SPRINT User Guide

Latest release	SPRINT 1.0.7 - 24.09.2014
Previous release	SPRINT 1.0.6 - 06.06.2014

Changes since release 1.0.6

- SPRINT now works with Linux (Fedora, Debian) running OpenMPI in addition to running on MPICH on Linux and OpenMPI and MPICH on Mac OS X as in release 1.0.6.
- A bug that could cause a seg fault error has been fixed in pcor().
- The psvm() method has been removed until it can be changed to meet CRAN guidelines.

Contents

1. Introduction

SPRINT (Simple Parallel R INTerface) is a parallel framework for R. It is intended to make High Performance Computing (HPC) accessible to R users who are not familiar with parallel programming and use of HPC architectures.

SPRINT does this by providing an HPC harness that allows R scripts to run on HPC clusters.

SPRINT contains a library of selected R functions that have been parallelized. Functions are named after the original R function with the added prefix 'p', i.e. the parallel version of *cor()* in SPRINT is called ***pcor()***. Calls to the parallel R functions are included directly in standard R scripts. SPRINT has been developed by staff at the Department of Pathway Medicine and EPCC at the University of Edinburgh.

SPRINT currently includes the following functions:

papply() – a parallel apply function
pboot() – a parallel bootstrapping function
pcor() – a parallel Pearson's correlation
pmaxT() – a parallel permutation test
ppam() – a parallel clustering function (partioning around medoids)
prandomForest() – a parallel machine learning classifier function
pRP() – a parallel rank product analysis function
pstringdistmatrix() – a parallel function to compute the hamming distance between strings
pterminate() – a function which shuts down the SPRINT library
ptest() – a simple test function to test SPRINT.

2. Requirements

Multi-core or HPC platform running:

- R v 2.9.2 – 3.1.0, SPRINT has been tested using 3.0.2, 2.15.3, 2.14.0, 2.12.1, 2.10.1, 2.10.0 and 2.9.2.

R is available from here: http://www.r-project.org

(follow “CRAN” link for downloading, or “Manuals” for download and install instructions)

- C compiler: The use of the gcc compiler is recommended.
- MPI: SPRINT needs MPI to allow the processors to communicate with each other while running the code in parallel. SPRINT can be run with either MPICH or OpenMPI.

  MPICH is available here: http://www.mpich.org/

  OpenMPI is available here: http://www.open-mpi.org/
- Unix/Linux - SPRINT is designed for HPC systems and these all run on Unix/Linux.
- Mac OSX is supported as well as Unix/Linux.
- SPRINT is not designed to run on Windows at present.

2.1. Notes

- User access to HPC platforms (clusters, supercomputers) will vary from service to service. The installation of software is likely to be limited to system administrators (unlike Unix/Linux or Mac OSX personal computers). Therefore help from your system administrator may be required to ensure that the required environment is set up on your HPC system. Running jobs is often only allowed through a batch queue system rather than interactively. In such case, R scripts using SPRINT will need to be submitted to the batch queue using the appropriate utility specific to your HPC system (i.e. mpiexec, qsub).

3. Installing Prerequisites on Unix/Linux

See Section 4 for how to install on Mac OSX.

Before installing SPRINT, the gcc compiler, MPICH, and R version 2.9 – 3.x must be installed.

- gcc compiler: http://gcc.gnu.org/
- MPICH: http://www.mpich.org/
- R: http://www.r-project.org

Skip to section 5 now for instructions on how to install SPRINT.

4. Installing Prerequisites on Mac OSX

Before installing SPRINT, the Xcode command line tools (to provide C and Fortran compilers), MPICH, and R version 2.9 – 3.0 must be installed.

4.1. Xcode

Install Xcode with Command Line Tools option selected.

You can check to see if the command line tools are already installed by running '`which gcc`' from the command line. If there's no response to this command, then you need to follow the install instructions below.

Mountain Lion 10.9

Xcode 5 you can get for free through the App Store, command line tools are installed if you enter "gcc" or "make" at the command line, after which Xcode will prompt to install these.

Mountain Lion 10.8 and Lion 10.7

For 10.8 and 10.7, you can install the Command Line Tools without the rest of Xcode. Go to Downloads for Apple Developers, search for "command line tools" and install the appropriate version for your OS.

Alternatively, if you already have Xcode installed, you can open the application and use the Xcode Downloads preferences pane to add command line tools.

Snow Leopard 10.6

Go to Downloads for Apple Developers, search for “Xcode 3.2.6” in the top left search field, and then you'll find a download for Xcode 3.2 for Snow Leopard. Select ‘Customise’ from the installer and then select ‘UNIX Development’ to install the command line tools.

Leopard 10.5

Go to Downloads for Apple Developers, search for “Xcode 3.1.4” in the top left search field, and then you'll find a download for Xcode for Leopard. Select ‘UNIX Development Support’ from the installer.

4.2. R

Install R version 3.0, available here: http://cran.r-project.org/

4.3. MPI

SPRINT depends on MPI – either MPICH or OpenMPI implementations of MPI can be used. At the command line, check to see if you already have MPI installed.

```
$ mpicc -v
```

If the command is not found then you'll have to install MPICH or OpenMPI (either with MacPorts or with homebrew):

Using homebrew

If running `'which brew'` on the command line returns a result, then you already have homebrew installed (or get it here: http://brew.sh/), then install MPICH as follows:

```
$ brew install mpich2
```

Alternatively install OpenMPI:

```
$ brew install open-mpi
```

Using MacPorts

If running `'which port'` on the command line returns a result, then you already have MacPorts installed (or get it here: http://www.macports.org/), then install MPICH as follows:

```
$ sudo port install mpich2
```

Alternatively install OpenMPI:

```
$ sudo port install openmp
$ sudo port select --set mpi openmpi-mp-fortran'
```

5. Installing SPRINT on Unix/Linux or Mac

Use the Package Installer in the R menu bar to install the SPRINT dependencies: rlecuyer, boot, randomForest, e1071 and ff. The ff package is used by SPRINT to handle data sets that are too large to fit into memory.

```
R-> Packages & Data -> Package Installer
```

Alternatively, install the SPRINT dependencies from the R GUI console as follows.

```
> install.packages("rlecuyer")
> install.packages("boot")
> install.packages("e1071")
> install.packages("ff")
> install.packages("randomForest")
```

Then install SPRINT.

```
> install.packages("sprint")
```

SPRINT can also be downloaded from http://www.r-sprint.org/ and installed from the command line as follows.

```
$ R CMD install sprint_1.0.6.tar.gz[1]
```

You should then be able to load SPRINT from the R console (or from within a script):

```
> library("sprint")
```

5.1. Notes

- If the warning message: “package ‘boot’ is not available (for R version 2.15.2)” appears, try installing from the R app console instead of running R from a terminal command line. If that fails you may have to download older versions of the packages from the CRAN archive. Install from R using the following command:

  ```
  > install.packages("~/Downloads/boot_1.3-7.tar.gz", repos = NULL)
  ```

- R tests if the installed package can be loaded during the installation. SPRINT requires MPI to run and if you try to install it without MPI then the installation will fail. If you are installing the SPRINT library on a cluster where MPI is only installed on the back-end nodes but not on the front-end nodes then you may need to use the "--no-test-load" flag during the installation process.

  ```
  $ R CMD INSTALL --no-test-load sprint
  ```

- The configure script automatically identifies the appropriate compiler for building SPRINT. This option should only be used if the script fails to locate the MPI compiler.

  Pass optional arguments to the installation command:

  ```
  --configure-args="--with-wrapper-script=$WRAPPER_SCRIPT"
  ```

  where:

  $WRAPPER_SCRIPT contains the compiler to be used for building SPRINT, e.g. "mpicc".

5.2. Testing the installation

The SPRINT library includes a function to test the installation called ***ptest()***. It simply prints a message identifying each processor in the compute cluster. For example, when using SPRINT with 4 processors you will get the following output:

```
[1] "HELLO, FROM PROCESSOR: 0" "HELLO, FROM PROCESSOR: 2"
```

[1] Throughout this document ‘>’ will indicate a command run from within R, and ‘$’ will indicate a command run from a terminal window.

```
[3] "HELLO, FROM PROCESSOR: 1" "HELLO, FROM PROCESSOR: 3"
```

This is obtained by running the following sample R script, install_test.R from the command line using the mpiexec command:

```
$ mpiexec -n 4 R -f install_test.R
```

```
library("sprint")

ptest()

pterminate()

quit()
```

6. Using SPRINT

SPRINT should be run on multiple processors to get the benefit of the parallelisation in the code. This is done by saving the R script that calls SPRINT to a file, and then using a command line call to run that file on several processors.

Write the R script

First, include the SPRINT library - within your R script use 'library("sprint")'. Then include calls to the SPRINT functions you wish to use. Finally, all SPRINT enabled scripts require that ***pterminate()*** is called before the final quit() command. This calls MPI_FINALIZE and shuts down SPRINT. You can run the script interactively from within the R console to test it, and when you're happy with it, save the file (as install_test.R in this example) and see the next section for how to run the code in parallel.

For example, a simple R script which calls one single function called ***ptest()*** will look like this:

```
library("sprint")

ptest()

pterminate()

quit()
```

Run the R SCRIPT in Parallel on Many Processors

The above script only gives access to SPRINT within R; it will not give you multiple processors. You will need to run MPICH to do this. How this is done depends on your system set-up. You will have to specify the location of the script name and the number of processors to be used.

For example, this command will run the install_test.R script on 4 processors. 'mpiexec –n 4' starts 4 MPICH processes running and 'R –f install_test.R' Runs the R code on each of the processes.

```
> mpiexec -n 4 R -f install_test.R
```

The available functions in SPRINT are: ***papply()*** – a parallel apply function; ***pboot()*** – a parallel bootstrapping function; ***pcor()*** – a parallel Pearson's correlation; ***pmaxT()*** – a parallel permutation test; ***ppam()*** – a parallel clustering function (partioning around medoids); ***prandomForest()*** – a parallel machine learning classifier function; ***pRP()*** – a parallel rank product analysis function; ***pstringdistmatrix()*** – a parallel function to compute the hamming distance between strings; ***pterminate()*** – a function which shuts down the SPRINT library and ***ptest()*** – a simple test function to test SPRINT.

7. SPRINT Functions

7.1. papply()

papply() is essentially an apply function. Apply functions are used to perform the same operation over all the elements of data objects like matrices, data frames or lists. For example, the function mean() might be applied to each row in a data matrix to obtain all row averages. This function provides a parallel implementation of both the *apply()* and *lapply()* functions from the core of the R programming language. *apply()* can be used with a vector, array or list, while *lapply()* has been optimised for using on lists. The function to be applied can be supplied to ***papply()*** either as a function name or as a function definition. When only the function name is provided, the package implementing the function has to be loaded before the SPRINT library is initialised in order to ensure that the name is recognised by all the processes involved in the computation.

The interface to the parallel function ***papply()*** combined the interfaces of *apply()* and *lappy()*:

```
papply(data, fun, margin = 1, out_filename = NULL)
```

where:

- 'data' is the input data matrix, list or ff object.
- 'fun' is the function to be applied.
- 'margin' is a vector indicating which elements of the matrix the function will be applied to. The default value is 1 and indicates the rows, 2 indicates the columns and the parameter is ignored if data is a list.
- 'out_filename' is not used at present..

Type '?papply' in the R console for more detail on this function.

Examples of valid calls to ***papply()*** are:

```
papply(data, mean, margin = 1)
papply(list, mean)
```

Citation

Apply any function to each row/column in a matrix. A generic function useful in many situations where for-loops may be slower. Based on function apply() in R base package.

7.2. pboot()

pboot() generates R bootstrap replicates of a statistic applied to data. For example, the bootstrapped standard error of the mean might be constructed from repeat application of the mean() function on random subsets of the same set of data. It implements a parallel version of the bootstrapping method *boot()* from the *boot* R package (http://cran.r-project.org/web/packages/boot/index.html). However, it is not compatible with other SPRINT functions, i.e. you cannot bootstrap other parallel functions from the SPRINT library. It is therefore recommended to use it only as a standalone function.

The interface and parameters to the parallel function ***pboot()*** are identical to the serial function *boot()*:

```
pboot(data, statistic, R, sim = "ordinary", stype = "i",
      strata = rep(1, n), L = NULL, m = 0, weights = NULL,
      ran.gen = function(d, p), mle = NULL, simple = FALSE, ...)
```

where:

- 'data' is the input data vector or matrix. If it is a matrix then each row is considered as one multivariate observation.
- 'statistic' is a function which when applied to data returns a vector containing the statistic(s) of interest. When sim is set to "parametric", the first argument to statistic must be the data. For each replicate a simulated dataset returned by ran.gen will be passed. In all other cases, statistic must take at least two arguments. The first argument passed will always be the original data. The second will be a vector of indices, frequencies or weights which define the bootstrap sample.
- 'R' is the number of bootstrap replicates.
- 'sim' is a character string indicating the type of simulation required. The default value is "ordinary". Other possible values are "parametric", "balanced", "permutation", and "antithetic". Importance resampling is specified by including importance weights; the

type of importance resampling must still be specified but may only be “ordinary” or “balanced” in this case.

- ‘stype’ is a character string indicating what the second argument of statistic represents. The default value is “i” for indices. Other possible values are “f” for frequencies and “w” for weights. It is not used when sim is set to “parametric”.
- ‘strata’ is an integer vector or factor specifying the strata for multi-sample problems. This may be specified for any simulation, but is ignored when sim is set to “parametric”. When strata is supplied for a nonparametric bootstrap, the simulations are done within the specified strata.
- ‘L’ is the vector of influence values evaluated at the observations. This is used only when sim is set to “antithetic”. If not supplied, they are calculated through a call to empinf. This will use the infinitesimal jackknife provided that stype is set to “w” otherwise the usual jackknife is used.
- ‘m’ is the number of predictions which are to be made at each bootstrap replicate. This is most useful for (generalized) linear models. This can only be used when sim is “ordinary”. m will usually be a single integer but, if there are strata, it may be a vector with length equal to the number of strata, specifying how many of the errors for prediction should come from each strata. The actual predictions should be returned as the final part of the output of statistic, which should also take an argument giving the vector of indices of the errors to be used for the predictions.
- ‘weights’ is a vector or matrix of importance weights. If a vector then it should have as many elements as there are observations in the input data. When simulation from more than one set of weights is required, weights should be a matrix where each row of the matrix is one set of importance weights. If weights is a matrix then the number of bootstrap replicates R must be a vector of length nrow(weights). This parameter is ignored if sim is not set to “ordinary” or “balanced”.
- ‘ran.gen’ is a function used only when sim is set to “parametric”. It describes how random values are to be generated. It should be a function of two arguments. The first argument should be the observed data and the second argument consists of any other information needed (e.g. parameter estimates). The second argument may be a list, allowing any number of items to be passed to ran.gen. The returned value should be a simulated data set of the same form as the observed data which will be passed to statistic to get a bootstrap replicate. It is important that the returned value be of the same shape and type as the original dataset. If ran.gen is not specified, the default is a function which returns the original input data in which case all simulation should be included as part of statistic. Setting sim to “parametric” and using a suitable ran.gen allows the user to implement any types of nonparametric resampling which are not supported directly.
- ‘mle’ is the second argument to be passed to ran.gen. Typically these will be maximum likelihood estimates of the parameters. For efficiency mle is often a list containing all of the objects needed by ran.gen which can be calculated using the original data set only.
- ‘simple’ is a boolean. It can only be set to TRUE if sim is set to “ordinary”, stype is set to “I” and n is set to 0. Otherwise it is ignored and generates a warning. By default a n by R

index array is created which can be large. If simple is set to TRUE, this is avoided by sampling separately for each replication, which is slower but uses less memory.
- '…' are other named arguments for statistic which are passed unchanged each time.

Examples of valid a calls to ***pboot()*** are:

```
b <- pboot(city, ratio, R=999, stype="w")
b <- pboot(discoveries, trimmedmean, R=1000, trim=5)
```

Citation

Bootstrap estimates of any given statistic. Based on boot() function in boot package. Cited source: Angelo Canty and Brian Ripley. "boot: Bootstrap R (S-PLUS) Functions", 2013.

7.3. pcor()

pcor() performs a parallel Pearson's correlation. It either takes a 2D array as input and correlates each row with every other row or takes two 2D arrays and correlates the columns of the first matrix with the columns of the second matrix. The output can either be the matrix of correlation coefficient or the distance matrix.

To use ***pcor()***:

```
pcor(data_x, data_y, distance = FALSE, caching_ = "mmeachflush",
      filename_ = NULL)
```

where:

- 'data_x' is the input matrix data.
- 'data_y' is the second input matrix with compatible dimensions to data_x.
- 'distance' is a boolean indicating whether the output is to be a distance matrix rather than the correlation coefficient matrix.
- 'caching_' caching scheme for the backend, currently "mmnoflush" or "mmeachflush" (flush mmpages at each swap) if no name is specified the default value is "mmeachflush".
- 'filename' is a string and is optional. It specifies the name of a file where the results will be saved. By default, the results are saved to a temporary file that is deleted after exiting from SPRINT.

Examples of valid calls to ***pcor()*** are:

```
ff_obj <- pcor(t(inData))
ff_obj <- pcor(t(inData_x), t(inData_y))
ff_obj <- pcor(t(inData), filename_="output.dat")
ff_obj <- pcor(data, caching_="mmeachflush", filename_="output.dat")
ff_obj <- pcor(t(inData), distance=TRUE, filename_="output.dat")
```

The first four are parallel equivalents to the call of the sequential *cor()*:

```
results <- cor(t(inData))
```

This last one also implements a parallel equivalent to *cor()* but returns a different output, that is the distance matrix.

Citation

Pearson correlation for pairs of numeric variables. For example used in obtaining gene adjacency networks through measured gene-gene similarities across a range of samples or conditions. Based on cor() function in package stats: Becker et al. The New S Language. Wadsworth & Brooks/Cole 1988.

7.4. pmaxT()

Note that pmaxT does not work on the HECToR supercomputer.

pmaxT() implements a parallel version of the mt.maxT function from the multtest package (http://www.bioconductor.org/packages/release/bioc/html/multtest.html). It computes the adjusted p-values for step-down multiple testing procedures.

To use ***pmaxT()***:

```
pmaxT(X, classlabel, test = "t", side = "abs", B = 10000,
       na = .mt.naNUM, fixed.seed.sampling = "y", nonpara = "n")
```

where:

- 'X' is the input data array.
- 'classlabel' is the class labels of the columns of the input dataset.
- 'test' is the statistical method used for testing the null hypothesis. The following six methods are supported:
  - t: Tests based on a two-sample Welch t-statistics (unequal variances)

    - t.equalvar: tests based on a two-sample t-statistics with equal variance for the two samples.
    - Wilcoxon: Tests based on standardized rank sum Wilcoxon statistics.
    - F: Tests based on F-statistics.
    - Pair-T: Tests based on paired t-statistics.
    - Block-F: Tests based on F-statistics which adjust for block differences.
- 'side' is the type of rejection region. The following values are available:
    - "abs" for absolute difference
    - "upper" for the maximum difference
    - "lower" for the minimum difference
- 'B' is the number of permutations. If set to "0" then the complete permutations of the data will be computed.
- 'na' is the representation used for missing values. Missing values are excluded from all computations.
- 'fixed.seed.sampling' can either be:
    - "y" to compute the permutations on the fly
    - "n" to save all permutations in memory prior to computations
- 'nonpara' can either be:
    - "y" for non-parametric test statistics
    - "n" otherwise.

The interface and parameters to the parallel ***pmaxt()*** are identical to those for the sequential *mt.maxt()*:

```
pmaxT(X, classlabel, test = "t", side = "abs", B = 10000,
       na = .mt.naNUM, fixed.seed.sampling = "y", nonpara = "n")
```

Citation

Permutation-adjusted p-values. Used in statistical testing of inference hypotheses (e.g. is a gene differentially expressed between two conditions) to provide robustly estimated p-values that are adjusted for multiple testing. Based on function mt.maxT() in package multtest, created by Yongchao Ge and Sandrine Dudoit. Cited source: Dudoit S et al. Multiple hypothesis testing in microarray experiments [Submitted].

7.5. ppam()

ppam() is a clustering function that performs a Parallel Partitioning Around Medoids and is based on the *pam()* function from the cluster R package (http://cran.r-project.org/web/packages/cluster/index.html).

The interface and parameters to parallel function ***ppam()*** are similar to the serial function *pam()* but not identical. ***ppam()*** requires a distance matrix as input parameters. Although, ***ppam()*** does not include the option to calculate the distance matrix, this can easily be done using SPRINT ***pcor()*** function with the 'distance' parameter set to TRUE.

To use ***ppam()***:

```
ppam (x, k, medoids = NULL, is_dist = inherits(x, "dist"),
       cluster.only = FALSE, do.swap = TRUE, trace.lev = 0)
```

where:

- 'x' is the input distance matrix or dissimilarity matrix, depending on the value of the "dist" argument. This can either be a matrix or an ff object.
- 'k' is a positive integer indicating the number of clusters. It must be less than the number of observations.
- 'medoids' is either a vector specifying the initial 'k' medoids or the default value NULL which indicates that the initial medoids will be selected by the algorithm.
- 'is_dist' is a boolean indicating whether the input matrix is a distance or dissimilarity matrix (TRUE) or a symmetric matrix (FALSE).
- 'cluster.only' is a boolean when set to TRUE only the clustering will be computed and returned. The default value is FALSE.
- 'do.swap' is a boolean indicating if the swap phase of the algorithm should take place. The default is TRUE. The swap phase is computer intensive and can be skipped by setting the 'do.swap' option to FALSE.
- 'trace.lev' is an integer specifying the trace level for printing diagnostics during the build and swap phases of the algorithm. The default value is 0 which does not produce any output. Increasing values print increasing level of detailed information.

Examples of valid calls to ***ppam()***:

```
# Pre-processing step using pcor() to return an ff object containing a
# distance matrix.
mcor <- pcor(matrix(rnorm(1:10000), ncol=100), distance = TRUE)

p1m <- ppam(mcor, 4)
p2m <- ppam(mcor, 4, medoids = c(1,16))
p3m <- ppam(mcor, 3, trace = 2)
p4m <- ppam(dist(x), 12)
```

Citation

Partitioning-Around-Medoids clustering. Used in identifying and grouping patterns in data, e.g. gene expression profiles in expression studies. Based on pam() function in package cluster, created by Martin Maechler. Cited source: Reynolds A et al. Clustering rules: A comparison of partitioning and hierarchical clustering algorithms. Journal of Mathematical Modelling and Algorithms 5, 475-504, 1992.

7.6. prandomForest()

The machine learning function ***prandomForest()*** is an ensemble tree classifier that constructs a forest of classification trees from bootstrap samples of a dataset. The random forest algorithm can be used to classify both categorical and continuous variables. This function provides a parallel equivalent to the serial *randomForest()* function from the *randomForest* package (http://cran.r-project.org/web/packages/randomForest/index.html).

The interface and parameters to the parallel function ***prandomForest()*** are identical to the serial function *randomForest()*.

```
prandomForest(x, y=NULL, xtest=NULL, ytest=NULL, ntree=500,
              mtry = if (!is.null(y) && !is.factor(y))
                        max(floor(ncol(x)/3), 1)
                     else floor(sqrt(ncol(x))),
              replace=TRUE, classwt=NULL, cutoff, strata,
              sampsize = if (replace) nrow(x)
                         else ceiling(.632*nrow(x)),
              nodesize = if (!is.null(y) && !is.factor(y)) 5 else 1,
              maxnodes=NULL, importance=FALSE, localImp=FALSE,
              nPerm=1, proximity, oob.prox=proximity, norm.votes=TRUE,
              do.trace=FALSE, keep.forest = !is.null(y) &&
                                            is.null(xtest),
              corr.bias=FALSE, keep.inbag=FALSE, ...)
```

where:

- 'x' is the input data matrix.
- 'y' is a vector. If a factor, classification is assumed, otherwise regression is assumed. If omitted, ***prandomForest()*** will run in unsupervised mode.
- 'xtest' is the data matrix of predictors for the test set.
- 'ytest' is the response for the test set.
- 'ntree' is an integer indicating the number of trees to grow.
- 'mtry' is the number of variables randomly sampled as candidates at each split. The default value is sqrt(p) for classification and p/3 for regression, where p is the number of variables in the data matrix x.
- 'replace' is a boolean indicating whether the sampling of cases is done with or without replacement. The default value is TRUE.
- 'strata' a variable used for stratified sampling.

- ‘sampsize’ is the size(s) of sample to draw. For classification, if sampsize is a vector of the length of the number of strata, then sampling is stratified by strata, and the elements of sampsize indicate the numbers to be drawn from the strata.
- ‘nodesize’ is an integer indicating the minimum size of the terminal nodes. The default value is 1 for classification and 5 for regression.
- ‘maxnodes’ is the maximum number of terminal nodes allowed for the trees. The default value is NULL.
- ‘importance’ is a boolean indicating whether the importance of predictors is assessed. The default value is FALSE.
- ‘localImp’ is a boolean indicating whether casewise importance measure is to be computed. The default value is FALSE.
- ‘proximity’ is a boolean indicating whether the proximity measure among the rows is to be calculated.
- ‘oob.prox’ is a boolean indicating whether the proximity is to be calculated for out-of-bag data. The default value is set to be the same as the value of the proximity parameter.
- ‘do.trace’ is a boolean which indicates whether a verbose output is produced. The default value is FALSE. If set to an integer i then the output is printed for every i trees.
- ‘keep.forest’ is a boolean which indicates whether the forest is returned in the output object. The default value is FALSE.
- ‘keep.inbag’ is a boolean indicating whether the matrix which keeps track of which samples are in-bag in which trees should be returned. The default value is FALSE.
- ‘...’ are optional parameters to be passed to the low level function randomForest.default.

The following are only used for classification and ignored for regression:

- ‘classwt’ is a vector of the priors of the classes. Its default value is NULL.
- ‘cutoff’ is a vector with k elements where k is the number of classes. The ‘winning’ class for an observation is the one with the maximum ratio of proportion of votes to cutoff. The default value is 1/k.
- ‘norm.votes’ is a boolean which indicates whether the final result of votes are expressed as fractions or whether the raw vote counts are returned. The default value is TRUE.

The following are only used for regression and ignored for classification:

- ‘nPerm’ indicates the number of times the out-of-bag data are permuted per tree for assessing variable importance. The default value is one.
- ‘corr.bias’ is a boolean indicating whether to perform a bias correction. The default value is FALSE.

An example of a valid a call to ***prandomForest()*** is:

```
rf <- prandomForest(x=data, y=classes, ntree=5000, ...)
```

Citation

Random Forest classification algorithm. Used in classifying (predicting the biological class or medical status) samples in a data set by constructing a large number of decision trees and aggregating their outcomes. Based on function randomForest() in package randomForest, created by Andy Liaw and Matthew Wiener. Cited source: Breiman L. Random Forests. Machine Learning 45(1),5-32, 2001.

7.7. pRP()

pRP() is a parallel rank product analysis algorithm. Rank products are a method of identifying differentially regulated genes in replicated microarray experiments. The SPRINT task parallel implementation of the rank product method is approximately twice as fast in serial as the existing *RP()* function from the *RankProd* package available at Bioconductor (http://www.bioconductor.org) and it shows excellent scaling.

The interface and parameters to the parallel function ***pRP()*** are identical to the serial function *RP()*.

```
pRP (data, cl, num.perm = 100, logged = TRUE, na.rm = FALSE,
      gene.names = NULL, plot = FALSE, rand = NULL, sum = FALSE)
```

where:

- 'data' is the input data matrix.
- 'cl' is a vector containing the class labels of the samples.
- 'num.perm' is an integer for the number of permutations used in the calculation of the null density. The default value is 100.
- 'logged' is a boolean indicating whether the data is logged or not. The default value is TRUE.
- 'na.rm' is a boolean indicating whether missing values are to be replaced by the gene-wise mean of the non-missing values and used in computing rank. The default value is FALSE.
- 'gene.names' the gene name to be assigned to the estimated percentage of false positive predictions. The default value is NULL.

- ‘plot’ is a boolean which indicates whether to plot the estimated percentage of false positive predictions against the rank of each gene. The default value is FALSE.
- ‘rand’ is an optional number used as the seed for the random number generator if specified. The default value is NULL.
- ‘sum’ is a Boolean which indicates whether to perform a rank sum analysis.

Examples of valid calls to ***pRP()*** are:

```
rp <- pRP(data, cl=classes, num.perm=100, logged=FALSE)
rp <- pRP(data, cl=classes, num.perm=100)
```

Citation

Rank-Product statistical testing. This non-parametric permutation-based statistical test is used in similar circumstances to parametric tests (e.g. t test) but is more robust for small sample sizes and focuses on between-sample ratios rather than per-group means. Based on function RP() in package RankProd, created by Fangxin Hong. Cited source: Breitling R et al. Rank Products: A simple, yet powerful, new method to detect differentially regulated genes in replicated microarray experiments. FEBS Letter, 57383-92.

7.8. pstringdistmatrix()

pstringdistmatrix() calculates the hamming distance between each pair of strings. Returns an ff result matrix.

The interface and parameters to the parallel function ***pstringdistmatrix()*** are similar to the *stringdistmatrix()*function from the stringdist package.

```
pstringdistmatrix(a, b, method = "h", filename = NULL, weight = NULL,
                  maxDist = 0, ncores = NULL)
```

where:

- ‘a’ R object (target); will be converted by ‘as.character’.
- ‘b’ R object (target); will be converted by 'as.character'. Must be the same as argument a in this version of the software.

- ‘method’ Method for distance calculation - only option 'h' for hamming distance is supported.
- ‘filename’ Results will be stored here as binary data
- ‘weight’ Not used in the hamming distance measure.
- ‘maxDist’ Not used in the hamming distance measure.
- ‘ncores’ Not used by SPRINT, please see the SPRINT user guide.

Examples of valid calls to ***pstringdistmatrix ()*** are:

```
strings <- c("lazy", "HaZy", "rAzY")
pstringdistmatrix(strings, strings, method="h", filename=”output”)
```

Citation

Hamming distance for pairs of character strings. Used for example in measuring distance between nucleotide sequences. Based on function stringdist() in package stringdist, created by Mark van der Loo, 2013. Cited source: Hamming RW. Error detecting and Error Correcting codes. The Bell System Technical Journal 29, 147-160.

7.9. pterminate()

The ***pterminate()*** function indicates the end of the use of the SPRINT library. It terminates the use of MPI and shut down the SPRINT library. It is therefore the last SPRINT instruction to be included in a R script using SPRINT. The execution of the R script returns from parallel to serial after ***pterminate()***.

7.10. ptest()

ptest() is a function that test the correct installation of the SPRINT library. It simply prints a message identifying each processor in the compute cluster.

7.11. Performance

SPRINT parallel functions run on multiple processors reducing the time taken for the calculation to complete. Note that the speed-up depends on the function. In particular, the performances of ***papply()*** depends on the complexity of the function to be applied. As a rule of thumb, the higher the complexity of the function, the higher the performances gain. The speed-up also depends on the size of the data set being analyzed. A small data set will show no speed-up on 3

or more processors. However, tests on larger data sets have shown an almost perfect scaling for up to 512 cores.

8. Troubleshooting

Known issues in Open MPI result in unreliable results when running ***pcor()*** on more than one node. Sometimes the result matrix will be wrong. The symptoms for this issue are entire columns of zero (0) values and data shifted towards the right, especially the expected diagonal line of one (1) values. See section 2.1 earlier in this document.

```
** testing if installed package can be loaded Error in
system2(file.path(R.home("bin"), "R"), c(if (nzchar(arch)) paste0("--
arch=",  :    error in running command
```

MPI library not imported.

This error has been seen on Linux with OpenMPI installed. SPRINT installs ok, but then fails to load.

Error message:

** testing if installed package can be loaded Error in system2(file.path(R.home("bin"), "R"), c(if (nzchar(arch)) paste0("--arch=", : error in running command

Solution:

```
export LD_PRELOAD= =$PATH_TO_libmpi.so
```

Or you may need to add a * at the end:

```
export LD_PRELOAD =$PATH_TO_libmpi.so.*
```

C compiler not found error on Mac

Error message:

configure: error: no acceptable C compiler found in $PATH

See `config.log' for more details.

ERROR: configuration failed for package ?sprint?

Solution:

You need to install Xcode command line tools, if using a Mac.

No MPI Error

Error message:

configure: error: "Unable to detect MPI compiler. Please use --with-wrapper-script option"

Solution:

Intstall MPI as described in the Pre-requisites section above.

Wrong architecture error on Mac

Error message:

Error in dyn.load(file, DLLpath = DLLpath, ...) :

unable to load shared object '/Library/Frameworks/R.framework/Versions/2.14/Resources/library/sprint/libs/i386/sprint.so' :

dlopen(/Library/Frameworks/R.framework/Versions/2.14/Resources/library/sprint/libs/i386/sprint.so, 6): no suitable image found. Did find:

/Library/Frameworks/R.framework/Versions/2.14/Resources/library/sprint/libs/i386/sprint.so: mach-o, but wrong architecture

Error: loading failed

Execution halted

ERROR: loading failed

Solution:

Add the correct arch flag for your system (alternatives include x86_64, i386, ppc) as follows.

```
> R --arch=x86_64 CMD INSTALL sprint_1.0.6.tar.gz
```

9. Configuration of systems used to test SPRINT

SPRINT has been developed and tested on Mac OSX and on Linux.

The setup and version details are listed below.

9.1. Mac OSX

Tested and working using MPICH version 3.1.2, clang-503.0.40 and R 3.1.1 on MacOSX 10.9.4 and also OpenMPI 1.8.1.

```
$ cc -v

Apple LLVM version 5.1 (clang-503.0.40) (based on LLVM 3.4svn)

Target: x86_64-apple-darwin13.3.0

Thread model: posix


$ R

R version 3.1.1 (2014-07-10) -- "Sock it to Me"

Copyright (C) 2014 The R Foundation for Statistical Computing

Platform: x86_64-apple-darwin13.1.0 (64-bit)


//OpenMPI

macproeg:bin egrant1$ mpicc -v

Apple LLVM version 5.1 (clang-503.0.40) (based on LLVM 3.4svn)

Target: x86_64-apple-darwin13.3.0

Thread model: posix

$ mpicc --showme

clang -I/usr/local/Cellar/open-mpi/1.8.1/include -
L/usr/local/opt/libevent/lib -L/usr/local/Cellar/open-
mpi/1.8.1/lib -lmpi
```

```
//MPICH

$ mpicc -v

mpicc for MPICH version 3.1.2

Apple LLVM version 5.1 (clang-503.0.40) (based on LLVM 3.4svn)

Target: x86_64-apple-darwin13.3.0

Thread model: posix

clang: warning: argument unused during compilation: '-I
/usr/local/Cellar/mpich2/3.1.2/include'


[1] "*** System info ***"

R version 3.1.1 (2014-07-10)

Platform: x86_64-apple-darwin13.1.0 (64-bit)


locale:

[1] en_GB.UTF-8/en_GB.UTF-8/en_GB.UTF-8/C/en_GB.UTF-8/en_GB.UTF-8


attached base packages:

[1] parallel  stats     graphics  grDevices utils     datasets
methods

[8] base


other attached packages:

 [1] sprint_1.0.7            randomForest_4.6-7
rlecuyer_0.3-3

 [4] stringdist_0.7.3        multtest_2.20.0         e1071_1.6-3

 [7] ShortRead_1.22.0        GenomicAlignments_1.0.2
BSgenome_1.32.0
```

```
[10] Rsamtools_1.16.1        GenomicRanges_1.16.3
GenomeInfoDb_1.0.2

[13] Biostrings_2.32.1       XVector_0.4.0
IRanges_1.22.9

[16] BiocParallel_0.6.1      boot_1.3-11
golubEsets_1.6.0

[19] Biobase_2.24.0          BiocGenerics_0.10.0
cluster_1.15.2

[22] ff_2.2-13               bit_1.1-12              RUnit_0.4.26


loaded via a namespace (and not attached):

 [1] BatchJobs_1.3       BBmisc_1.7          bitops_1.0-6

 [4] brew_1.0-6          checkmate_1.1       class_7.3-10

 [7] codetools_0.2-8     DBI_0.2-7           digest_0.6.4

[10] fail_1.2            foreach_1.4.2       grid_3.1.1

[13] hwriter_1.3         iterators_1.0.7     lattice_0.20-29

[16] latticeExtra_0.6-26 MASS_7.3-33         RColorBrewer_1.0-5

[19] Rcpp_0.11.2         RSQLite_0.11.4      sendmailR_1.1-2

[22] splines_3.1.1       stats4_3.1.1        stringr_0.6.2

[25] survival_2.37-7     tools_3.1.1         zlibbioc_1.10.0

               _
platform       x86_64-apple-darwin13.1.0

arch           x86_64

os             darwin13.1.0

system         x86_64, darwin13.1.0

status

major          3

minor          1.1
```

```
year           2014
month          07
day            10
svn rev        66115
language       R
version.string R version 3.1.1 (2014-07-10)
nickname       Sock it to Me

sysname

"Darwin"

release

"13.3.0"

version

"Darwin Kernel Version 13.3.0: Tue Jun  3 21:27:35 PDT 2014;
root:xnu-2422.110.17~1/RELEASE_X86_64"

machine

"x86_64"


[1] "*** End of system info ***"
```

9.2. Linux

Tested and working using MPICH2 version 1.4.1, gcc 4.4.7 and R 3.1.0 on Linux and also with OpenMPI 1.5.4 , gcc 4.4.7 and R 3.1.0.

```
// OpenMPI

$ mpirun -version

mpirun (Open MPI) 1.5.4


// MPICH

$ mpiexec -version

HYDRA build details:

    Version:                                 1.4.1p1

    Release Date:                            Thu Sep  1 13:53:02
CDT 2011

    CC:                              gcc -fPIC -fPIC

    CXX:                             g++ -fPIC

    F77:                             gfortran -fPIC

    F90:                             gfortran

    Configure options:                       '--
prefix=/opt/mpich2-gnu' '--with-pm=hydra' '--enable-cxx' '--
enable-debug' '--enable-fc' 'CFLAGS=-fPIC -O2' 'CPPFLAGS=-fPIC -
I/scratch/mpich2-1.4.1p1/src/mpl/include -I/scratch/mpich2-
1.4.1p1/src/mpl/include -I/scratch/mpich2-1.4.1p1/src/openpa/src
-I/scratch/mpich2-1.4.1p1/src/openpa/src -I/scratch/mpich2-
1.4.1p1/src/mpid/ch3/include -I/scratch/mpich2-
1.4.1p1/src/mpid/ch3/include -I/scratch/mpich2-
1.4.1p1/src/mpid/common/datatype -I/scratch/mpich2-
1.4.1p1/src/mpid/common/datatype -I/scratch/mpich2-
1.4.1p1/src/mpid/common/locks -I/scratch/mpich2-
1.4.1p1/src/mpid/common/locks -I/scratch/mpich2-
1.4.1p1/src/mpid/ch3/channels/nemesis/include -I/scratch/mpich2-
1.4.1p1/src/mpid/ch3/channels/nemesis/include -I/scratch/mpich2-
1.4.1p1/src/mpid/ch3/channels/nemesis/nemesis/include -
I/scratch/mpich2-
1.4.1p1/src/mpid/ch3/channels/nemesis/nemesis/include -
I/scratch/mpich2-
1.4.1p1/src/mpid/ch3/channels/nemesis/nemesis/utils/monitor -
I/scratch/mpich2-
1.4.1p1/src/mpid/ch3/channels/nemesis/nemesis/utils/monitor -
```

```
I/scratch/mpich2-1.4.1p1/src/util/wrappers -I/scratch/mpich2-
1.4.1p1/src/util/wrappers' 'FFLAGS=-fPIC -O2' 'F77=gfortran'
'FC=gfortran' 'CC=gcc' 'CXX=g++' '--disable-option-checking'
'LDFLAGS= ' 'LIBS=-lrt -lpthread '

    Process Manager:                          pmi

    Launchers available:                       ssh rsh fork slurm
ll lsf sge manual persist

    Topology libraries available:               hwloc plpa

    Resource management kernels available:    user slurm ll lsf
sge pbs

    Checkpointing libraries available:

    Demux engines available:                  poll select



$ cc -v

Using built-in specs.

Target: x86_64-redhat-linux

Configured with: ../configure --prefix=/usr --
mandir=/usr/share/man --infodir=/usr/share/info --with-
bugurl=http://bugzilla.redhat.com/bugzilla --enable-bootstrap --
enable-shared --enable-threads=posix --enable-checking=release --
with-system-zlib --enable-__cxa_atexit --disable-libunwind-
exceptions --enable-gnu-unique-object --enable-
languages=c,c++,objc,obj-c++,java,fortran,ada --enable-java-
awt=gtk --disable-dssi --with-java-home=/usr/lib/jvm/java-1.5.0-
gcj-1.5.0.0/jre --enable-libgcj-multifile --enable-java-
maintainer-mode --with-ecj-jar=/usr/share/java/eclipse-ecj.jar --
disable-libjava-multilib --with-ppl --with-cloog --with-
tune=generic --with-arch_32=i686 --build=x86_64-redhat-linux

Thread model: posix

gcc version 4.4.7 20120313 (Red Hat 4.4.7-4) (GCC)




$ R
```

```
R version 3.1.0 (2014-04-10) -- "Spring Dance"
Copyright (C) 2014 The R Foundation for Statistical Computing
Platform: x86_64-redhat-linux-gnu (64-bit)
```

SPRINT TEAM

EMAIL: SPRINT@ED.AC.UK

HTTP://WWW.R-SPRINT.ORG