Extending the JColibri open source architecture for managing high-dimensional data and large case bases

A. Bottrighi, G. Leonardi, S. Montani, L. Portinale
Dipartimento di Informatica
Università del Piemonte Orientale “A. Avogadro”, Alessandria (Italy)

Abstract

CBR systems designers and developers’ research can benefit from the availability of existing platforms, able to provide software design and implementation assistance. The JColibri platform [26], realized and maintained by the University of Madrid, is one of the most well known among such tools.

In this work, we describe a couple of extensions we have provided to the core JColibri open source software. In particular, our extensions are meant to optimize case retrieval performances, in data-rich applications. Specifically, we focused our attention on treating (i) large case bases, in which retrieval time may become unacceptable, and (ii) cases with high-dimensional features - namely time series features - on which proper case representation and retrieval solutions need to be studied.

The implemented code has been preliminarly tested, and it is now ready to be integrated with the JColibri code, and made available to the CBR research community. Additional extensions, always dealing with retrieval optimization, are foreseen as our future work.

1 Introduction

Case-based Reasoning (CBR) [1] is a reasoning paradigm based on the intuition that problems tend to recur. Basically, it relies on the idea that solutions of past instances of problems (cases henceforth), similar to the current one, can be used to solve the current problem itself, possibly after a proper adaptation phase. CBR is particularly applicable to problems where (many) earlier past cases are available, even when the domain is not fully understood, and a completely formalized domain theory is not available, or would required too much knowledge engineering effort.

CBR is now a mature and well established methodology, successfully applied in several domain (see e.g. [32]). As soon as CBR applications have become progressively widespread, the availability of tools to build CBR systems, able to provide design and implementation assistance with software engineering facilities, also exploitable for teaching proposes and for code reuse and interchange between researchers, has become a critical need.

Starting from these observations, a group of Artificial Intelligence researchers at the Computer University of Madrid has recently implemented a platform named JColibri [26], which proposes a reference architecture for the development of CBR systems, along with its corresponding implementation. JColibri is written in Java, and is meant to be used both by CBR system designers, and by CBR system developers, who are also allowed to define and implement additional modules, to be made publicly available together with the core JColibri open source software.

In this work, we describe a couple of extensions to the core JColibri code, which we have implemented working as CBR systems developers.

As it is well known, CBR can be summarized by the following four basic steps, known as the CBR cycle, or as the four “res” [1]:

- **retrieve** the most similar case(s) with respect to the input situation from the case base;
- **reuse** them, and more precisely their solutions, to solve the new problem;
- **revise** the proposed new solution (if needed);
- **retain** the current case for future problem solving.

In particular, we focused our attention on the case retrieval step, and proposed proper optimization strategies, to be adopted when dealing with:

- large case bases, in which retrieval time can become unacceptable, and
- cases with high-dimensional features, on which proper case representation and retrieval solutions need to be studied, in order to ensure efficiency. In detail, time
series features have been the main object of our analysis.

The implemented code has been preliminarly tested, and it is now ready to be made available to the CBR research community. Additional evaluations and extensions are foreseen as our future work.

The paper is organized as follows. Section 2 sketches the JColibri architecture. Section 3 motivates our work, and describes the methodological foundations and the technical details of the extensions we provided. Finally section 4 addresses our concluding observations.

2 JColibri architecture

To support both designers and developers, JColibri defines a multi-level architecture that has two layers, each oriented to one of these profiles. This architecture is shown in figure 1.

The bottom layer is oriented to developers and provides an architecture that is able to integrate diverse CBR techniques. This layer provides the basic blocks for building CBR systems that can be easily extended and reused by programmers. It offers a set of data types and of methods, meant to cover the basic needs of CBR system developers. For instance, a set of classical distance calculation functions are provided for the case retrieval step. However, it also allows to implement and include new data type definitions and modules, in order to enable the JColibri architecture to cover additional needs.

Regarding the top layer, it is oriented to designers since it includes a toolbox that partially automates the generation of CBR systems through the composition of the elements in the framework. This composition is guided by templates that abstract the behavior of CBR systems and can be instantiated with the different components of the bottom layer. This instantiation is driven by a semantic representation of these components that is based on an ontology named CBROnto. Therefore, this top layer defines another architecture that organizes the knowledge and elements required to implement such composition techniques.

Returning to the bottom layer, figure 1 shows the main components to be used to extend the functionalities of JColibri. In particular, our contribution is provided by extending or implementing the following components:

1. the StandardCBRApplication component, which defines the methods preCycle(), cycle() and postCycle(), to be implemented to build a CBR application. These methods must contain, respectively, the initialization procedures, the execution of the CBR cycle and the procedures to close the application;
2. the **DataType** component, which contains the interfaces **TypeAdaptor** to introduce and define new data types, and the interface **CaseComponent** which permits to use the new type as a case feature;

3. the **Similarity** component, which offers the following interfaces: **LocalSimilarity** to define the similarity metric for single features of a case, and **GlobalSimilarity** to define the global similarity measure to compare two cases;

4. the **Connector** component, which permits to create connections to deal with the new data types introduced. The connectors allow the CBR applications to access the case base, loading and storing cases and information.

Further details will be described in Section 3.

3 Extending JColibri

As anticipated in the Introduction, we have extended the JColibri architecture in two directions, namely:

- by adding a module for retrieval performance optimization on large case bases, and
- by adding proper modules for representing and retrieving cases with high-dimensional features, particularly focusing on time series.

The methodological and technical details of such extensions will be described in the next two subsections.

### 3.1 Dealing with large case bases

As the number of items in the case base becomes higher and higher, retrieval performances can progressively and significantly worsen, due to the need of calculating the distance between the input case and all the available past cases. However, retrieval time can be reduced, if a non-exhaustive search and distance calculation strategy is implemented. Several solutions can be identified to this end, being classification and intra-class retrieval (see e.g. [15, 18]) one of the most well known. In the past, we have proposed a different strategy to non-exhaustive search, which relies on a methodology called Pivoting-based retrieval (PBR; see [23] for details). The main idea in PBR consists in:

- choosing a pivot as a representative case;
- computing the distance between the representative case and all the other cases;
- computing the distance between the representative case and the input case;
- estimating the distance between the input case and all the remaining cases by using triangle inequality, thus finding a lower and an upper bound for the distance value.

The intervals whose lower bound is higher than the minimum of all the upper bounds can be pruned (see figure 2). The following iterative procedure (called the main cycle) is then applied:

1. Initialization: $BEST_p = \infty$ e $SOL = \{ \}$
2. Choose the Pivot case as the minimum of the mid-points of the intervals; compute the distance between the input case and the Pivot ($DIST$); set $BEST = DIST$;
3. If $BEST_p > BEST$ set $SOL = PIVOT$ and $BEST_p = BEST$
4. If $BEST_p = BEST$ set $SOL = \{ PIVOT, SOL \}$
5. Prune the intervals whose lower bound is bigger than $BEST$, and remove the Pivot from the set of cases (see figure 2)

![Figure 2. Bound pruning in PBR](image-url)
shown in figure 3. To test the PBR module, we have compared it with the k-NN one (see [5] for details), which is provided by JColibri. As a case study, we have used the case base of the Travel Recommender application, which is included in JColibri package. This case base contains 1024 cases. In the test phase, we have executed six different queries using the two techniques, in order to retrieve a solution stored in the case base. Table 1 reports the results of such a phase: we show, for every query, the execution time of k-NN, the execution time of PBR retrieval and the number of cases that were pruned by the PBR module. We can observe that execution time is expressed in milliseconds, and concerns only the retrieval phases, by disregarding the preprocessing phase where the needed data structures are created. It is possible to see that only in one case the PBR retrieval time is higher than k-NN one; this corresponds to a situation when only few cases are pruned, therefore the whole case base is actually considered during the main cycle of the retrieval phase. In the other cases, the PBR retrieval time is significantly lower than k-NN and depends on the number of cases which are actually pruned.

### 3.2 Dealing with high-dimensional data

Several real-world applications, e.g. in the financial and in the medical fields, are intrinsically data intensive. In particular, in many domains it is necessary to capture the evolution of the observed phenomenon over time, in order to describe its behaviour, and to exploit this information for future problem solving. In these applications, (many) process features are naturally collected in the form of time series, often automatically sampled and recorded by control instruments, as it happens e.g. in Intensive Care Unit patient monitoring [22], or in haemodialysis [18].

CBR is widely being recognized as a valuable knowledge management and decision support methodology in these domains. As a matter of fact, various CBR applications dealing with cases with time series features have been recently investigated in the medical domain [30, 29, 20, 18], as well as in different ones (e.g. robot control [25], process...
Adopting CBR is typically non trivial in these situations, since the need for describing the process dynamics impacts both on case representation and on case retrieval, as analysed in [17]. In particular, time series cannot be simply stored as feature values “as they are”: pre-processing techniques are required, in order to simplify feature mining and knowledge representation, and to optimize the retrieval activity.

Within JColibri, we have thus implemented a set of modules for time series representation and retrieval, able to support researches working on this kind of applications.

In particular, an efficient case representation and retrieval requires time series dimensionality reduction, which allows to reduce memory (i.e. case storage) space, still capturing the most important characteristics of the time series itself [8].

In the literature, dimensionality is typically reduced by means of a mathematical transform, able to preserve the distance between two time series (or to underestimate it). A widely used transform is the Discrete Fourier Transform (DFT) [2]. DFT maps time series to the frequency domain. DFT application for dimensionality reduction stems from the observation that, for the majority of real-world time series, the first (1-3) Fourier coefficients carry the most meaningful information, and the remaining ones can be safely discarded. Moreover, Parseval’s theorem [21] guarantees that the distance in the frequency domain is the same as in the time domain, when resorting to any similarity measure that can be expressed as the Euclidean distance between feature vectors in the feature space. In particular, resorting only to the first Fourier coefficients can underestimate the real distance, but never overestimates it. The definition of similarity can also be extended with invariance under a group of transformations, like amplitude scaling and shift (see [7, 24]). DFT-supported CBR has been proposed e.g. in [18], where the application domain was the one of haemodialysis.

A different approach to dimensionality reduction is Piecewise Constant Approximation (PCA) (see e.g. [11, 12]). This methodology consists in dividing a time series into k segments, and in using their average values as a k-dimensional feature vector (where obviously k << n, the original data dimensionality). The best value of k can also be estimated. PCA is robust to various transformations, such as scaling and offset translation, and its calculation has proved to be fast.

The choice of the most cost-effective transformation to apply should be done on the basis of the application at hand. In order to be as general as possible, our extensions to JColibri therefore make available both DFT and PCA strategies. Several distance functions (in the transformed time series space) have been defined as well.

In order to extend JColibri to deal with time series and with high-dimensional data, we first had to define a new data type which permits to store a time series and use it as a case feature. The interface TypeAdaptor has been thus implemented through the class TimeSeries, which contains all the methods and the data structures needed to store and return values and timestamps of a time series. Since the case base provided by JColibri stores each case in a single row of a data base, each feature composed by multiple values must be packed in a string to be stored in a single column. Therefore, our class TimeSeries also implements the methods fromString and toString (provided by the interface TypeAdaptor), to transform the string representation of the data (loaded from the case base) into the internal representation of the new data type, and vice-versa. A proper similarity measure must be associated to our new data type. For this purpose, we have implemented the interface LocalSimilarity with the class EuclideanDistance, to introduce the definition of Euclidean distance between two time series. We can now use the TimeSeries data type as a feature of a case. This can be done by defining a new case by implementing the interface CaseComponent, adding a TimeSeries feature to this case, and assigning the class EuclideanDistance as the similarity measure for this attribute. DFT and PCA have been introduced with the classes FourierTransform and PiecewiseTransform. Since the similarity measure for the reduced time series differs from the classical euclidean distance, two more classes implementing LocalSimilarity have been integrated, to be used when the features of the case have been transformed using DFT or PCA. As a final step, a new connector permits to load and store cases with attributes in the form of time series in the standard case base of JColibri.

The new components, depicted in figure 4, will be exploited in the future to test the JColibri framework’s performance when retrieving raw data, or retrieving the same data using DFT and PCA as dimensionality reduction techniques.

4 Conclusions

In this work, we have described two extensions we have provided to the JColibri open source core software. Such extensions are meant to make the framework usable for those who need to build efficient CBR systems, also when dealing with large case bases, or with high-dimensional data, as in the case of time series features.

The extensions have required to design and implement several software modules, and to introduce new data types, whose main features have been described in the paper. So
far, some tests have been made to verify the correctness and the efficiency of our modules. In the future, a larger evaluation is foreseen. However, the developed software is basically ready to be made available to the research community.

In the next future, we also plan to work on additional retrieval methods, always meant to be applied to large case bases. In particular, we will implement within the JColibri framework the Fish and Shrink methodology [28]. Indexing structures, such as K-d trees [31] and TV-Trees [13] for indexing time series features, can also be introduced to further optimize even further the retrieval performance.

As soon as the new modules will be operative, we plan to compare the retrieval performances of the various available methods (i.e. PBR, Fish and Shrink, and the default JColibri retrieval methods, essentially based on the exhaustive calculation of the Euclidean distance), on a set of different, data-rich applications.

References


