Perks of Being Lazy: Boosting Retrieval Performance

Mehmet Oğuz Mülâyim and Josep Lluís Arcos

IIIA, Artificial Intelligence Research Institute
CSIC, Spanish National Research Council
Bellaterra, Spain
{oguz,arcos}@iiia.csic.es

Abstract. Case-Based Reasoning (CBR) is a lazy learning method and, being such, when a new query is made to a CBR system, the swiftness of its retrieval phase proves to be very important for the overall system performance. The availability of ubiquitous data today is an opportunity for CBR systems as it implies more cases to reason with. Nevertheless, this availability also introduces a challenge for the CBR retrieval since distance calculations become computationally expensive. A good example of a domain where the case base is subject to substantial growth over time is the health records of patients where a query is typically an incremental update to prior cases. To deal with the retrieval performance challenge in such domains where cases are sequentially related, we introduce a novel method which significantly reduces the number of cases assessed in the search of exact nearest neighbors (NNs). In particular, when distance measures are metrics, they satisfy the triangle inequality and our method leverages this property to use it as a cutoff in NN search. Specifically, the retrieval is conducted in a lazy manner where only the cases that are true NN candidates for a query are evaluated. We demonstrate how a considerable number of unnecessary distance calculations is avoided in synthetically built domains which exhibit different problem feature characteristics and different cluster diversity.

Keywords: Lazy retrieval · Triangle inequality · Exact nearest neighbor search

1 Introduction

Being a lazy learning methodology, a Case-Based Reasoning (CBR) system will try to generalize its cases at the time a query is made to the system. And a typical CBR retrieval algorithm ends up calculating the similarity of the query to all of the cases in the case base (CB). While a CB grows larger by time as new experiences are incorporated as cases, especially the time spent at the retrieval phase in reasoning episodes is likely to become a performance issue. Computationally expensive distance calculations in the search of nearest neighbors (NNs) eventually leads to the utility problem (a.k.a. swamping problem) which is commonly known within the CBR community. Indeed, the utility problem has proved to be

one of the most studied subfields to be able to overcome the issues faced in real world implementations of CBR systems [3, 4, 7]. Two major approaches to cope with this problem have been smart indexing of the cases and reducing the size of the case base in terms of competence [8, 10]. We should note that efficiency in NN search has not been only a problem of the CBR community, and there has been considerable work on finding approximate NNs instead of exact NNs in large-scale high dimensional spaces (for a survey see [12]).

Health sciences are increasingly becoming one of the major focuses for CBR research (for a survey of applications see [1]). A medical CBR system where health records of patients constitute the CB is expected to grow considerably over time and hence, it is prone to face the utility problem if extra care is not taken at the retrieval process.

In such a domain, a new problem is typically an incremental update to its predecessor case in the CB. By saying incremental, we mean that it comprises (partially or completely) the information of its sequentially related predecessors. In the above mentioned medical CBR system, after the initial session for a patient, consecutive sessions would be updates to previous sessions. These sessions would altogether form a sequence of cases, i.e. the case history of the patient (see Figure 1 for a visual representation of such a CB). A query to find patients with similar health records should take into account these sequences of cases for the patients in the CB. As the cases are appended, the sequence grows longer by time and the incremental difference introduced by the new problem becomes minimal compared to the shared long history between the latest cases of the sequence. Then, it is intuitive to think that a new problem's NNs are very likely to be similar to those of its predecessor case. The basic assumption of CBR that "similar problems have similar solutions" also confirms this thought. Following this intuition, while we are calculating the NNs of a new problem, it makes sense to start from the ranked neighbors of the predecessor. These two suggestions lead us to the thought that if we could have an oracle that showed us to what extent we should be doing the similarity calculations (i.e. making sure that we could not find a nearer neighbor than the last one we have calculated), we could save invaluable time at the retrieval phase, especially if the CB is subject to substantial growth over time as it is in this example and/or the distance calculation is expensive for the domain.

In this work, we introduce a new approach to limit the number of cases assessed in the search of *exact* NNs in domains where a case is sequentially related to another and where the distance measures are metrics. The method leverages the triangle inequality as a cutoff to calculate the NNs of a new problem based on its distance to its predecessor case. In the sense of reducing the number of cases to assess in retrieval, our method bears resemblance to the "Fish and Shrink" strategy [6] which also exploits the triangle inequality to set bounds on similarities between cases. "Fish and Shrink" assumes that "if a case does not fit the actual query, then this may reduce the possible usefulness of its neighbors" and it uses similarity bounds to successively eliminate (shrink) similarity regions. However, our method is unique in its *incremental approach* to use similarities

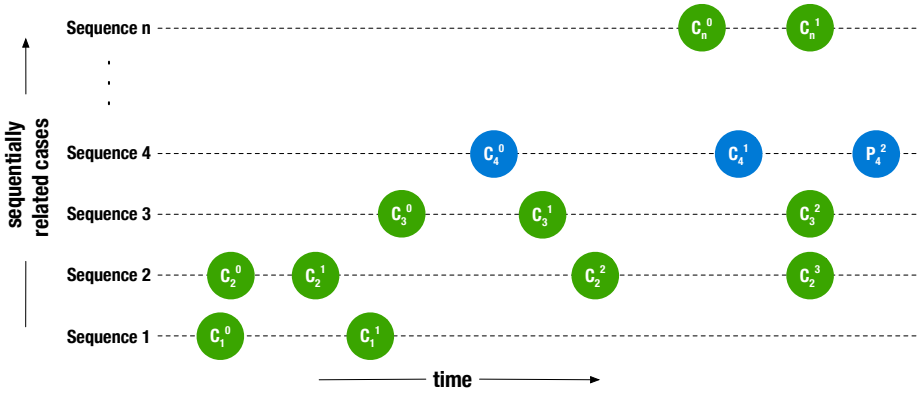

Fig. 1. A visual representation of a case base where a new problem is typically an incremental update of its predecessor case. In a health care CBR system where patient sessions form the cases, each sequence would represent the case history of a different patient. For e.g., P_4^2 would be the new query for the patient with id=4 for her upcoming third consecutive session. This query would bear implicitly and/or explicitly the information of the prior session's case C_4^1, which in its own right was an update to the initial case C_4^0.

between sequentially related cases to determine the cutoff point in the search of exact NNs.

Specifically, we describe how to implement an oracle that indicates the cutoff points in NN search in metric spaces and how to use it in a *Lazy KNN search* algorithm in Section 2. Next, in Section 3 we describe the experiments conducted in synthetic domains of different problem feature characteristics and different cluster diversity and we report the results in which we can observe the gain in terms of avoided similarity calculations by our algorithm. Finally, in Section 4 we conclude with discussion and future work.

2 Lazy Retrieval

In a CB of n cases, for a problem sequence S which has had u updates so far including the initial problem, a standard linear kNN search algorithm would have to make a total number of $u \times n$ similarity calculations to assess the kNNs of the updates of S throughout its evolution, assuming the CB does not change during these u updates. Our aim here is to try to reduce the number of calculations when a new query arrives as an update to S by using the knowledge we have acquired from the calculations made for prior updates.

Our approach to accomplish this objective is to find an oracle in such a domain that can help us carry out as few similarity (or distance) calculations as possible during the assessment of the kNNs. For a new query, if this oracle indicates a point in the CB, onward from which it is useless to search a nearer

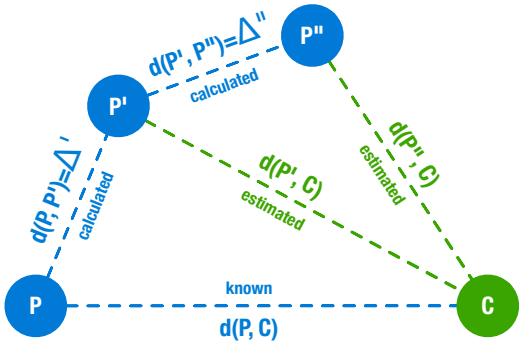

Fig. 2. Using the **triangle inequality** to estimate the distance (as well as the similarity) of two sequential problems to an existing case. Where P' is the incremental update of the problem P, and P'' is the incremental update of the problem P', C is any case in the CB, and d is the distance metric; we have $d(P, C) \leq d(P', C) + \Delta'$ and $d(P, C) \leq d(P'', C) + \Delta' + \Delta''$.

neighbor (since it guarantees that there is none ahead), we could hope to save some precious time spent for calculations.

In the following subsection, we propose such an oracle which leverages the triangle inequality in metric spaces and then we introduce our algorithm for the lazy assessment of the kNNs using this oracle.

2.1 Leveraging the Triangle Inequality in kNN Assessment

The most popular metric space used by CBR systems is the Euclidean space and being a metric space, distance metrics used in this space should satisfy the triangle inequality among other axioms [2]. When we already know the similarity of a case to a previous update of a given problem sequence, we show how we exploit this property to calculate the upper-bound of the similarity of this case to the current update of that problem.

Let us use Figure 2 to illustrate the triangle inequality property. Given three points in the problem space $\langle P, P', C \rangle$, P' being the incremental update of the problem P, and C any case in the CB, and given d as a distance metric, the three points will satisfy the following triangle inequality:

$$d(P, C) \leq d(P, P') + d(P', C)$$

This property also holds for P'' which is the incremental update of the problem P'. Thus, new distances $d(P', C)$ and $d(P'', C)$ are conditioned by the distances among the incremental updates of the problem.[1]

[1] Following Figure 1, you can think of P and P' as the problem parts of the cases C_4^0 and C_4^1 respectively, and P'' as P_4^2, and C as any case C_x^y of any Sequence x where x$\neq$4.

Finally, since we already know $d(P, C)$, and we may calculate $d(P, P')$ and $d(P', P'')$, using the triangle inequality we may obtain the *upper* limits of similarity of two sequences to a case C, namely $sim(P', C)$ and $sim(P'', C)$ without the need to actually calculate them in the following way:

The triangle inequality allows us to write the following two inequalities:

$$\begin{aligned} d(P, C) &\leq \overbrace{d(P, P')}^{\Delta'} + d(P', C) \\ d(P', C) &\leq \overbrace{d(P', P'')}^{\Delta''} + d(P'', C) \end{aligned} \tag{1}$$

Then, using the latter inequality in the former (1), we get:

$$d(P, C) \leq d(P'', C) + \Delta' + \Delta'' \tag{2}$$

Because we are more interested in similarities rather than distances, we can transform above inequalities into similarity inequalities. When all the distances d are normalized values into the range of $[0, 1]$ in accordance with the common, and most of the times necessary, practice in CBR systems to compare distances and/or similarities in a CB, the inequalities (1) and (2) can be written in terms of similarity as follows where $sim(x, y) = 1 - d(x, y)$:

Following the inequality (1):

$$\begin{aligned} \underbrace{1 - d(P, C)} &\geq \underbrace{1 - d(P', C)} - \Delta' \\ sim(P, C) &\geq sim(P', C) - \Delta' \end{aligned}$$

which leads to:

$$sim(P', C) \leq sim(P, C) + \Delta' \tag{3}$$

Following the inequality (2):

$$\begin{aligned} \underbrace{1 - d(P, C)} &\geq \underbrace{1 - d(P'', C)} - \Delta' - \Delta'' \\ sim(P, C) &\geq sim(P'', C) - \Delta' - \Delta'' \end{aligned}$$

which leads to:

$$sim(P'', C) \leq sim(P, C) + \Delta' + \Delta'' \tag{4}$$

The inequality (3) gives us the oracle we need. Just by calculating Δ', we know that a new update P' can be more similar to *any* case C than its predecessor P is to C, at best by a degree of Δ'. We see it important to reemphasize the word "*any*" here, because note that Δ' calculation does not involve any case but only the problem P and its update P'.

Consecutively, a following update P'' can be more similar to any case C in the CB than P is to it, at best by a degree of $\Delta' + \Delta''$ as shown in the inequality (4).

Intuitively if we generalize the inequality (4), we have:

$$sim(P^i, C) \leq sim(P^j, C) + \sum_{s=j+1}^{i} \Delta_P^s \tag{5}$$

where

$$\Delta_P^s = d(P^{s-1}, P^s)$$

Now, we can say that any update P^i of a problem P can get more similar to any case C in the CB, than a prior update P^j is to it, at best by a degree of the sum of Δs calculated between updates $j+1$ and i (both inclusive).

In the following subsection, we explain how we leverage this knowledge as a cutoff in the search of exact kNNs of the problem updates.

2.2 Lazy assessment of the kNNs

The working hypotheses are that i) any problem P will be updated many times, ii) the CBR system will have to provide the k-nearest neighbors for each update of P, and iii) we are interested in reducing the retrieval effort as much as possible. As a consequence, the CBR system should keep some information (state) regarding previous calculations in the retrieval step. The simplest state is a ranked list of similarities between a problem and all the cases in the case base.

The first time a new problem P is presented to the system, the retrieval step will be performed as usual: calculating the similarity of the new problem to all cases in the case base[2]. Beside providing the k-nearest cases, the retrieval step will keep this rank ($RANK_P$) for future calculations.

Later, each time the problem P is amended with an update P', the CBR system will exploit the inequality presented in (3) to try to minimize similarity calculations adopting the following strategy:

1. updating the similarities between the current problem P' and, previously calculated k-nearest neighbors in rank $RANK_P$;
2. calculating an upper bound Δ' to determine the best possible improvement in similarity, by computing the similarity between the current problem P' and its previous update (i.e. the initial problem P);

[2] As we will discuss in a following section, this calculus can be improved by introducing some proposals already existing in the CBR literature.

3. adding the Δ' upper bound to all the rest (i.e. starting from the $k+1^{th}$ neighbor) of similarities previously stored in rank $RANK_P$ to find their "optimistic" similarity values to the current problem;
4. calculating the actual similarity of P' to any such case whose "optimistic" similarity to the current problem surpasses the k^{th} neighbor's calculated similarity (in the first step above) to it.

Remembering that $RANK_P$ is a ranked list, the CBR system may iterate the list from the top and stop at a position l when the "optimistic" similarity of case C_l does not beat the similarity calculated for the k^{th} neighbor in $RANK_P$. And this would mean that starting from C_l, the remaining cases ahead in $RANK_P$ certainly cannot beat the k^{th} neighbor in $RANK_P$ either.

The second observation is that, since Δ' depends only on the distance between the current problem P' and its predecessor P, the procedure described above can be improved by delaying the calculation of "optimistic" similarities as much as possible. As presented in (5), when we know the similarities of a group of cases to the problem sequence P^j, we can calculate their "optimistic" similarities to the current problem sequence P^i by adding the sum of all Δs between iteration $j+1$ and i to their known similarity values in iteration j. Thus, all the similarities calculated in the same iteration i share the same accumulated Δs. As a consequence, instead of updating all "optimistic" similarities in each iteration, it is enough to keep a sub-rank for each iteration.

Finally, above described *Lazy KNN search* algorithm can be found as pseudo-code in Algorithm 1. As it can be seen in the pseudo-code as well, throughout the iterations, we delay the assessment of a case as a nearest neighbor in a *lazy* manner until we consider it as a true candidate for the kNN list.

In the formalization of the upper bound calculation and in the algorithm presented, we have assumed that the CB remains unchanged between the updates of a problem P for the sake of simplicity. To be able to tackle the changes to the CB, the algorithm can be improved easily by adding following behaviors:

1. If new cases are incorporated after the last update P^i, their similarities to the next update P^{i+1} have to be calculated in the $i+1^{th}$ update;
2. If old cases are modified and if these modifications affect the similarity calculations, these cases should be removed from the $RANK_P$ list and their similarities to the next update P^{i+1} have to be calculated in the $i+1^{th}$ update;
3. If old cases are deleted, they have to be deleted from the $RANK_P$ list as well.

3 Experimentation

Experiments were performed on a collection of synthetically generated datasets to assess 1) the impact of the distribution of the cases in the CB, and 2) the impact of problem changes in each problem update on the gain in the number of calculations achieved by our algorithm. For NN_P^i the list of cases whose

Algorithm 1: Lazy KNN search algorithm

Where
k is the number of nearest neighbors to be returned;
P^i is the i^{th} update to the problem P, for $i \geq 0$ and P^0 is the initial problem P;
d is the distance metric;
$\Delta_P^i = d(P^{i-1}, P^i)$;
NN_P^i is the list of (c, sim) 2-tuples formed for every case c to which P^i's similarity is actually calculated where $sim = 1 - d(P^i, c)$, this list is ranked in a descending order, the most similar case being at the top;
$NN_P^i[k].similarity$ is the similarity part of the k^{th} 2-tuple in NN_P^i;
$NN_P^i[:k]$ is the list of top k members of NN_P^i, i.e. the kNNs of P^i;
$RANK_P$ is the list of (NN_P^i, Δ_P^i) 2-tuples that are calculated for all occurred updates of the problem P. When P^i arrives, the content of this list is as follows:

$$RANK_P = [(NN_P^{i-1}, \Delta_P^{i-1}), (NN_P^{i-2}, \Delta_P^{i-2}), \ldots, (NN_P^1, \Delta_P^1), (NN_P^0, null)]$$

input : k,
P^i,
P^{i-1}, // $P^{i-1} = null$ `if` $P^i \equiv P^0$
$RANK_P$
output: $(NN_P^i[:k], RANK_P)$ // 2-tuple

```
1  NN_P^i ← [ ]
2  if P^{i-1} = null then        // Treatment of the initial problem, P^i ≡ P^0
3  |  Δ_P^i ← null
4  |  foreach c in case base do                                  // Linear search
5  |  |  NN_P^i .append((c, 1−d(P^i, c)))
6  |  end
7  |  NN_P^i .sort_descending()
8  else                        // Treatment of the following problem updates
9  |  foreach (c, sim) in NN_P^{i-1}[:k] do  // Calc sims of P^{i-1}'s kNNs to P^i
10 |  |  NN_P^{i-1} .remove((c, sim))
11 |  |  NN_P^i .append((c, 1−d(P^i, c)))
12 |  end
13 |  NN_P^i .sort_descending()
14 |  Δ_P^i ← d(P^i, P^{i-1})
15 |  SumΔ_P ←Δ_P^i
16 |  foreach (NN_P^j, Δ_P^j) in RANK_P do                  // Iterate RANK_P
17 |  |  foreach (c, sim) in NN_P^j do
18 |  |  |  if (sim+SumΔ_P)> NN_P^i[k].similarity then    // c a candidate?
19 |  |  |  |  NN_P^j .remove((c, sim))
20 |  |  |  |  NN_P^i .append((c, 1−d(P^i, c)))       // Calc the actual sim
21 |  |  |  |  NN_P^i .sort_descending()
22 |  |  |  else
23 |  |  |  |  break      // Continue with the next 2-tuple in RANK_P
24 |  |  |  end
25 |  |  end
26 |  |  SumΔ_P ←SumΔ_P +Δ_P^j                          // Accumulate Δs
27 |  end
28 end
29 RANK_P.append((NN_P^i, Δ_P^i))
30 return (NN_P^i[:k], RANK_P)
```

similarity was calculated at iteration i, we define the gain at an iteration i as follows:

$$gain(P^i) = \frac{|CB| - |NN_P^i|}{|CB|}$$

where $|A|$ denotes the cardinality of a set A.

Datasets with a uniform distribution of cases will probably result in higher distances among cases but smoother changes on neighborhoods. Conversely, datasets with dense but distant clusters of cases will present less advantages at the beginning, but significant gains in the long run (see Subsection 3.2 for a more detailed discussion).

The amount of change among problem updates will determine the margin for deltas. Since we are interested in domains with cases keeping information over long periods of time, we could represent cases with many sequences of tiny updates to guarantee small deltas. However, fragmenting cases in many sequences will result in an increase of calls to the Lazy KNN search algorithm.

3.1 Datasets

To perform comparable experiments, the number of sequences was set to 10,000 and the number of problem features was set to 100 in all datasets. All the experiments were performed using a normalized euclidean similarity as the similarity measure.

To assess the gain effect of the proposed algorithm when the granularity of problem updates varies, we have played with two parameters: the number of the problem updates (U) and the maximum number of feature changes (V). For instance, since each update in a problem sequence is a case, a U=40 would mean that there are $10,000 \times 40 = 400,000$ cases in the CB. And a V=10 means that from one update to another update 10 units will be distributed randomly to increment feature values where one unit represents the minimal change for a feature value. Note that one feature slot may receive more than one unit of change. In a given dataset, V is fixed for all cases and problem updates.

To assess the gain effect of the proposed algorithm over different case base densities, we have introduced a parameter B to control the number of clusters (blobs). A total of number of 48 datasets were generated by combining the values of parameters detailed in Table 1.

To generate the datasets, we have used the blobs generator available in the scikit-learn library [5]. To guarantee the generation of overlapped blobs as B increased, the parameter *cluster_std* was set to 5.0: The generator of blobs was used to obtain prototypes of cases which were later fragmented in sequences of length U with a random distribution of values satisfying V changes from update to update[3].

Parameter	Values
B	$[1, 5, 10, 20]$
U	$[10, 20, 30, 40]$
V	$[10, 25, 50]$

Table 1. Range of values for each dataset parameter.

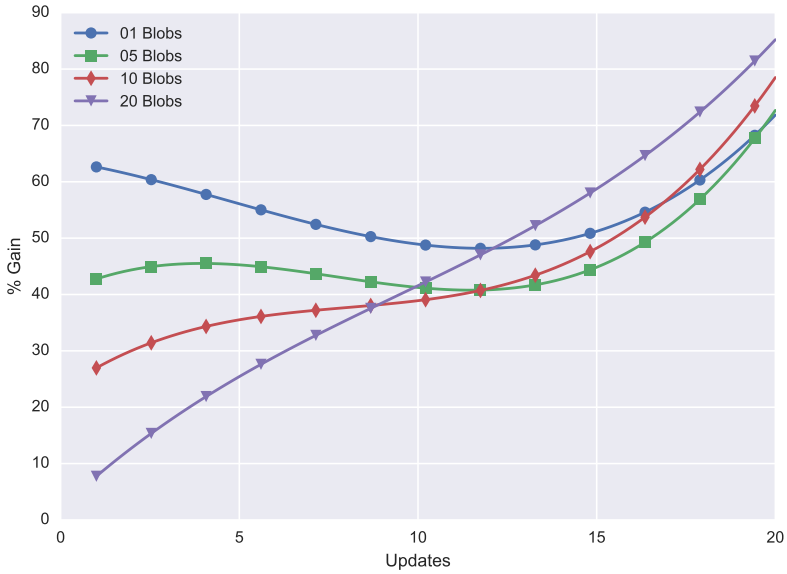

Fig. 3. Impact of the number of blobs (U=20, V=10).

3.2 Results with varying CB densities

The first study conducted was the analysis of the gains with varying case base densities, i.e. varying parameter B, and fixing parameters U and V. The behavior of the gain along U was similar on all configurations of pairs $[U, V]$.

Take as example[4] Figure 3 where U=20 and V=10. In the first half of iterations, the number of blobs is inversely correlated with the gain. However, as iterations increase, the number of blobs (more blobs imply more sub-regions in the CB) is directly correlated with gains. For datasets generated with a low number of blobs, neighborhoods tend to be less dense and cases more uniformly distributed on the problem space, therefore the change in nearest neighbors is

[3] Dataset generation code can be found at http://www.iiia.csic.es/∼oguz/lazy/

[4] A document with supplementary material with figures summarizing all datasets can be found at http://www.iiia.csic.es/∼oguz/lazy/.

Num. Blobs	Acc. Gain
1	54.95 %
5	46.28 %
10	43.17 %
20	42.91 %

Table 2. Effect of increasing the number of blobs on accumulated gain (U=20, V=10).

more smooth but more constant. As a consequence, gains start higher than other datasets but remain more constant over time.

Gains on datasets generated with a high number of blobs behave differently. In the beginning many cases are similar enough and this generates the effect of having many nearest neighbor candidates, i.e. at first gains are low, but as time goes by, a subset of nearest neighbors become very close allowing high gains. In Figure 3 we can observe this extreme behavior for dataset with B=20.

In Figure 3 we can see the gain for each iteration, however the accumulated gains throughout iterations are not evident. We can find summarized accumulated gains in Table 2 where we can observe that the constant gains of datasets generated with less blobs have their reward as a higher accumulated gain in the end.

3.3 Results with varying problem changes

The second study conducted was the analysis of the gains with varying problem changes, i.e. varying parameters U or V, for a fixed parameter B. Because we are using a normalized similarity measure among cases, parameters U and V are correlated. Low values of change (V) together with short problem sequences (smaller U) have a similar effect as high values of change combined with long problem sequences.

Num. Iterations	Acc. Gain
10	20.64 %
20	43.17 %
30	55.93 %
40	60.71 %

Table 3. Effect of increasing the number of iterations on accumulated gain (B=10, V=10).

Take as example Figure 4 where B=10 and V=10. We can observe that datasets with less iterations depart from lower gains but are able to achieve higher gains in the end. The second observation is that when iterations are long enough (i.e. U=30 and U=40) the evolution of gains converges to the same

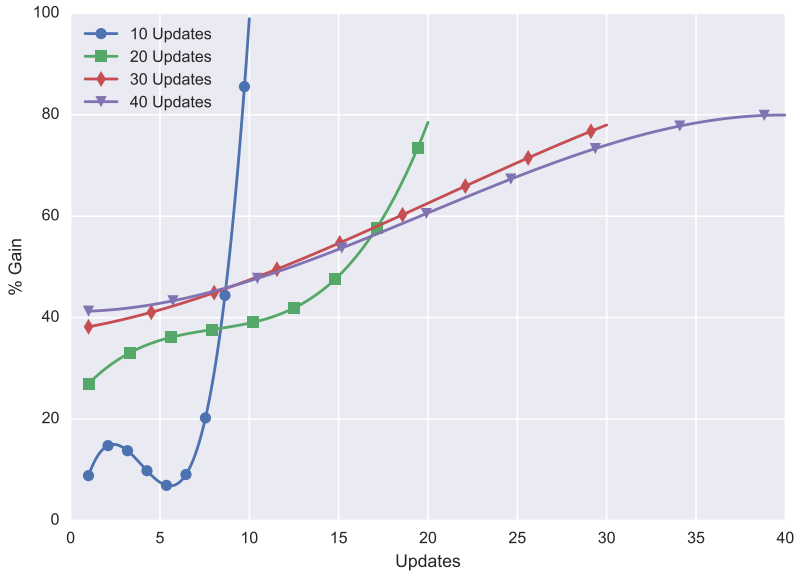

Fig. 4. Impact of the amount of problem updates (B=10, V=10).

behavior. This result is important because the main motivation of the proposed algorithm was to deal efficiently with long sequences of problems, i.e. to take advantage of cases that will be updated many times. Table 3 summarizes the accumulated gains for Figure 4. As expected, the accumulated gain increases on datasets with longer sequences.

4 Discussion and Future work

In this article, we have presented a novel approach that can significantly improve the retrieval performance of CBR systems that are designed for domains where a case is typically a sequential update to a prior case in the case base. The improvement is achieved by limiting the number of the cases that are evaluated in the exact k-nearest neighbor (kNN) search for consecutive updates of a problem. And we can reduce the number of calculations thanks to the triangle inequality property which holds for metric spaces.

When a new update of a problem is introduced as a query to the CBR system, we leverage this property to calculate the upper bound of the similarity change for all cases in the case base for which we had calculated their similarities to previous updates of the same problem. Using this optimistic similarity change, we determine which cases are true candidates for the kNN list of the current query.

We have provided a formal description of our methodology and our algorithm for the lazy assessment of the kNNs. Then we have shared the description and results of our experiments that we conducted in synthetically built domains of different problem feature and cluster characteristics. We have demonstrated gains obtained by our algorithm which avoided redundant similarity calculations to a significant degree.

We believe our method can boost the retrieval performance in domains where we deal with large case bases constituted of packages of sequentially related cases as described. We note that although our method is based on metric spaces, non-metric distance measures for which there are ways to transform them into metric measures can also make use of our method. A good example of this could be transforming the popular cosine similarity which is non-metric by nature into a metric distance [11].

At the current implementation, the actual similarity of a problem update to a case is calculated from scratch, i.e. it is treated the same as any initial problem. However, since these updates are incremental in their nature, we can make use of the previously calculated similarity of its predecessor to the same case and only calculate the contribution of change in similarity which is introduced by the current update. As future work, we are planning to implement such an incremental computation of similarity measures for yet a better improvement of retrieval performance in similar domains where this computation is relevant.

A further improvement would come by taking advantage of the *footprint cases* concept in CBR literature [9] in our method. Since a footprint case represents a group of cases sharing a similar competence contribution, our method can choose kNN candidates within these footprint cases instead of a larger group of candidates.

Acknowledgements

This work has been partially funded by project Innobrain, COMRDI-151-0017 (RIS3CAT comunitats), and Feder funds. Mehmet Oğuz Mülâyim is a PhD Student of the doctoral program in Computer Science at the Universitat Autònoma de Barcelona.

References

1. Begum, S., Ahmed, M.U., Funk, P., Xiong, N., Folke, M.: Case-based reasoning systems in the health sciences: a survey of recent trends and developments. IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) **41**(4), 421–434 (2011)
2. Deza, M.M., Deza, E.: Encyclopedia of distances. In: Encyclopedia of Distances, pp. 1–583. Springer (2009)
3. Francis, A.G., Ram, A.: The utility problem in case-based reasoning. In: Case-Based Reasoning: Papers from the 1993 Workshop. pp. 160–161 (1993)

4. Houeland, T.G., Aamodt, A.: The utility problem for lazy learners-towards a non-eager approach. In: International Conference on Case-Based Reasoning. pp. 141–155. Springer (2010)
5. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine learning in Python. Journal of Machine Learning Research **12**, 2825–2830 (2011)
6. Schaaf, J.W.: Fish and shrink. a next step towards efficient case retrieval in large scaled case bases. In: European Workshop on Advances in Case-Based Reasoning. pp. 362–376. Springer (1996)
7. Smyth, B., Cunningham, P.: The utility problem analysed. In: Smith, I., Faltings, B. (eds.) Advances in Case-Based Reasoning. pp. 392–399. No. 1168 in Lecture Notes in Computer Science, Springer Berlin Heidelberg (1996)
8. Smyth, B., Keane, M.T.: Remembering to forget. In: Proceedings of the 14th international joint conference on Artificial intelligence. pp. 377–382. Citeseer (1995)
9. Smyth, B., McKenna, E.: Competence guided incremental footprint-based retrieval. Knowledge Based Systems **14**(3–4), 155–161 (2001)
10. Smyth, B., McKenna, E.: Competence models and the maintenance problem. Computational Intelligence **17**(2), 235–249 (2001)
11. Van Dongen, S., Enright, A.J.: Metric distances derived from cosine similarity and pearson and spearman correlations. arXiv preprint arXiv:1208.3145 (2012)
12. Wang, J., Shen, H.T., Song, J., Ji, J.: Hashing for similarity search: A survey. arXiv preprint arXiv:1408.2927 (2014)