incohérence. Le coût de transformation de la règle $\texttt{r}_1$ est à la fois supérieur et inférieur au coût de transformation de la règle $\texttt{r}_2$. Il est évident que cet ensemble d'inéquations est incohérent.

Il nous faut à présent savoir détecter ces cas, et comment résoudre le problème lorsque l'un d'eux se présente.

3.2.1 Cas 2.1 : `MRU` cohérent

Nous allons dans un premier temps étudier le cas où `MRU` est un multi-ensemble de contraintes cohérent, une solution satisfaisant les inéquations existe donc forcément. Les entrées du problème sont toujours l'ensemble des règles de transformation, le multi-ensemble `MRU` est une fonction de coût. Or, comme précisé auparavant, cette fonction de coût ne satisfait pas les contraintes de l'utilisateur. Le problème ici est d'en trouver une nouvelle qui affecte des coûts respectant toutes les inéquations.

Nous avons une fonction de coût initiale que l'on nommera `x`, et une fonction de coût respectant `MRU` nommée `y`. Cela signifie que $\texttt{x}_i$ correspond au coût initialement attribué à la règle numéro i. Notre objectif est de trouver une distance minimale entre `x` et `y`. Si nous revenons sur l'exemple utilisé plus tôt, avec deux règles $\texttt{r}_1$ et $\texttt{r}_2$ dans l'ensemble `MRU`, nous pourrions avoir un cas comme tel :

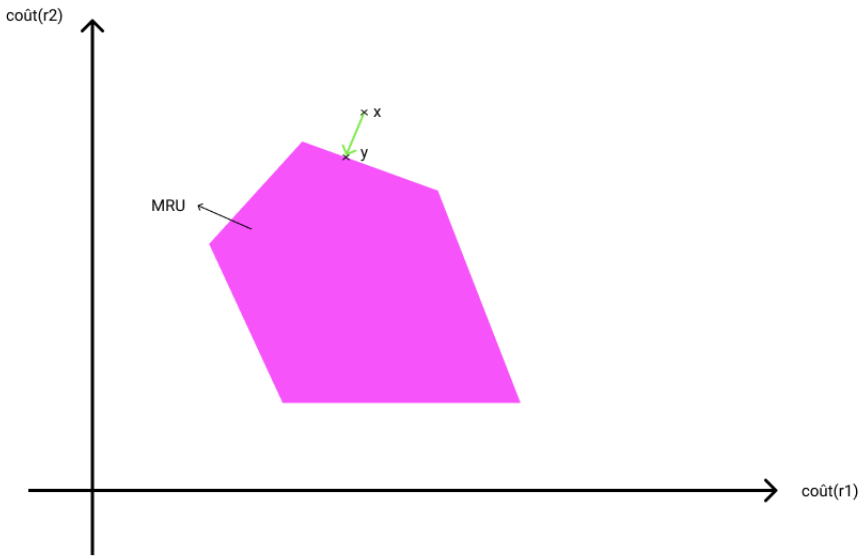


FIGURE 3 – Représentation graphique du cas 2.1

La fonction `y` satisfait `MRU`, le point le représentant graphiquement se trouve donc à l'intérieur du polygone tracé par les contraintes. Il s'agit maintenant de déterminer, à partir de `x`, quelle est la fonction `y` telle que la distance entre les deux soit minimale. Tout d'abord, il faut définir quelle distance nous allons utiliser.

Étant donné que nous avons posé le problème sous la forme d'un problème d'optimisation linéaire, il faut utiliser une distance la plus linéaire possible. Pour mesurer cette distance, il faut comparer chaque valeur de coût des deux fonctions deux à deux. La différence entre ces valeurs reflète l'écart entre les deux, nous allons donc utiliser une distance de Manhattan [4] pour l'évaluer. En effet, entre deux points A et B, de coordonnées respectives (X_A, Y_A) et (X_B, Y_B), la distance de Manhattan est définie par :

$$d(A, B) = |X_B - X_A| + |Y_B - Y_A|$$

Ici, nous voulons évaluer la différence entre les coûts de `x` et le coût de `y` correspondant. On en vient donc à la conclusion que la distance se mesure avec la formule :

$$\sum_{i=1}^{n} \mathtt{w}_i \cdot |\mathtt{x}_i - \mathtt{y}_i|$$

pour un problème à n règles de transformations. Les valeurs $\mathtt{w}_i$ correspondent au poids pour la règle i. Nous les fixerons tous dans un premier temps à 1, mais il pourra s'avérer utile de les changer par la suite.

Comme la cohérence de `MRU` a été vérifiée auparavant, les contraintes peuvent à nouveau être représentées par des inéquations larges dans notre problème. Il s'agit donc bien ici d'un problème d'optimisation quasi-linéaire : `MRU` comprend des contraintes linéaires, mais la fonction objectif comporte des valeurs absolues. Elle est donc elle-même quasi-linéaire. La question qui se pose alors est de savoir comment ramener ce problème à un problème d'optimisation linéaire.

Le souci ici est que la valeur absolue dans la formule rend la fonction objectif quasi-linéaire.

Nous allons donc remplacer cette partie par quelque chose de linéaire. La formule devient alors

$$\sum_{i=1}^{n} \mathtt{w}_i \cdot \mathtt{z}_i$$

De plus, pour tout i $\in$ [1 ;n], on ajoute les contraintes suivantes en plus de celles de `MRU` :

$$\mathtt{z}_i \geqslant \mathtt{x}_i - \mathtt{y}_i$$
$$\mathtt{z}_i \geqslant \mathtt{y}_i - \mathtt{x}_i$$

La fonction objectif est linéaire ainsi que toutes les contraintes ajoutées. Sachant que celles de `MRU` le sont également, on se retrouve donc ici avec un problème d'optimisation linéaire qu'un solveur peut résoudre.

3.2.2 Cas 2.2 : `MRU` incohérent

Jusqu'à maintenant, nous avons présenté les situations où `MRU`, c'est-à-dire les contraintes posées par l'utilisateur, était un ensemble cohérent. Ici, nous nous plaçons dans le dernier scénario, celui où au contraire, le multi-ensemble n'est pas cohérent.

L'incohérence d'un multi-ensemble `MRU` étant définie, détaillons le cas dans lequel nous nous trouvons. Nous avons ici une fonction de coût et un ensemble de contraintes `MRU`. Lorsque nous effectuons l'optimisation sur ces données par le biais d'un solveur, cela nous permet de détecter que `MRU` est incohérent. En effet, l'optimisation met en évidence l'inexistence de solution. Cette absence est due au fait que les contraintes du programme définissent les zones où une solution optimisée peut se trouver. Étant donné que cette solution doit satisfaire toutes les contraintes, elle doit appartenir à toutes les zones définies par celles-ci en même temps. S'il n'existe pas de solution appartenant à toutes les zones, alors c'est que l'ensemble des contraintes, c'est-à-dire `MRU`, est incohérent.

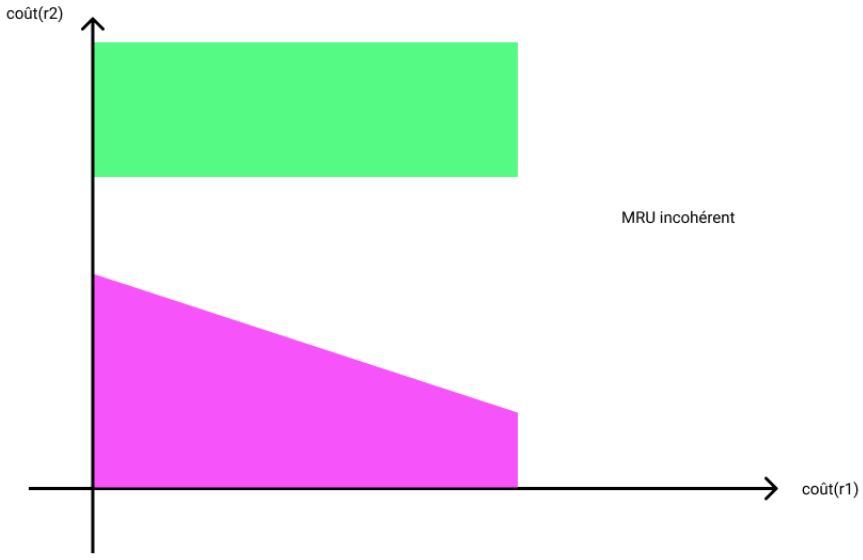


FIGURE 4 – Représentation graphique du cas 2.2

Sur ce la figure 4, on peut observer qu'il y a deux zones distinctes (c'est-à-dire deux contraintes) : une verte et une rose. Pour qu'une solution soit valide, elle doit appartenir à la fois à la zone verte

et à la zone rose. Or, dans cet exemple, en partant du constat que tous les coûts de transformation sont positifs, il est impossible de trouver une telle solution. Il s'agit donc bien d'un cas où `MRU` est incohérent.

Notre travail de recherche doit donc porter sur la manière dont nous pourrions trouver un nouvel ensemble `MRU′` cohérent. Une intuition évidente que nous avons eue est que `MRU′` est un sous-ensemble de `MRU`. En effet, il existe forcément un sous-ensemble de `MRU` qui sera cohérent. Dans le pire des cas, ce sous-ensemble ne contiendra qu'une seule contrainte, le rendant automatiquement cohérent. Dans le meilleur des cas, le retrait d'une seule contrainte de `MRU` le rendra cohérent.

Partant de cette intuition, nous avons établi un algorithme déterministe proposant de manière sûre un sous-ensemble `MRU′` issu de `MRU`.

Notre première proposition a été de faire une énumération exhaustive de tous les sous-ensembles de `MRU` existants. Une fois ce recensement fait, il suffit de sélectionner le plus grand sous-ensemble cohérent.

```
Fonction plusGrandMultiEnsembleCoherent( MRU : multi-ensemble) : multi-ensemble
    pgMRU' : multi-ensemble
    pgMRU' ← ensVide()
    Pour tout MRU' sous-ensemble de MRU faire
        Si (card(MRU') > card(pgMRU')  ET (estCoherent(MRU')) Alors
            pgMRU' ← MRU'
        Fin Si
    Fin Pour
    Retourner pgMRU'
Fin
```

Algorithme 2: algorithme de recherche exhaustive du plus grand sous-ensemble `MRU′` de `MRU`.

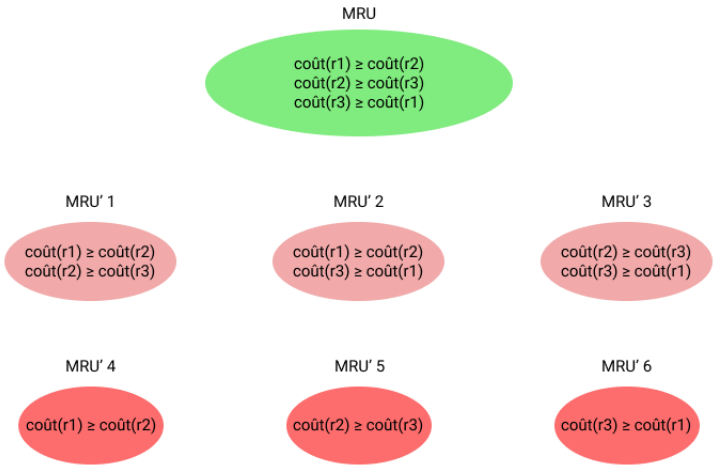


FIGURE 5 – Enumération exhaustive des sous-ensembles de `MRU`.

La figure 5 présente tous les sous ensembles trouvés par l'algorithme.

Cet algorithme permet de trouver une solution à coup sûr. Cependant, cette recherche a une complexité élevée : l'énumération exhaustive des sous-ensembles de `MRU` prend 2^n opérations, et la recherche du plus grand sous-ensemble cohérent, n opérations. Ceci nous donne $2^n + n$, soit une complexité algorithmique de $O(2^n)$. Même si cet algorithme fournit une solution de manière sûre, la complexité est exponentielle.

Dans un second temps, nous avons proposé un nouvel algorithme, qui reste exhaustif avec une complexité en $O(2^n)$ dans le pire des cas, mais qui, dans la pratique, nous propose une complexité plus faible. Cet algorithme fonctionne de la sorte : partant de `MRU`, on prend le sous-ensemble $\texttt{MRU}' = \texttt{MRU}\backslash\{c\}$ avec c une contrainte issue de `MRU`. Si ce sous-ensemble est cohérent, il est le plus grand sous-ensemble $\texttt{MRU}'$ cohérent, l'algorithme s'arrête. Sinon, on sélectionne un autre sous-ensemble issu de `MRU` où l'on aura retiré une autre contrainte que c, et on réitère jusqu'à avoir retiré au moins 1 fois chaque contrainte. Si aucun sous-ensemble cohérent n'a été trouvé, alors on recommence les même étapes décrite précédemment, mais en retirant cette fois 2 contraintes différentes. On recommence l'opération jusqu'à trouver un sous-ensemble cohérent contenant 2 contraintes ou plus, sinon le premier sous-ensemble à 1 seule contrainte est cohérent, et on le sélectionne.

```
Fonction plusGrandMultiEnsembleCoherent( ensIncoherents : file de multi-ensembles) :
multi-ensemble
    // Au premier appel à l'algorithme, initialiser ensIncoherents avec MRU incohérent
    c : contrainte
    pgMRU' : multi-ensemble
    MRU : multi-ensemble
    pgMRU' ← ensVide()
    MRU ← ensIncoherents.defiler()
    Pour tout c contrainte de MRU faire
        Si (estCoherent(MRU\{c})) Alors
            Retourner pgMRU'
        Fin Si
        ensIncoherents.enfiler(MRU\{c})
    Fin Pour
    Retourner plusPetitMru'(ensIncoherents)
Fin
```

Algorithme 4: amélioration de l'algorithme de recherche exhaustive du plus grand sous-ensemble $\texttt{MRU}'$ de `MRU`.

Cette solution est une variante du premier algorithme, mais la recherche est faite par un parcours en largeur permettant d'arrêter la recherche dès le premier sous-ensemble $\texttt{MRU}'$ cohérent rencontré, qui est forcément le plus grand sous-ensemble cohérent de `MRU`.

Cette recherche nous donne l'arbre de la figure 6 qui est parcouru en largeur :

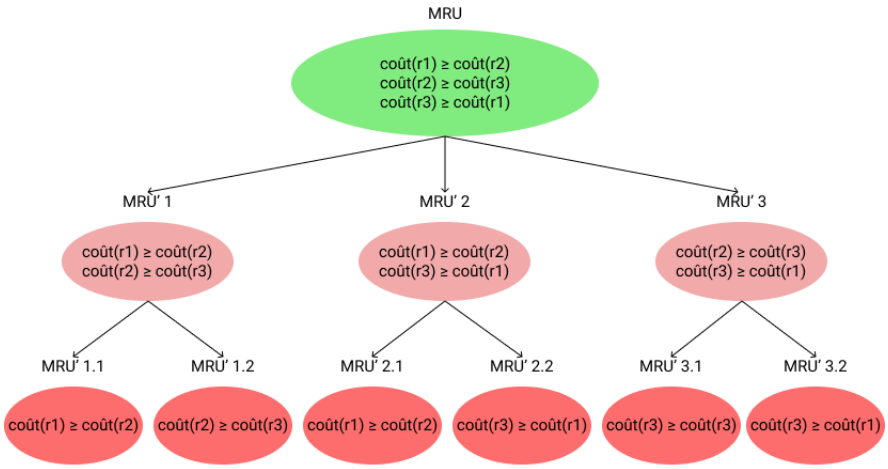


FIGURE 6 – Arbre des sous-ensembles de `MRU`.

À ce stade, nous avons détecté la présence de `MRU` incohérent et nous sommes capables de trouver `MRU`$'$. Il nous reste maintenant à résoudre le problème d'optimisation. En l'état, nous avons une fonction de coût et `MRU`$'$ notre ensemble de contraintes. Il semble alors évident que nous n'avons plus qu'à déterminer de nouveau dans quel cas nous nous trouvons parmi le cas 1 et le cas 2.1 (le cas 2.2 étant d'office exclu). La suite de la résolution, que nous nous trouvions dans le cas 1 ou 2.1 ayant été décrite précédemment, la résolution du cas 2.2 est maintenant terminée.

Par souci de rapport entre temps restant et qualité de la solution, nous nous contenterons ici de la solution proposée, qui possède une complexité élevée, mais un temps de recherche et de développement rapide. Le développement ne sera pas effectué dans le cadre de ce travail d'introduction à la recherche.

4 L'implémentation du cas où `MRU` est cohérent, mais la fonction de coût ne satisfait pas `MRU`

Une fois la théorie du cas 2.1 expliquée et comprise, nous l'avons implémentée. Nous avons écrit un programme en Java (version 1.8) qui fait appel au solveur lp_solve [3]. Il s'agit là d'un solveur de problèmes d'optimisation linéaires. Le code a été adapté de telle sorte qu'il soit possible de l'utiliser avec différents solveurs. Pour en changer, il suffit de créer une nouvelle classe pour chaque nouveau solveur, cette classe devant hériter de `AbstractSolver`. Cette nouvelle classe devra traiter les informations récupérées suite à l'exécution du solveur, particulièrement dans le cas où celles-ci ont un affichage différent que celui de lp_solve. Le programme prend en entrée un fichier texte, et produit un fichier lp.

lp_solve s'exécute en prenant en paramètre un fichier portant l'extension lp et respectant une syntaxe que nous avons définie. Il est également possible de spécifier des options d'exécution. Pour chaque option, lp_solve nous donne un résultat contenant plus ou moins de détails sur l'optimisation en sortie, avec un format différent.

3. http ://lpsolve.sourceforge.net

Nous avons préféré utiliser un fichier texte en entrée et non un fichier lp, afin de ne pas utiliser trop d'expressions régulières lors de la lecture de ce fichier. Le solveur ayant plusieurs options changeant l'affichage de la sortie standard, ces expressions doivent s'y adapter. Cela rend le code moins lisible en multipliant les conditions et les chaînes de caractères à filtrer. Le fichier texte nous donne donc accès à plusieurs informations, nous évitant d'avoir à les rechercher avec les expressions régulières. Ainsi la création des différents fichiers à transmettre au solveur nécessite juste une analyse du fichier texte et non pas des résultats donnés par l'exécution de lp_solve.

Nous avons spécifié une structure de fichier texte de telle sorte qu'il soit adapté au fonctionnement de lp_solve. Cependant, cette structure peut ne pas être adaptée à l'utilisation d'autres solveurs. En effet, nous n'avons travaillé qu'avec l'outil lp_solve jusqu'à maintenant. Il est possible qu'un autre solveur ait besoin de différentes informations pour résoudre le problème d'optimisation. Dans ce fichier, on y retrouve toutes les informations séparées par différents en-têtes permettant, lors de l'analyse des chaînes de caractères, de construire un fichier lp. La structure du fichier est donc composée d'une première ligne désignant l'optimisation souhaitée, suivit d'un en-tête symbolisé par « // ». Ce premier en-tête indique que la ligne suivante est la fonction de coût à analyser, et éventuellement, à optimiser. Elle contient donc les différentes valeurs composant la fonction de coût. Ensuite vient un ou plusieurs en-têtes indiquant chacun que la ligne qui suit est une contrainte de `MRU`. Cet en-tête est composé du numéro de la contrainte, ainsi que du type d'inégalité. La ligne suivante est composée des variables de la contrainte, ainsi que de la borne.

Voici la structure que nous avons spécifiée pour le fichier texte :

```
min (Resp. max, le type d'optimisation à effectuer)
// fonction (Écrire les valeurs des variables de la fonction de coût)
Votre fonction de coût (Par exemple "1 2 -3")
// ci eq (Le numéro de la contrainte et l'égalité permettant de fixer les valeurs d'une variable)
x1 5
A répéter pour toutes les valeur de la fonction de coût
// cj sup (Resp. inf, le numéro de la contrainte ainsi que le type d'inégalité, avec j commençant
à i+1)
x1 1 (Les variables de la contrainte ainsi que la valeur de la borne)
A répéter pour toutes les contraintes du problème
```

Ces informations sont récupérées et transposées dans le fichier fournit au solveur, afin d'être évaluées par celui-ci. Après avoir exécuté une première fois lp_solve sur le fichier lp, on obtient différents résultats nous permettant de déterminer si le problème se situe dans le cas 1 ou dans le cas 2. Une fois le cas détecté, nous avons deux alternatives : soit le problème se situe dans le cas 1, et dans ce cas aucun autre traitement n'est nécessaire, la fonction de coût est juste. Soit le problème se situe dans le cas 2, il faut donc déterminer dans quel sous-cas nous nous trouvons en exécutant une deuxième fois lp_solve sur un nouveau fichier lp.

Cette deuxième exécution permet de savoir si le problème se trouve dans le sous cas 2.1 ou 2.2. Soit il se trouve dans le sous cas 2.2, donc `MRU` est incohérent, soit il se trouve dans le sous cas 2.1 et une dernière exécution est nécessaire pour la résolution. Ce qu'il se passe lors des différentes exécutions, nous permettant de déterminer les cas, est expliqué dans la suite du rapport.

Un dossier contenant les tests unitaires ainsi que les fichiers de tests est fourni dans le projet. Ces tests permettent de vérifier que les problèmes liés à un mauvais format de fichier ou à des fichiers

vides soient gérés dans l'application. Ils permettent aussi de vérifier que les résultats obtenus lors de l'exécution du programme sur un fichier soient les résultats attendus.

La saisie des paramètres se fait comme suit :

java -jar `introductionALaRecherche.jar` solver file_path [solver_options]

4.1 La détection des cas

La détection des cas se fait lors du premier lancement de lp_solve. On obtient un résultat nous indiquant, soit que la fonction de coût initiale est correcte dans le problème d'optimisation, soit que le problème est infaisable. Si lp_solve nous donne un résultat et des informations sur l'optimisation du problème, alors nous sommes dans le cas 1. Il n'y a alors pas besoin de traitement car la fonction de coût satisfait déjà `MRU`. Si lp_solve nous indique que le problème est infaisable, il y a alors deux cas possibles : soit nous nous trouvons dans le cas 2.1, la fonction de coût fournie ne satisferait donc pas `MRU` ; soit nous nous trouvons dans le cas 2.2 et `MRU` est incohérent.

Afin de déterminer dans quel cas nous nous trouvons, il faut réécrire un fichier lp. Dans ce nouveau fichier, les valeurs de la fonction de coût sont remplacées par des variables, afin que lp_solve nous donne des valeurs adaptées. Il a donc fallu conserver le nombre de variables composant la fonction de coût, ainsi que leur valeur, et remplacer celles-ci par les variables présentes dans les contraintes. Ce faisant, l'exécution nous permet de détecter dans quel cas nous sommes. Si malgré ce changement le problème reste infaisable, l'erreur vient des contraintes, donc `MRU` est incorrect : nous sommes dans le cas 2.2. Si lp_solve nous donne un résultat, alors nous nous trouvons dans le cas 2.1. Or le résultat fourni par lp_solve n'est pas satisfaisant. En effet, nous obtenons un résultat qui est optimisé en fonction d'une minimisation ou d'une maximisation, donc nous avons les valeurs que prendront les différentes variables. Ces valeurs seront donc maximales ou minimales, mais pas forcément avec la distance de Manhattan la plus faible entre la fonction de coût et `MRU`. Il faut donc maintenant passer à la résolution du problème.

La restitution des résultats pour les cas 1 et 2.2 sont disponibles en annexe A et B.

4.2 La résolution automatique du problème

Afin de résoudre le problème, nous devons trouver une fonction de coût satisfaisant `MRU`, ayant la distance de Manhattan la plus faible possible entre celle-ci et la fonction de coût initiale. Il faut donc réécrire le fichier lp afin d'y insérer les calculs permettant d'obtenir cet élément. Pour cela, il faut écrire une nouvelle fonction à optimiser correspondant à la somme des $\mathtt{w}_i \cdot \mathtt{z}_i$. Ensuite, il suffit d'ajouter les encadrements des $\mathtt{z}_i$ aux contraintes de `MRU` afin que lp_solve résolve le problème. Enfin, une fois le fichier lp écrit, il faut exécuter lp_solve sur ce fichier, et nous obtiendrons les résultats correspondant aux nouvelles valeurs de la fonction de coût, ainsi que sa distance avec l'ancienne fonction de coût.

La restitution des résultats du cas 2.1 est disponible en annexe C.

5 L'évaluation de la méthode

Il est important d'évaluer l'efficacité de la méthode employée. Même si l'on peut trouver des résultats satisfaisant au cas par cas, il est nécessaire dans une étude de tester les limites de la démarche.

5.1 La première méthode d'évaluation suggérée

La première idée qui nous est venue afin d'évaluer l'efficacité de notre méthode était de faire une évaluation la plus proche possible de l'utilisation réelle. Pour cela, nous avons imaginé développer une interface graphique ainsi qu'une base de données. L'interface graphique contiendrait tous les éléments nécessaires à l'utilisation du logiciel : un formulaire pour entrer sa requête, un bouton pour lancer la recherche, une zone où afficher les résultats de la requête, un bouton pour signaler si les résultats sont satisfaisants et un autre pour réclamer plus de résultats. Cette façon de faire avait pour principal avantage de correspondre à une évaluation humaine, qui nous donnerait des résultats les plus proches de la réalité. Une maquette de l'interface que nous avons imaginée est disponible en annexe D.

Cependant, cette solution soulève quelques problèmes dans le cadre de notre travail d'initiation à la recherche. En effet, il est d'abord nécessaire de développer une interface graphique ainsi que la base de données qui va avec. Cette étape requiert un temps de travail conséquent, il faut spécifier les besoins précisément, effectuer la conception de l'interface ainsi que de la base de données, développer, puis déployer et tester. Après le développement de l'interface vient la partie de test par des utilisateurs humains. Il faut donc trouver un panel d'utilisateurs représentatif, et suffisamment grand, afin que les résultats soient cohérents et proches de la réalité. Ceci nécessite de trouver ces utilisateurs, ce qui n'est pas chose aisée. Il faut également leur faire effectuer des tests. A nouveau, cette étape prendrait beaucoup de temps, des jours voire des semaines pour effectuer un travail de qualité.

Il a donc rapidement été évident pour nous, que dans le cadre de notre travail d'initiation à la recherche, notre temps étant limité, et ce travail étant fastidieux et long, nous n'allions pas plus explorer cette piste ou la développer.

5.2 La méthode d'évaluation utilisée

Une seconde hypothèse à évaluer est la suivante : un nouveau coût $\mathtt{y}$ trouvé par optimisation linéaire sera meilleur qu'un coût $\mathtt{y}'$ choisit aléatoirement. Pour commencer, nous allons poser comme prérequis de cette hypothèse que le coût $\mathtt{x}$ intial est proche du coût optimal $\mathtt{x}^*$.

La première idée pour vérifier cette hypothèse a été la suivante : au départ de l'évaluation nous avons $\mathtt{x}^*$ à l'origine du repère (soit $\mathtt{x}^* = (0, 0, ..., 0)$ de taille n, n étant le nombre de règles), $\mathtt{x}_0$ et $\mathtt{MRU}_0 = \emptyset$. On effectue plusieurs itérations. Au temps t, l'ensemble $\mathtt{MRU}_t$ va évoluer par l'ajout d'une contrainte aléatoire. Grâce à l'optimisation linéaire, on détermine une nouvelle fonction de coût satisfiant $\mathtt{MRU}_t$. $\mathtt{x}_t$, à savoir la fonction de coût au temps t, est donc remplacé par $\mathtt{x}_{t+1}$ qui elle satisfait $\mathtt{MRU}_t$. Cependant, la contrainte ajoutée doit avoir les caractéristiques suivantes : elle doit être satisfaite par $\mathtt{x}^*$ mais pas $\mathtt{x}_t$. En effet, si la contrainte ajoutée était satisfaite par $\mathtt{x}_t$, ce point n'aurait pas besoin d'évoluer et donc rien ne se passerait pendant cette itération. En plus de cela, on choisit un point $\mathtt{y}_t$ aléatoirement, tel que cette fonction de coût aléatoire satisfait $\mathtt{MRU}_t$.

Le but ici est de montrer que la méthode d'optimisation permet d'obtenir un meilleur résultat qu'une fonction de coût aléatoire. Pour vérifier cela, il faut comparer à chaque itération la distance entre le coût optimal $\mathtt{x}^*$ et les résultats obtenus $\mathtt{x}_t$ et $\mathtt{y}_t$. Elle se mesure comme auparavant, avec une distance de Manhattan. Il peut arriver dans certains cas que la fonction de coût tirée au hasard soit plus proche de $\mathtt{x}^*$ que ne l'est $\mathtt{x}_t$. Notre objectif est donc d'expérimenter pour trouver la probabilité $\mathcal{P} = P(\mathtt{d}(\mathtt{x}^*, \mathtt{x}_t) \geqslant \mathtt{d}(\mathtt{x}^*, \mathtt{y}_t))$.

Cette probabilité dépend d'une part de $\mathtt{y}_t$ qui est fixé aléatoirement. Il n'y a donc pas de contrôle possible sur ce paramètre. En revanche, elle dépend également de $\mathtt{x}_0$, qui est fixée au départ de manière à ce qu'elle soit proche de $\mathtt{x}^*$. Le terme « proche » étant flou, on peut imaginer poser

une constante C telle que $\mathtt{d}(\mathtt{x}^*, \mathtt{x}_t) \leqslant \mathtt{C}$. Il s'agit alors d'observer les variations de la probabilité $\mathcal{P}$ en fonction des valeurs de C.

5.3 Les résultats de l'évaluation

Après avoir développé l'implémentation du cas 2.1 (voir 4), nous avons commencé à mettre en place l'évaluation de notre méthode. Pour ce faire, nous avons développé un programme en Java (version 1.8), mettant en pratique l'évaluation présentée précédemment (voir 5.2).

5.3.1 Le déroulement du programme

Il se présente sous la forme d'une classe `Evaluation`. Le programme se déroule de la sorte : on fournit en entrée un fichier qui décrit une fonction de coût. Tout ce qui est écrit sur ce fichier est enregistré. On génère ensuite une contrainte aléatoirement, en tirant au hasard un coefficient pour chaque variable de la contrainte, ainsi que la valeur de la borne. Cette contrainte est alors écrite dans le fichier texte fourni en entrée. Celui-ci va ainsi devenir le fichier d'entrée pour le programme précédemment développé. Si jamais la contrainte qui vient d'être générée rend MRU incohérent, cela est signalé à la sortie du programme utilisant le solveur. Il faut donc effacer la contrainte du fichier texte et en produire une nouvelle. Cette opération est répétée jusqu'à ce que l'on obtienne une sortie indiquant que le problème a pu être résolu. La nouvelle fonction de coût est alors récupérée et remplacée dans le fichier texte d'entrée initial (que nous appellerons fichier d'évaluation). À présent, on génère une fonction de coût aléatoire, qui satisfait toutes les contraintes générées précédemment. De la même manière que pour la contrainte aléatoire, on tire au hasard une valeur de coût pour chaque règle du problème. Tant qu'il existe au moins une des contraintes du problème non satisfaite, la fonction de coût y est générée de nouveau. Une fois que tout cela est fait, il est enfin possible de calculer la distance entre la solution optimale et y, ainsi que la distance entre la solution optimale et celle trouvée par le solveur.

5.3.2 Les détails de l'implémentation

Dans cette partie, nous verrons quelques spécificités du programme. Pour son bon fonctionnement, la mise en forme du fichier d'évaluation doit être la même que celle du fichier d'entrée du programme présenté dans la partie 4. Il ne doit y figurer que la fonction de coût, ainsi que le type d'optimisation désiré (soit uniquement les trois premières lignes du fichier).

Au lancement de l'évaluation, l'utilisateur entre le nombre de variables n du problème (soit le nombre de règles) ainsi que le nombre d'itérations t sur lesquelles l'évaluation va être effectuée. L'ensemble MRU est représenté par un tableau à deux dimensions de taille t sur n+1. Pour que cette évaluation se rapproche au mieux de la réalité, tous les coefficients générés sont positifs. En effet, en observant la manière dont nous avons posé le problème (comme montré sur la figure 1), on constate que si une règle apparaît dans une contrainte, le coefficient de sa variable sera positif. Les contraintes sont donc représentées de la sorte : la valeur MRU[i][j] représente le coefficient de la variable correspondant à la règle j dans la contrainte générée à la $\mathrm{i}^{ème}$ itération (avec $0 \leqslant i < \mathtt{t}$ et $0 \leqslant j < \mathtt{n}$). La valeur MRU[i][n] représente la valeur à droite de l'inéquation pour la contrainte générée à la $\mathrm{i}^{ème}$ itération (avec $0 \leqslant i < \mathtt{t}$). Toutes les inéquations générées sont donc de la forme :

$$\sum_{j=0}^{n-1} \mathtt{MRU}[i][j] * \mathtt{x}_j < \mathtt{MRU}[i][n]$$

Or, cette représentation ne correspond pas forcément à la réalité ; il pourrait très bien y avoir besoin d'une inéquation supérieure. C'est pour cela qu'à chaque génération de contrainte, l'inéquation est mutitpliée par -1 avec une probabilité de 50%, car ainsi nous avons :

$$\sum_{j=0}^{n-1} -\texttt{MRU}[i][j] * \texttt{x}_j < -\texttt{MRU}[i][n] \iff \sum_{j=0}^{n-1} \texttt{MRU}[i][j] * \texttt{x}_j > \texttt{MRU}[i][n]$$

Cette méthode d'évaluation nécessite d'écrire dans le fichier d'évaluation. Lorsqu'une contrainte générée ne convient pas, il faut remplacer le texte dans le fichier. Nous n'avons pas trouvé de moyen de remplacer seulement une partie d'un fichier en Java 1.8. L'alternative trouvée est d'utiliser un objet StringBuilder. Toutes les données écrites dans le fichier d'évaluation sont enregistrées dans cet objet. Cette utilisation nous donne l'occasion d'utiliser la méthode `replace`, qui permet de remplacer la dernière contrainte ajoutée dans le fichier texte, si jamais elle ne satisfait pas `MRU`.

5.3.3 Les remarques sur l'implémentation

L'évaluation a été développée après l'implémentation du cas 2.1. Il a donc fallu que tous les tests unitaires soient déployés, et le programme complètement terminé pour lancer son implémentation. Cette partie a donc été prise en charge assez tard dans notre projet d'initiation à la recherche.

Une première chose que nous avons pu remarquer en débutant la programmation, est que nous n'avons sans doute pas suffisamment réfléchi au code du programme initial. Le fait que le programme ne puisse prendre en entrée qu'un fichier texte a été le premier problème constaté. Étant donné que les contraintes sont générées aléatoirement à chaque itération, il aurait été plus simple de ne travailler qu'avec le tableau `MRU` : cette configuration nous oblige à écrire dans un fichier texte, et à enregistrer toutes les données écrites pour pouvoir remplacer une contrainte en cas de besoin. Simplement écraser les anciennes valeurs de la contrainte dans le tableau aurait été un gain de temps et de mémoire. Il a également fallu modifier le programme initial : par exemple, pour savoir si le problème ne peut pas être résolu à cause de l'incohérence de `MRU` il a été rajouté un champ « statut » pour pouvoir indiquer le problème après la fin du programme. Ce développement a donc fait rejaillir le manque de temps que nous avons passé sur la réflexion avant le code. Mais cette absence est majoritairement due à la durée de notre projet de recherche. Nous avons en effet mis un certain temps à nous approprier le sujet et à poser le problème, le code a donc plus difficilement avancé durant cette période. À l'époque, nous avons envisagé au vue de ce qu'il nous restait à faire, le mieux était d'avancer sur le programme afin de pouvoir évaluer la méthode, car cela représente une partie essentielle du projet.

5.3.4 Les résultats de l'évaluation

Les contraintes générées, dans un premier temps, ne satisfaisaient pas forcément le coût optimal $\texttt{x}^*$. Néanmoins, le problème d'optimisation cherche à minimiser la fonction de coût, donc calculer la distance entre le résultat et $\texttt{x}^*$ reste pertinent dans cette évaluation.

Dans un premier temps, la valeur maximale des coefficients d'une contrainte a été fixée à 200. Cela signifie donc que `MRU`[i][j] $\in$ [0,200] pour $0 \leqslant i < \texttt{t}$ et $0 \leqslant j < \texttt{n}+1$. Ce nombre a été fixé arbitrairement, mais tout de même dans l'optique d'obtenir des résultats variés. Le nombre de variables du problème a été initialement fixé à 2 pour 5 itérations. Les constatations que nous avons pu remarquer ont été les suivantes : d'une part plus le nombre de variables est grand, plus la génération de coût aléatoire n'aboutit jamais. En moyenne, nous avons pu observer qu'à partir de 4 variables dans le problème, si `MRU` comprend une contrainte avec des coefficients positifs, le programme ne parvient pas à générer un coût aléatoire. En effet, lorsqu'il s'agit de ne satisfaire que

des contraintes avec coefficients négatifs de la forme $\sum_{j=0}^{n-1} -\texttt{MRU}[i][j] * \texttt{x}_j < -\texttt{MRU}[i][n]$, on voit ici qu'il est plus probable d'obtenir une fonction qui satisfait cette contrainte : plus le coût donné sera grand, plus la contrainte sera satisfaite. Il est donc cohérent que ce problème se déclenche lors du premier ajout d'une contrainte à coefficients positifs.

Un souci majeur de cette évaluation a donc été le fait que la génération de la fonction de coût aléatoire provoque l'arrêt du programme. Il a donc été difficile d'en extraire des résultats pertinents. Les évaluations de plus de 5 itérations ne sont pas fréquentes, mais nous avons néanmoins extrait des résultats.

Nombre de Variables	Numéro de l'itération	Nombre d'itérations prévues	$P(\texttt{d}(\texttt{x}^*, \texttt{x}_t) \geqslant \texttt{d}(\texttt{x}^*, \texttt{y}_t))$
2	5	5	0.2
2	1	5	1.0
2	5	10	1.0
2	2	10	1.0
2	3	10	1.0
2	4	10	0.75
2	5	10	0.8
2	2	10	1.0
2	3	10	1.0
2	2	10	1.0

Ce tableau recense les résultats de 10 exécutions. Le détail de chaque itération intermédiaire est disponible en annexe E. Les lignes gardées ici sont les dernières itérations du programme.

On constate donc que dans 9 cas sur 10, le programme n'a pas effectuée toutes les itérations prévues. L'exécution a systématiquement été stoppée lors de la génération d'un coût aléatoire. On peut remarquer que les probabilités les plus faibles correspondent aux cas où les itérations sont plus nombreuses. Nous pourrions alors supposer que sur un plus grand échantillon, le résultat de ces probabilités serait beaucoup plus varié. Néanmoins, on constate qu'elles sont pour la majorité très proches de 1. Cela peut nous laisser penser que, dans la plupart des cas, la solution que nous avons développée est meilleure que celle trouvée aléatoirement.

6 Conclusion

Initialement, ce projet proposait un sujet relativement vaste : à partir d'une requête initiale, estimer le coût de transformation de cette requête vers d'autres requêtes transformées. De cette première problématique, nous en avons extrait deux sujets : estimer le coût de transformation, c'est-à-dire estimer la priorité des règles de transformation, et la ré-évaluation des coûts déjà affectés aux règles de transformation. Nous avons initialement porté notre travail vers de la recherche bibliographique. Ces recherches nous ont mené vers le calcul de distance entre deux requêtes. Pour cela, nous avons trouvé des articles portant sur la distance sémantique entre deux mots, le premier mot correspondant à un concept présent dans la requête initiale, et le deuxième dans la requête transformée. Cette distance nous donnait donc un potentiel coût de transformation entre deux requêtes. Après avoir effectué ces premières recherches, nous avons pris la décision de concentrer notre travail sur la réestimation des coûts de transformation. Le temps imparti pour ce travail de recherche ne nous a pas laissé l'occasion d'explorer les deux possibilités, et cette piste nous a le plus attirée.

Dans un premier temps, nous avons posé explicitement notre problématique, qui porte sur la réestimation des coûts de transformation, en prenant en compte les retours de satisfaction d'utilisateurs. L'ensemble des retours utilisateurs a mené notre travail vers la résolution de problèmes d'optimisation linéaire. Nous avons travaillé et rédigé une partie théorique résultante du problème que nous avions posé, tout en développant en parallèle un programme Java permettant de répondre concrètement au problème posé selon la théorie exposée. Enfin, nous avons cherché à tester la validité de notre travail de recherche. Pour cela, nous avons exposé dans une dernière partie deux hypothèses d'évaluation, permettant de vérifier le bon fonctionnement de notre travail, et avons développé la deuxième hypothèse.

L'implémentation de l'évaluation de notre méthode ne nous a pas permis d'obtenir des résultats concluants. Nous pouvons néanmoins émettre une hypothèse sur la validité de notre résultat. Cette méthode de réestimation semble prometteuse pour trouver des coûts satisfaisants. Nous espérons que notre travail saura être utile aux chercheurs qui travailleront sur cette piste.

Ce projet s'inscrit dans une étude ayant pour ressources le corpus des correspondances de Henri Poincaré, où l'on applique des règles de transformation. Le but de ce travail de recherche était d'améliorer ce système en le liant au travail qui a été fait sur les coûts afin de mieux gérer les priorités. Par la suite cela a pour objectif d'améliorer les résultats de l'étude. Dans le cadre de leur recherche, il leur sera possible d'opérer une évaluation humaine, ce qui n'a pas pu avoir lieu dans le cadre de ce projet comme expliqué précédemment. Jusqu'à présent, la réestimation a été faite sur des règles de transformations dénuées de signification ; le contexte de leur application n'importait pas. La reprise de notre travail permettra de constater son efficacité avec des exemples contextualisés, à savoir, le corpus des correspondances d'Henri Poincaré.

Lors de ce travail, nous avons appris qu'un problème, même s'il semble explicité correctement lors d'une première lecture, peut s'avérer bien plus complexe et conséquent que prévu. Nous avons découvert le fait de pousser la réponse théorique à un problème bien plus loin que ce que nous avons l'habitude d'effectuer, rechercher des travaux similaires ou s'approchant de notre problématique, utiliser des domaines qui ne semblent pas directement liés à nos recherches, comme la résolution de problèmes d'optimisations linéaires.

Table des annexes

A Restitution des résultats du programme lors du cas 1

```
La solution est dans MRU
```

B Restitution des résultats du programme lors du cas 2.2

```
Problème infaisable

Optimisation de la solution :

MRU incohérent
```

C Restitution des résultats du programme lors du cas 2.1

```
Problème infaisable

Optimisation de la solution :

min: z1 + z2;
c1: z1 >= y1 - 5;
c2: z1 >= 5 - y1;
c3: z2 >= y2 - 2;
c4: z2 >= 2 - y2;
c5: y1 >= 1;
c6: y1 <= 3;
c7: y2 >= 1;
c8: y2 <= 3;

La solution est dans MRU

Value of objective function: 2.00000000

Actual values of the variables:
z1                              2
z2                              0
y1                              3
y2                              2
```

D Maquette de l'interface graphique

x

E Résultat détaillé de 10 exécutions de l'évaluation

Nombre de variables	Numéro de l'itération	Nombre d'itérations prévues	$P(\mathtt{d}(\mathtt{x}^*, \mathtt{x}_t) \geqslant \mathtt{d}(\mathtt{x}^*, \mathtt{y}_t))$
2	1	5	0,0
2	2	5	0,0
2	3	5	0,0
2	4	5	0,0
2	5	5	0,2
2	1	5	1,0
2	1	10	1,0
2	2	10	1,0
2	3	10	1,0
2	4	10	1,0
2	5	10	1,0
2	1	10	1,0
2	2	10	1,0
2	1	10	1,0
2	2	10	1,0
2	3	10	1,0
2	1	10	0,0
2	2	10	0,5
2	3	10	0,6666667
2	4	10	0,75
2	1	10	1,0
2	2	10	0,5
2	3	10	0,6666667
2	4	10	0,75
2	5	10	0,8
2	1	10	1,0
2	2	10	1,0
2	1	10	1,0
2	2	10	1,0
2	3	10	1,0
2	1	10	1,0
2	2	10	1,0

Références

[1] Ismaïl Biskri et Sylvain Delisle : Les n-grams de caractères pour l'aide à l'extraction de connaissances dans des bases de données textuelles multilingues. *Proceedings of TALN-2001*, pages 93–102, 2001.

[2] Mouna Khatraoui, Nabila Bousbia et Amar Balla : Détection de similarité sémantique entre pages visitées durant une session d'apprentissage. *Atelier Mesures de Similarité Sémantique (EGC'08). 8èmes Journées Francophones Extraction et Gestion des Connaissances*, pages 121–129, 2008.

[3] Andon Tchechmedjiev : État de l'art : mesures de similarité sémantique locales et algorithmes globaux pour la désambiguïsation lexicale à base de connaissances. *In Proceedings of the Joint Conference JEP-TALN-RECITAL 2012, volume 3 : RECITAL*, pages 295–308, 2012.

[4] Wikipédia : Distance de manhattan — wikipédia, l'encyclopédie libre, 2019. [En ligne ; Page disponible le 29-mai-2019].

9 Déclaration contre le plagiat

Nous soussigné(e)s, BARBIER Adèle, COLNÉ Clément, ROBERGE-MENTEC Corentin, Régulièrement inscrit à l'Université de Lorraine Année universitaire : 2020-2021 Niveau d'études : Master Parcours : Informatique

Certifions qu'il s'agit d'un travail original et que toutes les sources utilisées ont été indiquées dans leur totalité. Nous certifions, de surcroît, que nous n'avons ni recopié ni utilisé des idées ou des formulations tirées d'un ouvrage, article ou mémoire, en version imprimée ou électronique, sans mentionner précisément leur origine et que les citations intégrales sont signalées entre guillemets.

Conformément à la loi, le non-respect de ces dispositions nous rend passible de poursuites devant la commission disciplinaire et les tribunaux de la République française.

Fait à Nancy, le 13 décembre 2021

10 Résumé

Ce rapport étudie le problème d'obtenir des résultats approchés supplémentaires lors de l'interrogation d'une base de données. Pour ce faire, la requête initiale doit être transformée, sans trop dériver de son sens premier. Une piste à étudier est de savoir quelle transformation appliquer en priorité sur cette requête. Pour cela, un coût lui est affecté ; intuitivement, plus celui-ci est faible et moins la signification de la nouvelle requête est modifié. Le travail a commencé par des recherches bibliographiques, afin de comprendre le sujet et s'en imprégner. Au fur et à mesure de ces recherches, deux pistes se sont formées. La première est l'estimation initiale des coûts de transformation d'une requête. Ce sujet n'a pas été exploré dans le cadre du travail effectué. La deuxième, celle sur laquelle le travail a porté, permet de savoir comment réestimer le coût de transformation d'une requête, en fonction des retours utilisateurs. Il a donc fallu étudier comment procéder en passant par un problème d'optimisation linéaire, puis développer un programme Java effectuant cette réestimation. Après avoir développé ce programme, une évaluation a été effectuée, afin d'estimer la pertinence de l'approche. Plusieurs idées ont été trouvées pour cette évaluation. La première est la prise en compte des avis d'utilisateurs via une interface graphique. La deuxième est d'utiliser des contraintes et des fonctions de coût aléatoires, afin de montrer que la méthode s'appuyant sur l'optimisation linéaire reste meilleure. La limite de temps et la quantité de travail requis pour développer et tester l'interface graphique ont conduit à étudier puis implémenter la deuxième idée.

Abstract

This report studies the problem of getting similar results when a database is queried. The query has to be transformed, but not divert too much from its primary meaning. A research focus is knowing what transformation to apply in priority on this query. For this purpose, it is assigned a cost ; intuitively the lower it is, the less the meaning of the new query is changed. The work began with bibliographic research to understand the subject and to immerse ourselves in it. As this research progressed, two leads were formed. The first one is the initial estimate of the processing costs of a query. This topic has not been explored in this work. The second one, the one on which this work has focused on, allows to know how to re-estimate the cost of transforming a query, based on user feedback. It was therefore necessary to study how to proceed through a linear optimization problem, and then developed a Java program that performed this re-estimation. After developing this program, an evaluation was completed, to estimate the relevance of the approach. Several ideas for this evaluation has been found. The first one is to consider user reviews via a graphical interface. The second one is to use random constraints and cost functions to reveal that linear optimization method remains better. The time limit and the amount of work required to develop and test the graphical interface led to study and then implement the second idea.