Knowledge-Based Systems 24 (2011) 230–247

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

Adaptive case-based reasoning using retention and forgetting strategies

Maria Salamó [*], Maite López-Sánchez

Dept. de Matemática Aplicada i Anàlisi, Universitat de Barcelona, Gran Via de les Corts Catalanes, 585 – 08007 Barcelona, Spain

A R T I C L E I N F O

Article history:
Received 5 November 2008
Received in revised form 2 July 2010
Accepted 9 August 2010
Available online 21 August 2010

Keywords:
Case-based reasoning
Case base maintenance
Case retention
Case forgetting
Learning efficiency
Case goodness measure

A B S T R A C T

Case-based reasoning systems need to maintain their case base in order to avoid performance degradation. Degradation mainly results from memory swamping or exposure to harmful experiences and so, it becomes vital to keep a compact, competent case base. This paper proposes an adaptive case-based reasoning model that develops the case base during the reasoning cycle by adding and removing cases. The rationale behind this approach is that a case base should develop over time in the same way that a human being evolves her overall knowledge: by incorporating new useful experiences and forgetting invaluable ones. Accordingly, our adaptive case-based reasoning model evolves the case base by using a measure of "case goodness" in different retention and forgetting strategies. This paper presents empirical studies of how the combination of this new goodness measure and our adaptive model improves three different performance measures: classification accuracy, efficiency and case base size.

1. Introduction

Case-based reasoning (CBR) [1] is the process of solving new problems by retrieving the most relevant past problems from an existing knowledge base and adapting them to fit the new situations. Problems are referred as cases and the previously experienced ones are stored in the *case base* or memory. One of the main advantages of CBR is that it can deal with very complex forms of knowledge. Examples of its applicability are: computer aided diagnosis systems for cancer detection [2]; knowledge discovery frameworks for textual case-based reasoning [3]; web-based applications [4,5]; or visual spacial problem solvers [6].

Aamodt and Plaza in [7] described the classical CBR model, which defines the problem-solving cycle in four different phases: *Retrieval*, *Reuse*, *Revise* and *Retain*. Fig. 1 shows this CBR cycle, where a new case is solved by first *retrieving* the most similar case from the case base, then *reusing* this retrieved case to provide a solution, *revising* this solution, and finally, *retaining* the new experience by incorporating it into the case base.

As shown in Fig. 1, the case base participates throughout the CBR cycle, and so it is vital for the system's problem-solving efficiency. This is particularly the case when CBR is applied to large-scale and lasting applications, where compact case bases are required. For this reason, the role of memory and the maintenance of case bases have been of great interest in CBR. More specifically, *case base maintenance* is defined as the process of revising the contents or organization of a case base in order to facilitate future reasoning for a particular set of performance objectives [8].

The *utility problem* [9] occurs when the cost of maintaining and searching in a large case base outweighs the benefit of storing its knowledge. This problem results in a decrease of performance (i.e., a decrease in problem-solving efficiency). There are two main factors that affect the overall utility of a CBR system: "the swamping problem" and "the harmful experience". The former relates to the expense of searching large case bases for appropriate cases with which to solve the current problem. The latter states that some experience (irrelevant, incorrect or redundant cases) may degrade the system's performance [10]. Different strategies for coping with these problems have been applied in CBR, and fall into three main categories: *Case addition*, *Case deletion*, and *Indexing* policies. *Case addition* strategies aim to control retention in order to avoid excessive growth of the case base [11]. *Case deletion* eliminates cases that do not contribute positively to performance, thus limiting case base size [12]. Finally, *Case indexing* partitions the case base into groups of cases sharing similar features [13] with the aim of overcoming the effects of the utility problem by improving retrieval efficiency.

By its very nature, CBR reflects human usage of previously solved—and remembered—problems as a starting point for providing solutions to new problems. Cognitive Science was a major influence in early CBR research with the study of human story understanding [14], especially as it led to investigations of the role

* Corresponding author. Tel.: +34 934039372.
E-mail addresses: maria@maia.ub.es (M. Salamó), maite@maia.ub.es (M. López-Sánchez).

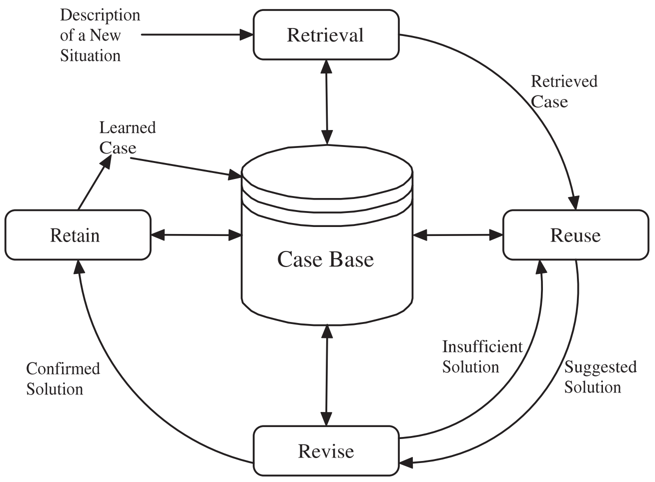

Fig. 1. Case-based reasoning cycle.

of memory in understanding [15]. Recently, in the Cognitive Science area, research has been devoted to analyzing how experience is transformed into memory. As an example, Paller and Wagner [16] focus on the ability to remember or forget one's past depending on neurocognitive processing which establishes durable episodic memories. We do not aim to model neurocognitive processing, but to approach CBR systems by using these human remembering and forgetting capabilities as a metaphor to maintain cases in the case base. In this manner, we propose that retention and forgetting strategies are basic processes to be combined in an adaptive case-based reasoning model. Following the neurocognition metaphor, in the same way that neural measures are used to predict which events will be subsequently remembered or forgotten, we propose a goodness measure for use by our retention and forgetting strategies. The ultimate goal of this approach is to provide a method to keep case bases compact, so that efficiency is not negatively affected in the long term.

The structure of the paper is as follows. Section 2 describes the adaptive case-based reasoning model we propose. Then, Section 3 details a series of retention and forgetting strategies used in our model. Section 4 analyzes and compares the effectiveness of the overall learning process when using different strategies. Section 5 provides a related work overview, and finally, Section 6 concludes.

2. Adaptive case-based reasoning (ACBR) model

The previous section described how CBR systems follow the classical 4-step reasoning cycle (see Fig. 1). In this section we propose a Case-based reasoning model that extends this previous cycle, as shown in Fig. 2. Throughout this section we present the model and its aim of developing the case base by incorporating new useful cases whilst forgetting worthless ones. Our model tries to do this by an iterative approach which promotes the cases that are well suited for solving new problems and reduces the influence of the cases that misclassify. Promotion and demotion are based on a *goodness measure* that is iteratively computed over each case. In this manner, we introduce retention and forgetting strategies and use this goodness measure to *review* cases in the case base.

Case review can be seen as an additional step in the CBR cycle. As a consequence, we extend the classical CBR cycle and redefine it as an Adaptive CBR cycle. Fig. 2 shows *Case review* inclusion between *Revise* and *Retain* phases. As we will see below, the review phase is divided into two main steps: a goodness update and a forgetting strategy. As their name indicates, retention strategies are considered to be part of the *Retain* phase, so we are in fact adapting our case base in the last two phases of our proposed Adaptive CBR cycle (that is, *Review* and *Retain* phases).

CBR is iterative by definition: the CBR cycle is applied once for each new problem case (i.e., a case that needs to be solved). Furthermore, including retention and forgetting strategies implies constant changes to the *case base*, and so iteration is intrinsic to our proposed reasoning method. Accordingly, the next subsection presents CBR knowledge components required and defines them to be dependent on the iteration steps.

2.1. Required definitions

Let $c = \langle c^P, c^S \rangle$ be a case, where c^P represents the problem description (P) and c^S its associated solution (S) (in classification tasks $c^S = class(c)$). An incoming new problem can thus be notated as $c_{new} = \langle c^P_{new}, \emptyset \rangle$, since it still lacks of solution.

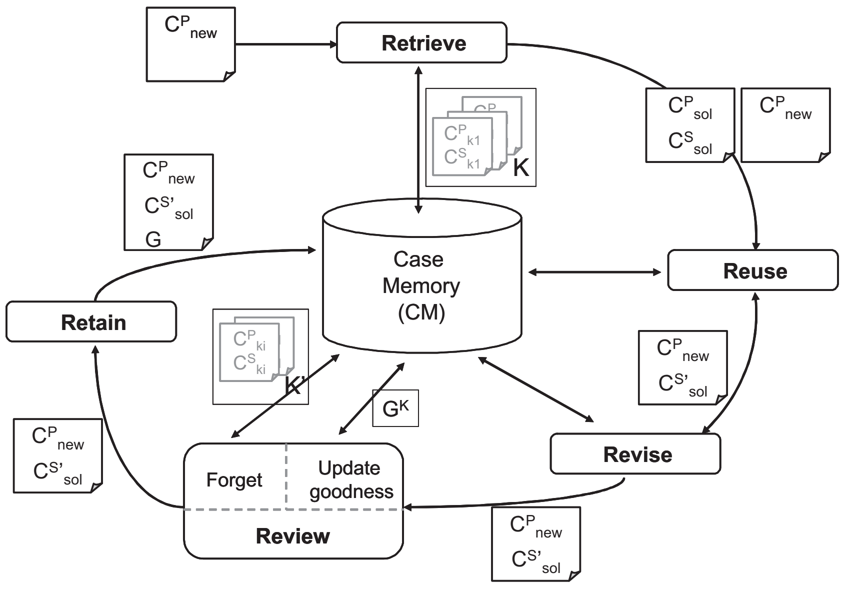

Fig. 2. Adaptive case-based reasoning cycle.

Let $TC = \left\{c_j^T : 1 \leqslant j \leqslant m\right\}$ be a set of testing cases c_j^T (T denotes testing). $|TC| = m$ and thus, our CBR cycle will be traversed along m iteration steps.

Let $CB_j = \{c_{i,j}: 1 \leqslant i \leqslant n_j\}$ be the case base in our CBR system at iteration step j. $|CB_j| = n_j$ because the set of cases in the case base changes for each iteration j.

Considering C as a set of cases, let $Goodness_j(c)$ be a function over each of those cases $c \in C$ such that $Goodness_j(c) : C \times \mathbb{N} \to \mathbb{R}$ defines the goodness value of case c at iteration j. Goodness values belong to the $[0\ldots1]$ range and depend on j because they are computed iteratively. To summarize, we have $0 \leqslant Goodness_j(c) \leqslant 1$. The next subsection provides the details of such computation (which corresponds to goodness value initialization and update methods).

Then, we also define $G_j^{CB} = \{Goodness_j(c_{i,j}) : c_{i,j} \in CB_j\}$ as the set of goodness values for all cases in CB_j. For short, we will refer to G_j^{CB} as goodness of CB_j.

Finally, we define a Case Memory $CM_j = \left(CB_j, G_j^{CB} G_0^{CB}\right)$ to be a tuple composed of case base CB_j and two sets containing its current and initial goodness values: goodness values at iteration j—i.e., G_j^{CB}—and initial[1] goodness values G_0^{CB}.

2.2. Reasoning method

As stated above, we propose a general reasoning model named Adaptive Case-Based Reasoning (ACBR). This subsection presents its pseudocode in Algorithm 1 and provides the corresponding explanation in relation to Fig. 2.

Algorithm 1: Adaptive case-based reasoning (ACBR)

```
   Input: CaseBase $CB$, TestingCases $TC$, $\mathbb{N}k$.
 1 begin
 2   $CB_0 \leftarrow CB$
 3   $G_0^{CB} \leftarrow \{0.5\}$   //*
 4   $\forall c_{i,0} \in CB_0$: $Goodness_0(c_{i,0}) = 0.5$
     $CM_0 \leftarrow \left(CB_0, G_0^{CB}, G_0^{CB}\right)$   //*
 5   initialize CM for $j \leftarrow 1$ to $|TC|$ do
 6     $c_{new} \leftarrow next(TC)$   //*
 7     consider a new test case // Retrieval
 8     $K \leftarrow retrieveKNN(CB_{j-1}, c_{new}, k)$   //*
 9     $K \subseteq CB_{j-1}$ $c_{sol} \leftarrow bestNN(K, c_{new})$
10     // Reuse: if required, adapt ($c_{new}$,$c_{sol}$),
11     //           otherwise: $c_{sol}$
12     // Review:
13     for $k \leftarrow 1$ to $|K|$ do
         // update goodness for cases $c_k \in K$:
14       $c_k \leftarrow next(K)$   //*
15       consider another case in K
         $r(c_k) \leftarrow \begin{cases} 1 & if \quad Class(c_k) = Class(c_{new}) \\ 0 & otherwise \end{cases}$.
16       $g \leftarrow Goodness_{j-1}(c_k)$   //*
17       $[r]$ $G_{j-1}^{CB}$ update
18       $Goodness_j(c_k) \leftarrow g + \alpha \cdot |r(c_k) - g|$
19     end
20     $CM_j \leftarrow$ Forgetting_Strategy($K$,$CM_{j-1}$)
21     // Retain:
22     $CM_j \leftarrow$ Retention_Strategy($c_{new}$, $c_{sol}$,$K$,$CM_j$)
23   end
24 end
```

As parameters, this algorithm receives the case base (CB) together with the set of testing cases (TC) and the number of cases to retrieve in the retrieval phase ($k > 0$). Firstly, the algorithm requires some initializations that consist in building the initial Case Memory (CM_0), which comes from assigning a default goodness value of 0.5 to all cases in the case base, CB. The rationale behind this initialization is that all cases are considered equally useful when problem-solving experience is lacking. Nevertheless, this initial goodness value could also be considered to be a case base maintenance metric obtained from the acquisition stage[2]. Furthermore, at this initialization stage, both current and initial values of goodness coincide, and code line 4 repeats these values to build our initial case memory tuple.

Fifth line in the algorithm is devoted to the general iteration step, which processes each case from the testing set. Therefore, for each iteration we consider the current testing case under consideration to be c_{new}, meaning that it corresponds to the next[3] new problem to solve (see c_{new} at top left corner in Fig. 2).

The *retrieval* phase is then performed in the following lines 8 and 9. First, we retrieve the k-Nearest Neighbors to c_{new} from the current case base, and group them in set K (see top center in Fig. 2). As we have previously mentioned, the number of cases to retrieve (k) is a parameter, and thus, we establish the cardinality of $K(|K| = k)$. Afterwards, from K we choose[4] the most similar case to c_{new}, so this case becomes the solution (we refer to it as c_{sol}, see top right area of Fig. 2).

Notice that, since we are dealing with a classification problem, our *reuse* mechanism is direct: we assume the class of c_{sol} to be the solution—i.e., the class—for c_{new} ($c_{new}^S = c_{sol}^S = class(c_{sol})$, see right side in Fig. 2). For a general CBR process, an adaptation process should then be invoked, just as line 10 shows. The *revise* phase is also omitted.

Next, our algorithm implements our *review* phase through two main steps: goodness update and oblivion (in Fig. 2 they appear as Update goodness and Forget). First, current goodness values in G^K are updated so that those cases in $K \subseteq CB_{j-1}$ are revised (lines 13 and 14 in the algorithm traverse all cases c_k in the retrieved set of cases K). Goodness update is inspired in Reinforcement Learning (RL) [18] and depends on both a modification- or learning-rate α, which averages values coming from different episodes; and R, defined as the set of reward values associated to all cases in K. More precisely, we have:

$$R = \{r(c_k), c_k \in K\} \tag{1}$$

being r a reward function r: $C \to \{1,0\}$ that assigns 1 to those cases c_k whose class coincides with the target class, and assigns 0 otherwise. In our algorithm, line 15 computes the reward value for current c_k, and the following lines 16 and 17 update its associated goodness value. This is done by applying our proposed goodness update formula, which modifies the goodness value of any case c in the following manner:

$$Goodness_j(c) = Goodness_{j-1}(c) + \alpha \cdot |r(c) - Goodness_{j-1}(c)| \tag{2}$$

Intuitively, this formula follows a reinforcement learning approach because a reward is generated when a case helps towards the correct classification, and thus, its goodness measure is incremented. Otherwise, a negative feedback is generated.

[1] In our algorithm, initial goodness values need to be considered throughout the process. For this reason they are also included in the Case Memory.

[2] Due to the lack of space we do not provide details on experiments performed with certain maintenance metrics; we just mention that initial values of goodness do not seem to play a really significant role in the overall performance [17].

[3] Notice that, although elements in a set do not follow any order, the algorithm invokes *next (TC)* function into the For loop to express that all cases in the incoming Testing Case set are treated once.

[4] K will actually be used in subsequent phases.

Table 1
Example of the evolution of goodness values.

Case memory CM_0			Iteration 1 $class(c_{new}) = 0$			Iteration 2 $class(c_{new}) = 1$		
CB	Class	G_0^{CB}	*K*	*R*	G_1^{CB}	*K*	*R*	G_2^{CB}
1	0	0.5	✓	1	0.6	-		0.6
2	0	0.5	✓	1	0.6	✓	0	0.48
3	1	0.5	✓	0	0.4	✓	1	0.52
4	2	0.5	–		0.5	–		0.5
5	0	0.5	–		0.5	–		0.5
6	2	0.5	–		0.5	–		0.5
7	1	0.5	–		0.5	✓	1	0.6
8	1	0.5	–		0.5	–		0.5

As the second part of our proposed *review* phase, line 19 invokes a forgetting strategy (i.e., oblivion). This strategy reviews all cases in K to consider if any of them should be deleted from the current Case Memory CM_{j-1}. As a consequence, it returns a subsequent version of case memory CM_j (Fig. 2 illustrates this by depicting K' as a variation of K). The next section provides further details.

Finally, our algorithm implements the *case retention* phase by invoking a retention strategy (see line 21 and left area in Fig. 2). This strategy decides whether the current problem case should be added to our case base. Again, the next section provides further details; here we just mention that it may consider current and retrieved cases (c_{new},c_{sol}),K, and Case Memory CM_j. If case retention is actually decided, the current CM_j will be enlarged by c_{new} together with its associated goodness values to define the next CM_j. In this manner, the retention strategy eventually returns a new version of case memory CM_j. Overall, our ACBR model improves or degrades the goodness of a case depending on its resolution accuracy. In fact, as subsequent sections will show, the most notable aspect of this adaptive case-based reasoning model is that the CBR system uses goodness values to improve the case base during its own problem-solving process. The next Section 2.3 illustrates goodness update and Section 3 which details how these goodness values are used by retention and forgetting strategies. Furthermore, it is worth noticing that these strategies make use of the retrieved k-Nearest Neighbors K to accelerate the maintenance process of the case base.

2.3. Goodness usage example

Here we present an example of how goodness is updated. Let us consider a case base of eight different cases. Table 1 shows its goodness evolution during the first two iterations of our adaptive case-based reasoning model.

The first main column in Table 1 contains the description of the initial case memory CM_0: each case identifier in CB_0, its corresponding class (0, 1 or 2) and its initial goodness value G_0^{CB} (set to 0.5 for all cases). The next two main columns detail subsequent iterations (1 and 2, each iteration corresponding to a new case to be solved). As a reference, both column headings include the actual class of the new considered case (i.e., $class(c_{new})$). Additionally, these columns show which cases have been retrieved (and thus added to K, where $|K| = 3$), their reward value (R is computed as shown in Eq. 1) and their resulting goodness value. The goodness update formula has been described in Eq. 2 (here, we consider $\alpha = 0.2$).

As we see, the first iteration considers a case whose class is 0, and the retrieval phase in the model selects as nearest neighbors the first three cases (that is, $K = \{1,2,3\}$). Comparing their classes, we see that cases 1 and 2 classify the new case correctly (obtaining a reward of 1) whereas case 3 misclassifies it (obtaining a reward of 0). As a consequence, cases 1 and 2 increase their goodness values up to 0.6 (e.g., $Goodness_1(c_1) \leftarrow 0.5 + 0.2 \cdot |1 - 0.5| = 0.6$) but for case 3, the goodness value falls to 0.4. The remaining cases do not modify their goodness values because they are not retrieved.

The next iteration is devoted to solving a new case whose class is 1. The most similar cases to c_{new} are cases 2, 3 and 7. The reward for these cases are 0, 1 and 1, respectively, thus changing the goodness value of the previous iteration to 0.48 for case 2, 0.52 for case 3 and 0.6 for case 7. Notice that so far, although both cases 2 and 3 have each classified and misclassified once, their goodness values are different. This is because they classified correctly at different iterations. In fact, the more recent is a correct classification, the higher the value of goodness we obtain.

3. Retention and forgetting strategies

Our ACBR model is said to be adaptive because it evolves the case base over time. This is done taking into account the generational experience (i.e., history of problem-solving episodes) of each case. We represent locally this generational experience by means of our *goodness* measure. Case-base evolution occurs by retaining and forgetting cases. This section shows different retention strategies that, apart from deciding if it is needed to retain a case, they also compute its associated goodness. Additionally, last part of the section introduces an oblivion strategy which uses goodness values to decide whether a case should be forgotten or not.

3.1. Degree of Disagreement retention strategy (DD)

In the *retrieval* phase, our system computes K as the set of candidate solutions to the test case under consideration c_{new}. The Degree of Disagreement (DD) retention strategy is based on a measure of dispersion—or disagreement—of this K set. It is inspired by the *active learning* policy [19], which, in turn, uses a variation of the *Query by Committee* (QbC) algorithm [20]. We present a version adapted to both the classification task and the usage of goodness values.

Our proposed algorithm uses the K set as a *virtual committee*. So, each member of the committee can be seen as a case in K, and the vote it generates as the class associated to this case. We compute the disagreement among the possible solutions (K set of retrieved cases), as follows:

$$d = \frac{\#remaining\_cases}{(\#classes - 1) \times \#majority\_cases} \tag{3}$$

Thus, Eq. 3 measures d as the degree of disagreement over the $|K| = k$ retrieved cases, where *#classes* is the number of different classes[5] in CB_j, *#majority_cases* is the number of cases with the most assigned class (i.e., the majority class in K) and *#remaining_cases* is the number of cases with classes other than the majority one. In this manner, $|K| = \#majority\_cases + \#remaining\_cases$. Thus, disagreement value d reaches value 1 when every case in K provides a different class as solution (that is, when each case in K yields to a different classification of c_{new}) and so, cases disagree the most, while it approximates to 0 when there is a clear majority. $\#classes - 1$ is introduced in order to normalize disagreement from 0 to 1.

[5] Notice, though, that [19] considers *#classes* to be the number of possible solutions. This is a general definition which, in our case, could correspond to all the classes in *CB* or just the classes appearing in *K*. We take the former option, so all classes are considered at each iteration step.

Algorithm 2 corresponds to the Degree of Disagreement retention strategy. As input, it receives the case under consideration (c_{new}) together with K and current version of Case Memory CM_j. Initially, it sets C to be the number of classes in CB_j,MC as the most assigned class[6] (i.e., majority class), and MK as a subset of K containing those cases having class MC. Then, in line 5, it computes the Degree of Disagreement d by applying the previous Eq. 3. If d exceeds a given threshold, the retention strategy considers that it will be useful to retain the new case because it may help in solving future disagreements. Otherwise, c_{new} will be discarded, since the system seems to have enough information to solve cases of this kind. If c_{new} is retained, then it will be added to current case base CB_j (see line 7). Afterwards, the method will associate to it a goodness value that corresponds to the highest[7] goodness value of the majority class (that is, the highest value associated to cases in MK). This goodness value is also considered to be the initial value of this case, and therefore, it is also added to G_0^{CB} (see code line number 10).

Finally, the method returns Case Memory CM_j updated with current versions of case base CB_j and Goodness G_j^{CB} together with its initial Goodness G_0^{CB}.

The rationale of this retention strategy is that disagreement may generate wrong solutions, and so the cases generating disagreement in the retrieval phase should be kept in order to contribute to future retrievals. Nevertheless, it is important to note that this is a non-supervised method: it does not consider whether c_{new} is correctly classified or not, but merely relies on the majority of classes in K. In this manner, it is dependent on the 'quality' of K, but in fact, this is also a general requirement of any CBR system: the system's performance always depends on the quality of its knowledge base.

Algorithm 2: Degree of disagreement retention strategy (DD)

```
    Input: TestingCase $c_{new}$, CaseSet $K$, CaseMemory $CM_j$
    Output: CaseMemory $CM_j$
 1  begin
 2      $C \leftarrow number\_of\_classes(CB_j)$;    //*
 3      $CB_j \in CM_j \quad MC \leftarrow majority\_class(K)$
 4      $MK \leftarrow \{c_k \in K | Class(c_k) = MC\}$
 5      $d = \frac{|K|-|MK|}{(C-1)\cdot|MK|}$    //*
 6      degree of disagreement
 7      if $d \geqslant threshold$ then
            // add $c_{new}$ and its goodness value to $CM_j$:
 8          $CB_j \leftarrow CB_j \cup \{c_{new}\}$
 9          $Goodness_j(c_{new}) \leftarrow \max_{c_k \in MK}(goodness_j(c_k))$
10              $G_j^{CB} \leftarrow G_j^{CB} \cup Goodness_j(c_{new})$
11              $G_0^{CB} \leftarrow G_0^{CB} \cup Goodness_j(c_{new})$
12      end
13      $CM_j \leftarrow (CB_j, G_j^{CB}, G_0^{CB})$
14  end
```

3.2. Detrimental retention strategy (DE)

Detrimental retention strategy (DE) uses the class of a given new case and its corresponding retrieved case in order to decide whether this new case should be retained. Since it requires the class of the problem case, it is a supervised strategy, and thus, it is more informed than the Degree of Disagreement retention strategy (DD) above. The main principle of this strategy is that a new case is retained if it has been misclassified and the majority class in K produces a misclassification.

Algorithm 3 details the process of this retention strategy. Initially, it analyzes whether the retrieved case c_{sol} used to solve the new case produces a misclassification: in other words, as line 2 shows, if the class of c_{sol} is different than the one for the testing case c_{new}. If this is the case, we again use the K retrieved cases for obtaining the majority class label MC. If the majority class also errs (see line 4), we consider that the system does not contain enough cases to produce a satisfactory retrieval, and so we say that it produces a *detrimental* retrieval. In order to compensate for this misclassification, c_{new} is added to current case base. Retention is performed by first adding c_{new} to $CB_j \in CM_j$ (see line 5), and then by considering its associated goodness value to be 0.5, which is added both to current goodness G_j^{CB} and initial G_0^{CB} (see code lines from 6 to 8). In the DD retention strategy above we used best goodness value from majority class for assigning the goodness value to c_{new}. Nevertheless, we cannot apply this policy into this *Detrimental* strategy, since there are no suitable cases in our set of K retrieved cases. This is why we adopt the simplest policy, and initialize the case goodness with an intermediate value of 0.5. This assignment does not represent a problem because, as shown later by the results, if the case has good efficiency, its goodness value will increase; if it does not its goodness value will be decreased by the adaptive case-based reasoning model.

If c_{sol} and c_{new} do have the same class, then we will assume that the system contains enough information to produce a good solution in the future and, therefore, the new case will be discarded. Additionally, if c_{new} has not been solved correctly but the majority class MC is able to solve it, the new case c_{new} is also discarded because the case base has not been able to solve the current case but it presents enough case diversity to solve similar cases in the future. So, the expectation is that future cases will be correctly classified. In any case, the method will eventually return Case Memory CM_j updated with current versions of case base CB_j, Goodness G_j^{CB}, and initial Goodness G_0^{CB}.

As we have seen, this detrimental retention strategy is a supervised diversity enhancing mechanism. It includes new cases if the system is not able to solve them correctly. It also assigns an initial goodness value to the added case, although is not necessary for the retention strategy itself: it will be later used by the forgetting strategy.

Algorithm 3: Detrimental retention strategy (DE)

```
    Input: TestingCase $c_{new}$, RetrievedCase $c_{sol}$, CaseSet $K$,
    CaseMemory $CM_j$
    Output: CaseMemory $CM_j$
 1  begin
 2      if $Class(c_{new}) \neq Class(c_{sol})$ then
 3          $MC \leftarrow majority\_class(K)$
 4          if $Class(c_{new}) \neq MC$ then
                //add $c_{new}$ and its goodness to $CM_j$:
 5              $CB_j \leftarrow CB_j \cup \{c_{new}\}$
 6              $Goodness_j(c_{new}) \leftarrow 0.5$
 7              $G_j^{CB} \leftarrow G_j^{CB} \cup Goodness_j(c_{new})$
 8              $G_0^{CB} \leftarrow G_0^{CB} \cup Goodness_j(c_{new})$
 9          end
10      end
11      $CM_j \leftarrow \left(CB_j, G_j^{CB}, G_0^{CB}\right)$
12  end
```

[6] As an implementation detail, note that, for all retention strategies that consider a majority class, we will solve ties by just considering the first one that has been found. So, it is the most assigned and most similar class since K is ordered according to similarity.

[7] It could be the highest value or the intermediate value 0.5. However, we choose the highest value because it allows us to assign a value of a group that has little diversity.

3.3. Minimal Goodness retention strategy (MG)

The retention strategies above use *goodness* as a way to transmit generational experience to the cases added to the case base. However, this experience is not utilized by the retention strategies themselves. For this reason, here we propose an additional retention strategy that focuses its behavior on the *goodness* of the set of retrieved cases.

Minimal Goodness (MG) retention strategy is an unsupervised strategy that decides whether a case should be learned—i.e., retained—or not without considering the solution (class) of the new case. Instead, this retention strategy uses the goodness associated to the cases in the majority class of the set *K* of retrieved cases. We name this strategy *minimal goodness* meaning that a minimum goodness value is required. Thus, this strategy will be useful for testing the usefulness of goodness.

Algorithm 4 shows the process of this retention strategy. As before, we use the *K* set of retrieved cases to compute the majority class label *MC* and the subset $MK \subseteq K$ of cases with this majority class (see code lines 2 and 3). Second, we consider *g* to be the maximum goodness value in the majority class (see code line 4). As code line 9 shows, if this goodness *g* is lower than a *threshold*—defined using Eq. 4 explained below—then the new case c_{new} will be stored in the case base. Otherwise, the policy considers that the majority class has a case *c* whose goodness subsumes that of c_{new}, and so, no addition is done. If required, this case would be computed as follows:

$$c = arg \max_{c_k \in MK}(goodness_j(c_k))$$

As Eq. 4 shows, the threshold that controls whether a new case is subsumed is computed by averaging *MaxGoodness* and *MinGoodness* values, which correspond to the maximum goodness and minimum goodness present in the current case base CB_j for those cases whose class coincides with the majority class. Therefore, in order to compute these values, we first need to obtain G_j^{MC} as the set of goodness values for all cases in current CB_j with class equal to *MC*:

$$G_j^{MC} = \{Goodness_j(c) : \; c \in CB_j, \; Class(c) = MC\}$$

MaxGoodness and *MinGoodness* values (g_{max} and g_{min} in the algorithm) are then computed in code lines 5 to 8 in our algorithm. This threshold lets the system increase the diversity in the groups that have cases with low goodness values and restricts the growth of the groups that are well adapted and have high goodness values.

$$threshold = \frac{MaxGoodness + MinGoodness}{2} \quad (4)$$

If the previous goodness value *g* is below this threshold, this retention strategy will decide to store the new case c_{new}, and so it requires its corresponding goodness value. As subsequent code lines show, our algorithm assigns *g* to be the associated goodness values (both current and initial) of c_{new}. In fact, *g* comes from previous execution cycles of case *c*—i.e., the one with maximum goodness in *MK*, the majority class set. Finally, current versions of case base CB_j and its associated goodness values G_j^{CB} and G_0^{CB} are updated to include c_{new} and its goodness values. In any case (whether c_{new} is added or not), our retention strategy ends at code line 15 by returning current case memory CM_j.

Algorithm 4: Minimal goodness retention strategy (MG)

```
   Input: TestingCase $c_{new}$, CaseSet $K$, CaseMemory $CM_j$
   Output: CaseMemory $CM_j$
1  begin
2    $MC \leftarrow majority\_class(K)$
3    $MK \leftarrow \{c_k \in K | Class(c_k) = MC\}$
4    $g \leftarrow \max_{c_k \in MK}(goodness_j(c_k))$
5    $G_j^{MC} \leftarrow \{Goodness_j(c) : \; c \in CB_j, Class(c) = MC\}$
6    $g_{max} \leftarrow max\left(G_j^{MC}\right)$
7    $g_{min} \leftarrow min\left(G_j^{MC}\right)$
8    $threshold \leftarrow \frac{g_{max}+g_{min}}{2}$
9    if ($g \leqslant threshold$) then
           //add $c_{new}$ and its goodness to $CM_j$:
10         $CB_j \leftarrow CB_j \cup \{c_{new}\}$
11         $Goodness_j(c_{new}) \leftarrow g$
12         $G_j^{CB} \leftarrow G_j^{CB} \cup Goodness_j(c_{new})$
13         $G_0^{CB} \leftarrow G_0^{CB} \cup Goodness_j(c_{new})$
14     end
15     $CM_j \leftarrow \left(CB_j, G_j^{CB}, G_0^{CB}\right)$
16 end
```

Like DD, MG is an unsupervised strategy (see Algorithm 2). The main difference between them is the condition for retention. DD is based on disagreement between all the possible solutions (set of *K* retrieved cases), as detailed in Eq. 3, while MG is based on a goodness boundary. MG decides to retain a case when the goodness of the majority class is lower than the average goodness of the case base (this is the boundary or threshold), thus denoting that the majority class needs to increase diversity.

3.4. Learning based on misclassification retention strategy (LE)

The Learning based on misclassification (LE) is the simplest strategy we propose for introducing generational experience in a CBR system[8]. Its basic rule is: *If a case is misclassified using the case base, it is added to it* (i.e., it is learned). Checking misclassification requires the strategy to be supervised and so this rule is applied to avoid further misclassification. As with all retention strategies, once decided whether a new case c_{new} needs to be stored, this strategy needs to assign a *goodness* value to it. This goodness value will be taken from the case most closely related to c_{new}.

Initially, Algorithm 5 receives both the case under consideration c_{new} and the retrieved solution case c_{sol} together with *K* and CM_j. As we have already mentioned, this is a supervised strategy, and so, first step (code line 2) tests whether c_{new} has been correctly classified, or in other words, whether its class corresponds to the retrieved one (i.e., c_{sol}'s class). If it has been correctly classified, nothing more needs to be done, because the strategy considers *CB* to have enough information. Otherwise—if the case has been misclassified—it checks if there are cases in *K* with the same class as c_{new}. We call them candidate cases (see code line 4). If this is not the case (line 5), then it searches for candidates in the whole case base CB_j (line 6). Candidate cases are stored in set *CC*. If there are any candidate cases after these previous searches (line 8), then we choose the one that is most similar to the new one, c_{new}, so as to assign its goodness to c_{new} before adding it to our current case base (see code lines 10

[8] In fact, we have worked on this problem elsewhere [21].

to 13). The most similar is chosen to increase diversity in an area of the problem-solving space that produces misconceptions to the CBR.

Finally, this retention strategy ends by returning current version of case memory CM_j, which is composed of current case base CB_j, its associated goodness values G_j^{CB}, and initial goodness values G_0^{CB}.

Algorithm 5: Learning based on misclassification (LE)

```
Input: TestingCase $c_{new}$, RetrievedCase $c_{sol}$, CaseSet $K$, CaseMemory $CM_j$
Output: CaseMemory $CM_j$
1  begin
2      if $Class(c_{new}) \neq Class(c_{sol})$ then
3          $CC \leftarrow \emptyset$        //*
4          set of candidate cases
5          $CC \leftarrow \{c_k \in K | Class(c_k) = Class(c_{new})\}$
6          if $CC = \emptyset$ then
7              $CC \leftarrow \{c_{cb} \in CB_j | Class(c_{cb}) = Class(c_{new})\}$
8          end
9          if $CC \neq \emptyset$ then
10             $c \leftarrow arg\ \max_{c_c \in CC}(similarity(c_c, c_{new}))$
               //add $c_{new}$ and its goodness to $CM_j$:
11             $CB_j \leftarrow CB_j \cup \{c_{new}\}$
12             $Goodness_j(c_{new}) \leftarrow Goodness_j(c)$
13             $G_j^{CB} \leftarrow G_j^{CB} \cup Goodness_j(c_{new})$
14             $G_0^{CB} \leftarrow G_0^{CB} \cup Goodness_j(c_{new})$
15         end
16     end
17     $CM_j \leftarrow (CB_j, G_j^{CB}, G_0^{CB})$
18 end
```

In summary, Algorithm 5 enriches the case base with new cases when its knowledge is incomplete. In other words, only if a case is misclassified using the case base, it is added to it with the aim of avoiding further misclassification. Later, Algorithm 5 proportionates generational experience—which we represent through *goodness* values—to the case that is being added. This is done by considering the most closely related experiences to the input case c_{new} we are processing. In this algorithm, relatedness is defined by nearest neighbors, and therefore, this strategy is similar in its foundations to the *Condensed Nearest Neighbor* rule [22].

3.5. Oblivion by goodness forgetting strategy (O)

Studies in Cognitive Science have shown that humans transform experiences into memories by remembering and forgetting their past. By its very nature, CBR reflects the human use of remembered problems and solutions for new problem-solving. However, it is also essential to consider forgetting, as well as remembering, as part of CBR process. This required relationship between remembering and forgetting was studied by Markovitch and Scott [23] who argued that remembering may be of negative value even though cases stored are correct and forgetting may lead to substantial improvement on performance.

ACBR approaches CBR to human remembering and forgetting capabilities. ACBR's ability to forget occurs at the *review* phase which considers cases to be deleted by applying a forgetting strategy. The strategy we propose applies a simple strategy to decide which cases should be forgotten (that is, removed from the case memory) based on goodness loss. We call it Oblivion by goodness (O) and it is shown in Algorithm 6.[9]

This policy analyzes all cases present in the set of retrieved cases K and deletes from the case base those cases whose goodness values have fallen below their initial values. Thus, the cases that produce misconception at the beginning or repeatedly during the problem-solving episodes are deleted. Only those cases that allow the system to classify correctly on some occasions are kept. Therefore, for example, a case will be deleted if it classifies a new case incorrectly the first time it is retrieved, because its goodness value will be lower than its initial one. In contrast, a case will have additional opportunities if it has classified correctly other cases before. This means that its goodness has already increased, and so its current goodness value may still decrease as long as it does not fall below its initial value.

Case oblivion in this forgetting strategy algorithm is performed over retrieved cases in K, and so, as code line 3 shows, a case $c \in K$ is removed from current case base CB_j if its current goodness value ($Goodness_j(c)$) is lower than its initial one ($Goodness_0(c)$). Actual case oblivion is performed in code lines 4 to 6. Accordingly, its corresponding goodness value is also deleted from both set G_j^{CB} of current goodness values and from set G_0^{CB} of initial ones. Algorithm 6 ends by returning these three sets as the new version of our case memory CM_j.

Algorithm 6: Oblivion by goodness forgetting strategy (O)

```
Input: CaseSet $K$, CaseMemory $CM_j$
Output: CaseMemory $CM_j$
1  begin
2      foreach $c \in K$ do
3          if $(Goodness_j(c) < Goodness_0(c))$ then
               // delete c from case base
4              $CB_j \leftarrow CB_j - \{c\}$
5              $G_j^{CB} \leftarrow G_j^{CB} - \{Goodness_j(c)\}$
6              $G_0^{CB} \leftarrow G_0^{CB} - \{Goodness_0(c)\}$
7          end
8      end
9      $CM_j \leftarrow (CB_j, G_j^{CB}, G_0^{CB})$
10 end
```

3.6. Computational complexity analysis

Along this section we have presented a number of retention and forgetting strategies whose most salient characteristic is its simplicity, in terms of their conceptualisation as well as in terms of their computational complexity. This subsection elaborates on their time complexity based on the algorithms previously introduced. In particular, in their computations, most proposed retention mechanisms traverse K, the set of retrieved cases, and thus, they have a computational complexity of the order of the number of its elements $k = |K|$. Therefore, we can state that DD, DE, MG and LE are $O(k)$ where k is fixed and small. As next section shows, constant k has been fixed to small values (3, 5, and 7 in our experiments, see Section 4.4).

Regarding the forgetting (i.e., oblivion) strategy, it also traverses K when choosing the cases to delete, and so it can be considered to be equivalent to the retention strategies in terms of time complexity—i.e., $O(k)$.

Finally, it is also worth mentioning the time complexity related to the update of goodness values in Algorithm 1, the one describing our adaptive case-based reasoning proposal. Goodness values of

[9] We have previously worked in a similar oblivion method in [21], where we use the concept of coverage instead of goodness.

Table 2
Detail of the data sets used in this article. #Cases = Number of cases, #Num. = Number of numeric attributes, #Nom = Number of nominal attributes, #Cla. = Number of classes, Dev.Cla. = Deviation of class distribution, Maj.Cla. = Percentage of cases belonging to the majority class, Min.Cla. = Percentage of cases belonging to the minority class, MV = Percentage of values with missing values $\left(\frac{\#missing}{\#cases \times \#attributes}\right)$.

	Domain	#Cases	#Num.	#Nom.	#Cla.	Dev.Cla. (%)	Maj.Cla. (%)	Min.Cla. (%)	MV (%)
	Adult	48,842	6	8	2	26.07	76.07	23.93	0.95
	Audiology	226	–	69	24	6.43	25.22	0.44	2.00
	Autos	205	15	10	6	10.25	32.68	1.46	1.15
*	Balance scale	625	4	–	3	18.03	46.08	7.84	–
*	Breast cancer Wisconsin	699	9	–	2	20.28	70.28	29.72	0.25
*	Bupa	345	6	–	2	7.97	57.97	42.03	–
*	cmc	1473	2	7	3	8.26	42.70	22.61	–
	Horse-Colic	368	7	15	2	13.04	63.04	36.96	23.80
*	Connect-4	67,557	–	42	3	23.79	65.83	9.55	–
	Credit-A	690	6	9	2	5.51	55.51	44.49	0.65
*	Glass	214	9	–	2	12.69	35.51	4.21	–
*	TAO-Grid	1,888	2	–	2	0.00	50.00	50.00	–
	Heart-C	303	6	7	5	4.46	54.46	45.54	0.17
	Heart-H	294	6	7	5	13.95	63.95	36.05	20.46
*	Heart-Statlog	270	13	–	2	5.56	55.56	44.44	–
	Hepatitis	155	6	13	2	29.35	79.35	20.65	6.01
	Hypothyroid	3772	7	22	4	38.89	92.29	0.05	5.54
*	Ionosphere	351	34	–	2	14.10	64.10	35.90	–
*	Iris	150	4	–	3	–	33.33	33.33	–
*	Kropt	28,056	–	6	18	5.21	16.23	0.10	–
*	Kr-vs-kp	3196	–	36	2	2.22	52.22	47.78	–
	Labor	57	8	8	2	14.91	64.91	35.09	55.48
*	Lymph	148	3	15	4	23.47	54.73	1.35	–
	Mushroom	8124	–	22	2	1.80	51.80	48.20	1.38
*	Mx	2048	–	11	2	0.00	50.00	50.00	–
*	Nursery	12,960	–	8	5	15.33	33.33	0.02	–
*	Pen-based	10,992	16	–	10	0.40	10.41	9.60	–
*	Pima-Diabetes	768	8	-	2	15.10	65.10	34.90	–
*	SatImage	6,435	36	-	6	6.19	23.82	9.73	–
*	Segment	2,310	19	–	7	0.00	14.29	14.29	-
	Sick	3772	7	22	2	43.88	93.88	6.12	5.54
*	Sonar	208	60	–	2	3.37	53.37	46.63	–
	Soybean	683	–	35	19	4.31	13.47	1.17	9.78
*	Splice	3190	–	60	3	13.12	51.88	24.04	–
*	Vehicle	946	18	–	4	0.89	25.77	23.52	–
	Vote	435	-	16	2	11.38	61.38	38.62	5.63
*	Vowel	990	10	3	11	0.00	9.09	9.09	–
*	Waveform	5,000	40	-	3	0.36	33.84	33.06	–
*	Wine	178	13	–	3	5.28	39.89	26.97	–
*	Zoo	101	1	16	7	11.82	40.59	3.96	–

retrieved (k) cases are updated by following Eq. 2, which only takes into account the current case (its previous goodness value and current reward). Therefore, its computational complexity is again $O(k)$.

Consequently, we can conclude that, in total, the time complexity that previous proposed methods add to classical CBR includes the time devoted to update the goodness values as well as the time for the forgetting and retention strategies: $O(k) + O(k) + O(k) = O(3 \cdot k) \approx O(k)$. Note that k is small (i.e., 3, 5 or 7) and constant. Thus, our cost is nearly negligible. Our experiments (see Section 4.4) show that ACBR model perform well with small values of k (i.e., 3, 5 or 7). Additionally, they also denote that ACBR efficiency is not significantly larger than the basic CBR.

4. Evaluation

In this article so far we have argued that problem-solving efficiency in a CBR system is degraded if the case base becomes over-populated or if it contains harmful knowledge. To cope with this problem, we have proposed an adaptive case-based reasoning model that adapts its case base by using retention and forgetting strategies. This section evaluates the performance of our ACBR model in a series of experiments. First, it describes the data sets used, their general design and the statistics used for comparison. Second, it presents different aspects of the evaluation of the performance, and finally, some lessons learned from the results obtained.

4.1. Data

Evaluation is performed by using a suite of 40 benchmark data sets which are briefly described in Table 2. All these data sets are obtained from the UCI machine learning repository[10] [24]. The selected data sets have been chosen to provide a varying number of cases, features and classes and differing proportions of nominal to numeric attributes. Additionally, some of them contain a significant percentage of missing values.

4.2. Design

We use BASTIAN [25], a CBR system for classification tasks, which has been setup with the same parameters for all versions of the model. The case memory, formally defined in Section 2.1, is a tuple composed of a case base CB_j and two sets corresponding to its current $\left(G_j^{CB}\right)$ and initial goodness $\left(G_0^{CB}\right)$ values. Initially, $G_j^{CB} = G_0^{CB}$ and cases $c_{i,0} \in CB_0$ are assigned a $Goodness_0(c_{i,0}) = 0.5$.

The goodness update formula (see line 17 in Algorithm 1) contains the learning rate α which is usually set up to value 0.1 or 0.2 in RL systems. If the states are updated quite often it is set to value

[10] Except for the TAO-Grid which is a synthetic data set generated by discretizing a Tao symbol (a 2D representation of a yin-yang circle).

Table 3
Summary of our proposed retention strategies. Columns detail if the strategy is supervised (Sup?), the decision criterion for retaining a new case and its goodness assignment. Notation: *K*: set of retrieved cases, *MC*: majority class label, *MK*: subset of cases in *K* having class *MC*, and *CC*: set of candidate cases in *K* or *CB*.

	Sup?	Decision criterion	Goodness
DD	No	if too much variation in classes of K	$g = max(goodness(MK))$
DE	Yes	if misclassification and $class(c_{new}) \neq$ MC	$g = 0.5$
MG	No	if $max(goodness(MK)) \leqslant goodness(CB^{MC})$	$g = max(goodness(MK))$
LE	Yes	if misclassification and $\exists CC$ s.t. $class(CC) = class(c_{new})$	$g = goodness(max(similarity(CC, c_{new})))$

0.1, otherwise to 0.2. The selection of cases in a CBR cycle may not be repeated really often, so we set up this learning rate to be $\alpha = 0.2$ in order to accelerate *goodness* differences in fewer iterations.

In order to evaluate our ACBR model, we analyze the different retention and forgetting options we have presented so far. Therefore the question that naturally arises is how oblivion affects performance when combined with retention. To answer this question, first we set up ACBR model configurations that consider retention alone, and second we compare their performance with configurations that also include oblivion. We differentiate these configurations by labeling them with the acronyms presented in Section 3 above. We thus obtain eight different versions of our ACBR model: **DD**, **DE**, **MG**, **LE**, **DD-O**, **DE-O**, **MG-O** and **LE-O** (note that last four include oblivion).

For ease of reference in subsequent comparisons, Table 3 summarizes our proposed retention strategies. Notice though that DD requires a threshold parameter (see line 6 in Algorithm 2) which has been set to 0.3 in our experiments.

Before comparing our different model configurations with each other, we should first compare their performance with respect to standard CBR methods. So now the question is whether our overall model improves standard ones. We have chosen two models that consider opposite retention strategies: Never Retain (NR), which does not change the case base, and Different Similarity (DS), which increases its case base with new testing cases whenever they are different from the existing ones. Therefore, in terms of case base reduction, NR and DS represent lower and upper references.

Finally, every model is tested on each data set using a 10-fold cross validation. An *s*-fold cross validation divides a data set into *s* equal-size subsets. Each subset is used in turn as a test set with the remaining $(s - 1)$ data sets used for training.

4.3. Statistics

This subsection describes the variety of statistics used to evaluate and compare the performance of each algorithm in our experiments.

First of all, we compute the **mean rank** of each configuration considering all the experiments (10 algorithms and 40 data sets). We rank alternative algorithms, for each data set, following the practice of Friedman [26] [27]: the one that attains the best performance is ranked 1, the second best ranked 2, so on and so forth. Then, a method's mean rank is obtained by averaging its ranks across all data sets. Compared with mean performance values, the mean rank can reduce the susceptibility to outliers which, for instance, allows a classifier's excellent performance on one data set to compensate for its overall bad performance [28]. Second, we apply the Friedman and Nemenyi tests to analyze whether the difference between algorithms is statistically significant.

The **Friedman test**, recommended by Demšar [28], is effective for comparing multiple algorithms across multiple data sets. It compares mean ranks of algorithms to decide whether to reject the null hypothesis, which states that all the methods are equivalent and so their ranks should be equal. When we apply the Friedman test in our experimental setup with 10 algorithms and 40 data sets, F_F is distributed according to the F distribution with $(10 - 1) = 9$ and $(10 - 1) \times (40 - 1) = 351$ degrees of freedom. The critical value of $F(9, 351) = 1.91$ at the 0.05 critical level. Therefore, if the value of F_F—calculated from the mean rank of each algorithm—is higher than 1.91, we can reject the null hypothesis.

Once we have checked that Friedman test rejects its null hypothesis, we can apply a post hoc test, the **Nemenyi test** [29], to check whether there are statistically significant differences between algorithms. Two algorithms are significantly different if the corresponding average ranks differ by at least the critical difference value, $CD = q_\alpha \sqrt{\frac{k(k+1)}{6N}}$, where the critical value q_α is based on the studentized range statistic divided by $\sqrt{2}$. In our case, when comparing ten algorithms with a critical value $\alpha = 0.05$, $q_{0.05} = 3.164$ for a two-tailed Nemenyi test. Substituting, we obtain a critical difference value $CD = 2.14$. Thus, for any two pairs of algorithms whose rank difference is higher than 2.14, we can infer—with a confidence of 95%—that there exists a significant difference between them.

4.4. Performance evaluation

In this subsection we try to answer the questions that have arisen regarding our design: (1) is our overall model able to improve on standard ones? and (2) how does oblivion affect performance when combined with retention?

Performance can be measured in a variety of ways. In our experiments we consider three main performance measures: *accuracy*, *efficiency* and *case base size*. Regarding case base size, it can be decomposed in *percentage of retention* and *percentage of oblivion*. Classification *accuracy* (i.e., the average of correctly classified cases) is the most common performance measure used in CBR. However, we consider that *efficiency* (i.e., average problem-solving time) should also be taken into account in an adaptive case-based reasoning model.

Furthermore, when evaluating performance, it is important to study the influence of the number of cases that our ACBR model retrieves ($|K|$). For this reason, in this section we present the results for three different values of this parameter: $k = 3$, 5 and 7.

4.4.1. Classification accuracy results

As explained before, to compare the classification accuracy of each configuration, first of all we rank them. Fig. 3 illustrates their mean rank of accuracy, and indicates that of the configurations, DE-O and MG-O are the most accurate, and NR is the least. It also reveals that the versions that only retain are less accurate than the ones that retain and forget cases (e.g., DD-O, DE-O, MG-O and LE-O). In fact this effect can be seen for all k values, despite some minor differences (for example, DD shows that it performs better if it increases k values).

Next, we evaluate whether there are significant differences between configurations by applying Friedman and Nemenyi tests. We start with the Friedman test to determine whether we can reject the null hypothesis. Computing F_F we obtain 4.47, 3.03 and 2.95 for $k = 3$, 5 and 7, respectively. Since all of them are higher than

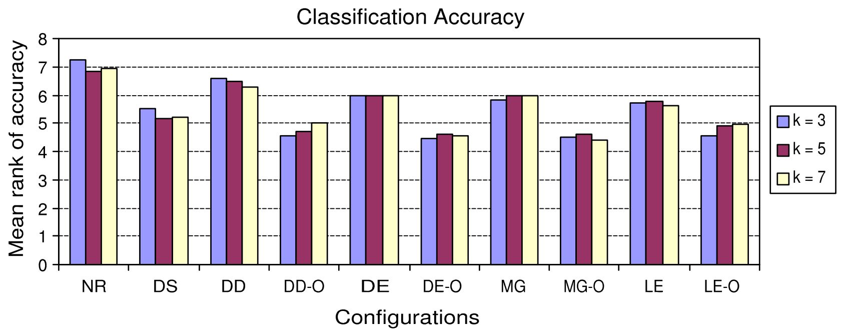

Fig. 3. Comparison of configurations in terms of mean rank of accuracy (NR and DS are standard CBR configurations used as lower and upper references of performance).

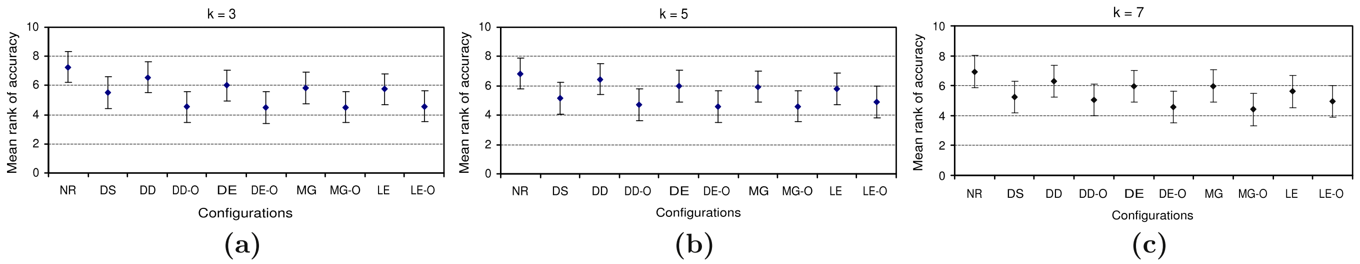

Fig. 4. Application of the Nemenyi test to alternative configurations' mean ranks of accuracy. Each figure represents a different test on parameter k where (a) corresponds to $k = 3$, (b) $k = 5$ and (c) $k = 7$.

1.91, we can reject the null hypothesis and infer that there is a significant difference between configurations.

Second, we perform the Nemenyi test, whose results are illustrated in Fig. 4, to find out exactly which configurations are significantly different. In the graph, diamonds represent the mean ranks of each strategy. Vertical lines across diamonds indicate the 'critical difference'. The performance of two configurations is significantly different if their corresponding mean ranks differ by at least the critical difference. That is, two methods are significantly different if their vertical lines are not overlapping. For instance, Fig. 4(a) reveals that MG-O is ranked best with a mean rank value of 4.5 and it is significantly better than NR (mean rank value 7.26). We cannot say the same with regard to the rest of methods, though.

In response to the first question, Fig. 3 clearly shows that our overall model obtains better accuracy than standard CBR configurations. Moreover, Fig. 4 illustrates that the ACBR model significantly improves accuracy compared with NR and maintains it compared with DS.

In order to answer the second question, we analyze in depth the configurations that consider retention alone and those that also include oblivion. Fig. 4 shows that DS, DD, DE, MG and LE are more effective in accuracy than NR. This is expected, since a CBR uses the case base to solve new problems and the acquisition of new cases improves its problem-solving capability. Therefore, a CBR needs to learn new cases. On the other hand, DS is slightly more effective in accuracy than DD, DE, MG and LE although, as we will see, in the case base analysis it is the one that stores the most cases. The DD, DE, MG and LE versions of our model are all quite similar in all configurations of parameter k although they provide slightly better results for the largest value of k. For example, DE presents a mean rank value of 6, 5.98 and 5.96 for $k = 3$, 5 and 7, respectively. Among them, the best turns out to be LE with a mean rank value of 5.61 for $k = 7$.

It can also be seen in Fig. 4 that DD-O, DE-O, MG-O and LE-O versions significantly improve accuracy compared with NR. Notice that the mean rank difference between NR and our overall model is sometimes reduced when parameter k is increased. As an example, in Fig. 4(c) we observe that DD-O and LE-O are not significantly better than NR, while the only significant models are DE-O and MG-O. This is mostly due to the oblivion strategy which increases the number of cases to be considered for deletion when K is increased (as we will see in Section 4.4.3, this reduces the case base). The greater the reduction, the smaller the case base, and as a result the accuracy of the CBR may be reduced.

It is clear from the results in Fig. 4 that all versions of our model improve NR to some degree. However, the only ones that improve significantly in terms of accuracy are those that retain and forget cases at the same time, such as DD-O, DE-O, MG-O and LE-O in Fig. 4(a). Apart from retaining cases, therefore, a CBR can lead to substantial improvement in accuracy if it also forgets cases. This conclusion reaffirms Markovitch and Scott[23]'s comment that retention and forgetting are complementary processes which construct or maintain useful representations of experience. So, in response to second question, we can conclude that oblivion has a positive effect on accuracy.

4.4.2. Efficiency results

In this subsection, we analyze the two questions posed at the beginning of our evaluation in terms of efficiency (i.e., understanding efficiency as average problem-solving time).

Fig. 5 illustrates each strategy's mean rank of efficiency. The Friedman test indicates that there are significant differences between configurations on efficiency and Fig. 6 shows the results of the Nemenyi test which reveal what these differences are.

Our expectation is that our configurations will present an efficiency in between NR and DS. This hypothesis is founded on two facts. First, NR does not devote time to maintaining the case base or to searching for additional cases (those added during the reasoning), so we expect it will be the most efficient. Second, DS is adding new cases all the time, and therefore will be the least efficient.

Fig. 5 shows that our previous hypothesis has been confirmed; all our configurations are in between NR and DS. NR is the most efficient with a mean rank value of 2.34 whilst DS is the least

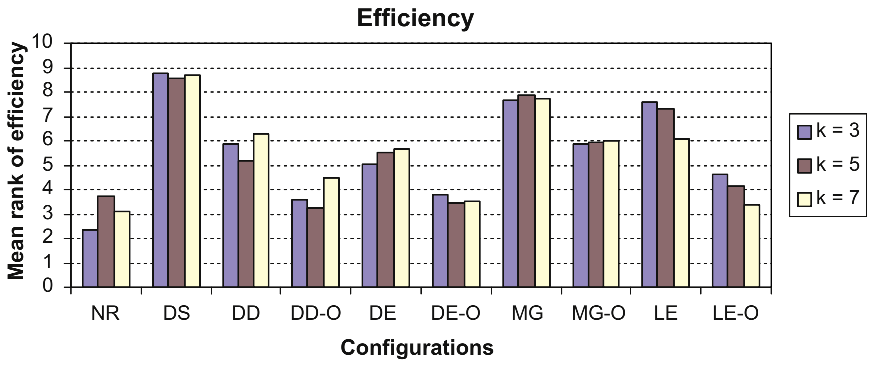

Fig. 5. Comparison of ACBR model's configurations in terms of mean rank of efficiency (NR and DS are standard CBR configurations used as lower and upper references of performance).

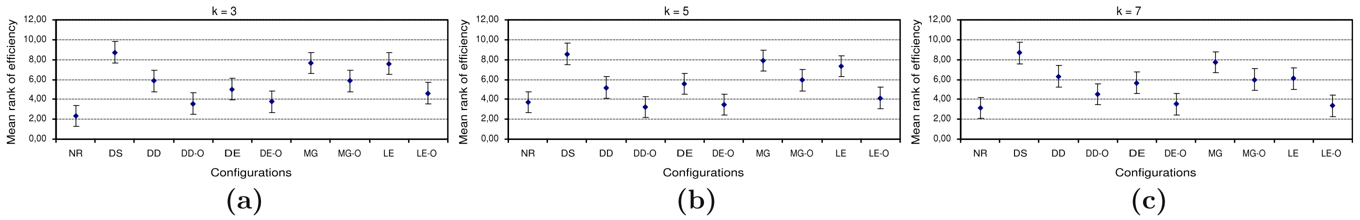

Fig. 6. Application of the Nemenyi test to alternative configurations' mean ranks of efficiency. Each figure represents a different test on parameter k where (a) corresponds to $k = 3$, (b) $k = 5$ and (c) $k = 7$.

efficient with a mean rank value of 8.73 for $k = 3$. Considering all the proposals, DD-O and DE-O are the best at reducing problem-solving times, with mean rank values of 3.58 and 3.76 respectively. Neither of them are significantly worse than NR, so we can state that they present a good efficiency. This is not surprising since, as well as adding new cases, DD-O and DE-O also forget the cases that are not considered useful for the problem-solver; so they work with a smaller case base and are faster classifiers.

The answer in terms of efficiency to first question is that our overall model is much efficient than DS and approaches the efficiency obtained by NR in some configurations. These results are satisfactory since our model performs retention and oblivion while NR does not spend time on these processes.

To answer the second question, we look carefully at the results of each configuration. Among DD, DE, MG and LE (see Fig. 6(b)) DD and DE configurations show a significant improvement on efficiency with respect to DS, and come close to the level of NR. Although MG and LE are less efficient than DD and DE, they are more efficient than DS. The differences in efficiency can be explained by the fact that DD and DE contain a simple process which only involves k cases, while MG considers the entire case base (CB_j) and LE may do so if it requires information (criteria are summarized in Table 3). So we deduce that efficiency depends on the number of cases treated by the model. The next section shows the case base size obtained by each configuration.

Comparing DD-O, DE-O, MG-O and LE-O, it can be seen in Fig. 6(c) that the most efficient configuration corresponds to LE-O (with a mean rank value of 3.36) whilst NR presents a mean rank value of 3.13. Surprisingly, NR is not significantly more efficient than LE-O. DD-O and DE-O also present notable results in comparison to NR. Fig. 6 also shows that augmenting k parameter results in an increase of the rank difference between configurations (i.e., DE-O and LE-O) and DS. This phenomenon is clearly observed in Fig. 6(c), where all configurations except MG are significantly more efficient than DS. The reason for this is, as we will confirm in Section 4.4.3, that augmenting k implies an increase of forgotten

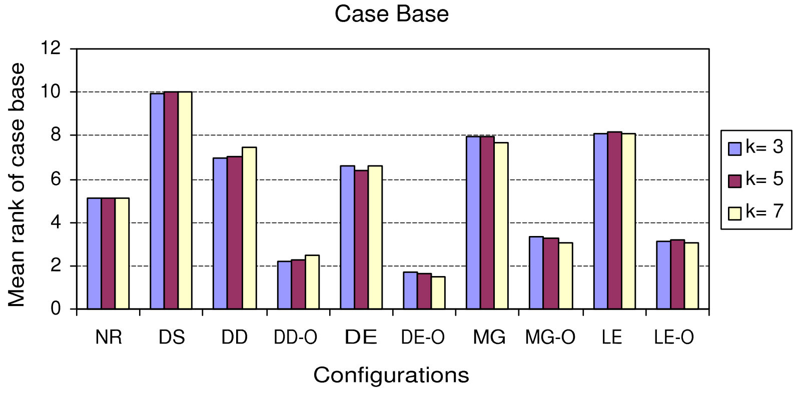

Fig. 7. Comparison of ACBR model's configurations in terms of mean rank of case base size (NR and DS are standard CBR configurations used as lower and upper references of performance).

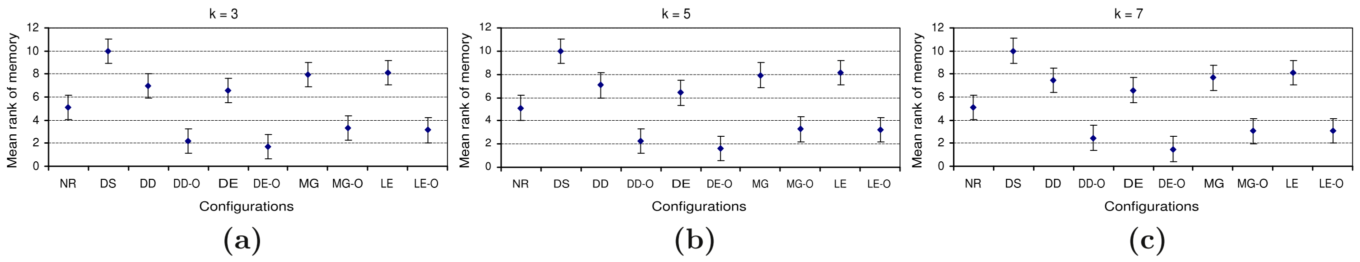

Fig. 8. Apply the Nemenyi test to alternative configurations' mean ranks of case base size. Each figure represents a different test on parameter k where (a) corresponds to $k = 3$, (b) $k = 5$ and (c) $k = 7$.

cases; efficiency is increased because there are fewer cases in the case base.

These results reflect that the configurations which include oblivion are more efficient than the ones which consider retention alone. So, in response to the second question, we can conclude that oblivion leads to an increase on efficiency.

4.4.3. Case base size results

With the aim of analyzing the dimension of the case base, Fig. 7 illustrates each configuration's mean rank for case base size. The Friedman test showed significant differences in case base size between configurations. The Nemenyi test (Fig. 8) shows what these differences are.

As shown in Fig. 7, the best configuration for reducing case base size is DE-O; as expected, the worst is DS. In general, the case base size of the DD, DE, MG and LE configurations is in between those of NR and DS. This is largely due to two facts: (1) NR never retains a case, so its case base will always be smaller than a configuration which adds cases; and (2) DS retains nearly all cases. DD, DE, MG and LE configurations just retain a reduced amount of test cases and, therefore, they confirm our general statement: all of them have larger case bases than NR but smaller than DS. Nevertheless, DD-O, DE-O, MG-O and LE-O case bases are smaller than those of DS and NR. This reduction in case base size is because few cases are retained and many are forgotten.

We conclude, in response to the first question, that the ACBR model controls case base size and in consequence it produces a more compact case base than standard CBR.

Fig. 8(c) illustrates that DS is significantly outperformed by all configurations. DD, DE, MG and LE case base size is in between DS and NR. In addition, the configurations that include oblivion, apart from significantly reducing case base size in comparison to DS, also outperform NR. Specifically, MG-O and LE-O are much more efficient in terms of case base size than NR. Most importantly, DD-O and DE-O are significantly more effective on case base size than NR. We argue that the outstanding reduction in case base size achieved by DD-O and DE-O is the major reason for their reduction in problem-solving time shown in Section 4.4.2.

Our answer to the second question is that oblivion exerts a positive effect on case base maintenance. As a result, it prevents an increase of the case base size and inhibits the swamping problem.

Another question that arises from our analysis on the case base is how does our model distribute cases within the case base. Fig. 9 shows the case base distribution for each configuration. In our experiments, we have tested a 10-fold cross validation, so that our original case base represents 90% and there is a 10% of cases that will be solved. Notice that never retain (NR) and different similarity (DS) are used as lower and upper references. NR never retains a new case, so that its final case base will be 90% of the total possible value of 100%. DS retains a new case if there is no similar case in the case base, so it retains nearly all the cases it is solving. As shown in Fig. 9(a), DS final case base is 99.19%, comprising 90% of the original case base plus a 9.19% of cases retained. Put differently, DS adds 9.19% (named Retain in the graph) when the maximum of cases to retain is a 10%; so its percentage of retention is 91.9%. Looking at the rest of retention configurations, it can be observed that DD, DE, MG and LE retention is considerably lower than DS.

To obtain more insight into the case base distribution, we show the percentages that correspond to (a) cases retained and (b) cases forgotten.

Fig. 10 shows that there exist differences in the percentage of retention between configurations when considering different k values. We have not included DS (91.19%) in the chart, in order to highlight these differences. Fig. 10 also shows that the percentage of retention is very similar between configurations that consider retention alone and the ones that also include oblivion (e.g., DD and DD-O, MG and MG-O). Therefore, it is worth noticing that retention is slightly affected by oblivion. The results vary from a minimum of 10% (DE) to a maximum of 19% (LE)—a substantial reduction on retention compared with the figure of 91.9% recorded for DS.

Next, we detail the conclusions of a careful examination of each configuration. Interestingly, DD and DD-O present greater retention for $k = 7$. This is mostly due to the disagreement measure

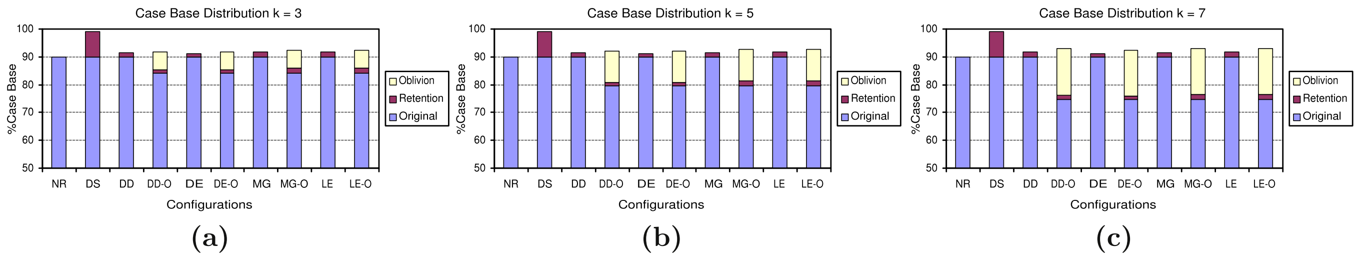

Fig. 9. Case base distribution at the end of problem-solving episodes. Each figure represents a different test on parameter k where (a) corresponds to $k = 3$, (b) $k = 5$ and (c) $k = 7$.

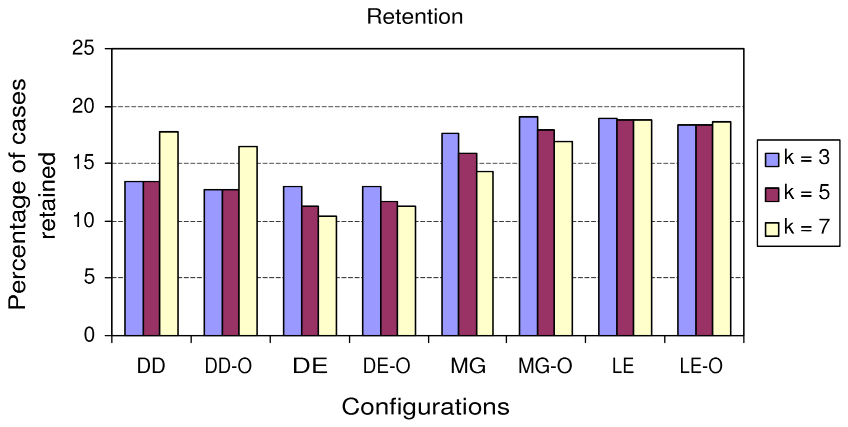

Fig. 10. Comparison of the percentage of retention between models. DS percentage of retention is 91.9%.

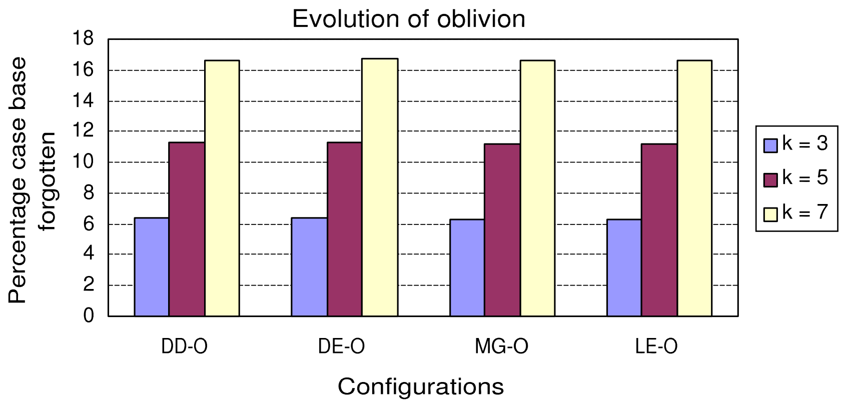

Fig. 11. Evolution of oblivion for different *k* values.

(again, see Table 3). The greater the parameter *k*, the greater the possible disagreement in our set of retrieved cases (*K*). Hence, DD and DD-O retention becomes greater for *k* = 7 because there is more likely to be disagreement between candidate cases. As for DE and DE-O, their retention is the lowest and decreases still further when increasing *k*. This is not surprising as their policy is to retain a case if it has been misclassified and if the majority in *K* is unable to classify it. When we increase the number of *k* there is more chance of obtaining a majority in the set *K* of retrieved cases, especially for those data sets with a large number of classes, so retention falls in both DE and DE-O. The same effect of *k* happens in MG and MG-O. Increasing *k* results in a greater selection of candidate cases and there is more chance of obtaining a case with a high goodness value *g*, so it will be more difficult to exceed the threshold and, as a result, the new case will not be stored. LE and LE-O strategies retain the cases that have been misclassified. This decision for retaining a case is almost the same as in DE and DE-O. The basic difference is that DE and DE-O do not store the new case when it has been misclassified but there is a case in *K* able to solve it, whereas in such conditions LE and LE-O will always retain the new case. For this reason, LE retention is always greater.

Now, in order to complete the case base distribution analysis, we focus our attention on oblivion. Going back to Fig. 9, we observe that DD-O, DE-O, MG-O and LE-O also include the percentage of oblivion. Recall that our oblivion strategy investigates whether a case goodness value is below its initial goodness value—i.e., whether it has misclassified new cases. The percentage of oblivion is directly related to the value of parameter *k*, as it denotes the number of cases whose goodness is updated at each cycle. The greater the parameter *k*, the greater the likelihood that a case will be updated. Hence, as shown in Fig. 9, the oblivion policy in the model increases the number of cases it considers for deletion, so larger *k* means higher oblivion.

Table 4
Comparison, in terms of accuracy, efficiency and case base size, of mean rank differences of each model configuration *versus* standard CBR methods: (a) DS and (b) NR. The results depicted correspond to *k* = 3. Symbols ••/○○ stand for a significant improvement/degradation of the configuration analyzed respect its reference according to the Nemenyi test. In case there is not a significant difference, a •/○ shows whether the configuration improves/degrades mean rank.

	Configuration	Accuracy	Efficiency	Case base size
(a) Reference = DS				
	DS	5.15	8.73	9.95
1	DD	○6.56	••5.85	••6.96
2	DE	○6.00	••5.01	••6.59
3	MG	○5.84	•7.66	•7.96
4	LE	○5.74	•7.60	•8.09
5	DD-O	•4.54	••3.58	••2.18
6	DE-O	•4.48	••3.76	••1.73
7	MG-O	•4.50	••5.85	••3.33
8	LE-O	•4.58	••4.63	••3.14
(b) Reference = NR				
	NR	7.26	2.34	5.09
1	DD	•6.56	○○5.85	○6.96
2	DE	•6.00	○○5.01	○6.59
3	MG	•5.84	○○7.66	○○7.96
4	LE	•5.74	○○7.60	○○8.09
5	DD-O	••4.54	○3.58	••2.18
6	DE-O	••4.48	○3.76	••1.73
7	MG-O	••4.50	○○5.85	•3.33
8	LE-O	••4.58	○○4.63	•3.14

Fig. 11 shows that there are hardly any differences in the percentage of oblivion between configurations.

The minor differences between DD-O, DE-O, MG-O and LE-O denote that the percentage of oblivion depends primarily on k and not on the policy used for retention. Furthermore, we deduce that this is mainly due to the fact that all configurations apply a retention policy that adds few cases and, as a result, the set K of retrieved cases used to solve a new case will be quite similar in all of them. As a consequence, the oblivion policy is unlikely to differ.

4.5. Lessons learned

From previous results we can now present a global analysis on performance. Table 4(a) and (b) presents summarized comparisons of our eight proposed model configurations with the two standard CBR methods NR and DS. This is done in terms of the three different performance measures considered so far. Standard methods are used as references in our experiments. For example, since NR does not add new cases, its case base size can be considered as the best reference we can compare with, but it provides the poorest reference in terms of accuracy. From the results discussed above, we can highlight the following conclusions:

– **Learning new cases is actually useful** since DD, DE, MG and LE configurations improve on the lower reference NR in terms of accuracy (see first four single bullets in accuracy column in Table 4(b).
– Furthermore, **adding all cases is not the solution**. Retention in DD, DE, MG and LE ranges from 10% (DE) to 19% (LE): they present better case base size than the lower[11] reference (DS, with 91.9%) while keeping similar accuracy (see case base size and accuracy columns in Table 4(a). Note that this benefit is also enjoyed in some configurations that use a non supervised retention strategy (i.e., DD improves significantly DS). From our results we deduce that a minimum retention is necessary to improve accuracy and, although too much retention does not decrease accuracy, it degrades efficiency. Thus, we conclude that retention becomes an important element in the design of a new CBR system.
– **Oblivion is beneficial for CBR**. Our experiments indicate that accuracy increases when our ACBR model configurations include oblivion. DD-O, DE-O, MG-O and LE-O significantly improve accuracy with respect to the lower reference NR (double bullets in first column of Table 4(b) and present more accuracy than upper reference DS (see accuracy column in Table 4(a). Therefore, another conclusion to rise is that case retention and forgetting can be performed in a case-base reasoning system without losing accuracy. Additionally, such good results come from forgetting few cases at each iteration, controlling the level of oblivion in our ACBR. We show that the percentage of cases forgotten is directly dependent on the k value in the ACBR model. Thus, ACBR provides control over the maximum level of reduction allowed.
– **ACBR maintains the case base without degrading efficiency**. Maintaining accuracy and the case base can be performed without increasing the reasoner's problem-solving time. Our results show a major benefit in efficiency for some of the configurations that retain and forget cases. The reason for this efficiency effect is that apart from adding cases, DD-O, DE-O, MG-O and LE-O also forget those cases that are not considered useful for the problem-solver. The most efficient—in terms of problem-solving time—are DD-O and DE-O, which are significantly better than the lower reference (DS) and do not significantly decrease efficiency in comparison with the upper reference (NR)—see efficiency columns in Table 4(a) and (b), respectively. DD-O and DE-O are the most effective because their process for retaining and forgetting involves only k cases, whereas MG-O may consider all case base and LE-O sometimes considers the whole case base.
– **ACBR keeps case bases compact**. According to our experiments, DD, DE, MG and LE perform better on case base size than the lower reference DS (see column case base size in Table 4(a) and worse than the upper reference NR (see Table 4(b). This result was as expected: since NR never retains, its case base size will be always lower than a configuration that does retain. Nevertheless, DD-O, DE-O, MG-O and LE-O configurations are able to keep more compact case bases than NR. The configurations that are best at reducing case base size are DD-O and DE-O which significantly outperform the upper reference method (NR)—see Table 4(b). Note that DD-O, DE-O, MG-O and LE-O configurations completely implement the ACBR model. So, we can conclude that it is able to keep case bases compact.
– **ACBR model works in supervised and non-supervised modes**. According to our experiments, our model provides excellent performance benefits in both mode types. DD-O and MG-O are not supervised while DE-O and LE-O are supervised. As our results have shown, DD-O and DE-O are more efficient in terms of accuracy, efficiency and case base size than DS and NR (see rows 5 and 6 in Table 4(a) and (b)).

[11] Notice that in terms of case base size performance follows an inverse ordering.

Table 5
Comparison of standard CBR methods and ACBR model configurations with LSVM in terms of mean rank of accuracy ($Accuracy_{rank}$) and mean rank of case base size ($CaseBase_{rank}$). Symbols ●●/○○ stand for a significant improvement/degradation with regard to LSVM according to the Nemenyi test. Similarly, symbols ●/○ stand for no significant differences. Right most column shows mean case base sizes ($CaseBase_{size}$).

	Reference = LSVM			
	Configuration	$Accuracy_{rank}$	$CaseBase_{rank}$	$CaseBase_{size}$ (%)
	LSVM	3.77	4.98	88.74
1	NR	○4.63	○5.46	90.00
2	DS	○3.94	○○6.94	99.63
3	DD-O	○4.11	●●2.61	80.70
4	DE-O	●3.73	●●1.65	80.29
5	MG-O	●3.65	●●3.09	80.88
6	LE-O	○4.15	●3.25	81.05

To summarize, DD-O, DE-O, MG-O and LE-O configurations perform significantly better than the lower and upper references. Importantly, the proposed models assist in the retention and deletion of cases, allowing the system to adaptively retain and forget cases that offer the best predictions in the domain. We have demonstrated that this approach is highly effective, even in situations where only a few cases describe the data set and there is a great percentage of missing values (e.g., data set labor—see Table 2). The most notable aspect of the adaptive case-based reasoning process is that it is able to improve the case base by using the generational experience of individual cases in both supervised and unsupervised modes. ACBR represents generational experience as goodness and it is improved or degraded depending on each case resolution accuracy.

4.6. Performance comparison with LSVM

In order to get a better insight of the actual contribution of our work, we compare—in terms of classification accuracy and case base size—our ACBR model to Local Support Vector Machines (LSVM) [30,31]. LSVM is an editing algorithm (see Section 5 for further details) that has been chosen for two main reasons. First of all, it is one of the latest noise reduction algorithms proposed in literature for instance-based learning and case-based reasoning. Secondly, it has shown considerably good results in comparison to the state-of-the-art case-base maintenance methods. Additionally, we complete our comparison by including again previous standard CBR methods: NR and DS.

From considering the performance evaluation above (see Section 4.4), we have chosen for comparison a $k = 5$ in the DD-O, DE-O, MC-O and LE-O configurations of our ACBR model. We have chosen this intermediate k value because it provides a good balance between classification accuracy and case-base reduction. On the other hand, for LSVM, we have used its implementation in FaLKM-lib [32] and it has been configured as detailed in [31]. It is important to remark that LSVM is only able to process numerical datasets with no missing values. Consequently, we use a subset of 26 out of the 40 previous data sets briefly described in Table 2 (these datasets correspond to the ones without missing values[12] and have been noted with an asterisk in the table). The selected data sets are all scaled in the range [0,1] and those data sets with nominal features have been previously transformed to numerical ones.

In order to compare all the strategies (i.e., DD-O, DE-O, MC-O and LE-O, LSVM, NR and DS) we initially rank them as explained in Section 4.3. We also evaluate whether there are significant differences between configurations by applying Friedman and Nemenyi tests. In this case, our experimental setup applies the Friedman test with 7 algorithms and 26 data sets, and thus, F_F is distributed according to the F distribution with $(7-1) = 6$ and $(7-1) \times (26-1) = 150$ degrees of freedom. The critical value of $F(6,150) = 2.46$ at the 0.05 critical level. Therefore, if the value of F_F—calculated from the mean rank of each algorithm—is higher than 2.46, then we can reject the null hypothesis. Table 5 reports each strategy's mean rank for classification accuracy as well as their mean rank for case base size.

In terms of accuracy, Table 5 indicates that DE-O and MG-O are the most accurate ones, whereas NR is the least. Nevertheless, if we compute F_F, we obtain 0.63. Since it is lower than 2.46, we can not reject the null hypothesis. Thus, we infer that none of these strategies present significant differences on accuracy.

On the other hand, in terms of case base size, Table 5 also reports that the best configuration for reducing the case base size is DE-O, whereas the worst is DS. The Friedman test obtains a $F_F = 69.03$, which is much higher than 2.46. Therefore, we can state that there is a significant difference between strategies when considering the case base size. We have also performed the Nemenyi test (whose critical difference value is $CD = 1.76$). As shown in Table 5 ($CaseBase_{rank}$ column), DD-O, DE-O and MG-O are significantly better (denoted by ••) than LSVM, which outperforms standard CBR methods (i.e., NR and DS). Additionally, LE-O's improvement is close to be significant. With the aim of analyzing the dimension of the case base reduction, last column of Table 5 provides the mean case base size for each strategy. Note that all ACBR configurations are much more efficient in terms of case base size reduction than LSVM, NR and DS. In particular, ACBR keeps on average a 80.73% of the case base whereas LSVM case base size is 88.74%.

Finally, this comparison of our ACBR model with LSVM leads us to the overall conclusion that ACBR confirgurations are able to outperform LSVM in terms of case base reduction without loosing (or even improving) classification accuracy.

[12] Breast-cancer wisconsin is a numerical dataset that contains a small percentage of missing values. In this particular dataset, missing values have been substituted by 0 values.

5. Related work

As we have seen, our ACBR proposal aims to better represent the domain during problem-solving episodes. It deals with performance (i.e., accuracy, efficiency and a compact case base) by gradually evolving the case base by means of case addition and case deletion.

In the literature, many proposals have been made for maintaining the case base to cope with the utility problem. In this section, we review the proposals that resemble ACBR.

Case addition is a key factor for any CBR system since cases are the most important source of knowledge for problem solving. Case addition mainly occurs in the retention phase[13] and it usually depends on the application context (e.g., classification, multi-agent or planning). Racine and Yang [33] showed that addition of cases may result in contradictions and inconsistences within a case base and, as a consequence, it can lead to degradation in performance. Therefore, they advocate validating a new case before it is added in order to detect redundancies and inconsistencies in a case authoring application context. In the same application context, CaseMaker [34] identifies useful cases to add to the case base by evaluating the additional coverage[14] provided by each candidate case. On the other hand, Ontañón and Plaza [19] describe case-addition strategies in multi-agent CBR systems. One of their strategies uses a measure of dispersion which comes from a variation of the *Query by Committee* (QbC) algorithm [20]. Our Degree of Disagreement retention strategy (DD) also uses this dispersion measure though it is adapted to classification tasks and to the usage of goodness values. Regarding planning tasks, Muñoz-Avila [35] presents a case-based planner performing case adaptation by derivation reply based on the contribution of retrieved cases to the overall adaptation effort. This strategy follows a simple principle to decide whether to store a solution: if the effort—measured in terms of the number of planning decisions made in solving the problem—is not greater than a pre-defined threshold, retrieval is considered non-beneficial or detrimental. In fact, our Detrimental retention strategy (DE) owes its name to the term coined by Muñoz-Avila. Although inspired by the same idea, instead of planning, DE belongs to the context of classification and its retrieval is based on the classification ability of retrieved set of cases (K) rather than on adaptation effort.

In CBR, there have also been many proposals for adding or deleting cases based on competence models [11,36–42]. Competence models evaluate the contributions of individual cases to problem-solving competence[15] [12]. ACBR manages the case base without the need for any specific competence model. Although related, goodness is not a measure of competence, but it measures the usefulness of a case for solving cases up to the current iteration. Apart from ACBR, we have also shown in conversational recommenders[16] [43] that a goodness measure enhances the retrieval quality.

In [44] Leake and Wilson discuss the role of analysis of performance over time in responding to environmental changes. Later, in planning tasks, they define RP [45] which is a fine-grained performance metric that considers a case's relative adaptation performance rather than its competence to guide case addition or deletion. In contrast to the simplicity of our goodness measure, the authors show that RP is an extremely expensive method to

[13] For this reason, case addition strategies are also known as retention strategies.

[14] Coverage of a case is the set of problems that it can solve.

[15] The range of target problems that a system can successfully solve.

[16] Although conversational recommenders have their origins in *conversational case-based reasoning (CCBR)*, they are a different application context of CBR. They guide users through a product space, alternatively making product suggestions and eliciting user feedback.

guide deletion as it requires recalculating RP values after additions or deletions.

On the other hand, Portinale et al. propose the LFF strategy that allows the addition of cases with off-line forgetting phases [46,47] in the context of multi-modal architectures combining Model-Based Reasoning (MBR) and CBR. Although their approach is quite similar in terms of intentions to ACBR, there are noticeable differences between them. Firstly, ACBR is not a hybrid architecture: it is a classic CBR approach since it only uses its own problem-solving ability for the promotion and demotion of cases. Secondly, LFF adds cases if the CBR component fails and uses MBR to solve this, whereas ACBR case-addition searches for an increase in diversity rather than considering misclassification alone. Finally, forgetting in LFF involves the analysis of all cases in the case base in order to identify the ones that are useless[17]; for this reason, it is performed only occasionally. ACBR case deletion can occur at each iteration because of its low computational cost and it only forgets those cases that err repeatedly.

Minton demonstrated in EBL[18] that performance can be improved by *discarding experience selectively* [48]. Later, Markovitch and Scott [23] also showed that forgetting has an important role in machine learning systems. In fact, many approaches (for an overview, see [8,49] and more recently [50]) have focused on controlling the case base size through case deletion or case selection policies. Most state-of-the-art approaches are editing techniques.[19]

The first editing contribution was Hart's Condensed Nearest Neighbour rule (CNN). Next, some improvements over the CNN rule were presented: RNN (Reduced Nearest Neighbour rule) [51] and SNN (Selective Nearest Neighbour) [52]. Furthermore, in recent years, different variations of the CNN rule have also been proposed: GCNN (Generalized Condensed Nearest Neighbour Rule) [53]; FCNN (Fast Condensed Nearest Neighbor rule) [54]; and a variation for text categorization [55].

In addition to CNN, the most well-known editing techniques are: ENN (Edited Nearest Neighbour) [56]; RENN (Repeated Edited Nearest Neighbour) [57]; instance-based learning algorithms (IBL) [58]; and ordering removal algorithms (DROP1—Decremental Reduction Optimization Procedure—to DROP5) [49]. All these editing techniques are pre-processing algorithms that remove noise and a varying degree of boundary cases. Nevertheless, they provide no control over the level of reduction. More recently, has been proposed TER (Threshold Error Reduction) [59] which controls the level of reduction while identifying and removing noisy and boundary cases with the aid of a local complexity measure. Furthermore, Pan et al. [60] introduced a new case-base mining framework and the KGCM (Kernel-based Greedy Case-base Mining) algorithm. The level of reduction in the ACBR model is controlled by means of the k parameter.

Moreover, most of the deleting techniques based on a competence model (e.g., [12,37,40]) are editing techniques, too. More recent approaches use the competence properties of the training cases to determine which ones to include in the edited set. Thus, for example, Brighton and Mellish [50] use the case competence properties of cases in their Iterative Case Filtering (ICF) algorithm. On the other hand, Delany [61] presented a case profiling technique that categorizes each case in the case-base. The profile is based on three characteristics that are derived from constructing a competence model of the case base. Her evaluation identifies which cases are of interest to maintain and which ones to remove in a case base. It is interesting to note that our ACBR model keeps those cases of interest and forgets those suggested to be deleted.

To the best of our knowledge, the most recent editing proposals are MACE (Maintenance by A Committee of Experts) [62] and LSVM (Local Support Vector Machines) [30,31]. The idea behind MACE is that the strengths of one expert algorithm will compensate for the weaknesses of another. MACE presents different combinations of the most commonly-used maintenance algorithms. Regarding LSVM, it brings the benefits of maximal margin classifiers to bear on noise reduction. LSVM is based on the probabilistic output of the Local Support Vector Machine classifier trained on the neighbourhood of each training set example. LSVM has shown to be more effective in terms of accuracy than most well-known state-of-the-art approaches.

Since most case deletion approaches are editing techniques we chose for our comparison the most recent editing approach: LSVM. Additionally, we selected LSVM for its effectiveness in terms of accuracy and case-base reduction. Nevertheless, ACBR is not an editing or a pre-processing algorithm; it makes full use of training set of cases and controls case deletion with the aid of parameter k while testing problem-solving episodes.

Regarding the CBR cycle, we can safely state that nowadays the maintenance [63] of the case base is considered a constituent part of the CBR. Similarly, our extension of the CBR cycle neither represents a fundamental change in the reasoning process. Previously, at the workshop 'Automating the Construction of Case-Based Reasoners' [64], many participants recognized that there are more than four phases. The workshop's conclusion was to propose two new phases for the CBR: review and reflect. The review phase is inserted before the retain phase and it ensures that only interesting and valuable cases are stored. The second additional phase reflects feature weights, similarity metrics, case coverage, and so on. ACBR also includes a review phase before retention although it is intended to provide goodness update and forgetting capability. Previously, Aha in [65] had proposed an additional review phase that evaluated the outcome of the constructed solution. If the outcome was not acceptable, then it decided to continue revising this solution to solve the current case. As we have seen, our *review* phase is meant to review the knowledge in our case base rather than refining a specific solution. Reinartz et al. [66] also integrated maintenance in the overall CBR process. They propose two additional phases after retention: review and restore. Their review monitors and assesses the quality of the case base by defining several case properties, whereas the restore phase uses modify operators to change its contents. There are three basic differences vis-á-vis ACBR: (1) their review phase is after the retention phase; (2) it is computationally more expensive since it considers each case in relation to other cases in the case base to assess quality; and (3) case properties only handle nominal attributes, so numerical attributes should be discretized. In addition, as the author notes, the crucial question is which modify operator is useful to resolve which quality problem. Goodness in ACBR could be seen as a new case property. However, it represents more than this: it measures the usefulness of a case during problem-solving episodes. In other words, goodness reminds a case of its own history of experiences (i.e., the generational experience) for problem-solving. We argue that the personal experiences of each case are enough for promotion and demotion of cases. Consequently, ACBR evolves the case base based on goodness, so we consider our approach as a goodness-based maintenance.

6. Conclusions

This paper introduces an adaptive case-based reasoning model (ACBR) which approximates CBR to the human ability to remember and forget. We argue that both capabilities introduced as basic processes to be combined in an adaptive case-based reasoning model

[17] Useless cases in LFF are those that have never been retrieved or produce failure.

[18] EBL: Explanation-Based Learning.

[19] Reduce case base size by revising the training set.

are appropriate to ensure performance (i.e., accuracy, efficiency and a compact case base) in the long term. ACBR iteratively develops the case base by retaining and forgetting cases. We have proposed different retention and forgetting strategies that use a goodness measure to predict which cases will subsequently be remembered or forgotten. To represent goodness we have chosen a simple value and a fast update formula. Thus, its computational cost is negligible.

We have demonstrated the efficacy of our ACBR model against a benchmark of 40 data sets from UCI Machine Learning repository. In general, our experiments indicate that ACBR model has the potential to deliver worthwhile performance benefits. The evaluation results confirm that all model configurations tested (i.e., DD-O, DE-O, MG-O and LE-O) yield statistically significant improvement—in terms of accuracy, efficiency and case base size—compared with the standard CBR methods used as reference (NR or DS). Nevertheless, our statistical analysis show that, if we are particularly interested in improving one of these performance measures, some configurations may be more convenient than others.

ACBR obtains better accuracy than standard CBR configurations. This good accuracy is achieved because it gently and continuously adapts its case base, thus keeping diversity high and removing harmful knowledge. Our results also show that ACBR significantly improves efficiency compared with a CBR method that includes retention. There are two possible reasons for the efficiency of our model. First, a reduction in the case base allows faster retrieval and consequently improves efficiency. Second, goodness is simple and it is rapidly updated so it does not increase the computational effort of CBR and there is no loss on efficiency. Moreover, ACBR generates more compact case bases than standard CBR methods. This is largely due to continuous promotion and demotion of cases in the long term.

In conclusion, the evaluation of our ACBR model provides evidence that the maintenance of the case base is effective even when done smoothly. The promotion and demotion of cases during problem-solving episodes allows: (1) an increase in diversity because new cases will be added; (2) a decrease in harmful knowledge since the cases that impair accuracy will be forgotten; and (3) a control of the case base growth to reduce the swamping problem. These outcomes are especially critical in certain domains, such as diagnosis, where a decrement in performance may not be acceptable. It is also encouraging to see that our model provides excellent performance benefits in unsupervised mode in which cases are not assigned class labels but where instead we can measure disagreement or search for a minimal goodness.

These results encourage us to keep exploiting the goodness concept so to use it during other phases of our ACBR cycle. In this manner, we plan to study how goodness can be used in the Retrieve phase, since it could be combined with our current similarity criteria. Similarly, indexing tasks could potentially benefit from using the goodness concept. Finally, in the same way that we have studied several retention strategies, in future work we also expect to consider alternative oblivion strategies that consider temporal features, since time is naturally coupled with adaptive approaches.

Acknowledgement

This work has been supported in part by projects CONSOLIDER-INGENIO CSD 2007-00018, CSD 2007-00022, TIN2009-14702-C02-02 and TIN2009-14404-CO2-02.

References

[1] C.K. Riesbeck, R.C. Schank, Inside Case-Based Reasoning, Lawrence Erlbaum Associates, Hillsdale, NJ, US, 1989.
[2] E. Golobardes, X. Llorà, M. Salamó, J. Martí, Computer aided diagnosis with case-based reasoning and genetic algorithms, Knowledge-Based Systems 15 (1–2) (2002) 45–52.
[3] D. Patterson, N. Rooney, M. Galushka, V. Dobrynin, E. Smirnova, Sophia-tcbr: a knowledge discovery framework for textual case-based reasoning, Knowledge-Based Systems 21 (5) (2008) 404–414.
[4] I.I. Bittencourt, E. Costa, M. Silva, E. Soares, A computational model for developing semantic web-based educational systems, Knowledge-Based Systems 22 (4) (2009) 302–315.
[5] F. Gasparetti, A. Micarelli, F. Sciarrone, A web-based training system for business letter writing, Knowledge-Based Systems 22 (4) (2009) 287–291.
[6] J. Jim Davies, A.K. Goel, P.W. Yaner, Proteus: visuospatial analogy in problem-solving, Knowledge-Based Systems 21 (7) (2008) 636–654.
[7] A. Aamodt, E. Plaza, Case-based reasoning: foundations issues, methodological variations, and system approaches, AI Communications 7 (1994) 39–59.
[8] D.B. Leake, D.C. Wilson, Case-base maintenance: dimension and directions, in: Proceedings of the Fourth European Workshop on Case-Based Reasoning, 1998, pp. 196–207.
[9] B. Smyth, P. Cunningham, The utility problem analysed: a case-based reasoning perspective, in: European Workshop on Case-Based Reasoning, 1996, pp. 392–399.
[10] P.D. Scott, S. Markovitch, Knowledge considered harmful, in: Proceedings of IEEE Colloquium on Knowledge Engineering, London, 1990, pp. 1–5.
[11] B. Smyth, E. McKenna, Competence models and the maintenance problem, Computational Intelligence 17 (2) (2001) 235–249.
[12] B. Smyth, M. Keane, Remembering to forget: a competence-preserving case deletion policy for case-based reasoning systems, in: Proceedings of the 13th International Joint Conference on Artificial Intelligence, 1995, pp. 377–382.
[13] M. Mykola Galushka, D. Patterson, Intelligent index selection for case-based reasoning, Knowledge-Based Systems 19 (8) (2006) 625–638.
[14] R.C. Schank, R.P. Abelson, Scripts Plans Goals and Understanding: An Inquiry into Human Knowledge Structures, Lawrence Erlbaum Associates, Hillsdale, NJ, 1977.
[15] R. Schank, Dynamic Memory: A Theory of Reminding and Learning in Computers and People, Cambridge University Press, New York, NY, 1982.
[16] K.A. Paller, A.D. Wagner, Observing the transformation of experience into memory, Trends in Cognitive Science 6 (2002) 93–102.
[17] M. Salamó, E. Golobardes, Dynamic case-base maintenance for a case-based reasoning system, in: IX Ibero-American Conference on Artificial Intelligence, Advances in Artificial Intelligence – IBERAMIA 2004, Springer, 2004, pp. 93–103.
[18] R.S. Sutton, A.G. Barto, Reinforcement Learning. An introduction, The MIT Press, 1998.
[19] S. Ontañón, E. Plaza, Collaborative case retention strategies for cbr agents, in: K. Ashley, D. Bridge (Eds.), Proceedings of Fifth International Conference on Case-Based Reasoning, Springer, 2003, pp. 392–406.
[20] H.S. Seung, M. Opper, H. Sompolinsky, Query by Committee, Computation Learning Theory (1992) 287–294.
[21] M. Salamó, E. Golobardes, Dynamic experience update for a case-based reasoning system, in: Proceedings of Second Starting AI Researcher Symposium, Frontiers in Artificial Intelligence and Applications, vol. 109, 2004, pp. 158–164.
[22] P.E. Hart, The condensed nearest neighbour rule, IEEE Transactions on Information Theory 14 (1968) 515–516.
[23] S. Markovitch, P.D. Scott, The role of forgetting in learning, in: Proceedings of the Fifth International Conference on Machine Learning, 1988, pp. 459–465.
[24] A. Asuncion, D.J. Newman, UCI machine learning repository, 2007. <http://www.ics.uci.edu/~mlearn/MLRepository.html>.
[25] M. Salamó, E. Golobardes, BASTIAN: incorporating the rough sets theory into a case-based classifier system, in: Butlletí de l'acia: III Congrés Català d'Intel·ligència Artificial, Barcelona, Spain, 2000, pp. 284–293.
[26] M. Friedman, The use of ranks to avoid the assumption of normality implicit in the analysis of variance, Journal of the American Statistical Association 32 (200) (1937) 675–701.
[27] M. Friedman, A comparison of alternative tests of significance for the problem of m rankings, The Annals of Mathematical Statistics 11 (1) (1940) 86–92.
[28] J. Demšar, Statistical comparisons of classifiers over multiple data sets, Journal Machine Learning Research 7 (2006) 1–30.
[29] P. Nemenyi, Distribution-free multiple comparisons, Ph.D. Thesis, Princeton University, 1963.
[30] N. Segata, E. Blanzieri, P. Cunningham, A scalable noise reduction technique for large case-based systems, in: ICCBR'09: Proceedings of the Eighth International Conference on Case-Based Reasoning, Springer, 2009, pp. 328–342.
[31] N. Segata, E. Blanzieri, S. Delany, P. Cunningham, Noise reduction for instance-based learning with a local maximal margin approach, Journal of Intelligent Information Systems (2010), doi:10.1007/s10844-009-0101-z.
[32] N. Segata, Falkm-lib v1.0: a library for fast local kernel machines, Technical Report, DISI-09-025, DISI, University of Trento, Italy, 2009. Available from: <http://disi.unitn.it/segata/FaLKM-lib>.
[33] K. Racine, Q. Yang, Maintaining unstructured case-bases, in: Proceedings of the Second International Conference on Case-Based Reasoning, 1997, pp. 553–564.
[34] D. McSherry, Automating case selection in the construction of a case library, Knowledge-Based Systems 13 (2–3) (2000) 133–140.
[35] H. Muñoz-Avila, Case-base maintenance by integrating case-index revision and case-retention policies in a derivational replay framework, Computational Intelligence 17 (2) (2001) 280–294.

[36] L. Portinale, P. Torasso, P. Tavano, Dynamic case memory management, in: Proceedings European Conference on Artificial Intelligence, 1998, pp. 73–77.
[37] B. Smyth, E. McKenna, Building compact competent case-bases, in: Proceedings of the Third International Conference on Case-Based Reasoning, 1999, pp. 329–342.
[38] J. Zhu, Q. Yang, Remembering to add: competence-preserving case-addition policies for case base maintenance, in: Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence, vol. 1, 1999, pp. 234–239.
[39] M. Salamó, E. Golobardes, Rough sets reduction techniques for case-based reasoning, in: Proceedings of the Fourth International Conference on Case-Based Reasoning, Vancouver, BC, Canada, 2001, pp. 467–482.
[40] M. Salamó, E. Golobardes, Deleting and Building Sort Out Techniques for Case Base Maintenance, in: S. Craw, A. Preece (Eds.), Proceedings of the Sixth European Conference on Case-Based Reasoning, Springer, 2002, pp. 365–379.
[41] M. Salamó, E. Golobardes, Hybrid Deletion Policies for Case Base Maintenance, in: I. Russell, S. Haller (Eds.), Proceedings of the 16th International Florida Artificial Intelligence Research Society Conference, AAAI Press, 2003, pp. 150–154.
[42] M. Salamó, E. Golobardes, Global, Local And Mixed Case Base Maintenance Techniques, Proceedings of the Sixth Congrés Català d'Intel·ligència Artificial, vol. 13, IOS press, 2004, pp. 127–134.
[43] M. Salamó, S. Escalera, P. Radeva, Quality enhancement based on reinforcement learning and feature weighting for a critiquing-based recommender, in: Case-Based Reasoning Research and Development, Proceedings of Eighth International Conference on Case-Based Reasoning, 2009, pp. 298–312.
[44] D.B. Leake, D.C. Wilson, When experience is wrong: examining CBR for changing tasks and environments, in: Proceedings of the Third International Conference on Case-Based Reasoning, 1999, pp. 218–232.
[45] D.B. Leake, D.C. Wilson, Remembering why to remember: performance-guided case-base maintenance, in: Proceedings of the Fifth European Workshop on Case-Based Reasoning, 2000, pp. 161–172.
[46] L. Portinale, P. Torasso, P. Tavano, Speed-up, quality and competence in multi-modal reasoning, in: Proceedings of the Third International Conference on Case-Based Reasoning, 1999, pp. 303–317.
[47] L. Portinale, P. Torasso, Automatic case base management in a multi-modal reasoning system, in: Proceedings of the Fifth European Workshop on Advances in Case-Based Reasoning, Springer-Verlag, London, UK, 2000, pp. 234–246.
[48] S. Minton, Selectively generalizing plans for problem solving, in: Ninth International Joint Conference on Artificial Intelligence, Morgan Kaufmann, 1985, pp. 596–599.
[49] D.R. Wilson, T.R. Martinez, Reduction techniques for instance-based learning algorithms, Machine Learning 38 (2000) 257–286.
[50] H. Brighton, C. Mellish, Advances in instance selection for instance-based learning algorithms, Data Mining and Knowledge Discovery 6 (2002) 153–172.
[51] G.W. Gates, The reduced nearest neighbor rule, IEEE Transactions on Information Theory 18 (3) (1972) 431–433.
[52] G.L. Ritter, H.B. Woodruff, S.R. Lowry, T.L. Isenhour, An algorithm for a selective nearest neighbor decision rule, IEEE Transactions on Information Theory 21 (6) (1975) 665–669.
[53] C.-H. Chou, B.-H. Kuo, F. Chang, The generalized condensed nearest neighbor rule as a data reduction method, in: ICPR'06: Proceedings of the 18th International Conference on Pattern Recognition, IEEE Computer Society, 2006, pp. 556–559.
[54] F. Angiulli, Fast nearest neighbor condensation for large data sets classification, IEEE Transactions on Knowledge and Data Engineering 19 (11) (2007) 1450–1464.
[55] X. Hao, C. Zhang, H. Xu, X. Tao, S. Wang, Y. Hu, An improved condensing algorithm, in: Proceedings of the Seventh IEEE/ACIS International Conference on Computer and Information Science, IEEE Computer Society, 2008, pp. 316–321.
[56] D.L. Wilson, Asymptotic properties of nearest neighbor rules using edited data, IEEE Transactions on Systems, Man and Cybernetics 2 (3) (1972) 408–421.
[57] I. Tomek, An experiment with the edited nearest-neighbor rule, IEEE Transactions on Systems, Man and Cybernetics 6 (6) (1976) 152–448.
[58] D. Aha, D. Kibler, Instance-based learning algorithms, Machine Learning 6 (1991) 37–66.
[59] S. Massie, S. Craw, N. Wiratunga, When similar problems don't have similar solutions, in: Proceedings of the Seventh International Conference on Case-Based Reasoning, 2007, pp. 92–106.
[60] R. Pan, Q. Yang, S.J. Pan, Mining competent case bases for case-based reasoning, Artificial Intelligence 171 (16–17) (2007) 1039–1068.
[61] S.J. Delany, The good, the bad and the incorrectly classified: Profiling cases for case-base editing, in: ICCBR'09: Proceedings of the 8th International Conference on Case-Based Reasoning, Springer-Verlag, 2009, pp. 135–149.
[62] L. Cummins, D. Bridge, Maintenance by a committee of experts: the mace approach to case-base maintenance, in: ICCBR'09: Proceedings of the 8th International Conference on Case-Based Reasoning, Springer-Verlag, 2009, pp. 120–134.
[63] R.L. de Mantaras, D. McSherry, D. Bridge, D. Leake, B. Smyth, S. Craw, B. Faltings, M.L. Maher, M.T. Cox, K. Forbus, M. Keane, A. Aamodt, I. Watson, Retrieval, reuse, revision and retention in case-based reasoning, Knowledge Engineering Review 20 (3) (2006) 215–240.
[64] I. Watson, Report of the workshop on automating the construction of case-based reasoners at the Sixteenth International Joint Conference on Artificial Intelligence, Technical Report, 1999.
[65] D.W. Aha, The omnipresence of case-based reasoning in science and application, Knowledge-Based Systems 11 (5–6) (1998) 261–273.
[66] T. Reinartz, I. Iglezakis, T. Roth-Berghofer, Review and Restore for Case-Base Maintenance, Computational Intelligence: Special Issue on Maintaining Case-Based Reasoning Systems 17 (2) (2001) 214–234.