# Contest on Graph Embedding for Pattern Recognition

M. Vento
mvento@unisa.it

P. Foggia
pfoggia@unisa.it

November 30, 2009

**Abstract**

This document describes the data files and application requirements for the participants to the contest on Graph Embedding on Pattern Recognition, hosted by the 20th International Conference on Pattern Recognition (ICPR2010).

## 1 Introduction

This document is aimed at providing the researchers willing to participate to the Contest on Graph Embedding for Pattern Recognition with all the technical information needed to use the datasets distributed for the contest and to produce the application output in the right format for performance evaluation. For more information about the contest motivations, please refer to the web site of the contest [1].

The rest of this document is organized as follows: in section 2 there is a short description of all the files in the archive given to the contestants. In section 3 are detailed the requirements the software must satisfy to be compatible with the evaluation procedure. Section 4 describes the format of the input files, while section 5 describes the format of the output file the software must produce. The performance index is presented in section 6. Finally, section 7 describes the scripts provided in the archive to allow the contestants to tune their algorithms.

## 2 Provided files

The archive you have already unpacked (in order to read the present document) contains the following files:

- `readme.pdf`: the document you are reading.
- `aloi-1.grf`, `coil-1.grf`, `odbk-1.grf`: the files containing the graphs extracted from three image databases: the Amsterdam Library of Object Images (ALOI) [2], the Columbia Object Image Library (COIL) [3] and the Object Databank (ODBK) [4]. The format of these files is described in section 4.

- `aloi-1.gtr`, `coil-1.gtr`, `odbk-1.gtr`: the ground truth files, used by the script that computes the performance index.
- `perfomance_index.py`: the script that computes the performance index, documented in section 7.
- `simple_explicit.py` and `simple_implicit.py`: two scripts that compute a very simple graph embedding, provided as a reference for the output file format; the scripts are documented in section 7.

# 3 Application requirements

Contest participants must submit a *console application*, that receives on the command line the name of its input file and of its output file (the file formats are detailed in sections 4 and 5). For instance, supposing the name of the application is `grfembed`, it will be invoked using the operating system shell as follows:

```
grfembed  aloi-1.grf  aloi-1.out
```

In this example, the application is expected to take its input graphs from the file `aloi-1.grf` and to store the computed embeddings in the file `aloi-1.out` .

It is possible, if the authors desire so, to add some other command line arguments before the name of the input and output files, e.g. for specifying some parameter setting. It is also possible to require different parameters to be provided for each of the three databases. The authors should clearly state if this is the case, and the exact value of the arguments (no tuning or adjustment will be performed by the contest organization).

Please remember that for the contest evaluation, three different sets of graphs will be used from the ones provided in this archive, although they will be extracted from the same databases.

The application should be submitted in binary form for one of the following operating systems:

- Windows XP, on a 32 bit X86 processor;
- Linux 2.6.x, on a 32 bit X86 processor; the tests will be performed using the Ubuntu Linux distribution;
- Mac OS X 10.5, on a 32 bit X86 processor.

If you cannot provide a program executable on any of these platforms, please contact us to find an arrangement (if possible) for a different experimental setup.

Please take note that, since the program can either perform an explicit embedding or an implicit one, and the output format is different for the two cases, you should clearly state which kind of embedding is performed by your application.

# 4 The input file format

The input file contains a set of graphs from a single image database. The file is in text format, using the ASCII character encoding. The line termination

follows the Unix convention (a single Line-Feed, code 10, marks the end of the line). Within each line of the file, the different fields are separated by ASCII spaces (code 32).

Within the file, the data for the graphs in the set are written sequentially. Each graph has the following format:

<header line>
<number of nodes $n$>
<attributes of node 0>
...
<attributes of node $n-1$>
<number of edges of node 0>
<first edge of node 0>
...
<last edge of node 0>
<number of edges of node 1>
<first edge of node 1>
...
<last edge of node 1>
...
<number of edges of node $n-1$>
<first edge of node $n-1$>
...
<last edge of node $n-1$>

where <header line> is a line of the form:

`G` $id$

with $id$ equal to an integer uniquely identifying the graph within the dataset.
<attributes of node $i$> are lines of the form:

$i \quad s_i \quad r_i \quad g_i \quad b_i$

with $i$ a progressive number identifying the node (starting from 0), and $s_i$, $r_i$, $g_i$, $b_i$ are node attributes encoding the size and the average color of the image area represented by this node.
<number of edges of node $i$> is the number of edges $(i, j)$ with $i < j$. The graphs are undirected, so the edge $(i, j)$ is to be considered equivalent to the edge $(j, i)$; for the sake of compactness of the input files, only the edges with $i < j$ are explicitly represented.
<$k$-th edge of node $i$> are lines of the form:

$i \quad j$

where $i < j$, and $(i, j)$ is an edge of the graph.

As an example, consider the representation of a completely connected graph with four nodes, with an identifying number 1742:

```
G 1742
4
0 0.44 33 77 99
1 0.17 111 87 10
2 0.22 43 10 21
3 0.11 11 39 18
3
0 1
0 2
```

```
0 3
2
1 2
1 3
1
2 3
0
```

# 5 The output file format

As stated in the contest description, the contest participants may provide either an *explicit* embedding, producing a vector for each graph, or an *implicit* embedding, where for each pair of graphs $g_1$ and $g_2$, the value of their scalar product is provided. Consequently, the format of the output file is different for the two cases.

In either case, however, the output must be a text file using the ASCII encoding; for the line termination, the software may either follow the convention of the operating system it runs on, or the Unix convention of using a single Line Feed character (code 10). Within each line of the file, the different fields are separated by ASCII spaces (code 32).

## 5.1 Explicit embedding

In the case of explicit embedding, the output file must contain one line for each graph in the input file. The format of these lines must be:

$id \;\; x_1 \; \ldots \; x_k$

where $id$ is the identifier of the graph, and $x_1 \ldots x_k$ are the components of the vector embedding the graph. Each $x_i$ must be formatted according to the C language rules for the input of floating point numbers.

The dimension $k$ of the vectors can be freely chosen by the software, with the only constraint that all the vectors in the output file must have the same dimension.

## 5.2 Implicit embedding

For the implicit embedding, for each pair of graphs $g_i$ and $g_j$ (including the case $i = j$) a scalar product must be computed. Each product is represented by a line in the file.

Since the scalar product must be commutative by definition, for the sake of compactness only the cases with $i \leq j$ are explicitly stored in the file.

The format of the lines is:

$i \;\; j \;\; p_{ij}$

where $i$ and $j$ are the identifiers of the two graphs (and $i \leq j$), and $p_{ij}$ is the corresponding scalar product; $p_{ij}$ must be formatted according to the C language rules for the input of floating point numbers.

# 6 The performance index

In order to measure the performance of an embedding algorithm, we consider the dataset divided in classes on the basis of the object represented in the original image, and then use a clustering validation index to evaluate the separation between the classes when represented by the vectors explicitly or implicitly produced by the algorithm.

More specifically, we compute first the distances $d_{ij}$ between each pair of graphs; for the explicit embedding, $d_{ij}$ is the Euclidean distance, while for the implicit embedding the distance is computed as follows:

$$d_{ij} = \sqrt{p_{ii} + p_{jj} - 2p_{ij}}$$

where $p_{ij}$ is the scalar product between $g_i$ and $g_j$.

Given the distances, the *C index* by Hubert and Schultz [5] is computed. The C index is defined as follows: first we compute the set $S_w$ of the distances $d_{ij}$ such that $g_i$ and $g_j$ lie in the same class; $M$ is the cardinality of $S_w$. Then, the sets $S_{min}$ and $S_{max}$ are computed taking respectively the $M$ shortest distances and the $M$ largest distances among all the possible values of $d_{ij}$. Finally, the index is computed as:

$$C = \frac{sum(S_w) - sum(S_{min})}{sum(S_{max}) - sum(S_{min})}$$

Notice that the smaller the value, the better is the separation of the classes; the index value is in the interval $[0, 1]$ reaching 0 in the ideal case in which all the inter-class distances are smaller than all the intra-class distances.

The C index has been chosen on the basis of the following considerations:

- it requires only the distances $d_{ij}$, and not the centroids of each class, that would not be trivial to define for the implicit embeddings;
- the value does not change if all the distances are multiplied by a same positive costant, so it is independent of the scale used for the vectors or the products;
- it provides an integral measure that is not significantly affected by outliers.

Since we have three datasets, a way to combine the three corresponding indices must be defined in order to obtain a single figure to evaluate an algorithm. We have decided to use the geometric mean of the indices for attributing a ranking to each algorithm.

Please remember that for the contest evaluation, three different sets of graphs will be used from the ones provided in this archive, although they will be extracted from the same databases. The actual test datasets will be disclosed only at the moment of the contest evaluation.

# 7 Software tools

In order to allow the participants to tune their algorithms and to check the correctness of the file formats, we have included three scripts. They have been realized using the Python programming language, and require Python 2.5 or 2.6 (they do not work with Python 3.x).

### 7.1 The `performance_index.py` script

The `performance_index.py` script computes the performance index given the output file produced by the embedding algorithm and the ground truth file for the graphs.

For explicit embedding, the script must be invoked as follows:

`python performance_index.py` *vector-file ground-truth-file*

where *vector-file* is the file output by the embedding algorithm and *ground-truth-file* is the file containing the ground truth for the used dataset.

For instance, if the output file for the ALOI dataset has been called `aloi-1.out`, the script can be invoked as:

```
python performance_index.py aloi-1.out aloi-1.gtr
```

For implicit embedding, the script must be invoked as follows:

`python performance_index.py -implicit` *prod-file ground-truth-file*

where *prod-file* is the file output by the embedding algorithm and *ground-truth-file* is the file containing the ground truth for the used dataset.

For instance, if the output file for the ALOI dataset has been called `aloi-1.out`, the script can be invoked as:

```
python performance_index.py -implicit aloi-1.out aloi-1.gtr
```

Be aware that the algorithm used to compute the performance index keeps the whole distance matrix in memory, so this script must be run on a computer with a substantial amount of memory (say 1 GB).

### 7.2 The `simple_explicit.py` and `simple_implicit.py` scripts

The two scripts `simple_explicit.py` and `simple_implicit.py` implement a trivial form of embedding, provided in order to check the format of the output files and to test the computation of the performance index.

Namely, `simple_explicit.py` produces an explicit embedding in which for each graph a two-components vector is generated, having as its components the number of nodes and the number of edges in the graph. On the other hand, `simple_implicit.py` produces an implicit embedding, using as the scalar product between two graphs the product of their number of nodes.

Both the scripts require as their parameters the name of the input file and the name of the output file. An example invocation could be:

```
python simple_explicit.py coil-1.grf coil-1.out
python simple_implicit.py odbk-1.grf odbk-1.out
```

## References


[1] Contest on Graph Embedding for Pattern Recognition, `http://nerone.diiie.unisa.it/contest`

[2] Amsterdam Library of Object Images, from the Research Group on Intelligent Sensory Information Systems (ISIS) of the University of Amsterdam, `http://staff.science.uva.nl/~aloi/`

[3] Columbia Object Image Library (COIL-100), Computer Vision Laboratory (CAVE), Columbia University, http://www.cs.columbia.edu/CAVE/software/softlib/coil-100.php

[4] Object Databank, Center for the Neural Basis of Cognition (CNBC), Carnegie-Mellon University, http://www.cnbc.cmu.edu/tarrlab/stimuli/objects/index.html

[5] L. Hubert and J. Schultz, Quadratic assignment as a general data-analysis strategy, British Journal of Mathematical and Statistical Psychology, vol. 29, pp. 190–241, 1976.