

The Utility Problem for Lazy Learners - Towards a Non-eager Approach

Tor Gunnar Houeland and Agnar Aamodt

Department of Computer and Information Science,
Norwegian University of Science and Technology,
NO-7491 Trondheim, Norway
{houeland,agnar}@idi.ntnu.no

Abstract. The utility problem occurs when the performance of learning systems degrade instead of improve when additional knowledge is added. In lazy learners this degradation is seen as the increasing time it takes to search through this additional knowledge, which for a sufficiently large case base will eventually outweigh any gains from having added the knowledge. The two primary approaches to handling the utility problem are through efficient indexing and by reducing the number of cases during case base maintenance. We show that for many types of practical case based reasoning systems, the encountered case base sizes do not cause retrieval efficiency to degrade to the extent that it becomes a problem. We also show how complicated case base maintenance solutions intended to address the utility problem can actually decrease the combined system efficiency.

1 Introduction

A concern for case-based reasoning (CBR) systems that are deployed and will keep running for many years is how the system will change over time. The capability for learning is an important aspect of many such systems, but by its very nature the act of learning will change the system from its current state to something that is partially unknown. There will normally be a desirable improvement from learning, but its effects may also include unwanted changes.

One of these changes is that as the system's knowledge increases, the space needed to store the knowledge and the time it takes to process it also increases. The storage space and time taken to process the knowledge will increase without bounds, and eventually go far beyond the space and time taken by the original system.

Because of this behavior, there will always be some theoretical point where the total performance of the system is degraded by adding additional knowledge. Many different methods have been suggested to address this problem, which is often included under the wider umbrella of case base maintenance.

The maintenance methods used to address this problem can be split in two: maintaining the case base indexes, and maintaining the case base contents.

Indexing methods work by quickly identifying the relevant parts of the knowledge base, which can allow a system to examine only a small fraction of its available knowledge.

Methods for maintaining case base contents aim to reduce the size of the system's case base, making it both faster to examine since there is less knowledge, and reducing the storage space needed to store the data.

The *utility problem* in learning systems occurs when knowledge learned in an attempt to improve a system's performance degrades it instead [17,3]. This is most common in speed-up learning systems, where the system's knowledge is used to reduce the amount of reasoning required to solve a problem.

For pure speed-up learners it is assumed that there is already a slower method available for finding an acceptable solution to the problem. From a simplified perspective, cases in a CBR system may be viewed as a form of speed-up knowledge, where storing, retrieving, and adapting cases provides for more efficient problem solving than first-principles or model-based methods [15,9]. The goal is to produce acceptable results more quickly, and hence the time taken to perform the system's reasoning is of primary concern.

Case-based reasoning is also known as a *lazy* approach to learning and problem solving. The very essence of lazy learning is that choices regarding the solution of a problem will be postponed as long as possible, until the problem query is posed and as much information as possible is available.

Building index structures and deleting cases from the case base are both eager methods, and hence somewhat counter-intuitive to the CBR idea. Indexing and deletion reduce the amount of knowledge available, without knowing whether that information could have been useful for solving a future problem. Hence, indexing and deletion methods should only be used when they are really needed. In the work reported here we explore the hypothesis that for a wide range of CBR application systems, addressing real world problems, they may not be needed.

This work is situated within our research on a new architecture for meta-level reasoning and introspective learning [10]. A less eager approach to indexing and case deletion, if feasible, will allow more freedom to the meta-level reasoner.

This paper examines the utility problem in CBR as it applies to most of the CBR systems we have built and are building today, with case bases of reasonable sizes. Based on existing literature and several example utility models for case-based reasoners, we show that neither indexing nor case deletion policies are necessary for a wide range of CBR systems, and in fact can be detrimental to a CBR system's overall goal.

In section 2 the background of the utility problem is summarized, and some earlier research results relevant to our work is discussed. This is followed in section 3 by an analysis of the utility concept and a comparison of three different speed-up learner scenarios. Section 4 discusses the use of indexing strategies for speeding up retrieval. Section 5 discusses the benefit of limiting case base size and shows how the cost of advanced maintenance methods may negate the benefits of a reduced case base through an illustrative experiment. Concluding remarks end the paper.

2 Background and Related Research

A substantial amount of research has addressed the utility problem in CBR [19,13]. Over the past few years there has been a broad range of research addressing specific issues of case deletion, addition and efficient indexing [20,25,2]. Wilson and Leake [14] present a thorough examination of the dimensions of maintenance strategies and a survey of maintenance research in terms of those dimensions.

The utility problem is also referred to as the “swamping problem”. In their seminal paper on the utility problem in CBR, however, Francis and Ram [9] refer to swamping as one of three types of utility problems in CBR. Swamping is the phenomenon that a single unit of knowledge added to the knowledge base - i.e. a case added to the case base - improves problem solving speed at the individual level, because a more similar case to a query may be found, while the performance over the knowledge base as a whole is degraded due to increased retrieval and matching overhead. Swamping is also referred to as the “core” utility problem, which is probably why the two have become synonyms. The other types of utility problems listed are the “expensive chunks problem” and the “search-space utility problem”. The first refers to the problem of performance degradation at the level of individual knowledge units, because of the matching cost of a single unit (a macro operator, for example). The second refers to degradation due to increased complexity of search control knowledge, of particular relevance to the learning of meta-level knowledge. Although all three have relevance for CBR systems, the focus in this paper is on the swamping problem.

The processing power available for modern CPUs continues to increase, continually reducing the problems associated with large amounts of data, and allowing large case bases to be handled which would have been considered impossible for a reasonable budget 10 years ago. However, this is typically only true for polynomial-time algorithms, and especially for algorithms that run in (sub-)linear time. Other algorithms that have an exponential running time are unlikely to ever be practical for large inputs, such as many graph-matching algorithms. This means that the “expensive chunks problem” is unlikely to be greatly affected simply by advances in computer hardware, since they are NP-hard in the worst case [23]. On the other hand these advances affect the degradations experienced due to the swamping problem, since algorithms for searching the knowledge base are typically either $O(N)$ or $O(\log N)$.

The main cause of the swamping problem is that retrieval time will increase with a growing case base while adaptation time will decrease. The latter is due to a smaller distance between a query case and the best matching case on average. As the case base grows retrieval time is likely to dominate, however, which leads to a logarithmic or higher increase in processing time. For a speed-up learner this increase may negate the efficiency gains during adaptation and eventually even cause the system to be slower than the underlying “slow” solver. The speed increase has been reported as being substantial in some systems where this logarithmic increase has no significance, for example a thousand-fold increase in CASEY [12]. Other machine learning systems have reported more modest

figures, such as a factor up to six in SOAR and ten in PRODIGY/EBL [9]. Systems with these speed-up figures that are left running and collecting experience for a long time will gradually slow down, and eventually be slower than the unassisted “slow” solver [9].

The trade-off is perhaps most clearly illustrated and explored for some types of control rule learning (CRL) systems, where every individual control rule is guaranteed to have a positive utility (improve performance) but, in concert, also have a negative utility (degrade performance) [9,17].

Francis and Ram [8] describe a framework for modeling CRL and CBR systems, and for analyzing the utility problem for these reasoning models. The authors identified that the retrieval costs for CBR increase without bound, and that in the limit, CBR systems will get swamped and the cost of retrieval will outweigh the benefits of case adaptation. The authors conclude that CBR is nevertheless relatively resistant to the utility problem, compared to CRL systems, because the cases have the potential to greatly reduce the amount of problem solving needed, and that the cost of retrieval is amortized across many adaptation steps.

Smyth and Cunningham [19] examine the utility problem in CBR through experimenting with a path finding system that combines Dijkstra’s algorithm with CBR for speed-up. They show how case-base size, coverage and solution quality affect the utility. The authors find that varying these characteristics significantly alters how the system responds to different case base sizes. This indicates that the need for case deletion and indexing is strongly related to requirements for solution quality and retrieval time. We will discuss this issue later in the paper.

A proposed policy for case deletion with minimal effects on case base competence was presented by Smyth and Keane [20]. A footprint-driven method that also accounted for utility gave the best test results. An alternative method was proposed by Zhu and Yang [25] with the emphasis on careful addition of new cases rather than on deleting old ones. Their method performed better than the footprint deletion method, under some conditions.

A deliberate case addition process may be viewed as a form of case deletion as well (i.e. by not adding cases that might otherwise have ended up in the case base). From the perspective of more or less eager case base maintenance methods, assuming an existing case base and only incremental updates, a considerate case addition policy will generally be a more lazy approach than a deletion approach. An even lazier approach is of course to keep all the cases, and rely on the retrieval and adaptation methods to do the “deletion” on the fly.

One solution to the indexing problem is to apply suitable methods for refining indexing features and matching weights. Jarmulak et al. [11] use genetic algorithms to refine indexing features and matching weights. Another approach is to view this problem from a meta level perspective, and use introspective learning techniques to handle the refinement of indexes, triggered by retrieval failures [7,4].

We are developing an introspective architecture for lazy meta-level reasoning characterized by creating and combining multiple components to perform the system's reasoning processes [10]. Each component represents part of a reasoning method and its parameters using a uniform interface, which can be used and modified by the meta-level reasoner. This will enable selecting which reasoning methods to use after a description of the problem to be solved is known.

Although it is known that laziness and indexing strategies impact each other [1], we have not come across work which expressly views indexing also from a maximally lazy perspective. This means to avoid building such indexes at all for the purposes of improving search efficiency, as a way to avoid committing to eager indexing decisions. Indexes may still be built, but in order to improve matching quality only. A typical example is a structure of abstract indexes, which interpret lower-level data in the input cases in order to achieve an improved understanding of the case information and hence more accurate solutions.

Watson [24] discusses case base maintenance for *Cool Air*, a commercially fielded CBR system that provides support for HVAC engineers. The system retrieves cases for similar previous installations, and Watson explains the case-base maintenance required for the system, most notably the removal of redundant cases from a rapidly growing case base. By the nature of the application domain, installing the system in different locations means that the same product will operate under several similar conditions, and result in very similar cases being created. In their client-server design the server selects a small set of cases to be sent to the client, and the client then uses the engineers' custom-tailored similarity measure to rank them. The problem with this design is that many redundant cases are sent to the client, and it was decided to remove the redundant cases from the case base. Although primarily motivated by other purposes than the utility problem, this type of case redundancy avoidance in client-server architectures seems to be generally useful.

3 Describing and Analyzing Utility

In general, the utility of a reasoning system can be expressed as the benefit it brings, minus the costs associated with the system. The benefits are typically achieved over time while the system is in operation, while a large part of the costs of the system are up-front, such as gathering expert knowledge, developing the system and integrating it with the organization that will be using it. Another large source of costs is the continued maintenance of the system, which is often overlooked but should be included when the system is initially planned [24].

The benefit of a generated solution can be measured as its usefulness for addressing the task at hand, which is primarily characterized by the solution accuracy and solution time. When considered in this manner, the time taken to solve a problem can be naturally expressed as a lessened benefit. Then the direct costs associated with the problem solving are related to the resources spent computing them, which are very small for typical systems.

For clarity, we define our use of the terms as follows:

General System Utility: GSU The combined benefit of the reasoning system, minus the associated costs. This consists of the solution usefulness for the solutions generated by the system, the usability of the system for human operators, and the costs associated with developing, running and maintaining the system.

Solution Usefulness: SU The benefit of a generated solution, estimated as a function of solution accuracy, solution time and resource costs, not including human factors.

Solution Accuracy: SA The accuracy of a generated solution, which depends on both the case base and the methods used by the system.

Solution Time: ST The time it takes the system to solve a problem.

Resource Cost: RC The cost of solving a problem. This is primarily the time spent operating the systems, but also includes hardware costs that can potentially be significant for long computations in large systems.

Using the same approach as Smyth and Cunningham [19] for analyzing these concerns, we assume that the solution accuracy increases with a larger case base, and that the solution time is divided into two parts: retrieval time, which increases with a larger case base, and adaptation time, which decreases with a larger case base (or stays the same). By noting that the retrieval time increases with the size of the case base, which is unbounded [17], that the retrieval time is similarly unbounded for any retrieval approach that can potentially reach all the cases, and that the reduction in adaptation time is bounded (since it can never be faster than 0 time units), we see that there will be some point where adding further cases to the case base will slow down the total solution time.

The utility problem is most easily analyzed for speed-up learners, where the solution time is of primary importance. We explore three different scenarios for speed-up learners with different time complexities, and examine the different amounts of utility degradation experienced by modeling the solution time **ST** as a function of case base size N . We use a simplified model of a speed-up learner, where the system will always produce the same correct answer, and the only criterion for the solution's utility will be its solution time. We model the solution usefulness $\mathbf{SU} = 1/\mathbf{ST}$ for such systems, ignoring the solution accuracy.

As reported by Smyth and Cunningham [19], when increasing the size of the case base, the solution time for a case-based speed-up learner will typically consist of an initial rapid improvement, followed by a continuing degradation as the retrieval part begins to dominate the total time spent to solve the problem. For a typical speedup-learner, the total solution time will initially be approximately monotonically decreasing, followed by an approximate monotonic increase, and the optimal solution time and preferred case base size will be where the rate of increase in retrieval time matches the rate of decrease in adaptation time. The exact behavior of the total solution time and the order of magnitude of this preferred case base size depends greatly on the algorithmic complexity classes of the algorithms used, and in general there is no guarantee that the solution time will follow this pattern at all. The exact characteristics also depend on how the case base content is created

and maintained, since a maintained case base can have different case distributions and behave quite differently than an approach retaining all cases.

Fig. 1 shows the solution time for an idealized speed-up learner, where retrieval is a linear search through the case base, adaptation time is proportional to the distance to the retrieved case, and there is no overhead: $\mathbf{ST} = N/5 + 100/N$. In this situation the efficiency of the system initially improves quickly, and then starts degrading slowly as the increased time to perform retrieval eventually becomes greater than the time saved during adaptation. In systems with retrieval and adaptation algorithms displaying this kind of behavior, the solutions will be generated most quickly when retrieval and adaptation times are approximately equal, since that coincides with their derivatives having the same magnitude and opposite signs. Smyth and Cunningham [19] report very similar results to this speed-up learner scenario from experimenting with the PathFinder system.

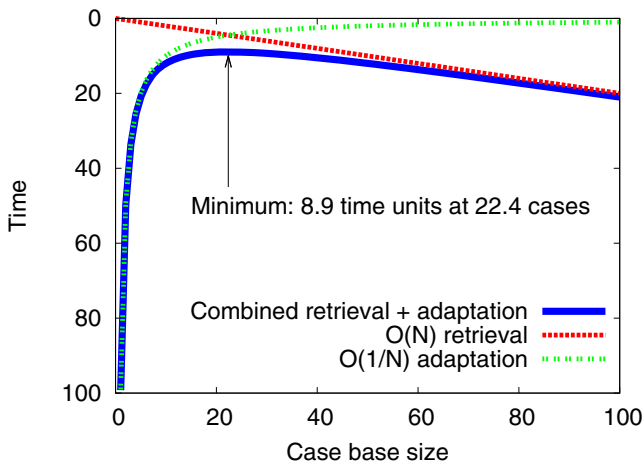


Fig. 1. Retrieval, adaptation and combined solution time compared to case base size for a speed-up learner using an $O(N)$ retrieval algorithm and a $O(1/N)$ adaptation algorithm

Fig. 2 shows the solution time for similar speed-up learning systems, but using different algorithms. The graph on the left uses an indexing scheme for retrieval that causes the retrieval step to run in $O(\log N)$ time, and with a comparatively larger constant factor (5 vs 0.2) than the previous example: $\mathbf{ST} = 5 * \ln(N) + 100/N$. With this change to the retrieval function, the slowdown associated with adding more cases to the case base happens very slowly, and the total solution time for the full case base is just 20% slower than the minimum solution time, even though the full case base is 5 times larger. The graph on the right shows a much more drastic increase in solution time. A more complicated $O(N \log N)$ case retrieval algorithm is shown that compares the retrieved cases against each other, and case adaptation has a significant overhead of 20 time units: $\mathbf{ST} = N * \ln(N) + 100/N + 20$. In this situation the combined solution

time increases quickly as more cases are added beyond the optimal amount. For domains where this type of algorithm is desirable, a similar increase in solution quality would be expected, otherwise the case base should be kept relatively small through aggressive case base content maintenance. Due to the very limited number of cases the system can handle before slowing down, these latter types of algorithms appear to be a poor choice for pure speed-up learners, although the constant factors could potentially be of very different magnitudes for some domains. These alternative combinations scale very differently, and illustrate the importance of examining the algorithms used when analyzing the effect of larger case bases.

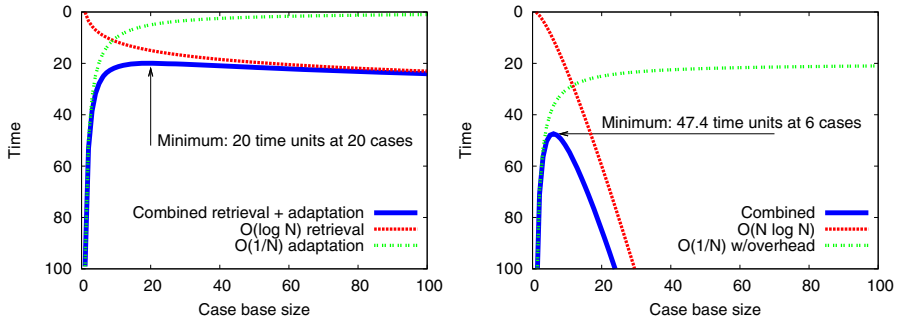


Fig. 2. Retrieval, adaptation and combined solution time compared to case base size for two other speed-up learner scenarios: On the left, using an $O(\log N)$ retrieval algorithm and a $O(1/N)$ adaptation algorithm. On the right, using an $O(N \log N)$ retrieval algorithm and a $O(1/N)$ adaptation algorithm with overhead. The difference in shapes illustrate the importance of considering the specific algorithms used when examining the utility problem.

For general case-based reasoners, the utility function becomes much more complicated. When still examining only the usefulness of solving one problem while the system is running, and just moving away from the simplified speed-up learner model, we need to also include the accuracy of the solution in our evaluations. The impact the accuracy has on the usefulness of the system will vary greatly based on the domain and the specifics of the application.

Smyth and Cunningham [19] report empirical results from the PathFinder system, where at one point the quality of solutions increased from 94% to 96%, while the solution time increased by 50%. Whether such a trade-off is considered beneficial or not depends on the application and the initial starting values for the retrieval accuracy and solution time. For a speed-up learner this might be unacceptable, while for many other applications a 33% reduction in errors (from 6% to 4%) at the expense of waiting longer or using two computers instead of one would be a great improvement. As in the fielded *Cool Air* system, the solution time might simply be considered acceptable and not be a problem that has to be addressed at all, and then a larger case-base might be purely beneficial without encountering this trade-off.

4 Indexing vs. No Indexing

There are many possible indexing strategies for speeding up retrieval, and their effects are highly dependent on the domain and the similarity measures used. Selecting a good indexing strategy can often require significant expert knowledge, although various automated methods exist [11,7,4] for refining the indexing strategy. This can be particularly helpful for index maintenance, since an appropriately chosen method can potentially handle most indexing maintenance operations without manual intervention.

However, from a lazy learning perspective, an indexing structure that allows the retrieval method to only consider a subset of the case base is an eager optimization, which is made before all the potentially useful information about the target problem is known, and is therefore not always appropriate.

In systems that handle thousands of cases or less, the processing time is not necessarily a critical factor, and might very well increase slower than the increase in processing power available over time, i.e. the solution usefulness **SU** is primarily a function of the solution accuracy **SA**. This is particularly true for knowledge-intensive CBR, where less than a hundred cases is common (but the time required to perform individual similarity measurements is often extensive).

By matching with every case in the case base, the retrieval method is guaranteed not to miss any cases in the case base, and without needing to regularly maintain the indexing structures.

Unlike pure speed-up learners, producing results as quickly as possible is rarely the main concern for fielded CBR systems. The cost of developing and maintaining a system is usually much larger than the cost of using and maintaining the hardware it runs on, and the direct resource cost **RC** can sometimes even be considered negligible. CBR systems with extensive reasoning also do not usually act as speed-up learners, since they can actually produce better solutions with a larger case base. For such systems the utility problem is a trade-off between solution quality and the efficiency degradation associated with a large case base [19].

As an alternative to purely eager indexing, footprint-based retrieval [21] allows for a kind of mix of indexing and similarity measurements. The indexing is eagerly pre-generated based on competence evaluations performed before the input problem is known. During retrieval, the index is used to quickly identify what is believed to be the most relevant knowledge items, which are then evaluated lazily with full similarity-based knowledge. Although the retrieval is less efficient than purely eager methods, this partially lazy approach can produce good results for some domains.

For the commercially fielded *Cool Air* [24] CBR system, processing time was much cheaper for the company than consultancy time for developing the system. The efficiency slow-down associated with an increasing case base did not become a problem, even though the case base doubled over two years.

In another commercially fielded system, the DrillEdge system for fault prediction in oil well drilling [22], case retrieval is an automatic process triggered by a pattern in the continuous stream of drilling data. The cases are indexed by abstract features derived from the numerical drilling data stream. This is done

in order to improve the matching process for retrieval of the most relevant cases. The indexes are not used to improve retrieval efficiency - the case base is always searched sequentially. As long as the number of cases is in the range of hundreds, this is not regarded as a performance problem.

For some applications, like the one just mentioned, solution quality is of utmost importance. In oil well drilling, the costs of time lost due to unwanted events can be huge, since drilling operations typically cost around 200 000 USD per day [18]. In this case the value of the positive utility associated with higher quality solutions is of a different order of magnitude than typical costs of negative utility caused by decreased efficiency. For the knowledge-intensive oil well drilling system, the main cost of a large case base is the amount of expert knowledge required, not the computer systems it runs on.

As an alternative to performing eager indexing at all, a two-step approach [6] to case retrieval has often been employed for systems with expensive retrieval operations, e.g. for knowledge-intensive CBR systems. This consists of first using a fast and resource-efficient scan through the case base to identify relevant knowledge, and then performing more advanced (and comparatively slow) reasoning for this restricted set of cases. This is conceptually very similar to indexing, but is done using a lazy approach, entirely after the input problem query is known.

In this way, there is no need to update indexing structures, and more powerful methods can be performed for identifying relevant knowledge when you already know the problem to be solved. Similarity assessment is usually very important for CBR systems, because there is often no easy way to model the structure of the entire problem space, and there may even be no expert knowledge that directly applies to all problem instances in general. Using similarity measurements to locally identify relevant knowledge for a specific problem is thus likely to produce better results than pre-generated structures.

To perform large numbers of similarity assessments quickly, it might be necessary to increase the amount of computational resources available by examining the cases in the case base in parallel. Many modern distributed computing frameworks available for processing very large data sets in parallel are based around ideas similar to the MapReduce [5] algorithm. MapReduce works by first chopping up a problem into many parts, then distributes each of these parts across a cluster of computers and each node processes only a subset of the problems. The answers are then returned to a master node, which combines them to create a final answer for the entire problem. This is very similar to parallel case retrieval, where each case is assigned a similarity score and then ranked at the end.

While this form of case evaluation producing independent results for every query-to-case comparison does not let us express the most general forms of case retrieval, they are sufficient for most systems that are used in practice. The kind of similarity assessment methods supported by this approach are also typically more flexible than those supported by common indexing schemes. Avoiding the need for additional expert knowledge that is often required to create a good indexing solution is another potential benefit of this approach.

For commercial applications, these kinds of large parallel processing frameworks are typically used to process terabytes or petabytes of data, and provide a possible means to perform full sequential evaluations for the complete case base during retrieval even for very large case bases.

5 Case Base Maintenance

Case-based reasoning system maintenance is important and can involve processes that modify all parts of the system, including updates to each of the knowledge containers. Most often this includes reducing the number of cases in the case base, which is primarily useful for two purposes:

- Reducing the size of the case base used by retrieval methods, which can make retrieval faster.
- Reducing the space required for storing the case base.

Various case base content maintenance algorithms exist for reducing the size of the case base, while optimizing the remaining cases according to some criteria. A fast and simple content maintenance strategy is to delete cases at random, which has been reported to produce good results [16]. Since the case base essentially becomes a random subset of all encountered cases, or effectively just a smaller case base, this strategy also has the added benefit of maintaining the same case distribution as the encountered cases, on average. Other approaches for content maintenance usually examine the relations between cases in the case base, and e.g. attempt to maximize the coverage of the remaining cases in the reduced case base through adding, deleting or combining cases [20].

We conducted a set of experiments to compare these two approaches, using a random set of cases versus the coverage-based case addition algorithm proposed by Zhu and Yang [25] as the content maintenance strategy. The results shown in figs. 3-5 are the average from running each test 10 times. Cases were described by 5 features, each with values ranging from 0 to 1, and new cases were picked uniformly from this 5-dimensional space. Euclidean distance was used as the basis for the similarity measure, and a case was considered to be solvable by another case for the purpose of competence evaluation if the distance between the two cases was less than 0.25. We used the same similarity measure to estimate the solution accuracy **SA** on the basis of the distance between the retrieved case and the query, which is optimistic and more advantageous for the maintained case base strategy than a real world scenario, since the competence evaluations will be flawless. Thus using larger case bases can be expected to usually be at least as good compared to this kind of computationally expensive content maintenance strategy for real-world systems as in the experiments.

Fig. 3 shows the estimated coverage and error for an optimized case base of size N compared to a case base consisting of N random cases. The case base generated by the case addition algorithm has higher resulting coverage (measured as the covered proportion of new queries randomly generated from the underlying domain, which competence-driven maintenance strategies seek

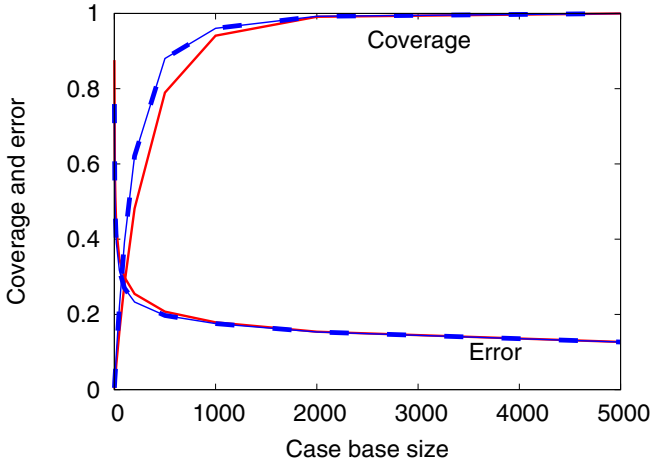


Fig. 3. Coverage and error comparing retaining the full case base of size N (*straight lines*) and using a content maintenance strategy (*dotted lines*) to create a case base of size N based on a larger initial set of 5000 cases. Higher coverage and lower error is better. The time required to perform the maintenance is not considered. In this setting the maintenance strategy outperforms retaining all cases, with higher coverage and slightly lower error.

to optimize), and lower error (measured as the average distance from the best retrieved case to randomly generated new queries). Only the sizes of the case bases are considered, and the computations required to perform the maintenance operations are ignored. Approximately this situation can occur when there are established procedures to run case base maintenance while the system is not being used, e.g. at night, during weekends or during vacations.

However, the computational costs of running case base content reduction algorithms can be extensive. Figs. 4 and 5 show the coverage and error rates for the same two case base content maintenance strategies, but compared according to the time required to perform both maintenance and retrieval. This was examined by running experiments for many different combinations of initial and reduced case base sizes, and choosing the Pareto efficient combinations that gave better results than any faster combinations. The size of the resulting reduced case base size used for retrievals is included in the figures. For each data point the case base maintenance was run only once, and its potentially costly computation was amortized over a large number of retrievals. However, this maintenance cost can still be very high, depending on the number of retrievals performed compared to maintenance operations.

The examples shown in the figures consists of an up-front case maintenance step followed by 1000 and 10000 retrievals respectively (chosen as examples of large numbers of retrievals, since more retrievals favors the maintenance strategy), and shows the combined time for these operations. Even with this relatively large number of retrievals, the simpler strategy of retaining all cases

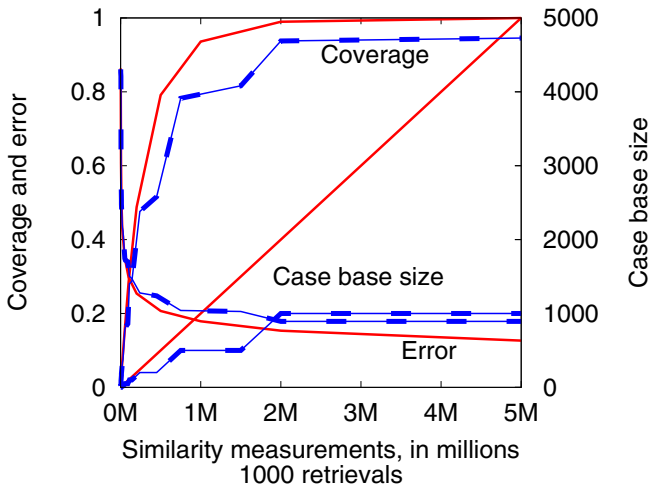


Fig. 4. Coverage and error shown according to the amount of computation required (measured as number of similarity measurements), when retaining the full case base (*straight lines*) and using a content maintenance strategy (*dotted lines*). For the situation with 1000 retrievals, the larger case bases supported by not having a high maintenance cost means that the strategy of retaining all cases performs better, with higher coverage and lower error.

generally performs as well or better than the content maintenance strategy, due to supporting larger case bases in the same time frame. This means that using a maintenance strategy to reduce the case base size for efficiency reasons may sometimes be counter-productive, in addition to size reduction being an eager strategy that limits the potential options available for further problem solving.

The other aspect of reducing the number of cases in the case base is the reduced storage capacity required to hold the case base. Current computer systems intended for personal use can store hundreds of gigabytes of data, which is much much larger than many typical CBR application case bases. Maintaining the set of cases exposed to the retrieval method can be a very useful approach for some applications, but the case base used for retrieval at any given moment does not have to be the full set of cases archived by the system.

Based on this observation, we conclude that many practical CBR system can instead flag the cases as no longer being active and store them in another location that is not searched by the retrieval methods, since conserving disk space is not required for systems that do not generate vast amounts of data. In these situations the archival storage can be done at negligible cost, and provide the advantage that deletions are no longer completely irreversible.

During later system maintenance some time in the future, the reason for the original deletion may no longer be relevant or the algorithms used by the system may have changed, and in such cases it would be beneficial to be able to undo such eager deletion optimizations, in the spirit of lazy learning.

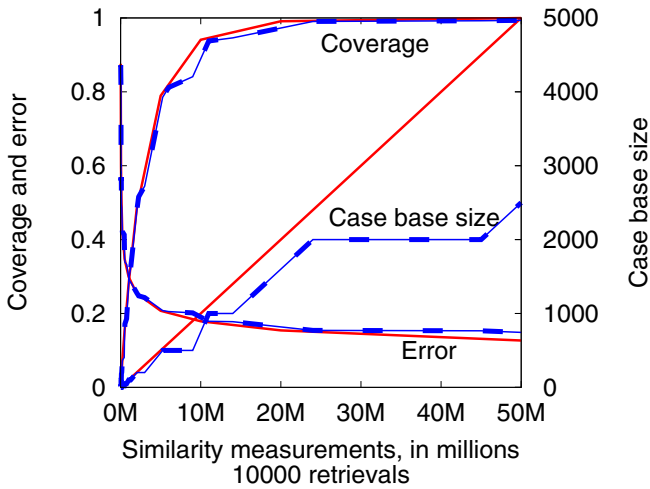


Fig. 5. Even when performing 10000 retrievals, the strategy of retaining all cases generally performs slightly better, with higher coverage and lower error

6 Conclusions

In this paper we have examined the utility problem from a lazy learning perspective, as it applies to speed-up learners and general case-based reasoners. The two primary approaches to addressing the utility problem are through indexing and by reducing the size of the case base itself during case base maintenance.

These approaches are eager compared to the lazy core CBR process, and we have shown how many practical CBR systems do not require the use of these eager optimizations and can be limited by committing to decisions prematurely.

References

1. Aha, D.W.: The omnipresence of case-based reasoning in science and application. *Knowledge-Based Systems* 11, 261–273 (1998)
2. Bergmann, R., Richter, M., Schmitt, S., Stahl, A., Vollrath, I.: Utility-oriented matching: A new research direction for case-based reasoning. In: 9th German Workshop on Case-Based Reasoning, Shaker Verlag, Aachen (2001)
3. Chaudhry, A., Holder, L.B.: An empirical approach to solving the general utility problem in speedup learning. In: IEA/AIE '94: 7th international conference on Industrial and engineering applications of artificial intelligence and expert systems, USA, pp. 149–158. Gordon and Breach Science Publishers Inc., Newark (1994)
4. Cox, M.T.: Multistrategy learning with introspective meta-explanations. In: *Machine Learning: Ninth International Conference*, pp. 123–128. Morgan Kaufmann, San Francisco (1992)
5. Dean, J., Ghemawat, S.: Mapreduce: simplified data processing on large clusters. In: *OSDI 2004: 6th conference on Symposium on Operating Systems Design & Implementation*, pp. 137–150. USENIX Association, Berkeley (2004)
6. Forbus, K.D., Gentner, D., Law, K.: MAC/FAC: A Model of Similarity-Based Retrieval. *Cognitive Science* 19(2), 141–205 (1995)

7. Fox, S., Leake, D.B.: Combining case-based planning and introspective reasoning. In: 6th Midwest Artificial Intelligence and Cognitive Science Society Conference, pp. 95–103 (1995)
8. Francis, A., Ram, A.: A comparative utility analysis of case-based reasoning and control-rule learning systems. In: Lavrač, N., Wrobel, S. (eds.) ECML 1995. LNCS, vol. 912, pp. 138–150. Springer, Heidelberg (1995)
9. Francis, A.G., Ram, A.: The utility problem in case based reasoning (1993)
10. Houeland, T.G., Aamodt, A.: Towards an introspective architecture for meta-level reasoning in clinical decision support systems. In: 7th Workshop on CBR in the Health Sciences, ICCBR 2009 (2009)
11. Jarmulak, J., Craw, S., Rowe, R.: Genetic Algorithms to Optimise CBR Retrieval. In: Blanzieri, E., Portinale, L. (eds.) EWCBR 2000. LNCS (LNAI), vol. 1898, pp. 136–147. Springer, Heidelberg (2000)
12. Koton, P.A.: A method for improving the efficiency of model-based reasoning systems. *Applied Artificial Intelligence* 3(2-3), 357–366 (1989)
13. Leake, D.B., Smyth, B., Wilson, D.C., Yang, Q.: Introduction to the special issue on maintaining case-based reasoning systems. *Computational Intelligence* 17(2), 193–195 (2001)
14. Leake, D.B., Wilson, D.C.: Categorizing case-base maintenance: Dimensions and directions. In: Smyth, B., Cunningham, P. (eds.) EWCBR 1998. LNCS (LNAI), vol. 1488, pp. 196–207. Springer, Heidelberg (1998)
15. de Mántaras, R.L., McSherry, D., Bridge, D.G., Leake, D.B., Smyth, B., Craw, S., Faltings, B., Maher, M.L., Cox, M.T., Forbus, K.D., Keane, M.T., Aamodt, A., Watson, I.D.: Retrieval, reuse, revision and retention in case-based reasoning. *Knowledge Eng. Review* 20(3), 215–240 (2005)
16. Markovitch, S., Scott, P.D.: The role of forgetting in learning. In: Fifth International Conference on Machine Learning, pp. 459–465. Morgan Kaufmann, Ann Arbor (1988)
17. Minton, S.: Quantitative results concerning the utility of explanation-based learning. *Artif. Intell.* 42(2-3), 363–391 (1990)
18. Shokouhi, S.V., Aamodt, A., Skalle, P., Sørmo, F.: Determining root causes of drilling problems by combining cases and general knowledge. In: McGinty, L., Wilson, D.C. (eds.) Case-Based Reasoning Research and Development. LNCS, vol. 5650, pp. 509–523. Springer, Heidelberg (2009)
19. Smyth, B., Cunningham, P., Cunningham, P.: The utility problem analysed - a case-based reasoning perspective. In: Smith, I., Faltings, B.V. (eds.) EWCBR 1996. LNCS, vol. 1168, pp. 392–399. Springer, Heidelberg (1996)
20. Smyth, B., Keane, M.T.: Remembering to forget: A competence-preserving case deletion policy for case-based reasoning systems. In: 13th International Joint Conference on Artificial Intelligence, pp. 377–382. Morgan Kaufmann, San Francisco (1995)
21. Smyth, B., McKenna, E.: Footprint-based retrieval. In: Althoff, K.-D., Bergmann, R., Branting, L.K. (eds.) ICCBR 1999. LNCS (LNAI), vol. 1650, pp. 343–357. Springer, Heidelberg (1999)
22. Sørmo, F.: Real-time drilling performance improvement. *Scandinavian Oil & Gas Magazine*, No. 7/8 (2009)
23. Tambe, M., Newell, A., Rosenbloom, P.S.: The problem of expensive chunks and its solution by restricting expressiveness. *Machine Learning* 5, 299–348 (1990)
24. Watson, I.: A case study of maintenance of a commercially fielded case-based reasoning system. *Computational Intelligence* 17, 387–398 (2001)
25. Zhu, J., Yang, Q.: Remembering to add: Competence-preserving case-addition policies for case-base maintenance. In: IJCAI 1999: 16th international joint conference on Artificial intelligence, pp. 234–239. Morgan Kaufmann Publishers Inc., San Francisco (1999)