An Algorithm for Adaptation in Case-based Reasoning
Béatrice Fuchs, Jean Lieber, Alain Mille, Amedeo Napoli

To cite this version:

HAL Id: inria-00099035
https://inria.hal.science/inria-00099035
Submitted on 26 Sep 2006

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
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.

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 Sol(pb), and is denoted by the pair (pb, Sol(pb)). A case in the case base is denoted by the pair (src, Sol(src)) and src is called the source problem. The main steps of the CBR cycle are: retrieval, that consists in choosing in the case base a problem src similar to tgt, adaptation, that consists in using the solution Sol(src) of src in order to build Sol(tgt) and thus to solve the problem tgt, and case memorisation, that consists in storing the new solved case (tgt, Sol(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:

```
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.

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 Sol(pb), and is denoted by the pair (pb, Sol(pb)). A case in the case base is denoted by the pair (src, Sol(src)) and src is called the source problem. The main steps of the CBR cycle are: retrieval, that consists in choosing in the case base a problem src similar to tgt, adaptation, that consists in using the solution Sol(src) of src in order to build Sol(tgt) and thus to solve the problem tgt, and case memorisation, that consists in storing the new solved case (tgt, Sol(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 idx(tgt) of tgt (step (a)) that consists in selecting relevant descriptors. Starting from this index idx(tgt), the index idx(src) of a source case src is retrieved (step (b)) that allows to reach src (step (c)). Step (d) switches from the problem space to the solution space. The adaptation process begins with the generalisation of Sol(src) into Sol(idx(src)) (step (e)), in accordance with step (c). Then step (f) transforms Sol(idx(src)) into Sol(idx(tgt)) by an abstraction/specialisation process. Finally, Sol(idx(tgt)) is specialised into Sol(tgt), in accordance with step (a).

Horizontal lines (a) to (c) correspond to a similarity path between problems in the problem space: src 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
adaptive 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 \( \text{Sol}(\text{src}) \) into \( \text{Sol}(\text{tgt}) \). 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 HYPOTHÉSES AND PRINCIPLES OF THE APPROACH

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

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 components satisfying the user’s needs: masterboard (Ma), screen (Scr), etc. For example, to any problem whose attributes have symbolic values that 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 components satisfying the user’s needs: masterboard (Ma), screen (Scr), etc. For example, to any problem whose attributes have symbolic values that 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 solution. For example, the sum of the prices of the different con-

tinuous components are constant. In practice, it is easy to know the

dependencies are known in advance and are

value of the descriptor G in the source problem has a direct influence

I^! > 0 means that the values of the descriptors D_ and D_{a} vary in

I^! < 0 means that the values of the descriptors D_ and D_{a} vary in

I^! = 0 means that the variations of the values of the descriptors

descriptor. Hence, it is considered that the value 3 can be preserved

More precisely, I^! = \frac{\Delta D_{a}}{\Delta d_{a}}, where \Delta d_{a} (respectively, \Delta D_{a}) is

a small variation of d_{a} (respectively, D_{a}), under the assumption that

the other descriptors are constant. In practice, it is easy to know the

sign of I^!, but the assessment of its value is more difficult. If the

assessment of \frac{\Delta D_{a}}{\Delta d_{a}} is 0, then there is no dependency and D_{a} does

not vary when d_{a} varies around its value.

The set of dependencies is denoted by Dep\(\omega\)\(\text{oce}\). It can be

defined by the person who record the source case in the case base.

In some CBR systems (see e.g., [7]), Dep\(\omega\)\(\text{oce}\) is assessed automatically.

Note that the dependencies such that I^! = 0 are useless, so they do not need to appear in Dep\(\omega\)\(\text{oce}\).

The following example shows some dependencies between prob-

lem descriptors d_{a} (at the left) and solution descriptors D_{a} (at the right):

\[
\begin{align*}
G & \in [7, 9] & Ma & = 8 \\
Mu & \in [0, 10] & CPU & = 8 \\
WP & \in [6, 10] & AP & = 6 \\
Pgm & \in [5, 8] & CD & = 4 \\
IP & \in [7, 10] & Col & = 3 \\
ETH & \in [8, 10] & Scr & = 6 \\
Pr & \in [1800, 2200] & SC & = 0 \\
& & Pr & = 5
\end{align*}
\]

Links are associated with dependency labels I^!: solid lines repres-

ent positive dependency labels and the dashed line the case base

represents a negative dependency label. For the sake of simplicity, for every posi-

tive I^!, the value is assumed to be I^! = +1 and for the negative I^!, the value is assumed to be I^! = -1.

In the above example, we see that there exists solid lines between

G in \textsc{arce} and CPU, AP and CD in \textsc{sol(arce)}. 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. These dependencies are known in advance and are

recorded in the source case (they can be seen as explanations associ-

ated with it, as in [6]).

The output \textsc{sol(tgt)} of the adaptation algorithm is a set of so-

lution descriptors D_{t} defined as follows. Each D_{t} is given by an

attribute name a taken from the attribute names of \textsc{sol(arce)} (such as

Ma or CPU) and a pair (RI, EI) of intervals such that

- RI = [R_{\text{min}}, R_{\text{max}}], the restricted interval, is the interval in which

  the default value of the attribute a can be chosen and

- EI = [R_{\text{min}}, R_{\text{max}}], the extended interval, is a maximum error inter-

  val where the value of a can be chosen.

In the following, a descriptor D_{t} of \textsc{sol(tgt)} is denoted by

D_{t} = (a = [R_{\text{min}}, [R_{\text{min}}, R_{\text{max}}], [R_{\text{max}}]), the intervals RI and EI computed thanks to the algorithm of the next section satisfy the set inclu-

sion RI \subseteq EI.

A global constraint expresses a property that must be satisfied by

the solution. For example, the sum of the prices of the different con-

figuration elements must be in the price interval specified in the tar-

get 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 algo-

rithm does not detail how they are used in the adaptation process.

However, we show at the end of section 4 how global constraints can

taken into account.

4 AN ALGORITHM FOR CASE ADAPTATION

The algorithm for adaptation presented in this section considers each

descriptor D_{a} \in \textsc{sol(arce)} and adapt it in taking into account the

dependencies and the matching between \textsc{arce} and \textsc{tgt}. The func-

tion adapt\(\text{ descriptor}\) performs the adaptation of D_{a} and returns

D_{a} = (a = [R_{\text{min}}, [R_{\text{min}}, R_{\text{max}}], [R_{\text{max}}]), where a is an attribute name and

[R_{\text{min}}, R_{\text{max}}] and [R_{\text{min}}, R_{\text{max}}] denote the extended and the restricted

intervals of variation of the value of a. Before giving the algorithm

of adapt\(\text{ descriptor}\), three examples are presented in order to make

explicit the steps of this algorithm.

1st example. Let us consider D_{a} = (Col = 3) \in \textsc{sol(arce)}.

This descriptor does not depend on any descriptor of \textsc{arce}. Thus,

the variations from \textsc{arce} to \textsc{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 \textsc{sol(tgt)}. However, as there is no dependency

on Col, any other value in [0, 10] is allowed. So, RI = [3, 3] and

EI = [0, 10] and thus Col = [0, 3, 10] is the adapted descriptor.

2nd example. D_{a} = (Ma = 8) is an example of a descriptor of

\textsc{sol(arce)} depending on only one descriptor of \textsc{arce}, namely

d_{a} = (IP \in [7, 10]). The label of this dependency is I^! = +1. The

matching label associated with IP is I^! = (\Delta_{\text{IP}}(IP), \Delta_{\text{IP}}(IP)) =

(-3, -2) (see above the description of the current target problem).

The dependency label I^! = \frac{\Delta D_{a}}{\Delta d_{a}} = \frac{\Delta Ma}{\Delta IP}

tells two variations for Ma, one for \Delta_{\text{IP}}(IP) and one for \Delta_{\text{IP}}(IP):

\[
\Delta_{\text{IP}}(Ma) = I^! \cdot \Delta_{\text{IP}}(IP) = (+1) \cdot (-3) = -3 \\
\Delta_{\text{IP}}(Ma) = I^! \cdot \Delta_{\text{IP}}(IP) = (+1) \cdot (-2) = -2
\]

According to this reasoning, the proposed value for Ma in \textsc{sol(tgt)}

lies between Ma = 8 + \Delta_{\text{IP}}(IP) = 5 and Ma = 8 + \Delta_{\text{IP}}(IP) =

6. This provides a first interval of values for Ma in \textsc{sol(tgt)}:

RI = [5, 6].

The determination of RI is mainly based on the value of I^! which

can be difficult to assess exactly in practice. If now it is assumed that the

sign of I^! -whether DL < 0 or DL > 0- is known with certainty,

the signs of \Delta_{\text{IP}}(Ma) and \Delta_{\text{IP}}(Ma) are also known with certainty,

though their values are not. In the example, these two signs are nega-

tive, involving that the variation of Ma from \textsc{sol(arce)} to \textsc{sol(tgt)}

is negative. Therefore, the value of Ma in \textsc{sol(tgt)} is lower than the

value of Ma in \textsc{sol(arce)}. Thus, according to this reasoning, the

interval of values of Ma in \textsc{sol(tgt)} that has to be chosen is

EI = [0, 8]. Indeed, 8 is the value associated with Ma in \textsc{sol(arce)}

and 0 is the lowest possible value. Finally, Ma = [0, 5, 6, 8] is the

adapted descriptor D_{t}.

3rd example. D_{a} = (Scr = 6) is an example of descriptor of

\textsc{sol(arce)} depending on two descriptors of \textsc{arce}, namely

(Pgm \in [5, 8]) with DL_{\text{Pgm}} = +1 and (ETH \in [8, 10]) with

DL_{\text{ETH}} = -1. If D_{a} depends only on (Pgm \in [5, 8]) then the low

variation of Scr would be

\[
\Delta_{\text{IP}}(Scr) = DL_{\text{Pgm}} \cdot \Delta_{\text{IP}}(Pgm) = (+1) \cdot (+1) = +1
\]
If $D_s$ depended only on $(ETH \in [8, 10])$ then the low variation of Scr would be
\[
\Delta_{low, ETH}(Scr) = DL_{ETH} \cdot \Delta_{low}(ETH) = (-1) \cdot (+2) = -2
\]

The variation $\Delta_{low}(Scr)$ is computed by the sum of the above variations:
\[
\Delta_{low}(Scr) = \Delta_{low, p_{_0}}(Scr) + \Delta_{low, ETH}(Scr) = -1
\]

The variation $\Delta_{up}(Scr)$ is computed in a similar way:
\[
\Delta_{up}(Scr) = \Delta_{up, p_{_0}}(Scr) + \Delta_{up, ETH}(Scr) = (+1) \cdot (-1) + (-1) \cdot (+0) = -1
\]

Since $D_s = (Scr = 6)$ is the descriptor to be adapted, the above variations give the interval $RI = [5, 5]$. $EI$ is computed thanks to the signs of the four values of variation $\Delta_{low, p_{_0}}(Scr)$, $\Delta_{low, ETH}(Scr)$, $\Delta_{up, p_{_0}}(Scr)$ and $\Delta_{up, ETH}(Scr)$. 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 $Sol(Scr)$ would have given an interval $EI = [6, 10]$ (respectively, $EI = [0, 6]$). This is not the case, so the way this value changes is undefined and thus, $EI = [0, 10]$. Finally, $Scr = [0, [5, 5], 10]$ is the adapted descriptor.

The algorithm for adaptation of a single descriptor of $Sol(Scr)$, generalised from these three examples, is the following:

<table>
<thead>
<tr>
<th>adapt descriptor</th>
<th>(adaptation of one descriptor)</th>
</tr>
</thead>
</table>

**Input:**
- $D_s = (a = V) \in Sol(Scr)$ (a: attribute, V: value);
- $Dep$: a set of dependencies relating the descriptors of $D_s$ to the descriptor $D_I$;
- $mat(Scr, tgt)$, a matching between $Scr$ and $tgt$.

**Output:** a solution descriptor $D_I = (a = (RI, EI))$.

**begin (algorithm)**

- if $Dep = \emptyset$ /* no dependency, as in $1^o$ example */
  then return $(a = ([V, V], [0, 10]))$: /*$RI = [V, V]$ and $EI = [0, 10]$ */
- $\Delta_{low}(D_s) = 0$
- $\Delta_{up}(D_s) = 0$
- $signs = \emptyset$ /* The set of signs is initially empty */
  for each $(d_x, D_s, DL) \in Dep$
    Let $d_x$ and $ML = (\Delta_{low}(d_x), \Delta_{up}(d_x))$ such that $(d_x, D_s, ML) \in mat(Scr, tgt)$
    $\Delta_{low}(D_s) = \Delta_{low}(D_s) + DL \cdot \Delta_{low}(d_x)$
    $\Delta_{up}(D_s) = \Delta_{up}(D_s) + DL \cdot \Delta_{up}(d_x)$
    Compute the signs of $DL \cdot \Delta_{low}(d_x)$ and $DL \cdot \Delta_{up}(d_x)$
    $(-, +$ or $0$ for negative, positive or null values)
    and add them to the set $signs$.
  end (for each)
- $RI = \min\{r_1, r_2\}$, $\max\{r_1, r_2\}$
- if $signs = \{0\}$ /* all the signs are null */
  then return $(a = ([V, V]));$
- if $signs = \{+\}$ or $signs = \{0, +\}$ /* all the signs are positive */
  then $RI \leftarrow [V, 10]$
- if $signs = \{-\}$ or $signs = \{-, 0\}$ /* all the signs are negative */
  then $RI \leftarrow [0, V]$

**The main algorithm of adaptation is the following:**

**adaptation** (of a whole case)

**Input:**
- $((src, Sol(src))$, a source case;
- $Dep^{\text{src}}(src)$, the set of dependencies between $src$ and $Sol(src)$, i.e., a set of triples $(d_x, D_s, DL)$;
- $tgt$, a target problem;
- $mat(src, tgt)$, a matching between src and tgt.

**Output:** a solution $Sol(tgt)$ of tgt.

**begin (algorithm)**

$Sol(tgt) \leftarrow \emptyset$

for each $D_s \in Sol(src)$

Let $Dep$ be the set of the dependencies between descriptors of src and $D_s$:

$Dep \leftarrow \{d_s, D_s, DL \in Dep^{\text{src}}(src) | D = D_s\}$

$D_I \leftarrow adapt\_descriptor(D_s, Dep, mat(src, tgt))$

$Sol(tgt) \leftarrow Sol(tgt) \cup \{D_I\}$

end (for each)

return $Sol(tgt)$

end (algorithm)

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

<table>
<thead>
<tr>
<th>Sol(tgt)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Ma = [0, [5, 6], 8]$</td>
</tr>
<tr>
<td>$CPU = [0, [6, 7], 10]$</td>
</tr>
<tr>
<td>$AP = [6, [7, 7], 10]$</td>
</tr>
<tr>
<td>$CD = [4, [5, 5], 10]$</td>
</tr>
<tr>
<td>$Col = [0, [3, 3], 10]$</td>
</tr>
<tr>
<td>$Scr = [0, [5, 5], 10]$</td>
</tr>
<tr>
<td>$SC = [0, [2, 4], 10]$</td>
</tr>
<tr>
<td>$Ptr = [5, [5, 5], 5]$</td>
</tr>
</tbody>
</table>

The choice of a precise value associated with each attribute of $Sol(tgt)$ 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 $Pr \in [2000, 2300]$ specified in $tgt$, the price being calculated by the sum of the prices of each component. The local constraints are given by the intervals $EI$. For example, the value of the masterboard must be chosen in $EI = [0, 6]$. The interval $RI$ gives some more accurate suggestions for the choice of the value: since $RI = [5, 6]$ 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, $(SC = 0) \in Sol(src)$ can be substituted by $(SC = 3) \in Sol(tgt)$. Since $SC = 0$ means “no sound card”, the substitution of 0 by 3 is interpreted, in this context, by the addition of a sound card.
A classical approach of retrieval [9] consists in choosing a source case minimizing the differences between the source and target problems, i.e. the $|\Delta_{IP}(d_i)|$ and the $|\Delta_{IP}(d_j)|$ 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 $|\Delta_{IP}(D_s)|$ and the $|\Delta_{IP}(D_t)|$. The variations of the solution descriptors are linked with the variations of the problem descriptors by the dependencies. For example, minimising $|\Delta_{IP}(Ma)|$ consists in minimising $|\Delta_{IP}(PL) - \Delta_{IP}(IP)|$.

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