Adapting Propositional Cases Based on Tableaux Repairs Using Adaptation Knowledge*

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

[1] Université de Lorraine, LORIA, UMR 7503 — 54506 Vandœuvre-lès-Nancy, France
{Gabin.Personni,Alice.Hermann,Jean.Lieber}@loria.fr
[2] CNRS — 54506 Vandœuvre-lès-Nancy, France
[3] Inria — 54602 Villers-lès-Nancy, France

Abstract. Adaptation is a step of case-based reasoning that aims at modifying a source case (representing a problem-solving episode) in order to solve a new problem, called the target case. An approach to adaptation consists in applying a belief revision operator that modifies minimally the source case so that it becomes consistent with the target case. Another approach consists in using domain-dependent adaptation rules. These two approaches can be combined: a revision operator parametrized by the adaptation rules is introduced and the corresponding revision-based adaptation uses the rules to modify the source case. This paper presents an algorithm for revision-based and rule-based adaptation based on tableaux repairs in propositional logic: when the conjunction of source and target cases is inconsistent, the tableaux method leads to a set of branches, each of them ending with clashes, and then, these clashes are repaired (thus modifying the source case), with the help of the adaptation rules. This algorithm has been implemented in the REVISOR/PLAK tool and some implementation issues are presented.

Keywords: Case-based reasoning, adaptation, tableaux repairs, propositional logic, belief revision, adaptation rules.

1 Introduction

Case-based reasoning (CBR [1]) is a reasoning paradigm based on the reuse of chunks of experience called cases. A *case* is a representation of a problem-solving episode, often separated in a problem part and a solution part: this separation is not formally necessary but is useful to the intuitive understanding of the notion of case. The input of a CBR system is a target case, which represents an underspecified case (intuitively, its problem part is well specified, whereas its solution part is not). A classical way to implement a CBR system consists in (1) selecting a case from a case base that is similar to the target case, (2) adapting this retrieved case in order to solve the target case, i.e., in order to add information to it (intuitively, this consists in specifying the solution part of the target case by reusing the solution part of the retrieved case). Variants of this approach to CBR and other steps related to it can be found in, e.g., [2].

* This research was partially funded by the project Kolflow of the French National Agency for Research (ANR), program ANR CONTINT (http://kolflow.univ-nantes.fr).

L. Lamontagne and E. Plaza (Eds.): ICCBR 2014, LNCS 8765, pp. 390–404, 2014.

This paper concentrates on step (2), *adaptation*. Despite its importance, this step of CBR has been a little bit neglected in the CBR literature, though it has recently received some attention (see, e.g., [3–6]). In particular, the paper concentrates on an approach to adaptation that we introduced some years ago [7] and studied according to various aspects (see [8] for a synthesis). This approach called (now) *revision-based adaptation* or $\dot{+}$*-adaptation* is based on a belief revision operator $\dot{+}$. According to the AGM postulates (called after the names of [9]), the *belief revision* of a belief base ψ by a belief base μ—$\psi \dot{+} \mu$—consists in minimally modifying ψ into ψ' such that the conjunction of ψ' and μ is consistent; then $\psi \dot{+} \mu$ is this conjunction. It must be noticed that, since there are many ways to "measure" modifications, there are also many ways to "minimally modify" a belief base, hence there are multiple revision operators satisfying the AGM postulates.

Roughly said, $\dot{+}$-adaptation consists in using $\dot{+}$ in order to modify the retrieved case so that it is consistent with the target case, and the adaptation process returns the result of this revision. So, implementing a revision-based adaptation amounts to implementing a revision operator. This implementation depends on the formalism used in the CBR system. In particular, we have studied how $\dot{+}$-adaptation can be implemented within the description logic $\mathcal{ALC}$ [10].[1] The approach to adaptation in $\mathcal{ALC}$ consisted in applying *tableaux repairs*. The same idea of tableaux repairs can be used for revision of ψ by μ according to the following principle: the tableaux method is applied separately on ψ and μ, then the consistent branches are combined, which leads to a set of inconsistent branches (unless the conjunction of ψ and μ is consistent). Then, tableaux repair consists in removing the parts of the clashes whose origin is ψ and the formulas of the branch from which these parts of clashes are deduced. This involves a weakening of ψ into ψ' such that ψ' is consistent with μ, and the result of the revision is the conjunction of ψ' and μ. This approach for belief revision algorithm has been found in parallel by Camilla Schwind [11], who has applied it to propositional logic with a finite number of variables and thus, this work can be used for $\dot{+}$-adaptation of propositional cases.

This paper proposes to go one step beyond Camilla Schwind's work, by integrating, in the tableaux repairs, some domain-specific adaptation knowledge in the form of *adaptation rules* (also called reformulations in [12]). Such rules represent the fact that, in a given context, a given part of a case can be substituted by something. Thus, the idea is to use such rules for tableaux repairs.

The paper is organized as follows. In Section 2, some preliminaries introduce the notions and the notations used throughout the paper. In Section 3, the adaptation process in CBR is presented, pointing out the notions of adaptation rules and of revision-based adaptation. Then, the algorithm for adaptation by tableaux repairs using adaptation rules is presented in Section 4. The approach has been implemented in an inference engine called REVISOR/PLAK: this system and some implementation issues are described, in Section 5, as well as a concrete example. Section 6 concludes the paper.

The research report [13] is a long version of this paper including the proofs.

[1] Technically, we have not defined a revision operator in $\mathcal{ALC}$, since the implemented operator violates some of the postulates of [9], but we have implemented an adaptation operator inspired from the ideas of revision-based adaptation.

2 Preliminaries

2.1 Propositional Logic

Let $\mathcal{V} = \{a_1, \ldots, a_n\}$ be a set of n distinct symbols called propositional variables. A propositional formula built on $\mathcal{V}$ is either a variable a_i or of one of the forms $\varphi_1 \wedge \varphi_2$, $\varphi_1 \vee \varphi_2$, $\neg\varphi_1$, $\varphi_1 \Rightarrow \varphi_2$, and $\varphi_1 \Leftrightarrow \varphi_2$, where φ_1 and φ_2 are two propositional formulas. Let $\mathcal{L}$ be the set of the propositional formulas.

Let $\mathbb{B} = \{\mathtt{T}, \mathtt{F}\}$ be a set of two elements. Given $x = (x_1, \ldots, x_n) \in \mathbb{B}^n$ and $\varphi \in \mathcal{L}$, $\varphi^x \in \mathbb{B}$ is defined as follows: $a_i^x = x_i$, $(\varphi_1 \wedge \varphi_2)^x = \mathtt{T}$ iff $\varphi_1^x = \mathtt{T}$ and $\varphi_2^x = \mathtt{T}$, $(\varphi_1 \vee \varphi_2)^x = \mathtt{T}$ iff $\varphi_1^x = \mathtt{T}$ or $\varphi_2^x = \mathtt{T}$, $(\neg\varphi_1)^x = \mathtt{T}$ iff $\varphi_1^x = \mathtt{F}$, $(\varphi_1 \Rightarrow \varphi_2)^x = (\neg\varphi_1 \vee \varphi_2)^x$, and $(\varphi_1 \Leftrightarrow \varphi_2)^x = ((\varphi_1 \Rightarrow \varphi_2) \wedge (\varphi_2 \Rightarrow \varphi_1))^x$.

For $\varphi \in \mathcal{L}$, let $\mathcal{M}(\varphi) = \{x \in \mathbb{B}^n \mid \varphi^x = \mathtt{T}\}$ called the set of models of φ. Given $\varphi_1, \varphi_2 \in \mathcal{L}$, $\varphi_1 \models \varphi_2$ if $\mathcal{M}(\varphi_1) \subseteq \mathcal{M}(\varphi_2)$, and $\varphi_1 \equiv \varphi_2$ if $\mathcal{M}(\varphi_1) = \mathcal{M}(\varphi_2)$. $(\mathcal{L}, \models)$ is called the propositional logic with n variables.

A literal ℓ is a propositional formula of the form a_i (positive literal) or $\neg a_i$ (negative literal), where $a_i \in \mathcal{V}$. If $\ell = \neg a_i$ is a negative literal, then $\neg\ell$ denotes the positive literal a_i (instead of the equivalent formula $\neg\neg a_i$). A formula is in disjunctive normal form or DNF if it is a disjunction of conjunctions of literals. A formula is in negative normal form (NNF) if it contains only the connectives $\wedge$, $\vee$ and $\neg$, and if $\neg$ appears only in front of propositional variables. Every formula can be put in NNF by applying, as rewriting rules oriented from left to right, the following equivalences, until none of these equivalences is applicable (for $\varphi, \varphi_1, \varphi_2 \in \mathcal{L}$):

$$\begin{aligned}
\varphi_1 \Leftrightarrow \varphi_2 &\equiv (\varphi_1 \Rightarrow \varphi_2) \wedge (\varphi_2 \Rightarrow \varphi_1) \\
\varphi_1 \Rightarrow \varphi_2 &\equiv \neg\varphi_1 \vee \varphi_2 \\
\neg\neg\varphi &\equiv \varphi \\
\neg(\varphi_1 \vee \varphi_2) &\equiv \neg\varphi_1 \wedge \neg\varphi_2 \\
\neg(\varphi_1 \wedge \varphi_2) &\equiv \neg\varphi_1 \vee \neg\varphi_2
\end{aligned}$$

A formula in DNF is necessarily in NNF (the converse is false).

A set of literals $\mathtt{L}$ is often assimilated to the conjunction of its elements, for example $\{a, \neg b, \neg c\}$ is assimilated to $a \wedge \neg b \wedge \neg c$ and vice-versa. In particular, $\mathtt{L}$ is satisfiable iff there is no literal $\ell \in \mathtt{L}$ such that $\neg\ell \in \mathtt{L}$.

An implicant of φ, I is a conjunction or a disjunction of literals, such that $I \models \varphi$ and I is satisfiable. However, as there exists a duality between conjunctive and disjunctive implicants, only conjunctive implicants will be considered in this paper. An implicant I of φ is prime if for any conjunction of literals C such that $C \subset I$, $C \not\models \varphi$. That is, a prime implicant I is minimal. Let $\mathtt{PI}(\varphi)$ be the set of prime implicants of φ. Then:

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

An algorithm to find efficiently prime implicants of a formula is detailed in [14].

2.2 Distances

Let $\mathcal{U}$ be a set. In this paper, a distance on $\mathcal{U}$ is a function $d : \mathcal{U} \times \mathcal{U} \to [0; +\infty]$ such that $d(x, y) = 0$ iff $x = y$ (the other properties of a distance function are not required in this paper). Given $A, B \in 2^{\mathcal{U}}$ and $y \in \mathcal{U}$, the following shortcuts are used:

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

where $\inf X$ denotes the infimum of the set X, with the convention $\inf \varnothing = +\infty$ (e.g., $d(A, \varnothing) = +\infty$).

The Hamming distance on propositional logic interpretations, d_H, is defined by $d_H(x, y) = |\{i \mid i \in \{1, 2, \ldots, n\}, x_i \neq y_i\}|$, for $x, y \in \mathbb{B}^n$. In other words, $d_H(x, y)$ is the number of variable flips to go from x to y.

2.3 Belief Revision

Let ψ be the beliefs of an agent about the world, expressed in a logic $(\mathcal{L}, \models)$. This agent is confronted to new beliefs expressed by $\mu \in \mathcal{L}$. These new beliefs are assumed to be non revisable, whereas the old beliefs ψ can be changed. If μ does not contradict ψ (i.e. if $\psi \wedge \mu$ is consistent), then the new beliefs are simply added to the old beliefs. Otherwise, according to the *minimal change principle* [9], ψ has to be modified minimally into $\psi' \in \mathcal{L}$ such that $\psi' \wedge \mu$ is consistent, and then the revision of ψ by μ, denoted by $\psi \dot{+} \mu$, is this conjunction.

There are multiple ways of measuring modifications of beliefs, hence multiple revision operators $\dot{+}$. In [9], a set of postulates has been defined that a revision operator is supposed to verify. In [15], these postulates have been reformulated in propositional logic and a family of revision operators based on distances has been defined as follows. Let d be a distance on $\mathcal{U} = \mathbb{B}^n$. Let ψ and μ be two formulas. The revision of ψ by μ according to $\dot{+}^d$ ($\psi \dot{+}^d \mu$) is a formula whose models are the models of μ that are the closest ones to the models of ψ according to d (the change from an interpretation x to an interpretation y is measured by $d(x, y)$). Formally, $\psi \dot{+}^d \mu$ is such that:

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

Note that this definition specifies $\psi \dot{+}^d \mu$ up to logical equivalence, but this is sufficient since a revision operator has to satisfy the irrelevance of syntax principle (this is one of the [15]'s postulates: if $\psi_1 \equiv \psi_2$ and $\mu_1 \equiv \mu_2$ then $\psi_1 \dot{+} \mu_1 \equiv \psi_2 \dot{+} \mu_2$).

2.4 A* Search

A* is a heuristic-based best-first search algorithm [16]. It is suited for searching state spaces where there are a finite number of transitions from a given state to its successor states and for which the cost of a path is additive (the cost of a path is the sum of the cost of the transitions it contains). A search problem is given by a finite set of initial states and by a goal giving a condition for a state to be final. Given a state $\mathtt{S}$, let $\mathcal{F}^*(\mathtt{S})$

be the minimum of the costs of the paths from an initial state to a final state that contain $\mathtt{S}$. $\mathcal{F}^*(\mathtt{S}) = \mathcal{G}^*(\mathtt{S}) + \mathcal{H}^*(\mathtt{S})$ where $\mathcal{G}^*(\mathtt{S})$ (resp., $\mathcal{H}^*(\mathtt{S})$) is the minimal of the costs of the paths from an initial state to $\mathtt{S}$ (resp., from $\mathtt{S}$ to a final state). In general, $\mathcal{F}^*$ is unknown and is approximated by a function $\mathcal{F} = \mathcal{G} + \mathcal{H}$. $\mathcal{F}$ is said to be *admissible* if $\mathcal{G} \geq \mathcal{G}^*$ and $\mathcal{H} \leq \mathcal{H}^*$. If $\mathcal{F}$ is admissible, then the A* procedure is optimal: if there is a solution (a path from an initial to a final state), then this solution has a minimal cost. The A* procedure consists in searching the state space, starting from the initial states, by increasing $\mathcal{F}(\mathtt{S})$: among two successors of the current state, the one that is minimum for $\mathcal{F}$ is preferred. Usually, $\mathcal{G}(\mathtt{S})$ is the cost of the path that has already been generated for reaching $\mathtt{S}$ and thus, $\mathcal{G} \geq \mathcal{G}^*$. Then, the main difficulty is to find an admissible $\mathcal{H}$ (the constant function 0 is an admissible $\mathcal{H}$—and using it corresponds to dynamic programming—but the closer an admissible $\mathcal{H}$ is to $\mathcal{H}^*$, the faster the search is).

3 Adaptation in Case-Based Reasoning

Let $(\mathcal{L}, \models)$ be the logic in which the knowledge containers of the CBR application are defined. A source case $\mathtt{Source} \in \mathcal{L}$ is a case of the case base. Often, such a case represents a specific experience: $\mathcal{M}(\mathtt{Source})$ is a singleton. However, this assumption is not formally necessary (though it has an impact on the complexity of the algorithms). A target case $\mathtt{Target} \in \mathcal{L}$ represents a problem to be solved. This means that there is some missing information about $\mathtt{Target}$ and solving $\mathtt{Target}$ leads to adding information to it. So, adaptation of $\mathtt{Source}$ to solve $\mathtt{Target}$ consists in building a formula $\mathtt{ComplTarget} \in \mathcal{L}$ that makes $\mathtt{Target}$ precise in the sense that it adds information and, therefore, reduces the set of models: $\mathtt{ComplTarget} \models \mathtt{Target}$. To perform adaptation, the domain knowledge $\mathtt{DK} \in \mathcal{L}$ can be used. Therefore, the adaptation process has the following signature:

$$\mathtt{Adaptation} : (\mathtt{DK}, \mathtt{Source}, \mathtt{Target}) \mapsto \mathtt{ComplTarget}$$

$(\mathtt{Source}, \mathtt{Target})$ is called the adaptation problem ($\mathtt{DK}$ is supposed to be fixed).

Of course, this does not completely specify the adaptation process. Several approaches are introduced in the CBR literature. Two of them are presented below, followed by a combination of them.

Revision-Based Adaptation. Let $\dot{+}$ be a revision operator in the logic $(\mathcal{L}, \models)$ used for a given CBR system. The $\dot{+}$-adaptation is defined as follows:

$$\mathtt{ComplTarget} = (\mathtt{DK} \wedge \mathtt{Source}) \dot{+} (\mathtt{DK} \wedge \mathtt{Target})$$

Intuitively, the source case is modified minimally (according to $\dot{+}$) so that it satisfies the target case. Both cases are considered w.r.t. the domain knowledge.

Rule-Based Adaptation. is a general approach to adaptation relying on domain-specific adaptation knowledge in the form of a set $\mathtt{AK}$ of adaptation rules (see, e.g., [12], where adaptation rules are called reformulations). An adaptation rule $\mathtt{R} \in \mathtt{AK}$, when applicable on the adaptation problem $(\mathtt{Source}, \mathtt{Target})$, maps $\mathtt{Source}$ into $\mathtt{ComplTarget}$

which makes $\mathtt{Target}$ precise. Adaptation rules can be composed (or chained): if $\mathtt{R}_1, \mathtt{R}_2, \ldots, \mathtt{R}_q \in \mathtt{AK}$ are such that there exist $q+1$ cases $\mathtt{C}_0$, $\mathtt{C}_1$, …, $\mathtt{C}_q$ verifying:

- $\mathtt{C}_0 = \mathtt{Source}$,
- $\mathtt{C}_q$ makes $\mathtt{Target}$ precise ($\mathtt{C}_q \models \mathtt{Target}$),
- for each $i \in \{1, \ldots, q\}$, $\mathtt{R}_i$ is applicable on $(\mathtt{C}_{i-1}, \mathtt{C}_i)$ and maps $\mathtt{C}_{i-1}$ into $\mathtt{C}_i$,

then $\mathtt{C}_q = \mathtt{ComplTarget}$ is the result of the adaptation of $\mathtt{Source}$ to solve $\mathtt{Target}$. The sequence $\mathtt{R}_1; \mathtt{R}_2; \ldots; \mathtt{R}_q$ is called an *adaptation path*.

Given an adaptation problem $(\mathtt{Source}, \mathtt{Target})$, there may be several adaptation paths to solve it. In order to make a choice among them, a cost function is introduced: $\mathtt{cost} : \mathtt{R} \in \mathtt{AK} \mapsto \mathtt{cost}(\mathtt{R}) > 0$. The cost of an adaptation path is the sum of the costs of its adaptation rules.

In this paper, an adaptation rule $\mathtt{R}$ is defined by two sets of literals, $\mathtt{left}$ and $\mathtt{right}$, and is denoted by $\mathtt{R} = \mathtt{left} \rightsquigarrow \mathtt{right}$. Let $(\mathtt{Source}, \mathtt{Target})$ be an adaptation problem. Several cases have to be considered:

- If $\mathtt{Source}$ is a conjunction of literals represented by a set of literals $\mathtt{L}$, then $\mathtt{R}$ is applicable on $\mathtt{Source}$ if $\mathtt{left} \subseteq \mathtt{L}$. In this situation:

$$\mathtt{R}(\mathtt{Source}) = \mathtt{R}(\mathtt{L}) = (\mathtt{L} \setminus \mathtt{left}) \cup \mathtt{right}$$

- If $\mathtt{Source}$ is in DNF, such that $\mathtt{Source} = \bigvee_i \mathtt{L}_i$, where the $\mathtt{L}_i$'s are conjunctions of literals, $\mathtt{R}$ is applicable on $\mathtt{Source}$ if it is applicable on at least one $\mathtt{L}_i$. Then:

$$\mathtt{R}(\mathtt{Source}) = \bigvee_i \begin{cases} \mathtt{R}(\mathtt{L}_i) & \text{if } \mathtt{R} \text{ is applicable to } \mathtt{L}_i, \\ \mathtt{L}_i & \text{otherwise.} \end{cases}$$

- If $\mathtt{Source}$ is not in DNF, it is replaced by an equivalent formula in DNF.

Given an adaptation rule $\mathtt{R} = \mathtt{left} \rightsquigarrow \mathtt{right}$, $\mathtt{repairs}(\mathtt{R}) = \mathtt{left} \setminus \mathtt{right}$. $\mathtt{left}$ is the set of literals whose presence is necessary to apply $\mathtt{R}$, and is then removed by application of $\mathtt{R}$. Every adaptation rule must be such that $\mathtt{repairs}(\mathtt{R}) \neq \varnothing$.

Combining Rule-Based and Revision-Based Adaptation. Let us consider the following distance on $\mathcal{U} = \mathbb{B}^n$ (for $x, y \in \mathcal{U}$):

$$\delta_{\mathtt{AK}}(x, y) = \inf\{\mathtt{cost}(p) \mid p\text{: adaptation path from } x \text{ to } y \text{ based on rules from } \mathtt{AK}\}$$

(x and y are interpretations that are assimilated to conjunctions of literals). By convention, $\inf \varnothing = +\infty$ and thus, if there is no adaptation path relating x to y, then $\delta_{\mathtt{AK}}(x, y) = +\infty$ and vice-versa. Otherwise, it can be shown that the infimum is always reached and so, $\delta_{\mathtt{AK}}(x, y) = 0$ iff $x = y$ (corresponding to the empty adaptation path).

It has been shown [8] that rule-based adaptation can be simulated by revision-based adaptation with no domain knowledge (i.e., $\mathtt{DK}$ is a tautology) and with the $\dot{+}^{\delta_{\mathtt{AK}}}$ revision operator. When rule-based adaptation fails (no adaptation path from $\mathtt{Source}$ to $\mathtt{Target}$), $\dot{+}^{\delta_{\mathtt{AK}}}$-adaptation gives $\mathtt{ComplTarget}$ equivalent to $\mathtt{Target}$ (no added information).

A failure of rule-based adaptation is due to the fact that no adaptation rules can be composed in order to solve the adaptation problem. In order to have an adaptation that always provides a result, the idea is to add $2n$ adaptation rules, one for each literal: given a literal ℓ, the flip of the literal ℓ is the adaptation rule $\mathtt{F}_\ell = \ell \rightsquigarrow \top$, where $\top$ is the empty conjunction of literals.[2] It can be noticed that a cost is associated to each literal flip and that it may be the case that $\mathtt{cost}(\mathtt{F}_{\neg\ell}) \neq \mathtt{cost}(\mathtt{F}_\ell)$.

Now, let $d_{\mathtt{AK}}$ be the distance on $\mathcal{U} = \mathbb{B}^n$ defined, for $x, y \in \mathcal{U}$, by

$$d_{\mathtt{AK}}(x,y) = \inf \left\{ \mathtt{cost}(p) \;\middle|\; \begin{array}{l} p\text{: adaptation path from } x \text{ to } y \text{ based on} \\ \text{rules from } \mathtt{AK} \text{ and on flips of literals} \end{array} \right\}$$

Let $\dot{+}_{\mathtt{AK}} = \dot{+}^{d_{\mathtt{AK}}}$. $\dot{+}_{\mathtt{AK}}$-adaptation is a revision-based adaptation using the adaptation rules, thus combining rule-based adaptation and revision-based adaptation, and that can take into account domain knowledge.

It can be noticed that if $\mathtt{AK} = \varnothing$ and $\mathtt{cost}(\mathtt{F}_\ell) = 1$ for every literal ℓ then $d_{\mathtt{AK}} = d_H$. Moreover the following assumption is made:

$$\text{For any } \mathtt{R} \in \mathtt{AK},\ \mathtt{cost}(\mathtt{R}) \leq \sum_{\ell \in \mathtt{repairs}(\mathtt{R})} \mathtt{cost}(\mathtt{F}_\ell) \tag{1}$$

In fact, this assumption does not involve any loss of generality: if an adaptation rule violates (1), then it does not appear in any optimal adaptation path, since applying the flips of all the literals of its repair part will be less costly then applying the rule itself.

4 Algorithm of Adaptation Based on Tableaux Repairs

The adaptation algorithm is based on the revision of the source case by the target case, both w.r.t. the domain knowledge. As such, the algorithm performs the revision of a formula ψ by a formula μ, with:

$$\psi = \mathtt{DK} \wedge \mathtt{Source} \qquad \mu = \mathtt{DK} \wedge \mathtt{Target}$$
$$\text{and so:} \qquad \mathtt{ComplTarget} = \psi \dot{+}_{\mathtt{AK}} \mu$$

This section presents the algorithm, which uses a heuristic function $\mathcal{H}$, that is defined and proven to be admissible. Then, an example is detailed. Finally, the termination and the complexity of the algorithm are studied.

4.1 Algorithm

Let φ be a formula and let $\mathtt{branches}(\varphi)$ be the set of branches of the tableaux of φ, or implicants of φ, as a set of sets of literals, that is:

- For any literal ℓ: $\mathtt{branches}(\ell) = \{\{\ell\}\}$;

[2] In the literature, a flip is more frequently a rule of the form $\ell \rightsquigarrow \neg\ell$ but it can be shown that at the definition level, this amounts to the same kind of adaptation (using either $\ell \rightsquigarrow \top$ or $\ell \rightsquigarrow \neg\ell$) but the first form makes explanations easier for the paper.

– For any formulas φ_1 and φ_2:
  $\texttt{branches}(\varphi_1 \vee \varphi_2) = \texttt{branches}(\varphi_1) \cup \texttt{branches}(\varphi_2)$
  $\texttt{branches}(\varphi_1 \wedge \varphi_2) = \{B_1 \cup B_2 \mid B_1 \in \texttt{branches}(\varphi_1), B_2 \in \texttt{branches}(\varphi_2)\}$.

Furthermore, in order to preserve the independence to the syntax of the algorithm, we introduce the function `min-branches`, that converts the output of the `branches` function into the set of prime implicants of the formula, represented as sets of literals.

For any set of literals $\texttt{L}$, let $\texttt{L}^\neg = \{\neg\ell \mid \ell \in \texttt{L}\}$. For example, if $\texttt{L} = \{a, \neg b, \neg c\}$ then $\texttt{L}^\neg = \{\neg a, b, c\}$. $\texttt{L}$ is said to be consistent if and only if $\texttt{L} \cap \texttt{L}^\neg = \varnothing$.

The algorithm is an A* search, where a state is an ordered pair $(\texttt{L}, \texttt{M})$ of consistent sets of literals. Given the propositional formulas ψ and μ, the set of the initial states is:

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

A final state is a state $(\texttt{L}, \texttt{M})$ such that $\texttt{L} \cap \texttt{M}^\neg = \varnothing$ (which is equivalent to the fact that $\texttt{L} \cup \texttt{M}$ is consistent). If $(\texttt{L}, \texttt{M})$ is a non final state, then there exits $\ell \in \texttt{L}$ such that $\ell \in \texttt{L} \cap \texttt{M}^\neg$: there is a clash on ℓ.

The transitions from a state to another state are defined as follows. There is a transition σ from the state $x = (\texttt{L}_x, \texttt{M}_x)$ to the state $y = (\texttt{L}_y, \texttt{M}_y)$ if $\texttt{M}_x = \texttt{M}_y$ and:

– there exists a literal $\ell \in \texttt{L}_x$ such that $\texttt{L}_y = \texttt{F}_\ell(\texttt{L}_x)$ or
– there exists an adaptation rule $\texttt{R}$ such that $\texttt{R}$ is applicable on $\texttt{L}_x$ and $\texttt{L}_y = \texttt{R}(\texttt{L}_x)$.

The cost of a transition σ is the cost of the rule (literal flip or adaptation rule) it uses.

Using a slight variant of the A* algorithm, it is possible to determine the set of least costly transition sequences leading to final states. That is, the algorithm does not stop after finding the first optimal solution, but after finding every other optimal solutions (i.e., the solutions with the same minimal cost). The detailed algorithm is presented in Algorithm 1 and is explained hereafter.

The algorithm first creates the initial states (line 3). For every initial state $\texttt{S}_0$, $\mathcal{G}(\texttt{S}_0) = 0$, that is the cost of reaching $\texttt{S}_0$ is 0. In the case no value of $\mathcal{G}$ is set for a state $\texttt{S}$, $\mathcal{G}(\texttt{S})$ evaluates to $+\infty$, meaning there is no known path from an initial state to $\texttt{S}$. The algorithm then finds a state $\texttt{S}_c$ minimizing $\mathcal{F}(\texttt{S}_c) = \mathcal{G}(\texttt{S}_c) + \mathcal{H}(\texttt{S}_c)$, that is the cost of reaching $\texttt{S}_c$ from an initial state plus the heuristic cost of reaching a final state from $\texttt{S}_c$ (line 10). For each rule or flip σ that can be applied on $\texttt{S}_c$, it creates the state $\texttt{S}_d$ such that $\texttt{S}_c \xrightarrow{\sigma} \texttt{S}_d$. If $\mathcal{G}(\texttt{S}_d) > \mathcal{G}(\texttt{S}_c) + \texttt{cost}(\sigma)$, that is there is no known less or equally costly path to $\texttt{S}_d$, then $\mathcal{G}(\texttt{S}_d)$ is set to $\mathcal{G}(\texttt{S}_c) + \texttt{cost}(\sigma)$. Each generated $\texttt{S}_d$ is added to the set of states to explore, while $\texttt{S}_c$ is removed from it (lines 15 to 20). The algorithm repeats the previous step until it finds a state $\texttt{S}_f$ minimizing $\mathcal{F}(\texttt{S}_f)$ and that is a final state (lines 11 to 13). From this point, the algorithm carries on but ignores any state $\texttt{S}_c$ such that $\mathcal{F}(\texttt{S}_c) > \mathcal{F}(\texttt{S}_f)$, which cannot lead to a less costly solution. It stops when there is no more states that can lead to a less costly solution (line 9). Finally, all final states $\texttt{S}_f$ minimizing $\mathcal{F}(\texttt{S}_f)$ are returned in the form of tableaux branches (line 22), that is for each state $\texttt{S}_f$ defined by $(\texttt{L}_f, \texttt{M}_f)$, the branch containing the literals $\texttt{L}_f \cup \texttt{M}_f$ is returned.

1 $\texttt{revise}(\psi, \mu, \texttt{AK}, \texttt{cost})$
Input:
- ψ and μ, two propositional formulas in NNF. ψ has to be revised by μ.
- $\texttt{AK}$, the set of adaptations rules, used for repairing clashes.
- $\texttt{cost}$, a function that associates to every literal flip and adaptation rule a positive real number.

Output: $\psi \dot{+}_{\texttt{AK}} \mu$ in DNF

2 **begin**
  `// open-states is initiated by initial states with cost 0.`
3 $\texttt{open-states} \leftarrow \texttt{min-branches}(\psi) \times \texttt{min-branches}(\mu)$
4 $\mathcal{G}(\texttt{S})$ evaluates to $+\infty$ by default
5 $\mathcal{G}(\texttt{S}) \leftarrow 0$ for each $\texttt{S} \in \texttt{open-states}$
6 $\mathcal{F}(\texttt{S})$ evaluates to $\mathcal{G}(\texttt{S}) + \mathcal{H}(\texttt{S})$
7 $\texttt{Solutions} \leftarrow \varnothing$
8 $\texttt{solutionCost} \leftarrow +\infty$
9 **while** $\{\texttt{S} \mid \texttt{S} \in \texttt{open-states} \text{ and } \mathcal{F}(\texttt{S}) \leq \texttt{solutionCost}\} \neq \varnothing$ **do**
    `//` $(\texttt{L}_c, \texttt{M}_c)$ `is the current state.`
10 $(\texttt{L}_c, \texttt{M}_c) \leftarrow$ one of the $\texttt{S} \in \texttt{open-states}$ that minimizes $\mathcal{F}(\texttt{S})$
11 **if** $(\texttt{L}_c \cap \texttt{M}_c^{\neg}) = \varnothing$ **then**
12 $\texttt{Solutions} \leftarrow \texttt{Solutions} \cup \{\texttt{L}_c \cup \texttt{M}_c\}$
13 $\texttt{solutionCost} \leftarrow \mathcal{F}((\texttt{L}_c, \texttt{M}_c))$
14 **else**
15 **for each** $\sigma \in \texttt{AK} \cup \{\texttt{F}_\ell \mid \ell$: literal$\}$ such that σ is applicable on $\texttt{L}_c$ **do**
16 $\texttt{open-states} \leftarrow \texttt{open-states} \cup \{(\sigma(\texttt{L}_c),\ \texttt{M}_c)\}$
17 $\mathcal{G}((\sigma(\texttt{L}_c),\ \texttt{M}_c)) \leftarrow \min(\mathcal{G}((\sigma(\texttt{L}_c),\ \texttt{M}_c)),\ \ \mathcal{G}((\texttt{L}_c, \texttt{M}_c)) + \texttt{cost}(\sigma))$
18 **end**
19 **end**
20 $\texttt{open-states} \leftarrow \texttt{open-states} \setminus \{(\texttt{L}_c, \texttt{M}_c)\}$
21 **end**
22 **return** Solutions
23 **end**

Algorithm 1: Algorithm of revision based on tableaux repairs using adaptation knowledge.

4.2 Heuristics

The function $\mathcal{H}$ used in our algorithm is defined by: for $\texttt{S} = (\texttt{L}, \texttt{M})$,$\mathcal{H}(\texttt{S}) = \texttt{ERC}(\texttt{L} \cap \texttt{M}^{\neg})$ where ERC is defined as follows (ERC stands for Estimated Repair Cost):

$$\texttt{ERC}(\{\ell\}) = \min\{\texttt{cost}(\texttt{F}_\ell)\} \cup \left\{ \frac{\texttt{cost}(\texttt{R})}{|\texttt{repairs}(\texttt{R})|} \;\middle|\; \begin{array}{l} \texttt{R} \in \texttt{AK}, \text{ such that} \\ \ell \in \texttt{repairs}(\texttt{R}) \end{array} \right\} \quad \text{for any literal } \ell$$

$$\texttt{ERC}(\texttt{L}) = \sum_{\ell \in \texttt{L}} \texttt{ERC}(\{\ell\}) \qquad \text{for any set of literals } \texttt{L}$$

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

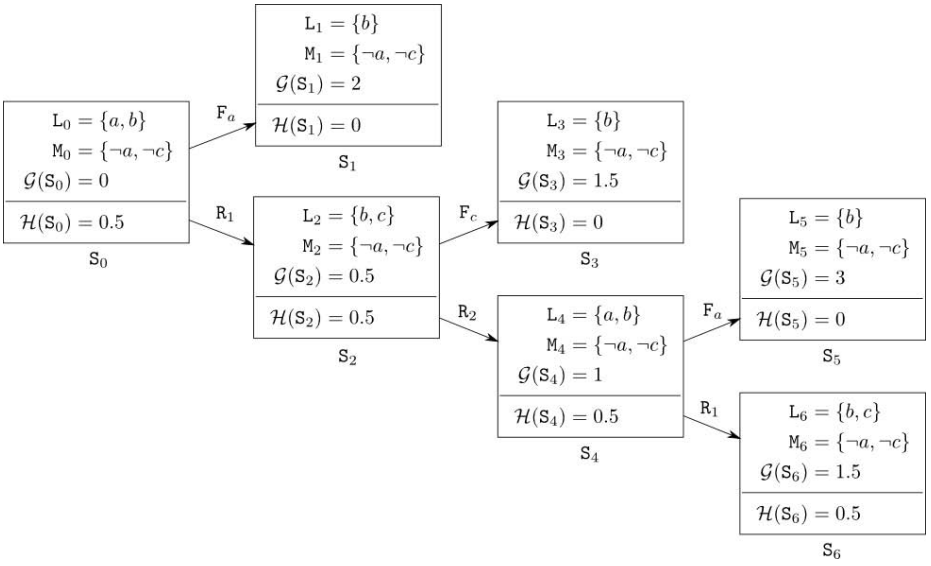

Fig. 1. Application of the algorithm on the running example

4.3 Example

Let us consider the following inputs:

$$\psi = a \wedge b \qquad \mu = \neg a \wedge \neg c$$
$$\mathtt{AK} = \{\mathtt{R}_1, \mathtt{R}_2\}$$
$$\mathtt{R}_1 = a \wedge b \rightsquigarrow b \wedge c \qquad \mathtt{cost}(\mathtt{R}_1) = 0.5$$
$$\mathtt{R}_2 = b \wedge c \rightsquigarrow a \wedge b \qquad \mathtt{cost}(\mathtt{R}_2) = 0.5$$
$$\mathtt{cost}(\mathtt{F}_\ell) = 1 \text{ for } \ell \neq a$$
$$\mathtt{cost}(\mathtt{F}_a) = 2$$

Here, $\mathtt{branches}(\psi) = \{\{a, b\}\}$ and $\mathtt{branches}(\mu) = \{\{\neg a, \neg c\}\}$. Figure 1 shows the different branches developed by the algorithm. At the initial state $\mathtt{S}_0$, the cost is 0. The algorithm repairs the clash on the variable a, for a minimal cost of 0.5. Two branches are developed, the first using a flip on a and the second using the rule R_1. The state $\mathtt{S}_1$ is a final state with a cost of 2. The variable `solutionCost` is set to that cost of 2. That cost is defined as the best final cost. The state $\mathtt{S}_2$ has a clash on the variable c. To repair that clash, the minimal cost is 0.5. Adding the cost of that state, the cost is lower than `solutionCost`. Thus, the clash is repaired to possibly find the solution with a lower final cost. To repair the clash on the variable c, the rule R_2 and the flip $\mathtt{F}_c$ are used. The use of $\mathtt{F}_c$ leads to a final state $\mathtt{S}_3$ with a final cost of 1.5. That cost becomes the best final cost. The use of R_2 gets back to a state $\mathtt{S}_4$ equivalent to the initial state $\mathtt{S}_0$ but with a cost of 1. To repair the clash on the variable a, the minimal cost is 0.5. As the cost to reach that state plus the cost to repair that clash is equal to `solutionCost`, the clash on

the variable a is repaired again. The state $\mathtt{S}_5$ is reached using $\mathtt{F}_a$. That final state is not a best solution because the cost of that state is higher than `solutionCost`. The state $\mathtt{S}_6$ is not final but that branch is not developed because the cost to reach that state plus the cost to repair the clash on the variable c is higher than `solutionCost`. The algorithm returns the solution $L_3 \cup M_3$, i.e. $\{b, \neg a, \neg c\}$.

4.4 Termination and Complexity of the Algorithm

In this section, the termination and complexity of the algorithm are studied when no heuristics is used ($\mathcal{H} = 0$). Since the heuristics introduced above is admissible, the complexity without heuristics is an upper bound for the complexity with heuristics.

Proposition 2. *The algorithm described in this paper always terminates. Its complexity in the worst case is in* $O(4^n \times t^{\varrho+1} \times (n + \varrho \log(t)))$*, where* n *is the number of variables,* $t = |\mathtt{AK}| + |\mathcal{V}|$*, and* ϱ *is the sum of each flip cost divided by the minimum transition cost.*

According to [17], the complexity of a revision operator is $\mathsf{P}^{\mathsf{NP}[O(\log n)]}$-hard[3] hence this high worst-case complexity of the algorithm is not a surprise.

4.5 Optimization

The algorithm can be improved by considering the pairs of rules that commute. That optimization is an application of symmetry breaking constraints [18]. The adaptation rules $\mathtt{R}_1$ and $\mathtt{R}_2$ commute if the two sequences of rules $\mathtt{R}_1$-$\mathtt{R}_2$ and $\mathtt{R}_2$-$\mathtt{R}_1$ are applicable on the same set of formulas and give equivalent results. Formally, $\mathtt{R}_1$ and $\mathtt{R}_2$ commute if, for every set of literals $\mathtt{L}$

- The two following assertions are equivalent:
  - (i) $\mathtt{R}_1$ is applicable on $\mathtt{L}$ and $\mathtt{R}_2$ is applicable on $\mathtt{R}_1(\mathtt{L})$.
  - (ii) $\mathtt{R}_2$ is applicable on $\mathtt{L}$ and $\mathtt{R}_1$ is applicable on $\mathtt{R}_2(\mathtt{L})$.
- and, when (i) (or (ii)) holds, $\mathtt{R}_2(\mathtt{R}_1(\mathtt{L})) = \mathtt{R}_1(\mathtt{R}_2(\mathtt{L}))$.

So, the algorithm is changed as follows. First, an arbitrary total order $\leq$ is defined on the set of rules (including the $\mathtt{F}_\ell$'s). Then, the algorithm only enables the application of a rule $\sigma \in \mathtt{AK} \cup \{\mathtt{F}_\ell \mid \ell$: literal$\}$ on a state such that there is no rule σ' commuting with σ and such that $\sigma' < \sigma$, that has been applied at the previous step to build the current state (this can be changed on line 15).

For example, if $\mathtt{R}_1$ and $\mathtt{R}_2$ commute and $\mathtt{R}_1 < \mathtt{R}_2$, then with the previous version of the algorithm there may be two branches developed in the search space that contain respectively the sequences $\mathtt{R}_1$-$\mathtt{R}_2$ and $\mathtt{R}_2$-$\mathtt{R}_1$ while, without loss of results, the new version of the algorithm generates only the former sequence.

In order to determine whether two adaptation rules commute, the following proposition can be used.

[3] More precisely, the complexity of the Dalal revision operator $\dot{+}_{\text{Dalal}}$ is $\mathsf{P}^{\mathsf{NP}[O(\log n)]}$-complete and the algorithm presented in this paper can be used for computing $\dot{+}_{\text{Dalal}}$: $\dot{+}_{\text{Dalal}} = \dot{+}_{\mathtt{AK}}$ with $\mathtt{AK} = \varnothing$ and $\mathtt{cost}(\mathtt{F}_\ell) = 1$ for each literal ℓ.

Table 1. Average time and standard deviation under REVISOR/PLAK according to d^*, with and without optimization

	REVISOR/PLAK without optimization		REVISOR/PLAK with optimization	
d^*	Average time (ms)	Standard deviation	Average time (ms)	Standard deviation
1	0.521	0.941	0.466	0.415
2	1.965	1.978	1.309	1.146
3	127.993	708.208	10.366	24.77
4	1349.659	2347.341	43.331	54.685

Proposition 3. *Let* $\mathtt{R}_1 = \mathtt{left}_1 \rightsquigarrow \mathtt{right}_1$ *and* $\mathtt{R}_2 = \mathtt{left}_2 \rightsquigarrow \mathtt{right}_2$ *be two adaptation rules.* $\mathtt{R}_1$ *and* $\mathtt{R}_2$ *commute iff the two following conditions hold:*

$$\mathtt{repairs}(\mathtt{R}_1) \cap \mathtt{repairs}(\mathtt{R}_2) = \varnothing$$
$$(\mathtt{right}_1 \cup \mathtt{right}_2) \cap (\mathtt{repairs}(\mathtt{R}_1) \cup \mathtt{repairs}(\mathtt{R}_2)) = \varnothing$$

5 Implementation Issues

5.1 REVISOR/PLAK

REVISOR/PLAK implements the algorithm for revision-based and rule-based adaptation based on tableaux repairs in propositional logic. REVISOR/PLAK has been implemented in Java and is available for download on the site `http://revisor.loria.fr`. Java version 7 is required to launch the software.

So far, empirical data shows that the computing time is largely dependent upon ϱ. However, a million-fold increase of ϱ may result in only a tenfold increase of computing time. The current implementation is vulnerable to sets of rules $\{(\mathtt{left}_i, \mathtt{right}_i) \mid i \in \{0, 1, \ldots, n\}\}$ such that $\mathtt{left}_i \subseteq \mathtt{right}_{i-1}$ for $i \in \{1, \ldots, n\}$ and $\mathtt{left}_0 \subseteq \mathtt{right}_n$, that is, rules that could potentially create infinite paths. While the algorithm always terminates, the algorithm could generate very long finite paths using such rules, if their cost is very small compared to the cost of the solution. The most important factor for computing time is the distance between ψ and μ, d^*. Table 1 and Figure 2 present the average time of computation of REVISOR/PLAK according to d^*, with and without optimization. The tested adaptation problems are for $n = 50$ variables and 40 adaptation rules. Each rule and flip has a cost of 1. The average time are computed on series of 1000 tests, for each value of d^*, on a computer with a 2.60 Ghz processor and 10 GB of available memory. For example, for $d^* = 3$, without optimization REVISOR/PLAK solves the problems in 127.993 ms in average (standard deviation of 708.208) and with optimization in 10.366 ms in average (standard deviation of 24.77). Figure 2 shows that REVISOR/PLAK with optimization is much more efficient.

5.2 A Concrete Example

Bob searches for a pie recipe without cinnamon, egg or pear. No recipe of the case base exactly matches Bob request. The only pie recipe is a recipe of pear pie containing eggs:

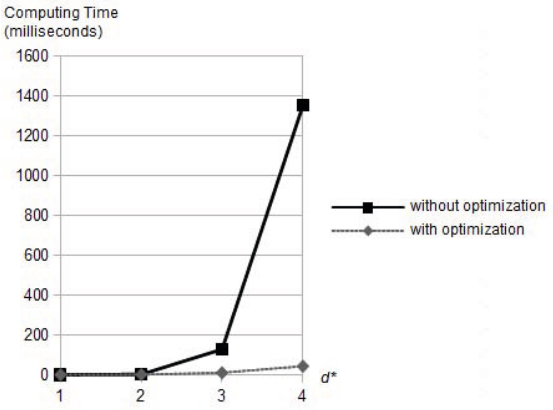

Fig. 2. Average time under REVISOR/PLAK according to d^*, with and without optimization

$$\texttt{Source} = \text{pie} \wedge \text{pie\_shell} \wedge \text{pear} \wedge \text{sugar} \wedge \text{egg}$$
$$\texttt{Target} = \text{pie} \wedge \neg\text{pear} \wedge \neg\text{cinnamon} \wedge \neg\text{egg}$$

REVISOR/PLAK adapts `Source` to solve `Target`, using the following knowledge:

$$\texttt{DK} = (\text{apple} \vee \text{peach} \vee \text{pear}) \Leftrightarrow \text{fruit}$$

(apples, peaches and pears are fruits and, conversely, the only available fruits are apples, peaches and pears)

$$\texttt{AK} = \{\texttt{R}_1, \texttt{R}_2, \texttt{R}_3, \texttt{R}_4, \texttt{R}_5, \texttt{R}_6\}$$
$$\texttt{R}_1 = \text{cake} \wedge \text{egg} \rightsquigarrow \text{cake} \wedge \text{banana}$$
$$\texttt{R}_2 = \text{pie} \wedge \text{egg} \rightsquigarrow \text{pie} \wedge \text{flour} \wedge \text{cider\_vinegar}$$
$$\texttt{R}_3 = \text{pear} \rightsquigarrow \text{peach}$$
$$\texttt{R}_4 = \text{pear} \rightsquigarrow \text{apple} \wedge \text{cinnamon}$$
$$\texttt{R}_5 = \text{cinnamon} \rightsquigarrow \text{orange\_blossom}$$
$$\texttt{R}_6 = \text{cinnamon} \rightsquigarrow \text{vanilla\_sugar}$$

Each rule has a cost of 0.3 except $\texttt{R}_3$ which has a cost of 0.7. Indeed, pears are considered to be more similar to apples than peaches. Each flip has a cost of 1.

REVISOR/PLAK gives the following result (in 1.4 ms and in 1.2 ms with the optimization after the computation of the function `min-branches` in 9.0 ms):

$$\text{pie} \wedge \text{pie\_shell} \wedge \text{sugar} \wedge \text{fruit} \wedge \neg\text{egg} \wedge \text{flour} \wedge \text{cider\_vinegar} \wedge$$
$$\neg\text{pear} \wedge \text{apple} \wedge \neg\text{cinnamon} \wedge (\text{vanilla\_sugar} \vee \text{orange\_blossom})$$

Two recipes are proposed. In both, pears have been replaced by apples, and eggs by flour and cider vinegar. For the cinnamon, two choices were available: either vanilla

sugar or orange blossom could replace it: since $\texttt{cost}(\mathtt{R}_5) = \texttt{cost}(\mathtt{R}_6)$, the disjunction of there possibilities is presented; if $\texttt{cost}(\mathtt{R}_5) < \texttt{cost}(\mathtt{R}_6)$, only the orange blossom alternative would have been given.

6 Conclusion and Future Work

This paper proposes an original algorithm for revision-based and rule-based adaptation based on tableaux repairs in propositional logic. This algorithm modifies minimally the source case so that it becomes consistent with the target case. The tableaux method is applied separately on the source and target cases. Then the consistent branches are combined, which leads to a set of branches, each of them ending with a clash (unless the source case is consistent with the target case and need not to be adapted). Those clashes are repaired with adaptation rules which modify the source case. Adaptation rules allow to substitute a given part of a source case by something. If no rule is available, a flip on the literal leading to the clash deletes it from the source case.

This algorithm has been implemented in the REVISOR/PLAK tool. For each literal ℓ, a heuristic function computes the minimal cost to repair a clash on ℓ. Thus, at each step, the branch developed is the one for which the cost plus the cost to repair the clash is the lowest. All branches whose cost is lower or equal to the best final cost are developed. The algorithm is optimized by considering adaptation rules that commute which experimentally proves to improve the computational efficiency.

Belief revision is one of the operations of belief change. There are other ones (see [19], for a synthesis) such as contraction ($\psi \dot{-} \mu$ is a belief base obtained by minimally modifying ψ so that it does not entail μ) or integrity constraint belief merging [20] ($\Delta_\mu(\{\psi_1, \ldots, \psi_n\})$ is a belief base obtained by minimally modifying the belief bases ψ_i into ψ_i' such that $\mu \wedge \bigwedge_i \psi_i'$ is consistent). A possible line of future work consists of studying how the tableaux repair approach presented in this paper can be modified for such belief change operations. This would have an impact on CBR; in particular, integrity constraint belief merging can be used for multiple case adaptation (i.e., combining several source cases to solve the target case), see [21] for details.

References

1. Riesbeck, C.K., Schank, R.C.: Inside Case-Based Reasoning. Lawrence Erlbaum Associates, Inc., Hillsdale (1989)
2. Aamodt, A., Plaza, E.: Case-based Reasoning: Foundational Issues, Methodological Variations, and System Approaches. AI Communications 7(1), 39–59 (1994)
3. Minor, M., Bergmann, R., Görg, S., Walter, K.: Towards Case-Based Adaptation of Workflows. In: Bichindaritz, I., Montani, S. (eds.) ICCBR 2010. LNCS, vol. 6176, pp. 421–435. Springer, Heidelberg (2010)
4. Manzano, S., Ontañón, S., Plaza, E.: Amalgam-Based Reuse for Multiagent Case-Based Reasoning. In: Ram, A., Wiratunga, N. (eds.) ICCBR 2011. LNCS, vol. 6880, pp. 122–136. Springer, Heidelberg (2011)
5. Coman, A., Muñoz-Avila, H.: Diverse plan generation by plan adaptation and by first-principles planning: A comparative study. In: Díaz-Agudo, B., Watson, I. (eds.) ICCBR 2012. LNCS, vol. 7466, pp. 32–46. Springer, Heidelberg (2012)

6. Rubin, J., Watson, I.: Opponent type adaptation for case-based strategies in adversarial games. In: Díaz-Agudo, B., Watson, I. (eds.) ICCBR 2012. LNCS, vol. 7466, pp. 357–368. Springer, Heidelberg (2012)
7. Lieber, J.: Application of the Revision Theory to Adaptation in Case-Based Reasoning: The Conservative Adaptation. In: Weber, R.O., Richter, M.M. (eds.) ICCBR 2007. LNCS (LNAI), vol. 4626, pp. 239–253. Springer, Heidelberg (2007)
8. Cojan, J., Lieber, J.: Applying belief revision to case-based reasoning. In: Prade, H., Richard, G. (eds.) Computational Approaches to Analogical Reasoning: Current Trends. SCI, vol. 548, pp. 133–162. Springer, Heidelberg (2014)
9. Alchourrón, C.E., Gärdenfors, P., Makinson, D.: On the Logic of Theory Change: partial meet functions for contraction and revision. Journal of Symbolic Logic 50, 510–530 (1985)
10. Cojan, J., Lieber, J.: An Algorithm for Adapting Cases Represented in an Expressive Description Logic. In: Bichindaritz, I., Montani, S. (eds.) ICCBR 2010. LNCS, vol. 6176, pp. 51–65. Springer, Heidelberg (2010)
11. Schwind, C.: From Inconsistency to Consistency: Knowledge Base Revision by Tableaux Opening. In: Kuri-Morales, A., Simari, G.R. (eds.) IBERAMIA 2010. LNCS, vol. 6433, pp. 120–132. Springer, Heidelberg (2010)
12. Melis, E., Lieber, J., Napoli, A.: Reformulation in Case-Based Reasoning. In: Smyth, B., Cunningham, P. (eds.) EWCBR 1998. LNCS (LNAI), vol. 1488, pp. 172–183. Springer, Heidelberg (1998)
13. Personeni, G., Hermann, A., Lieber, J.: Adapting propositional cases based on tableaux repairs using adaptation knowledge – extended report (2014), `http://hal.archives-ouvertes.fr/docs/01/01/17/51/PDF/report_on_revisor_plak.pdf`
14. Marquis, P., Sadaoui, S.: A new algorithm for computing theory prime implicates compilations. In: AAAI/IAAI, vol. 1, pp. 504–509 (1996)
15. Katsuno, H., Mendelzon, A.: Propositional knowledge base revision and minimal change. Artificial Intelligence 52(3), 263–294 (1991)
16. Pearl, J.: Heuristics – Intelligent Search Strategies for Computer Problem Solving. Addison-Wesley Publishing Co., Reading (1984)
17. Eiter, T., Gottlob, G.: On the Complexity of Propositional Knowledge Base Revision, Updates, and Counterfactuals. Artificial Intelligence 57, 227–270 (1992)
18. Gent, I., Petrie, K., Puget, J.-F.: Symmetry in constraint programming. In: Rossi, F., van Beek, P., Walsh, T. (eds.) Handbook for Constraint Programming, ch. 10, pp. 329–376. Elsevier (2006)
19. Peppas, P.: Belief Revision. In: van Harmelen, F., Lifschitz, V., Porter, B. (eds.) Handbook of Knowledge Representation, ch. 8, pp. 317–359. Elsevier (2008)
20. Konieczny, S., Pino Pérez, R.: Merging information under constraints: a logical framework. Journal of Logic and Computation 12(5), 773–808 (2002)
21. Cojan, J., Lieber, J.: Belief Merging-Based Case Combination. In: McGinty, L., Wilson, D.C. (eds.) ICCBR 2009. LNCS, vol. 5650, pp. 105–119. Springer, Heidelberg (2009)