RÉVISOR : un ensemble de moteurs d'adaptation de cas par révision des croyances

Julien Cojan[1], **Valmi Dufour-Lussier**[2,3,4], **Alice Hermann**[2,3,4], **Florence Le Ber**[5], **Jean Lieber**[2,3,4], **Emmanuel Nauer**[2,3,4], **Gabin Personeni**[2,3,4]

[1] INRIA Sophia-Antipolis, projet Wimmics
[2] Université de Lorraine, LORIA, UMR 7503 — 54506 Vandœuvre-lès-Nancy, France
[3] CNRS — 54506 Vandœuvre-lès-Nancy, France
[4] Inria — 54602 Villers-lès-Nancy, France
[5] ICUBE, Université de Strasbourg/ENGEES, CNRS — 67000 Strasbourg, France
Julien.Cojan@inria.fr, Valmi.Dufour@loria.fr, Alice.Hermann@loria.fr, Florence.Leber@engees.unistra.fr, Jean.Lieber@loria.fr, Emmanuel.Nauer@loria.fr, Gabin.Personeni@loria.fr

Résumé

L'adaptation par révision est une approche de l'adaptation de cas fondée sur l'utilisation d'un opérateur de révision des croyances. Le principe de cette adaptation est de réviser le cas source par le problème cible, autrement dit, elle consiste à modifier minimalement le premier pour qu'il soit cohérent avec le second. Plusieurs implantations de l'adaptation par révision ont été développées dans plusieurs formalismes. Trois d'entre elles sont actuellement disponibles dans RÉVISOR, qui contient donc trois moteurs d'adaptation : RÉVISOR/PL en logique propositionnelle, RÉVISOR/CLC pour un formalisme de conjonction de contraintes linéaires et RÉVISOR/QA pour une algèbre qualitative (notamment, l'algèbre de Allen, INDU et RCC8).

Mots-clefs : adaptation, révision des croyances, logique propositionnelle, contraintes linéaires, algèbre qualitative, système *open-source*

Abstract

Revision-based adaptation is an approach to adaptation based on the use of a belief revision operator. The principle of this adaptation is to revise the source case with the target case, i.e. to minimally modify the former so that it becomes consistent with the latter. Several implementations of revision-based adaptation have been developed in various formalisms. Three of them are currently available in RÉVISOR, which contains three revision-based adaptation engines: RÉVISOR/PL for propositional logic, RÉVISOR/CLC for a formalism of conjunctions of linear constraints , and RÉVISOR/QA for a qualitative algebra formalism (including Allen calculus, INDU and RCC8).

Keywords: adaptation, belief revision, propositional logic, linear constraints, qualitative algebra, open-source system

1 Introduction

Le raisonnement à partir de cas (RÀPC [19]) a pour but de résoudre de nouveaux problèmes, à l'aide de cas, un cas étant la représentation d'un épisode de résolution de problème. L'approche la plus courante du RÀPC consiste à (1) retrouver un ou plusieurs cas d'une base de cas, (2) adapter ce ou ces cas afin de résoudre le nouveau problème, (3) appliquer éventuellement d'autres étapes qui ne sont pas considérées dans ce papier. Dans le modèle de [1], appelé modèle des 4 R, (1) s'appelle *retrieve*, (2) s'appelle *reuse* et (3) correspond aux étapes appelées *revise* et *retain* [1].

Cet article se concentre sur l'étape (2), l'adaptation. Plus spécifiquement, il se concentre sur l'adaptation d'un seul cas, par opposition à l'adaptation multiple (de plusieurs cas simultanément). Ces dernières années, l'adaptation a

1. La confusion suivante doit être évitée : l'étape *revise* de ce modèle diffère de la notion de révision des croyances utilisée dans cet article. On pourrait discuter d'un lien entre les deux, mais ceci se situe en-dehors des objectifs de l'article.

reçu un regain d'attention dans la communauté internationale du RÀPC. La notion d'amalgame a été introduite comme un modèle formel pour combiner plusieurs cas, ce qui est utile pour l'adaptation multiple (voir, p. ex. [15]). L'adaptation a aussi été étudiée pour des cas représentés par des flux opérationnels [16], des plans [8], des stratégies de jeux [20], etc.

Plus précisément, cet article concerne l'adaptation par révision, une approche fondée sur l'utilisation d'un opérateur de révision des croyances (voir [7] pour une synthèse des travaux sur le sujet dans notre équipe). Cette approche a été étudiée dans plusieurs formalismes et différents moteurs ont été développés. L'idée a alors émergé de partager ces moteurs avec la communauté du RÀPC et aussi la communauté de la révision des croyances. RÉVISOR contient trois de ces moteurs. Il est disponible sous licence GPL à l'adresse suivante :

`http://revisor.loria.fr`

Notons que chacun de ces moteurs a été développé indépendamment des autres, avec des choix d'implantation et de langages de programmation parfois différents. Néanmoins, tous ces moteurs sont interfacés avec Java et c'est le seul langage de programmation que les utilisateurs de RÉVISOR ont à connaître (il en va autrement pour ceux qui voudraient étendre ces moteurs, cependant).

Le but de cet article est de décrire RÉVISOR : sur quels principes il s'appuie, quels sont les différents moteurs qu'il contient actuellement et comment ils sont utilisés. La section 2 présente les principes de l'adaptation par révision. La section 3 décrit RÉVISOR en général. Puis, les trois moteurs actuellement disponibles dans RÉVISOR sont décrits : RÉVISOR/PL (en logique propositionnelle, section 4), RÉVISOR/CLC (dans un formalisme de conjonction de contraintes linéaires, section 5) et RÉVISOR/QA (dans une algèbre qualitative, telle que l'algèbre de Allen ou RCC8, section 6). Des exemples simples d'adaptation sont donnés afin d'illustrer comment ces trois moteurs peuvent être utilisés. La section 7 présente une conclusion et des perspectives.

2 Principes de l'adaptation par révision

2.1 Préliminaires

Soit $(\mathcal{L}, \models)$ une logique : $\mathcal{L}$ est un ensemble de formules et $\models$ est la relation de conséquence logique. $\varphi_1 \equiv \varphi_2$ signifie que φ_1 et φ_2 sont des formules équivalentes, i.e. $\varphi_1 \models \varphi_2$ et $\varphi_2 \models \varphi_1$.

Dans cet article, trois formalismes sont utilisés : $\varphi \in \mathcal{L}$ est soit une formule de logique propositionnelle (pour RÉVISOR/PL), soit une conjonction de contraintes linéaires (pour RÉVISOR/CLC), soit un réseau de contraintes qualitatives, i.e. une base de connaissances fondée sur une algèbre qualitative donnée (pour RÉVISOR/QA). Pour chacun de ces formalismes, une sémantique fondée sur la théorie des modèles peut être définie. Plus précisément, un ensemble $\mathcal{U}$ et une fonction $\mathcal{M} : \mathcal{L} \to 2^{\mathcal{U}}$ peuvent être définis de telle façon que, pour $\varphi_1, \varphi_2 \in \mathcal{L}$, $\varphi_1 \models \varphi_2$ ssi $\mathcal{M}(\varphi_1) \subseteq \mathcal{M}(\varphi_2)$. Une *interprétation* est un élément de $\mathcal{U}$ et un *modèle* de $\varphi \in \mathcal{L}$ est une interprétation x telle que $x \in \mathcal{M}(\varphi)$. Une formule $\varphi \in \mathcal{L}$ est dite *cohérente* (ou satisfiable) si $\mathcal{M}(\varphi) \neq \emptyset$.

Par exemple, si $\mathcal{L}$ est la logique propositionnelle avec n variables $a_1, \ldots, a_n$, avec $\mathbb{B} = \{\mathtt{V}, \mathtt{F}\}$, $\mathcal{U}$ l'ensemble des 2^n interprétations sur $a_1, \ldots, a_n$ ($\mathcal{U} = \mathbb{B}^n$) et $x \in \mathcal{M}(\varphi)$ ssi $\varphi^x = \mathtt{V}$, où φ^x est la valeur de vérité de φ étant donnée une interprétation x, définie de façon classique [2].

Toutes les logiques $(\mathcal{L}, \models)$ considérées dans ce papier sont supposées stables par conjonction : si $\varphi_1, \varphi_2 \in \mathcal{L}$ alors $\varphi_1 \wedge \varphi_2 \in \mathcal{L}$ et $\mathcal{M}(\varphi_1 \wedge \varphi_2) = \mathcal{M}(\varphi_1) \cap \mathcal{M}(\varphi_2)$. Cela nous permettra d'assimiler une base de connaissances (ou de croyances) $\{\varphi_1, \ldots, \varphi_n\}$ à la formule $\varphi_1 \wedge \ldots \wedge \varphi_n$.

Une distance sur $\mathcal{U}$ est une fonction $d : \mathcal{U} \times \mathcal{U} \to [0; +\infty]$ telle que $d(x, y) = 0$ ssi $x = y$ (les autres propriétés des distances ne sont pas requises dans ce papier). Étant donné $A, B \in 2^{\mathcal{U}}$ et $y \in \mathcal{U}$, les abréviations suivantes seront utilisées :

$$d(A, y) = \inf_{x \in A} d(x, y) \qquad d(A, B) = \inf_{x \in A, y \in B} d(x, y)$$

2.2 Révision des croyances

Soit ψ une représentation des croyances d'un agent, exprimées dans une logique $\mathcal{L}$. Cet agent est confronté à de nouvelles croyances exprimées par $\mu \in \mathcal{L}$. Ces nouvelles croyances sont supposées non révisables, alors que les anciennes croyances ψ peuvent être modifiées. Si μ n'est pas contradictoire avec ψ (i.e., $\psi \wedge \mu$ est cohérent), alors les nouvelles croyances sont simplement ajoutées aux anciennes. Sinon, selon le principe du changement minimal [2], ψ peut être modifié minimalement en $\psi' \in \mathcal{L}$ tel que $\psi' \wedge \mu$ est cohérent, et ainsi, la révision de ψ par μ, notée $\psi \dot{+} \mu$, est cette conjonction.

Il y a plusieurs façons de mesurer la modification des croyances et donc, il y a plusieurs opérateurs de révision $\dot{+}$. Dans [2], un ensemble de postulats qu'un opérateur de révision est supposé vérifier a été défini. Dans [14], ces postulats ont été reformulés en logique propositionnelle et une famille d'opérateurs de révision fondés sur des distances ont été définis comme suit. Soit d, une distance sur $\mathcal{U} = \mathbb{B}^n$. Soit ψ et μ, deux formules de logique propositionnelle. La révision de ψ par μ selon l'opérateur $\dot{+}^d$ ($\psi \dot{+}^d \mu$) est la formule dont les modèles sont les modèles

2. $\varphi^x \in \{\mathtt{V}, \mathtt{F}\}$. Si φ est une variable propositionnelle a_i, alors $\varphi^x = a_i^x = x_i$. Si $\varphi = \varphi_1 \wedge \varphi_2$ alors $\varphi^x = \mathtt{V}$ ssi $\varphi_1^x = \mathtt{V}$ et $\varphi_2^x = \mathtt{V}$. Si $\varphi = \varphi_1 \vee \varphi_2$ alors $\varphi^x = \mathtt{V}$ ssi $\varphi_1^x = \mathtt{V}$ ou $\varphi_2^x = \mathtt{V}$. Si $\varphi = \neg\varphi_1$ alors $\varphi^x = \mathtt{V}$ ssi $\varphi_1^x = \mathtt{F}$. $\varphi_1 \Rightarrow \varphi_2$ est une abréviation pour $\neg\varphi_1 \vee \varphi_2$. $\varphi_1 \Leftrightarrow \varphi_2$ est une abréviation pour $(\varphi_1 \Rightarrow \varphi_2) \wedge (\varphi_2 \Rightarrow \varphi_1)$.

de μ les plus proches des modèles de ψ selon d (la modification d'une interprétation x en une interprétation y est mesurée par $d(x, y)$). Formellement, $\psi \dot{+}^d \mu$ est tel que :

$$\begin{aligned}&\mathcal{M}(\psi \dot{+}^d \mu)\\ &= \{y \in \mathcal{M}(\mu) \mid d(\mathcal{M}(\psi), y) = d(\mathcal{M}(\psi), \mathcal{M}(\mu))\}\end{aligned} \quad (1)$$

On peut noter que cette définition ne définit $\psi \dot{+}^d \mu$ qu'à l'équivalence près, mais ceci est suffisant puisqu'un opérateur de révision doit satisfaire le principe de non pertinence de la syntaxe (cf. le postulat suivant de [14] : si $\psi_1 \equiv \psi_2$ et $\mu_1 \equiv \mu_2$ alors $\psi_1 \dot{+} \mu_1 \equiv \psi_2 \dot{+} \mu_2$). On peut aussi noter que pour que cette définition soit correcte, il est nécessaire que la partie droite de cette égalité corresponde à un sous-ensemble de $\mathcal{U}$ qui soit *représentable* dans $\mathcal{L}$ ($A \subseteq \mathcal{U}$ est représentable dans $\mathcal{L}$ s'il existe $\varphi \in \mathcal{L}$ tel que $\mathcal{M}(\varphi) = A$). Cependant, cela ne pose pas problème pour la logique propositionnelle (avec un nombre fini de variables) puisque tout ensemble d'interprétations $A \subseteq \mathcal{U} = \mathbb{B}^n$ est représentable dans cette logique. Pour les autres logiques considérées dans cet article, une définition similaire de $\dot{+}^d$ peut être donnée, mais la problématique de la représentabilité doit être étudiée.

2.3 Notations concernant le RàPC

Soit $(\mathcal{L}, \models)$ la logique dans laquelle les bases de connaissances d'une application du RàPC sont définies [3]. Un cas *source* $\texttt{Source} \in \mathcal{L}$ est un cas de la base de cas. Souvent, un tel cas représente une expérience spécifique, autrement écrit : $\mathcal{M}(\texttt{Source})$ est un singleton. Cependant, cette hypothèse n'est pas nécessaire pour les moteurs d'inférences de RÉVISOR (même si elle a un impact positif sur le temps de calcul). Un cas cible $\texttt{Cible} \in \mathcal{L}$ représente un problème à résoudre. Cela signifie que des informations à propos de `Cible` sont manquantes et résoudre `Cible` conduit à y ajouter des informations [4]. Ainsi, l'adaptation de `Source` pour résoudre `Cible` consistera à construire une formule $\texttt{CibleComplétée} \in \mathcal{L}$ rendant `Cible` précise : $\texttt{CibleComplétée} \models \texttt{Cible}$. Afin de réaliser l'adaptation, les connaissances du domaine $\texttt{CD} \in \mathcal{L}$ peuvent être utilisées. Ainsi, le processus d'adaptation a la signature suivante :

$$\texttt{Adaptation} : (\texttt{CD}, \texttt{Source}, \texttt{Cible}) \mapsto \texttt{CibleComplétée}$$

2.4 L'adaptation par révision

Soit $\dot{+}$ un opérateur de révision dans la logique $(\mathcal{L}, \models)$ utilisée dans un système de RàPC. La $\dot{+}$-adaptation est définie comme suit :

$$\texttt{CibleComplétée} = (\texttt{CD} \wedge \texttt{Source}) \dot{+} (\texttt{CD} \wedge \texttt{Cible})$$

Intuitivement, le cas source est modifié minimalement afin que soit satisfait le cas cible. Les deux cas sont considérés modulo les connaissances du domaine.

3. Ces bases de connaissances appelées *CBR knowledge containers* en anglais, ce que les auteurs répugnent à traduire, sont d'habitude au nombre de quatre : la base de cas, les connaissances du domaine (ou ontologie du domaine), la similarité (connaissance pour la recherche d'un cas) et les connaissances d'adaptation [18].

4. Intuitivement, `Cible` peut être vue comme la donnée d'une « partie problème » précisément spécifiée et d'une « partie solution » manquante.

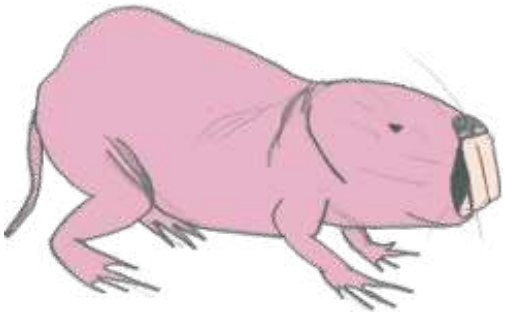

FIGURE 1 – Le logo de RÉVISOR est un rat taupe glabre. N'est-il pas tout simplement magnifique ?

3 Présentation générale de RÉVISOR

La figure 1 montre le logo de RÉVISOR.

Sur `http://revisor.loria.fr`, un fichier `.zip` contenant les moteurs de RÉVISOR peut être téléchargé. Cette archive contient, pour chacun des trois moteurs :

- Un fichier contenant des exemples (`TestRevisorPL.java`, `TestRevisorCLC.java` et `TestRevisorQA.java`),
- Un fichier contenant les méthodes principales, en particulier un opérateur de révision des croyances et un opérateur d'adaptation par révision (`RevisorPL.java`, `RevisorCLC.java` et `RevisorQA.java`) et
- Un répertoire contenant l'implantation de ces méthodes (en Java pour RÉVISOR/PL, en Java et C pour RÉVISOR/CLC, en Java et Perl for RÉVISOR/QA).

4 RÉVISOR/PL : un système pour la révision et l'adaptation par révision en logique propositionnelle

4.1 Description de RÉVISOR/PL

RÉVISOR/PL implante une famille d'opérateurs de révision en logique propositionnelle et les opérateurs d'adaptation par révision correspondants. Ces opérateurs sont les opérateurs $\dot{+}^d$ fondés sur les distances de Hamming pondérées définies ci-dessous. Soit $\mathcal{V}$ l'ensemble des variables propositionnelles apparaissant dans les formules et

soit $\{w_a\}_{a\in\mathcal{V}}$ une famille de poids ($w_a > 0$). La distance de Hamming pondérée par ces poids est définie par :

$$d(x,y) = \sum_{a\in\mathcal{V}} \textbf{si } a^x \neq a^y \textbf{ alors } w_a \textbf{ sinon } 0$$
$$\text{pour } x, y \in \mathcal{U}$$

Par défaut, ces poids sont tous égaux à 1 et, dans ce cas, $d(x,y)$ est le nombre de changements d'affectations de valeurs de variables nécessaires pour transformer x en y([5]). L'opérateur de révision est alors appelé opérateur de révision de Dalal [10].

RÉVISOR/PL est un programme Java organisé comme suit :

- `RevisorPL.java` est une classe appelant les méthodes principales (méthodes pour $\dot{+}^d$ et pour la $\dot{+}^d$-adaptation, et méthodes d'affectation des poids) ;
- `TestRevisorPL.java` contient deux exemples détaillés dans la section suivante ;
- Le reste du programme est dans le *package* `pl`.

La version de Java requise est la 1.7.

4.2 Exemples d'utilisation de RÉVISOR/PL

Les deux exemples ci-dessous peuvent être trouvés dans `TestRevisorPL.java`.

Exemple 1.

$$w_a = 1.0 \text{ pour toute variable propositionnelle } a$$
$$\texttt{Source} = \texttt{tarte} \wedge \texttt{pomme} \wedge \texttt{pâte\_à\_tarte} \wedge \texttt{sucre}$$
$$\texttt{Cible} = \texttt{tarte} \wedge \neg\texttt{pomme}$$
$$\texttt{CD} = (\texttt{pomme} \vee \texttt{poire}) \Leftrightarrow \texttt{fruit}$$

L'opérateur de révision est celui de Dalal. Le but est d'adapter une recette `Source` de tarte aux pommes à la requête « Je veux une tarte sans pomme. » Les connaissances du domaine consistent à affirmer d'une part que les pommes et les poires sont des fruits et, d'autre part, que les seuls fruits disponibles sont les pommes et les poires.

La $\dot{+}$-adaptation est calculée comme suit[6] :

```
RevisorPL.resetWeights();
PLFormula
    source1 = RevisorPL.parseFormula("pie &
        apple & pie_shell & sugar"),
    target1 = RevisorPL.parseFormula("pie &
        !apple"),
    dk1 = RevisorPL.parseFormula
       ("(pear | apple > fruit) & (fruit >
           pear | apple)"),
    result1 = RevisorPL.adapt(source1,
        target1, dk1);
RevisorPL.print(result1);
```

L'exécution de ce code donne le résultat suivant :

```
((fruit & pie_shell & !apple & pie & pear &
    sugar))
```

C'est-à-dire :

$$\texttt{CibleComplétée} \equiv \texttt{tarte} \wedge \neg\texttt{pomme} \wedge \texttt{poire} \wedge \texttt{fruit} \wedge \texttt{pâte\_à\_tarte} \wedge \texttt{sucre}$$

i.e., l'adaptation consiste à remplacer des pommes par des poires.

Exemple 2.

$$w_a = 1.0 \text{ pour toute variable propositionnelle } a \neq \texttt{chocolat}$$
$$w_{\texttt{chocolat}} = 2.0$$
$$\texttt{Source} = \texttt{menthe} \wedge \texttt{chocolat} \wedge \texttt{gâteau}$$
$$\texttt{Cible} = \neg(\texttt{chocolat} \wedge \texttt{menthe})$$
$$\texttt{CD} = \texttt{vrai} \quad \text{(i.e., base de connaissances vide)}$$

L'opérateur de révision s'appuie sur la distance de Hamming pondérée, avec un poids plus élevé pour la variable `chocolat`.

Le code suivant décrit comment effectuer cette adaptation :

```
RevisorPL.resetWeights();
RevisorPL.setWeight("chocolate", 2.0);
PLFormula
    source2 = RevisorPL.parseFormula("mint &
        chocolate & cake"),
    target2 =
        RevisorPL.parseFormula("!(chocolate &
        mint)"),
    dk2 = PL.TRUE,
    result2 = RevisorPL.adapt(source2,
        target2, dk2);
RevisorPL.print(result2);
```

L'exécution de ce code donne le résultat suivant :

```
((cake & chocolate & !mint))
```

Autrement écrit :

$$\texttt{CibleComplétée} \equiv \texttt{chocolat} \wedge \neg\texttt{menthe} \wedge \texttt{gâteau}$$

i.e., un gâteau avec chocolat et sans menthe, ce qui est le résultat attendu puisque $w_{\texttt{chocolat}} > w_{\texttt{menthe}}$ (se débarrasser du chocolat est considéré comme étant plus difficile que se débarrasser de la menthe, selon le goût de l'utilisateur).

4.3 Principe de l'algorithme de RÉVISOR/PL

L'opérateur de révision de RÉVISOR/PL s'appuie sur un algorithme similaire à celui présenté dans [21] et est inspiré du travail de [6], consacré à l'adaptation pour la logique de descriptions $\mathcal{ALC}$. Il s'appuie sur la méthode des tableaux

5. Les auteurs ont renoncé à regret à traduire *flip of variable* par « chiquenaude sur une variable ».

6. Syntaxiquement, dans RÉVISOR/PL, $\neg$, $\wedge$, $\vee$ et $\Rightarrow$ sont dénotés respectivement par `!`, `&`, `|` et `>`. Il n'y a pas de symbole pour $\Leftrightarrow$, c'est pourquoi $\varphi_1 \Leftrightarrow \varphi_2$ est remplacé par $(\varphi_1 \Rightarrow \varphi_2) \wedge (\varphi_2 \Rightarrow \varphi_1)$.

sémantiques, une méthode générale pour la déduction, utilisée pour tester la cohérence d'une base de connaissances. L'idée est de prendre la conjonction $\psi \wedge \mu$ et d'appliquer des règles de transformation de la méthode des tableaux sémantiques, ces règles de déduction conduisent à de nouvelles formules et, si $\psi \wedge \mu$ est incohérent, des conflits apparaissent (où un conflit est une paire de formules $\{a, \neg a\}$, avec a une variable propositionnelle). Puis l'algorithme de révision de RÉVISOR/PL consiste à *réparer* ces conflits, en « enlevant » des éléments de connaissances de ψ (i.e., en l'affaiblissant). Quand ces conflits sont réparés, ψ a été transformé en ψ' tel que $\psi' \wedge \mu$ est cohérent.

Il faut noter que la description ci-dessus reste très générale et, qu'en pratique, il faudrait présenter certains détails techniques pour pouvoir l'implanter correctement et de façon (relativement) efficace, mais la description de ces détails n'est pas un objectif de cet article.

4.4 À propos du temps de calcul de RÉVISOR/PL

Le temps de calcul de RÉVISOR/PL dépend davantage de la forme syntaxique des formules, plutôt que de leur nombre de variables. RÉVISOR/PL est très efficace sur des formules sous (ou proches de la) forme normale disjonctive, mais nécessite davantage de temps pour réviser une formule sous forme normale conjonctive. RÉVISOR/PL est aussi assez rapide dans le cas particulier où l'une des formules ψ ou μ est insatisfiable. Tous les résultats de révision ou d'adaptation produits par RÉVISOR/PL sont sous la forme de formules en forme normale disjonctive. Ainsi, les opérations de révision et d'adaptation successives peuvent s'effectuer avec un temps de calcul moindre. Plus généralement, RÉVISOR/PL est dépendant de la complexité de la distribution des conjonctions sur les disjonctions des formules.

RÉVISOR/PL s'avère bien plus efficace, en termes d'occupation mémoire et de temps de calcul, que la révision utilisant des tables de vérité [7]. La méthode des tables de vérité prend environ 1000 millisecondes pour effectuer la révision de ψ par μ, ψ et μ étant 2 formules contenant 12 variables distinctes (environ 25 littéraux). Pour cette même tâche, RÉVISOR/PL prend environ 10 millisecondes, dans le pire des cas. Pour la révision de formules de 50 littéraux (soit environ 25 variables distinctes), RÉVISOR/PL prend environ 250 millisecondes dans le pire des cas, tandis qu'on estime le temps de calcul pour la méthode des tables de vérité à une heure au moins. Il est cependant important de noter que la différence en temps de calcul entre les pire et meilleur cas croît exponentiellement avec la taille des formules.

7. Ce que nous appelons « méthode des tables de vérité pour $+^d$ » est l'implantation naïve énumérant tous les modèles x de ψ et tous les modèles y de μ, pour conserver les y tels que $\min_{x \in \mathcal{M}(\psi)} d(x, y)$ est minimal.

5 RÉVISOR/CLC : un système pour la révision et l'adaptation par révision dans un formalisme de conjonctions de contraintes linéaires

5.1 Description de RÉVISOR/CLC

RÉVISOR/CLC a été développé initialement pour la troisième version du système TAAABLE [4], un participant au troisième *computer cooking contest* (quoique le nom RÉVISOR/CLC soit bien plus récent).

RÉVISOR/CLC effectue une adaptation par révision numérique sur des cas représentés par des conjonctions de contraintes linéaires. Le problème de révision est réduit à un problème d'optimisation linéaire qui est résolu par la librairie LP_SOLVE [3].

LP_SOLVE est une librairie C sous licence LGPL. Elle est utilisée dans RÉVISOR/CLC via un *wrapper* Java. Pour installer la librairie, il faut copier les fichiers `liblpsolve55.so` et `liblpsolve55j.so` pour l'architecture appropriée dans un répertoire et ajouter le chemin de ce répertoire dans la variable d'environnement `LD_LIBRARY_PATH`.

Les cas considérés ici sont représentés dans un espace attribut-valeur, avec les attributs $a_1, \ldots, a_n$ qui prennent leurs valeurs dans les ensembles respectifs $\mathcal{U}_1, \ldots, \mathcal{U}_n$, où $\mathcal{U}_i$ est soit l'ensemble $\mathbb{R}$ des réels soit l'ensemble $\mathbb{Z}$ des entiers. L'opérateur de révision $\dot{+}^d$ utilisé pour l'adaptation est défini à l'aide d'une distance d sur l'espace de représentation composé des espaces associés aux attributs : $\mathcal{U} = \mathcal{U}_1 \times \ldots \times \mathcal{U}_n$.

d est défini comme suit, pour $x, y \in \mathcal{U}$, $x = (x_1, \ldots, x_n)$ et $y = (y_1, \ldots, y_n)$:

$$d(x, y) = \sum_{i=1}^{n} w_i |y_i - x_i|$$

où les $w_i > 0$ sont des coefficients qui donnent leurs importances relatives aux attributs. Plus grand est w_i, plus il est difficile de changer la valeur de l'attribut a_i.

Les cas et les connaissances du domaine sont exprimés par des conjonctions finies de contraintes linéaires de la forme :

$$\sum_{i=1}^{n} \beta_i \cdot a_i \geqslant v$$

où les paramètres β_i et v sont des valeurs réelles (positives ou négatives). Les types de contraintes sont $\geqslant$, $\leqslant$ et $=$, bien que les deux derniers puissent se ramener à des contraintes du premier type. Les modèles d'une telle formule sont simplement les ensembles de valeurs $x = (x_1, \ldots, x_n) \in \mathcal{U}$ telles que $\sum_{i=1}^{n} \beta_i \cdot x_i \geqslant v$.

5.2 Exemple d'utilisation de RÉVISOR/CLC

Considérons l'exemple suivant, dans lequel une tarte aux pommes est adaptée en remplaçant les pommes par des poires. Les attributs sont $\texttt{poire}_{\text{g}}$, $\texttt{pomme}_{\text{g}}$, $\texttt{pâte}_{\text{g}}$, $\texttt{saccharose}_{\text{g}}$, $\texttt{fruit}_{\text{g}}$ et $\texttt{sucre}_{\text{g}}$, qui prennent leurs valeurs dans $\mathcal{U}_i = \mathbb{R}$ ($\mathit{machin}_{\text{g}}$ est la masse de machins), et $\texttt{pomme}_{\text{nb}}$ et $\texttt{poire}_{\text{nb}}$, qui prennent leurs valeurs dans $\mathcal{U}_i = \mathbb{Z}$ ($\mathit{machin}_{\text{nb}}$ est le nombre de machins) :

$$\texttt{Source} = (1 \cdot \texttt{poire}_{\text{nb}} = 0) \wedge (1 \cdot \texttt{pomme}_{\text{nb}} = 3)$$
$$\wedge (1 \cdot \texttt{pâte}_{\text{g}} = 100) \wedge (1 \cdot \texttt{saccharose}_{\text{g}} = 40)$$
une recette avec 3 *pommes,* 100 *grammes de pâte et* 40 *grammes de saccharose*

$$\texttt{Cible} = (1 \cdot \texttt{pomme}_{\text{nb}} = 0)$$
La recette adaptée ne doit pas contenir de pomme.

$$\texttt{CD} = (1 \cdot \texttt{poire}_{\text{g}} \geqslant 0)$$
$$\wedge (1 \cdot \texttt{pomme}_{\text{g}} \geqslant 0)$$
$$\wedge (1 \cdot \texttt{pâte}_{\text{g}} \geqslant 0)$$
$$\wedge (1 \cdot \texttt{saccharose}_{\text{g}} \geqslant 0)$$
$$\wedge (1 \cdot \texttt{pomme}_{\text{g}} = 100 \cdot \texttt{pomme}_{\text{nb}})$$
La masse moyenne d'une pomme est de 100 *grammes.*
$$\wedge (1 \cdot \texttt{poire}_{\text{g}} = 80 \cdot \texttt{poire}_{\text{nb}})$$
$$\wedge (1 \cdot \texttt{fruit}_{\text{g}} = 1 \cdot \texttt{pomme}_{\text{g}} + 1 \cdot \texttt{poire}_{\text{g}})$$
Les seuls fruits disponibles sont des pommes et des poires, lesquels forment des catégories disjointes de fruits[8].
$$\wedge (1 \cdot \texttt{sucre}_{\text{g}} = 1 \cdot \texttt{saccharose}_{\text{g}} + 0.1 \cdot \texttt{pomme}_{\text{g}} + 0.15 \cdot \texttt{poire}_{\text{g}})$$

Les poids sont choisis afin de normaliser les valeurs prises (les poids de $\texttt{poire}_{\text{nb}}$ et $\texttt{pomme}_{\text{nb}}$ sont plus élevés que les poids des attributs mesurant des masses en grammes) et pour mettre en évidence leurs importances relatives ($\texttt{fruit}_{\text{g}}$ a un poids élevé afin de renforcer la compensation de la réduction de la masse des pommes par une augmentation de la masse des poires). Ces poids sont donnés dans les deux tableaux suivants :

variable	$\texttt{poire}_{\text{nb}}$	$\texttt{poire}_{\text{g}}$	$\texttt{pomme}_{\text{nb}}$	$\texttt{pomme}_{\text{g}}$
poids	10	1	10	1

variable	$\texttt{fruit}_{\text{g}}$	$\texttt{pâte}_{\text{g}}$	$\texttt{saccharose}_{\text{g}}$	$\texttt{sucre}_{\text{g}}$
poids	5	1	1	5

On peut noter qu'une petite modification de ces poids ne change pas du tout le résultat.

8. Cela constitue évidemment une simplification par rapport au domaine culinaire. Pour rendre cet exemple plus réaliste, on pourrait introduire une autre variable, $\texttt{poire\_ou\_pomme}_{\text{g}}$, pour la masse des poires et des pommes. La réification de cette variable, ne regroupant que les poires et les pommes, ainsi que l'axiome $\texttt{poire\_ou\_pomme}_{\text{g}} = \texttt{poire}_{\text{g}} + \texttt{pomme}_{\text{g}}$, serait là pour indiquer une similarité entre ces deux types de fruits.

Ci-dessous, un extrait de l'exemple de `TestRevisorCLC.java` est présenté (le signe `[...]` dénote les parties du code qui sont similaires aux lignes de code données sous ce signe) :

```
RevisorCLC.resetSpace();

// Building space
    IntegerVariable pearNb = new
        IntegerVariable("pear nb");
    RevisorCLC.setWeight(pearNb, new
        RealValue(10));

    RealVariable pearMass = new
        RealVariable("pear mass");
    RevisorCLC.setWeight(pearMass, new
        RealValue(1));

[...]
//Building Domain knowledge
Simplex s = new Simplex();

// fruitMass = appleMass + pearMass
LinearConstraint constraint = new
    LinearConstraint();
constraint.setType(ConstraintType.EQUAL);
constraint.setOffset(new RealValue(0));
constraint.addTerm(fruitMass, new
    RealValue(1));
constraint.addTerm(appleMass, new
    RealValue(-1));
constraint.addTerm(pearMass, new
    RealValue(-1));
s.addConstraint(constraint);
[...]
SimplexDomainKnowledge dk = new
    SimplexDomainKnowledge(s);

// Builing target case
    Simplex tgtSimplex = new Simplex();
LinearConstraint tgtContr = new
    LinearConstraint();
tgtContr.setType(ConstraintType.EQUAL);
tgtContr.addTerm(appleMass, new
    RealValue(1));
tgtContr.setOffset(new RealValue(0));
tgtSimplex.addConstraint(tgtContr);
SimplexCase tgt = new
    SimplexCase(tgtSimplex);

[...]
SimplexCase tgtAdapted =
    RevisorCLC.adapt(srce, tgt, dk);
System.out.println(tgtAdapted);
```

L'exécution de ce code donne le résultat suivant :

```
1.0*fruit mass   = 320.0
1.0*pear nb   = 4.0
1.0*pear mass   = 320.0
1.0*apple nb   = 0.0
1.0*apple mass   = 0.0
1.0*sugar mass   = 70.0
1.0*saccharose mass   = 22.0
1.0*dough mass   = 100.0
```

C'est-à-dire :

$$\begin{aligned}\texttt{CibleComplétée} \equiv \texttt{CD} \\ &\wedge (1 \cdot \texttt{poire}_{\text{nb}} = 4) \\ &\wedge (1 \cdot \texttt{pomme}_{\text{nb}} = 0) \\ &\wedge (1 \cdot \texttt{pâte}_{\text{g}} = 100) \\ &\wedge (1 \cdot \texttt{saccharose}_{\text{g}} = 25)\end{aligned}$$

qui est le résultat attendu : les 3 pommes sont substituées par 4 poires afin d'obtenir une meilleur conservation possible de la masse totale de fruits. La quantité de saccharose est réduite de 40 à 25 grammes afin de compenser le fait que les poires sont plus sucrées que les pommes.

5.3 Principe de l'algorithme de RÉVISOR/CLC

Étant donné ψ et μ, deux bases de connaissances dans le formalisme de RÉVISOR/CLC, la révision de ψ par μ selon $+^d$ revient à la résolution du problème d'optimisation suivant :

$$x \in \mathcal{M}(\psi) \tag{2}$$
$$y \in \mathcal{M}(\mu) \tag{3}$$
$$\text{minimiser } d(x, y) \tag{4}$$

Ce problème est non linéaire, bien que (2) et (3) soient des conjonctions de contraintes linéaires. En effet, la fonction objectif, $d(x, y)$, est non linéaire (à cause des valeurs absolues). Cependant, ce problème d'optimisation peut être résolu en s'appuyant sur la solution du problème d'optimisation linéaire suivant :

$$\begin{aligned} &x \in \mathcal{M}(\psi) \\ &y \in \mathcal{M}(\mu) \\ z_i \geqslant y_i - x_i \quad &\text{et} \quad z_i \geqslant x_i - y_i \quad (i \in \{1, \ldots, n\}) \\ &\text{minimiser } \sum_{i=1}^{n} z_i \end{aligned}$$

En effet, si (x, y, z) est solution de ce second problème d'optimisation alors (x, y) est solution du premier et, réciproquement, si (x, y) est solution du premier problème alors (x, y, z) est solution du deuxième avec $z = (z_1, \ldots, z_n)$ et $z_i = |y_i - x_i|$.

Étant donné que le résultat désiré est l'ensemble des valeurs de y, le résultat de $\psi +^d \mu$ est donné par une conjonction de contraintes linéaires définissant cet ensemble de valeurs, lequel est, par conséquent, représentable dans ce formalisme.

5.4 À propos du temps de calcul de RÉVISOR/CLC

La complexité de l'optimisation linéaire est polynomiale si toutes les variables sont continues mais devient NP-complète si les variables peuvent être entières. Avec des variables des deux types, le problème peut se traiter en des temps raisonnables pour une centaine de variables entières et plusieurs milliers de variables continues. La réduction du problème d'adaptation en un problème d'optimisation linéaire est linéaire en fonction de la taille de l'espace de représentation et du nombre de contraintes linéaires, par conséquent, la complexité de l'adaptation dans ce cadre formel est du même ordre de grandeur que la complexité de l'optimisation linéaire.

Le calcul de l'exemple donné en section 5.2 a pris environ 150 ms sur une machine avec 4 cœurs à 3 GHz.

6 RÉVISOR/QA : un système pour la révision et l'adaptation par révision dans une algèbre qualitative

6.1 Description de RÉVISOR/QA

RÉVISOR/QA implante un opérateur de révision dans des algèbres qualitatives tel qu'il a été défini dans [9], et l'opérateur correspondant d'adaptation par révision introduit dans [13]. Il prend en entrée deux réseaux de contraintes qualitatives (RCQ), un RCQ étant la représentation d'un problème de satisfaction de contraintes donné par des variables représentant, par exemple, des intervalles temporels ou des régions de l'espace, et un ensemble de contraintes entre ces variables. L'opérateur d'adaptation peut prendre en plus une liste de substitutions et un RCQ représentant les connaissances du domaine. Il retourne un ensemble de scénarios, où un scénario est un RCQ satisfiable avec une seule contrainte sur chaque couple de variables.

Le cœur de l'implantation est écrit en Perl, mais une interface écrite en Java vers cette implantation est disponible (et nécessite Java 1.2). Par ailleurs, un interpréteur Perl est également requis, avec une version 5.8 ou une version plus récente. Perl est pré-installé sur Linux et MacOS et est disponible gratuitement pour Windows sur `www.activeperl.com`.

6.2 Exemple d'utilisation de RÉVISOR/QA

Nous considérons l'exemple suivant, où le système remémore un cas représentant une petite exploitation agricole de maïs que l'on veut adapter pour la culture du miscanthus [11, 12]. Les deux cas source et cible sont caractérisés par une rivière (lit mineur) entourée d'une plaine inondable (ou lit majeur). Dans le cas source, une parcelle de maïs se trouve en partie dans cette plaine inondable. En utilisant l'algèbre spatiale qualitative RCC8 [17], cette configuration est représentée par les formules suivantes (où NTTP signifie « est une partie propre non-tangentielle » et

PO « recouvre partiellement ») :

$$\begin{aligned}\texttt{Source} &= (\texttt{lit\_mineur}\ \{\mathsf{NTPP}\}\ \texttt{lit\_majeur})\\ &\wedge (\texttt{parcelle}(\texttt{maïs})\ \{\mathsf{PO}\}\ \texttt{lit\_majeur})\\ \texttt{Cible} &= (\texttt{lit\_mineur}\ \{\mathsf{NTPP}\}\ \texttt{lit\_majeur})\end{aligned}$$

Les agriculteurs savent que le miscanthus ne peut pas être planté dans des zones inondables, ce qui peut être représenté par ({DC, EC} signifiant « est déconnecté ou connecté extérieurement ») :

$$\texttt{CD} = (\texttt{parcelle}(\texttt{miscanthus})\ \{\mathsf{DC}, \mathsf{EC}\}\ \texttt{lit\_majeur})$$

De plus, comme nous avons remémoré une exploitation maïsicole, où nous voulons planter du miscanthus, nous devons faire la substitution suivante :

$$\texttt{Substitutions} = \{\texttt{maïs} \rightsquigarrow \texttt{miscanthus}\}$$

L'adaptation est alors calculée grâce au code suivant :

```
String source_str = "lit mineur {ntpp} lit
    majeur ;"
                  + "parcelle(mais) {po}
                      lit majeur";
String target_str = "lit mineur {ntpp}  lit
    majeur";
String dk_str     = "parcelle(miscanthus)
    {dc, ec}  lit majeur";

QualitativeAlgebra algebra =
    RevisorQA.loadAlgebra("RCC8");
QualitativeConstraintNetwork
    psi = RevisorQA.parseFormula(psi_str,
        algebra),
     mu = RevisorQA.parseFormula(mu_str,
         algebra),
     dk = RevisorQA.parseFormula(dk_str,
         algebra);
QualitativeVariableSubstitution
    subst =
        RevisorQA.prepareSubstitution("mais",
        "miscanthus");

List<QualitativeConstraintScenario>
     solution = RevisorQA.adapt(psi, mu, dk,
         subst);

Iterator<QualitativeConstraintScenario> itr
    = solution.iterator();
while (itr.hasNext()) {
    QualitativeConstraintScenario scenario =
        itr.next();
    System.out.println(scenario);
}
```

L'exécution du code donne le résultat suivant :

```
lit mineur { dc } parcelle(X0) ; lit mineur
    { ntpp }  lit majeur ; lit mineur { dc }
    parcelle(miscanthus) ; parcelle(X0) { ec
    }  lit majeur ; parcelle(X0) { eq }
    parcelle(miscanthus) ;  lit majeur { ec }
    parcelle(miscanthus)
```

Autrement écrit :

$$\begin{aligned}\texttt{CibleComplétée} \equiv\ & (\texttt{lit\_mineur}\ \{\mathsf{NTPP}\}\ \texttt{lit\_majeur})\\ &\wedge (\texttt{lit\_mineur}\ \{\mathsf{DC}\}\ \texttt{parcelle}(x))\\ &\wedge (\texttt{lit\_mineur}\ \{\mathsf{DC}\}\ \texttt{parcelle}(\texttt{miscanthus}))\\ &\wedge (\texttt{lit\_majeur}\ \{\mathsf{EC}\}\ \texttt{parcelle}(x))\\ &\wedge (\texttt{lit\_majeur}\ \{\mathsf{EC}\}\ \texttt{parcelle}(\texttt{miscanthus}))\\ &\wedge (\texttt{parcelle}(x)\ \{\mathsf{EQ}\}\ \texttt{parcelle}(\texttt{miscanthus}))\end{aligned}$$

Ce résultat montre que la parcelle de maïs a d'abord été supprimée et remplacée par une parcelle de « x », et ensuite raffinée en une parcelle de miscanthus en supposant que la parcelle de x et la parcelle de miscanthus étaient identiques (selon la relation EQ de RCC8). Au passage, la parcelle a été réduite de sorte qu'elle soit connectée extérieurement à la plaine inondable (EC) au lieu de la recouvrir partiellement (PO) ; on respecte ainsi les connaissances du domaine avec un changement minimal.

6.3 Principe de l'algorithme de RÉVISOR/QA

Pour rendre possible la révision d'un réseau de contraintes qualitatives, nous posons comme postulat l'équivalence entre les modèles de formules logiques et les scénarios d'un réseau de contraintes qualitatives. Nous définissons une distance entre réseaux sur la base de graphes de voisinages (préexistants) qui établissent une distance entre relations d'une algèbre qualitative et permettent ainsi de comparer les contraintes de deux réseaux.

L'algorithme est constitué de deux étapes. Premièrement, la substitution s'applique en deux temps, par une abstraction α sur le réseau source — dans l'exemple, $\texttt{parcelle}(\texttt{maïs})$ est remplacé par $\texttt{parcelle}(x)$ — et par un raffinement ϱ sur le réseau cible — en ajoutant la contrainte $\texttt{parcelle}(\texttt{miscanthus})\ \{\mathsf{EQ}\}\ \texttt{parcelle}(x)$. La connaissance du domaine est ajoutée aux deux réseaux. On obtient ainsi $\psi = \alpha(\texttt{Source}) \wedge \texttt{CD}$ et $\mu = \texttt{Cible} \wedge \texttt{CD} \wedge \varrho$.

Deuxièmement, pour appliquer la révision $\psi \dot{+} \mu$, un parcours A* est mis en œuvre sur les scénarios de μ. À chaque étape de la recherche, une contrainte de μ est réduite à une seule des relations possibles. La fonction coût est obtenue en sommant les distances des contraintes réduites aux contraintes correspondantes dans ψ ; la fonction heuristique est obtenue en calculant la borne minimale sur les distances entre les contraintes restantes et leurs correspondantes dans ψ.

Ainsi le résultat obtenu est l'ensemble des scénarios de μ à distance minimale de ψ.

6.4 À propos du temps de calcul de RÉVISOR/QA

L'exploration de l'espace des modèles d'un réseau de contraintes qualitatives est exponentiel par rapport au nombre de variables. RÉVISOR/QA n'est donc efficace que sur

de petits problèmes. Le nombre de relations est également important.

Par exemple, avec un CPU i5 à 2,5 GHz, la révision prend moins d'1 seconde en moyenne pour des problèmes d'au plus 4 variables, dans l'algèbre d'Allen (9 relations) ou de RCC8 (8 relations), et moins d'1 minute pour des problèmes d'au plus 8 variables. Dans INDU (25 relations), les problèmes de 4-variables prennent environ 100 secondes.

La révision de l'exemple de 4-variables utilisé ci-dessus est calculée en 0.002 seconde.

7 Conclusion et perspectives

Ce papier présente RÉVISOR, un ensemble de moteurs d'adaptation fondés sur des opérateurs de révision des croyances. Dans la version actuelle (la 1.0), RÉVISOR contient trois moteurs, respectivement pour la logique propositionnelle, les conjonctions de contraintes linéaires et des algèbres qualitatives. Deux de ces moteurs ont été utilisés par le système TAAABLE (voir p. ex. [4, 13]), un participant aux *computer cooking contests* : RÉVISOR/CLC a été utilisé pour l'adaptation des quantités d'ingrédients et RÉVISOR/QA pour l'adaptation des préparations.

RÉVISOR est fait pour être utilisé à volonté par la communauté. Les retours sont les bienvenus (pour le moment, en envoyant des courriels aux auteurs de cet article). Certaines améliorations sont envisagées pour les versions futures.

RÉVISOR/PL utilise actuellement uniquement les distances de Hamming pondérées sur l'ensemble des interprétations. L'utilisation d'autres distances est considérée actuellement, en incorporant des règles d'adaptation dans ces distances (voir [5]).

RÉVISOR/CLC pourrait être amélioré grâce à un *parser* pour définir plus facilement les cas et les connaissances du domaine.

RÉVISOR/QA travaille actuellement avec une des algèbres qualitatives suivantes : l'algèbre de Allen, INDU et RCC8. L'utiliser avec une autre algèbre demande l'écriture d'un *package* Perl pour définir cette algèbre (cf. les fichiers `Allen.pm`, `INDU.pm` et `RCC8.pm`). Même pour les programmeurs qui apprécient Perl, ce n'est pas quelque chose d'immédiat. Par conséquent, dans une version future de RÉVISOR/QA, des algèbres pourraient être définies dans un format XML plus agréable qui paramétrerait le moteur.

Nous espérons que la vie de RÉVISOR sera longue et fructueuse et que de nouveaux moteurs, en particulier pour de nouveaux formalismes, seront intégrés dans le futur.

Remerciements

Les auteurs sont très reconnaissants envers Roseline Crowley qui a conçu le logo de RÉVISOR (cf. Figure 1). Ils tiennent également à remercier les relecteurs anonymes pour leurs remarques : ils ont fait leur possible pour en tenir compte afin d'améliorer la qualité de l'article et s'en serviront pour alimenter leurs réflexions futures.

Références

[1] A. AAMODT et E. PLAZA : Case-based Reasoning : Foundational Issues, Methodological Variations, and System Approaches. *AI Communications*, 7(1):39–59, 1994.

[2] C. E. ALCHOURRÓN, P. GÄRDENFORS et D. MAKINSON : On the Logic of Theory Change : partial meet functions for contraction and revision. *Journal of Symbolic Logic*, 50:510–530, 1985.

[3] M. BERKELAAR, K. EIKLAND et P. NOTEBAERT : LP_SOLVE a Mixed Integer Linear Programming (MILP) solver. `http://lpsolve.sourceforge.net/5.5/`, 2010.

[4] A. BLANSCHÉ, J. COJAN, V. DUFOUR-LUSSIER, J. LIEBER, P. MOLLI, E. NAUER, H. SKAF-MOLLI et Y. TOUSSAINT : TAAABLE 3 : Adaptation of ingredient quantities and of textual preparations. *In 18h International Conference on Case-Based Reasoning - ICCBR 2010, "Computer Cooking Contest" Workshop Proceedings*, 2010.

[5] J. COJAN et J. LIEBER : Adapter des cas en utilisant un opérateur de révision ou des règles. *In* L. CHOLVY et S. KONIECZNY, éditeurs : *Journée Intelligence Artificielle Fondamentale*, Strasbourg France, 06 2010.

[6] J. COJAN et J. LIEBER : An Algorithm for Adapting Cases Represented in $\mathcal{ALC}$. *In 22th International Joint Conference on Artificial Intelligence (IJCAI-2011)*, 2011.

[7] J. COJAN et J. LIEBER : Belief revision-based case-based reasoning. *In* G. RICHARD, éditeur : *Proceedings of the ECAI-2012 Workshop SAMAI : Similarity and Analogy-based Methods in AI*, pages 33–39, 2012.

[8] A. COMAN et H. MUÑOZ-AVILA : Diverse plan generation by plan adaptation and by first-principles planning : A comparative study. *In* B. DÍAZ-AGUDO et I. WATSON, éditeurs : *Case-Based Reasoning Research and Development (ICCBR 2012)*, volume 7466 de *LNCS*, pages 32–46. Springer, 2012.

[9] J.-F. CONDOTTA, S. KACI, P. MARQUIS et N. SCHWIND : A syntactical approach to qualitative constraint networks merging. *In Logic for*

Programming, Artificial Intelligence, and Reasoning (LPAR), volume 6397 de *Lecture Notes in Computer Science*, pages 233–247. Springer, 2010.

[10] M. DALAL : Investigations into a theory of knowledge base revision : Preliminary report. *In Proceedings of the Seventh National Conference on Artificial Intelligence (AAAI-88)*, pages 475–479, 1988.

[11] V. DUFOUR-LUSSIER, F. LE BER, J. LIEBER et L. MARTIN : Adaptation de cas spatiaux et temporels. *In 20ème atelier Français de Raisonnement à Partir de Cas*, Paris, France, juin 2012.

[12] V. DUFOUR-LUSSIER, F. LE BER, J. LIEBER et L. MARTIN : Adapting Spatial and Temporal Cases. *In* I. WATSON et B. DÍAZ-AGUDO, éditeurs : *International Conference for Case-Based Reasoning*, volume 7466 de *LNCS*, pages 77–91, Lyon, France, septembre 2012. Springer.

[13] V. DUFOUR-LUSSIER, J. LIEBER, E. NAUER et Y. TOUSSAINT : Text Adaptation Using Formal Concept Analysis. *In Proceedings of the 18th International Conference on Case-Based Reasoning (ICCBR-2010)*, volume 6176 de *Lecture Notes in Artificial Intelligence*, pages 96–110. Springer, 2010.

[14] H. KATSUNO et A. MENDELZON : Propositional knowledge base revision and minimal change. *Artificial Intelligence*, 52(3):263–294, 1991.

[15] S. MANZANO, S. ONTAÑÓN et E. PLAZA : Amalgam-based Reuse for Multiagent Case-based Reasoning. *In 19th International Conference on Case Based Reasoning - ICCBR'2011*, volume 6880 de *Lecture Notes in Computer Science*, pages 122–136. Springer, 2011.

[16] M. MINOR, R. BERGMANN, S. GÖRG et K. WALTER : Towards Case-Based Adaptation of Workflows. *In Proceedings of the 7th International Conference on Case-Based Reasoning*, LNCS 4626, pages 421–435, Belfast, 2007. Springer.

[17] D. A. RANDELL, Z. CUI et A. G. COHN : A spatial logic based on regions and connection. *In International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pages 165–176, 1992.

[18] M. M. RICHTER : Introduction. *In* M. LENZ, B. BARTSCH-SPÖRL, H.-D. BURKHARD et S. WESS, éditeurs : *Case-Based Reasoning Technologies. From Foundations to Applications*, LNCS 1400, chapitre 1, pages 1–15. Springer, 1998.

[19] C. K. RIESBECK et R. C. SCHANK : *Inside Case-Based Reasoning*. Lawrence Erlbaum Associates, Inc., Hillsdale, New Jersey, 1989.

[20] J. RUBIN et I. WATSON : Opponent type adaptation for case-based strategies in adversarial games. *In* B. DÍAZ-AGUDO et I. WATSON, éditeurs : *Case-Based Reasoning Research and Development (ICCBR 2012)*, volume 7466 de *LNCS*, pages 357–368. Springer, 2012.

[21] C. SCHWIND : From Inconsistency to Consistency : Knowledge Base Revision by Tableaux Opening. *In* A. KURI-MORALES et G. SIMARI, éditeurs : *Advances in Artificial Intelligence IBERAMIA 2010*, volume 6433 de *LNCS*, pages 120–132. Springer, 2010.