Versus: A Framework for General Content-Based Comparisons

Luigi Marini*, Peter Bajcsy†, Smruti Padhy*, Antoine Vandecreme†, Rob Kooper*, Benjamin Long†, Michal Ondrejcek*, Paul Khouri Saba†, Devin Bonnie*, Joe Chalfoun†, Kenton McHenry*
*National Center for Supercomputing Applications
University of Illinois at Urbana-Champaign
Email: {lmarini, mchenry}@illinois.edu
†National Institute of Standards and Technology
Email: peter.bajcsy@nist.gov

***Abstract*—We present a framework for the execution and dissemination of customizable content-based file comparison methods. Given digital objects such as files, database entries, or in-memory data structures, we are interested in establishing their proximity (i.e. similarity or dissimilarity) based on the information encoded within the files (text, images, 3D, video, audio, etc.). We provide an implementation of this abstraction as a Java API and a RESTful service API. This implementation includes a set of tools to support access and execution of content-based comparisons both on local and distributed computational resources, and a library of methods focused on images, 3D models, text, and documents comprised of the three. We provide three use cases to demonstrate the use of the framework: (1) content-based retrieval of handwritten text, (2) quantifying information loss and (3) the evaluation of image segmentation accuracy in cell biology.**

I. Introduction

The information age has been producing digital data at an unprecedented scale. In order to make use of this data, we rely on tools to quickly sift through these enormous collections of data. One common requirement of these tools is the ability to compare digital content with each other. Relevant application areas include content-based image retrieval, novel tools within content management systems, detecting duplicates or slightly modified versions of the same file, means of grouping and organizing large digital collections in an automated manner, and detecting differences in files such as how similar two files are when created under two different processes or information lost as a file goes through some process.

II. The Versus Framework

The Versus framework is comprised of five major components: data adapters, feature extractors, feature descriptors, proximity measures, proximity values, and indexers. Figure 1 shows the overall flow for comparing two files. The formalization of the task of establishing proximity between two files that we present is based on the general trend of many content based retrieval, machine learning, and data mining approaches. Since there are many file formats for what is a relatively smaller number of possible content types (e.g. text, images, video, 3D models), we use methods referred to as adapters to abstract away the differences between these formats. Adapters are specific to content types and provide ways to load the raw

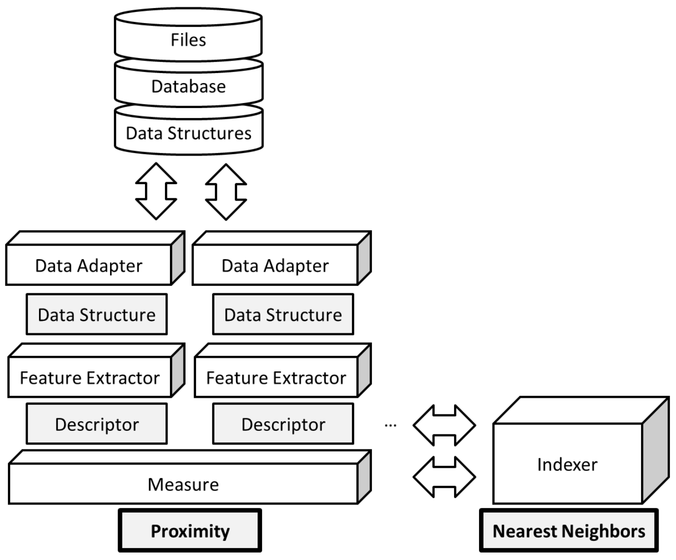

Fig. 1. *Overview of the Versus framework.*

content independently from the underlying byte encoding. It is important to note that as little information as possible should be lost when applying an adapter. Adapters are for translating content from one encoding to a more generic one and it should be easy, ideally, to reverse the process and go from the more generic data structure to the original encoding.

Extractors are methods used to identify features within the raw content. These features tend to be a subset of the data, reducing the amount of information to be considered during comparisons, and be somehow semantically meaningful with regards to an aspect of the content that will be considered during comparison. Once features are found within the content they must be described in such a way that they can be robustly identified or matched with features extracted from differing content. Feature descriptors can have a variety of forms with various pros and cons in terms of high level meaning, computational cost, and tolerance in terms of slightly differing features (i.e. noise). It is important to note that information will be lost during feature extraction, but this loss can be beneficial when it is intentional since it reduces the problem space.

Measures will take the descriptors obtained from extractors run on two different files and return a value indicating the proximity between the contents. This proximity value can be either a similarity or a dissimilarity. Proximities include information its potential minimum and maximum as well as what represents an equality between the two content descriptors.

The core package of the Java implementation provides a registry to retrieve lists of adapters, extractors, descriptors and measures at runtime. This package is primarily used in clients that need to provide a choice to the user or need to use the registry to check if a particular method is available in that instance. The framework also provides indexing facilities, since content based retrieval is one area where the ability of calculating pairwise distance between digital objects is a common task and using a linear search to compare the query object to all objects in a collection can be highly inefficient for large collections.

The current implementation provides two execution engines: a default single content descriptor engine and a comprehensive document comparison engine. The default execution engine accepts one job at a time and queues them into a threaded pool of fixed size. Here each job is represented by the two files to be compared, the identifiers of the two adapters (one per file in case the two have different formats), the identifier of an extractor, and the identifier of a measure. The comprehensive document comparison engine is used to apply different comparison measures to the same two files and return a linearly weighted combination of the results. We refer to this engine as being for documents as documents are often made of several data types (e.g. text and images) and thus several measures should be applied and then combined in some manner.

A RESTful web service, built on top of the core API, provides a service API for users who want to write distributed applications or prefer writing clients in languages other than Java. It also provides facilities to store the results of previous executions and a master-slave configuration to horizontally scale to a cluster of compute nodes. Comparisons over large archives of digital content can be executed more practically in a scalable manner over a variety of computational resources.

III. Applications

A. Content-Based Retrieval

We provide an application of the Versus content comparison framework for content-based retrieval of images of handwritten text. Based on the investigations of Diesendruck et al [1] we incorporate into the framework a new extractor that uses a Word Spotting technique [2] to extract feature descriptors from an image in a manner that is sensitive to handwritten text. To load the image contents to a compatible data structure we use a *Buffered Image Adapter*. To compare the resulting content descriptor returned from the extractor, which is a vector of frequency coefficients, we use a *EuclideanMeasure*. The power of this framework can be demonstrated in terms of its API, which can be programmed against in a variety of languages, and is used here to implement a CBIR system within a bash script possessing less than 40 lines of instructions:

```
#!/bin/bash

ADAPTER=edu.illinois.ncsa.versus.adapter.impl.BufferedImageAdapter
EXTRACTOR=edu.illinois.ncsa.versus.census.WordspottingExtractor
MEASURE=edu.illinois.ncsa.versus.measure.impl.EuclideanDistanceMeasure
SERVER=http://localhost:8081/versus/api/v1

#Upload query file
ID1=$(curl -s -F "fileToUpload=@$2" ${SERVER}/files/upload)

#Compare against directory of files
rm result

for f in $1/*; do
  ID2=$(grep $f ids.txt | awk '{print $1}')
  if [ -z "$ID2" ]; then
    ID2=$(curl -s -F "fileToUpload=@$f" ${SERVER}/files/upload)
    echo -e "$ID2\t$f" >> ids.txt
  fi

  POST="dataset1=${SERVER}/files/${ID1}&dataset2=${SERVER}/files/${ID2}"
  POST="${POST}&adapter=${ADAPTER}&extractor=${EXTRACTOR}&measure=${MEASURE}"
  MID=$(curl -s --data "$POST" ${SERVER}/comparisons)

  #Check results
  STATUS=$(curl -s ${SERVER}/comparisons/${MID}/status)
  while [ "${STATUS}" == "STARTED" ]; do
    STATUS=$(curl -s ${SERVER}/comparisons/${MID}/status)
  done

  #Print results
  if [ "$STATUS" == "DONE" ]; then
    VALUE=$(curl -s ${SERVER}/comparisons/${MID}/value)
    printf "%10.2f\t%s\n" $RESULT $VALUE $f >> result
  else
    echo "JOB '$MID' status = ${STATUS}"
  fi
done

sort -n result | head -10
```

As a linear search will be very inefficient for large collections of images one can include an additional indexer component to structure the collection descriptors. For example the *HierarchicalAgglomerativeClusteringIndexer* based on the hierarchical clustering used in [1] will construct a binary tree over the feature descriptors that allows for logarithmic search times of the content.

B. Quantifying Information Loss

A problem for archivists is that of preserving content within digital files across a variety of file formats. Many of the file formats encountered are proprietary, owned by a software vendor, and do not have open specifications by which to implement software needed to load the content from the file. In McHenry et al [3] the authors construct an extensible conversion engine utilizing 3^{rd} party software [4] (i.e. the software of the vendors owning proprietary formats) as the basis of carrying out conversions. With the automated conversion engine available the authors then attempt to empirically quantify the information loss incurred to a file's contents as it goes through a number of conversions. To do this quantification the Versus framework was used to examine the contents of a file before and after a conversion occurs. As [3] focuses on 3D models they consider a number of different comparisons for that type of content:

- *Mesh Adapter $\rightarrow$ Statistics Extractor $\rightarrow$ Euclidean Measure*
- *Mesh Adapter $\rightarrow$ Surface Area Extractor $\rightarrow$ Euclidean Measure*
- *Mesh Adapter $\rightarrow$ Spin Image Extractor $\rightarrow$ Euclidean Measure*
- *Mesh Adapter $\rightarrow$ Light Field Extractor $\rightarrow$ Euclidean Measure*

The purpose of utilizing four different comparison methods is in the realization that different content comparisons will have different properties in terms of the differences they pick up on as well as computational requirements. Results obtained from these comparisons allowed for an optimal format, in terms of least information loss when when converted to by

	manually labelled	manually thresholded
manually labelled	1	0.9999999990613203
manually thresholded	0.9999999990613203	1
Isodata	0.9999913417609221	0.9999913689826113
Maxen	0.9999906283525553	0.9999906593289739
Otsu	0.999991221608019	0.9999912497683915
Canny edge	0.9999999929599045	0.9999999929599043
k-means (k=2)	0.9999911129545518	0.9999911420536064
k-means (k=4)	0.9999999910825415	0.9999999957759406

	manually labelled	manually thresholded
manually labelled	1	0.9999999995306601
manually thresholded	0.9999999995306601	1
Isodata	0.9999956708617197	0.999995684472682
Maxen	0.9999953141543205	0.9999953296426748
Otsu	0.9999956107847444	0.999995624865054
Canny edge	0.9999999964799522	0.9999999964799522
k-means (k=2)	0.9999955564575309	0.9999955710071873
k-means (k=4)	0.9999999955412707	0.9999999978879702

	manually labelled	manually thresholded
manually labelled	0	2
manually thresholded	2	0
Isodata	192.0807122019283	191.7785180879235
Maxen	199.83743393068278	199.50689211152581
Otsu	193.40889328053143	193.09842050104916
Canny edge	5.477225575051661	5.477225575051661
k-means (k=2)	194.6021582614129	194.28329830430613
k-means (k=4)	6.164414002968976	4.242640687119285

TABLE I
Top: *Proximities obtained from the Jaccard measure. A value of one indicates two identical images.* **Middle:** *Proximities obtained from the Dice measure. A value of one indicates two identical images.* **Bottom:** *Proximities obtained from the Euclidean measure. A value of zero indicates two identical images.*

other formats, to be chosen according to a particular situations preservation requirements.

C. Cell Biology: Image Segmentation Accuracy Evaluations

In cell biology, large volumes of microscopy images are generated with the goal of extracting cell measurements. One of the basic software-based measurement steps involved is cell segmentation which precedes any extraction of cell characteristics or cell tracking. Reference cell image segmentations are established by manual means. Due to the large volume of microscopy images, a variety of machine image segmentation methods have been developed [5]. Accuracies and sensitivity to parameters have to be evaluated by comparing manual and machine segmentation outcomes. We study the accuracy of several machine segmentation methods as a function of a chosen comparison method:

- *BufferedImage Adapter → Grayscale Histogram Extractor → Jaccard Measure*
- *BufferedImage Adapter → Grayscale Histogram Extractor → Dice Measure*
- *BufferedImage Adapter → Grayscale Histogram Extractor → Euclidean Measure*

The range of similarity values for Jaccard and Dice [6] measures is from zero (very different images) to one (identical images). The Euclidean L2 measure is a standard mathematical distance ranging from zero (identical images) to infinity (dissimilar images). Based on results comparing manually segmented images and images automatically segmented via a variety of methods we see that the Euclidean based comparison shows the greatest distinction and from this are able to conclude that the highest similarity to the manual segmentation is a manual intensity thresholding, followed by k-means segmentation with $k = 4$, followed by Canny edge methods.

IV. Conclusion

We believe that a framework such as Versus could serve as the basis for a variety of applications requiring the comparison of digital content, from content based retrieval, to digital curation, to experimental evaluations with ground truth data. The ability to share adapters, extractors, and measures could also enable reuse of existing methodologies across application domains. Current and future work include the ability to provide decision support to users for selecting methods based on application specific data and previous executions as well as enhancing the web service with respect to performance and fault tolerance.

Acknowledgments

This research at NCSA/UIUC has been funded through the National Science Foundation Cooperative Agreement NSF OCI 05-25308 and Cooperative Support Agreement NSF OCI 05-04064 by the National Archives and Records Administration (NARA). The research at NIST has been conducted as a part of the Computational Science for Biological Metrology (CS-BIO-MET) project. We would like to acknowledge all members of the CS-BIO-MET project for their contributions.

References

[1] L. Diesendruck, L. Marini, R. Kooper, M. Kejriwal, and K. McHenry, "A Framework to Access Handwritten Information within Large Digitized Paper Collections," *IEEE eScience*, 2012.

[2] R. Manmatha, C. Han, and E. Riseman, "Word Spotting: A New Approach to Indexing Handwriting," *IEEE Conference on Computer Vision and Pattern Recognition*, 1996.

[3] K. McHenry, R. Kooper, and P. Bajcsy, "Towards a Universal, Quantifiable, and Scalable File Format Converter," *IEEE eScience Conference*, 2009.

[4] K. McHenry, R. Kooper, M. Ondrejcek, L. Marini, and P. Bajcsy, "A Mosaic of Software," *IEEE eScience Conference*, 2011.

[5] A. Dima, J. Elliott, J. Filliben, M. Halter, A. Peskin, J. Bernal, M. Kociolek, M. Brady, H. Tang, and A. Plant, "Comparison of segmentation algorithms for fluorescence microscopy images of cells," *Cytometry Part A*, 2011.

[6] S.-H. Cha, "Taxonomy of nominal type distogram distance measures," *American Conference on Applied Mathematics*, no. 2, pp. 325–330, 2008.