Adaptation de cas propositionnels par réparations fondées sur des connaissances d'adaptation

Gabin Personeni[1,2,3], Alice Hermann[1,2,3] et Jean Lieber[1,2,3]

[1] Université de Lorraine, LORIA, UMR 7503 — 54506 Vandœuvre-lès-Nancy, France, Prénom.Nom.fr
[2] CNRS — 54506 Vandœuvre-lès-Nancy, France
[3] Inria — 54602 Villers-lès-Nancy, France

Résumé

L'adaptation est une étape du raisonnement à partir de cas qui consiste à modifier un cas source représentant un épisode de résolution de problème pour résoudre un nouveau problème, appelé le cas cible. Une approche de l'adaptation consiste à appliquer un opérateur de révision de croyances qui modifie de manière minimale le cas source pour le rendre cohérent avec le cas cible. Une autre approche consiste à utiliser des règles d'adaptation relatives au domaine. Ces deux approches peuvent être combinées : un opérateur de révision paramétré par des règles d'adaptation est introduit. Ce papier présente un algorithme d'adaptation par révision et règles d'adaptation fondé sur la réparation de branches dans des tableaux en logique propositionnelle : quand la conjonction des cas source et cible est incohérente, la méthode des tableaux crée un ensemble de branches, toutes terminant par des conflits, qui sont ensuite réparés à l'aide de règles d'adaptation (modifiant ainsi le cas source). Cet algorithme a été implanté dans l'outil REVISOR/PLAK et des informations sur cette implantation sont présentées.

Mot-clés : raisonnement à partir de cas, adaptation, tableaux sémantiques, logique propositionnelle, révision de croyances, règles d'adaptation

1 Introduction

Le raisonnement à partir de cas (RàPC [17]) est un paradigme de raisonnement fondé sur la réutilisation d'éléments d'expérience appelés cas. Un *cas* est une représentation d'un épisode de résolution de problème, souvent séparé en une partie problème et une partie solution : cette séparation n'est pas formellement nécessaire mais rend plus intuitive la notion de cas. L'entrée d'un système de raisonnement à partir de cas est un cas cible, qui représente un cas en partie renseigné (intuitivement, sa partie problème est bien définie, tandis que sa partie solution ne l'est pas). L'approche la plus courante du RàPC consiste à (1) retrouver un cas source similaire au cas cible, (2) adapter ce cas source de manière à résoudre le cas cible, c'est-à-dire lui ajouter des informations (cela consiste à spécifier la partie solution du cas cible en réutilisant la partie solution du cas source). Des variantes de cette approche et d'autres étapes relatives à cette approche peuvent être trouvées dans, par exemple [1].

Ce papier se concentre sur l'étape (2). En dépit de son importance, cette étape du RàPC a été un peu négligée dans la littérature liée au RàPC, bien qu'elle ait récemment reçu davantage d'attention [14, 12, 7, 18]. En particulier, ce papier détaille une approche de l'adaptation que nous avons introduite il y a quelques années [11] et dont nous avons étudié les aspects (voir [6] pour une synthèse). Cette approche maintenant appelée *adaptation par révision* ou $\dot{+}$*-adaptation* est fondée sur un opérateur de révision de croyances $\dot{+}$. D'après les postulats AGM [2], la révision d'une base de croyances ψ par une seconde μ, notée $\psi \dot{+} \mu$, consiste à modifier ψ de manière minimale pour obtenir ψ' tel que la conjonction de ψ' et μ soit cohérente. Dans ce cas $\psi \dot{+} \mu = \psi' \wedge \mu$. Il existe cependant de multiples façons de « mesurer » des modifications, il existe donc différentes manières de « modifier de manière minimale » une base de croyances : il y a donc de multiples opérateurs de révision satisfaisant les postulats AGM.

Intuitivement, la $\dot{+}$-adaptation consiste à utiliser un opérateur de révision $\dot{+}$ pour modifier le cas source de manière à le rendre cohérent avec le cas cible, et le processus d'adaptation renvoie le résultat de cette révision. Ainsi, implanter un opérateur d'adaptation par révision se réduit à implanter un opérateur de révision. Cette implantation dépend du formalisme utilisé pour le système de RàPC. En particulier, nous avons étudié comment la $\dot{+}$-adaptation peut être implantée pour la logique de description $\mathcal{ALC}$ [5]. Techniquement, nous n'avons pas implanté un opérateur de révision en $\mathcal{ALC}$, puisque cet opérateur ne vérifie pas certains des postulats de [2], mais nous avons implanté un opérateur d'adaptation inspiré des idées de l'adaptation par révision. Cette approche de l'adaptation en $\mathcal{ALC}$ consiste à appliquer des réparations sur des branches de tableaux comportant des conflits. Cette même idée de réparation peut être utilisée pour la révision de ψ par μ, d'après le principe suivant : la méthode des tableaux est appliquée séparément sur ψ et μ, puis les branches cohérentes de ψ sont combinées avec celles de μ. Cela résulte en un ensemble de branches incohérentes (sauf si la conjonction de ψ est μ est cohérente). Ensuite, les parties de ces branches provenant de ψ causant des conflits sont supprimées. Cela implique un affaiblissement de ψ en ψ' de manière à ce que ψ' est cohérent avec μ. Cet algorithme de révision de croyances a été découvert en parallèle par Camilla Schwind [19], qui l'a appliqué sur la logique des propositions avec un nombre fini de variables. Ainsi, ces travaux peuvent être utilisés pour la $\dot{+}$-adaptation de cas propositionnels.

Ce papier propose d'aller plus loin en intégrant aux réparations sur les branches des connaissances d'adaptations relatives au domaine, sous la forme de *règles d'adaptation* (aussi nommées reformulations [13]). Chaque règle d'adaptation représente le fait que, dans un certain contexte, une partie donnée d'un cas peut être remplacée par d'autres éléments. Notre proposition est d'utiliser ces règles pour réparer des conflits dans des branches de tableaux.

Le papier est organisé comme suit : dans un premier temps, des préliminaires introduisent les notions et les notations utilisées au cours de ce papier (Section 2). Dans la Section 3, le processus d'adaptation en RàPC est présenté, soulignant les notions de règles d'adaptation et d'adaptation par révision. Ensuite, l'algorithme d'adaptation utilisant la réparation de branches de tableaux est présenté dans la Section 4. Cette approche a été implantée dans un moteur d'inférence appelé REVISOR/PLAK : ce système et des informations sur l'implantation sont décrits dans la Section 5, ainsi qu'un exemple concret d'adaptation. La Section 6 présente des travaux proches. La Section 7 conclut.

2 Préliminaires

2.1 Logique propositionnelle

Soit $\mathcal{V} = \{a_1, \ldots, a_n\}$ un ensemble de n symboles distincts appelés variables propositionnelles. Une formule propositionnelle construite sur $\mathcal{V}$ est soit une variable a_i, ou d'une de ces formes : $\varphi_1 \wedge \varphi_2$, $\varphi_1 \vee \varphi_2$, $\neg\varphi_1$, $\varphi_1 \Rightarrow \varphi_2$, et $\varphi_1 \Leftrightarrow \varphi_2$, où φ_1 et φ_2 sont deux formules propositionnelles. Soit $\mathcal{L}$ l'ensemble des formules propositionnelles.

Soit $\mathbb{B} = \{\mathtt{V}, \mathtt{F}\}$ un ensemble de deux éléments. Étant donné $x = (x_1, \ldots, x_n) \in \mathbb{B}^n$ et $\varphi \in \mathcal{L}$, $\varphi^x \in \mathbb{B}$ est défini suivant : $a_i^x = x_i$, $(\varphi_1 \wedge \varphi_2)^x = \mathtt{V}$ ssi $\varphi_1^x = \mathtt{V}$ et $\varphi_2^x = \mathtt{V}$, $(\varphi_1 \vee \varphi_2)^x$ ssi $\varphi_1^x = \mathtt{V}$ ou $\varphi_2^x = \mathtt{V}$, $(\neg\varphi_1)^x = \mathtt{V}$ ssi $\varphi_1^x = \mathtt{F}$, $(\varphi_1 \Rightarrow \varphi_2)^x = (\neg\varphi_1 \vee \varphi_2)^x$, et $(\varphi_1 \Leftrightarrow \varphi_2)^x = ((\varphi_1 \Rightarrow \varphi_2) \wedge (\varphi_2 \Rightarrow \varphi_1))^x$.

Pour $\varphi \in \mathcal{L}$, $\mathcal{M}od(\varphi) = \{x \in \mathbb{B}^n \mid \varphi^x = \mathtt{V}\}$ est appelé l'ensemble des modèles de φ. Étant donné $\varphi_1, \varphi_2 \in \mathcal{L}$, $\varphi_1 \models \varphi_2$ si $\mathcal{M}od(\varphi_1) \subseteq \mathcal{M}od(\varphi_2)$, et $\varphi_1 \equiv \varphi_2$ si $\mathcal{M}od(\varphi_1) = \mathcal{M}od(\varphi_2)$. $(\mathcal{L}, \models)$ est appelée la logique propositionnelle avec n variables.

Un littéral ℓ est une formule propositionnelle sous la forme a_i (littéral positif) ou $\neg a_i$ (littéral négatif), où $a_i \in \mathcal{V}$. Si $\ell = \neg a_i$ est un littéral négatif, alors $\neg\ell$ dénote le littéral positif a_i (au lieu de la formule équivalente $\neg\neg a_i$). Une formule est en forme normale disjonctive ou FND si elle est sous la forme d'une disjonction de conjonctions de littéraux. Une formule est en forme normale négative ou FNN si elle contient uniquement les connecteurs $\wedge$, $\vee$ et $\neg$, et si $\neg$ n'apparaît que devant une variable propositionnelle. Toute formule peut être mise en FNN en appliquant, comme des règles de réécriture orientées de gauche à droite, les équivalences suivantes

jusqu'à ce qu'aucune d'entre elles ne soit applicable (pour $\varphi, \varphi_1, \varphi_2 \in \mathcal{L}$) :

$$\varphi_1 \Leftrightarrow \varphi_2 \equiv (\varphi_1 \Rightarrow \varphi_2) \wedge (\varphi_2 \Rightarrow \varphi_1) \qquad \varphi_1 \Rightarrow \varphi_2 \equiv \neg\varphi_1 \vee \varphi_2$$
$$\neg\neg\varphi \equiv \varphi \qquad \neg(\varphi_1 \vee \varphi_2) \equiv \neg\varphi_1 \wedge \neg\varphi_2 \qquad \neg(\varphi_1 \wedge \varphi_2) \equiv \neg\varphi_1 \vee \neg\varphi_2$$

Une formule en FND est nécessairement en FNN (mais l'inverse est faux).

Un ensemble de littéraux $\mathtt{L}$ est souvent assimilé à la conjonction de ses littéraux, par exemple $\{a, \neg b, \neg c\}$ est assimilé à $a \wedge \neg b \wedge \neg c$, et vice-versa. En particulier, $\mathtt{L}$ est dite *cohérente* (ou satisfiable) ssi il n'y a pas de littéral $\ell \in \mathtt{L}$ tel que $\neg\ell \in \mathtt{L}$.

Un impliquant de φ, I est une conjonction ou une disjonction de littéraux, tel que $I \models \varphi$ et I est cohérent. Cependant, puisqu'il existe une dualité entre les impliquants conjonctifs et disjonctifs, seulement les impliquants conjonctifs seront considérés dans ce papier. Un impliquant I de φ est dit premier si et seulement si pour toute conjonction de littéraux C, telle que $C \subset I$, alors $C \not\models \varphi$. C'est-à-dire, un impliquant premier I est minimal. Soit $\mathtt{PI}(\varphi)$ l'ensemble des impliquants premiers de φ. Alors :

$$\varphi \equiv \bigvee_{I \in \mathtt{PI}(\varphi)} I$$

Un algorithme calculant les impliquants premiers d'une formule φ est décrit en [20].

2.2 La méthode des tableaux sémantiques

En logique propositionnelle, la méthode des tableaux sémantiques permet de mettre une formule en FND. On procède comme suit : φ est d'abord mise en FNN. Le tableau de φ est un arbre dont la racine est φ. Deux règles permettent de développer les formules dans le tableau :

$$\frac{\varphi_1 \wedge \varphi_2}{\begin{array}{c}\varphi_1\\ \varphi_2\end{array}}\ (\wedge) \qquad \frac{\varphi_1 \vee \varphi_2}{\varphi_1 \mid \varphi_2}\ (\vee)$$

Lorsqu'une conjonction de deux formules $\varphi_1 \wedge \varphi_2$ apparaît dans une branche, on applique la règle $(\wedge)$ en ajoutant à cette branche les formules φ_1 et φ_2. Lorsqu'une disjonction de deux formules $\varphi_1 \vee \varphi_2$ apparaît dans une branche, on applique la règle $(\vee)$ en créant deux nouvelles branches contenant respectivement les formules φ_1 et φ_2. Quand une branche contient un littéral ℓ et son opposé $\neg\ell$, cette branche est fermée, et plus aucune règle n'est appliquée aux formules de cette branche. On dit alors qu'il y a un conflit sur ℓ. La procédure se termine quand aucune règle ne peut plus être appliquée. Une formule φ est mise en FND en prenant la disjonction des branches du tableau de φ, où chaque branche est la conjonction des littéraux qu'elle contient.

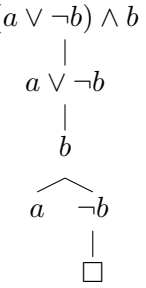

FIGURE 1 – Tableau pour la formule $(a \vee \neg b) \wedge b$.

La figure 1 présente un exemple d'application de la méthode des tableaux sur la formule $\varphi = (a \vee \neg b) \wedge b$. La règle $(\wedge)$ est d'abord appliquée sur la conjonction entre $(a \vee \neg b)$ et b, puis la règle $(\vee)$ est appliquée sur la disjonction entre a et $\neg b$. La première branche contient les littéraux a et b, et la seconde les littéraux $\neg b$ et b et est donc fermée. Une écriture de φ en FND est donc $(a \wedge b)$.

2.3 Distances

Soit $\mathcal{U}$ un ensemble. Dans ce papier, 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 d'une fonction de distance ne sont pas requises dans ce papier). Étant donné $A, B \in 2^{\mathcal{U}}$ et $y \in \mathcal{U}$, les notations suivantes sont utilisées :

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

La distance de Hamming sur des interprétations en logique propositionnelle, d_H, est définie par $d_H(x, y) = |\{i \mid i \in \{1, 2, \ldots, n\}, x_i \neq y_i\}|$, pour tout $x, y \in \mathbb{B}^n$. C'est-à-dire, $d_H(x, y)$ est le nombre de changements d'affectations de valeurs de variables nécessaires pour transformer x en y. On appellera *flip* sur a_i le changement d'affectation de valeur de a_i.

2.4 Révision de croyances

Soit ψ les croyances d'un agent à propos du monde, exprimées dans une logique $(\mathcal{L}, \models)$. Cet agent est confronté à de nouvelles croyances exprimées par $\mu \in \mathcal{L}$. Ces nouvelles croyances sont supposées êtres non révisables, tandis que les anciennes croyances ψ peuvent être changées. Si μ n'est pas contradictoire avec ψ, (c'est-à-dire $\psi \wedge \mu$ est cohérent), alors les nouvelles croyances sont simplement ajoutés aux anciennes. Sinon, d'après le *principe de changement minimum* [2], ψ doit être modifié minimalement en $\psi' \in \mathcal{L}$ tel que $\psi' \wedge \mu$ soit cohérent, et que la révision de ψ par μ, notée $\psi \dot{+} \mu$, soit cette conjonction.

Il existe différentes manières de mesurer les modifications de croyances, et donc, il existe plusieurs opérateurs de révision $\dot{+}$. Dans [2], un ensemble de postulats qu'un opérateur de révision est supposé vérifier a été défini. Dans [9], ces postulats ont été reformulés en logique propositionnelle et une famille d'opérateurs de révision fondés sur des distances a été définie comme il suit. Soit d une distance sur $\mathcal{U} = \mathbb{B}^n$. Soit ψ et μ, deux formules. La révision de ψ par μ selon l'opérateur $\dot{+}^d$ ($\psi \dot{+}^d \mu$) est une formule dont les modèles sont les modèles de μ les plus proches des modèles de ψ d'après d (le changement d'une interprétation x vers une interprétation y est mesurée par $d(x, y)$). Formellement, $\psi \dot{+}^d \mu$ est tel que :

$$\mathcal{M}od(\psi \dot{+}^d \mu) = \{y \in \mathcal{M}od(\mu) \mid d(\mathcal{M}od(\psi), y) = d(\mathcal{M}od(\psi), \mathcal{M}od(\mu))\}$$

On peut noter que cette définition ne définit $\psi \dot{+}^d \mu$ à qu'à l'équivalence logique près, cependant ceci est suffisant puisqu'un opérateur de révision doit satisfaire le principe de non pertinence de la syntaxe (c'est un des postulats défini dans [9] : si $\psi_1 \equiv \psi_2$ et $\mu_1 \equiv \mu_2$ alors $\psi_1 \dot{+} \mu_1 \equiv \psi_2 \dot{+} \mu_2$).

2.5 Algorithme A*

A* est un algorithme de recherche de type meilleur-d'abord fondé sur une heuristique [15]. Il est utilisé pour effectuer des recherches dans des espaces d'états, dans lesquels il existe pour chaque état un nombre fini de transitions vers d'autres états, et où le coût d'un chemin est additif (le coût d'un chemin est la somme des coûts des transitions qu'il contient). Un problème de recherche est donné par un ensemble fini d'états initiaux et par un objectif définissant les conditions auxquelles un état doit répondre pour être considéré comme final. Étant donné un état $\mathtt{E}$, soit $\mathcal{F}^*(\mathtt{E})$ le minimum des coûts des chemins d'un état initial à un état final passant par $\mathtt{E}$. $\mathcal{F}^*(\mathtt{E}) = \mathcal{G}^*(\mathtt{E}) + \mathcal{H}^*(\mathtt{E})$ où $\mathcal{G}^*(\mathtt{E})$ (resp., $\mathcal{H}^*(\mathtt{E})$) est le minimum des coûts des chemins depuis un état initial vers $\mathtt{E}$ (resp., depuis $\mathtt{E}$ vers un état final). En général, $\mathcal{F}^*$ est inconnu, et est approchée par une fonction $\mathcal{F} = \mathcal{G} + \mathcal{H}$. $\mathcal{F}$ est *admissible* si $\mathcal{G} \geq \mathcal{G}^*$ et $\mathcal{H} \leq \mathcal{H}^*$. Si $\mathcal{F}$ est admissible, alors la procédure A* est optimale : s'il existe une solution trouvée par A* (un chemin depuis un état initial vers un état final), alors cette solution a un coût minimal. La procédure A* consiste à rechercher dans l'espace d'états, en commençant depuis les états initiaux, en augmentant $\mathcal{F}(\mathtt{E})$: parmi l'ensemble des états initiaux, les minima pour $\mathcal{F}$ sont privilégiés. $\mathcal{G}(\mathtt{E})$ est le coût minimal des chemins atteignant $\mathtt{E}$ déjà générés, ainsi, $\mathcal{G} \geq \mathcal{G}^*$. La principale difficulté est de trouver une fonction $\mathcal{H}$ admissible (la fonction constante 0 est une heuristique $\mathcal{H}$ admissible, cependant, plus la différence entre $\mathcal{H}$ et $\mathcal{H}^*$ est faible, plus la recherche est rapide).

3 L'adaptation dans le raisonnement à partir de cas

Soit $(\mathcal{L}, \models)$ la logique dans laquelle les bases de connaissances (appelées *CBR Knowledge containers* en anglais) d'une application du RàPC sont définies. Un cas source, $\texttt{Source} \in \mathcal{L}$ est un cas de la base de cas. Souvent, ce cas représente une expérience spécifique : $\mathcal{M}od(\texttt{Source})$ est un singleton. Cependant cette hypothèse n'est pas formellement nécessaire (bien que cela ait un impact positif sur la complexité des algorithmes). Un cas cible $\texttt{Cible} \in \mathcal{L}$ représente un problème à résoudre. Cela signifie qu'il manque des informations à propos de Cible, la résolution de Cible conduit donc à y ajouter des informations. Ainsi, l'adaptation de Source pour résoudre Cible consiste à construire une formule $\texttt{CibleCompletée} \in \mathcal{L}$ qui précise $\texttt{Cible} : \texttt{CibleCompletée} \models \texttt{Cible}$.

Pour réaliser l'adaptation, des connaissances du domaines $\texttt{CD} \in \mathcal{L}$ peuvent être utilisées. Ainsi, le processus d'adaptation possède la signature suivante :

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

La paire $(\texttt{Source}, \texttt{Cible})$ est appelée le problème d'adaptation (CD est supposé être fixe). Cela ne suffit pas à spécifier le processus d'adaptation. Plusieurs approches sont introduites dans la littérature concernant le RàPC. Deux d'entre elles sont présentées ci-dessous, suivie d'une approche les combinant.

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

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

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

L'adaptation par règles. C'est une approche générale de l'adaptation qui repose sur des connaissances d'adaptation propres au domaine sous la forme d'un ensemble CA de règles d'adaptation (voir, par exemple, [13], où les règles d'adaptation sont appelées reformulations). Une règle d'adaptation $\texttt{R} \in \texttt{CA}$, lorsqu'elle est applicable au problème $(\texttt{Source}, \texttt{Cible})$, associe Source à CibleCompletée qui précise Cible. Les règles d'adaptation peuvent êtres composées, ou chaînées : si $\texttt{R}_1, \texttt{R}_2, \ldots, \texttt{R}_q \in \texttt{CA}$ sont telles qu'il existe $q+1$ cas $\texttt{C}_0, \texttt{C}_1, \ldots, \texttt{C}_q$ vérifiant :

- $\texttt{C}_0 = \texttt{Source}$,
- $\texttt{C}_q$ précise Cible ($\texttt{C}_q \models \texttt{Cible}$),
- pour chaque $i \in \{1, \ldots, q\}$, $\texttt{R}_i$ est applicable à $(\texttt{C}_{i-1}, \texttt{C}_i)$ et associe $\texttt{C}_{i-1}$ à $\texttt{C}_i$,

alors $\texttt{C}_q = \texttt{CibleCompletée}$ est le résultat de l'adaptation de Source pour résoudre Cible. La séquence $\texttt{R}_1; \texttt{R}_2; \ldots; \texttt{R}_q$ est appelée un *chemin d'adaptation*.

Étant donné un problème d'adaptation $(\texttt{Source}, \texttt{Cible})$, il peut exister plusieurs chemins d'adaptation pour le résoudre. Pour choisir parmi ces chemins, une fonction de coût est introduite : $\texttt{coût} : \texttt{R} \in \texttt{CA} \mapsto \texttt{coût}(\texttt{R}) > 0$. Le coût d'un chemin d'adaptation est la somme des coûts de ses règles d'adaptation.

Dans ce papier, une règle d'adaptation R est définie par deux ensembles de littéraux, gauche et droite, et est dénotée par $\texttt{R} = \texttt{gauche} \rightsquigarrow \texttt{droite}$. Soit $(\texttt{Source}, \texttt{Cible})$ un problème d'adaptation. Plusieurs cas sont à considérer :

- Si Source est une conjonction de littéraux représenté par un ensemble de littéraux L, alors R est applicable sur Source si $\texttt{gauche} \subseteq \texttt{L}$. Dans ce cas :

$$\texttt{R}(\texttt{Source}) = \texttt{R}(\texttt{L}) = (\texttt{L} \setminus \texttt{gauche}) \cup \texttt{droite}$$

- Si Source est en FND, tel que $\texttt{Source} = \bigvee_i \texttt{L}_i$, où les $\texttt{L}_i$ sont des conjonctions de littéraux, R est applicable sur Source si elle est applicable sur au moins un $\texttt{L}_i$. Alors :

$$\texttt{R}(\texttt{Source}) = \bigvee_i \begin{cases} \texttt{R}(\texttt{L}_i) & \text{si R est applicable sur } \texttt{L}_i, \\ \texttt{L}_i & \text{sinon.} \end{cases}$$

– Si $\texttt{Source}$ n'est pas en FND, il est alors remplacé par une formule équivalente en FND.

Étant donnée une règle d'adaptation $\texttt{R} = \texttt{gauche} \rightsquigarrow \texttt{droite}$, on note $\texttt{réparations}(\texttt{R}) = \texttt{gauche} \setminus \texttt{droite}$, c'est-à-dire l'ensemble des littéraux nécessaire à l'application de $\texttt{R}$, et qui sont supprimés par l'application de $\texttt{R}$. Chaque règle d'adaptation doit être telle que $\texttt{réparations}(\texttt{R}) \neq \varnothing$.

L'adaptation par révision et par règles d'adaptation. Considérons la distance suivante sur $\mathcal{U} = \mathbb{B}^n$ (pour $x, y \in \mathcal{U}$) :

$$\delta_{\texttt{CA}}(x, y) = \inf\{\texttt{coût}(p) \mid p : \text{chemin d'adaptation de } x \text{ vers } y \text{ fondé sur des règles de } \texttt{CA}\}$$

(x et y sont des interprétations assimilées à des conjonctions de littéraux). Par convention, $\inf \varnothing = +\infty$ et donc, s'il n'existe aucun chemin d'adaptation de x vers y, alors $\delta_{\texttt{CA}}(x, y) = +\infty$ et vice-versa. Sinon, il peut être montré que l'infimum est toujours atteint et donc, $\delta_{\texttt{CA}}(x, y) = 0$ ssi $x = y$ correspondant à un chemin d'adaptation vide.

Il a été montré [6] que l'adaptation par règles d'adaptation peut être simulée par l'adaptation par révision sans connaissances du domaine (c'est-à-dire $\texttt{CD}$ est une tautologie) et avec l'opérateur de révision $\dot{+}^{\delta_{\texttt{CA}}}$. Quand l'adaptation par règles échoue (il n'existe aucun chemin d'adaptation de $\texttt{Source}$ vers $\texttt{Cible}$), la $\dot{+}^{\delta_{\texttt{CA}}}$-adaptation donne comme résultat $\texttt{CibleCompletée}$ équivalente à $\texttt{Cible}$ (aucune information n'est ajoutée).

L'échec de l'adaptation par règles est dû au fait que les règles d'adaptation ne peuvent pas être composées de manière à résoudre le problème d'adaptation. Afin d'obtenir un opérateur qui retourne toujours un résultat, $2n$ règles d'adaptation sont ajoutées, une pour chaque littéral : étant donné un littéral ℓ, le *flip* de ce littéral ℓ est la règle d'adaptation $\texttt{F}_\ell = \ell \rightsquigarrow \top$, où $\top$ est la conjonction de littéraux vide [1]. On peut noter qu'un coût est associé à chaque *flip* de littéral et qu'il est possible que ces coûts soient tels que $\texttt{coût}(\texttt{F}_{\neg\ell}) \neq \texttt{coût}(\texttt{F}_\ell)$.

On considère une seconde distance $d_{\texttt{CA}}$ sur $\mathcal{U} = \mathbb{B}^n$, définie, pour $x, y \in \mathcal{U}$, par :

$$d_{\texttt{CA}}(x, y) = \inf \left\{ \texttt{coût}(p) \;\middle|\; \begin{array}{c} p : \text{chemin d'adaptation de } x \text{ vers } y \\ \text{fondé sur les règles de } \texttt{CA} \text{ et les } \textit{flips} \text{ de littéraux} \end{array} \right\}$$

Soit $\dot{+}_{\texttt{CA}} = \dot{+}^{d_{\texttt{CA}}}$. La $\dot{+}_{\texttt{CA}}$-adaptation est une adaptation par révision utilisant des règles d'adaptation, combinant ainsi l'adaptation par révision et l'adaptation par règles d'adaptation. La $\dot{+}_{\texttt{CA}}$-adaptation prend également en compte les connaissances du domaine.

Il peut être noté que si $\texttt{CA} = \varnothing$ et $\texttt{coût}(\texttt{F}_\ell) = 1$ pour chaque littéral ℓ alors $d_{\texttt{CA}} = d_H$. De plus, il est supposé que :

$$\text{Pour tout } \texttt{R} \in \texttt{CA}, \texttt{coût}(\texttt{R}) \leq \sum_{\ell \in \texttt{réparations}(\texttt{R})} \texttt{coût}(\texttt{F}_\ell) \tag{1}$$

Cette supposition ne cause aucune perte de généralité : si une règle d'adaptation ne vérifie pas (1), alors elle n'apparaît dans aucun chemin d'adaptation optimal, puisque l'application des *flips* de $\texttt{réparations}(R)$ serait moins coûteuse que l'application de R elle-même. La combinaison de l'adaptation par règles et de la $\dot{+}$-adaptation présente plusieurs avantages par rapport à ces deux méthodes appliquées séparement. Contrairement à $\dot{+}^{\delta_{\texttt{CA}}}$ n'utilisant que les règles d'adaptation , $\dot{+}_{\texttt{CA}}$ trouve toujours une solution en utilisant les *flips* quand les règles ne suffisent pas. L'opérateur $\dot{+}_{\texttt{CA}}$ permet de prendre en compte à la fois les connaissances du domaine (ce qui n'est pas possible avec $\dot{+}^{\delta_{\texttt{CA}}}$) et les connaissances d'adaptation.

4 Algorithme d'adaptation fondé sur des réparations dans des tableaux

L'algorithme d'adaptation est fondé sur la révision du cas source par le cas cible, chacun dans le contexte des connaissances du domaine. Ainsi, l'algorithme effectue la révision d'une formule ψ par une formule μ,

1. Pour être tout à fait exact, l'application du *flip* de ℓ sur un ensemble de littéraux $\texttt{L}$ consiste à faire $\texttt{F}_\ell \wedge \neg\ell$. Cependant, comme nous nous servirons du *flip* pour réparer un conflit entre deux ensembles de littéraux $\texttt{L}$ et $\texttt{M}$ avec $\ell \in \texttt{L}$ et $\neg\ell \in \texttt{M}$, faire la conjonction avec $\neg\ell$ (déjà contenu dans $\texttt{M}$) apparaît comme inutile.

avec :

$$\psi = \texttt{CD} \wedge \texttt{Source} \qquad \mu = \texttt{CD} \wedge \texttt{Cible}$$
$$\text{et donc :} \qquad \texttt{CibleCompletée} = \psi \dot{+}_{\texttt{CA}} \mu$$

Cette section présente l'algorithme (Section 4.1), qui utilise une heuristique $\mathcal{H}$, qui est définie dans la Section 4.2. La preuve de son admissibilité y est également détaillée. La Section 4.3 détaille un exemple et la Section 4.4 étudie la terminaison et la complexité de l'algorithme.

4.1 Algorithme

Soit φ une formule et $\texttt{branches}(\varphi)$, l'ensemble des branches de tableaux de φ, aussi impliquants de φ, sous la forme d'un ensemble d'ensembles de littéraux. La fonction `branches` est définie comme suit :

- Pour tout litéral ℓ : $\texttt{branches}(\ell) = \{\{\ell\}\}$;
- Pour toute formule φ_1 et φ_2 :
  $\texttt{branches}(\varphi_1 \vee \varphi_2) = \texttt{branches}(\varphi_1) \cup \texttt{branches}(\varphi_2)$
  $\texttt{branches}(\varphi_1 \wedge \varphi_2) = \{B_1 \cup B_2 \mid B_1 \in \texttt{branches}(\varphi_1), B_2 \in \texttt{branches}(\varphi_2)\}$.

Pour assurer la non pertinence de la syntaxe, nous introduisons une fonction `min-branches`, qui, à partir du résultat de $\texttt{branches}(\varphi)$ renvoie l'ensemble des impliquants premiers de φ.

Pour tout ensemble de littéraux $\texttt{L}$, soit $\texttt{L}^\neg = \{\neg\ell \mid \ell \in \texttt{L}\}$. Par exemple, si $\texttt{L} = \{a, \neg b, \neg c\}$ alors $\texttt{L}^\neg = \{\neg a, b, c\}$. $\texttt{L}$ est dit cohérent ssi $\texttt{L} \cap \texttt{L}^\neg = \varnothing$.

L'algorithme est fondé sur A*, où un état est défini par une paire ordonnée $(\texttt{L}, \texttt{M})$ d'ensembles de littéraux cohérents. Étant donné les formules propositionnelles ψ et μ, l'ensemble des états initiaux est :

$$\texttt{min-branches}(\psi) \times \texttt{min-branches}(\mu)$$

Un état final est un état $(\texttt{L}, \texttt{M})$ tel que $\texttt{L} \cap \texttt{M}^\neg = \varnothing$ (ce qui équivaut à ce que $\texttt{L} \cup \texttt{M}$ soit cohérent). Si $(\texttt{L}, \texttt{M})$ n'est pas un état final, alors il existe $\ell \in \texttt{L}$ tel que $\ell \in \texttt{L} \cap \texttt{M}^\neg$: il y a un conflit sur ℓ.

Les transitions d'un état à un autre sont définies comme suit. Il existe une transition σ d'un état $x = (\texttt{L}_x, \texttt{M}_x)$ à un état $y = (\texttt{L}_y, \texttt{M}_y)$ si $\texttt{M}_x = \texttt{M}_y$ et :

- il existe un littéral $\ell \in \texttt{L}_x$ tel que $\texttt{L}_y = \texttt{F}_\ell(\texttt{L}_x)$ ou
- il existe une règle d'adaptation $\texttt{R}$ telle que $\texttt{R}$ est applicable sur $\texttt{L}_x$ et $\texttt{L}_y = \texttt{R}(\texttt{L}_x)$.

Le coût d'une transition σ est le coût de la règle (*flip* ou règle d'adaptation) qu'elle utilise.

En utilisant l'algorithme A*, il est possible de déterminer l'ensemble des séquences de transitions les moins coûteuses conduisant à un état final. Ceci signifie que l'algorithme ne s'arrête pas après avoir trouvé la première solution optimale, mais après avoir trouvé toute les solutions optimales (c'est-à-dire toutes les solutions avec le même coût minimal). L'algorithme détaillé est présenté en Algorithme 1 et est détaillé ci-après.

L'algorithme crée d'abord les états initiaux (ligne 3). Pour ce faire, la fonction `min-branches` utilisant la méthode des tableaux sémantiques est appliquée. Pour chaque état initial $\texttt{E}_0$, $\mathcal{G}(\texttt{E}_0) = 0$, c'est-à-dire le coût pour atteindre $\texttt{E}_0$ est de 0. Dans le cas où $\mathcal{G}$ n'a pas de valeur assignée pour un état $\texttt{E}$, $\mathcal{G}(\texttt{E})$ prend la valeur $+\infty$, signifiant qu'il n'existe pas de chemin connu depuis un état initial vers $\texttt{E}$. L'algorithme trouve ensuite un état $\texttt{E}_c$ qui minimise $\mathcal{F}(\texttt{E}_c) = \mathcal{G}(\texttt{E}_c) + \mathcal{H}(\texttt{E}_c)$, c'est-à-dire le coût pour atteindre $\texttt{E}_c$ depuis un état initial plus le coût heuristique pour atteindre un état final depuis $\texttt{E}_c$ (ligne 10). Pour chaque règle ou *flip* σ applicable sur $\texttt{E}_c$, l'état $\texttt{E}_d$ est crée tel que $\texttt{E}_c \xrightarrow{\sigma} \texttt{E}_d$. Si $\mathcal{G}(\texttt{E}_d) > \mathcal{G}(\texttt{E}_c) + \texttt{coût}(\sigma)$, c'est-à-dire qu'il n'existe aucun autre chemin connu dont le coût est inférieur ou égal pour atteindre $\texttt{E}_d$, alors $\mathcal{G}(\texttt{E}_d)$ prend pour valeur $\mathcal{G}(\texttt{E}_c) + \texttt{coût}(\sigma)$. Chaque $\texttt{E}_d$ ainsi généré est ajouté à l'ensemble des états à explorer, tandis que $\texttt{E}_c$ en est retiré (lignes 15 à 20). L'algorithme répète cette étape jusqu'à ce qu'il trouve un état final $\texttt{E}_f$ minimisant $\mathcal{F}(\texttt{E}_f)$ (lignes 11 à 13). Ensuite l'algorithme continue de la même manière mais ignore tous les états $\texttt{E}_c$ tels que $\mathcal{F}(\texttt{E}_c) > \mathcal{F}(\texttt{E}_f)$, puisqu'ils ne pourront pas conduire à une solution moins coûteuse (ligne 9). Finalement, tous les états finaux $\texttt{E}_f$ minimisant $\mathcal{F}(\texttt{E}_f)$ sont renvoyés sous la forme de branches de tableaux (ligne 22), c'est-à-dire que, pour chaque état $\texttt{E}_f$ défini par $(\texttt{L}_f, \texttt{M}_f)$, la branche contenant les littéraux $\texttt{L}_f \cup \texttt{M}_f$ est renvoyée.

réviser(ψ, μ, CA, coût)
Entrées : ψ et μ, deux formules propositionnelles en FNN. ψ sera révisée par μ.
CA, l'ensemble des règles d'adaptation pour effectuer l'adaptation.
coût, une fonction associant à chaque *flip* ou règle d'adaptation un coût strictement positif.
Sortie : $\psi \dot{+}_{\mathrm{CA}} \mu$ en FND
début
// états est initialisé à l'ensemble des états initiaux.
états $\leftarrow$ min-branches(ψ) $\times$ min-branches(μ)
$\mathcal{G}(\mathrm{E})$ a pour valeur $+\infty$ par défaut.
$\mathcal{G}(\mathrm{E}) \leftarrow 0$ pour chaque $\mathrm{E} \in$ états
$\mathcal{F}(\mathrm{E})$ a pour valeur $\mathcal{G}(\mathrm{E}) + \mathcal{H}(\mathrm{E})$
Solutions $\leftarrow \varnothing$
coûtSolution $\leftarrow +\infty$
tant que $\{\mathrm{E} \mid \mathrm{E} \in \text{états}, \mathcal{F}(\mathrm{E}) \leq \text{coûtSolution}\} \neq \varnothing$ **faire**
// $(\mathrm{L}_c, \mathrm{M}_c)$ est l'état courant.
$(\mathrm{L}_c, \mathrm{M}_c) \leftarrow$ un $\mathrm{E} \in$ états qui minimise $\mathcal{F}(\mathrm{E})$
si $(\mathrm{L}_c \cap \mathrm{M}_c^{\neg}) = \varnothing$ **alors**
Solutions $\leftarrow$ Solutions $\cup \{\mathrm{L}_c \cup \mathrm{M}_c\}$
coûtSolution $\leftarrow \mathcal{F}((\mathrm{L}_c, \mathrm{M}_c))$
sinon
pour chaque $\sigma \in \mathrm{CA} \cup \{\mathrm{F}_\ell$ pour chaque littéral $\ell\}$ tel que σ est applicable sur L_c **faire**
états $\leftarrow$ états $\cup \{(\sigma(\mathrm{L}_c),\ \mathrm{M}_c)\}$
$\mathcal{G}((\sigma(\mathrm{L}_c),\ \mathrm{M}_c)) \leftarrow \min(\mathcal{G}((\sigma(\mathrm{L}_c),\ \mathrm{M}_c)), \mathcal{G}((\mathrm{L}_c, \mathrm{M}_c)) + \text{coût}(\sigma))$
fin
fin
états $\leftarrow$ états $\setminus \{(\mathrm{L}_c, \mathrm{M}_c)\}$
fin
retourne Solutions
fin

Algorithme 1 : Algorithme d'adaptation fondé sur des réparations dans des branches de tableaux.

4.2 Heuristique

La fonction $\mathcal{H}$ utilisée dans notre algorithme est définie par : pour $\mathrm{E} = (\mathrm{L}, \mathrm{M})$, $\mathcal{H}(\mathrm{E}) = \mathrm{ERC}(\mathrm{L} \cap \mathrm{M}^{\neg})$ où ERC est défini comme suit (ERC signifiant *Estimated Repair Cost*) :

$$\mathrm{ERC}(\{\ell\}) = \min\{\text{coût}(\mathrm{F}_\ell)\} \cup \left\{ \frac{\text{coût}(\mathrm{R})}{|\text{réparations}(\mathrm{R})|} \;\middle|\; \begin{array}{c} \mathrm{R} \in \mathrm{CA}, \text{ tel que} \\ \ell \in \text{réparations}(\mathrm{R}) \end{array} \right\} \quad \text{pour tout littéral } \ell$$

$$\mathrm{ERC}(\mathrm{L}) = \sum_{\ell \in \mathrm{L}} \mathrm{ERC}(\{\ell\}) \quad \text{pour tout ensemble de littéraux L}$$

Proposition 1. *$\mathcal{H}$ est admissible.*

Preuve. Une heuristique $\mathcal{H}$ est cohérente si elle vérifie $\mathcal{H}(\mathrm{E}_x) \leq c(\mathrm{E}_x, \mathrm{E}_y) + \mathcal{H}(\mathrm{E}_y)$, où $c(\mathrm{E}_x, \mathrm{E}_y)$ est le coût minimal des chemins de E_x à E_y. Une heuristique cohérente est également admissible [15]. La cohérence de $\mathcal{H}$ peut être prouvée par récurrence, comme montré ci-dessous. Dans cette preuve la notation $X \setminus Y$ est utilisée, où X et Y sont deux ensembles de littéraux. $X \setminus Y = X \cap \overline{Y}$, où $\overline{Y} = \{\ell \mid \ell \in \mathcal{V} \cup \mathcal{V}^{\neg}, \ell \notin Y\}$.

Initialisation. Soit σ une transition de $\mathrm{E}_x = (\mathrm{L}_x, \mathrm{M})$ à $\mathrm{E}_y = (\mathrm{L}_y, \mathrm{M})$: $\mathrm{E}_x \xrightarrow{\sigma} \mathrm{E}_y$. Prouver l'initialisation revient à prouver que : $\mathcal{H}(\mathrm{E}_x) \leq c(\mathrm{E}_x, \mathrm{E}_y) + \mathcal{H}(\mathrm{E}_y)$, où $c(\mathrm{E}_x, \mathrm{E}_y) = \text{coût}(\sigma)$. Deux cas sont considérés :

– σ est un *flip* $\mathtt{F}_\ell$. Dans ce cas $\ell \in \mathtt{L}_x$ et $\mathtt{L}_y = \mathtt{L}_x \setminus \{\ell\}$. Ainsi :
$\mathcal{H}(\mathtt{E}_y) = \mathtt{ERC}((\mathtt{L}_x \setminus \{\ell\}) \cap \mathtt{M}^\neg) = \mathtt{ERC}((\mathtt{L}_x \cap \mathtt{M}^\neg) \setminus \{\ell\})$
$\mathcal{H}(\mathtt{E}_y) = \mathtt{ERC}((\mathtt{L}_x \cap \mathtt{M}^\neg) \setminus \{\ell\}) \geq \mathtt{ERC}(\mathtt{L}_x \cap \mathtt{M}^\neg) - \mathtt{ERC}(\{\ell\}) = \mathcal{H}(\mathtt{E}_x) - \mathtt{ERC}(\{\ell\})$.
Alors, $\mathcal{H}(\mathtt{E}_x) \leq \mathcal{H}(\mathtt{E}_y) + \mathtt{ERC}(\{\ell\})$ ce qui implique $\mathcal{H}(\mathtt{E}_x) \leq \mathtt{coût}(\mathtt{F}_\ell) + \mathcal{H}(\mathtt{E}_y)$.
– σ est une règle d'adaptation $\mathtt{R} = \mathtt{gauche} \rightsquigarrow \mathtt{droite}$, avec : $\mathtt{gauche} \subseteq \mathtt{L}_x$, puisque $\mathtt{R}$ est applicable sur $\mathtt{E}_x$; $\mathtt{réparations}(\mathtt{R}) = \mathtt{gauche} \setminus \mathtt{droite} \neq \varnothing$; et $\mathtt{L}_y = (\mathtt{L}_x \setminus \mathtt{gauche}) \cup \mathtt{droite}$.
$\mathcal{H}(S_x) - \mathcal{H}(S_y) = \mathtt{ERC}(\mathtt{L}_x \cap \mathtt{M}^\neg) - \mathtt{ERC}(\mathtt{L}_y \cap \mathtt{M}^\neg) \leq \mathtt{ERC}((\mathtt{L}_x \cap \mathtt{M}^\neg) \setminus (\mathtt{L}_y \cap \mathtt{M}^\neg))$.
On prouve que $(\mathtt{L}_x \cap \mathtt{M}^\neg) \setminus (\mathtt{L}_y \cap \mathtt{M}^\neg) \subseteq \mathtt{réparations}(\mathtt{R})$:

$$\begin{aligned}
(\mathtt{L}_x \cap \mathtt{M}^\neg) \setminus (\mathtt{L}_y \cap \mathtt{M}^\neg) &= (\mathtt{L}_x \cap \mathtt{M}^\neg) \setminus ((\mathtt{L}_x \setminus \mathtt{gauche}) \cup \mathtt{droite} \cap \mathtt{M}^\neg) \text{ par def. de } \mathtt{L}_y \\
&= \mathtt{L}_x \cap \mathtt{M}^\neg \cap \overline{(\mathtt{L}_x \cap \overline{\mathtt{gauche}}) \cup \mathtt{droite}) \cap \mathtt{M}^\neg} \\
&= \mathtt{L}_x \cap \mathtt{M}^\neg \cap (((\overline{\mathtt{L}_x} \cup \mathtt{gauche}) \cap \overline{\mathtt{droite}}) \cup \overline{\mathtt{M}^\neg}) \\
&= (\mathtt{L}_x \cap \mathtt{M}^\neg \cap ((\overline{\mathtt{L}_x} \cup \mathtt{gauche}) \cap \overline{\mathtt{droite}})) \cup (\underbrace{\mathtt{L}_x \cap \mathtt{M}^\neg \cap \overline{\mathtt{M}^\neg}}_{\varnothing}) \\
&= (\underbrace{\mathtt{L}_x \cap \mathtt{M}^\neg \cap \overline{\mathtt{L}_x} \cap \overline{\mathtt{droite}}}_{\varnothing}) \cup (\mathtt{L}_x \cap \mathtt{M}^\neg \cap \mathtt{gauche} \cap \overline{\mathtt{droite}}) \\
&= \mathtt{L}_x \cap \mathtt{M}^\neg \cap (\mathtt{gauche} \setminus \mathtt{droite}) \\
&= \mathtt{L}_x \cap \mathtt{M}^\neg \cap \mathtt{réparations}(\mathtt{R}) \subseteq \mathtt{réparations}(\mathtt{R})
\end{aligned}$$

Par conséquent, $\mathcal{H}(S_x) - \mathcal{H}(S_y) \leq \mathtt{ERC}((\mathtt{L}_x \cap \mathtt{M}^\neg) \setminus (\mathtt{L}_y \cap \mathtt{M}^\neg)) \leq \mathtt{ERC}(\mathtt{réparations}(\mathtt{R}))$, et $\mathcal{H}(S_x) \leq \mathtt{ERC}(\mathtt{réparations}(\mathtt{R})) + \mathcal{H}(S_y)$. Il est donc suffisant de prouver que $\mathtt{ERC}(\mathtt{réparations}(\mathtt{R})) \leq \mathtt{coût}(\mathtt{R})$, ce qui est fait ci-après.
Pour $\ell \in \mathtt{réparations}(\mathtt{R})$, $\mathtt{ERC}(\{\ell\}) \leq \dfrac{\mathtt{coût}(\mathtt{R})}{|\mathtt{réparations}(\mathtt{R})|}$ par définition de $\mathtt{ERC}$. Donc :

$$\begin{aligned}
\mathtt{ERC}(\mathtt{réparations}(\mathtt{R})) &= \sum_{\ell \in \mathtt{réparations}(\mathtt{R})} \mathtt{ERC}(\{\ell\}) \\
&\leq \sum_{\ell \in \mathtt{réparations}(\mathtt{R})} \frac{\mathtt{coût}(\mathtt{R})}{|\mathtt{réparations}(\mathtt{R})|} = \mathtt{coût}(\mathtt{R})
\end{aligned}$$

Ainsi, pour toute transition σ, $\mathtt{E}_x \xrightarrow{\sigma} \mathtt{E}_y$, $\mathcal{H}(\mathtt{E}_x) \leq c(\mathtt{E}_x, \mathtt{E}_y) + \mathcal{H}(\mathtt{E}_y)$, ce qui prouve l'initialisation.

Hérédité. Par hypothèse de récurrence, on suppose que la proprieté est vraie pour un $n \geq 1$:

$$\text{Pour tout chemin de longueur } n, \text{ de } \mathtt{E} \text{ à } \mathtt{F}, \mathcal{H}(\mathtt{E}) \leq c(\mathtt{E}, \mathtt{F}) + \mathcal{H}(\mathtt{F}) \tag{2}$$

Soit $\mathtt{E}_x = \mathtt{E}^0 \xrightarrow{\sigma_1} \ldots \xrightarrow{\sigma_n} \mathtt{E}^n \xrightarrow{\sigma_{n+1}} \mathtt{E}^{n+1} = \mathtt{E}_y$ un chemin de longueur $n+1$. $\mathtt{E}_x \xrightarrow{\sigma_1} \ldots \xrightarrow{\sigma_n} \mathtt{E}^n$ est un chemin de longueur n, donc, d'après (2) :

$$\mathcal{H}(\mathtt{E}_x) \leq c(\mathtt{E}_x, \mathtt{E}^n) + \mathcal{H}(\mathtt{E}^n) \tag{3}$$

$\mathtt{E}^n \xrightarrow{\sigma_{n+1}} \mathtt{E}_y$ est un chemin de longueur 1, donc, d'après l'initialisation :

$$\mathcal{H}(\mathtt{E}^n) \leq c(\mathtt{E}^n, \mathtt{E}_y) + \mathcal{H}(\mathtt{E}_y) \tag{4}$$

Finalement, puisque c est additif :

$$c(\mathtt{E}_x, \mathtt{E}_y) = c(\mathtt{E}_x, \mathtt{E}^n) + c(\mathtt{E}^n, \mathtt{E}_y) \tag{5}$$

À partir de (3), (4) et (5) il peut être déduit que $\mathcal{H}(\mathtt{E}_x) \leq c(\mathtt{E}_x, \mathtt{E}_y) + \mathcal{H}(\mathtt{E}_y)$ pour tout chemin de $\mathtt{E}_x$ à $\mathtt{E}_y$ de longueur $n+1$. Ainsi, la proposition est prouvée.

□

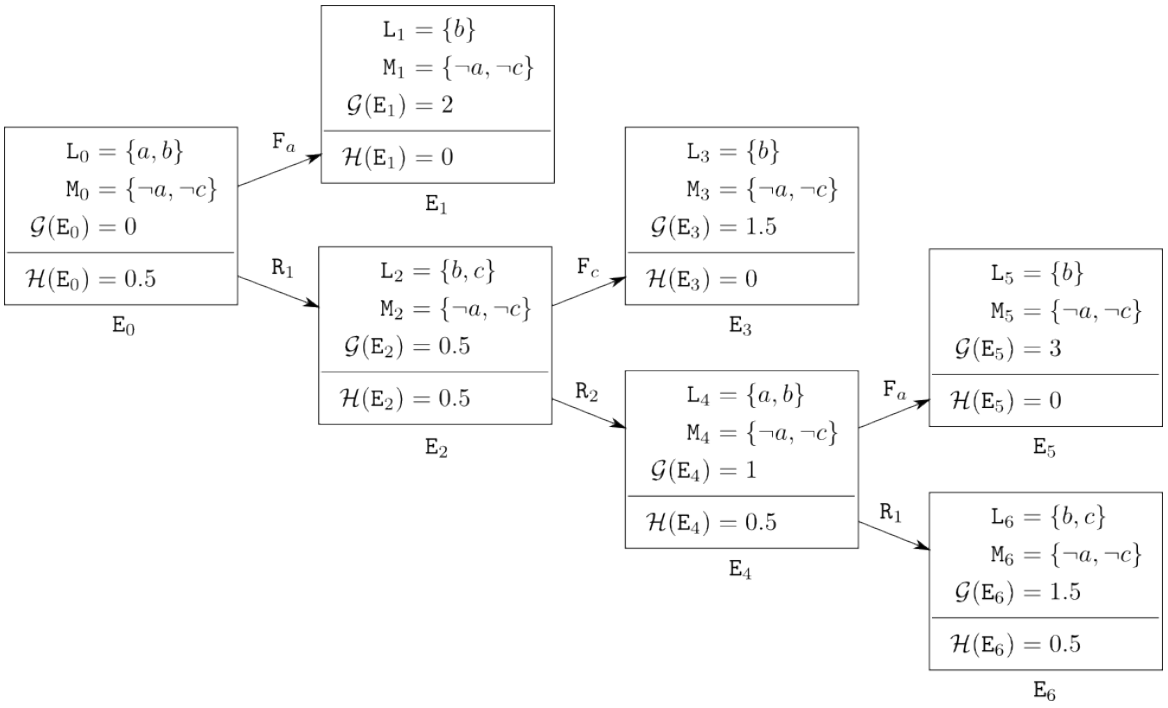

FIGURE 2 – Application de l'algorithme sur l'exemple.

4.3 Exemple

Considérons les entrées suivantes :

$$\begin{array}{lll} \psi = a \wedge b & \mathtt{CA} = \{\mathtt{R}_1, \mathtt{R}_2\} & \mathtt{coût}(\mathtt{R}_1) = \mathtt{coût}(\mathtt{R}_2) = 0.5 \\ \mu = \neg a \wedge \neg c & \mathtt{R}_1 = a \wedge b \rightsquigarrow b \wedge c & \mathtt{coût}(\mathtt{F}_\ell) = 1 \text{ pour } \ell \neq a \\ & \mathtt{R}_2 = b \wedge c \rightsquigarrow a \wedge b & \mathtt{coût}(\mathtt{F}_a) = 2 \end{array}$$

Pour tout état $\mathtt{E} = (\mathtt{L}, \mathtt{M})$, rappelons que $\mathcal{H}(\mathtt{E}) = \mathtt{ERC}(\mathtt{L} \cap \mathtt{M}^\neg)$. ERC prend les valeurs suivantes :

$$\mathtt{ERC}(\{\ell\}) = \begin{cases} 0.5 \text{ pour } \ell \in \{a, c\}, \\ 1 \text{ sinon.} \end{cases}$$

Dans cet exemple, $\mathtt{min\text{-}branches}(\psi) = \{\{a, b\}\}$ et $\mathtt{min\text{-}branches}(\mu) = \{\{\neg a, \neg c\}\}$. La figure 2 montre les différents états explorés par l'algorithme. Pour l'état initial $\mathtt{E}_0$, $\mathcal{F}(\mathtt{E}_0) = 0.5$ puisque $\mathcal{F}(\mathtt{E}) = \mathcal{G}(\mathtt{E}) + \mathcal{H}(\mathtt{E})$, $\mathcal{G}(\mathtt{E}_0) = 0$ car $\mathtt{E}_0$ est un état initial, et $\mathcal{H}(\mathtt{E}_0) = \mathtt{ERC}(\mathtt{L}_0 \cap \mathtt{M}_0^\neg) = \mathtt{ERC}(\{a\}) = 0.5$. Étant donné l'ensemble des *flips* et des règles d'adaptation, le coût pour réparer un conflit sur a est d'au minimum 0.5. Deux états sont développés, le premier, $\mathtt{E}_1$, généré par un *flip* sur a et le second, $\mathtt{E}_2$, par la règle $\mathtt{R}_1$. L'état $\mathtt{E}_1$ est un état final, avec $\mathcal{F}(\mathtt{E}_1) = 2$. L'état $\mathtt{E}_2$ est un état non-final, avec $\mathcal{G}(\mathtt{E}_2) = 0.5$ et $\mathcal{H}(\mathtt{E}_2) = 0.5$. Bien que $\mathtt{E}_1$ soit un état final, ce n'est pas une condition suffisante pour l'accepter comme solution optimale car il existe d'autres états à explorer avec une valeur de $\mathcal{F}$ plus faible : il est donc encore possible de trouver une solution moins coûteuse. L'algorithme développe donc l'état $\mathtt{E}_2$ qui minimise $\mathcal{F}$. Deux transitions sont applicables sur $\mathtt{E}_2$: l'algorithme développe deux états, le premier, $\mathtt{E}_3$, en appliquant un *flip* sur c, le second, $\mathtt{E}_4$ en appliquant la règle $\mathtt{R}_2$. L'état $\mathtt{E}_3$ est un état final, avec $\mathcal{F}(\mathtt{E}_3) = 1.5$. L'état $\mathtt{E}_4$ est un état non-final, avec $\mathcal{G}(\mathtt{E}_4) = 1$ et $\mathcal{H}(\mathtt{E}_4) = 0.5$. L'état $\mathtt{E}_3$ est un état final minimisant $\mathcal{F}$, la valeur $\mathcal{F}(\mathtt{E}_3) = 1.5$ est donc attribuée à $\mathtt{coûtSolution}$. Cela signifie que l'état $\mathtt{E}_1$ ne sera jamais exploré, puisque $F(\mathtt{E}_1) > \mathtt{coûtSolution}$. L'état $\mathtt{E}_4$ est le seul état avec une valeur de $\mathcal{F}$ égale à $\mathtt{coûtSolution}$. Deux transitions sont applicables sur $\mathtt{E}_4$: l'algorithme génère deux états, le premier, $\mathtt{E}_5$, en appliquant un *flip* sur a, le second, $\mathtt{E}_6$ en appliquant la règle $\mathtt{R}_1$. $\mathcal{F}(\mathtt{E}_5) = 3$ et $\mathcal{F}(\mathtt{E}_6) = 2$, donc aucun de ces états ne mène à une solution optimale. Ainsi la seule solution optimale est le chemin d'adaptation menant à $\mathtt{E}_3$, l'algorithme renvoie donc $\mathtt{L}_3 \cup \mathtt{M}_3$, soit $\{\neg a, b, \neg c\}$ correspondant à la formule $\neg a \wedge b \wedge \neg c$.

4.4 Terminaison et complexité de l'algorithme

Dans cette section, la terminaison et la complexité de l'algorithme sont étudiées dans le cas où aucune heuristique n'est utilisée ($\mathcal{H} = 0$). Puisque l'heuristique introduite dans la section 4.2 est admissible, la complexité sans heuristique est une borne supérieure de la complexité avec heuristique.

Proposition 2. *L'exécution de l'algorithme dans ce papier se fait en un temps fini. Sa complexité dans le pire cas est d'ordre $O(4^n \times t^{\varrho+1} \times (n + \varrho \log(t)))$, où n est le nombre de variables, $t = |\mathtt{CA}| + |\mathcal{V}|$, et ϱ est la somme du coût de chaque* flip *divisée par le coût minimal d'une transition.*
D'après [8], la complexité d'un opérateur de révision est $\mathsf{P}^{\mathsf{NP}[O(\log n)]}$-difficile [2]. Ainsi, cette haute complexité dans le pire cas n'est pas une surprise.

Preuve. Soit $\mathtt{G}$ le graphe d'exploration généré par l'algorithme : c'est un graphe dirigé acyclique (correspondant à l'union des arbres d'exploration depuis chaque état initial), dont les nœuds représentent des états, et les arcs représentent des transitions : si (v, w) est un arc de $\mathtt{G}$, alors $\mathtt{E}_x \xrightarrow{\sigma} \mathtt{E}_y$ est une transition où $\mathtt{E}_x = \lambda(v)$, $\mathtt{E}_y = \lambda(w)$ et $\sigma = \lambda((v, w))$ (λ est la fonction de représentation associée à $\mathtt{G}$). Les racines de $\mathtt{G}$ (nœuds de rang 0) correspondent aux s_0 états initiaux. Soit s_k le nombre de nœuds de rang k. s_k vérifie $s_k \leq s_0 t^k$, où $t = n + |\mathtt{CA}|$ est une borne supérieure du nombre de transitions depuis un état $\mathtt{E}$ (pour rappel, il existe n variables propositionnelles, et si un *flip* $\mathtt{F}_\ell$ est applicable sur $\mathtt{E}$, alors $\mathtt{F}_{\neg\ell}$ ne l'est pas). Afin de prouver la terminaison de l'algorithme, il est suffisant de prouver que $\mathtt{G}$ est de taille finie : chaque étape de la boucle est exécutée en un temps fini, et chaque nœud de $\mathtt{G}$ n'est considéré qu'une seule fois.

Il existe une solution, c'est-à-dire un chemin p d'un nœud représentant un état initial à un autre représentant un état final. En effet, soit $(\mathtt{L}_0, \mathtt{M})$ un état initial. Alors, l'application de tous les flips $\mathtt{F}_\ell$ pour $\ell \in (\mathtt{L}_0 \cap \mathtt{M}^\neg) \subseteq \mathtt{L}_0$ mène à un état $(\mathtt{L}_0 \backslash (\mathtt{L}_0 \cap \mathtt{M}^\neg), \mathtt{M}) = (\mathtt{L}_0 \cap \overline{\mathtt{M}^\neg}, \mathtt{M})$ qui est cohérent puisque $\mathtt{L}_0 \cap \overline{\mathtt{M}^\neg} \cap \mathtt{M}^\neg = \varnothing$. Soit $\mathcal{C}_\mathtt{F} = \mathtt{coût}(p)$.

Cela implique qu'il existe (au moins) une solution optimale p^* : l'ensemble des $\{p \mid p \text{ est un chemin avec } \mathtt{coût}(p) \leq \mathcal{C}_\mathtt{F}\}$ est fini, donc son infimum est atteint. Soit $\mathcal{C}^* = \mathtt{coût}(p^*)$ et v^* le nœud final de p^*. Il peut être prouvé que le rang de v^*, r^*, vérifie :

$$r^* \leq \left\lfloor \frac{\mathcal{C}^*}{\mathtt{mc}} \right\rfloor \leq \left\lfloor \frac{\mathcal{C}_\mathtt{F}}{\mathtt{mc}} \right\rfloor \leq \left\lfloor \frac{\mathtt{fc}}{\mathtt{mc}} \right\rfloor = \varrho \tag{6}$$

$$\text{où } \mathtt{mc} = \min\{\mathtt{coût}(\sigma) \mid \sigma \in \mathtt{CA} \cup \{\mathtt{F}_\ell \text{ pour chaque littéral } \ell\}\},$$

$$\mathtt{fc} = \sum_{\ell \in \mathcal{V} \cup \mathcal{V}^\neg} \mathtt{coût}(\mathtt{F}_\ell), \text{et } \lfloor x \rfloor \text{ est la partie entière de } x$$

En effet, $\mathcal{C}^* = \sum_{i=1}^{r^*} \mathtt{coût}(\sigma_i)$, où σ_i est la $i^{\text{ème}}$ transition de p^*. Alors, $\mathcal{C}^* \geq r^* \times \mathtt{mc}$ et donc, $r^* \leq \dfrac{\mathcal{C}^*}{\mathtt{mc}}$. Puisque r^* est un nombre entier, l'inéquation (6) est vérifiée.

Par conséquent, chaque chemin d'adaptation optimal a une longueur inférieure ou égale à $\left\lfloor \dfrac{\mathcal{C}^*}{\mathtt{mc}} \right\rfloor$. Puisque le graphe $\mathtt{G}$ est parcouru dans l'ordre croissant des $\mathcal{G}(\mathtt{E})$ (en admettant $\mathcal{H} = 0$), chaque nœud v de $\mathtt{G}$ a un rang inférieur à $\left\lfloor \dfrac{\mathcal{C}^*}{\mathtt{mc}} \right\rfloor$. Cela signifie que $\mathtt{G}$ est de taille finie, et donc prouve la terminaison de l'algorithme. On peut également en déduire que le nombre de nœuds $N_\mathtt{G}$ vérifie :

$$N_\mathtt{G} \leq \sum_{k=0}^{r^*} s_k \leq \sum_{k=0}^{r^*} s_0 t^k = s_0 \frac{t^{r^*+1} - 1}{t - 1} \leq s_0 t^{r^*+1} \leq s_0 t^{\varrho+1}$$

qui donne une borne supérieure pour le nombre d'itérations de la boucle de l'algorithme.

Le nombre de nœuds de rang 0, s_0, est borné par $|\mathtt{min\text{-}branches}(\psi)| \times |\mathtt{min\text{-}branches}(\mu)|$, qui est dans le pire des cas de $2^n \times 2^n = 4^n$. On suppose que chaque nœud v est stocké dans une structure de

2. Plus précisément, la complexité de l'opérateur de Dalal, $\dot{+}_{\text{Dalal}}$, est $\mathsf{P}^{\mathsf{NP}[O(\log n)]}$-complet et l'algorithme présenté dans ce papier peut être utilisé pour calculer $\dot{+}_{\text{Dalal}}$: $\dot{+}_{\text{Dalal}} = \dot{+}_{\mathtt{CA}}$ avec $\mathtt{CA} = \varnothing$ et $\mathtt{coût}(\mathtt{F}_\ell) = 1$ pour chaque littéral ℓ.

données triée sur la valeur de $\mathcal{F}(\lambda(v))$. Ainsi, trouver un état $\mathtt{E}$ minimisant $\mathcal{F}(\mathtt{E})$ est fait en temps constant, tandis que créer un état est fait en temps logarithmique par rapport à la taille de la structure, bornée par N_{G}. Ainsi, la complexité de l'algorithme est de $O(N_{\mathrm{G}} \log(N_{\mathrm{G}}))$, et N_{G} est d'ordre $O(4^n \times t^{\varrho+1})$. Plus précisément, la complexité de l'algorithme est de $O(4^n \times t^{\varrho+1} \times (n + \varrho \log(t)))$.

□

5 Implantation de l'algorithme

5.1 REVISOR/PLAK

REVISOR/PLAK est une implantation de l'algorithme d'adaptation par révision et par règles en logique propositionnelle. L'implantation actuelle n'implante pas la fonction `min-branches` qui calcule l'ensemble des impliquants premiers d'une formule, cette opération doit donc être effectuée au préalable. REVISOR/PLAK a été implanté en Java et est disponible au téléchargement sur le site `http://revisor.loria.fr`. Un environnement Java en version 7 est requis pour l'exécution de REVISOR/PLAK.

Jusqu'ici, les données expérimentales montrent que le temps de calcul est très dépendant de ϱ. Cependant, l'augmentation de ϱ n'entraîne pas une augmentation exponentielle du temps de calcul dans la plupart des cas. Cependant, le temps de calcul est susceptible d'augmenter considérablement lorsque des sous-ensembles de $\mathtt{CA}$, $\{(\mathtt{gauche}_i, \mathtt{droite}_i) \mid i \in \{0, 1, \ldots, n\}\} \subseteq \mathtt{CA}$ existent tels que $\mathtt{gauche}_i \subseteq \mathtt{droite}_{i-1}$ pour $i \in \{1, \ldots, n\}$ et $\mathtt{gauche}_0 \subseteq \mathtt{droite}_n$. Ces ensembles de règles sont en effet capables de créer des chemins d'adaptation de très grande longueur avant que l'algorithme ne trouve une solution, d'autant plus si ces règles ont un coût très faible (relativement aux coûts des autres règles d'adaptation et *flips*).

L'implantation actuelle résoud 87.5% (écart-type de 4.6 sur des séries de 50 problèmes) des problèmes d'adaptation testés avec $n = 50$ et 40 règles d'adaptation en moins d'une seconde, sur une machine équipée d'un processeur à 2.13Ghz et 3Go de mémoire vive disponible.

5.2 Un exemple concret

Bob cherche une recette de tarte, sans cannelle, ni œufs, ni poires. Aucune recette dans la base de cas ne correspond à la requête de Bob. La seule recette de tarte est une recette de tarte aux poires contenant des œufs :

$$\mathtt{Source} = \text{tarte} \wedge \text{pâte\_à\_tarte} \wedge \text{poire} \wedge \text{sucre} \wedge \text{œufs}$$
$$\mathtt{Cible} = \text{tarte} \wedge \neg\text{poire} \wedge \neg\text{cannelle} \wedge \neg\text{œufs}$$

REVISOR/PLAK adapte $\mathtt{Source}$ pour résoudre $\mathtt{Cible}$, en utilisant les connaissances du domaine suivantes :

$$\mathtt{CD} = (\text{pomme} \vee \text{pêche} \vee \text{poire}) \Leftrightarrow \text{fruit}$$

Les pommes, les pêches et les poires sont des fruits, et réciproquement, tous les fruits sont des pommes, des pêches ou des poires.

$$\mathtt{CA} = \{\mathtt{R}_1, \mathtt{R}_2, \mathtt{R}_3, \mathtt{R}_4, \mathtt{R}_5, \mathtt{R}_6\}$$

$\mathtt{R}_1 = \text{gâteau} \wedge \text{œufs} \rightsquigarrow \text{gâteau} \wedge \text{banane}$
$\mathtt{R}_2 = \text{tarte} \wedge \text{œufs} \rightsquigarrow \text{tarte} \wedge \text{farine} \wedge \text{vinaigre\_de\_cidre}$
$\mathtt{R}_3 = \text{poire} \rightsquigarrow \text{pêche}$
$\mathtt{R}_4 = \text{poire} \rightsquigarrow \text{pomme} \wedge \text{cannelle}$
$\mathtt{R}_5 = \text{cannelle} \rightsquigarrow \text{fleur\_d'oranger}$
$\mathtt{R}_6 = \text{cannelle} \rightsquigarrow \text{sucre\_vanillé}$

Chaque règle d'adaptation a un coût de 0.3 excepté $\mathtt{R}_3$ qui a un coût de 0.7. En effet, les poires sont plus similaires aux pommes qu'aux pêches. Chaque *flip* a un coût de 1.

REVISOR/PLAK renvoie le résultat suivant (en moins de 20ms) :

$$\text{tarte} \wedge \text{pâte\_à\_tarte} \wedge \text{sucre} \wedge \text{fruit} \wedge \neg\text{œufs} \wedge \text{farine} \wedge \text{vinaigre\_de\_cidre} \wedge$$
$$\neg\text{poire} \wedge \text{pomme} \wedge \neg\text{cannelle} \wedge (\text{sucre\_vanillé} \vee \text{fleur\_d'oranger})$$

Deux recettes sont proposées. Dans les deux, les poires ont été remplacées par des pommes, et les œufs par de la farine et du vinaigre de cidre. Pour remplacer la cannelle, deux ingrédients sont proposés : le sucre vanillé et la fleur d'oranger. En effet $\texttt{coût}(\texttt{R}_5) = \texttt{coût}(\texttt{R}_6)$, la disjonction de ces possibilités est donc présentée. Si les coûts de $\texttt{R}_5$ et $\texttt{R}_6$ étaient différents, alors l'algorithme choisirait uniquement la fleur d'oranger si $\texttt{coût}(\texttt{R}_5) < \texttt{coût}(\texttt{R}_6)$, ou le sucre vanillé si $\texttt{coût}(\texttt{R}_5) > \texttt{coût}(\texttt{R}_6)$. Ces deux recettes sont obtenues par l'application des règles $\texttt{R}_2$ et $\texttt{R}_4$, puis de la règle $\texttt{R}_5$ ou $\texttt{R}_6$.

6 Travaux proches

Dans [5], un algorithme d'adaptation par révision a été défini pour la logique $\mathcal{ALC}$. Cet algorithme utilise également des réparations dans des branches de tableaux. En parallèle des travaux de Camilla Schwind [19] sur la révision en logique propositionnelle fondée sur des réparations dans des tableaux, nous avons défini et implanté un algorithme de révision similaire dans l'outil REVISOR/PL [3] [3]. Camilla Schwind décrit plusieurs implantations d'opérateurs fondées sur cette méthode, dont l'opérateur de révision de Dalal utilisant la distance de Hamming d_H, et les opérateurs utilisant des d_H pondérées. Ces opérateurs sont implantés efficacement dans REVISOR/PL qui utilise les d_H pondérées et dans REVISOR/PLAK qui utilise $d_{\texttt{CA}}$ avec $\texttt{CA} = \varnothing$. Dans ce dernier cas, le coût des *flips* permet d'obtenir l'expressivité des d_H pondérées. L'apport de l'algorithme présenté dans ce papier repose dans l'ajout des connaissances d'adaptation dans un opérateur de révision.

7 Conclusion et perspectives

Ce papier propose un algorithme original pour l'adaptation par révision et par règles d'adaptation en logique propositionnelle. Cette algorithme modifie minimalement le cas source afin de le rendre cohérent avec le cas cible. La méthode des tableaux est appliquée sur le cas source et le cas cible, ensuite, les branches cohérentes de la source sont combinées avec les branches cohérentes de la cible. Ceci conduit à un nouvel ensemble de branches, toutes comportant des conflits (à l'exception du cas où le cas source est déjà cohérent avec le cas cible). Ces conflits sont réparés à l'aide de règles d'adaptation qui modifient le cas source. Les règles d'adaptation permettent de substituer une partie donnée de la source par d'autre éléments. Quand les règles ne peuvent pas fournir une solution optimale, des *flips* sont utilisés pour retirer des littéraux du cas source.

Cet algorithme a été implanté dans l'outil REVISOR/PLAK. Pour chaque littéral ℓ, une heuristique calcule le coût minimal pour réparer un conflit sur ℓ. Ainsi, à chaque itération de l'algorithme, l'état exploré est celui pour qui le coût actuel plus le coût heuristique des prochaines réparations est le plus bas. Une fois qu'une solution optimale est trouvée, seuls les états ayant un coût total estimé égal au coût de la solution sont considérés.

Actuellement, l'algorithme génère des chemins d'adaptations identiques à l'ordre des règles utilisées près, mais qui conduisent parfois au même état final, et donc à une solution identique. Il existe donc certains ensembles de règles dites commutatives entre elles, c'est-à-dire que l'ordre dans lequel elles sont appliquées n'affecte pas la solution. Une perspective consisterait à trouver un moyen efficace d'identifier ces règles commutatives entre elles, afin de ne pas générer de chemins conduisant aux même résultats, ce qui pourrait diminuer le temps de calcul de manière importante.

La figure 2 et un relecteur anonyme attentif (que nous remercions au passage) donnent à penser qu'il pourrait être utile dans l'algorithme de ne pas génerer plusieurs fois le même état (cf. états $\texttt{E}_0$ et $\texttt{E}_4$ de la figure). Bien que l'utilisation de A* permet de limiter la génération d'états déjà explorés (puisqu'ils sont souvent plus coûteux), certains sont tout de même générés. Modifier l'algorithme de manière à ne plus générer ces états inutiles est envisageable : il serait alors nécessaire d'étudier l'impact sur le temps de calcul et la mémoire utilisée.

La révision de croyances est une opération de changement de croyances. Il en existe d'autres (voir [16], pour une synthèse) comme la contraction ($\psi \dot{-} \mu$ est une base de croyances obtenue par la modification minimale de ψ tel qu'il n'implique pas μ) ou la fusion de croyances avec contrainte d'intégrité [10] ($\Delta_\mu(\{\psi_1, \ldots, \psi_n\})$ est une base de croyances obtenue en modifiant les bases de croyances ψ_i en ψ'_i telles que $\mu \wedge \bigwedge_i \psi'_i$ est cohérent). Une perspective consisterait à étudier comment l'approche des réparations dans les tableaux présentée dans ce

3. Disponible au téléchargement sur `http://revisor.loria.fr`.

papier pourrait être adaptée pour ces opérations. Cela aurait un impact sur le RàPC ; en particulier la fusion de croyances avec contrainte d'intégrité peut être utilisée pour l'adaptation de cas multiples (c'est-à-dire, la combinaison de plusieurs cas source pour résoudre le cas cible), voir [4] pour les détails.

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] J. Cojan, V. Dufour Lussier, A. Hermann, F. Le Ber, J. Lieber, E. Nauer, et G. Personeni. REVISOR : un ensemble de moteurs d'adaptation de cas par révision des croyances. In *Journée Intelligence Artificielle Fondamentale*, 2013.

[4] J. Cojan et J. Lieber. Belief Merging-based Case Combination. In *Case-Based Reasoning Research and Development (ICCBR 2009)*, pages 105–119, 2009.

[5] J. Cojan et J. Lieber. An Algorithm for Adapting Cases Represented in an Expressive Description Logic. In *International Conference on Case-Based Reasoning (ICCBR)*, LNAI 6176. Springer, 2010.

[6] J. Cojan et J. Lieber. Belief revision-based case-based reasoning. In G. Richard, editor, *Proceedings of the ECAI-2012 Workshop SAMAI : Similarity and Analogy-based Methods in AI*, pages 33–39, 2012.

[7] A. Coman et H. Muñoz-Avila. Diverse plan generation by plan adaptation and by first-principles planning : A comparative study. In *Case-Based Reasoning Research and Development (ICCBR)*, LNCS 7466. Springer, 2012.

[8] T. Eiter et G. Gottlob. On the Complexity of Propositional Knowledge Base Revision, Updates, and Counterfactuals. *Artificial Intelligence*, 57 :227–270, 1992.

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

[10] S. Konieczny et R. Pino Pérez. Merging information under constraints : a logical framework. *Journal of Logic and Computation*, 12(5) :773–808, 2002.

[11] J. Lieber. Application of the Revision Theory to Adaptation in Case-Based Reasoning : the Conservative Adaptation. In *International Conference on Case-Based Reasoning (ICCBR)*, LNCS 4626. Springer, 2007.

[12] 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*, pages 122–136, 2011.

[13] E. Melis, J. Lieber, et A. Napoli. Reformulation in Case-Based Reasoning. In *European Workshop on Case-Based Reasoning, EWCBR-98*, LNCS 1488, pages 172–183. Springer, 1998.

[14] M. Minor, R. Bergmann, S. Görg, et K. Walter. Towards Case-Based Adaptation of Workflows. In *International Conference on Case-Based Reasoning*, LNCS 4626, pages 421–435. Springer, 2007.

[15] J. Pearl. *Heuristics – Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley Publishing Co., Reading, MA, 1984.

[16] P. Peppas. Belief Revision. In F. van Harmelen, V. Lifschitz, et B. Porter, editors, *Handbook of Knowledge Representation*, chapter 8, pages 317–359. Elsevier, 2008.

[17] C. K. Riesbeck et R. C. Schank. *Inside Case-Based Reasoning*. Lawrence Erlbaum Associates, Inc., Hillsdale, New Jersey, 1989.

[18] J. Rubin et I. Watson. Opponent type adaptation for case-based strategies in adversarial games. In *Case-Based Reasoning Research and Development (ICCBR)*, LNCS 7466, pages 357–368. Springer, 2012.

[19] C. Schwind. From Inconsistency to Consistency : Knowledge Base Revision by Tableaux Opening. In *Advances in Artificial Intelligence (IBERAMIA)*, LNCS 6433, pages 120–132. Springer, 2010.

[20] R. Socher. Optimizing the clausal normal form transformation. *J. Autom. Reason.*, 7(3) :325–336, September 1991.