DSC 2003 Working Papers
(Draft Versions)
http://www.ci.tuwien.ac.at/Conferences/DSC-2003/

Distributed Storage and Analysis of Microarray Data in the Terabyte Range: An Alternative to Bioconductor

Christian Stratowa *

Abstract

Novel high-throughput technologies such as DNA microarray analyses are allowing biologists to generate sets of data in the terabyte realm. Many of these data will be deposited in the public domain, necessitating a common standard. Currently available database systems are not appropriate for these intentions.

In this paper, I will introduce ROOT (`http://root.cern.ch`), an object-oriented framework that has been developed at CERN for distributed data warehousing and data mining of particle data in the petabyte range. Data are stored as sets of objects in machine-independent files, and specialized methods are used to get direct access to separate attributes of selected data objects. ROOT has been designed in such a way that it can query its databases in parallel on SMP/MPP machines, on clusters of PC's, or using common GRID services.

In order to demonstrate the applicability of ROOT to microarray data, I will present a functional prototype system, called XPS - eXpression Profiling System, which can be considered to be an alternative to the Bioconductor project. The current implementation handles the storage of Affymetrix GeneChip schemes and data, and the pre-processing, normalization and filtering of GeneChip data. Based on this system, I will propose a novel standard for the distributed storage of microarray data.

Finally, I will emphasize the similarities between R and ROOT, and show how R could be easily extended to access ROOT from within R.

[1]Boehringer Ingelheim Austria GmbH, A-1121 Vienna, cs@vie.boehringer-ingelheim.com

1 Introduction

DNA microarray technology allows scientists to monitor the expression of genes on a genomic scale with the promise to provide key insights into gene function and gene regulatory networks. However, management and analysis of the huge amounts of data produced by microarray experiments have become a major bottleneck. Currently, public data are available as flat files or from public database repositories, where they are not easily accessible for statistical analysis. Furthermore, these databases are implemented as relational database management systems (RDBMS), which suffer from a number of limitations when dealing with data in the terabyte range. The major disadvantage is that data are restricted to tables and simple attributes. The semantics of the data are split between the relations and thus maintaining the integrity of the schema is difficult. As the size of the database increases the performance of the system suffers. In contrast, object-oriented database (OODB) systems handle objects rather than data, providing features such as object identification, data abstraction, class structure, encapsulation, inheritance, polymorphism and extensibility [1]. The performance capabilities of OODB systems are considerably higher than RDBMS for scaling to the terabyte regime.

The size and complexity of microarray data, and the need of utilizing biological meta-data proposes the presentation of these data in an object model and the storage of the data as persistent objects in an OODB system. However, microarray data deposited in the public domain, demand decentralized access to these data in a standardized manner. Since the corresponding datasets have usually already been cleaned and validated, an obvious choice is their storage in a distributed data warehouse. Powerful data mining techniques can then be applied to discover hidden patterns and to extract knowledge from microarray data [2].

Considering the ever-increasing amount of microarray data and meta-data, the growing demand for access to large-scale national and international facilities, and the increasing computing requirements for large-scale data mining and analysis, it is proposed that an international standard for storage of microarray data and the corresponding meta-data should be based on ROOT.

2 Why ROOT?

Currently, the European Laboratory for Particle Physics, CERN, is building a new particle accelerator, the Large Hadron Collider (LHC), which is scheduled to commence operation in 2007. Four detectors are built for different experiments, each of which will accumulate 1-8 Petabyte of data per year, giving rise to a total volume of more than 250 PB over the expected twenty-year lifetime of the project. This volume of data exceeds by many orders of magnitude what is traditionally termed a very large database. Today, 1 TB is considered large - one of the experiments at LHC will be storing 1 TB every 10 minutes [3]!

Since the demands of LHC are far beyond the capabilities of any commercial database system, Rene Brun has started in November 1994 to develop ROOT as an object-oriented framework for large-scale scientific data analysis and data mining [4].

Today, ROOT has not only become the de facto standard tool for data storage and analysis in Particle Physics, but is also used in other sciences as well as the financial and medical industries. As an example, MammoGrid, an EU-funded project for remote image analysis and interactive online diagnosis of mammograms, will be based on ROOT.

ROOT is a modular object-oriented framework aimed at solving the data analysis challenges of high-energy physics (HEP). The main features of ROOT are as follows [5]:

- *Architecture*: The ROOT architecture is a layered class hierarchy with over 500 classes divided into different categories (Figure 1). Most of the classes inherit from a common base class `TObject`, which provides the default behavior and protocol for all objects.

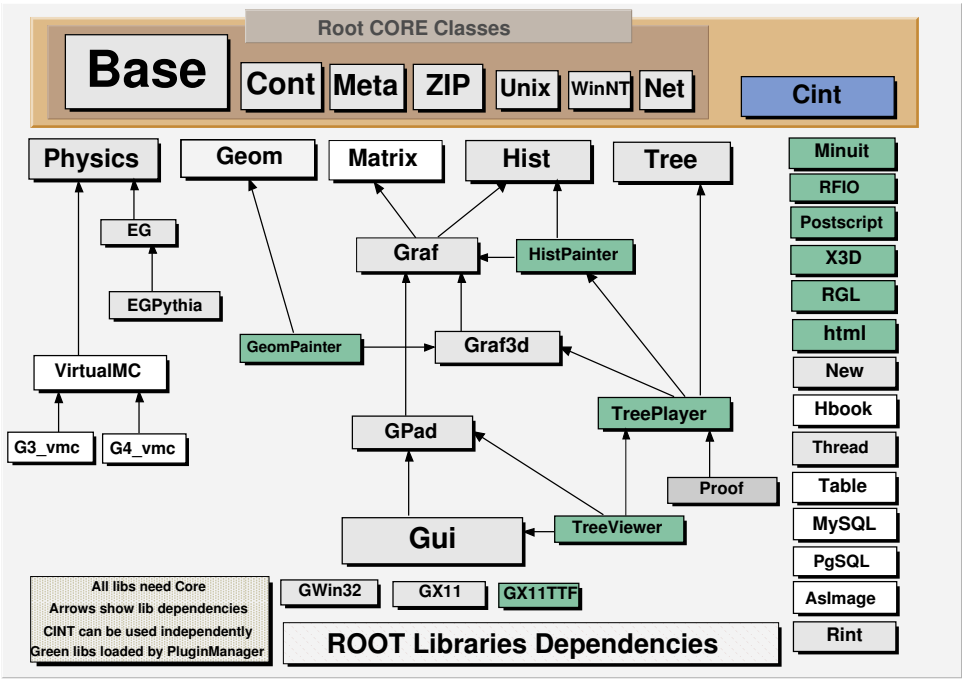


Figure 1: The ROOT classes are organized in different libraries. The diagram shows the library dependencies. Courtesy of Rene Brun.

- *Object Input/Output Facility*: Object input/output is handled by class `TFile`, which has a UNIX-like directory structure and provides a hierarchical sequential and direct access persistent object store. ROOT files store information in a machine independent format and support on-the-fly data compression. Furthermore, ROOT files are self-describing: for every object stored in `TFile`, a

dictionary describing the corresponding class is written to the file. A dictionary generator, called ROOTCINT, parses the class header files and generates a dictionary.

- *Data Trees*: Any object derived from `TObject` can be written to a file with an associated key `TKey`. However, each key has an overhead in the directory structure in memory. To reduce this overhead, a novel concept, called Trees (class `TTree`) has been developed. Trees are designed to support very large numbers of complex objects in a large number of files. A Tree consists of branches (`TBranch`) with each branch described by its leaves (`TLeaf`). Trees allow direct and random access to any entry of a selected subset of branches. Thus, Trees extend and replace the usual data tables. The concept of Tree friends allows the joining of many trees as one virtual tree. However, unlike table joins in an RDBMS, the processing time is independent of the number of tree friends.

- *Folder concept*: One of the major critiques of OODB systems is the problem of database queries. Since connections between objects are usually established via chains of pointers, a querying method has to loop along pointer chains to find the object containing the requested data [1]. To circumvent these limitations, ROOT provides the Folder concept (`TFolder`). A producer class places a pointer to its data into a folder, and the consumer class retrieves a reference to the folder. Thus, the use of folders reduces class dependencies and improves modularity.

- *Statistical analysis*: The current implementation of ROOT supports the statistical analysis of HEP-specific data. Histogram and minimization classes offer statistical data analysis features, like 1D, 2D and 3D histograming, profile histograms, data fitting, formula evaluation and minimization. Fitting in ROOT is based on the well-known minimization package MINUIT (class `TMinuit`), which is able to fit data using pre-defined or user-defined functions.

- *ROOTD*: Remote database access supports the construction of distributed data warehouses: The multithreaded `rootd` daemon manages LAN and WAN data access (`TNetFile`), while the addition of a ROOT specific module to the Apache web server allows the distribution of ROOT files to any ROOT user (`TWebFile`).

- *PROOF*: GRID computing is supported by the Parallel ROOT Facility PROOF, which implements the master/slave concept. However, the slave servers are the active components that ask the master server for new work whenever they are ready. The PROOF GRID interface (`TGrid`) allows the use of a Grid resource broker, Grid file catalogue and replication manager, and the Grid monitoring services.

- *CINT*: CINT is an interactive C/C++ interpreter, which is aimed at processing C/C++ scripts, called macros [6]. Currently, CINT covers 99% of ANSI C and 95% of ANSI C++. CINT offers a gdb-like debugger for interpreted

programs and allows the automatic compilation of scripts using ACLiC, the automatic compiler of libraries for CINT. Although available as independent program, CINT is embedded in ROOT as command line interpreter and macro processor, as well as dictionary generator.

- *User interaction*: The ROOT system can be accessed from the command line, by writing macros, or via a graphic user interface (e.g. ObjectBrowser, TreeViewer). Furthermore, it is possible to write libraries and applications. The ROOT GUI classes allow the development of full-featured standalone applications.

- *Platform independence*: The ROOT system is available for most platforms and operating systems, including Linux, MacOS X, and the major flavors of UNIX and Windows. ROOT and ROOT-derived applications can be compiled for any supported platform on 10 different compilers.

- *SQL interface*: ODBC access to SQL compliant systems such as Oracle is implemented. Since ROOT is currently not a full featured database, a composite approach has been adopted as ROOT database model: Write-once data are put in ROOT files as object store, and an RDBMS is used to store the corresponding Run/File catalog.

Other features of ROOT include its own collection classes, physics vector classes, a matrix package, a geometry package, support for 2D and 3D graphics, networking classes to build client/server applications, and documentation classes for automatic document generation.

3 XPS - eXpression Profiling System

After reviewing different technologies for large-scale expression profiling [7], two topics were obvious for me: i) DNA-chips will become the standard technology for expression profiling, and ii) management and analysis of the huge amounts of data produced by DNA-chips will become a major bottleneck. Being aware of the immense amounts of data generated by particle detectors, and the challenges of the LHC project on data handling and analysis, I came across ROOT, which is aimed at solving these problems. In December 1999 I started to build a first prototype system to test the feasibility of using ROOT as framework for the storage of microarray data. After the successful implementation of a prototype, I started to develop XPS as system for large-scale microarray data storage and analysis. In the following I will describe the current status of XPS, which can be considered to be an alternative to the Bioconductor project [8] for large-scale data mining of expression profiling experiments.

The aim of XPS is to provide a complete software environment for data warehousing and data mining of microarray data in the terabyte range. Features of XPS will include: distributed data warehousing, local and remote data access, interactive and remote data analysis, tools for statistical analysis, ability to write simple

or complex macros, an intuitive graphical user interface, support for multiple microarray technologies, platform independency.

The current implementation of XPS handles the storage of Affymetrix GeneChip schemes and data, and the pre-processing, normalization and filtering of GeneChip data. The general concept of XPS is outlined in Figure 2.

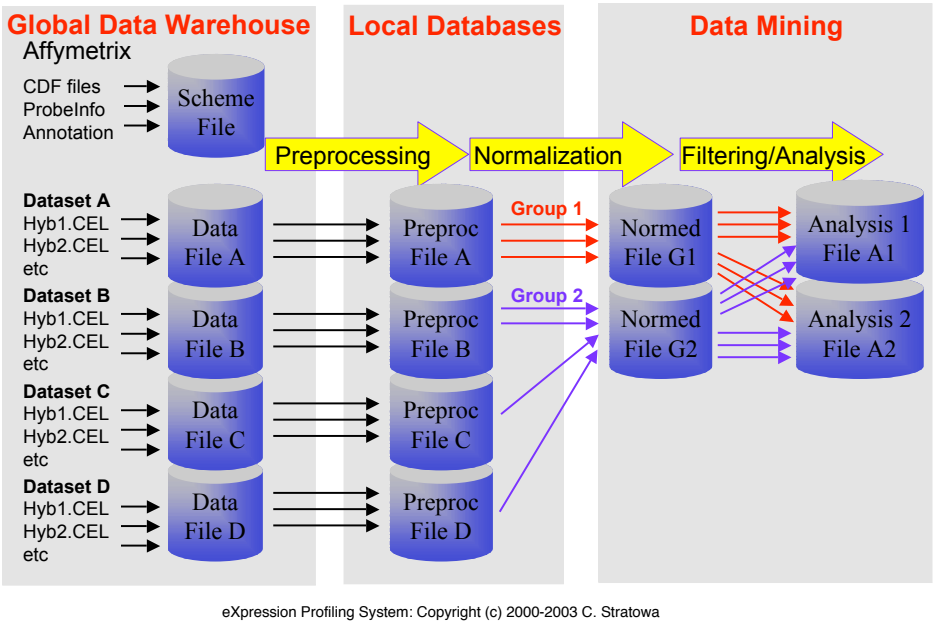


Figure 2: Concept and workflow for storage and analysis of microarray data.

All data are stored in portable ROOT files. Data describing the physical microarray layout, the probe information and the metadata for genes are stored as ROOT Trees in the *Scheme File*. Only this file needs to be updated regularly to include new and updated gene annotation data. Any microarray project consists of at least one dataset. The tabular raw data for every dataset are stored as ROOT Trees in a *Data File*. The collection of all data files can be considered to be the data warehouse. The necessary "File Catalog" could be easily implemented as an RDBMS, which would also contain the MIAME-compliant dataset information [9]. In this way, microarray data deposited in the public domain as data files will assemble a distributed global data warehouse, which will store unlimited amounts of data in a compressed, machine independent format. Every laboratory/user can access these data, apply the preferred pre-processing algorithms (e.g. for background correction, condensation), and store the processed data in a local database as Trees in *Preprocess Files*. Data Trees to be analyzed together will be normalized using an algorithm of choice, and can be stored as groups in *Normed Files*. Normalized data Trees can then be subjected to gene filtering and further analysis.

XPS is built as a modular system with the different modules reflecting the workflow of a typical microarray experiment. This is outlined in Figure 3.

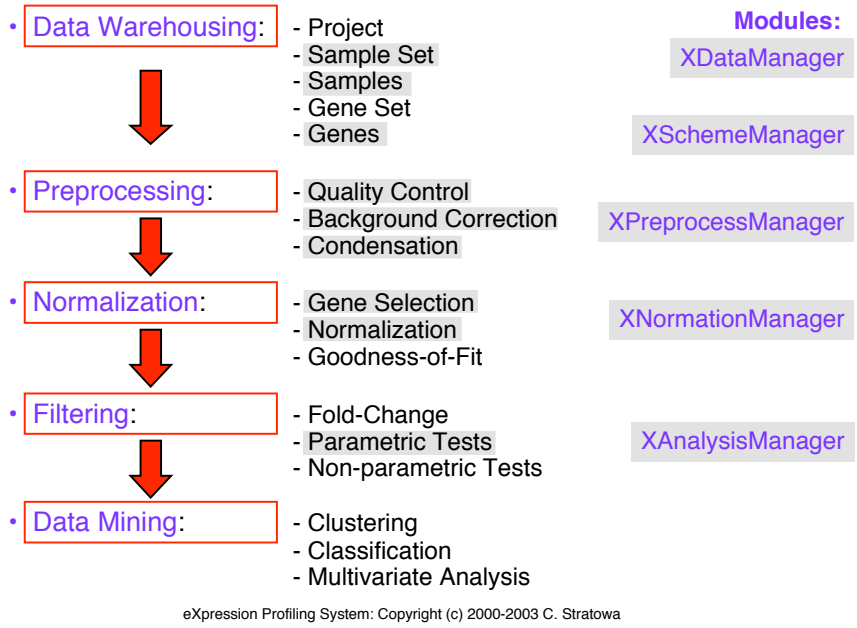


Figure 3: The modular structure of XPS reflects the workflow of a typical microarray experiment. (Items highlighted in gray indicate the current implementation status.)

`XDataManager` is responsible to store the microarray measurements for every sample, i.e. the results of the chip hybridizations, as Trees in data files, while `XSchemeManager` is used to store and update chip layout, probe information and gene annotation. `XPreprocessManager` handles quality control, background correction and data condensation. Selection of non-variant genes, normalization of the selected gene set, and application of the normalization function to all genes is the task of the `XNormationManager`. Finally, gene filtering and data mining is the responsibility of `XAnalysisManager`. All Manager classes inherit their common behavior from the base class `XManager`. This is shown in Figure 4 which outlines the most important classes and their dependencies.

Class `XPlot` provides methods to draw various graphs, and for diagnostic plots such as histograms, boxplots, scatterplots and spatial color images. `XSetting` is a helper class responsible for the parameter settings. Class `XTreeSet` and its derived classes (Figure 5) are the essential classes of XPS: `XDNAChip` imports chip layout, probe information and gene annotation and stores the data in a set of ROOT Trees, called scheme tree, unit tree, probe tree and annotation tree, respectively, for every microarray scheme.

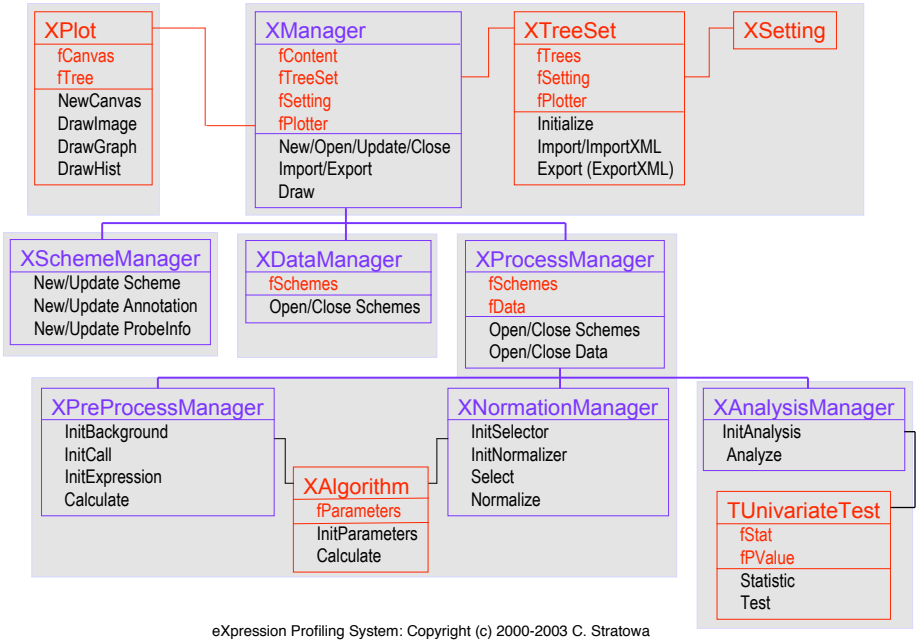


Figure 4: Overview of the main XPS classes and modules. Libraries (gray regions) and their dependencies are displayed.

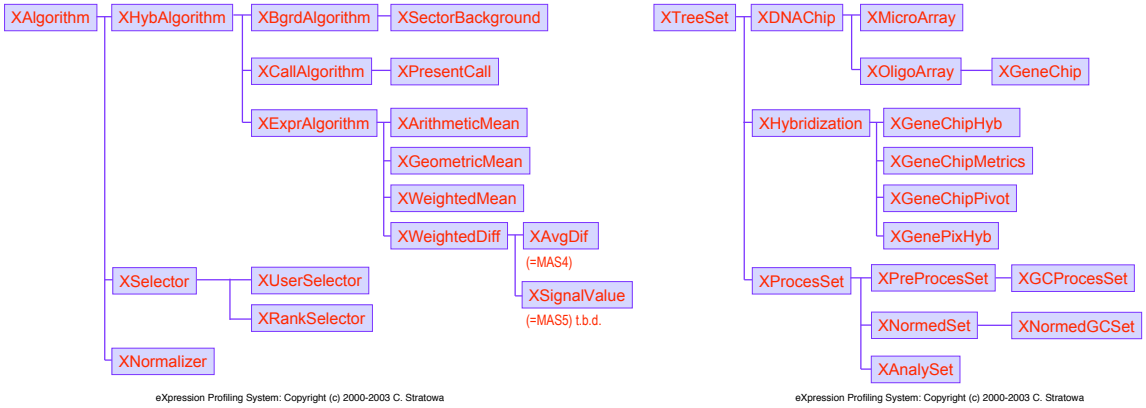


Figure 5: Inheritance diagrams for classes `XAlgorithm` and `XTreeSet`.

`XHybridization` is used to import the hybridization data: Subclass `XGeneChipHyb` imports GeneChip *.CEL files. In this case, the treeset consists of data tree and mask tree, respectively. (In contrast, `XGeneChipMetrics` and `XGeneChipPivot` can be used to import data already processed by the Affymetrix Microarray Suite [10].) `XPreProcesSet` transforms the raw data into the final microarray data by employing the appropriate algorithms. The resultant treeset, namely background tree, expression tree and call tree, is obtained by applying the respective subclass of class `XHybAlgorithm` (see Figure 5). Normalization of the data is done by class `XNormedSet`, which employs i) a descendent of `XSelector` to select non-differentially expressed genes, and ii) `XNormalizer` to subject the selected subset of genes to a normalization method of choice. The resultant normalization parameters are then applied to all data; the corresponding treeset consists of expression tree and mask tree, respectively. The final step, gene filtering and data analysis, is the task of class `XAnalySet`, which calls the appropriate algorithms. Currently, t-test and resampling-based multiple testing methods, as described in Dudoit et al. [11], are implemented in class `TUnivariateTest`.

```
void ImportSchemes()
{
// create new scheme manager
   XSchemeManager *manager = new XSchemeManager("MyManager");

// create new root schemes file
   manager->New("GeneChipCDFs","","GeneChip");

// store chip definitions , probe sequences and annotations
// Hu6800:
   manager->NewScheme("Hu6800","/cdf/Hu6800.CDF");
   manager->InitQC("Hu6800", 6, 7130,7129,7133,7134,7132,7131);
   manager->NewProbeInfo("Hu6800","/cdf/HuGeneFL_probe.tab");
   manager->NewAnnotation("Hu6800","/cdf/Hu6800_annot.txt");

// HG_U95Av2:
   manager->NewScheme("HG_U95Av2","/cdf/HG_U95Av2.CDF");
   manager->InitQC("HG_U95Av2",6,12629,12630,12626,12625,12627,12628);
   manager->NewProbeInfo("HG_U95Av2","/cdf/HG-U95Av2_probe.tab");
   manager->NewAnnotation("HG_U95Av2","/cdf/HG_U95Av2_annot.txt");

// cleanup
   delete manager;
}//ImportSchemes
```

Script 1: Import Affymetrix chip definition, probe information and gene annotation.

XPS is designed as a collection of libraries (Figure 4) extending the set of ROOT libraries. This setup allows interactive usage of XPS from within the ROOT runtime environment, by means of a series of basic commands or by writing simple macros. In addition, these libraries can be wrapped in a graphic user interface for compilation

as a standalone program. In the following paragraphs I will demonstrate how each phase of a typical microarray experiment (Figure 3) can be implemented as a simple macro/script:

Scheme Import: The current implementation of XPS supports the storage of all relevant information for Affymetrix GeneChips in a scheme file. As a first step, the chip definition file (e.g. Hu6800.CDF) is imported and stored in an efficient way as scheme tree and unit tree, respectively. After this step, the CDF-file is no longer needed. (A typical user of XPS will never get in touch with CDF files.) Optionally, the corresponding probe information file (e.g. HuGeneFL_probe.tab) can be downloaded from the Affymetrix NetAffx web-site, and stored as probe tree. Finally, gene annotation data can be stored as annotation tree. Currently, the necessary information has to be supplied as tab-separated text file. Script 1 gives an example how to import the relevant information for GeneChips Hu6800 and HG_U95Av2.

```
void ImportHybridizations()
{
// create new data manager
   XDataManager *manager = new XDataManager("MyManager");

// initialize chip type and variable list
   manager->Initialize("GeneChip");
   manager->InitInput("Hu6800","cel","MEAN/D:STDV/D:NPIXELS/I","RawData");

// create new root data file
   manager->New("MyHybridizations","~/mypath/test","GeneChip");

// open root scheme file
   manager->OpenSchemes("~/path/GeneChipCDFs.root");

// store *.CEL data as tree in data file
   manager->Import("Hyb1","/mypath/data/chip1.CEL");
   manager->Import("Hyb2","/mypath/data/chip2.CEL");
// store *.CEL files exported from MAS5 as *.XML files
   manager->Import("Hyb3","/mypath/mage_data/chip3.XML");
   manager->Import("Hyb4","/mypath/mage_data/chip4.XML");

// cleanup
   delete manager;
}//ImportHybridizations
```

Script 2: Import GeneChip oligonucleotide-array data.

Note that every script/macro consists of a series of simple member function calls to the appropriate manager class: First, the user creates the corresponding manager object. If required, initialization, i.e. the settings of various parameters is done next. A new ROOT file is created, and existent data files are opened. The

actual task is managed as a series of method calls. Finally, the manager object is deleted.

Data Import: Usually, the raw GeneChip data files will be imported and stored as data trees. XPS is able to import the current format *.CEL as well as data exported from MAS5 as *.XML files. Alternatively, it is possible to import the preprocessed metrics files or pivot tables. Script 2 shows how to import CEL files.

Quality Control: Several plotting functions are useful for diagnosing problems with the data. Plotting the image can be used to detect spatial artifacts, while a histogram displays the intensity distribution. Figure 6 provides clear evidence of saturation effects.

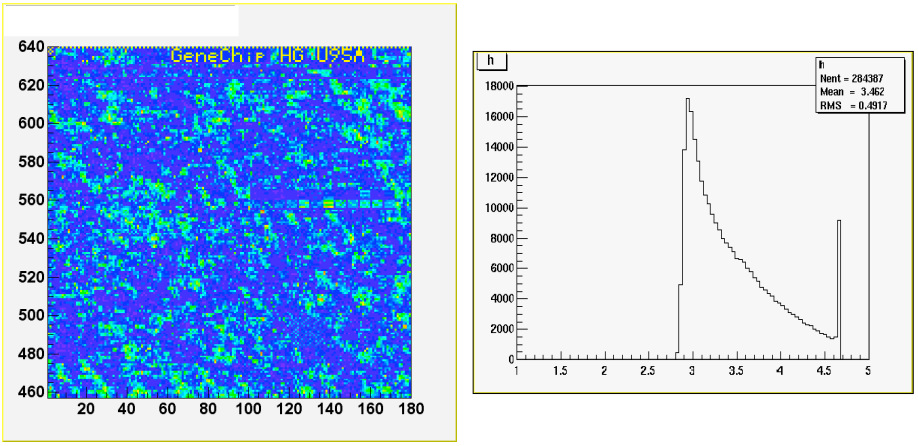


Figure 6: Zoomed image of log intensities and a histogram revealing intensity saturation.

The PM and MM intensities for every single Affymetrix probe set can be displayed as graph, either in the default order, or sorted according to different criteria. Figure 7 offers a hint, that the GC-content of the oligonucleotides presenting a probe pair set may affect the hybridization of the target RNA to each probe.

Preprocessing: The term *preprocessing* summarizes a series of operations necessary to obtain the final expression measures, such as data cleaning, background correction, and, in the case of Affymetrix GeneChips, condensation of probe set values into one expression measure and detection call to indicate whether a transcript is detected. The current background algorithm divides the microarrays into (n x m) sectors, and allows local smoothing. Currently implemented algorithms for calculating an expression measure include arithmetic/geometric trimmed mean of

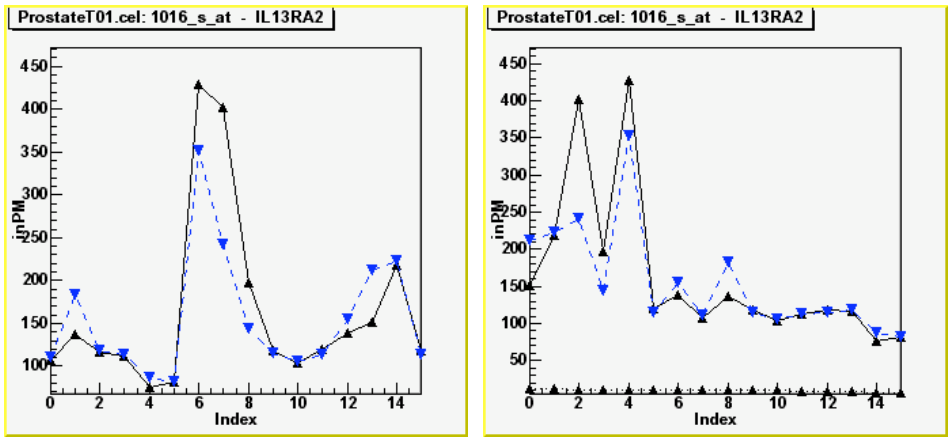


Figure 7: The intensities of the PM (black) and MM (blue) oligonucleotides for probe set 1016_s_at are shown (left) in the default order and (right) sorted according to decreasing GC-content of the probes.

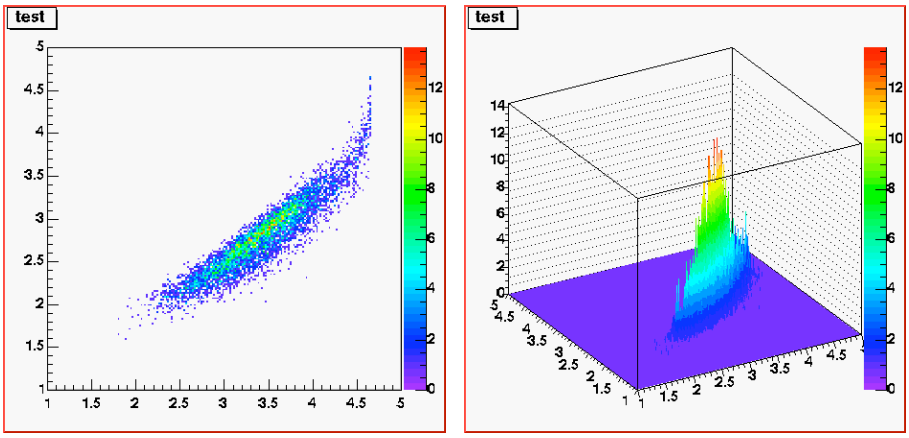


Figure 8: 2D-histogram comparing two arrays, drawn with option COLor (left) or SURFace (right).

PM only, weighted mean of PM or weighted mean of PM-MM, and the AvgDiff algorithm of MAS4, respectively. Script 3 converts probe level data to expression values using the arithmetic trimmed mean algorithm.

```
void PreprocessData()
{
// create new preprocessing manager
   XPreProcessManager *manager = new XPreProcessManager("MyManager");

   manager->Initialize("GeneChip");
//  initialize  preprocessing  algorithms, e.g. arithmetic trimmed mean
   manager->InitExpression("amn", 1, 0.15);

// create new root expression data file
   manager->New("MySamples_amn","~/mypath/test","GeneChip");

// open root scheme file
   manager->OpenSchemes("~/GeneChipCDFs.root");

// open root raw data file for reading only
   manager->OpenData("~/mypath/test/MyHybridizations.root");

// calculate trimmed mean values and store as trees in new file
   manager->Calculate("Hyb1.cel", "Sample1");
   manager->Calculate("Hyb2.cel", "Sample2");
   manager->Calculate("Hyb3.cel", "Sample3");
   manager->Calculate("Hyb4.cel", "Sample4");

// cleanup
   delete manager;
}//PreprocessData
```

Script 3: Calculate expression values from raw data.

Visual examination using scatterplots, MvA-plots and boxplots is helpful to determine whether there is a need for normalization and to select an appropriate algorithm; an example is shown in Figure 8.

Normalization: The comparison of gene expression results across multiple experiments relies crucially on effective normalization algorithms. While initial methods for normalization assumed a linear relationship between experiments, Dudoit et al. [11] were the first to point out the need for non-linear normalization methods. Preferably, these algorithms should be applied only to genes considered to be invariant in the respective experiments. XPS allows the user to select these genes, or supplies a rank-based algorithm. Currently implemented algorithms for normalization include trimmed `mean`, `median`, `ksmooth`, `lowess` and `supsmu` (see Figure 5). The implemented algorithms for gene selection and normalization are described in

[12]. Script 4 reveals how to initialize theses algorithms, select the expression data, normalize the data and store the normalized expression values in a new ROOT file.

```
void Normalize()
{
// create new normalization manager
   XNormationManager *manager = new XNormationManager("MyManager");

// create new normalized expression file
   manager->New("MySamples_sup","~/mypath/test","GeneChip");

   manager->Initialize("GeneChip");
// initialize algorithm used to select genes for normalization
   manager->InitSelector("rank", "separate", 4,0,0.3,400,0);
// initialize normalization algorithm
   manager->InitNormalizer("supsmu", "", "log10", 2,0.0,0.0);

// open root scheme file
   manager->OpenSchemes("~/GeneChipCDFs.root");

// open root expression data file
   manager->OpenData("~/mypath/test/MySamples_amn.root");

// select trees for normalization
   manager->Select("SampleSet","Sample1.amn",1,"reference");
   manager->Select("SampleSet","Sample2.amn");
   manager->Select("SampleSet","Sample3.amn");
   manager->Select("SampleSet","Sample4.amn");

// normalize data
   manager->Normalize("SampleSet");

// cleanup
   delete manager;
}//Normalize
```

Script 4: Normalize expression values.

Filtering: Currently, methods to filter differentially expressed genes are based on univariate tests for each gene and correction for multiple hypothesis testing using adjusted p-values, as described in [11] . The following resampling-based multiple testing procedures are implemented for controlling the family-wise error rate: Bonferroni, Hochberg, Holm, Westfall and Young step-down adjustment. In addition, the FDR method of Benjamini and Hochberg to control the false discovery rate is available. These procedures are currently implemented for tests based on z-statistics, t-statistics and paired t-statistics. The results of these procedures are summarized using adjusted p-values (`p-adjust`), which may be computed from the

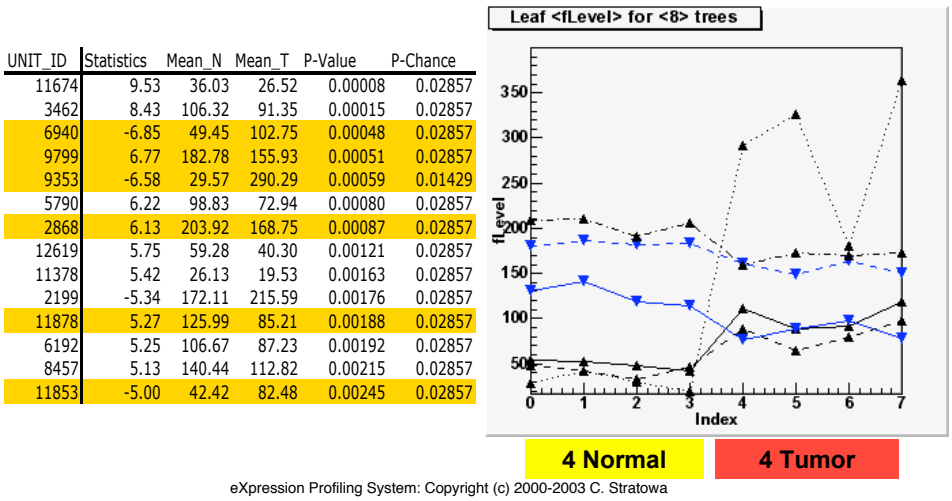
nominal distribution of the test statistic (`p-vlaue`) or by permutation (`p-chance`). Script 5 shows how to initialize multiple testing, select and group samples for filtering, and export the results of the analysis as tab-separated text-file.

```
void FilterData()
{
// create new analysis manager
   XAnalysisManager *manager = new XAnalysisManager("MyManager");

// create new root analysis file
   manager->New("WY_Test","~/mypath/test","UnivariateAnalysis");

// init default settings : type should be type of analysis
   manager->Initialize("UnivariateAnalysis");
// init analysis : twosided t-test and Westfall & Young step-down adjustment
   manager->InitAnalysis("ttest", "twosided", "wy", 5, 10000, 0, 0, 0.95, 1);

// open root scheme file
   manager->OpenSchemes("~/GeneChipCDFs.root");

// open normalized data file
   manager->OpenData("~/mypath/test/MySamples_sup.root");

// add trees to treeset "SetTTest" and set group affiliation
   manager->AddTree("SetTTest","Sample1.sup", 1, "Group1");
   manager->AddTree("SetTTest","Sample2.sup", 1, "Group1");
   manager->AddTree("SetTTest","Sample3.sup", 2, "Group2");
   manager->AddTree("SetTTest","Sample4.sup", 2, "Group2");

// do t-test and Westfall & Young step-down adjustment
   manager->Analyse("SetTTest", "fLevel", "TreeTTest");

// export results
   manager->Export("SetTTest.*.stt", "stat:pval", "SampleTest_wy.txt");

// cleanup
   delete manager;
}//FilterData
```

Script 5: Filter genes using Westfall and Young step-down adjusted p-values.

To demonstrate the feasibility of this approach, data from the prostate cancer experiment of Singh et al. [13] were used as case study. Starting from the public available CEL-files, raw data were converted to expression measures, subjected to normalization, and differentially expressed genes identified by multiple testing. The results of the analysis, using a subset of samples only, is shown in Figure 9.

Data Mining: Although currently not implemented, both unsupervised and supervised learning methods will be made available within XPS.



UNIT_ID	Statistics	Mean_N	Mean_T	P-Value	P-Chance
11674	9.53	36.03	26.52	0.00008	0.02857
3462	8.43	106.32	91.35	0.00015	0.02857
6940	-6.85	49.45	102.75	0.00048	0.02857
9799	6.77	182.78	155.93	0.00051	0.02857
9353	-6.58	29.57	290.29	0.00059	0.01429
5790	6.22	98.83	72.94	0.00080	0.02857
2868	6.13	203.92	168.75	0.00087	0.02857
12619	5.75	59.28	40.30	0.00121	0.02857
11378	5.42	26.13	19.53	0.00163	0.02857
2199	-5.34	172.11	215.59	0.00176	0.02857
11878	5.27	125.99	85.21	0.00188	0.02857
6192	5.25	106.67	87.23	0.00192	0.02857
8457	5.13	140.44	112.82	0.00215	0.02857
11853	-5.00	42.42	82.48	0.00245	0.02857

Figure 9: Genes differentially expressed in prostate cancer. Expression values of genes highlighted in yellow are plotted.

4 A Novel Standard for Microarray Data Storage

It has often been mentioned that the adoption of common standards for the management and sharing of microarray data is essential. Stoeckert et al. [14] state that "if data from every microarray research project were stored in the same type of database with exactly the same structure, transferring data from one database to another would be a relatively straightforward proposition".

Here I propose a novel standard for the storage of microarray data, which will not only allow easy exchange of data, but also the distributed storage of microarray and oligonucleotide-array data. It is proposed to adopt ROOT files (`TFile`) as common file format for data storage, and ROOT trees (`TTree`) as storage format for the results of chip hybridizations as well as chip layout, probe information and gene annotation. Endorsing ROOT files and trees as common standards has many advantages, already described earlier, such as a machine-independent file format, built-in data compression, LAN and WAN network access as well as GRID access. Data could be accessed in a number of ways: It is possible to access ROOT trees directly from within any C++ or Java [15] program. APIs are currently being developed for Java (JavaRoot) and Python (RootPython). Furthermore, an interface exists already to access ROOT data from within Mathematica, namely MathROOT [16]. In an analogous way, R could be extended (see below). Alternatively, it is easy to export selected data from ROOT trees as text-files.

As an example, how microarray data could be stored as ROOT trees in a standardized manner, a ROOT script is attached as supplementary information, which implements the proposed standard for Affymetrix GeneChip CEL-files. Note, that the size of the ROOT database is 6.3 MB (using the minimal default compression

of `TFile`), while the size of the raw CEL-files is 28 MB; this presents a more than 4-fold size reduction!

5 Proposal for the Extension of R

Statistical analysis of microarray data requires sophisticated software tools. Interestingly, the "statistical microarray analysis community" has settled on one main tool, namely R, not only to analyze the data, but also to develop new algorithms for microarray analysis [17]. Although R is a great analysis environment, it has been mentioned numerous times, that R provides limited support for very large datasets. In order to understand the reason for the proposed extensions to R it may be worth to compare the properties of R and ROOT. This is accomplished in the following table:

Property	R	ROOT
Statistical functions	almost all	HEP-specific
Language	R	C++
Interpreter	yes	CINT
Ability to write scripts	yes	yes, "macro.C"
Compilation of scripts	no	ACLiC: "macro.C+"
Compilation of code	no	yes
Writing of functions	yes, R	yes, C++
Graphical User Interface	limited (Tcl/Tk)	yes, GUI classes
Data storage capacity	Megabyte	Petabyte
Data warehouse	no	TFile, TTree, TChain
RDBMS access	MySQL, Oracle	MySQL,Oracle,PostGres
GRID interface	no	PROOF
MMP, SMP, PC-Clusters	snow	PROOF
Supported architectures	Linux, Windows, MacOS X, Sun, SGI, HP, DEC	
Public domain	GPL	yes
Homepage	cran.r-project.org	root.cern.ch
Online help	r-help@stat.math.ethz.ch	roottalk@pcroot.cern.ch

It is obvious that extending R to access ROOT files and trees could greatly expand the use of R as data mining tool with ROOT being used to build a data warehouse for microarray data. The following simple extensions to read/write ROOT trees could be sufficient to increase the ability of R to handle large datasets:

```
df1 <- read.table(file="/mypath/myfile.root/tree1",
                            col.names=c("leaf1","leaf2"))

write.table(df1[,1:2],file="/mypath/myfile.root/tree2",
                            col.names=c("leaf1","leaf2"))
```

The first statement would convert the data stored in `leaf1` and `leaf2` of `tree1` to an R data.frame, while the second command would store columns 1 and 2 of data.frame `df1` as `leaf1` and `leaf2`, respectively, of `tree2` in `myfile.root`.

Storing other R data as objects in ROOT files would require some more effort, but could allow to use `TFile` as alternative machine-independent workspace in addition to `.RData`.

6 Summary

The size and complexity of the data generated by expression profiling requires novel approaches to data storage and analysis. A couple of commercial software providers offer enterprise software solutions to cope with these problems, however, these solutions seem to be inadequate for very large datasets. While the statistical software environment R and the dedicated Bioconductor package are at the forefront in the development and application of methods for microarray analysis, an analogous software tool for the development of novel approaches to the storage of microarray data seems to be missing. For this purpose, the current paper introduces ROOT, a framework for large-scale data storage and analysis, as an alternative to current database systems. As a truly scalable distributed data storage and retrieval system, ROOT is perfectly suited as basis of a common standard for the management and sharing of microarray data. The proposed extensions to R would permit to access a ROOT based data warehouse from R.

XPS is an attempt to develop a self-contained data warehouse and data mining software system for microarray data in the terabyte realm, which is based entirely on the ROOT data analysis framework. In this paper the current status of the project is described. Currently, only data storage and analysis of Affymetrix GeneChip data is supported, however, class headers for the support of cDNA microarray data, scanned using the Axon GenePix Pro software, exist already. As described, the current implementation of XPS is script based, however, a standalone application with an intuitive graphical user interface is already under development (Figure 10). XPS is work in progress: additional pre-processing and normalization algorithms need to be implemented. Furthermore, the major microarray analysis tools such as supervised and unsupervised machine learning methods are still missing. However, the ability of XPS to export all data as text files permits the usage of external statistical tools such as R, until the corresponding algorithms are implemented in XPS.

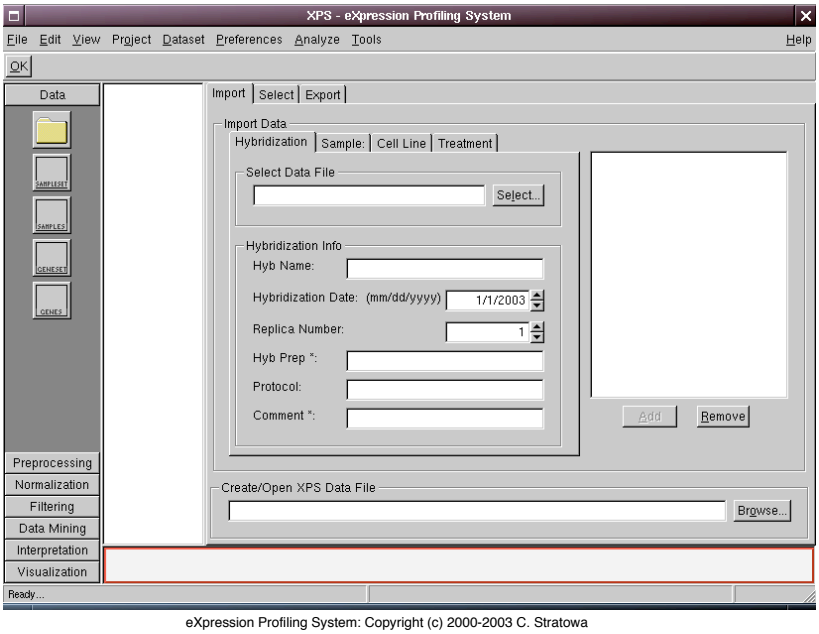


Figure 10: A developmental version of the XPS graphical user interface. Compare with Figure 3.

Acknowledgement

The author would like to thank all people from the `roottalk` mailing list for their continuous online help. In particular, I would like to thank Rene Brun (CERN) for permanent assistance and for many useful suggestions how to implement various ROOT features, and for critically reading the manuscript and testing the supplementary macro.

References

[1] Rolland F. D. *The Essence of Databases*. Prentice Hall, London, 1998.

[2] Han J. and Kamber M. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers, San Francisco, 2001.

[3] Shiers J. Building a Multi-Peta Database: The RD45 Project at CERN. In: A. B. Chaudhri and M. E. S. Loomis. *Object Databases in Practice*. Prentice Hall, New Jersey, 1998.

[4] Brun R. and Rademakers F. ROOT - An Object Oriented Data Analysis Framework,. Proceedings AIHENP'96 Workshop, Lausanne, Sep. 1996, Nucl. Inst. & Meth. in *Phys. Res. A* 389 (1997) 81-86.

[5] Brun R. et al. *The ROOT Users Guide*. Edited by S. Panacek, 2002. `ftp://root.cern.ch/root/Users_Guide_3_2c.pdf`

[6] Goto M. Concept and application of Cint C++ interpreter. *Interface Magazine* 1996 Aug-Nov, CQ publisihing.

[7] Stratowa C. and Wilgenbus K. K. Gene expression profiling in drug discovery and development. *Curr. Opin. Molec. Therapeut.* 1 (1999) 671-679.

[8] Gentleman R. and Carey V. Bioconductor: Open source bioinformatics using R. *R News* 2 (2002) 11-16.

[9] Brazma A. et al. Minimum information about a microarray experiment (MIAME) - towards standards for microarray data. *Nature Genetics* 29 (2001) 365-371.

[10] Affymetrix. *Affymetrix Microarray Suite User Guide*, version 5. Affymetrix, Santa Clara, CA, 2001.

[11] Dudoit S. et al. Statistical methods for identifying differentially expressed genes in replicated cDNA microarray experiments. *Technical Report* 578 (2000), Department of Statistics, UC Berkeley, CA. (`http://www.stat.Berkeley.EDU/users/terry/zarray/TechReport/578.pdf`)

[12] Stratowa C. and Abseher R. The importance of normalization for the analysis of gene expression profiles across multiple microarrays. *International Genomic/Proteomic Technology* 1 (2001) 44-48.

[13] Singh D. et al. Gene expression correlates of clinical prostate cancer behavior. *Cancer Cell* 1 (2002) 203-209.

[14] Stoeckert C. J., Causton H. C. and Ball C. A. Microarray databases: standards and ontologies. *Nature Genetics* Supplement 32 (2002) 469-473.

[15] FreeHEP Java library: `http://java.freehep.org/lib/freehep/doc/root/index.html`

[16] Langston M. D. MathROOT: `http://mathroot.sourceforge.net/`

[17] Stratowa C. Correlation of gene expression profiles with clinical data. *Current Drug Discovery* (Sept. 2002) 29-33.