Gene expression

runibic: a Bioconductor package for parallel row-based biclustering of gene expression data

Patryk Orzechowski [1,2]*, Artur Pańszczyk [2], Xiuzhen Huang [3], and Jason H. Moore [1,*]

[1]Institute for Biomedical Informatics, University of Pennsylvania, Philadelphia, PA 19104, USA,
[2]Department of Automatics and Biomedical Engineering, AGH University of Science and Technology, al. Mickiewicza 30, 30-059 Krakow, Poland, and
[3]Department of Computer Science, Arkansas State University, Jonesboro, AR 72467, USA

*To whom correspondence should be addressed.

Associate Editor: XXXXXXX

Received on XXXXX; revised on XXXXX; accepted on XXXXX

Abstract

Motivation: Biclustering (called also co-clustering) is an unsupervised technique of simultaneous analysis of rows and columns of input matrix. From the first application to gene expression data, multiple algorithms have been proposed. Only a handful of them were able to provide accurate results and were fast enough to be suitable for large-scale genomic datasets.
Results: In this paper we introduce a Bioconductor package with parallel version of UniBic biclustering algorithm: one of the most accurate biclustering methods that have been developed so far. For the convenience of usage, we have wrapped the algorithm in an R package called *runibic*. The package includes: (1) a couple of times faster parallel version of the original sequential algorithm, (2) much more efficient memory management, (3) modularity which allows to build new methods on top of the provided one, (4) integration with the modern Bioconductor packages such as *SummarizedExperiment*, *ExpressionSet* and *biclust*.
Availability: The package is implemented in R (3.4) and will be available in the new release of Bioconductor (3.6). Currently it could be downloaded from the following URL: *http://github.com/athril/runibic/*
Contact: patryk.orzechowski@gmail.com, jhmoore@upenn.edu
Supplementary information: Supplementary informations are available in vignette of the package.

1 Introduction

The recent advantages in transcriptomic analysis, including development of high-throughput and high-resolution platforms including RNA-seq, single-cell RNA-sequencing (scRNA-seq) or high-throughput PCR have allowed to design experiments that provide datasets with even hundreds of thousands columns and thousands rows. This have set new requirements for data analytics. Modern methods need to yield accurate results. They are required to handle large datasets and are expected to finish computations in reasonable time.

With growing amount of genomic data there is an urgent need for efficient and precise methods that are able to capture the underlying patterns in gene expression datasets. One of the techniques that proved to be very insightful in gene expression analysis is biclustering, which allows to detect subsets of genes and samples in complex and noisy data. Biclustering is considered NP-hard as it investigates relations between multiple rows that occur in different subsets of columns. The running time of the algorithms is usually highly dependent on the size of the input data.

The vast majority of biclustering methods are sequential. There are a couple of common reasons for this. Some methods are specifically designed to yield only one bicluster at a time. Each run of the algorithm depends on the previous findings. Other methods use graph-based structures, which are difficult to parallelize, or perform hardly scalable statistical analyses. For some group of the methods parallel implementation may not be beneficial, as they extensively use binary operations. Bioconductor in version 3.5 provides the following biclustering methods and packages for gene expression analysis:

- ISA (Bergmann *et al.*, 2003) - implemented in eisa and isa2 Bioconductor packages (Csardi *et al.*, 2010),
- CC (Cheng and Church, 2000), Plaid methods Lazzeroni and Owen (2002), Bimax (Prelić *et al.*, 2006), xMotifs (Murali and Kasif, 2003), Quest (Kaiser, 2011), Spectral Kluger *et al.* (2003) - all available in biclust package (Kaiser *et al.*, 2015),

- FABIA, FABIAS, FABIAP - available in Bioconductor package fabia (Hochreiter *et al.*, 2010),
- HapFABIA - implemented in package hapfabia (Hochreiter, 2013)
- QUBIC (Li *et al.*, 2009) - implemented in more modern package QUBIC (Zhang *et al.*, 2017) and package older rqubic (Zhang, 2015),
- MCbiclust - available in Bioconductor package MCbiclust (Bentham, 2017),
- SSVD (Lee *et al.*, 2010) and S4VD (Sill *et al.*, 2011) - available in Bioconductor package s4vd Sill and Kaiser (2015),
- Iterative Binary Biclustering of Gene sets - available in Bioconductor package iBBiG Gusenleitner *et al.* (2012)

The vast majority of the method are implemented in R, which is slower than C. Some of the methods, e.g. QUBIC, benefit from calls to high-performance C++ linear algebra libraries, such as *Rcpp* (Eddelbuettel and François, 2011) and *RcppArmadilo* (Eddelbuettel and Sanderson, 2014). The comparison of R packages functionality is presented in Table 1.

Table 1. Comparison of functionalities of different R packages. (*) - Only Bimax algorithm uses wrapped C function call.

Description	runibic	QUBIC	biclust*	s4vd	fabia	isa2
Support for numeric and integer datasets						
Parallel implementation of methods						
Integration with Biclust						
Integration with SummarizedExperiment						
Uses C/C++ routines			(*)			

One of the recent breakthroughs in gene expression analysis was UniBic (Wang *et al.*, 2016). The algorithm originally implemented in C managed to capture biologically meaningful trend-preserving patterns and proved to outperform multiple other methods. The method also showed great potential for parallelization. Unfortunately the implementation of the method wasn't efficient enough and the code had some memory leaks.

2 Methods

In this paper we introduce a Bioconductor package called *runibic* with parallel implementation of one of the most accurate biclustering methods: UniBic. The algorithm, orignally released as sequential, has proven to outperform multiple popular biclustering state-of-the-art biclustering methods (Wang *et al.*, 2016). We have redesigned the code and reimplemented the method into more modern C++11 programming language. By parallelizing chunks of the code using OpenMP standard Dagum and Menon (1998), we obtained approximately up to a couple of times speedup in terms of execution time for popular genomic datasets. By fixing some of the memory management bugs, our package provides more stable and reliable implementation of UniBic algorithm.

2.1 Implementation

In the provided implementation of UniBic algorithm, we migrated the original code from C to C++11 programming language and added OpenMP support. Code refactoring allowed us to take advantage of multiple aspects of modern-style language programming:

- safer and more modern memory management replaced difficult to maintain C style memory allocations and deallocations,
- fast and efficient containers from Standard Template Library (STL), such as vectors, sets and algorithms, were used for acceleration of common operations like iterate, sort, search, count, or copy,
- the original implementation in most cases allocated a large number of simple arrays and used loops with slow indexing for common operations,
- removing many redundant copying and memory allocations,
- fixing a couple of memory leaks, which caused segmentation fault for some datasets.

Porting the code improved interpretability of the code allowed to remove multiple redundancies present in the previous UniBic implementation. For example, we replaced the original four functions that calculated the Longest Common Subsequence with a single one with multiple options. Similar improvements were made in other code sections, for example: in discretization, in calculation of Longest Common Subsequence between each pair of rows, in clustering and bicluster expansion parts. In order to provide more insightful analysis into the modules of UniBic algorithm, we separated and exported the major steps of the original method. Thus, the algorithm may be run using either a single command, or executed step by step. This provides much better control, improves clarity of the method, and allows its future customization. The algorithm provided in *runibic* package is divided into the following sections:

- *set_runibic_params* - a function that sets parameters for algorithm,
- *runiDiscretize* - the original UniBic discretize approach, which take into account the number of ranks from 'div' parameter and quantile value from 'q' parameter (Step 1 of the method),
- *unisort* - a function that sorts the rows of a matrix and returns the indexes of sorted columns in each row (Step 2) ,
- *calculateLCS* - a function that calculates the Longest Common Subsequence (LCS) between each unique pair of rows in the matrix, returns a list of LCS lengths and row pairs (Step 3),
- *cluster* - the main biclustering method which builds biclusters based on the input data and *calculateLCS* results (Steps 4 and 5).

By designing a modular structure of the package we intended to simplify flexible modifications of the original algorithm. Such methods may use different preprocessing or ranking techniques, or expand bicusters using different rows as seeds. An example includes different method of sorting results from *calculateLCS*. The proposed method, which is based on a stable STL sort, could be used as an alternative to the old C style pointer and sorting based on Fibonacci Heap. In our opinion the proposed method is more robust and better reflects the original intention. The choice of the method may implicate the outcome of the algorithm, as different LCSes of the same length may be chosen as seeds.

2.2 Parallelization

In order to improve the algorithm execution time the most crucial and computationally intensive parts of the code were parallelized using OpenMP standard. One of the most time consuming steps of UniBic is calculating Longest Common Subsequence (LCS) between unique pairs of rows. We rearranged the code and achieved parallelization where each core of the CPU calculates unique LCS between unique pair of rows simultaneously. Similarly, we also paralleled the data preprocessing required by the method, so as expansions of each of the biclusters, which required calculations of LCS between each row and the seed. All mentioned operations allowed us to obtain biclustering results in several minutes on the modern computer with modern processor.

2.3 Integration with Bioconductor packages

The *runibic* package takes advantage of *Rcpp* library that allows seamless integration of C++ code with R environment. The *runibic* package is also integrated with *biclust* package methods for biclustering process. Results

returned from *runibic* are wrapped into a *Biclust* object, which can be used for further examination, including visualization and analysis provided by *biclust* package. The examples of usage are presented in Supplementary material as well as in the package manual.

```
library(runibic)
library(biclust)
test <- matrix(rnorm(1000), 100, 100)
res <- runibic(test)
res <- biclust::biclust(test, method = BCUnibic())
```

Similarly, the biclust method could be applied to any matrix extracted from *ExpressionSet* using $exprs()$ function.

2.4 Support for SummarizedExperiment

Apart from allowing analysis of genomic data from historical *ExpressionSet*, *runibic* package is compatible with *SummarizedExperiment* class (Morgan *et al.*, 2017). This class offers much more flexibility in terms of experiment design and supports both RNA-Seq and Chip-Seq. This makes *runibic* a very easy tool for performing biclustering analysis of modern genomic experiments. An example of using *runibic* with Single-Cell RNA-Seq Datasets is provided in Supplementary material.

3 Results

To investigate running times of the method, we have applied it to several popular datasets. The running times of the revised and the original UniBic algorithm as well as the revised parallel version are presented in Table 2.

Table 2. Running times of the original version of UniBic Wang et al. (2016) and parallel UniBic in R from Bioconductor package.

Dataset	Rows	Columns	UniBic run time(s)	R-UniBic run time(s)	Improved
Escherichia coli	4297	466	1478.3	290.9	80.3%
GSE66913	16436	167	546.7	67.6	87.6%
GSE42408	25662	208	3305.3	821.6	75.1%
airway	64102	8	903.9	487.0	47.1%

By refactoring and optimizing the code, the new parallel version of UniBic biclustering algorithm provided by *runibic* package is approximately 2-5 times faster than the original version of the algorithm for popular genomic datasets.

4 Conclusions

In this paper we introduce *runibic* package with revised and parallelized version of UniBic algorithm. The package is going to be available in the newest Bioconductor 3.6 release. . Providing a modular structure of the package allows to easily understand steps of the method and makes code much more interpretable. The *runibic* package provide *runibic* method that could be applied to any matrix in R, expression set extracted from *ExpressionSet* or *SummarizedExperiment* class. Integration with many common R and Bioconductor packages (e.g. *biclust*, *QUBIC*), as well as extensive documentation on one of the most accurate biclustering methods developed so far, make *runibic* package easily accessible and very flexible for gene expression analysis.

Funding

This research was supported in part by PL-Grid Infrastructure and by grant LM012601 from the National Institutes of Health (USA).

References

Bentham, R. (2017). *MCbiclust: Massive correlating biclusters for gene expression data and associated methods*. R package version 1.0.1.

Bergmann, S., Ihmels, J., and Barkai, N. (2003). Iterative signature algorithm for the analysis of large-scale gene expression data. *Physical review E*, **67**(3), 031902.

Cheng, Y. and Church, G. M. (2000). Biclustering of expression data. In *Proceedings of the eighth international conference on intelligent systems for molecular biology*, volume 8, pages 93–103.

Csardi, G., Kutalik, Z., and Bergmann, S. (2010). Modular analysis of gene expression data with r. *Bioinformatics*, **26**, 1376–7.

Dagum, L. and Menon, R. (1998). Openmp: an industry standard api for shared-memory programming. *IEEE computational science and engineering*, **5**(1), 46–55.

Eddelbuettel, D. and François, R. (2011). Rcpp: Seamless R and C++ integration. *Journal of Statistical Software*, **40**(8), 1–18.

Eddelbuettel, D. and Sanderson, C. (2014). Rcpparmadillo: Accelerating r with high-performance c++ linear algebra. *Computational Statistics & Data Analysis*, **71**, 1054–1063.

Gusenleitner, D., Howe, E. A., Bentink, S., Quackenbush, J., and Culhane, A. C. (2012). ibbig: iterative binary bi-clustering of gene sets. *Bioinformatics*, **28**(19), 2484–2492.

Hochreiter, S. (2013). Hapfabia: identification of very short segments of identity by descent characterized by rare variants in large sequencing data. *Nucleic acids research*, **41**(22), e202–e202.

Hochreiter, S., Bodenhofer, U., Heusel, M., Mayr, A., Mitterecker, A., Kasim, A., Khamiakova, T., Van Sanden, S., Lin, D., Talloen, W., Bijnens, L., G"ohlmann, H. W. H., Shkedy, Z., and Clevert, D.-A. (2010). FABIA: Factor analysis for bicluster acquisition. *Bioinformatics*, **26**(12), 1520–1527. doi:10.1093/bioinformatics/btq227.

Kaiser, S. (2011). *Biclustering: methods, software and application*. Ph.D. thesis, lmu.

Kaiser, S., Santamaria, R., Khamiakova, T., Sill, M., Theron, R., Quintales, L., Leisch, F., and De Troyer., E. (2015). *biclust: BiCluster Algorithms*. R package version 1.2.0.

Kluger, Y., Basri, R., Chang, J. T., and Gerstein, M. (2003). Spectral biclustering of microarray data: coclustering genes and conditions. *Genome research*, **13**(4), 703–716.

Lazzeroni, L. and Owen, A. (2002). Plaid models for gene expression data. *Statistica sinica*, pages 61–86.

Lee, M., Shen, H., Huang, J. Z., and Marron, J. (2010). Biclustering via sparse singular value decomposition. *Biometrics*, **66**(4), 1087–1095.

Li, G., Ma, Q., Tang, H., Paterson, A. H., and Xu, Y. (2009). QUBIC: a qualitative biclustering algorithm for analyses of gene expression data. *Nucleic acids research*, **37**(15), e101–e101.

Morgan, M., Obenchain, V., Hester, J., and PagÃ¨s, H. (2017). *SummarizedExperiment: SummarizedExperiment container*. R package version 1.6.5.

Murali, T. and Kasif, S. (2003). Extracting conserved gene expression motifs from gene expression data. In *Pacific symposium on biocomputing*, volume 8, pages 77–88.

Prelić, A., Bleuler, S., Zimmermann, P., Wille, A., Bühlmann, P., Gruissem, W., Hennig, L., Thiele, L., and Zitzler, E. (2006). A systematic comparison and evaluation of biclustering methods for gene expression data. *Bioinformatics*, **22**(9), 1122–1129.

Sill, M. and Kaiser, S. (2015). *s4vd: Biclustering via Sparse Singular Value Decomposition Incorporating Stability Selection*. R package version 1.1-1.

Sill, M., Kaiser, S., Benner, A., and Kopp-Schneider, A. (2011). Robust biclustering by sparse singular value decomposition incorporating stability selection. *Bioinformatics*, **27**(15), 2089–2097.

Wang, Z., Li, G., Robinson, R. W., and Huang, X. (2016). Unibic: Sequential row-based biclustering algorithm for analysis of gene expression data. *Scientific reports*, **6**.

Zhang, J. D. (2015). *rqubic: Qualitative biclustering algorithm for expression data analysis in R*. R package version 1.22.0.

Zhang, Y., Xie, J., Yang, J., Fennell, A., Zhang, C., and Ma, Q. (2017). QUBIC: a bioconductor package for qualitative biclustering analysis of gene co-expression data. *Bioinformatics*, **33**(3), 450–452.