An Algorithm for Adaptation in Case-Based Reasoning

Béatrice Fuchs[1] and **Jean Lieber**[2] and **Alain Mille**[3] and **Amedeo Napoli**[2]

Abstract. The adaptation process is an important and complex step of case-based reasoning (CBR) and is most of the time designed for a specific application. This article presents a domain-independent algorithm for adaptation in CBR. Cases are mapped to a set of numerical descriptors filled with values and local constraint intervals. The algorithm computes every target solution descriptor by combining a source solution, a matching expressed as intervals of variations and dependencies between the source problem and its solution. It determines for every target solution descriptor an interval of the admissible values. In this interval, actual values satisfying global constraints can be chosen. This generic approach to adaptation is operational and introduces general and domain-independent adaptation operators. Therefore, this study is a contribution to the design of a general algorithm for adaptation in CBR.

1 INTRODUCTION

The case based reasoning (CBR) process relies on three main operations: retrieval, adaptation and case memorisation [9, 10]. Adaptation is at the heart of the CBR process and plays a central role. Moreover, adaptation is usually considered as a domain-dependent operation, that is complex and difficult to understand and to apprehend. In this paper we propose a generic and operational view of adaptation that is designed to be (adapted and) reused in the context of "numerical problems", i.e. problems whose characteristics can be described by attributes having partially ordered values. In the following, we study:

(1) A strategy for adaptation based on a matching between the past case and the new case, and on dependencies within the past case;
(2) A general algorithm for adapting a past case to a new case based on the determination of intervals of variations for the attributes of the new case.

We illustrate this general approach to adaptation with a generic and working example. This approach contributes to the design of domain-independent approaches to adaptation. In this way, adaptation, that is usually considered as a crucial and complex task for the CBR process, can be more easily taken into account and implemented in a specific CBR system, using our general purpose algorithm as an implementation guideline. The present study is of main interest for anyone wanting to design a CBR system working on "numerical problems", and, more generally, for anyone wanting to understand more deeply the adaptation mechanism.

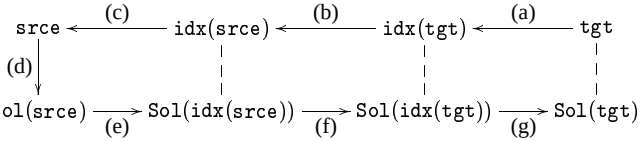
The paper is organised as follows. First, we present the works underlying the present research work, and then the hypotheses and the main principles of the present approach. After that, we propose and detail a domain-independent algorithm for case adaptation. This section is followed by a discussion on the qualities of our approach and its limitations, as well as research perspectives.

[1] Université Lyon 3, IAE-Modeme, 15 quai C. Bernard, 69007 Lyon, France.
[2] LORIA, BP 239, 54506 Vandœuvre-lès-Nancy, France.
[3] Université C. Bernard, Lyon 1, LISI, bâtiment 710, 69622 Villeurbanne.

2 TWO GENERAL MODELS OF ADAPTATION

CBR uses past solved cases, called *source* cases, stored in a case base, in order to solve a new problem, called the *target* problem and denoted by `tgt`. A case describes a problem `pb` and its solution $\texttt{Sol}(\texttt{pb})$, and is denoted by the pair $(\texttt{pb}, \texttt{Sol}(\texttt{pb}))$. A case in the case base is denoted by the pair $(\texttt{srce}, \texttt{Sol}(\texttt{srce}))$ and `srce` is called the source problem. The main steps of the CBR cycle are: *retrieval*, that consists in choosing in the case base a problem `srce` similar to `tgt`, *adaptation*, that consists in reusing the solution $\texttt{Sol}(\texttt{srce})$ of `srce` in order to build $\texttt{Sol}(\texttt{tgt})$ and thus to solve the problem `tgt`, and *case memorisation*, that consists in storing the new solved case $(\texttt{tgt}, \texttt{Sol}(\texttt{tgt}))$ in the case base.

General models of adaptation have already been proposed. A first general model, described in [5], proposes to connect the retrieval and adaptation steps in a unified way illustrated by the following diagram:



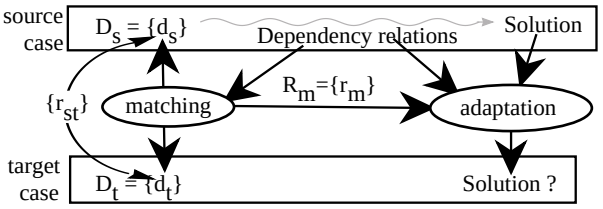
Adapting a case consists in building the list of solution descriptors leading to the satisfaction of a goal, a goal being one of the objectives to be reached in order to build a complete solution. Retrieval begins with the elaboration of an index $\texttt{idx}(\texttt{tgt})$ of `tgt` (step (a)) that consists in selecting relevant descriptors. Starting from this index $\texttt{idx}(\texttt{tgt})$, the index $\texttt{idx}(\texttt{srce})$ of a source case `srce` is retrieved (step (b)) that allows to reach `srce` (step (c)). Step (d) switches from the problem space to the solution space. The adaptation process begins with the generalisation of $\texttt{Sol}(\texttt{srce})$ into $\texttt{Sol}(\texttt{idx}(\texttt{srce}))$ (step (e)), in accordance with step (c). Then step (f) transforms $\texttt{Sol}(\texttt{idx}(\texttt{srce}))$ into $\texttt{Sol}(\texttt{idx}(\texttt{tgt}))$ by an abstraction/specialisation process. Finally, $\texttt{Sol}(\texttt{idx}(\texttt{tgt}))$ is specialised into $\texttt{Sol}(\texttt{tgt})$, in accordance with step (a).

Horizontal lines (a) to (c) correspond to a *similarity path* between problems in the problem space: `srce` and `tgt` are linked with a similarity path composed of a sequence of relations between problems. Horizontal lines (e) to (g) correspond to the application of a sequence of adaptation operators in the solution space. Vertical dashed lines represent dependency relations between a problem and a solution; they express that problem descriptors play a role in the determination of the solution.

A second general model [6] aims at describing the adaptation process at the knowledge level. The matching process is the starting

point of the adaptation process because it provides a set of relations between the source and the target problems. These relations encode the similarities and dissimilarities of the source and the target descriptors that will guide the adaptation process. A matching R_m is a set of triples $r_m = (d_s, d_t, r_{st})$ where d_s is a descriptor of the source problem, d_t is a descriptor of the target problem, and r_{st} is an *explanation* of the relations existing between d_s and d_t.

The adaptation process modifies the set of target solution descriptors given a matching R_m and on a set of dependencies between a source problem and its solution. The following scheme summarises this model:



On the basis of these two models, we propose a general adaptation algorithm describing the transformation of $\texttt{Sol}(\texttt{srce})$ into $\texttt{Sol}(\texttt{tgt})$. This algorithm takes as input a matching between a source problem and a target problem pointing out adaptation objectives, and a set of dependencies expressing the modifications that have to be performed on the appropriate solution descriptors. The algorithm is illustrated by an example presented in the next section.

3 HYPOTHESES AND PRINCIPLES OF THE APPROACH

The adaptation process can be specified as follows: given a source case $(\texttt{srce}, \texttt{Sol}(\texttt{srce}))$, a target problem $\texttt{tgt}$, a matching between $\texttt{srce}$ and $\texttt{tgt}$ and a set of dependencies between $\texttt{srce}$ and $\texttt{Sol}(\texttt{srce})$, what is the solution $\texttt{Sol}(\texttt{tgt})$ of $\texttt{tgt}$?

Our proposal is restricted to a particular problem category of "numerical problems", where the values of the problem attributes can be partially ordered. Thus, this explains why we use numbers for illustrating it. However, the following considerations may be applied to any problem whose attributes have symbolic values that can be partially ordered: a linear extension (i.e. a total order) can always be built on a partial order [4].

The example in the next section is based on a Personal Computer configuration problem, inspired from [2]. The user specifies some needs such as games (G), music (Mu), programming (Pgm), etc., with intervals of numerical values $[min, max]$ expressing the importance of the needs and such that $0 \leq min \leq max \leq 10$. A solution to a PC configuration problem is a list of components satisfying the user's needs: masterboard (Ma), screen (Scr), etc. For example,

$$\text{Mu} \in [2, 2] \text{ means that Mu} = 2.$$
$$\text{Pgm} \in [0, 10] \text{ means that Pgm is unconstrained.}$$

A source problem $\texttt{srce}$ is composed of a set of descriptors, of them describing a constraint on an attribute. For every attribute, there is a single constraint to which is attached an interval of legal values. A descriptor of $\texttt{srce}$ is denoted by d_s; d_s is an element of the set of descriptors $\texttt{srce}$.

The solution $\texttt{Sol}(\texttt{srce})$ is composed of a set of descriptors denoted by D_s, each of them corresponding to an attribute and a value, e.g., Ma $= 8$. The scale of the values is supposed to be the interval $[0, 10]$ for every attribute, and every attribute is supposed to be normalised according to this scale. Ma $= 0$ means the worst masterboard, Ma $= 10$ means the best masterboard of the catalogue, SC $= 0$ means no sound card (since it is possible to have a PC without a sound card, the worst sound card is "no sound card").

The following example of source case will be used to illustrate the algorithm of the next section:

$\texttt{srce}$ =

games	G	$\in$	[7, 9]
music	Mu	$\in$	[0, 0]
word processing	WP	$\in$	[6, 10]
programming	Pgm	$\in$	[5, 8]
image processing	IP	$\in$	[7, 10]
easiness to handle	ETH	$\in$	[8, 10]
price	Pr	$\in$	[1800, 2200]

$\texttt{Sol}(\texttt{srce})$ =

masterboard	Ma	=	8
processor	CPU	=	8
additional pointer	AP	=	6
CD-Rom	CD	=	4
color	Col	=	3
screen	Scr	=	6
sound card	SC	=	0
printer	Ptr	=	5

The source problem is described by a list of pairs attributes-intervals, where each attribute denotes a characteristic of a PC and where the corresponding interval denotes the importance of this characteristic. For example, the pair $(\text{G}, [7, 9])$ means that games are of importance for this PC configuration, while the pair $(\text{Mu}, [0, 0])$ means that the music has no importance at all. Looking at the importance intervals of word processing and programming, one can deduce that this PC configuration is general purpose and, according to the importance of the attribute ETH, that the PC should be rather easy to move.

A *matching* $\texttt{mat}(\texttt{srce}, \texttt{tgt})$ between a source problem $\texttt{srce}$ and a target problem $\texttt{tgt}$ is a set of triples $(d_s, d_t, \texttt{ML})$ where:

- d_s is a descriptor of the source case;
- d_t is the descriptor of the target problem having the same attribute name as d_s and
- $\texttt{ML}$ is a *matching label* expressing the variations of the constraints on attributes:
  $\texttt{ML} = (\Delta_{\texttt{low}}(d_s), \Delta_{\texttt{up}}(d_s))$ where $\Delta_{\texttt{low}}(d_s)$ (respectively, $\Delta_{\texttt{up}}(d_s)$) is the variation from $\texttt{srce}$ to $\texttt{tgt}$ for the lower bound (respectively, upper bound) of the interval.

For example, $(d_s, d_t, \texttt{ML}) \in \texttt{mat}(\texttt{srce}, \texttt{tgt})$ with:

$$d_s = (\text{Pgm} \in [5, 8])$$
$$d_t = (\text{Pgm} \in [6, 7])$$
$$\texttt{ML} = (\Delta_{\texttt{low}}(\text{Pgm}), \Delta_{\texttt{up}}(\text{Pgm})) = (+1, -1)$$

The following example shows a matching between a source problem (left part) and a target problem (right part):

G	$\in$	[7, 9]	$\longrightarrow +1, +1 \longrightarrow$	G	$\in$	[8, 10]
Mu	$\in$	[0, 0]	$\longrightarrow +2, +4 \longrightarrow$	Mu	$\in$	[2, 4]
WP	$\in$	[6, 10]	$\longrightarrow \pm 0, \pm 0 \longrightarrow$	WP	$\in$	[6, 10]
Pgm	$\in$	[5, 8]	$\longrightarrow +1, -1 \longrightarrow$	Pgm	$\in$	[6, 7]
IP	$\in$	[7, 10]	$\longrightarrow -3, -2 \longrightarrow$	IP	$\in$	[4, 8]
ETH	$\in$	[8, 10]	$\longrightarrow +2, \pm 0 \longrightarrow$	ETH	$\in$	[10, 10]
Pr	$\in$	[1800, 2200]	$\longrightarrow +200, +100 \longrightarrow$	Pr	$\in$	[2000, 2300]

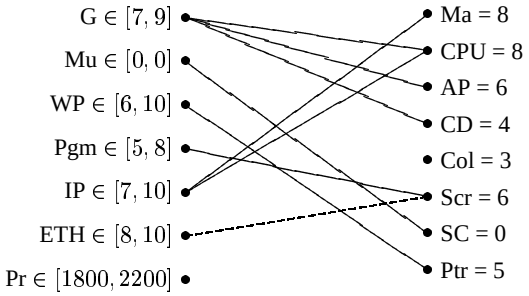
A *dependency* is a triple $(d_s, D_s, \texttt{DL})$ where $d_s \in \texttt{srce}$, $D_s \in \texttt{Sol}(\texttt{srce})$ and $\texttt{DL}$ is a *dependency label*. A dependency label $\texttt{DL}$ is a real value that can be interpreted as follows:

- $\texttt{DL} > 0$ means that the values of the descriptors d_s and D_s vary in the same way;
- $\texttt{DL} < 0$ means that the values of the descriptors d_s and D_s vary in opposite ways and
- $\texttt{DL} = 0$ means that the variations of the values of the descriptors d_s and D_s are independent.

More precisely, $\texttt{DL} = \dfrac{\Delta D_s}{\Delta d_s}$, where Δd_s (respectively, ΔD_s) is a small variation of d_s (respectively, D_s), under the assumption that the other descriptors are constant. In practice, it is easy to know the sign of DL, but the assessment of its value is more difficult. If the assessment of $\dfrac{\Delta D_s}{\Delta d_s}$ is 0, then there is no dependency and D_s does not vary when d_s varies around its value.

The set of dependencies is denoted by $\texttt{Dep}^{\underline{\texttt{cies}}}(\texttt{srce})$. It can be defined by the person who record the source case in the case base. In some CBR systems (see e.g., [7]), $\texttt{Dep}^{\underline{\texttt{cies}}}(\texttt{srce})$ is assessed automatically. Note that the dependencies such that $\texttt{DL} = 0$ are useless, so they do not need to appear in $\texttt{Dep}^{\underline{\texttt{cies}}}(\texttt{srce})$.

The following example shows some dependencies between problem descriptors d_s (at the left) and solution descriptors D_s (at the right):



Links are associated with dependency labels DL: solid lines represent positive dependency labels and the dashed line represents a negative dependency label. For the sake of simplicity, for every positive DL, the value is assumed to be $\texttt{DL} = +1$ and for the negative DL, the value is assumed to be $\texttt{DL} = -1$.

In the above example, we see that there exists solid lines between G in srce and CPU, AP and CD in $\texttt{Sol}(\texttt{srce})$. This means that the value of the descriptor G in the source problem has a direct influence on the values of the descriptors CPU, AP and CD in the solution of this problem. This dependencies are known in advance and are recorded in the source case (they can be seen as *explanations* associated with it, as in [6]).

The output $\texttt{Sol}(\texttt{tgt})$ of the adaptation algorithm is a set of solution descriptors D_t defined as follows. Each D_t is given by an attribute name a taken from the attribute names of $\texttt{Sol}(\texttt{srce})$ (such as Ma or CPU) and a pair $(\texttt{RI}, \texttt{EI})$ of intervals such that:

- $\texttt{RI} = [\texttt{R}_{\texttt{min}}, \texttt{R}_{\texttt{max}}]$, the *restricted interval*, is the interval in which the default value of the attribute a can be chosen and
- $\texttt{EI} = [\texttt{E}_{\texttt{min}}, \texttt{E}_{\texttt{max}}]$, the *extended interval*, is a maximum error interval where the value of a can be chosen.

In the following, a descriptor D_t of $\texttt{Sol}(\texttt{tgt})$ is denoted by $D_t = (a = [\texttt{E}_{\texttt{min}}, [\texttt{R}_{\texttt{min}}, \texttt{R}_{\texttt{max}}], \texttt{E}_{\texttt{max}}])$. The intervals RI and EI computed thanks to the algorithm of the next section satisfy the set inclusion $\texttt{RI} \subseteq \texttt{EI}$.

A *global constraint* expresses a property that must be satisfied by the solution. For example, the sum of the prices of the different configuration elements must be in the price interval specified in the target problem. Another example of global constraint is linked with the masterboard: it must accept the other components (processor, sound card, etc.). Global constraints are not formalised because the algorithm does not detail how they are used in the adaptation process. However, we show at the end of section 4 how global constraints can be taken into account.

4 AN ALGORITHM FOR CASE ADAPTATION

The algorithm for adaptation presented in this section considers each descriptor $D_s \in \texttt{Sol}(\texttt{srce})$ and adapt it in taking into account the dependencies and the matching between srce and tgt. The function **adapt_descriptor** performs the adaptation of D_s and returns $D_t = (a = [\texttt{E}_{\texttt{min}}, [\texttt{R}_{\texttt{min}}, \texttt{R}_{\texttt{max}}], \texttt{E}_{\texttt{max}}])$, where a is an attribute name and $[\texttt{E}_{\texttt{min}}, \texttt{E}_{\texttt{max}}]$ and $[\texttt{R}_{\texttt{min}}, \texttt{R}_{\texttt{max}}]$ denote the extended and the restricted intervals of variation of the value of a. Before giving the algorithm of **adapt_descriptor**, three examples are presented in order to make explicit the steps of this algorithm.

1st example. Let us consider $D_s = (\text{Col} = 3) \in \texttt{Sol}(\texttt{srce})$. This descriptor does not depend on any descriptor of srce. Thus, the variations from srce to tgt do not cause any variation on this descriptor. Hence, it is considered that the value 3 can be preserved for the attribute Col in $\texttt{Sol}(\texttt{tgt})$. However, as there is no dependency on Col, any other value in $[0, 10]$ is allowed. So, $\texttt{RI} = [3, 3]$ and $\texttt{EI} = [0, 10]$ and thus $\text{Col} = [0, [3, 3], 10]$ is the adapted descriptor.

2nd example. $D_s = (\text{Ma} = 8)$ is an example of a descriptor of $\texttt{Sol}(\texttt{srce})$ depending on only one descriptor of srce, namely $d_s = (\text{IP} \in [7, 10])$. The label of this dependency is $\texttt{DL} = +1$. The matching label associated with IP is $\texttt{ML} = (\Delta_{\texttt{low}}(\text{IP}), \Delta_{\texttt{up}}(\text{IP})) = (-3, -2)$ (see above the description of the current target problem). The dependency label $\texttt{DL} = \dfrac{\Delta D_s}{\Delta d_s} = \dfrac{\Delta \text{Ma}}{\Delta \text{IP}}$ entails two variations for Ma, one for $\Delta_{\texttt{low}}(\text{IP})$ and one for $\Delta_{\texttt{up}}(\text{IP})$:

$$\Delta_{\texttt{low}}(\text{Ma}) = \texttt{DL} \cdot \Delta_{\texttt{low}}(\text{IP}) = (+1) \cdot (-3) = -3$$
$$\Delta_{\texttt{up}}(\text{Ma}) = \texttt{DL} \cdot \Delta_{\texttt{up}}(\text{IP}) = (+1) \cdot (-2) = -2$$

According to this reasoning, the proposed value for Ma in $\texttt{Sol}(\texttt{tgt})$ lies between $\text{Ma} = 8 + \Delta_{\texttt{low}}(\text{IP}) = 5$ and $\text{Ma} = 8 + \Delta_{\texttt{up}}(\text{IP}) = 6$. This provides a first interval of values for Ma in $\texttt{Sol}(\texttt{tgt})$: $\texttt{RI} = [5, 6]$.

The determination of RI is mainly based on the value of DL which can be difficult to assess exactly in practice. If now it is assumed that the sign of DL –whether $\texttt{DL} < 0$ or $\texttt{DL} > 0$– is known with certainty, the signs of $\Delta_{\texttt{low}}(\text{Ma})$ and $\Delta_{\texttt{up}}(\text{Ma})$ are also known with certainty, though their values are not. In the example, these two signs are negative, involving that the variation of Ma from $\texttt{Sol}(\texttt{srce})$ to $\texttt{Sol}(\texttt{tgt})$ is negative. Therefore, the value of Ma in $\texttt{Sol}(\texttt{tgt})$ is lower than the value of Ma in $\texttt{Sol}(\texttt{srce})$. Thus, according to this reasoning, the interval of values of Ma in $\texttt{Sol}(\texttt{tgt})$ that has to be chosen is $\texttt{EI} = [0, 8]$. Indeed, 8 is the value associated with Ma in $\texttt{Sol}(\texttt{srce})$ and 0 is the lowest possible value. Finally, $\text{Ma} = [0, [5, 6], 8]$ is the adapted descriptor D_t.

3d example. $D_s = (\text{Scr} = 6)$ is an example of descriptor of $\texttt{Sol}(\texttt{srce})$ depending on two descriptors of srce, namely $(\text{Pgm} \in [5, 8])$ with $\texttt{DL}_{\text{Pgm}} = +1$ and $(\text{ETH} \in [8, 10])$ with $\texttt{DL}_{\text{ETH}} = -1$. If D_s depended only on $(\text{Pgm} \in [5, 8])$ then the low variation of Scr would be

$$\Delta_{\texttt{low},\text{Pgm}}(\text{Scr}) = \texttt{DL}_{\text{Pgm}} \cdot \Delta_{\texttt{low}}(\text{Pgm}) = (+1) \cdot (+1) = +1$$

If D_s depended only on $(\text{ETH} \in [8, 10])$ then the low variation of Scr would be

$$\Delta_{\texttt{low},\text{ETH}}(\text{Scr}) = \texttt{DL}_{\text{ETH}} \cdot \Delta_{\texttt{low}}(\text{ETH}) = (-1) \cdot (+2) = -2$$

The variation $\Delta_{\texttt{low}}(\text{Scr})$ is computed by the sum of the above variations:

$$\Delta_{\texttt{low}}(\text{Scr}) = \Delta_{\texttt{low},\text{Pgm}}(\text{Scr}) + \Delta_{\texttt{low},\text{ETH}}(\text{Scr}) = -1$$

The variation $\Delta_{\texttt{up}}(\text{Scr})$ is computed in a similar way:

$$\begin{array}{rcccccl} \Delta_{\texttt{up}}(\text{Scr}) & = & \Delta_{\texttt{up},\text{Pgm}}(\text{Scr}) & + & \Delta_{\texttt{up},\text{ETH}}(\text{Scr}) & & \\ & = & (+1) \cdot (-1) & + & (-1) \cdot (\pm 0) & = & -1 \end{array}$$

Since $D_s = (\text{Scr} = 6)$ is the descriptor to be adapted, the above variations give the interval $\texttt{RI} = [5, 5]$.

$\texttt{EI}$ is computed thanks to the signs of the four values of variation $\Delta_{\texttt{low},\text{Pgm}}(\text{Scr})$, $\Delta_{\texttt{low},\text{ETH}}(\text{Scr})$, $\Delta_{\texttt{up},\text{Pgm}}(\text{Scr})$ and $\Delta_{\texttt{up},\text{ETH}}(\text{Scr})$. These signs are respectively $+$, $-$, $-$ and 0. If all these signs were positive (respectively, negative) then a positive variation (respectively, a negative variation) of the value of Scr from $\texttt{Sol}(\texttt{srce})$ to $\texttt{Sol}(\texttt{tgt})$ would have given an interval $\texttt{EI} = [6, 10]$ (respectively, $\texttt{EI} = [0, 6]$). This is not the case, so the way this value changes is undefined and thus, $\texttt{EI} = [0, 10]$. Finally, $\text{Scr} = [0, [5, 5], 10]$ is the adapted descriptor.

The algorithm for adaptation of a single descriptor of $\texttt{Sol}(\texttt{srce})$, generalised from these three examples, is the following:

adapt_descriptor *(adaptation of one descriptor)*

Input: • $D_s = (a = V) \in \texttt{Sol}(\texttt{srce})$ (a: attribute, V: value);
• $\texttt{Dep}$: a set of dependencies relating the descriptors of d_s to the descriptor D_s;
• $\texttt{mat}(\texttt{srce}, \texttt{tgt})$, a matching between $\texttt{srce}$ and $\texttt{tgt}$, i.e., a set of triples $(d_s, d_t, \texttt{ML})$.
Output: a solution descriptor $D_t = (a = (\texttt{RI}, \texttt{EI}))$.

begin (algorithm)
  if $\texttt{Dep} = \emptyset$ /* *no dependency, as in 1st example* */
    then return $(a = ([V, V], [0, 10]))$
      /* $\texttt{RI} = [V, V]$ *and* $\texttt{EI} = [0, 10]$. */
  $\Delta_{\texttt{low}}(D_s) \leftarrow 0$
  $\Delta_{\texttt{up}}(D_s) \leftarrow 0$
  $\texttt{signs} \leftarrow \emptyset$ /* *The set of signs is initially empty* */
  for each $(d_s, D_s, \texttt{DL}) \in \texttt{Dep}$
    Let d_t and $\texttt{ML} = (\Delta_{\texttt{low}}(d_s), \Delta_{\texttt{up}}(d_s))$ such that $(d_s, d_t, \texttt{ML}) \in \texttt{mat}(\texttt{srce}, \texttt{tgt})$
    $\Delta_{\texttt{low}}(D_s) \leftarrow \Delta_{\texttt{low}}(D_s) + \texttt{DL} \cdot \Delta_{\texttt{low}}(d_s)$
    $\Delta_{\texttt{up}}(D_s) \leftarrow \Delta_{\texttt{up}}(D_s) + \texttt{DL} \cdot \Delta_{\texttt{up}}(d_s)$
    Compute the signs of $\texttt{DL} \cdot \Delta_{\texttt{low}}(d_s)$ and $\texttt{DL} \cdot \Delta_{\texttt{up}}(d_s)$ ($-$, $+$ or 0 for negative, positive or null values) and add them to the set $\texttt{signs}$.
  end (for each)
  $ri_1 \leftarrow V + \Delta_{\texttt{low}}(D_s)$
  $ri_2 \leftarrow V + \Delta_{\texttt{up}}(D_s)$
  $\texttt{RI} \leftarrow [\min\{ri_1, ri_2\}, \max\{ri_1, ri_2\}]$
  if $\texttt{signs} = \{0\}$ /* *all the signs are null* */
    then $\texttt{EI} \leftarrow [V, V]$
  if $\texttt{signs} = \{+\}$ **or** $\texttt{signs} = \{0, +\}$ /* *all the signs are positive* */
    then $\texttt{EI} \leftarrow [V, 10]$
  if $\texttt{signs} = \{-\}$ **or** $\texttt{signs} = \{-, 0\}$ /* *all the signs are negative* */
    then $\texttt{EI} \leftarrow [0, V]$
  if $\texttt{signs} = \{-, +\}$ **or** $\texttt{signs} = \{-, 0, +\}$ /* *other situations* */
    then $\texttt{EI} \leftarrow [0, 10]$
  return $(a = (\texttt{RI}, \texttt{EI}))$
end (algorithm)

The main algorithm of adaptation is the following:

adaptation *(of a whole case)*

Input: • $(\texttt{srce}, \texttt{Sol}(\texttt{srce}))$, a source case;
• $\texttt{Dep}^{\texttt{cies}}(\texttt{srce})$, the set of dependencies between $\texttt{srce}$ and $\texttt{Sol}(\texttt{srce})$, i.e., a set of triples $(d_s, D_s, \texttt{DL})$;
• $\texttt{tgt}$, a target problem;
• $\texttt{mat}(\texttt{srce}, \texttt{tgt})$, a matching between $\texttt{srce}$ and $\texttt{tgt}$.
Output: a solution $\texttt{Sol}(\texttt{tgt})$ of $\texttt{tgt}$.

begin (algorithm)
  $\texttt{Sol}(\texttt{tgt}) \leftarrow \emptyset$
  for each $D_s \in \texttt{Sol}(\texttt{srce})$
    Let $\texttt{Dep}$ be the set of the dependencies between descriptors of $\texttt{srce}$ and D_s:
    $\texttt{Dep} \leftarrow \{(d, D, \texttt{DL}) \in \texttt{Dep}^{\texttt{cies}}(\texttt{srce}) \mid D = D_s\}$
    $D_t \leftarrow$ **adapt_descriptor**$(D_s, \texttt{Dep}, \texttt{mat}(\texttt{srce}, \texttt{tgt}))$
    $\texttt{Sol}(\texttt{tgt}) \leftarrow \texttt{Sol}(\texttt{tgt}) \cup \{D_t\}$
  end (for each)
  return $\texttt{Sol}(\texttt{tgt})$
end (algorithm)

After the execution of this algorithm, the values associated with the attributes of $\texttt{Sol}(\texttt{srce})$ are given by the intervals $\texttt{RI}$ and $\texttt{EI}$. For the example, the result is:

$$\texttt{Sol}(\texttt{tgt}) = \begin{array}{lcc} \text{Ma} & = & [0, [5, 6], 8] \\ \text{CPU} & = & [0, [6, 7], 10] \\ \text{AP} & = & [6, [7, 7], 10] \\ \text{CD} & = & [4, [5, 5], 10] \\ \text{Col} & = & [0, [3, 3], 10] \\ \text{Scr} & = & [0, [5, 5], 10] \\ \text{SC} & = & [0, [2, 4], 10] \\ \text{Ptr} & = & [5, [5, 5], 5] \end{array}$$

The choice of a precise value associated with each attribute of $\texttt{Sol}(\texttt{tgt})$ remains to be done. This choice must be made in accordance with the global and local constraints. As an example of global constraint, the price of the PC has to satisfy the relation $\text{Pr} \in [2000, 2300]$ specified in $\texttt{tgt}$, the price being calculated by the sum of the prices of each component. The local constraints are given by the intervals $\texttt{EI}$. For example, the value of the masterboard must be chosen in $\texttt{EI} = [0, 6]$. The interval $\texttt{RI}$ gives some more accurate suggestions for the choice of the value: since $\texttt{RI} = [5, 6]$ the values between 5 and 6 are suggested for Ma.

A last remark remains to be done about the interpretation of the results of the adaptation algorithm for the current problem, i.e. the PC configuration problem. The adaptation operations being described are substitutions of values. In certain situations, these substitutions can be interpreted as additions or removals of components of the PC. For example, $(\text{SC} = 0) \in \texttt{Sol}(\texttt{srce})$ can be substituted by $(\text{SC} = 3) \in \texttt{Sol}(\texttt{tgt})$. Since $\text{SC} = 0$ means "no sound card", the substitution of 0 by 3 is interpreted, in this context, by the addition of a sound card.

A Few words about case retrieval. A classical approach of retrieval [9] consists in choosing a source case minimising the differences between the source and target problems, i.e. the $|\Delta_{\mathtt{low}}(d_s)|$ and the $|\Delta_{\mathtt{up}}(d_s)|$ in the present work. By contrast, according to the adaptation-guided retrieval philosophy [11], the differences to be minimised are between the solution of the target problem (to be built) and the solution of the source problem, i.e. the $|\Delta_{\mathtt{low}}(D_s)|$ and the $|\Delta_{\mathtt{up}}(D_s)|$. The variations of the solution descriptors are linked with the variations of the problem descriptors by the dependencies. For example, minimising $|\Delta_{\mathtt{low}}(\mathrm{Ma})|$ consists in minimising $|\mathtt{DL} \cdot \Delta_{\mathtt{low}}(\mathrm{IP})|$.

This suggests that the dependency labels should be used for choosing the weights of the problem attributes in a similarity metric: the more a problem descriptor influences the solution, the more important it is. This can be likened to the use of foot-printing similarity metrics defined in [12]. A precise and detailed study of retrieval guided by the present adaptation procedure must still be carried on.

5 DISCUSSION

Adaptation has been studied under different points of view. Among operational points of view are adaptation-guided retrieval [11], adaptation as configuration [13] and adaptation as planning [5]. There are other works on adaptation, especially on adaptation knowledge, but the three preceding approaches are the main inspiration sources for the present work.

In all these works, the main concern is to design a practical and domain-independent approach to adaptation. The works mentioned above are general purpose and defined at an abstract level. By contrast, in the present research work, we try to introduce a practical domain-independent and working approach to adaptation. The important features that can be enlightened and that are also mostly present in the other approaches are: (1) matching between the source and the target descriptors, (2) dependencies between the source problem and its solution and (3) transformation on the descriptors and on the cases. Depending on the existing positive, null or negative dependencies, the value of a source descriptor is changed in the target case accordingly. Moreover, descriptors are manipulated one by one in the source case (this corresponds to a simple case decomposition).

Regarding other approaches to adaptation, and especially the description of adaptation knowledge (see for example [3], [10], and [9]), we "implement" in our proposal the main adaptation operators that are commonly mentioned: substitution, transformation (including copy), composition (decomposition) and specialisation (generalisation), even if these operations appear under a simplified form. In this way, we have proposed a domain-independent algorithm for adaptation, contributing to show that adaptation can be automated (at least in more situations than usually accepted).

However, although being operational, simple and easy to "adapt", our approach suffers some limitations, mainly due to the fact that we only deal with numerical descriptors, and that, in the terms of [13], we perform simple adaptation, i.e. apply adaptation operators on simple attributes valued with partially ordered values. A more precise study has to be carried out to take into account more complex adaptation operators such as composition/decomposition or specialisation/generalisation. One way to improve our approach is to unify the work presented in [5] about adaptation as planning with the present operational work, taking advantage of other recent advances on adaptation (such as the one presented in [13]). The extension of the algorithm can rely on the design of adaptation rules (controlling the application of adaptation operators) and a general strategy for rule application (an algorithm) based on the notions of adaptation as planning and similarity paths. As a final but important perspective, our approach remains to be fully implemented and tested in order to be faced with realistic problems.

6 CONCLUSION

In this paper, we have presented an operational and domain-independent approach to adaptation in CBR. This approach is mainly based on matching between source and target problems and on dependencies within the source case. It can be used for problems whose characteristics can be described by attributes having numerical or partially ordered values. This approach has the advantage of being general and easy to understand and to reuse. In particular, this approach fits well with the "real world" application described in [8]. However, work remains still to be done to extend this approach to a wider category of problems and to the exploitation of complex adaptation knowledge.

REFERENCES

[1] *Case-Based Reasoning Reasearch and Development — Third International Conference on Case-Based Reasoning (ICCBR-99)*, eds., K.-D. Althoff, R. Bergmann, and L. K. Branting, Lecture Notes in Artificial Intelligence 1650, Springer, Berlin, 1999.

[2] R. Bergmann and W. Wilke, 'Towards a New Formal Model of Transformational Adaptation in Case-Based Reasoning', in *Proceedings of the 13th European Conference on Artificial Intelligence (ECAI-98), Brighton, United Kingdom*, ed., H. Prade, pp. 53–57, (1998).

[3] J. G. Carbonell, 'Learning by analogy: Formulating and generalizing plans from past experience', in *Machine Learning, An Artificial Intelligence Approach*, ed., R. S. Michalski and J. G. Carbonell and T. M. Mitchell, chapter 5, 137–161, Tioga Press, (1983).

[4] B. A. Davey and H. A. Priestley, *Introduction to Lattices and Order*, Cambridge University Press, Cambridge, UK, 1990.

[5] B. Fuchs, J. Lieber, A. Mille, and A. Napoli, 'Towards a Unified Theory of Adaptation in Case-Based Reasoning', In Althoff et al. [1], pp. 104–117.

[6] B. Fuchs and A. Mille, 'A Knowledge-Level Task Model of Adaptation in Case-Based Reasoning', In Althoff et al. [1], pp. 118–131.

[7] B. Fuchs, A. Mille, and B. Chiron, 'Operator Decision aiding by Adaptation of Supervision Strategies', in *First International Conference on Case-Based Reasoning - ICCBR-95*, eds., M. Veloso and A. Aamodt, Lecture Notes in Artificial Intelligence 1010, pp. 23–32. Springer, Berlin, (1995).

[8] O. Herbeaux and A. Mille, 'ACCELERE: a Case-Based Design Assistant for Closed Cell Rubber Industry', *Knowledge-Based Systems*, **12**, 231—238, (1999).

[9] J. Kolodner, *Case-Based Reasoning*, Morgan Kaufmann, Inc., 1993.

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

[11] B. Smyth and M. T. Keane, 'Retrieving Adaptable Cases', in *Topics in Case-Based Reasoning – First European Workshop (EWCBR'93), Kaiserslautern*, eds., S. Wess, K.-D. Althoff, and M. M. Richter, Lecture Notes in Artificial Intelligence 837, 209–220, Springer, Berlin, (1994).

[12] M. M. Veloso, *Planning and Learning by Analogical Reasoning*, Lecture Notes in Artificial Intelligence 886, Springer, Berlin, 1994.

[13] W. Wilke, B. Smyth, and P. Cunningham, 'Using Configuration Techniques for Adaptation', in *Case-Based Reasoning Technologies. From Foundations to Applications*, eds., M. Lenz, B. Bartsch-Spörl, H.-D. Burkhard, and S. Wess, Lecture Notes in Artificial Intelligence 1400, chapter 6, 139–168, Springer, (1998).