Universidade de São Paulo

Instituto de Ciências Matemáticas e de Computação

Projection Explorer (PEx) Manual

Version 1.5.9

Fernando Vieira Paulovich

Supervisor: Prof[a]. Dr[a]. Rosane Minghim

May 17, 2008

Contents

List of Figures

1 Introduction

Projection Explorer (PEx) [6][1] is a Java based tool which can be used to create and explore visual representations of multi-dimensional data sets, specially document collections, helping the user to understand similarity relationships which may occur amongst the multi-dimensional objects. In these representations, each multi-dimensional object is identified as a circle on a plane. If two circles are close, the objects they represent are similar. If the circles are far apart the objects are uncorrelated.

PEx can be used to analyze any kind of multi-dimensional data set, been only necessary to provide a way to calculate the dissimilarities between the objects (instances) of the data set. In addition, the structure of the source code makes it easier to include new different dissimilarities, dimensionality reduction and projection techniques (it is also included on PEx some mechanisms to evaluate the new dissimilarity and the results of employing it to generate projections). To understand some of the techniques implemented inside PEx consult [6, 5, 1, 4, 2, 8, 3, 7].

It is a part of the PhD of Fernando Vieira Paulovich[2] (supervised by Prof[a]. Dr[a]. Rosane Minghim) at Instituto de Ciências Matemáticas e de Computação (ICMC) of the Universidade de São Paulo (USP), São Carlos/Brazil, and it is freely available to educational purpose.

2 Installing and System Requirements

In order to install and run the *Projection Explorer* (PEx), decompress the proper zip file and execute a double click over **run.bat**.

It is necessary to be installed on the machine the JDK 1.5.0 (`http://www.java.sun.com`) or an earlier version. Curretly, the JDK 1.6.* versions present a bug when opening files. In this case it can take a long time to select a file. Also, it is necessary to have at least 512 MB of RAM. If you computer has less than that, change the **run.bat** script. For instance, if you machine has 256 MB, change it to:

```
java -Xmx256m -Xms256m -Djava.library.path=./lib -jar ProjectionExplorer.jar
```

One of the available projection techniques (*Least-Square Projection - LSP* [5]) solve a sparse linear system. For large systems it employs a library that, on the current version, is only available for windows machines. For small systems it uses a Java based solver, but it can be very slow.

3 Data Sources Types

There are four different types of *Data Sources* (DS) which can be used to create a projection: Corpus; Google; Points File; and Distance Matrix File.

The **Corpus** data source is a compressed file (in the zip format) containing text documents (ASCII format); currently PEx does not support other file formats, as RTF or PDF, but there is a feature inside it to convert PDF to ASCII. If the files are pre-classified it is possible to automatically color the documents on the final map according to its class. To do that, the first two letters of each document must identify its class. For example, if a file is

[1] see `http://infoserver.lcad.icmc.usp.br/infovis2/PEx`

[2] see `http://fpaulovich.googlepages.com/`

named as `CB_filename.txt`, every other file with `CB` at the beginning will be colored with the same color. This coloring is created as a *cdata* scalar on the projection.

The second type is the **Google** data source. This DS provides the possibility to execute a query at the Internet, using an API provided by Google[3], and creates a map from the results.

The **Points File** data source is a file which contains points on a n-dimensional space. And The **Distance Matrix File** data source is a file which contains distances (disssimilarities) between objects. The formats of both files are discussed on the next section.

4 File Formats

4.1 Points File

There are two different formats for the points file: a dense and a sparse representation. Both formats starts with four lines composing the file header:

- **1st Line:** composed by two letters, one identifying if it is a sparse or dense representation (S or D, respectively), and the other one indicating if the file presents or not a class value (Y or N, respectively). The class value is a float number to identify the class which a point belongs to (used to color them on the visual representation), and it is the last column on the points representation;
- **2nd Line:** the number of points;
- **3rd Line:** the dimensionality of the points;
- **4th Line:** the identification (name) of each dimension (attribute) separated by semicolons. The number of attribute names must match the dimensionality of the points. Otherwise the file is rejected. If you do not have the attributes names, leave a blank line (in this case the check between the number of names and dimensionality is ignored).

The remaining lines represent the points and they are different depending on the file format. The next subsections detail how to represent a sparse or a dense point, showing the resulting files of the following matrix (representing a six 5-dimensional data set). Observe that the number of points (one point per line) must match the "the number of points" on the second line of the header, otherwise the file will be rejected.

	D0	**D1**	**D2**	**D3**	**D4**
P0	0.0	0.0	0.0	0.0	0.0
P1	1.0	0.4	0.5	0.0	0.0
P2	0.0	0.0	2.3	1.0	0.0
P3	3.2	0.1	0.7	1.3	2.0
P4	0.5	0.0	0.0	0.2	0.0
P5	0.3	0.0	0.0	0.0	0.3

Figure 1: Example of points which is used to generate the sparse and dense representations on the following sections. The **P*** are identifiers of the points and **D*** are the names of the attributes.

[3] see `http://www.google.com/apis`

4.1.1 Dense Representation

On the dense representation, a point starts with its unique identification (string), follows with its coordinates (float), and finishes with the class identification (float), if it exists. These values are separated using semicolons. Figure 2 shows the dense representation of Figure 1, considering that there is not a class identification.

```
DN
6
5
D0;D1;D2;D3;D4
P0;0.0;0.0;0.0;0.0;0.0
P1;1.0;0.4;0.5;0.0;0.0
P2;0.0;0.0;2.3;1.0;0.0
P3;3.2;0.1;0.7;1.3;2.0
P4;0.5;0.0;0.0;0.2;0.0
P5;0.3;0.0;0.0;0.0;0.3
```

Figure 2: Dense representation without class identification.

Figure 3 presents the dense representation supposing that the first 3 points belong to the class “1.0” and that the last 3 belong to the class “1.25”. In this case, the last column is the class identification.

```
DY
6
5
D0;D1;D2;D3;D4
P0;0.0;0.0;0.0;0.0;0.0;1.0
P1;1.0;0.4;0.5;0.0;0.0;1.0
P2;0.0;0.0;2.3;1.0;0.0;1.0
P3;3.2;0.1;0.7;1.3;2.0;1.25
P4;0.5;0.0;0.0;0.2;0.0;1.25
P5;0.3;0.0;0.0;0.0;0.3;1.25
```

Figure 3: Dense representation with class identification.

Observe that the class identification is not used to create the projection, it is only employed to color the points on the visual representation, resulting on a scalar called **cdata**.

4.1.2 Sparse Representation

On the sparse representation, a point starts with its unique identification (string) and finishes with the class identification (float), if it exists. However, in this case the coordinates are composed by two values separated using colons. The first is the dimension (integer starting on zero) and the second is the value of this dimension. Only non-zero values are represented. As on the dense representation, these pair of values are separated using semicolons. Figure 4 shows the dense representation of Figure 1, considering that there is not a class identification.

Figure 5 presents the sparse representation supposing that the first 3 points belong to the class “1.0” and that the last 3 belong to the class “1.25”. In this case, the last column is the class identification.

Observe that the class identification is not used to create the projection, it is only employed to color the points on the visual representation, resulting on a scalar called **cdata**.

```
SN
6
5
D0;D1;D2;D3;D4
P0;
P1;0:1.0;1:0.4;2:0.5
P2;2:2.3;3:1.0
P3;0:3.2;1:0.1;2:0.7;3:1.3;4:2.0
P4;0:0.5;3:0.2
P5;0:0.3;4:0.3
```

Figure 4: Sparse representation without class identification.

```
SY
6
5
D0;D1;D2;D3;D4
P0;1.0
P1;0:1.0;1:0.4;2:0.5;1.0
P2;2:2.3;3:1.0;1.0
P3;0:3.2;1:0.1;2:0.7;3:1.3;4:2.0;1.25
P4;0:0.5;3:0.2;1.25
P5;0:0.3;4:0.3;1.25
```

Figure 5: Sparse representation with class identification.

4.2 Distance Matrix File

The **Distance Matrix** data source is a file which contains all pairwise distances (similarities) between objects. This file starts with 3 lines containing:

- **1st Line:** the number of multi-dimensional objects;
- **2nd Line:** the unique identification (string) of each object separated by semicolons. The order of these ids are used for further processing inside PEx, thus the id of the first object is the first to appear, the second one is the second to appear, and so on. The number of object ids must match the number of objects, otherwise the file is rejected;
- **3rd Line:** the class identification (float) of each object (if it does not exist put 0 for each object). This class identification is used to color the points, it is not used on the projection process. The number of class ids must match the number of objects, otherwise the file is rejected, and the order must follow the objects ids order.

The remaining lines contain a lower triangular distance matrix without the diagonal ($\frac{1}{2}n(n-1)$ distances for n objects), with the values separated by semicolons. Figure 6 shows the format of a generic distance matrix file. In this figure, **D(i,j)** represents the distance between the objects **i** and **j**, **Pi** is the unique id of object **i** (string), and **Ci** represents the class id (float) associated with the object **i**.

```
n
P0;P1;P2;P3;. . .;P(n-1)
C0;C1;C2;C3;. . .;C(n-1)
D(1,0)
D(2,0);D(2,1)
D(3,0);D(3,1);D(3,2)
⋮  ⋮  ⋮  ⋮
D(n-1,0);D(n-1,1);D(n-1,2); . . .; D(n-1,n-2)
```

Figure 6: Distance Matrix file format.

5 Other File Formats

There are other files which can be used on the PEx. These files can be imported into a projection (to change the color the points, the titles associated with the points, the edges between the points, etc.) or exported to be used latter. There are five different files. Following each one is described, detailing its purpose and format.

5.1 Title file format

A title file, with extension **.titles**, is used to add to a projection different titles for the points (the names that are shown when the mouse is rolled over a point). In this file, the first line contains the names of the titles separated by semicolons. The remaining lines starts with the multi-dimensional object unique id (string), followed by the titles values (string), all separated by semicolons. Figure 7 presents an example of title file.

```
title one; title two
P0;title one zero;title two zero
P1;title one one;title two one
P2;title one two;title two two
P3;title one three;title two three
. . .
P(n-1);title one n-1;title two n-1
```

Figure 7: Title file format.

5.2 Scalar file format

A scalar file, with extension **.scalar**, is used to add to a projection different scalar for the points (values which can be used to color/size a point). In this file, the first line contains the names of the scalars separated by semicolons. The remaining lines starts with the multi-dimensional object unique id (string), followed by the scalars values (float), all separated by semicolons. Figure 8 presents an example of scalar file.

```
scalar one; scalar two
P0;0.15;0.0
P1;10.0;0.0
P2;1.35;0.0
P3;2.25;1.0
. . .
P(n-1);2.00;0.0
```

Figure 8: Scalar file format.

5.3 2D points file format

A 2D points file, with extension **.prj**, is a file which contains the 2D coordinates of a projection. This file is basically a dense points file (see Section 4.1.1) with a class identification, containing the coordinates **x** and **y** of each point on the projection. Figure 9 presents an example of 2D points file.

```
DY
n
2
x;y
P0;0.15;0.0;0.1
P1;10.0;0.0;0.1
P2;1.35;0.0;2.0
P3;2.25;1.0;2.0
...
P(n-1);2.00;0.0;2.0
```

Figure 9: 2D points file format.

5.4 Connectivity file format

A connectivity file, with extension **.con**, is a file which contains the edges attaching different points on the projection. In this file, the first line contains the connectivity name (string). Each remaining line represents an edge, and is composed by three values separated by semicolons, the id of the source object, the id of the target object, and the size of the edge. Figure 10 presents an example of connectivity file.

```
connectivity
P0;P1;0.1
P0;P3;0.5
P2;P1;2.0
P3;P2;2.0
P3;P1;0.5
...
P(n-1);P3;2.0
```

Figure 10: Connectivity file format.

The edges are undirected, thus adding an edge from **P0** to **P1** is equal to adding a edge from **P1** to **P0**.

5.5 Legend file format

A legend file, with extension **.leg**, is used to create a legend about the classification data of the multi-dimensional objects. It is used to show the relationship between the colors of the points, which reflect a given classification, and the name of the classes. In this file, the first line contains the name of the legend, and the remaining lines contain the class names and their values. These values must be the same as the class identification used on the points and distance matrix files (see Section 4). Figure 11 presents an example of a legend file, which reflects the class identification given on the points file of Figures 3 and 5.

```
class
class one;1.0
class two;1.25
```

Figure 11: Legend file format.

References

[1] A. M. Cuadros, F. V. Paulovich, R. Minghim, and G. P. Telles. Point placement by phylogenetic trees and its application to visual analysis of document collections. In *Proceedings of IEEE Symposium on Visual Analytics Science and Technology - IEEE VAST*, pages 99–106, Sacramento - CA, USA, October 2007. IEEE CS Press.

[2] A. A. Lopes, R. Pinho, R. Minghim, and F. V. Paulovich. Visual text mining using association rules. *Computers & Graphics, Special Issue on Visual Analytics*, 31(3):316–326, June 2007.

[3] R. Minghim, F. V. Paulovich, and A. A. Lopes. Content-based text mapping using multi-dimensional projections for exploration of document collections. In Robert F. Erbacher, Jonhathan C. Roberts, Matti T. Gröhn, and Katy Borner, editors, *Proceedings of SPIE-IS&T Electronic Imaging - Visualization and Data Analysis - VDA*, volume 6060, page 60600S, San Jose, California, 2006. SPIE Press.

[4] F. V. Paulovich and R. Minghim. Text map explorer: a tool to create and explore document maps. In *Proocedings of the 10th International Conference on Information Visualisation - IV*, pages 245–251, London - UK, 2006. IEEE CS Press.

[5] F. V. Paulovich, L. G. Nonato, R. Minghim, and H. Levkowitz. Least square projection: a fast high precision multidimensional projection technique and its application to document mapping. *IEEE Transactions on Visualization and Computer Graphics*, 14(3):564–575, May/Jun 2008.

[6] F. V. Paulovich, M. C. F. Oliveira, and R. Minghim. The projection explorer: A flexible tool for projection-based multidimensional visualization. In *SIBGRAPI '07: Proceedings of the XX Brazilian Symposium on Computer Graphics and Image Processing (SIBGRAPI 2007)*, pages 27–36, Washington, DC, USA, 2007. IEEE Computer Society.

[7] E. Tejada, R. Minghim, and L. G. Nonato. On improved projection techniques to support visual exploration of multidimensional data sets. *Information Visualization*, 2(4):218–231, 2003.

[8] G. P. Telles, R. Minghim, and F. V. Paulovich. Normalized compression distance for visual analysis of document collections. *Computers & Graphics, Special Issue on Visual Analytics*, 31(3):327–337, 2007.