

Releasing Memory Space through a Case-Deletion Policy with a Lower Bound for Residual Competence*

Flavio Tonidandel¹ and Márcio Rillo^{1,2}

¹ Universidade de São Paulo - Escola Politécnica
Av. Luciano Gualberto 158, trav3
05508-900 - São Paulo - SP - Brazil

² Faculdade de Engenharia Industrial
Av. Humberto de A. Castelo Branco, 3972
09850-901 - São Bernardo do Campo - SP - Brazil
e-mails: flavio@lac.usp.br ; rillo@lsi.usp.br - phone: +55 11 3818 5530

Abstract. The number of techniques that focuses on how to create compact case-base in case-base maintenance has been increasing over the last few years. However, while those techniques are concerned with choosing suitable cases to improve the system performance, they do not deal with the problem of a limited memory space, which may affect the performance as well. Even when a CBR system admits only a limited number of stored cases in memory, there will still exist the *storage-space problem* if it has cases that vary in size, as in most of case-based planning domains. This paper focuses on case-deletion policy to release space in the case memory, which can guarantee the competence-preserving property and establish a theoretical lower bound for residual competence.

1 Introduction

The importance of case-base maintenance has been increasing in the CBR community since it is essential to improve the performance of a system that gets information from a case-base. Recent works have shown that the system performance is affected by the competence - "*the range of target problems that can be successfully solved*" [7] - and the efficiency - "*the computational costs of solving a set of target problems*" [7] - of a case-base.

The main factor that affects the competence and the efficiency, and consequently the performance, of a CBR system is the size of the case memory. Much significant researches are concerned with reducing the size of the case-base. Some of them focus on developing methods that limit the number of cases and concerns about competence-preserving [12,7] or efficiency-preserving [2].

The competence and the efficiency can be extremely affected if the memory is full. In fact, how is it possible to improve the competence when there is no space left for a new case with high competence? And how is it possible to improve the efficiency when the memory is full and a new case, if it could be inserted, would solve many target problems of other cases? The *storage-space problem*, as we call the problem

* This work is supported by FAPESP under contract no. 98/15835-9.

that arises when the memory becomes full, may affect drastically the improvement of the system performance.

This paper is a detailed extension of [10]. It differs from previous approaches since it considers the possibility of cases varying in size and deals with the *storage-space problem* by releasing enough space for a useful new case. A case becomes useful when it improves the competence or even the efficiency of the system. To release space, a case-deletion policy is designed to choose cases to delete in order to ensure a lower bound for residual competence, similar to that one presented in [12].

This paper is organized as follow. Section 2 presents an overview of the case-base maintenance area and related works. Section 3 proposes a case-deletion policy and a lower bound for residual competence. Section 4 presents some experiments and finally section 5 concludes this paper.

2 Related Works in Case-Base Maintenance

In recent years, a great amount of interest has been paid to Case-Base maintenance (CBM), a CBR research branch.. The reason of that rises from the desire to improve the performance of CBR systems. Recent work highlighted the importance of the maintenance in a CBR process proposing a new CBR cycle that includes some maintenance steps - Review and Restore [6]. An excellent definition of CBM can be found in [3]:

A case-base maintenance implements policies for revising the organization or contents of the case-base in order to facilitate future reasoning for a particular set of performance objectives.

This definition shows that case-base maintenance is concerned with performance improvement, and two factors affect it significantly: the competence of the case-base and the efficiency. This is because none of CBR system can improve the performance if the case-base cannot solve efficiently a great number of problems.

However, increasing the competence does not mean increase the number of cases, otherwise the system efficiency can be degraded rather than improved. Therefore, strategies that are concerned with reducing the case-base size have been the focus of the case-base maintenance research.

Different strategies have been developed. One of these is the *selective deletion*. It deletes cases according to a case-deletion policy. It can be made with techniques involving the utility of each case [4,5] or involving the overall competence of a case-base [9]. Smyth and Keane's work [9] introduces a competence-preserving approach that guides their case-deletion policy through a classification of types of cases.

Another strategy is the *selective utilization*. It tries to improve the efficiency by making available just a subset of the case-base to the retrieval process [7,11,12].

The methods employed in each strategy are also different. They can consider the competence of each case [7,12], or the adaptation effort that each case can request [2].

Instead of guiding the development of other *selective deletion* techniques, the competence-preserving approach has been used to guide some *selective utilization* policies [7,12] that choose cases with more competence from the case-base and make

them available for the search engine. An interesting approach is RC-CNN [7], which is a *selective utilization* policy that constructs compact case-bases with high competence. It is based on the *CNN (Condensed Nearest Neighbor)* method with a suitable measurement called *Relative Coverage*.

Another *selective utilization* approach is the case-addition policy proposed by Zhu and Yang [12]. They achieved a lower bound that is around 63% of the relation between the optimal choices of cases and the choices made by their algorithm.

Similar to competence-preserving approach, efficiency-preserving techniques can also improve the performance of a CBR system, but they are concerned with the adaptation costs. Leake and Wilson [2] recently addressed an efficiency-preserving algorithm called RP-CNN. The approach is similar to RC-CNN, but it focuses on preserving the efficiency of the original case-base through a measurement called *Relative Performance*. It takes into account the relative adaptation costs of each case.

However, none of these strategies addresses the problem that may arise when limited space is available for storing cases. This limitation can also create a barrier for a case-based system and its performance, and we call it *storage-space problem*. Even when a limited number of stored cases are defined, the problem can still appear. This is because there are many domains where cases vary in size and therefore use different amounts of memory. One example of these domains is a case-based planning systems domain, where cases are a sequence of actions, and they can have different numbers of actions in their composition [10,11].

One could argue that there is no problem because the quantity of memory can be augmented. However, the quantity of memory will always be limited to the store case capacity. Additionally, depending on the complexity of the domain and how many problems the domain has, the *storage-space problem* persists.

In fact, memory space limitation requires that some cases should be deleted. The previous deletion-based approaches do not present good results when applied to solve the *storage-space problem*. In addition, although the *selective utilization* approaches [7,12] present good results and a lower bound for residual competence [12], they are not efficient for releasing space in memory, because in almost all situations the number of cases to be deleted is smaller than the number of cases to be retained.

Therefore, one way is to use a *selective deletion*. However, the Smyth and Keane's deletion-based policy [9], as shown in [12], cannot guarantee a lower bound for residual competence and does not perform as well as *selective utilization* approaches, as shown empirically in some experiments in section 4.

This paper does differ from previous approach in considering the possibility of cases having varied sizes. In addition, it proposes a competence-preserving case-deletion policy that can be applied to release space in order to permit the storage of new cases. As shown in section 4, in some circumstances, this case-deletion policy can preserve the competence as high as the *selective utilization* approaches can do but ten times faster.

3 A Case-Deletion Policy

After the adaptation phase succeeds, a CBR system can find a solution to a target problem. This solution might be a new case and, consequently, will have the possibility of being stored in the memory for future use. A new case will be stored if it is useful, i.e., if it increases the total competence of the case-base. However, when the memory is full, a new case is stored in the memory only if the lost competence imposed by the deletion of some cases is less than the competence increasing resulted by the storage of a new case.

Therefore, it is necessary to design a case-deletion policy which releases enough space only if a new case can increase the competence of the case-base.

3.1 Case Competence

The competence, or coverage, is the set of problems that a case, or case-base, can solve [7,9]. The same notion is given in [12], that also highlights the importance of determining the competence through adaptation costs and a similarity metric.

For competence definition, we will use the definition $CoverageSet(c \in C) = \{t \in T: Solves(c,t)\}$, from [7], with the definition of neighborhood function $N(x)$, from [12]. In this paper, we consider the $N(x)$ formula equal to the set $CoverageSet(c \in C)$, where x and c are the same case in the set of cases C for the same target problem t in the space of problem T . Considering a set $M \subseteq C$ of cases in the case-base, we can establish the following definition for competence of a group of cases $X \subseteq M$:

$$Competence(X) = |N(X)| \quad (1)$$

where:

$$N(X) = \bigcup_{x \in X} N(x)$$

With the above definition, we can define the shared competence among cases in the case-base:

$$Shared(X,Y) = \{t \in N(X) : N(X) \cap N(Y)\} \quad (2)$$

The shared competence is the set of target problems that a group of cases X and a group of cases Y can solve together. Conversely, it is possible to define, as in [7], a set of cases that can be used to solve a specific target problem:

$$ReachabilitySet(t \in T) = \{c \in C \mid t \in N(c)\} \quad (3)$$

These definitions are used to redefine the **benefit** formula proposed in [12]. It calculates the number of target problems that only the case x can solve with respect to a set W of chosen cases. Assuming the frequency function $P(x)$ equal to 1, the **benefit** becomes:

$$Bft(x) = |N(x) - Shared(x,W)| \quad (4)$$

Following the same idea of benefit formula, the *injury* of a set of cases X is defined:

$$Injury(X \in M) = |N(X) - Shared(X, M-X)| \quad (5)$$

The *injury* is calculated by the analysis of the damage that can be caused in the total competence if X is removed from the case-base. It is the competence of X minus the competence of X that is reached by other cases in $M-X$. This *injury* formula calculates the competence that the case decreases from the case-base when deleted.

```

func Recalc(c)
for each  $t \in N(c)$  do
    Determine ReachabilitySet( $t$ )
    if |ReachabilitySet( $t$ )|=1 then
         $Injury(c' \in ReachabilitySet(t)) = Injury(c') + 1$ 
    endif
endfor
    
```

Fig. 1. The **Recalc()** algorithm, where c is a deleted case

However, formula 5 is not useful to be computationally implemented, because it will takes many cases in the shared set to calculate the *injury* of a single case. Another feasible way to determine the *injury* is calculating the variation of the *injury* of a case c' caused by a deletion of a case c . It is made by the **Recalc** (c) algorithm presented in figure 1. It recalculates the *injury* of cases after the deletion of a case c .

The complexity when recalculating the *injury* through **Recalc()** algorithm and the *benefit* in Zhu and Yang's case-addition algorithm, are similar. The difference between both is that the *injury* is recalculated with respect to the retained cases in the memory, and the *benefit* is recalculated with respect to the chosen cases.

Until now, we have been concerned with the competence and *injury* of cases; from this point we will be concerned with the total competence of a case-base as well. However, the determination of the total competence is not a trivial process because any case can share competence with a great number of other cases in the case-base. Therefore, the following simple Lemma can be stated for the total competence, $TC(M)$ for a case-base M :

Lemma: $TC(M) \leq \sum_{x \in M} Competence(x)$

The Lemma above states that the total competence, TC , is at most, the summation of the competence of each case in the case-base. The proof can be obtained directly from formula 1 and the union of cases. This Lemma will be useful in Theorems ahead.

3.2 The Case-Deletion Algorithm

There are different types of cases. Smyth and Keane [9] classify four types of cases: (1) pivotal cases are the ones which cannot be solved by other cases; (2) spanning

cases are the ones that share the competence independently covered by other cases; (3) auxiliary cases are the ones whose competence is subsumed by other cases competence; and (4) support cases are redundant cases that solve the same problems and, consequently, have the same competence.

According to formula 5, auxiliary cases, support cases and some spanning cases can result in $Injury(x) = 0$. Thus, it is necessary to identify which cases with same *injury* should be deleted. As highlighted in [9], some spanning cases can turn to pivotal cases if some auxiliary cases are deleted; so deleting a spanning case in this circumstance is not a good choice.

```
func minimal-injury (c', M): set_cases
1. set Mb = available(memory);
2. While size(c') > Mb and M has auxiliary cases
   2.1. Select an auxiliary case y with larger size;
   2.2. Recalc (y).
   2.3. M = M - y;
   2.4. Mb = available(memory)
3. While size(c') > Mb
   3.1. Select a case c with the minimal injury.
   3.2. Recalc (c).
   3.3. M = M - c;
   3.4. Mb = available (memory)
4. return (M);
```

Fig. 2. The case-deletion algorithm, where c' is a new case and M is a set of stored cases.

To avoid wrong choices, a heuristic method is used to delete auxiliary cases first. This is because by their definition, they do not cause any competence decreasing when removed.

The auxiliary cases can be labeled together with the determination of the competence of each case. There is no need to do the process of labeling and determining the competence each time the deletion algorithm is performed. When a case c is deleted, it is just necessary to recalculate the competence and the shared groups with respect to this case c . Therefore, the process to determine auxiliary cases, shared groups and competence for each case are done just one time and off-line.

The case-deletion algorithm is presented in Figure 2. It incorporates a heuristic method that considers the deletion of auxiliary cases first. Consequently, the first choices of the case-deletion algorithm, called *minimal-injury*, are identical to the first choices of the Smyth and Keane's algorithm [9]. However, instead of using all types of cases, the proposed case-deletion algorithm uses only the auxiliary type. It avoids the re-labeling of cases as Smyth and Keane's approach, because the deletion of an auxiliary case does not change the type of any other auxiliary case in the case-base.

Some other heuristic methods are also performed by *minimal-injury* algorithm. One of them chooses the case with larger size among auxiliary cases and cases with the same *injury*. It also considers support cases as a special case of a pivotal case with some auxiliary cases inside.

The case-deletion algorithm presented in Figure 2 is a greedy algorithm that, at each step, chooses a case with minimal *injury* to be deleted. The algorithm will mark cases to be deleted until it releases enough space. However, if the lost competence is

bigger than the benefit of a new case, the algorithm will stop the process and will not delete the marked cases. If at that point enough space is already released, the marked cases will be deleted in order to allow the system inserting the new case.

The implementation of the case-deletion algorithm has its complexity imposed on by the recalculation of the *injury*, similar to the recalculation of the *benefit* in Zhu and Yang's algorithm [12]. If we do not assume the *representativeness assumption* [7], the performance of both algorithms becomes dependent of the case-base concentration, as we see in the empirical results in section 4.

The *minimal-injury* algorithm performs the best choice at each step because it is a greedy algorithm, although it does not guarantee the resulting set of selected cases to be optimal. However, it is possible to define, similar to Zhu and Yang's work [12], a lower bound for the residual competence.

3.3 Establishing a Lower Bound

Let \bar{S} be the set of cases that remain following the deletion of a set \bar{S} of cases from the case-base M . In this notation, $B \cup \bar{B} = A \cup \bar{A} = M$, where \bar{B} is the optimal set of cases for deletion and \bar{A} is the set of cases selected for deletion by the algorithm. Any X and \bar{X} are always disjoint sets. Thus, the notation for competence is:

Definition 1: $X^C = Competence(X) = |N(X)|$

The set $N(X)$ allows the calculation of the competence of cases group X . Similarly, we can define the calculation of the lost competence from the disjoint set of $N(X)$:

Definition 2: $\bar{X}^{LC} = LostCompetence(\bar{X}, M) = |N(M) - N(X)|$

The lost competence is the total competence M minus the competence remaining after the deletion of cases in the set \bar{X} . These definitions allow that the total competence may be the sum of the competence X^C and the lost competence \bar{X}^{LC} .

However, although the definition of *injury* is different of *Lost Competence* definition, we can prove that the calculation of the *injury* is correct with respect to definition 2 directly by some sets operations, i.e., considering that \bar{X} is a set of deleted cases from M , $Injury(\bar{X}) = LostCompetence(\bar{X}, M)$.

It shows that the calculation of the *injury* is accurate and that it really represents the competence that is lost after the deletion of cases.

The *injury* formula is used by the case-deletion algorithm presented in Figure 2. As emphasized before, this algorithm is a greedy algorithm and does not perform the optimal choice at each step. However, it is possible to find a lower bound for the optimal choices for deletion and the choices performed by the algorithm.

In order to define a lower bound for residual competence, denoted by A^C , we must proof some theorems and derive algebraically a formula for A^C/M^C .

Focusing on lost competence, the following Theorem shows the relation between the r choices of the deletion algorithm and the optimal deletion of r cases.

Theorem 1: The optimal lost competence for r cases is at least 58% of the lost competence resulting from the case-deletion algorithm after the deletion of r cases.

According to the Theorem above, we have $\bar{B}^{LC}/\bar{A}^{LC} \geq 0.58$. In fact, the competence of the case-base decreases from M^C to A^C after the deletion of cases chosen by the algorithm (see Figure 3). Thus, the most important is to establish a lower bound for the relation between A^C and M^C . If we now let $G = \bar{B}^{LC}/\bar{A}^{LC}$, we will obtain the following formula.

$$G = \frac{M^C - B^C}{M^C - A^C}$$

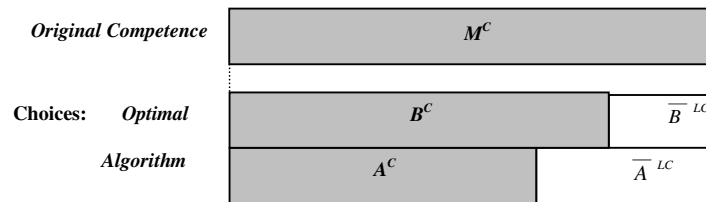


Fig. 3. The meaning of all competences involved in this paper.

Developing the equation and dividing both sides by M^C , and considering that B^C is a percentage of M^C , i.e., $B^C = z.M^C$, the formula becomes:

$$\frac{A^C}{M^C} = \frac{(G - 1 + z)}{G} \quad (6)$$

The formula above expresses the value obtained by the relation between M^C and A^C , with respect to G , that represents the relation between \bar{B}^{LC} and \bar{A}^{LC} , and the percentage $z = B^C/M^C$.

However, the percentage z must have a lower bound with respect to the percentage of cases that will be deleted in order to achieve a lower bound for A^C/M^C .

In some case-based systems, as in case-based planning, cases have different sizes and occupy different quantities of memory in their storage. However, in most domains, it is possible to calculate the maximal number of cases that will be deleted in order to release space in memory. The worst case is when a case with the maximal size is required to be stored and we must delete cases with minimal size. So, the maximal number of cases to be deleted is:

$$r = \left\lceil \frac{C_{\max}}{C_{\min}} \right\rceil \quad (7)$$

Where C_{max} is the maximal size that a case can have in a certain domain, and C_{min} is the minimal size. With the formula 7, we can define, in the worst-case scenario, the percentage of deleted cases from a case-base with m cases:

$$D = \frac{r}{m} \quad (8)$$

This percentage of deleted cases permits the determination of a lower bound for the optimal residual competence B^C when it is necessary to delete the maximal number of cases, r , from the case-base. First, we must prove the following Theorem:

Theorem 2 : A competence KC for optimal k choices from m cases with competence MC , with $k < m$, is $KC \geq k/m MC$.

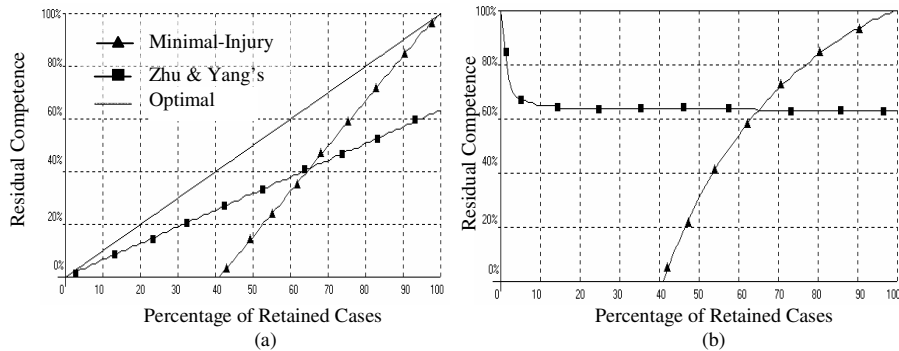


Fig. 4. Plotting of *Minimal-Injury* and *Zhu & Yang's* lower bounds with respect to total competence (a) and optimal lower bound (b).

The following Corollary provides a lower bound for z in terms of D :

Corollary: If $BC = z.MC$, and $D=r/m$, then $z \geq 1-D$.

This Corollary defines a lower bound for z relating to D . Now we have a lower bound for the optimal residual competence in the worst-case scenario, when r cases with minimal size must be deleted in order to release space for a case with maximal size. With this lower bound for z , and the results from Theorem 1, it is possible to find a generic lower bound for the residual competence from the algorithm, A^C , and the total competence M^C , for the worst-case scenario. It is obtained substituting the result of Corollary in (6):

$$\frac{A^C}{M^C} \geq \frac{(G - D)}{G} \quad (9)$$

where
$$G \geq \frac{1}{\left(\frac{r+1}{r}\right)^r - 1} \quad \text{and} \quad D = \frac{r}{m}.$$

This lower bound is similar to that one established by Zhu and Yang [12], however, it is dependent on D (percentage of deleted cases), and can achieve, for some values of D , better results than Zhu and Yang's lower bound. Figure 4 shows Zhu and Yang's and *Minimal-Injury* lower bounds for each percentage of retained cases ($1-D$).

By formula 9, the lower bound depends only on r and m . The r is constant in a specific domain, and m depends on the quantity of memory, where the worst-case scenario is m cases with C_{max} . Thus, for a certain domain, it is possible to define the quantity of memory that is necessary to achieve a specific lower bound, or the inverse, determine the lower bound for a specific quantity of memory.

It is important to note that the heuristic of choosing first the auxiliary cases for deletion does not alter the lower bound achieved above. This heuristic just improves the results of the algorithm.

4 Empirical Results

To validate our case-deletion policy, some empirical tests need to be performed. These tests were not performed in a specific domain framework, but use a case-base that is composed of by hypothetical cases that are randomly created by a case-base seeding system. Instead of considering the *representativeness assumption*, the competence is calculated by taking into consideration some hypothetical target problems that are distributed randomly by the seeding system.

With this generic framework, it is possible to create different case-bases with different concentrations. Concentration is calculated by the relation between the total competence and the sum of the competence of each case. It means that a high concentrated case-base has more target problems that can be solved by a great number of cases. Consequently, it also has more spanning and auxiliary cases compared to a low concentration case-base that has a great number of pivotal cases.

The tests were performed in opposite situations. We generated 10 different low concentration case-bases with 250 cases, and 10 high concentration case-bases with 250 cases. The presented results are the best representation of the average result for each concentration. These tests exploit the case-addition policy from [12], the compact competence case-base from [7] and the case-deletion from [9]. They are referred to as Max-Benef, RC-CNN and Type-based respectively. We named our approach in the experiments as Min-Injury. Although RC-CNN is an editing algorithm that preserves the competence of a case-base, we use it as an addition-based algorithm that chooses cases until a determined number of cases or amount of space is achieved.

It is important to note that each method has some incorporated heuristic in order to maximize their results. For example, Max-Benef chooses the case that occupies less

space among cases with the same benefit measure. RC-CNN and Type-based have similar heuristic.

4.1 Experiment 1

The first experiment takes into account case-bases with low concentration. It is concerned with the competence decreasing observation when it performs the deletion of determined number of cases and releases a specific quantity of memory space. The processing time of each method in this experiment takes less than 0.05 seconds.

The results show that the Min-Injury, Max-Benef and RC-CNN have similar performance when concentration is low, either deleting a determined number of cases (Figure 5a) or releasing space in memory (Figure 5b).

In our experiments, the Type-based had the best performance when the space to be released is bigger than 75% of the total memory. In other conditions, it proved to be the worst. Two reasons explain the performance of the Type-based approach. One is the heuristic that chooses the case with larger size among cases with the same type, and another is the low concentration of the case-base, which has a great number of pivotal cases. When 75% of memory space is released by the Type-based approach, almost all cases in a low concentration case-base are pivotal cases. In this circumstance, the Type-based approach performs choices that are near to optimal.

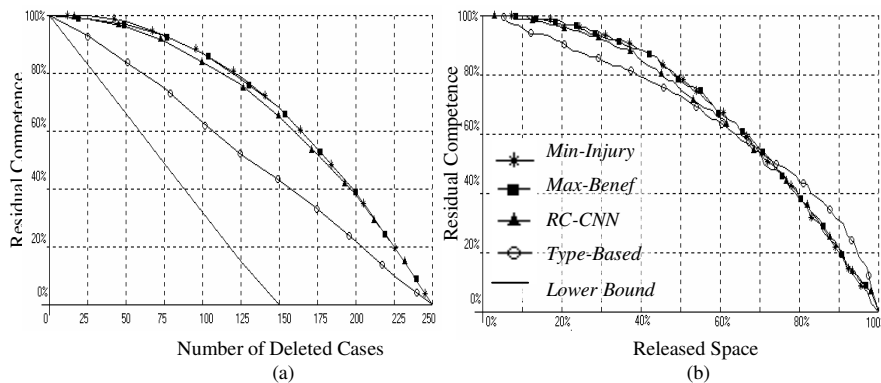


Fig. 5. Competence results after deleting a specific number of cases (a) and after releasing a percentage of memory space (b) in a case-base with low concentration.

The positive results obtained by Min-Injury lead us to conclude that it is a valid method to decrease the number of cases and to release space in memory, although it has a similar performance to other case-addition approaches.

4.2 Experiment 2

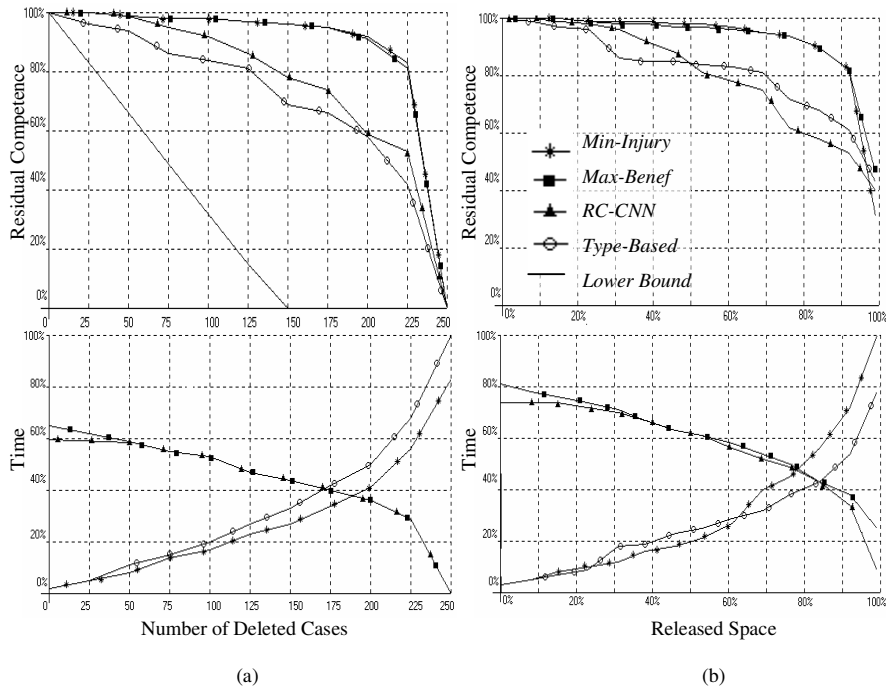
The second experiment is similar to the first experiment, but it focuses on high concentration case-bases. In high concentrations, the time of each method is different (Figure 6). The time is normalized according to the higher processing time presented in the test.

The results show that Min-Injury and Max-Benefit perform better than RC-CNN and Type-Based approach, either deleting a determined number of cases (Figure 6a) or releasing space in memory (Figure 6b). However, Min-Injury and Type-Based methods - case-deletion approaches - are more efficient in processing time than case-addition approaches when the purpose is to decrease the number of cases or to release less than 70% of the total space.

Each method has its own processing time *bottleneck*. For Min-Injury, it is the recalculation of the *injury*. For Max-Benefit, it is the recalculation of the benefit. For RC-CNN, it is the determination of the resulted case-base that solves cases in the training set, and for Type-based approach, it is the necessity to re-label the types of cases. The curves plotted in figure 6 are influenced by these *bottlenecks*.

This leads us to conclude that in spite of presenting the same results as Max-Benefit for residual competence, the Min-injury is faster when the purpose is to delete cases or release less than 70% of the space.

It certifies that Min-Injury is a useful approach, not only for releasing space to store a new case, but also for decreasing the number of cases in a case-base or even for advanced editing techniques.



(a)

(b)

Fig. 6. Competence and Time results after (a) deleting a determined number of cases; and (b) releasing a percentage of memory space. Both are performed in a case-base with high concentration

5 Conclusion

A competence-preserving deletion based algorithm is proposed to release space in case memory. It allows the user to define a lower bound for residual competence for each domain based on the available space to store cases. It is useful for the systems that have cases varying in size as case-based planning systems. It might also be applied in systems and as a policy that is concerned with reducing the size of case-base. The achieved results show that the *minimal-injury* policy can perform as well as *selective utilization* methods, but when the number of retained cases is more than half of total cases - mainly if it is more than 30% - the *minimal-injury* can be faster.

The *minimal-injury* approach maximizes its results by a heuristic method based on auxiliary cases that can be labeled off-line. A definition of such heuristic method in addition-based policy is heavily dependent on case-base features, and thus, it is difficult to be generally designed.

This paper concludes that case-deletion approach can be useful, and sometimes more efficient, than any *selective utilization* approach, to preserve the competence in some specific circumstances such as releasing memory space.

As our point of view, case-addition approaches and deletion-based algorithms are not concurrent, but complementary, and a suitable mix of both methods would probably improve the results.

The competence was chosen to be the main factor of our policy instead of adaptation costs in an efficient-preserving method, as [2], due to the fact that the dichotomy between competence and efficiency is not yet defined or proved to exist. We believe that if the competence measure were designed to take into consideration the adaptation costs, as in [12], the competence-preserving approach would approximate the efficiency-preserving property. However, it is not well known beyond some empirical tests, and probably many researches will arise in order to combine competence and efficient in one single approach.

References

1. Harinarayan, V., Rajaraman, A., Ullman J.D. Implementing Data Cubes Efficiently. In: Proceedings of ACM-SIGMOD. Computing Machinery, Montreal, Canada. (1996) 311-320.
2. Leake, D., Wilson, D. Remembering Why to Remember: Performance-Guided Case-Base Maintenance. In: Blanzieri, E., Portinale, L. (Eds.) Proceedings of the 5th European Workshop on Case-Based Reasoning (EWCBR2K). Lecture Notes in Artificial Intelligence, Vol 1898. Springer-Verlag (2000). 161-172.

3. Leake, D.B., Wilson, D.C. Categorizing Case-Base Maintenance: Dimensions and Directions. In: Smyth, B., Cunningham, P. (Eds.): 4th European Workshop on Case-Based Reasoning EWCBR-98. Lecture Notes in Artificial Intelligence, Vol 1488. Springer-Verlag (1998) 196-207.
4. Markovich, S., Scott, P. The Role of Forgetting in Learning. In: Proceedings of the Fifth International Conference on Machine Learning. Morgan Kaufmann Publishers. (1988) 459-465.
5. Minton, S. Qualitative Results Concerning the Utility of Explanation-based Learning. Artificial Intelligence, 42. AAAI Press. (1990) 363-391.
6. Reinartz, T., Iglezakis, I., Roth-Berghofer, T. On Quality Measures for Case-Base Maintenance. In: Blanzieri, E., Portinale, L (Eds.) Proceedings of the 5th European Workshop on Case-Based Reasoning (EWCBR2K). Lecture Notes in Artificial Intelligence, Vol 1898. Springer-Verlag (2000) 247-259.
7. Smyth, B., McKenna, E. Building Compact Competent Case-Bases. In: Althouff, K., Bergmann, R., Branting, K. (Eds.) Proceedings of the 3rd International Conference in Case-Based Reasoning. ICCBR'99. Lecture Notes in Artificial Intelligence, Vol 1650. Springer-Verlag (1999) 329-342.
8. Smyth, B., McKenna, E. Competence-Guided Case-Base Editing Techniques. In: Blanzieri, E., Portinale, L (Eds.) Proceedings of the 5th European Workshop on Case-Based Reasoning (EWCBR2K). Lecture Notes in Artificial Intelligence, Vol 1898. Springer-Verlag (2000) 186-197.
9. Smyth, B., Keane, M. Remembering to Forget: A Competence-preserving Case-deletion Policy for Case-based Reasoning Systems. In: Proceedings of the 14th International Joint Conference on Artificial Intelligence IJCAI'95. Morgan Kaufmann Publishers (1995) 377-382.
10. Tonidandel, F., Rillo, M. Handling Cases and the Coverage in a Limited Quantity of Memory for Case-Based Planning Systems. In: Sichman, J., Monard, C. (Eds). Proceedings of IBERAMIA/SBIA 2000. Lecture Notes in Artificial Intelligence, Vol 1952. Springer-Verlag (2000) 23-32.
11. Veloso, M. Planning and Learning by Analogical Reasoning. Lecture Notes in Artificial Intelligence, Vol 886. Springer-Verlag (1994).
12. Zhu J., Yang Q. Remembering to Add: Competence-preserving Case-Addition Policies for Case-Base Maintenance. In: Proceedings of the 16th International Joint Conference on Artificial Intelligence IJCAI'99. Morgan Kaufmann Publishers (1999).

Appendix: Proofs

Theorem 1: The optimal lost competence for r cases is at least 58% of the lost competence resulting from the case-deletion algorithm after the deletion of r cases.

Proof: Following the algorithm described in Figure 2, we have:

1. According to the definition of auxiliary cases, the deletion of an auxiliary case at step 2 does not affect the competence of the case-base.
2. According to step 3 of the algorithm, the proof is similar to that one for datacubes [1] and similar to that one for case-addition algorithm [12]. Due to space limitations, the proof is summarized and details can be obtained in [1]. Let m be the total number of cases, suppose that r cases are deleted and that $a_1 \leq a_2 \leq \dots \leq a_r$ are the injury of each case numbered in order of selection. Suppose that $b_1 \leq b_2 \leq \dots \leq b_r$ are the injury of each optimal case for deletion. Consider $x_{ij} =$

$|N(y_i) \cap N(x_j)|$, i.e., the number of problems solved by the optimal case y_i and by the case x_j chosen by the algorithm. Thus, the following formulas can be written:

$$\bar{A}^{LC} \geq \sum_{i=1}^r a_i; \quad \bar{B}^{LC} \geq \sum_{i=1}^r b_i \quad \text{and} \quad \sum_{i=1}^r x_{ij} \leq a_j.$$

Thus, for each choice:

- (1). For all i at the first choice: $b_i \geq a_1$
- (2). For all i at the second choice: $b_i + x_{i1} \geq a_2$
- ...
- (n). For all i at the n^{th} choice: $b_i + x_{i1} + x_{i2} + \dots + x_{i(n-1)} \geq a_n$

if we sum each above equations, the following inequalities are obtained:

- (1). $b_i \geq a_1 \Rightarrow \sum_{i=1}^r b_i \geq \sum_{i=1}^r a_1 \Rightarrow \bar{B}^{LC} \geq r \cdot a_1$
- (2). $b_i + x_{i1} \geq a_2 \Rightarrow \sum_{i=1}^r b_i + \sum_{i=1}^r x_{i1} \geq \sum_{i=1}^r a_2 \Rightarrow \bar{B}^{LC} \geq r \cdot a_2 - a_1$
- ...
- (r). $\bar{B}^{LC} \geq r \cdot a_r - a_1 - a_2 - \dots - a_{r-1}$

Observe that the step $i^{\text{th}} = r \cdot a_i - a_1 - a_2 - \dots - a_{i-1}$ and $(i+1)^{\text{th}} = r \cdot a_{i+1} - a_1 - a_2 - \dots - a_i$. Thus, the difference between i^{th} and $(i+1)^{\text{th}}$ is: $r \cdot a_{i+1} - (r+1)a_i$. Since this difference must be 0, the equality becomes: $a_i = \frac{r}{r+1} a_{i+1}$. Thus, $\bar{A}^{LC} =$

$$\sum_{i=0}^{r-1} \left(\frac{r}{r+1} \right)^i \cdot a_r \quad \text{and}$$

$$\bar{B}^{LC} \geq r \cdot \left(\frac{r}{r+1} \right)^{r-1} \cdot a_r. \quad \text{The relation becomes:} \quad \frac{\bar{B}^{LC}}{\bar{A}^{LC}} \geq \frac{1}{\left(\frac{r+1}{r} \right)^r - 1}.$$

$$\text{With } r \rightarrow \infty, \text{ the relation becomes:} \quad \frac{\bar{B}^{LC}}{\bar{A}^{LC}} \geq \frac{1}{e - 1} = 0.58. \quad \blacksquare$$

Theorem 2 : A competence KC for optimal k choices from m cases with competence MC, with $k < m$, is $KC \geq k/m$ MC.

Proof: By Lemma, the total competence is less or equal to the summation of individual case competence. The maximal competence is obtained when the cases are disjointed. Thus, the proof is an induction in k. Due to space, it is summarized as follow:

- $K=0$, $KC=0$; It results in $KC \geq k/m$ MC;
- $K=1$, if each case does not share any competence with another case and has a competence $KC < 1/m$ MC, the summation of each case competence does not

result in total competence MC . Therefore, one case, at least, must have $KC \geq 1/m MC$, that would be the optimal choice for $k=1$. It results in $KC \geq k/m MC$;

- $K=2$, considering that the cases are disjointed, any group of 2 cases will have $KC < 2/m MC$ with at least 1 case y with $KC \geq 1/m MC$; otherwise it would be the step $K=1$. The case y would have $YC = 1/m MC + \epsilon$ and any other case would have the competence less than $1/m MC - \epsilon$. The summation of the competence would be $1/m MC + \epsilon + (m-1).(1/m MC - \epsilon)$, that is equal to $MC + 2\epsilon - m.\epsilon$. The initial condition says that $k < m$, thus, $m > 2$ and the summation does not result in MC . Therefore, one group of 2 cases, at least, must have $KC \geq 2/m MC$, that would be the optimal choice for $k=2$. It results in $KC \geq k/m MC$;
- $K=n$, considering that the cases are disjointed, any group of n cases will have $KC < n/m MC$ with at least a group y of $n-1$ cases with $KC \geq (n-1)/m MC$; otherwise it would be the step $K=n-1$. The group y would have $YC = (n-1)/m MC + \epsilon$ and any other case would have the competence less than $(n-1)/m MC - \epsilon$. The summation of the competence would be $(n-1)/m MC + \epsilon + (m-n).(n-1)/m MC - \epsilon$, that is equal to $MC + n\epsilon - m.\epsilon$. As $m > n$, the summation does not result in MC . Therefore, one group of n cases, at least, must have $KC \geq n/m MC$, that would be the optimal choice for $k=n$. It results in $KC \geq k/m MC$.

Thus, for k from 0 to $n < m$ the competence is $KC \geq k/m MC$. ■

Corollary: If $BC = z.MC$, and $D=r/m$, then $z \geq 1-D$.

Proof: If r cases are deleted, then k cases will be left in the case-base, where $k=(1-D)m$. By Theorem 2, the optimal choices of k cases have the competence of $KC \geq k/m MC$. As $KC = BC$ by definition, we can conclude that $z \geq k/m = 1-D$. ■