Bactome III: OLIgonucleotide Variable Expression Ranker (OLIVER) 1.0, Tool for Identifying Suitable Reference (Invariant) Genes from Large Microarray Datasets

Oliver YW Chan
Raffles Institution, Singapore

Bryan MH Keng
Raffles Institution, Singapore

Maurice HT Ling
School of Chemical and Biomedical Engineering
Nanyang Technological University, Singapore
Department of Zoology
The University of Melbourne, Australia
mauriceling@acm.org

Abstract

Reference genes are crucial in gene expression analysis where varying gene expressions are reported as changes with respect to the expression of reference genes. Hence, reference genes are assumed to be stably expressed under most circumstances. Previous studies have shown that common algorithms for identifying of potential reference genes are not suitable for large microarray-sized datasets. Our previous work had derived methods for identification of reference genes from large microarray-sized datasets and the performances of these methods are linear to sample size. These methods had been implemented as a tool, OLIgonucleotide Variable Expression Ranker (OLIVER), which can be downloaded from http://sourceforge.net/projects/bactome/files/OLIVER/OLIVER_1.zip. This manuscript documents the implementation of OLIVER, which had been incorporated as part of Bactome project (http://www.sf.net/projects/bactome). These codes are licensed under GNU General Public License version 3 for academic and non-for-profit use.

1. Description

Gene expression analysis is examining the variations in gene expression by measuring DNA expression levels over time. An important aspect of such studies is the availability of reference genes, which are assumed to be stably expressed under most circumstances, as varying gene expressions are reported as changes with respect to the expression of reference genes. Microarrays, which usually contain thousands of probes, present a good source of data for identifying reference genes (Chia et al., 2010; Heng et al., 2011; Keng et al., 2013; Too and Ling, 2012; Too et al., 2014). However, existing methods of identifying potential reference genes, including geNorm (Vandesompele et al., 2002), NormFinder (Andersen et al., 2004), and BestKeeper (Pfaffl et al., 2004), are only capable of analyzing a small sample. Hence, there is a need for reference gene identification methods that are capable of processing large microarray datasets.

Our recent study (Chan et al., 2014) derives two statistical methods for identifying reference genes from large microarray datasets which are highly correlated (r = 0.892) to the results from NormFinder (Andersen et al., 2004) when evaluated on a smaller set of data. This correlation (r = 0.892) is higher than the correlations of output from NormFinder and BestKeeper (r = 0.700), suggesting that our methods are suitable for reference gene identification. We evaluated computational time needed at computationally intensive portion of the calculation. Our results (Chan et al., 2014) show that 260 samples of 100 thousand genes took 25 minutes to complete, which is linearly proportional to the number of data elements ($r^2 = 0.999$); thereby, suggesting an *O(n)* behaviour.

Both of our novel methods (Chan et al., 2014) are based on prior work (Keng et al., 2013) demonstrating that the average coefficient of detemination between stably expressed genes (potential reference genes) and genes of variable expression is significantly lower than the average coefficient of detemination between 2 sets of genes with variable expression. This concept is based on the probability of pairs of genes with variable expressions that are positively or negatively correlated is higher than a gene with variable expression and a gene that is stably expressed. We term this method (Chan et al., 2014) as "selfed correlation"and derive "selfed ratio correlation" (dataset X and the quotient of X and sample of non-X data), and "selfed product correlation" (dataset X and the product of X and sample of non-X data). These forms the fundamental methods for more complex methods; of which, two methods (our novel methods in Chan et al., 2014) give high correlation (r = 0.892) with the results from NormFinder (Andersen et al., 2004) – (1) geometric mean of exponent of selfed ratio correlation (e^ratio) and coefficient of variation, and (2) (e^ratio / average of e^ratio) + (cv / average of cv).

We implemented our methods as a command line tool, OLIgonucleotide Variable Expression Ranker (OLIVER; Chan et al., 2014). This manuscript documents the implementation of OLIVER, which had been incorporated as part of Bactome project (http://www.sf.net/projects/bactome). OLIVER is dependent on Collection of Python Algorithms and Data Structures (COPADS; http://github.com/copads/copads). OLIVER is licensed under GNU General Public License version 3 for academic and non-for-profit use while COPADS is licensed under Python Software Foundation License version 2.

Current reference gene identification methods in OLIVER are:

1. Coefficient of variation (Chia et al., 2010)
2. Gradient (Lee et al., 2007)
3. Regression ratio: quotient of coefficient of determination and gradient (Lee et al., 2007)
4. Average-Standard deviation: product of arithmetic mean and standard deviation (Lee et al., 2007)
5. Selfed correlation: average absolute pairwise correlation between dataset X and sample non-X data (Chan et al., 2014)
6. Selfed ratio correlation: average absolute pairwise correlation between dataset X and the quotient of X and sample of non-X data (Chan et al., 2014)
7. Selfed product correlation: average absolute pairwise correlation between dataset X and the product of X and sample of non-X data (Chan et al., 2014)
8. Geometric mean of exponent of selfed ratio correlation (e^ratio) and coefficient of variation (Chan et al., 2014)
9. (e^ratio / average of e^ratio) + (cv / average of cv) (Chan et al., 2014)

2. Code Files

File name: setup-oliver.py

```
from distutils.core import setup
import py2exe

# Make Windows executable: python setup.py py2exe

setup(name='oliver',
      version='1.0',
      description='OLIgonucleotide Variable Expression Ranker (OLIVER)',
      long_description='''
OLIgonucleotide Variable Expression Ranker (OLIVER): A set of tools for
identifying suitable reference (invariant) genes from large transcriptome
datasets.

Part of Bactome project (http://www.sf.net/projects/bactome)

References:
- Chan, OYW, Keng, BMH, Ling, MHT. 2014. Correlation and Variation Based
Method for Reference Genes Identification from Large Datasets. Electronic
Physician 6(1): 719-727.

Authors:
- Oliver YW Chan, Raffles Institution, Singapore
- Bryan MH Keng, Raffles Institution, Singapore
- Maurice HT Ling (mauriceling@acm.org), Department of Zoology, The
University of Melbourne, Australia

Copyright: Copyright (c) 2013-2014, Maurice Ling (on behalf of all
authors)
''',
      author='Maurice HT Ling',
      author_email='mauriceling@acm.org',
      license = 'General Public License version 3',
      console=['oliver.py'])
```

File name: oliver.py

```
"""
Methods for identifying invariant genes (suitable reference/
normalization genes) from expression data

Date created: 16th April 2013

@see: Chan, OYW, Keng, BMH, Ling, MHT. 2014. Correlation and Variation
Based Method for Reference Genes Identification from Large Datasets.
Electronic Physician 6(1): 719-727.

License: GNU General Public License version 3 for academic or
not-for-profit use only
"""

import sys
import time
import math
import sets

from invariant_gene import selfed_correlation
from invariant_gene import selfed_ratio_correlation
from invariant_gene import selfed_product_correlation
from invariant_gene import regression_ratio
from invariant_gene import average_stdev
from invariant_gene import cv
from invariant_gene import gradient

def datafile(filename):
    '''
    Reads input data file (containing probe and expression data) into a
    dictionary.

    Input data file is a comma-delimited file in the format of
    ProbeName1,Sample1Value,Sample2value,Sample3Value, ...
    ProbeName2,Sample1Value,Sample2value,Sample3Value, ...
    ProbeName3,Sample1Value,Sample2value,Sample3Value, ...
    @param filename: input data file name
    @return: dictionary where key is ProbeName and value is a list of
    SampleValues
    '''
    print 'Reading data from', filename
    t = time.time()
    f = open(filename, 'r').readlines()
    if len(f) == 1:
        f = f[0].split('\r')
    f = [x[:-1] for x in f]
    f = [x.split(',') for x in f]
    data = {}
    for x in f:
        try:
            x = [y for y in x if y != '']
            if len(x) > 2:
                name = str(x[0])
                values = [float(y) for y in x[1:]]
                data[name] = values
        except: 'Error formatting:', x
    print '     Time taken:', time.time()-t, 'seconds'
    return data

def p_gradient(data, options, methods, results):
    '''
    Processor for reference gene identification method: gradient

    @param data: dictionary of input data from datafile function
    @param options: options for current method (please see user manual)
    @param methods: list of methods used (current method will be appended
    for final result file)
    @param results: results dictionary in the format of {<ProbeName>:
    {<method>: <results from method>}}
    @return: tuple of (methods, results)
    '''
    print 'Calculating gradients ...'
    t = time.time()
    t_results = gradient(data)
    methods.append('gradient')
    for genename in t_results:
        if results.has_key(genename):
            results[genename]['gradient'] = t_results[genename]
        else:
            results[genename] = {'gradient': t_results[genename]}
    print '     Time taken:', time.time()-t, 'seconds'
    return (methods, results)

def p_cv(data, options, methods, results):
    '''
    Processor for reference gene identification method: coefficient of
    variation

    @param data: dictionary of input data from datafile function
    @param options: options for current method (please see user manual)
    @param methods: list of methods used (current method will be appended
    for final result file)
    @param results: results dictionary in the format of {<ProbeName>:
    {<method>: <results from method>}}
    @return: tuple of (methods, results)
    '''
    print 'Calculating coefficient of variations ...'
    t = time.time()
    t_results = cv(data)
    methods.append('cv')
    for genename in t_results:
        if results.has_key(genename):
            results[genename]['cv'] = t_results[genename]
        else:
            results[genename] = {'cv': t_results[genename]}
    print '     Time taken:', time.time()-t, 'seconds'
    return (methods, results)

def p_regression_ratio(data, options, methods, results):
    '''
    Processor for reference gene identification method: ratio of
    coefficient of determination (R^2) and gradient

    @param data: dictionary of input data from datafile function
    @param options: options for current method (please see user manual)
    @param methods: list of methods used (current method will be appended
    for final result file)
    @param results: results dictionary in the format of {<ProbeName>:
    {<method>: <results from method>}}
    @return: tuple of (methods, results)
    '''
    print 'Calculating regression ratios (R^2/slope) ...'
    t = time.time()
    t_results = regression_ratio(data)
    methods.append('R^2/slope')
    for genename in t_results:
        if results.has_key(genename):
            results[genename]['R^2/slope'] = t_results[genename]
        else:
            results[genename] = {'R^2/slope': t_results[genename]}
    print '     Time taken:', time.time()-t, 'seconds'
    return (methods, results)

def p_average_stdev(data, options, methods, results):
    '''
    Processor for reference gene identification method: product of average
    and standard deviation

    @param data: dictionary of input data from datafile function
    @param options: options for current method (please see user manual)
    @param methods: list of methods used (current method will be appended
    for final result file)
    @param results: results dictionary in the format of {<ProbeName>:
    {<method>: <results from method>}}
    @return: tuple of (methods, results)
    '''
    print 'Calculating average x stdev ...'
    t = time.time()
    t_results = average_stdev(data)
    methods.append('avg*SD')
    for genename in t_results:
        if results.has_key(genename):
            results[genename]['avg*SD'] = t_results[genename]
        else:
            results[genename] = {'avg*SD': t_results[genename]}
    print '     Time taken:', time.time()-t, 'seconds'
    return (methods, results)

def p_selfed_product_correlation(data, options, methods, results):
    '''
    Processor for reference gene identification method: average
    absolute pairwise correlation between dataset X and the
    product of X and sample of non-X data (n = randomsize)
    where large value represents higher stability

    @param data: dictionary of input data from datafile function
    @param options: options for current method (please see user manual)
    @param methods: list of methods used (current method will be appended
    for final result file)
    @param results: results dictionary in the format of {<ProbeName>:
    {<method>: <results from method>}}
    @return: tuple of (methods, results)
    '''
    print 'Calculating selfed product correlation ...'
    t = time.time()
    if not options.has_key('rss'): options['rss'] = 30
    t_results = selfed_product_correlation(data, options['rss'])
    methods.append('prod_corr')
    for genename in t_results:
        if results.has_key(genename):
            results[genename]['prod_corr'] = t_results[genename]
        else:
            results[genename] = {'prod_corr': t_results[genename]}
    print '     Time taken:', time.time()-t, 'seconds'
    return (methods, results)

def p_selfed_ratio_correlation(data, options, methods, results):
    '''
    Processor for reference gene identification method: average
    absolute pairwise correlation between dataset X and the
    quotient of X and sample of non-X data (n = randomsize)
    where small value represents higher stability

    @param data: dictionary of input data from datafile function
    @param options: options for current method (please see user manual)
    @param methods: list of methods used (current method will be appended
    for final result file)
    @param results: results dictionary in the format of {<ProbeName>:
    {<method>: <results from method>}}
    @return: tuple of (methods, results)
    '''
    print 'Calculating selfed ratio correlation ...'
    t = time.time()
    if not options.has_key('rss'): options['rss'] = 30
    t_results = selfed_ratio_correlation(data, options['rss'])
    methods.append('ratio_corr')
    for genename in t_results:
        if results.has_key(genename):
            results[genename]['ratio_corr'] = t_results[genename]
        else:
            results[genename] = {'ratio_corr': t_results[genename]}
    print '     Time taken:', time.time()-t, 'seconds'
    return (methods, results)

def p_selfed_correlation(data, options, methods, results):
    '''
    Processor for reference gene identification method: average
    absolute pairwise correlation between dataset X and sample
    non-X data (n = randomsize) where small value represents
    higher stability

    @param data: dictionary of input data from datafile function
    @param options: options for current method (please see user manual)
    @param methods: list of methods used (current method will be appended
    for final result file)
    @param results: results dictionary in the format of {<ProbeName>:
    {<method>: <results from method>}}
    @return: tuple of (methods, results)
    '''
    print 'Calculating selfed correlation ...'
    t = time.time()
    if not options.has_key('rss'): options['rss'] = 30
    t_results = selfed_correlation(data, options['rss'])
    methods.append('self_corr')
    for genename in t_results:
        if results.has_key(genename):
            results[genename]['self_corr'] = t_results[genename]
        else:
            results[genename] = {'self_corr': t_results[genename]}
    print '     Time taken:', time.time()-t, 'seconds'
    return (methods, results)

def p_geomean_expratio_cv(data, options, methods, results):
    '''
    Processor for reference gene identification method: geometric mean
    of exponent of ratio correlation (e^ratio) and coefficient of variation

    @param data: dictionary of input data from datafile function
    @param options: options for current method (please see user manual)
    @param methods: list of methods used (current method will be appended
    for final result file)
    @param results: results dictionary in the format of {<ProbeName>:
    {<method>: <results from method>}}
    @return: tuple of (methods, results)
    '''
    print 'Calculating geometric mean of ratio correlation and CV ...'
    (methods, results) = p_selfed_ratio_correlation(data, options,
                                                    methods, results)
    (methods, results) = p_cv(data, options, methods, results)
    methods.append('geomean_expratio_cv')
    for genename in results.keys():
        eratio = math.e ** results[genename]['ratio_corr']
        cv = results[genename]['cv']
        geomean = math.sqrt(eratio * cv)
        if results.has_key(genename):
            results[genename]['geomean_expratio_cv'] = geomean
        else:
            results[genename] = {'geomean_expratio_cv': geomean}
    return (methods, results)

def p_avgexpratio_avgcv(data, options, methods, results):
    '''
    Processor for reference gene identification method:
    (e^ratio_correlation / average of e^ratio_correlation) +
    (cv / average of cv)

    @param data: dictionary of input data from datafile function
    @param options: options for current method (please see user manual)
    @param methods: list of methods used (current method will be appended
    for final result file)
    @param results: results dictionary in the format of {<ProbeName>:
    {<method>: <results from method>}}
    @return: tuple of (methods, results)
    '''
    print 'Calculating sum of averaged ratio correlation and averaged CV ...'
    (methods, results) = p_selfed_ratio_correlation(data, options,
                                                    methods, results)
    (methods, results) = p_cv(data, options, methods, results)
    methods.append('avgexpratio_avgcv')
    avgexpratio = [math.e ** results[genename]['ratio_corr']
                   for genename in results.keys()]
    avgexpratio = sum(avgexpratio) / float(len(avgexpratio))
    avgcv = [results[genename]['cv']
             for genename in results.keys()]
    avgcv = sum(avgcv) / float(len(avgcv))
    for genename in results.keys():
        eratio = math.e ** results[genename]['ratio_corr']
        cv = results[genename]['cv']
        r = (float(eratio) / avgexpratio) + (float(cv) / avgcv)
        if results.has_key(genename):
            results[genename]['avgexpratio_avgcv'] = r
        else:
            results[genename] = {'avgexpratio_avgcv': r}
    return (methods, results)

def processor(data, options):
    '''
    Main processor loop to execute each required reference gene
    identification method processors

    @param data: dictionary of input data from datafile function
    @param options: options for current method (please see user manual)
    @return: tuple of (methods, results) where methods is list of methods
    used, and results is a dictionary in the format of {<ProbeName>:
    {<method>: <results from method>}}
    '''
    results = {}
    methods = []
    if options.has_key('all_methods') or options.has_key('gradient'):
        # Method: gradient
        (methods, results) = p_gradient(data, options, methods, results)
    if options.has_key('all_methods') or options.has_key('cv'):
        # Method: CV
        (methods, results) = p_cv(data, options, methods, results)
    if options.has_key('all_methods') or options.has_key('regression'):
        # Method: regression ratios (R^2/slope)
        (methods, results) = p_regression_ratio(data, options,
                                                methods, results)
    if options.has_key('all_methods') or options.has_key('avgstd'):
        # Method: average x stdev
        (methods, results) = p_average_stdev(data, options,
                                             methods, results)
    if options.has_key('all_methods') or options.has_key('pcorr'):
        # Method: selfed product correlation
        (methods, results) = p_selfed_product_correlation(data, options,
                                                          methods, results)
    if options.has_key('all_methods') or options.has_key('rcorr'):
        # Method: selfed ratio correlation
        (methods, results) = p_selfed_ratio_correlation(data, options,
                                                         methods, results)
    if options.has_key('all_methods') or options.has_key('scorr'):
        # Method: selfed correlation
        (methods, results) = p_selfed_correlation(data, options,
                                                  methods, results)
    # Method: geomean(e^ratio, CV)
    (methods, results) = p_geomean_expratio_cv(data, options,
                                               methods, results)
    # Method: (e^ratio/avg(e^ratio)) + (cv/avg(cv))
    (methods, results) = p_avgexpratio_avgcv(data, options,
                                             methods, results)
    return (methods, results)

def results_writer(filename, methods, results):
    '''
    Function to write out analysis results.

    @param filename: name of output (results) file
    @param methods: list of methods used
    @param results: results dictionary in the format of {<ProbeName>:
    {<method>: <results from method>}}
    @return: none
    '''
    out = open(filename, 'w')
    out.write('ResultFile,' + ','.join(methods) + '\n')
    for genename in results:
        values = ','.join([str(results[genename][m])
                           for m in methods])
        out.write(','.join([genename, values]) + '\n')
    out.close()

def option_processor(argv):
    '''
    Processor for options from command line into a dictionary (for example,
    -rss:30 --> {'rss': 30})
    '''
    options = {}
    options['application'] = argv[0]
    argv = [x[1:] for x in argv[1:] if x.startswith('-')]
    for arg in argv:
        arg = arg.split(':')
        if len(arg) == 2:
            options[arg[0]] = arg[1]
        else:
            options[arg[0]] = ''
    return options

def print_usage():
    '''
    Prints usage statement
    '''
    print '''
    OLIgonucleotide Variable Expression Ranker (OLIVER) 1.0:
    A tool for identifying suitable reference (invariant) genes from
    large transcriptomme datasets.

    Date created: 16th April 2013
    License: GNU General Public License version 3 for academic or not-
    for-profit use only

    Usage: python oliver.py <expression filename> <results filename> [options]
    where

    <expression filename> is a comma-delimited file in the
    format of
    ProbeName1,Sample1Value,Sample2value,Sample3Value, ...
    ProbeName2,Sample1Value,Sample2value,Sample3Value, ...
    ProbeName3,Sample1Value,Sample2value,Sample3Value, ...
    ...

    [options] in the format of -<option key>:<option value>
    For example, -outfmt:5 (option with value) and -fmt
    (option without value)

    Allowed options (please refer to user manual for detailed description):
```

```
    -all_methods    Calculate for all available methods of reference genes
                    identification
    -avgstd         Calculates average x standard deviation
    -cv             Calculates coefficient of variation
    -gradient       Calculates gradient
    -help           Prints / displays usage message
    -regression     Calculates regression ratio (R^2/slope)
    -pcorr          Calculates selfed product correlation
    -rcorr          Calculates selfed ratio correlation
    -rss            Define the number of random samples for pair-wise
                    correlations. If not given, the default is 30.
    -scorr          Calculates selfed correlation
    '''

if __name__=='__main__':
    if len(sys.argv) < 3:
        print print_usage()
    else:
        print '''
        OLIgonucleotide Variable Expression Ranker (OLIVER) 1.0:
        A tool for identifying suitable reference (invariant) genes from
        large transcriptomme datasets.

        Date created: 16th April 2013
        License: GNU General Public License version 3 for academic or
        not-for-profit use only
        '''
        options = option_processor(sys.argv)
        if options.has_key('help'):
            print print_usage()
            sys.exit(0)
        data = datafile(sys.argv[1])
        (methods, results) = processor(data, options)
        methods = list(sets.Set(methods))
        results_writer(sys.argv[2], methods, results)
        print
        print 'Summary ......'
        print 'Input data file:', sys.argv[1]
        print 'Number of genes/probes in input data file:', len(data)
        print 'Output results file:', sys.argv[2]
        print 'Number of methods used:', len(methods)
        print 'Methods:', ', '.join(methods)
        print '===== end of summary ====='
```

File name: invariant_gene.py

```
"""
Methods for identifying invariant genes (suitable reference/
normalization genes) from expression data

@see: Chan, OYW, Keng, BMH, Ling, MHT. 2014. Correlation and Variation
Based Method for Reference Genes Identification from Large Datasets.
Electronic Physician 6(1): 719-727.

Date created: 29th March 2012

License: General Public License version 3
"""

import random
from copads.samplestatistics import SingleSample, TwoSample

def selfed_correlation(data, randomsize):
    """
    The average absolute pairwise correlation between dataset X
    and sample non-X data (n = randomsize) where small value
    represents higher stability

    Pseudocode:
    1. tested_gene <-- list of expression for gene to be tested
    2. dataset_X <-- list of expression for any other gene
    3. q_set <-- {dataset_X(i) | 1 < i < n}
    4. correlation <-- {|r(dataset_X, q_set)|}, repeat steps 2 to 4
    for all genes that are not tested_gene
    5. average_correlation(tested_gene) <-- average of correlation
    6. repeat steps 1 to 5 to test for all genes

    @param data: input data as a dictionary (processed by datafile function
    in oliver.py) where key is ProbeName and value is a list of
    SampleValues
    @param randomsize: number of random samples to take
    @return: dictionary where key is ProbeName and value is result from
    current reference gene identification method
    """
    results = {}
    count = 0
    test_list = data.keys()
    if len(test_list) > randomsize:
        test_list = random.sample(test_list, randomsize)
    for genename in data.keys():
        gene = [float(x) for x in data[genename]]
        pcc = []
        for tester in test_list:
            tester = [float(x) for x in data[tester]]
            sample = TwoSample(gene, 'genename', tester, 'tester')
            try: pcc.append(abs(sample.pearson()))
            except: pass
        results[genename] = float(sum(pcc)) / len(pcc)
        count = count + 1
        if count % 500 == 0: print count, 'gene/probe processed'
    return results

def selfed_ratio_correlation(data, randomsize):
    """
    The average absolute pairwise correlation between dataset X
    and the quotient of X and sample of non-X data (n = randomsize)
    where small value represents higher stability

    Pseudocode:
    1. tested_gene <-- list of expression for gene to be tested
    2. dataset_X <-- list of expression for any other gene
    3. q_set <-- {dataset_X(i) / tested_gene(i) | 1 < i < n}
    4. correlation <-- {|r(dataset_X, q_set)|}, repeat steps 2 to 4
    for all genes that are not tested_gene
    5. average_correlation(tested_gene) <-- average of correlation
    6. repeat steps 1 to 5 to test for all genes

    @param data: input data as a dictionary (processed by datafile function
    in oliver.py) where key is ProbeName and value is a list of
    SampleValues
    @param randomsize: number of random samples to take
    @return: dictionary where key is ProbeName and value is result from
    current reference gene identification method
    """
    results = {}
    count = 0
    test_list = data.keys()
    if len(test_list) > randomsize:
        test_list = random.sample(test_list, randomsize)
    for genename in data.keys():
        gene = [float(x) for x in data[genename]]
        pcc = []
        for tester in test_list:
            tester = [float(x) for x in data[tester]]
            tester = [tester[i] / gene[i]
                      for i in range(len(gene))]
            sample = TwoSample(gene, 'genename', tester, 'tester')
            try: pcc.append(abs(sample.pearson()))
            except: pass
        results[genename] = float(sum(pcc)) / len(pcc)
        count = count + 1
        if count % 500 == 0: print count, 'gene/probe processed'
    return results

def selfed_product_correlation(data, randomsize):
    """
    The average absolute pairwise correlation between dataset X
    and the product of X and sample of non-X data (n = randomsize)
    where large value represents higher stability

    Pseudocode:
```

```
    1. tested_gene <-- list of expression for gene to be tested
    2. dataset_X <-- list of expression for any other gene
    3. q_set <-- {dataset_X(i) * tested_gene(i) | 1 < i < n}
    4. correlation <-- {|r(dataset_X, q_set)|}, repeat steps 2 to 4
    for all genes that are not tested_gene
    5. average_correlation(tested_gene) <-- average of correlation
    6. repeat steps 1 to 5 to test for all genes

    @param data: input data as a dictionary (processed by datafile function
    in oliver.py) where key is ProbeName and value is a list of
    SampleValues
    @param randomsize: number of random samples to take
    @return: dictionary where key is ProbeName and value is result from
    current reference gene identification method
    """
    results = {}
    count = 0
    test_list = data.keys()
    if len(test_list) > randomsize:
        test_list = random.sample(test_list, randomsize)
    for genename in data.keys():
        gene = [float(x) for x in data[genename]]
        pcc = []
        for tester in test_list:
            tester = [float(x) for x in data[tester]]
            tester = [tester[i] * gene[i]
                      for i in range(len(gene))]
            sample = TwoSample(gene, 'genename', tester, 'tester')
            try: pcc.append(sample.pearson())
            except: pass
        results[genename] = float(sum(pcc)) / len(pcc)
        count = count + 1
        if count % 500 == 0: print count, 'gene/probe processed'
    return results

def regression_ratio(data):
    """
    The quotient of coefficient of determination and gradient where
    large value represents higher stability.

    @see Lee et al. 2007. Identification of novel universal
    housekeeping genes by statistical analysis of microarray data.
    Journal of Biochemistry and Molecular Biology 40(2):226-231.

    @param data: input data as a dictionary (processed by datafile function
    in oliver.py) where key is ProbeName and value is a list of
    SampleValues
    @return: dictionary where key is ProbeName and value is result from
    current reference gene identification method
    """
    results = {}
    count = 0
    for genename in data.keys():
        gene = [float(x) for x in data[genename]]
        tester = range(len(gene))
        sample = TwoSample(gene, 'genename', tester, 'nominal')
        (gradient, intercept) = sample.linear_regression()
        if gradient == 0.0: gradient = 0.001
        pcc = sample.pearson()
        results[genename] = (pcc * pcc) / gradient
        count = count + 1
        if count % 500 == 0: print count, 'gene/probe processed'
    return results

def average_stdev(data):
    """
    The product of arithmetic mean and standard deviation where
    small value represents higher stability.

    @see Lee et al. 2007. Identification of novel universal
    housekeeping genes by statistical analysis of microarray data.
    Journal of Biochemistry and Molecular Biology 40(2):226-231.

    @param data: input data as a dictionary (processed by datafile function
    in oliver.py) where key is ProbeName and value is a list of
    SampleValues
    @return: dictionary where key is ProbeName and value is result from
    current reference gene identification method
    """
    results = {}
    count = 0
    for genename in data.keys():
        gene = [float(x) for x in data[genename]]
        gene = SingleSample(gene)
        results[genename] = (gene.variance() ** 0.5) * \
                            gene.arithmeticMean()
        count = count + 1
        if count % 500 == 0: print count, 'gene/probe processed'
    return results

def cv(data):
    """
    The coefficient of variation where small value represents
    higher stability.

    @param data: input data as a dictionary (processed by datafile function
    in oliver.py) where key is ProbeName and value is a list of
    SampleValues
    @return: dictionary where key is ProbeName and value is result from
    current reference gene identification method
    """
    results = {}
    count = 0
    for genename in data.keys():
        gene = [float(x) for x in data[genename]]
        gene = SingleSample(gene)
        results[genename] = (gene.variance() ** 0.5) / \
                            gene.arithmeticMean()
        count = count + 1
        if count % 500 == 0: print count, 'gene/probe processed'
    return results

def gradient(data):
    """
    The linear regression gradient where small value represents
    higher stability.

    @param data: input data as a dictionary (processed by datafile function
    in oliver.py) where key is ProbeName and value is a list of
    SampleValues
    @return: dictionary where key is ProbeName and value is result from
    current reference gene identification method
    """
    results = {}
    count = 0
    for genename in data.keys():
        gene = [float(x) for x in data[genename]]
        tester = range(len(gene))
        sample = TwoSample(gene, 'genename', tester, 'nominal')
        (gradient, intercept) = sample.linear_regression()
        results[genename] = abs(gradient)
        count = count + 1
        if count % 500 == 0: print count, 'gene/probe processed'
    return results
```

3. Sample Input and Result

Sample input:

```
01a03,1.603,3.404,0.085,1.760,0.271,2.104,1.229,2.724,2.566,3.754
01a06,1.974,8.079,0.544,6.103,0.484,3.952,2.612,5.051,9.688,8.040
01a08,0.923,5.429,0.208,3.913,0.390,3.352,1.301,5.117,8.427,8.526
01a19,2.164,4.948,0.263,2.984,0.410,4.152,1.763,2.138,6.868,6.752
01a22,2.182,9.645,0.541,5.651,0.464,3.692,2.056,4.351,7.640,9.072
01a23,3.810,15.807,1.153,10.521,1.146,15.520,5.312,18.293,26.404,23.921
01a24,2.877,12.220,0.664,8.323,0.666,10.427,3.377,9.968,12.845,18.233
01b05,1.022,2.194,0.276,2.949,0.163,3.591,0.914,3.195,6.528,4.078
01b11,0.616,4.032,0.370,3.566,0.387,4.690,1.474,5.570,2.522,4.507
01b12,0.308,1.980,0.263,2.709,0.329,5.018,1.493,4.515,7.587,9.748
```

```
01b13,0.917,5.206,0.484,5.058,0.502,6.428,1.842,6.568,6.105,4.349
01b23,1.118,3.274,0.251,3.436,0.192,0.239,0.989,2.303,4.373,4.567
01b24,1.284,6.205,0.360,4.330,0.314,2.127,1.205,3.988,4.279,5.903
01c01,2.267,8.480,0.639,6.684,0.730,7.873,3.191,7.424,15.500,16.488
01c04,1.584,9.192,0.435,4.745,0.654,8.716,1.743,4.618,7.416,5.245
01c06,1.993,8.276,0.512,5.663,0.369,3.733,1.632,5.298,5.912,3.539
01c09,1.390,6.099,0.403,4.069,0.386,4.462,1.787,3.888,8.053,3.042
01c12,1.681,5.342,0.396,5.101,0.304,3.566,1.508,3.667,8.463,3.034
01c16,2.138,9.750,0.634,6.918,0.641,9.738,2.665,7.968,14.481,13.318
01c20,1.179,7.496,0.633,6.737,0.763,9.393,3.007,6.436,12.518,11.090
```

Result (to 4 decimal places):

```
ResultFile,avgexpratio_avgcv,gradient,prod_corr,cv,self_corr,geomean_expratio_cv
,ratio_corr,avg*SD,R^2/slope
01b13,1.7881,0.5532,0.8749,0.6761,0.7929,0.9034,0.1881,9.4872,0.3871
01b12,2.5331,0.7579,0.9408,0.9653,0.7638,1.2797,0.5286,11.1256,0.8879
01b11,1.8583,0.7007,0.8762,0.7069,0.7089,0.9388,0.2206,5.4370,0.2938
01c06,1.7551,0.1035,0.8971,0.7034,0.7566,0.8850,0.1076,9.5912,0.0762
01c01,2.1194,0.3681,0.9404,0.8082,0.8580,1.0707,0.3495,38.7875,1.2589
01b23,2.1942,0.7673,0.9263,0.8377,0.7766,1.1085,0.3830,3.6040,0.2527
01c04,1.8854,0.2102,0.8923,0.7376,0.7801,0.9519,0.2057,14.5065,0.2454
01b05,2.1525,0.9746,0.9351,0.8005,0.8223,1.0874,0.3900,4.9675,0.4228
01a23,2.0785,0.2219,0.9250,0.7565,0.8788,1.0494,0.3755,112.3871,2.0583
01a22,1.8801,0.2900,0.9219,0.7452,0.8431,0.9487,0.1887,15.2881,0.3604
01c09,1.8230,0.3887,0.9261,0.7400,0.8035,0.9184,0.1308,8.3436,0.2618
01a24,2.0961,0.2958,0.9173,0.7386,0.8704,1.0567,0.4132,46.7973,1.1153
01c16,2.0944,0.3548,0.9235,0.7487,0.8927,1.0566,0.3996,34.8736,1.0106
01a06,1.8396,0.4199,0.9239,0.7053,0.8818,0.9293,0.2024,15.2683,0.4933
01c20,1.9781,0.4403,0.9165,0.7362,0.8745,0.9993,0.3048,25.8463,0.9140
01a03,1.8214,1.0090,0.8799,0.6244,0.8223,0.9163,0.2962,2.3744,0.1632
01c12,1.9181,0.3721,0.9313,0.7651,0.7850,0.9675,0.2016,8.3632,0.2598
01b24,1.8983,0.4375,0.9123,0.7388,0.8232,0.9586,0.2182,6.6467,0.2343
01a19,1.8744,0.6325,0.9252,0.7319,0.8441,0.9464,0.2020,7.7027,0.3890
01a08,2.4119,0.6242,0.9370,0.8290,0.8767,1.2137,0.5748,11.7115,0.6611
```

4. References

Andersen, CL, Jensen, JL, Ørntoft, TF. 2004. Normalization of real-time quantitative reverse transcription-PCR data: a model-based variance estimation approach to identify genes suited for normalization, applied to bladder and colon cancer data sets. Cancer Research 64: 5245-5250.

Chan, OYW, Keng, BMH, Ling, MHT. 2014. Correlation and Variation Based Method for Reference Genes Identification from Large Datasets. Electronic Physician 6(1): 719-727.

Chia, CY, Lim, CW, Leong, WT, Ling, MHT. 2010. High expression stability of microtubule affinity regulating kinase 3 (MARK3) makes it a reliable reference gene. IUBMB Life 62(3): 200-203.

Heng, SSJ, Chan, OYW, Keng, BMH, Ling, MHT. 2011. Glucan biosynthesis protein G, (mdoG) is a suitable reference gene in Escherichia coli K-12. ISRN Microbiology 2011: Article ID 469053.

Keng, BMH, Chan, OYW, Heng, SSJ, Ling, MHT. 2013. Transcriptome analysis of *Spermophilus lateralis* and *Spermophilus tridecemlineatus* liver does not suggest the presence of Spermophilus-liver-specific reference genes. ISRN Bioinformatics. 2013: Article ID 361321.

Lee, S, Jo, M, Lee, J, Koh, SS, Kim, S. 2007. Identification of novel universal housekeeping genes by statistical analysis of microarray data. Journal of Biochemistry and Molecular Biology 40: 226-231.

Pfaffl, MW, Tichopad, A, Prgomet, C, Neuvians, TP. 2004. Determination of stable housekeeping genes, differentially regulated target genes and sample integrity: BestKeeper–Excel-based tool using pair-wise correlations. Biotechnology Letters 26: 509-515.

Too, HK, Ling, MHT. 2012. Signal peptidase complex subunit 1 and hydroxyacyl-CoA dehydrogenase beta subunit are suitable reference genes in human lungs. ISRN Bioinformatics. 2012: Article ID 790452.

Too, IHK, Heng, SSJ, Chan, OYW, Keng, BMH, Chia, CY, Lim, CWX, Leong, WT, Chu, QH, Ang, EJG, Lin, YJ, Ling, MHT. 2014. Identification of Reference Genes by Meta-Microarray Analyses. In: James V. Rogers (editor). Microarrays: Principles, Applications and Technologies. New York: Nova Science Publishers, Inc.

Vandesompele, J, de Preter, K, Pattyn, F. 2002. Accurate normalisation of real-time quantitative RT-PCR data by geometric averaging of multiple internal control genes. Genome Biology 3: research0034.1–research0034.11.

Appendix A: OLIVER User Manual

NAME

OLIgonucleotide Variable Expression Ranker (OLIVER)

SYNOPSIS

```
oliver.exe <expression filename> <results filename> [options]
python oliver.py <expression filename> <results filename>
[options]
```

where

<expression filename> is a comma-delimited file in the format of

```
    ProbeName1,Sample1Value,Sample2value,Sample3Value, ...
    ProbeName2,Sample1Value,Sample2value,Sample3Value, ...
    ProbeName3,Sample1Value,Sample2value,Sample3Value, ...
    ...
```

[options] in the format of -<option key>:<option value>
For example, -outfmt:5 (option with value) and -fmt (option without value)

DESCRIPTION

OLIgonucleotide Variable Expression Ranker (OLIVER) - A set of tools for identifying suitable reference (invariant) genes from large transcriptome datasets. Please refer to our manuscript(s) [1] for methods development.

OLIVER will always calculate the 2 methods suggested in our manuscript [1]:

- geomean_expratio_cv: Geometric mean of the exponent of ratio correlation (e^{ratio}) and coefficient of variation (cv)
- avgexpratio_avgcv: (e^{ratio} / arithmetic mean of e^{ratio}) + (cv / arithmetic mean of cv)

OPTIONS

-all_methods	Calculate for all available methods of reference genes identification.
-avgstd	Calculates average x standard deviation [2].

-cv	Calculates coefficient of variation.
-gradient	Calculates gradient or regression slope.
-help	Prints / displays usage message
-regression	Calculates regression ratio (R^2/slope) [2].
-pcorr	Calculates selfed product correlation. The average absolute pairwise correlation between dataset X and the product of X and sample of non-X data (n = randomsize) where large value represents higher stability Pseudocode: 1. tested_gene <-- list of expression for gene to be tested 2. dataset_X <-- list of expression for any other gene 3. q_set <-- {dataset_X(i) * tested_gene(i) | 1 < i < n} 4. correlation <-- {|r(dataset_X, q_set)|}, repeat steps 2 to 4 for all genes that are not tested_gene 5. average_correlation(tested_gene) <-- average of correlation 6. repeat steps 1 to 5 to test for all genes
-rcorr	Calculates selfed ratio correlation. The average absolute pairwise correlation between dataset X and the quotient of X and sample of non-X data (n = randomsize) where small value represents higher stability Pseudocode: 1. tested_gene <-- list of expression for gene to be tested 2. dataset_X <-- list of expression for any other gene 3. q_set <-- {dataset_X(i) / tested_gene(i) | 1 < i < n} 4. correlation <-- {|r(dataset_X, q_set)|}, repeat steps 2 to 4 for all genes that are not tested_gene 5. average_correlation(tested_gene) <-- average of correlation 6. repeat steps 1 to 5 to test for all genes
-rss	Define the number of random samples for pair-wise correlations. If not given, the default is 30.
-scorr	Calculates selfed correlation. The average absolute pairwise correlation between dataset X and sample non-X data (n = randomsize) where small value represents higher stability Pseudocode: 1. tested_gene <-- list of expression for gene to be tested 2. dataset_X <-- list of expression for any other gene 3. q_set <-- {dataset_X(i) | 1 < i < n} 4. correlation <-- {|r(dataset_X, q_set)|}, repeat steps 2 to 4 for all genes that are not tested_gene 5. average_correlation(tested_gene) <-- average of correlation 6. repeat steps 1 to 5 to test for all genes

EXAMPLES

Given "testExpressionData.csv" as expression data file and "results.csv" as file name to write the results into,

Command #1 → `oliver.exe testExpressionData.csv results.csv`

This will execute the 2 methods proposed in our manuscript [1], namely, (a) Geometric mean of exponential of selfed ratio correlation and coefficient of variation; and (b) sum of the ratio of selfed ratio correlation and average selfed ratio correlation, and coefficient of variation and average coefficient of variation. The results file will also contain the values for selfed ratio correlation and coefficient of variation.

Command #2 → `oliver.exe testExpressionData.csv results.csv -rss:50`

This is identical to Command #1 but a sample of 50 randomly chosen probes will be used for selfed ratio correlation instead of a sample of 30.

Command #3 → `oliver.exe testExpressionData.csv results.csv -cv`

This calculates and outputs only coefficient of variation of each probe.

Command #4 → `oliver.exe testExpressionData.csv results.csv -cv -rss:50`

Since coefficient of variation calculation does not use pair-wise correlations, -rss option is not used and this command (Command #4) gives the same results as Command #3.

AUTHORS

Oliver YW Chan, Raffles Institution, Singapore
Bryan MH Keng, Raffles Institution, Singapore
Maurice HT Ling (mauriceling@acm.org), Department of Zoology, The University of Melbourne, Australia

COPYRIGHT

LICENCE

REFERENCES

[1] Chan, OYW, Keng, BMH, Ling, MHT. 2014. Correlation and Variation Based Method for Reference Genes Identification from Large Datasets. Electronic Physician 6(1): 719-727.
[2] Lee et al. 2007. Identification of novel universal housekeeping genes by statistical analysis of microarray data. Journal of Biochemistry and Molecular Biology 40(2):226-231.