Libertas Academica
FREEDOM TO RESEARCH

Open Access: Full open access to this and thousands of other papers at http://www.la-press.com.

Evolutionary Bioinformatics

RapidMic: Rapid Computation of the Maximal Information Coefficient

Dongming Tang[1], Mingwen Wang[2], Weifan Zheng[1] and Hongjun Wang[3]

[1]Institute of Information Research, Southwest Jiaotong University, Chengdu, China. [2]School of Mathematics, Southwest Jiaotong University, Chengdu, China. [3]School of Information Science and Technology, Southwest Jiaotong University, Chengdu, China.

ABSTRACT: To discover relationships and associations rapidly in large-scale datasets, we propose a cross-platform tool for the rapid computation of the maximal information coefficient based on parallel computing methods. Through parallel processing, the provided tool can effectively analyze large-scale biological datasets with a markedly reduced computing time. The experimental results show that the proposed tool is notably fast, and is able to perform an all-pairs analysis of a large biological dataset using a normal computer. The source code and guidelines can be downloaded from https://github.com/HelloWorldCN/RapidMic.

KEYWORDS: algorithms, gene expression, software, computational biology, statistical analysis

CITATION: Tang et al. RapidMic: Rapid Computation of the Maximal Information Coefficient. *Evolutionary Bioinformatics* 2014:10 11–16 doi: 10.4137/EBO.S13121.

RECEIVED: September 3, 2013. **RESUBMITTED:** November 14, 2013. **ACCEPTED FOR PUBLICATION:** November 17, 2013.

ACADEMIC EDITOR: Jike Cui, Associate Editor

TYPE: Short Report

FUNDING: Supported by the National Natural Science Foundation of China under Grant No. 61100118, 61373009 and 61003142. Supported by the Fundamental Research Funds for the Central Universities under Grant No. SWJTU11BR093 and Science and Technology Support Project of Sichuan province under Grant No. 2013GZX0166.

COMPETING INTERESTS: Author(s) disclose no potential conflicts of interest.

CORRESPONDENCE: tdm_2010@swjtu.edu.cn

Introduction

David N. Reshef and his colleagues recently published a paper that introduced a measure of dependence for two-variable relationships: the maximal information coefficient (MIC)[1]. MIC can be used as a metric for the exploration of large datasets, and the detection of close associations between tens of thousands of variable pairs in large datasets. MIC uncovers variables that not only have functional associations but are also statistically independent. MIC is part of a larger family of maximal information-based nonparametric exploration (MINE) statistics. In addition to the MIC measure, MINE provides three additional measures: maximum asymmetry score (MAS), maximum edge value (MEV), and minimum cell cumber (MCN). As a measure of dependence, the MIC has two heuristic properties: generality and equitability. These characteristics make the MIC particularly suitable for applications in bioinformatics; David N. Reshef and others have tested the MIC using many real biology datasets to demonstrate its characteristics and compared the MIC to a wide range of methods, including Pearson/Spearman correlations and Mutual Information.

To date, the MIC has been applied successfully in various bioinformatics fields. For example, the MIC has been used as a measure to convert records of biological annotations into networks of the associated annotations.[2] Das et al.[3] applied the MIC to explore the global expression dynamics of different interaction datasets from humans and yeast. Lin et al.[4] adopted the MIC to define a co-expression network and indicated that the MIC has clear theoretical advantages in regard to capturing general dependence patterns. Co-expression methods have been widely applied for the analysis of gene expression data. Co-expressed genes with correlated pathways may share common functional tasks and regulatory mechanisms.[5] The MIC can help detect co-expressed genes or tissues and find biologically meaningful relationships among genes. In addition, correlation-based methods are the most straightforward way to explore a gene co-expression network.[5] Therefore, the MIC can also help define gene co-expression similarity relationships in a co-expression network. The cluster analysis method can employ the MIC to measure the pair-wise transcription correlation coefficients between genes and to then cluster together genes or tissues that manifest similar expression pattern.[6]

The authors have a website that provides a compiled Java program named MINE.jar, which can be run in a Java runtime environment. However, the provided MINE.jar exhibits several limitations that hinder its availability in real applications. First, the time and space complexities of execution are too large for the analysis of large-scale datasets. Second, MINE.jar often throws a Java runtime exception, such as OutOfMemoryError. In addition, it is not an open-source software. In brief, MINE.jar is not feasible for the analysis of very long biology sequences, particularly for the analysis of all pairs of variables against each other. To reduce memory requirements and computing time, Davide Albanese et al. provided a C implementation of the MIC measure.[7] However, the developed program is a serial program and does not achieve a distinct improvement. Current computer architectures are increasingly relying on hardware level parallelism to improve performance: multiple execution units, pipelined instructions, and multi-core. With the development of next-generation sequencing (NGS) technologies, high-dimensional gene datasets have been produced. To exploit these large-scale datasets, bioinformatics should make full use of modern computational approaches to process the collected data. At the same time, the quality and speed of computational approaches, such as the construction of large-scale networks of gene interactions for protein-protein interactions (PPI), gene co-expression, and data and gene coregulation data, must be considered carefully. To this end, we propose a rapid maximal information-based nonparametric exploration tool based on parallel computing for the discovery of relationships and associations in large-scale datasets.

Methods

From a general point of view, parallel computing is the simultaneous use of multiple computer resources to solve a computational problem. In theory, the introduction of additional resources to perform a task will shorten the time required to complete the task and result in potential cost savings. Compared with serial computing, parallel computing is much better suited for modeling, simulating, and understanding complex, real-world phenomena. A single computer resource can only perform one task at a time. Multiple computing resources can perform many tasks simultaneously. In our implementation, we employ the standard POSIX thread specification, which defines a set of C language programming types and procedure calls. Following the rule of "Don't Reinvent the Wheel",[8] we partly reused, optimized, and adapted the libmine.[7] To design a parallel program, the first step is to break the problem into discrete parts that can be distributed into multiple tasks to be performed concurrently. In our program, we provide four analysis styles, each of which involves a different decomposition or partitioning strategy.

First, we introduce our notational conventions and definitions. Let $D=\{(x_i, y_i), i=1...n\}$ represent a set of ordered pairs (x, y), and G represent an x-*by*-y grid of D, where the x-values of D are divided into x bins and the y-values are divided into y bins. The $D|_G$ can be the distribution of pairs (x, y) on the x-*by*-y grid. Obviously, different grids G lead to different distributions $D|_G$ for a fixed D. First, we introduce the definition of MIC as the following formula: (1) $I^*(D, x, y) = \max I(D|_G)$, where $I^*(D, x, y)$ denotes the maximum mutual information of $D|_G$ with the x-*by*-y grid. As a result, we can define a characteristic matrix $M(D)$ of set D in terms of I^*. The entry $M(D)_{x,y}$ of matrix $M(D)$ can be defined as $M(D)_{x,y} = \frac{I^*(D,x,y)}{\log \min\{x, y\}}$. The MIC value of a given set D of two-variable data of size n and a grid size less than $B(n)$ is defined as $MIC(D) = \max_{xy< B(n)} \{M(D)_{x,y}\}$, where $\omega(1) < B(n) \leq O(n^{1-\varepsilon})$ and $0 < \varepsilon < 1$. The function $B(n)$ limits the sizes of the grids when searching over feasible partitions. If the value of $B(n)$ is too high, the random dataset will obtain a nonzero MIC score. If the value of $B(n)$ is too low, the search space is clearly limited. To approximate the choice of $B(n)$, the original authors suggest a default $B(n) = n^{0.6}$. In practice, we can set the value of α for function $B(n) = n^a$ based on an empirical test, the size of the data set, or a balance of the time available and size of the data.

Next, we briefly describe the serial implementation of MIC using pseudocode as follows:

Algorithm 1. Compute the maximal information coefficient.

```
Require: D={(x_i, y_i), i=1...n} is a set of ordered (x, y) pairs
Require: α is the exponent in B(n) = n^a and must be in (0,1]
Require: c determines the maximal number gx*c of clumps when attempting to draw x-axis gridlines with gx, c must be > 0.
compute_mic(D, α, c)
{
    B = MAX(n^a, 4);
    bin_max = MAX((int)floor(B/2.0), 2);
    characterMatrix[][];
    /* x vs. y */
    for (i = 2 to bin_max)
        characterMatrix[i][] = FixOnePartition(x,y,i);
    /* y vs. x */
    for (i = 2 to bin_max)
        characterMatrix[][i] = MAX( FixOnePartition(y, x, i), characterMatrix[][i] );
    return getResult();
}
FixOnePartition(x, y, nbins)
{
    gx = (int) floor(B / nbins);
    K = MAX( (c * gx), 1);
    EquipartitionYAxis(y, nbins);
    nK = GetSuperclumpsPartition(x, K);
    return ApproxOptimizeXAxis(x, y, nK);
}
```

From the outline of MIC algorithm, we can observe that the core of the MIC is to generate the characteristic matrix. If D is distributed into an m-*by*-k grid, the data associated with the *x-axis* will be divided into m bins, and the data associated with the y-axis will be partitioned into k bins. To obtain the highest mutual information of the m-*by*-k grid, we should try every possible y-axis partition of size k and every possible x-axis partition of size m. Obviously, it is infeasible to check all of the possible partitions. Therefore, the MIC algorithm adopts a natural heuristic approach to solve this problem. The main idea of the heuristic approach is to try only those grids in which at least one axis is equipartitioned. As a consequence, the FixOnePartition function will find and return the highest mutual information array for different x partitions with a fixed partition of size *nbins* on y. The variable *characterMatrix*[i][j] denotes the entry $M(D)_{i,j}$ of the character matrix $M(D)$. In FixOnePartition, EquipartitionYAxis first equipartitions y into fixed *nbins* bins, and ApproxOptimizeXAxis then finds different partitions for x that will achieve the maximal mutual information using dynamic programming. For (x, y) pairs of D, there exists a partition on x that relies on the partition Q of y of the corresponding pairs (x, y). The ApproxOptimizeXAxis function restricts itself to drawing x-axis partition lines only between runs of consecutive points that fall in the same row of the y-axis partition Q (called clumps); thus, the GetSuperclumpsPartition function limits the number of clumps for the x-axis created by a given y-axis partition Q of size y imposed on D.

Finally, we introduce four parallel analysis styles in the RapidMic tool. The data analyzed by the tool is a comma-separated values (CSV) file; we can regard the data as an $m \times n$ matrix, in which the rows represent the variables and the columns represent the feature values of the variables. In the following, we provide a detailed description of these four parallel analysis styles.

1. *OnePairAnalysis* will compare one pair variables i and j. If the length of the input variable sequence is greater than a specific value, the calculation will initiate the processing of multiple threads to avoid the time consumption associated with the synchronization between the threads from affecting the gains afforded by the parallel execution of a short sequence. The method divides the for loop subroutine in the abovementioned Algorithm 1 into multiple work threads and each work thread will find the highest mutual information attainable for each given grid (x, y). The argument passed to each thread has a structure that contains a series of grids (x, y) with x columns and y rows on the data D. The advantage of using a thread group instead of a normal serial program is that several operations may be carried out in parallel; thus, events can be handled immediately as they arrive. As described in the abovementioned Algorithm 1, the compute_mic function will call the FixOnePartition function $2^*(bin\_max\text{-}1)$ times. In addition, the FixOnePartition function is a heavy-computing task due to the use of dynamic programming. From the outline of MIC algorithm, we can conclude that the operations EquipartitionYAxis and GetSuperclumpsPartition have worst case time $O(n)$. The operation ApproxOptimizeXAxis has worst case time $O(K^{2*}B) = O((B/nbin^*c)^{2*}B)$. As a result, the MIC algorithm has worst case time $O(2^* \sum_{nBins=2}^{bin\_max} (B / nbins^* c)^2 * B)$. In real implementation, we first design a data structure to encapsulate the parameters for the FixOnePartition function, then divide $2^*(bin\_max\text{-}1)$ times FixOnePartition calling of two "for (i = 2 to *bin_max*)" loops in the compute_mic function into multiple work threads, and immediately start each working thread. At this time, the process will wait for a condition variable until all work threads are completed.
2. *AllPairsAnalysis* will compare all pairs of variables against each other. For an $m \times n$ matrix, *AllPairsAnalysis* will compute the MIC value $(m^*(m\text{-}1))/2$ times. To speed up the calculation, *AllPairsAnalysis* divides these computing tasks into multiple work threads. In algorithm 1, $D=\{(x_i, y_i), i=1...n\}$ is a set of ordered pairs. At the same time, the process of computing the MIC requires the performance of a sorting operation on x and y sequences several times. A partly batch sort algorithm based on pairwise sort scheme is employed for the implementation; in this algorithm, the several sorts can be performed by a single sorting operation. The analysis of the same variable to other variable will be divided into one work thread to avoid the frequent operation of data sorting and copying. In addition, we adopt a shared memory model to hold all of the global or common data and thus avoid many repeated calculations.
3. *TwoSetsAnalysis* will compare each of the first i variables for each of the rest of the variables. For an $m \times n$ matrix, *TwoSetsAnalysis* will compute the MIC value $i^*(m\text{-}i)$ times. Here, *TwoSetsAnalysis* employs a divide-and-conquer strategy similar to that of *AllPairsAnalysis* to divide the computational tasks.
4. *MasterAnalysis*, will compare variable i to the rest of the variables. For an $m \times n$ matrix, *MasterAnalysis* will compute the MIC value (m-1) times. Similarly, these computing tasks will be broken into multiple parts.

Table 1. Comparison of the computing time (seconds) for case 1.

N	MINE.JAR	MINERVA	MINEMAT	MINEC++	RAPIDMIC
10000	5.651	43.575	65.452	52.584	1.420
20000	13.689	375.707	376.086	334.551	5.456
50000	81.256	4172.569	4219.81	3616.941	27.05
100000	604.114	26442.04	25847.2	22448.676	101.608

Notes: n, number of values for the variable. The computing time excludes the time required to read the data file and write the result file. Minerva enables parallelization using a multicore package.

Table 2. Comparison of the computing time (seconds) for case 2.

DATA	*N*	*M*	MINE.JAR	MINERVA	MINEMAT	MINEC++	RAPIDMIC
Spellman	4381	23	–	4304.86	2933.59	2267.198	1060.649
MLB2008	132	337	1476.938	430.489	1170.25	1195.025	350.142

Notes: *n*, number of variables. *m*, number of features. The computing time excludes the time required to read the data file and write the result file. Minerva enables parallelization using a multicore package.

In general, heavy-computing tasks and huge datasets are decomposed in our threaded implementation, and one parallel task works on a portion of the data. Moreover, the implementation decreases the frequency of data copying, data moving, memory allocation and destroying, and loop and iteration operations. Finally, the main process obtains the computational results through a shared memory and thread signal condition variable synchronization mechanism.

Results

To evaluate the performance of the proposed implementation, we analyzed four typical cases using simulated and real-life datasets and compared the results with the implementation proposed by Davide Albanese et al.[7] (including the R wrapper Minerva, matlab wrapper Minemat, and C++ wrapper Minec++) and MINE.jar. In addition, we verified the reliability and consistency of the results between our parallel method and the serial method MINE.jar. All of the experiments were conducted on a Macbook Pro computer with Mac OS Lion 10.7, I5 CPU, and 6 GB of RAM. To be fair, we obtained the average result of five repeated executions, and employed the same parameters ($\alpha = 0.67$ and $c = 15$) as in the paper.[7] The experimental datasets can be downloaded from https://github.com/HelloWorldCN/RapidMic.

Case 1: Compare two long variables. In this case, the analysis is conducted on two long variables, which have a functional relationship: $y = x^2+x+1$. We generated four data series of 10,000, 20,000, 50,000, and 100,000 points; the value of x ranged from 1 to the number of points in steps of 1. Here, we selected the values of x and y as two input variables to perform the analysis of one pair of variables with parameters $\alpha = 0.67$ and $c = 15$. All of the results are shown in Table 1. The first column indicates the number of values for the variable; the rest of the columns contain the computing time for MINE.jar, Minerva, Minemat, Minec++, and RapidMic,. As observed in Table 1, RapidMic performs better than the others. In particular, RapidMic performs nearly four times faster than MINE.jar and 40 times faster than Minerva. MINE.jar also exhibits a good performance in this case and is significantly better than Minerva, Minemat, and Minec++. A general trend that can be observed is that the computing time tends to increase rapidly as the number of points is increased. However, even if the number of points is increased to 100,000, RapidMic still rapidly obtains the result, in 101.608 seconds. In contrast, Minec++ requires more than six hours to obtain a result for 100,000 points, which is not a reasonable and acceptable time. Obviously, excessive amounts of serial "for loop and while loop" iterations would reduce the performance of the application.

Case 2: Compare all pairs. In this case, we use the above-mentioned methods successively to analyze all of the variables with each other on Spellman and MLB2008. The Spellman dataset is a cdc15 expression dataset,[9] with 4381 time series. The MLB2008 dataset is the salaries of the Major League Baseball players for the 2008 season.[1] Here, we compare the time series against each other with parameters $\alpha = 0.67$ and $c = 15$. The various implementations discussed are compared based on their time performance; the experimental results are shown in Table 2. As observed in Table 2, the computing time of the methods is clearly proportional to the number of variables used in the analysis of all pairs. RapidMic significantly outperforms all of the other methods. Specifically, RapidMic performs two times faster than Minec++ on Spellman, and three times faster than Minec++ on MLB2008. The previous case has experimentally shown that MINE.jar is acceptable for the analysis of two long variables, but the previously observed advantage disappears when many variables are taken into account. In fact, MINE.jar performs significantly worse than the others in this case. More specifically, because of the large number of pairs in this case, MINE.jar fails to obtain a result for Spellman in our experimental computer. Finally, MINE.jar throws a Java exception (OutOfMemory) after a long time even if the Java Virtual Machine Runtime memory is increased to 5 GB. The data shown in Table 2 reveal several phenomena: first, Minerva performs better than Minemat and

Table 3. Comparison of the computing time (seconds) for case 3.

DATA	*N*	*M*	MINE.JAR	MINERVA	MINEMAT	MINEC++	RAPIDMIC
RNA	20422	16	2.440	2.905	1.275	0.831	0.378
Spellman	4381	23	0.41	1.954	1.283	1.108	0.219

Notes: *n*, number of variables. *m*, number of features. The computing time excludes the time required to read the data file and write the result file. Minerva enables parallelization using a multicore package.

Table 4. Comparison of the computing time (seconds) for case 4.

DATA	*N*	*M*	*I*	MINE.JAR	MINERVA	MINEMAT	MINEC++	RAPIDMIC
RNA	20422	16	3000	-	3164.7	3580.5	3060.2	1285.3
Spellman	4381	23	2190	-	3280.6	1297.8	1118.6	443.3

Notes: *n*, number of variables. *m*, number of features. *i*, the first *i* variables compared with each of the remaining variables. The computing time excludes the time required to read the data file and write the result file. Minerva enables parallelization using a multicore package.

Minec++ when the number of analysis pairs is low; second, Minemat and Minec++ have an advantage over Minerva for datasets with many variables, because the C/C++ core has superior capability for the rapid manipulation of data.

Case 3: Compare one to others. In this case, we run all of the tools comparing one variable to the others on two datasets: an RNA-sequencing dataset and the Spellman dataset.[7,9,10] The RNA sequencing dataset originated from the RNA sequencing of 15 lung adenocarcinomas, including eight with the KRAS mutation and seven without mutation.[10] Briefly, the dataset has 20,422 genes, and 16 features after preprocessing. Here, we compare the first gene against the others with parameters $\alpha = 0.67$ and $c = 15$. The experimental results are shown in Table 3. The results show that RapidMic substantially outperforms the others in the analysis of both the RNA sequencing data and the Spellman dataset. In particular, RapidMic delivers the best execution time, performing seven times and five times faster than Minerva on the RNA sequencing dataset and the Spellman dataset, respectively.

Case 4: Compare two sets. In this case, we compare each of the first *i* variables to each of the rest of the variables on the RNA sequencing and Spellman datasets.[7,9,10] We compare the first 2190 and 3000 genes on the RNA sequencing dataset and Spellman dataset, respectively, against the others using the parameters $\alpha = 0.67$ and $c = 15$. The experimental results are shown in Table 4. In this case, RapidMic also exhibits the best execution time for the analysis of the two datasets. MINE.jar fails to obtain a result in our experimental computer.

In these cases, RapidMic can obtain results faster than the others. Because Minerva, Minemat, and Minec++ have the same core C implementation, we hypothesize that Minerva is acceptable for the analysis of moderate-scale variable pairs. Minemat and Minec++ exhibit similar performance. The results also show that RapidMic based on parallelization technology always produces a better performance than the serial program.

Consistency analysis. To verify the reliability and consistency of the results between our parallel method and the serial method MINE.jar, we compare their MIC values. Here, we conduct "one vs. others" experiments on the Spellman dataset with $\alpha = 0.67$ and $c = 15$ and select the first variable "Time" as the master variable. All of the numerical values are rounded to five-digit precision. Table 5 shows the distribution range of the absolute difference between RapidMic and the serial method MINE.jar. The first row of Table 5 indicates the range of the absolute difference, and the second row represents the number of absolute differences in the corresponding range. As shown in Table 5, 2193 of the total 4381 MIC values are identical. The summary statistics for the values of these differences can then be concluded: Min is 0, 1st Quartile is 0, Median is 0, Mean is 0.000004995, 3rd Quartile is 0.00001001, and Max is 0.00001001. These subtle differences are partly caused by the use of different floating-point numbers in the computing process: RapidMic (double) and MINE.jar (float). In addition, some real implementation may cause subtle numerical differences.

Table 5. Analysis of the consistency between our parallel method RapidMic and the serial method MINE.jar.

Range	Identical	(0, 0.00001]	(0.00001, 0.00002]
Number	2193	644	1544

Conclusions

We present a new rapid maximal information-based nonparametric exploration tool for the statistical analysis of large-scale datasets. Through the parallel processing of the MIC algorithm, the provided tool can effectively analyze large-scale datasets and greatly reduce computing time. The experimental results show that the proposed tool is very rapid and is able to perform the analysis of all pairs on a large biological dataset using a normal computer. Overall, we find that our implementation yields competitive performance. In addition, the presented tool is relatively simple and easy to use. We also provide a matlab interface and wrapper. Our implementation is an open-source and cross-platform project that can help the design of software for the analysis of biological data that reuses and integrates the MIC algorithm, and enhances parallelizability and extensibility. This result becomes particularly interesting in the emerging field of bioinformatics, which is coping with the enormous quantities of biological data being generated by recently developed technological approaches.

Author Contributions

Conceived and designed the experiments: DT, MW, WZ, HW. Analyzed the data: DT, MW. Wrote the first draft of the manuscript: DT. Contributed to the writing of the manuscript: DT, MW, WZ, HW. Agree with manuscript results and conclusions: DT, MW, WZ, HW. Jointly developed the structure and arguments for the paper: DT, MW, WZ. Made critical revisions and approved final version: DT, MW,

WZ, HW. All authors reviewed and approved of the final manuscript.

DISCLOSURES AND ETHICS

As a requirement of publication the authors have provided signed confirmation of their compliance with ethical and legal obligations including but not limited to compliance with ICMJE authorship and competing interests guidelines, that the article is neither under consideration for publication nor published elsewhere, of their compliance with legal and ethical guidelines concerning human and animal research participants (if applicable), and that permission has been obtained for reproduction of any copyrighted material. This article was subject to blind, independent, expert peer review. The reviewers reported no competing interests.

REFERENCES

1. Reshef DN, Reshef YA, Finucane HK, et al. Detecting novel associations in large data sets. *Science*. December 16, 2011;334(6062):1518–24.
2. Karpinets TV, Park BH, Uberbacher EC. Analyzing large biological datasets with association networks. *Nucleic Acids Res*. September 1, 2012;40(17):e131.
3. Das J, Mohammed J, Yu H. Genome-scale analysis of interaction dynamics reveals organization of biological networks. *Bioinformatics*. July 15, 2012;28(14):1873–8.
4. Song L, Langfelder P, Horvath S. Comparison of co-expression measures: mutual information, correlation, and model based indices. *BMC Bioinformatics*. December 9, 2012;13(1):328.
5. Allen JD, Xie Y, Chen M, Girard L, Xiao G. Comparing statistical methods for constructing large scale gene networks. *PLoS One* 2012;7(1):e29348.
6. Priness I, Maimon O, Ben-Gal I. Evaluation of gene-expression clustering via mutual information distance measure. *BMC Bioinformatics*. 2007;8:111.
7. Albanese D, Filosi M, Visintainer R, Riccadonna S, Jurman G, Furlanello C. Minerva and minepy: a C engine for the MINE suite and its R, Python and MATLAB wrappers. *Bioinformatics*. Epub December 14, 2012.
8. Prlic A, Procter JB. Ten simple rules for the open development of scientific software. *PLoS Comput Biol*. Dec 2012;8(12):e1002802.
9. Spellman PT, Sherlock G, Zhang MQ, et al. Comprehensive identification of cell cycle-regulated genes of the yeast Saccharomyces cerevisiae by microarray hybridization. *Mol Biol Cell*. Dec 1998;9(12):3273–97.
10. Kalari KR, Rossell D, Necela BM, et al. Deep Sequence Analysis of Non-Small Cell Lung Cancer: Integrated Analysis of Gene Expression, Alternative Splicing, and Single Nucleotide Variations in Lung Adenocarcinomas with and without Oncogenic KRAS Mutations. *Frontiers in Oncology*. 2012;2:12.