Bactome II: Analyzing Gene List for Gene Ontology Over-Representation

Maurice HT Ling
Department of Zoology
The University of Melbourne, Australia
mauriceling@acm.org

Abstract

Microarray is an experimental tool that allows for the screening of several thousand genes in a single experiment and the analysis of which often requires mapping onto biological processes. This allows for the examination of processes that are over-represented. A number of tools have been developed but each differed in terms of organisms that can be analyzed. Gene Ontology website has a list of up-to-date annotation files for different organisms that can be used for over-representation analysis. Each file maps each gene of the organism to its ontological terms. It is a simple tool that allows users to use the up-to-date annotation files to generate the expected and observed counts for each GO identifier (GO ID) from a given gene list for further statistical analyses.

1. Introduction

Microarray is an experimental tool that allows for the screening of several thousand genes in a single experiment (Ling et al., 2008) and has been shown to be a useful tool to examine medical (Chon, 2001; Shen and Wu, 2009) and ecological conditions (Karr, 2008) for differential gene expression (Hale et al., 2011). However, it is common to yield several hundred differentially expressed genes from an experiment. One of the methods to make sense of these gene lists is to group them into pathways and biological functions (Eleftherohorinou et al., 2011; Osborne et al., 2007, Werner et al., 2008) using Gene Ontology (GO) (Beissbarth, 2006) to examine which functions are over-represented in the gene list. A number of studies have shown that over-represented functions correspond to biological significance (Abba et al., 2005; Carvalho et al., 2009).

A number of tools to analyze a gene list for functional over-representation have been published. These include agriGO (Du et al., 2010), AmiGO (Carbon et al., 2009), BiNGO (Maere et al., 2005), DAVID (Huang Da et al., 2009), g:Profiler (Reimand et al., 2007), GOFFA (Sun et al., 2006), GeneMANIA (Montojo et al., 2010), GeneMerge (Castillo-Davis and Hartl, 2003), GeneTools (Beisvag et al., 2006), GREAT (McLean et al., 2010), GOrilla (Eden et al., 2009), GraphWeb (Reimand et al., 2008), High-Throughput GoMiner (Zeeberg et al., 2005), Onto-Express (Khatri et al., 2007), and ToppGene Suite (Chen et al., 2009). However, each tool uses a different set annotations. For example, agriGO (Du et al., 2010) focused on agricultural species whereas GOrilla (Eden et al., 2009) only support 8 organisms. In addition, each tool updates their GO files at different frequencies. For example, g:Profiler (Reimand et al., 2007) updates at least once every 3 months whereas DAVID (Huang Da et al., 2009) does not publish its update frequency. Hence, it will require the user to examine each tool in order to determine whether it can work for the particular organism in question and whether the annotations are up to date.

In this article, a simple script-based tool is presented to allow users to use the latest GO annotation files (http://www.geneontology.org/GO.downloads.annotations.shtml) for the listed organisms or any GO annotation file that conforms to GO annotation format (http://www.geneontology.org/GO.format.annotation.shtml) version 1.0 or 2.0. In this aspect, this script is similar to the data "updated-ness" of AmiGO (Carbon et al., 2009) which also allows users to upload their own GO annotation file. The main function of this script is to generate the expected and observed counts of each GO identifier (GO ID) from a given gene list by mapping the list against the corresponding GO annotation file for the organism in question; this allows for downstream analysis such as Chi Square test or hypergeometric test. This script is released under Python Software Foundation License version 2 to allow for incorporation into other systems. Due to the differences in print statements between Python 2.x and Python 3.x, these codes can only be used in Python 2.x.

2. Code and Test Cases

File Name: go_overepresentation.py

```
"""
Extracting Gene List for Gene Ontology Over-representation.
This script only works with Python version < 3.0 due to print
statements.
Date created: 1st July 2011
Licence: Python Software Foundation License version 2
"""

import os
import sys

def parse_GAF(gaf_file):
    """
    Reader for GO Annotation File - http://www.geneontology.org/
    GO.format.gaf-2_0.shtml

    @param gaf_file: file path for GO annotation file
    @returns: a tuple - (gaf, columns, gaf_ver, header) -
        "gaf" is the annotation,
        "columns" is the column header in the annotation,
        "gaf_ver" is the GAF version (either 1.0 or 2.0),
        "header" contains all the comments in the annotation file.

    @raise AttributeError: when there is no GAF version or if GAF
    version is not 1.0 or 2.0
    """
    gaf = open(gaf_file, 'r').xreadlines()
    gaf = [x[:-1].strip() for x in gaf]
    header = [x for x in gaf if x.startswith('!')]
    gaf = [x for x in gaf if not x.startswith('!')]
    if '!gaf-version: 2.0' in header:
        gaf_ver = 2.0
    elif '!gaf-version: 1.0' in header:
        gaf_ver = 1.0
    else:
```

```
            raise AttributeError('File does not have GAF version')
        gaf = [x.split('\t') for x in gaf]
        gaf_16 = [x for x in gaf if len(x) == 16]
        gaf_17 = [x for x in gaf if len(x) == 17]
        gaf = [x for x in gaf if len(x) == 15]
        gaf = [(x[0].strip(), x[1].strip(), x[2].strip(),
                [y.strip() for y in x[3].split('|')], x[4].strip(),
                [y.strip() for y in x[5].split('|')], x[6].strip(),
                [y.strip() for y in x[7].split('|')], x[8].strip(),
                x[9].strip(), [y.strip() for y in x[10].split('|')],
                x[11].strip(), [y.strip() for y in x[12].split('|')],
                x[13].strip(), x[14].strip())
               for x in gaf]
        gaf_16 = [(x[0].strip(), x[1].strip(), x[2].strip(),
                   [y.strip() for y in x[3].split('|')], x[4].strip(),
                   [y.strip() for y in x[5].split('|')], x[6].strip(),
                   [y.strip() for y in x[7].split('|')], x[8].strip(),
                   x[9].strip(), [y.strip() for y in x[10].split('|')],
                   x[11].strip(), [y.strip() for y in x[12].split('|')],
                   x[13].strip(), x[14].strip(),
                   [y.strip() for y in x[15].split('|')])
                  for x in gaf_16]
        gaf_17 = [(x[0].strip(), x[1].strip(), x[2].strip(),
                   [y.strip() for y in x[3].split('|')], x[4].strip(),
                   [y.strip() for y in x[5].split('|')], x[6].strip(),
                   [y.strip() for y in x[7].split('|')], x[8].strip(),
                   x[9].strip(), [y.strip() for y in x[10].split('|')],
                   x[11].strip(), [y.strip() for y in x[12].split('|')],
                   x[13].strip(), x[14].strip(),
                   [y.strip() for y in x[15].split('|')],
                   [y.strip() for y in x[16].split('|')])
                  for x in gaf_17]
        gaf = gaf + gaf_16 + gaf_17
        if gaf_ver == 2.0:
            columns = ['DB', 'DB Object ID', 'DB Object Symbol',
                       'Qualifier', 'GO ID', 'DB:Reference', 'Evidence Code',
                       'With (or) From', 'Aspect', 'DB Object Name',
                       'DB Object Synonym', 'DB Object Type', 'Taxon',
                       'Date', 'Assigned By', 'Annotation Extension',
                       'Gene Product Form ID']
        if gaf_ver == 1.0:
            columns = ['DB', 'DB Object ID', 'DB Object Symbol',
                       'Qualifier', 'GO ID', 'DB:Reference', 'Evidence Code',
                       'With (or) From', 'Aspect', 'DB Object Name',
                       'DB Object Synonym', 'DB Object Type', 'Taxon',
                       'Date', 'Assigned By']
        return (gaf, columns, gaf_ver, header)

    def process_association_file(assoc_file):
        """
        Extracts information from GO annotation file for over-representation
        analysis

        @param assoc_file: File path for GO annotation file

        @return: A tuple - (genome, gene_set, length of gene_set,
            ontology_set, length of ontology_set) -
```

```
        "genome" is a list of (<gene symbol>, <GO ID>) representing the
        annotated genome
        "gene_set" is the set of genes in the genome
        "length of gene_set" is the number of genes in the genome
        "ontology_set" is the set of GO IDs used in the annotation
        "length of ontology_set" is the number of GO IDs used in the
        annotation
    """
    genome = parse_GAF(assoc_file)[0]
    genome = [(x[2].lower(), x[4].lower())
              for x in genome
                  if len(x) > 5]
    gene_set = set([x[0] for x in genome])
    ontology_set = set([x[1] for x in genome])
    return (genome, gene_set, len(gene_set), ontology_set,
            len(ontology_set))

def process_sample_file(samplefile, genome):
    """
    Processes the gene list file, the set of genes to be analyzed for
    GO over-representation, and generate an annotated sample genome
    (to represent the observed data in Chi Square test)

    @param samplefile: File path for the gene list for over-representation
        analysis - one gene per line
    @param genome: Genome to base expectation on. The genome is a list
        of (<gene symbol>, <GO ID>)
    """
    sample_list = open(samplefile, 'r').readlines()
    sample_list = [x[:-1].lower() for x in sample_list]
    sample_genome = [x for x in genome
                     if x[0].lower() in sample_list]
    gene_set = set([x[0] for x in sample_genome])
    sample_ontology = set([x[1] for x in sample_genome])
    return(sample_list, sample_genome, gene_set, len(gene_set),
           sample_ontology)

def find_ontology_by_gene(gene, g):
    """
    Find the set of GO IDs for a given gene

    @param gene: Gene symbol
    @param g: Genome to base expectation on. The genome is a list of
        (<gene symbol>, <GO ID>)

    @return: A set of GO IDs
    """
    return set([x[1] for x in g if x[0] == gene.lower()])

def find_gene_by_ontology(ontology, g):
    """
    Find the set of genes by a given ontology ID

    @param ontology: Gene ontology ID
    @param g: Genome to base expectation on. The genome is a list of
        (<gene symbol>, <GO ID>)
```

```
    @return: A set of gene symbols
    """
    return set([x[0] for x in g if x[1] == ontology.lower()])

def calculate_expectation(ontology, genome, genome_count):
    """
    Calculated the expectation (number of expected in Chi Square test)
    for each ontology ID and the given genome

    @param ontology: Gene ontology ID to calculate expectation
    @param genome: Genome to base expectation on. The genome is a list
        of (<gene symbol>, <GO ID>)
    @param genome_count: Number of genes in the genome

    @return: Expectation as a float number.
    """
    gene_w_ontology = len(find_gene_by_ontology(ontology, genome))
    return float(gene_w_ontology) / float(genome_count)

def _write_headers(f, assoc_file, samplefile, outputfile,
                   genome_count, ontology_count, sample_count,
                   mapped_count, genes_not_mapped, genes_mapped,
                   test_count, display=True):
    """
    Writes the information header for the analysis - not to be used
    externally
    """
    f.write(' '.join(['Gene Ontology association file:',
                      str(assoc_file), '\n']))
    f.write(' '.join(['Sample file:', str(samplefile), '\n']))
    f.write(' '.join(['Output file:', str(outputfile), '\n']))
    f.write(' '.join(['Number of genes in association file:',
                      str(genome_count), '\n']))
    f.write(' '.join(['Number of ontological terms in association file:',
                      str(ontology_count), '\n']))
    f.write(' '.join(['Number of genes in sample file:',
                      str(sample_count), '\n']))
    f.write(' '.join(['Number of genes in sample file mapped:',
                      str(mapped_count), '\n']))
    f.write(' '.join(['Genes in sample file mapped:',
                      str(genes_mapped), '\n']))
    if genes_not_mapped == '':
        genes_not_mapped = '(None - All genes are mapped)'
    f.write(' '.join(['Genes in sample file not mapped:',
                      str(genes_not_mapped), '\n']))
    f.write(' '.join(['Number of statistical tests:',
                      str(test_count), '\n']))
    if display:
        print ' '.join(['Gene Ontology association file:',
                        str(assoc_file)])
        print ' '.join(['Sample file:', str(samplefile)])
        print ' '.join(['Output file:', str(outputfile)])
        print ' '.join(['Number of genes in association file:',
                        str(genome_count)])
        print ' '.join(['Number of ontological terms in association file:',
                        str(ontology_count)])
        print ' '.join(['Number of genes in sample file:',
```

```
                    str(sample_count)])
        print ' '.join(['Number of genes in sample file mapped:',
                    str(mapped_count)])
        print ' '.join(['Genes in sample file mapped:',
                    str(genes_mapped)])
        print ' '.join(['Genes in sample file not mapped:',
                    str(genes_not_mapped)])
        print ' '.join(['Number of statistical tests:',
                    str(test_count)])

def main(assoc_file, samplefile, outputfile, display=True):
    """
    Runner for the GO over-representation analysis

    @param assoc_file: File path for GO annotation file
    @param samplefile: File path for the gene list for over-representation
        analysis - one gene per line
    @param outputfile: File path to write the analysis output
    @param display: Flag to display the results on console (default = True)
    """
    (genome, gene_set, genome_count, ontology_set, ontology_count) = \
             process_association_file(assoc_file)
    (sample_list, sample_genome, gene_set, sample_count, sample_ontology) = \
            process_sample_file(samplefile, genome)
    output_f = open(outputfile, 'w')
    genes_not_mapped = ', '.join([x for x in sample_list
                                  if x not in gene_set])
    genes_mapped = ', '.join([x for x in sample_list
                                  if x in gene_set])
    _write_headers(output_f, assoc_file, samplefile, outputfile,
                   genome_count, ontology_count, len(sample_list),
                   len(gene_set), genes_not_mapped, genes_mapped,
                   len(sample_ontology))
    output_f.write(','.join(['Term', 'Expected Count', '(Not) Expected',
                             'Actual Count', '(Not) Actual', '\n']))
    if display:
        print '\t'.join(['Term', 'Expected Count', '(Not) Expected',
                         'Actual Count', '(Not) Actual'])
    for ontology in sample_ontology:
        expectation = calculate_expectation(ontology, genome, genome_count)
        expected_count = expectation * sample_count
        expected_uncount = sample_count - expected_count
        actual_count = len(find_gene_by_ontology(ontology, sample_genome))
        actual_uncount = sample_count - actual_count
        output_f.write(','.join([ontology,
                                 str(expected_count),
                                 str(expected_uncount),
                                 str(actual_count),
                                 str(actual_uncount), '\n']))
        if display:
            print '\t'.join([ontology,
                             str(expected_count),
                             str(expected_uncount),
                             str(actual_count),
                             str(actual_uncount)])
    output_f.close()
```

```
def test():
    """
    Example execution using Escherichia coli GAF and Pseudomonas
    aeruginosa GAF
    """
    main('gene_association.ecocyc', 'e_coli_gene.txt', 'e_coli_output.csv')
    main('gene_association.pseudocap', 'p_aeruginosa.txt',
         'p_aeruginosa_output.csv')

if __name__ == '__main__':
    if len(sys.argv) != 4 and sys.argv[1] != 'test':
        print """Usage: python go_overexpression.py <association file>
        <gene list file> <output file> \n
        where \n
        <association file> is the Gene Ontology annotation file for the
        organism downloaded from http://www.geneontology.org/
        GO.downloads.annotations.shtml \n
        <gene list file> is the text file of gene list - one gene per line \n
        <output file> is the name of CSV file (comma-separated file) where
        output is stored \n
        For example: python go_overexpression.py gene_association.ecocyc
        e_coli_gene.txt e_coli_output.csv"""
    elif sys.argv[1] == 'test':
        test()
    else:
        main(str(sys.argv[1]), str(sys.argv[2]), str(sys.argv[3]))
```

File Name: tester.py

```
import unittest
import go_overexpression as g

assocfile_eco = 'gene_association.ecocyc'
assocfile_pse = 'gene_association.pseudocap'
gene_eco = 'e_coli_gene.txt'
gene_pse = 'p_aeruginosa.txt'

class testGO(unittest.TestCase):
    def testVersion(self):
        self.assertTrue(g.parse_GAF(assocfile_eco)[2] == 2.0)
        self.assertTrue(g.parse_GAF(assocfile_pse)[2] == 2.0)

    def testAssoc(self):
        assoc1 = ('EcoCyc', '3-OXOACYL-ACP-SYNTHII-MONOMER', 'FabF',
                  [''], 'GO:0006633', ['GOA:interpro', 'GO_REF:0000002'],
                  'IEA', ['InterPro:IPR017568'], 'P', 'FabF', ['fabF',
                  'vtr', 'cvc', 'fabJ', 'vtrB', 'b1095', 'ECK1081'],
                  'protein', ['taxon:511145'], '20110110', 'InterPro')
        assoc2 = ('EcoCyc', 'AMINEOXID-MONOMER', 'TynA', [''], 'GO:0008131',
                  ['GOA:interpro', 'GO_REF:0000002'], 'IEA',
                  ['InterPro:IPR000269'], 'F', 'TynA', ['tynA', 'feaA',
                  'maoA', 'b1386', 'ECK1383'], 'protein', ['taxon:511145'],
                  '20110110', 'InterPro')
        assoc3 = ('PseudoCAP', 'PA4967', 'parE', [''], 'GO:0006259',
                  ['PMID:99169751'], 'IDA', [''], 'P',
                  'topoisomerase IV subunit B', [''], 'protein',
```

```
                ['taxon:208964'], '20060206', 'PseudoCAP')
        assoc4 = ('PseudoCAP', 'PA4932', 'rplI', [''], 'GO:0044267',
                  ['PMID:96374488'], 'ISS', [''], 'P',
                  '50S ribosomal protein L9', [''], 'protein',
                  ['taxon:208964'], '20060206', 'PseudoCAP')
        self.assertTrue(assoc1 in g.parse_GAF(assocfile_eco)[0])
        self.assertTrue(assoc2 in g.parse_GAF(assocfile_eco)[0])
        self.assertTrue(assoc3 in g.parse_GAF(assocfile_pse)[0])
        self.assertTrue(assoc4 in g.parse_GAF(assocfile_pse)[0])

    def testOntologyByGene(self):
        eco_genome = g.process_association_file(assocfile_eco)[0]
        pse_genome = g.process_association_file(assocfile_pse)[0]
        onto1 = ['go:0006633', 'go:0005515', 'go:0005829', 'go:0008415',
                 'go:0016740', 'go:0003824', 'go:0009058', 'go:0016747',
                 'go:0008152', 'go:0008610']
        onto2 = ['go:0005737', 'go:0008652', 'go:0050661', 'go:0006561',
                 'go:0004735', 'go:0008152', 'go:0055114']
        onto3 = ['go:0009276', 'go:0009103']
        onto4 = ['go:0006091', 'go:0006810']
        self.assertEqual(list(g.find_ontology_by_gene('fabf', eco_genome)),
                         onto1)
        self.assertEqual(list(g.find_ontology_by_gene('proc', eco_genome)),
                         onto2)
        self.assertEqual(list(g.find_ontology_by_gene('rmd', pse_genome)),
                         onto3)
        self.assertEqual(list(g.find_ontology_by_gene('nosf', pse_genome)),
                         onto4)

    def testGeneByOntology(self):
        eco_genome = g.process_association_file(assocfile_eco)[0]
        pse_genome = g.process_association_file(assocfile_pse)[0]
        gene1 = ['acpp', 'ychm', 'fabr', 'accd', 'fabb', 'acca', 'fabd',
                 'accc', 'accb', 'acps', 'fabg', 'fabf', 'faba', 'fabz',
                 'plsx', 'lipoyl-acp', 'fabi', 'acph', 'fabh']
        gene2 = ['waac', 'wzm', 'glmu', 'lpxk', 'wzz', 'waaf', 'wbpm',
                 'wzy', 'wzt', 'wbpg', 'rmla', 'rmlb', 'wbpy', 'wbpx',
                 'rmd', 'glmm', 'wbpw']
        self.assertEqual(list(g.find_gene_by_ontology('go:0006633',
                                                      eco_genome)), gene1)
        self.assertEqual(list(g.find_gene_by_ontology('go:0009103',
                                                      pse_genome)), gene2)

    def testExpectation(self):
        eco_assoc = g.process_association_file('gene_association.ecocyc')
        pse_assoc = g.process_association_file('gene_association.pseudocap')
        self.assertAlmostEqual(g.calculate_expectation('go:0006633',
                                                       eco_assoc[0],
                                                       eco_assoc[2]),
                               0.00492994)
        self.assertAlmostEqual(g.calculate_expectation('go:0005515',
                                                       eco_assoc[0],
                                                       eco_assoc[2]),
                               0.23040996)
        self.assertAlmostEqual(g.calculate_expectation('go:0009276',
                                                       pse_assoc[0],
                                                       pse_assoc[2]),
```

```
                                0.07378129)
    def testReadSample(self):
        eco_assoc = g.process_association_file('gene_association.ecocyc')
        pse_assoc = g.process_association_file('gene_association.pseudocap')
        eco_list = ['mdog', 'dapa', 'crp', 'hslv', 'mrdb', 'fucu', 'yjgp',
                    'yigc', 'sun', 'gor', 'hflb', 'yqib', 'murg', 'yrbg',
                    'yejk', 'yfga', 'hflx', 'spot', 'holc', 'xerd', 'tolb',
                    'yhes', 'ntpa', 'yabb', 'lola', 'yggd', 'pnp', 'yrbb',
                    'rnc', 'xerc', 'rfaf', 'yigp', 'gyrb', 'nagc', 'nrdr',
                    'hemd', 'phet', 'frr', 'cls']
        pse_list = ['ppk', 'kdsa', 'dnaq', 'pcm', 'rpsk', 'thrh', 'thrb',
                    'zipa', 'rpsl', 'pcrv', 'rsal', 'msud', 'msue', 'flio',
                    'flip', 'fliq', 'flir', 'flhb', 'flha', 'aruc', 'exab',
                    'arnt', 'parc', 'pare', 'pa2018', 'pa2019', 'ftse',
                    'ftsx', 'leus', 'coae', 'yrfi', 'grx', 'pa4827', 'frr',
                    'wzx', 'phaf', 'bdha', 'metf', 'biob', 'ftsa', 'ftsq',
                    'upps', 'glcf', 'glce', 'glcd', 'glcc', 'typa', 'rplo',
                    'prpl', 'rplc', 'gyrb', 'pepa', 'kdpa', 'kdpb', 'kdpc',
                    'rnt', 'ppka', 'moea2', 'lola', 'ribe', 'pchf', 'pche',
                    'phhc', 'dnab', 'tata', 'tatb', 'tatc', 'exaa', 'biof',
                    'trmu', 'rlud', 'arsr', 'arsb', 'arsc', 'prlc', 'waap',
                    'waag', 'hflc', 'hflk', 'pa4133', 'aotj', 'aotq', 'aotm',
                    'aotp', 'argr', 'phne', 'kdpf', 'sure', 'nrda', 'hemn',
                    'wbpw', 'rmd', 'sbcd']
        self.assertEqual(g.process_sample_file(gene_eco, eco_assoc[0])[0],
                         eco_list)
        self.assertEqual(g.process_sample_file(gene_pse, pse_assoc[0])[0],
                         pse_list)

        def testSampleAnnotation(self):
            eco_assoc = g.process_association_file('gene_association.ecocyc')
            pse_assoc = \
                  g.process_association_file('gene_association.pseudocap')
            self.assertTrue(['go:0016051', 'go:0009376', 'go:0042150'] in \
                            g.process_sample_file(gene_eco, eco_assoc[0])[4])
            self.assertTrue(['go:0016787', 'go:0006950', 'go:0009636'] in \
                            g.process_sample_file(gene_pse, pse_assoc[0])[4])

if __name__ == '__main__':
    unittest.main()
```

3. References

ABBA, M. C., HU, Y., SUN, H., DRAKE, J. A., GADDIS, S., BAGGERLY, K., SAHIN, A. & ALDAZ, C. M. 2005. Gene expression signature of estrogen receptor alpha status in breast cancer. *BMC Genomics,* 6**,** 37.

BEISSBARTH, T. 2006. Interpreting experimental results using gene ontologies. *Methods in Enzymology,* 411, 340-352.

BEISVAG, V., JUNGE, F. K., BERGUM, H., JOLSUM, L., LYDERSEN, S., GUNTHER, C. C., RAMAMPIARO, H., LANGAAS, M., SANDVIK, A. K. & LAEGREID, A. 2006. GeneTools--application for functional annotation and statistical hypothesis testing. *BMC Bioinformatics,* 7**,** 470.

CARBON, S., IRELAND, A., MUNGALL, C. J., SHU, S., MARSHALL, B. & LEWIS, S. 2009. AmiGO: online access to ontology and annotation data. *Bioinformatics,* 25**,** 288-9.

CARVALHO, P. C., FISCHER, J. S., CHEN, E. I., DOMONT, G. B., CARVALHO, M. G., DEGRAVE, W. M., YATES, J. R., 3RD & BARBOSA, V. C. 2009. GO Explorer: A gene-ontology tool to aid in the interpretation of shotgun proteomics data. *Proteome Science,* 7**,** 6.

CASTILLO-DAVIS, C. I. & HARTL, D. L. 2003. GeneMerge--post-genomic analysis, data mining, and hypothesis testing. *Bioinformatics,* 19**,** 891-2.

CHEN, J., BARDES, E. E., ARONOW, B. J. & JEGGA, A. G. 2009. ToppGene Suite for gene list enrichment analysis and candidate gene prioritization. *Nucleic Acids Research,* 37**,** W305-11.

CHON, H. S. & LANCASTER, J. M. 2011. Microarray-based gene expression studies in ovarian cancer. *Cancer Control,* 18**,** 8-15.

DU, Z., ZHOU, X., LING, Y., ZHANG, Z. & SU, Z. 2010. agriGO: a GO analysis toolkit for the agricultural community. *Nucleic Acids Research,* 38**,** W64-70.

EDEN, E., NAVON, R., STEINFELD, I., LIPSON, D. & YAKHINI, Z. 2009. GOrilla: a tool for discovery and visualization of enriched GO terms in ranked gene lists. *BMC Bioinformatics,* 10**,** 48.

ELEFTHEROHORINOU, H., HOGGART, C. J., WRIGHT, V. J., LEVIN, M. & COIN, L. J. 2011. Pathway-driven gene stability selection of two rheumatoid arthritis GWAS identifies and validates new susceptibility genes in receptor mediated signalling pathways. *Human Molecular Genetics,* 20**,** 3494-506.

HALE, M. C., XU, P., SCARDINA, J., WHEELER, P. A., THORGAARD, G. H. & NICHOLS, K. M. 2011. Differential gene expression in male and female rainbow trout embryos prior to the onset of gross morphological differentiation of the gonads. *BMC Genomics,* 12**,** 404.

HUANG DA, W., SHERMAN, B. T. & LEMPICKI, R. A. 2009. Systematic and integrative analysis of large gene lists using DAVID bioinformatics resources. *Nature Protocols,* 4**,** 44-57.

KARR, T. L. 2008. Application of proteomics to ecology and population biology. *Heredity,* 100**,** 200-6.

KHATRI, P., VOICHITA, C., KATTAN, K., ANSARI, N., KHATRI, A., GEORGESCU, C., TARCA, A. L. & DRAGHICI, S. 2007. Onto-Tools: new additions and improvements in 2006. *Nucleic Acids Research,* 35**,** W206-11.

LING, M. H., LEFEVRE, C. & NICHOLAS, K. R. 2008. Filtering Microarray Correlations by Statistical Literature Analysis Yields Potential Hypotheses for Lactation Research. *The Python Papers,* 3**,** 4.

MAERE, S., HEYMANS, K. & KUIPER, M. 2005. BiNGO: a Cytoscape plugin to assess overrepresentation of gene ontology categories in biological networks. *Bioinformatics,* 21**,** 3448-9.

MCLEAN, C. Y., BRISTOR, D., HILLER, M., CLARKE, S. L., SCHAAR, B. T., LOWE, C. B., WENGER, A. M. & BEJERANO, G. 2010. GREAT improves functional interpretation of cis-regulatory regions. *Nature Biotechnology,* 28**,** 495-501.

MONTOJO, J., ZUBERI, K., RODRIGUEZ, H., KAZI, F., WRIGHT, G., DONALDSON, S. L., MORRIS, Q. & BADER, G. D. 2010. GeneMANIA Cytoscape plugin: fast gene function predictions on the desktop. *Bioinformatics,* 26**,** 2927-8.

OSBORNE, J. D., ZHU, L. J., LIN, S. M. & KIBBE, W. A. 2007. Interpreting microarray results with gene ontology and MeSH. *Methods in Molecular Biology,* 377**,** 223-42.

REIMAND, J., KULL, M., PETERSON, H., HANSEN, J. & VILO, J. 2007. g:Profiler--a web-

based toolset for functional profiling of gene lists from large-scale experiments. *Nucleic Acids Research,* 35**,** W193-200.

REIMAND, J., TOOMING, L., PETERSON, H., ADLER, P. & VILO, J. 2008. GraphWeb: mining heterogeneous biological networks for gene modules with functional significance. *Nucleic Acids Research,* 36**,** W452-9.

SHEN, Y. & WU, B. L. 2009. Microarray-based genomic DNA profiling technologies in clinical molecular diagnostics. *Clinical Chemistry,* 55**,** 659-69.

WERNER, T. 2008. Bioinformatics applications for pathway analysis of microarray data. *Current Opinions in Biotechnology,* 19**,** 50-4.

SUN, H., FANG, H., CHEN, T., PERKINS, R. & TONG, W. 2006. GOFFA: gene ontology for functional analysis--a FDA gene ontology tool for analysis of genomic and proteomic data. *BMC Bioinformatics,* 7 Suppl 2**,** S23.

ZEEBERG, B. R., QIN, H., NARASIMHAN, S., SUNSHINE, M., CAO, H., KANE, D. W., REIMERS, M., STEPHENS, R. M., BRYANT, D., BURT, S. K., ELNEKAVE, E., HARI, D. M., WYNN, T. A., CUNNINGHAM-RUNDLES, C., STEWART, D. M., NELSON, D. & WEINSTEIN, J. N. 2005. High-Throughput GoMiner, an 'industrial-strength' integrative gene ontology tool for interpretation of multiple-microarray experiments, with application to studies of Common Variable Immune Deficiency (CVID). *BMC Bioinformatics,* 6**,** 168.