MVGST: Tools For Multivariate and Directional Gene Set Testing

Dennis S. Mecham
Utah State University

Follow this and additional works at: https://digitalcommons.usu.edu/gradreports

Part of the Statistics and Probability Commons

Recommended Citation
Mecham, Dennis S., "MVGST: Tools For Multivariate and Directional Gene Set Testing" (2014). All Graduate Plan B and other Reports. 382.
https://digitalcommons.usu.edu/gradreports/382

This Report is brought to you for free and open access by the Graduate Studies at DigitalCommons@USU. It has been accepted for inclusion in All Graduate Plan B and other Reports by an authorized administrator of DigitalCommons@USU. For more information, please contact digitalcommons@usu.edu.
MVGST: TOOLS FOR MULTIVARIATE AND DIRECTIONAL GENE SET TESTING

by

Dennis S Mecham

A report submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE in

Statistics

Approved:

Dr. John R. Stevens
Major Professor

Dr. D. Richard Cutler
Committee Member

Dr. Daniel C. Coster
Committee Member

UTAH STATE UNIVERSITY
Logan, Utah
2014
Abstract

mvGST: Tools for Multivariate and Directional Gene Set Testing

by

Dennis S Mecham, Master of Science
Utah State University, 2014

Major Professor: Dr. John R. Stevens
Department: Mathematics and Statistics

There are many platforms available for simultaneously measuring the relative activity, or expression, levels of all genes in an organism. Genes that have systematically different expression levels between experimental factor levels are called “differentially expressed”. Because genes are annotated based on their known roles in biological processes (BP), molecular functions (MF) and cellular components (CC), gene expression levels can be used to determine relative activity levels of individual BP, MF, or CC between experimental factor levels (this is called gene set testing). Often multiple experimental differences are of interest simultaneously, which necessitates multivariate gene set testing. Only genes that contribute to a BP, MF, or CC are annotated to that gene set, and it is necessary to have a consensus of genes contributing for a BP, MF, or CC to function properly. Because of this, the direction of the differential expression (up of down regulated) matters. mvGST, which is platform and design independent, includes tools to summarize and visualize results from multivariate and directional gene set tests. This flexibility will be demonstrated on two examples. The first example comes from a study evaluating the effect of different types of fertilization on pig embryos at different stages in the embryos’ development. The second example comes from a publicly available gene expression data set from a study evaluating
the effect of different chemotherapy dosages on different leukemia blood cell lines. There
are plans to make mvGST available publicly through CRAN (Comprehensive R Archive
Network) or Bioconductor [1].
Acknowledgments

I am thankful for the help of my advisor, Dr. John Stevens. I am also very grateful for all of the work my wife, Rachel, did to make sure I had the time to complete this.

Dennis Mecham
# Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract</td>
<td>iii</td>
</tr>
<tr>
<td>Acknowledgments</td>
<td>v</td>
</tr>
<tr>
<td>List of Tables</td>
<td>viii</td>
</tr>
<tr>
<td>List of Figures</td>
<td>ix</td>
</tr>
<tr>
<td>1 Background</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Motivating Example 1</td>
<td>1</td>
</tr>
<tr>
<td>1.1.1 Gene Expression Technology</td>
<td>2</td>
</tr>
<tr>
<td>1.1.2 Gene Sets</td>
<td>2</td>
</tr>
<tr>
<td>1.1.3 Statistical Methods Used in Analysis of Motivating Example</td>
<td>3</td>
</tr>
<tr>
<td>1.1.4 Multivariate and Directional Differential Expression</td>
<td>3</td>
</tr>
<tr>
<td>1.1.5 Desired Summary Format</td>
<td>5</td>
</tr>
<tr>
<td>1.2 Motivating Example 2</td>
<td>5</td>
</tr>
<tr>
<td>1.2.1 Gene Expression Technology</td>
<td>5</td>
</tr>
<tr>
<td>1.2.2 Linear Model</td>
<td>6</td>
</tr>
<tr>
<td>1.3 P-value Combination</td>
<td>6</td>
</tr>
<tr>
<td>1.3.1 Fisher’s Method</td>
<td>7</td>
</tr>
<tr>
<td>1.3.2 Stouffer’s Method</td>
<td>7</td>
</tr>
<tr>
<td>1.3.3 Stouffer’s vs Fisher’s</td>
<td>8</td>
</tr>
<tr>
<td>1.3.3.1 Alternative Hypotheses</td>
<td>8</td>
</tr>
<tr>
<td>1.3.3.2 Symmetry</td>
<td>8</td>
</tr>
<tr>
<td>1.4 Multiple Hypothesis Testing</td>
<td>10</td>
</tr>
<tr>
<td>1.5 Potential for Generalization</td>
<td>11</td>
</tr>
<tr>
<td>1.5.1 Translating Gene Names to GO</td>
<td>11</td>
</tr>
<tr>
<td>1.5.1.1 Many to One, One to Many</td>
<td>11</td>
</tr>
<tr>
<td>2 Methods</td>
<td>13</td>
</tr>
<tr>
<td>2.1 profileTable</td>
<td>13</td>
</tr>
<tr>
<td>2.1.1 Note on Gene Name Translation Methods</td>
<td>14</td>
</tr>
<tr>
<td>2.1.1.1 Method 1</td>
<td>14</td>
</tr>
<tr>
<td>2.1.1.2 Method 2</td>
<td>15</td>
</tr>
<tr>
<td>2.1.1.3 Method 3</td>
<td>16</td>
</tr>
<tr>
<td>2.1.1.4 Method 4</td>
<td>17</td>
</tr>
<tr>
<td>2.1.2 Limitations</td>
<td>18</td>
</tr>
<tr>
<td>2.2 profileTable continued</td>
<td>18</td>
</tr>
<tr>
<td>2.2.1 profileTable Arguments</td>
<td>19</td>
</tr>
<tr>
<td>2.2.2 profileTable Returns</td>
<td>21</td>
</tr>
</tbody>
</table>
# 2.3 pickOut

- 2.3.1 *pickOut* Arguments ........................................................................... 22
- 2.3.2 *pickOut* Returns ............................................................................... 23

# 2.4 graphCell

- 2.4.1 *graphCell* Arguments ......................................................................... 23

# 2.5 go2Profile

- 2.5.1 *go2Profile* Arguments ......................................................................... 25
- 2.5.2 *go2Profile* Returns ............................................................................... 25

# 2.6 Demonstration of Package

- 2.6.1 Motivating Example 1 ........................................................................... 25
- 2.6.2 Motivating Example 2a .......................................................................... 28
- 2.6.3 Motivating Example 2b .......................................................................... 32

## 3 Discussion

- 3.1 Summary ..................................................................................................... 35
- 3.2 Noteworthy Challenges
  - 3.2.1 Gene Name Translation ....................................................................... 36
  - 3.2.2 Many-to-One
    - 3.2.2.1 Hartung’s Method ....................................................................... 36
    - 3.2.2.2 Hartung’s versus Stouffer’s ......................................................... 37
  - 3.2.3 One-to-Many ...................................................................................... 41
- 3.3 Possible Future Extensions ......................................................................... 42

## References ..................................................................................................... 43

## Appendix ........................................................................................................ 46
List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1 Summary of design of motivating example 1. The number in each cell of the table is the number of replicates for that combination.</td>
<td>1</td>
</tr>
<tr>
<td>2.1 Example of method 1 used on Affymetrix ID’s that are translated to Entrez ID’s</td>
<td>15</td>
</tr>
<tr>
<td>2.2 Example of method 2 used on Affymetrix ID’s that are translated to Entrez ID’s</td>
<td>16</td>
</tr>
<tr>
<td>2.3 Example of method 3 used on Affymetrix ID’s that are translated to Entrez ID’s</td>
<td>17</td>
</tr>
<tr>
<td>2.4 Example of method 4 used on Affymetrix ID’s that are translated to Entrez ID’s</td>
<td>17</td>
</tr>
<tr>
<td>2.5 Hypothetical gene name translation with many-to-one and one-to-many problems</td>
<td>18</td>
</tr>
<tr>
<td>2.6 Hypothetical gene name translation after running method 4</td>
<td>18</td>
</tr>
<tr>
<td>2.7 output from profileTable for first example:</td>
<td>27</td>
</tr>
<tr>
<td>2.8 Table showing profile of the gene set GO:0045916</td>
<td>29</td>
</tr>
<tr>
<td>2.9 output from profileTable for second example:</td>
<td>30</td>
</tr>
<tr>
<td>2.10 Table showing profile of the gene set GO:0001510</td>
<td>32</td>
</tr>
<tr>
<td>2.11 Table showing profile of the gene set GO:0006171</td>
<td>32</td>
</tr>
<tr>
<td>2.12 output from profileTable for second example, part 2:</td>
<td>33</td>
</tr>
<tr>
<td>2.13 Table showing profile of the gene set GO:0006813</td>
<td>34</td>
</tr>
</tbody>
</table>
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1 GO graph showing GO:0045916 (node 1) and GO:2000258 (node 2) within the framework of all parent GO BP terms</td>
<td>28</td>
</tr>
<tr>
<td>2.2 GO graph showing the 20 gene sets that fit the profile -1 -1 0 0 within the framework of all parent GO BP terms</td>
<td>31</td>
</tr>
<tr>
<td>2.3 GO graph showing the 7 gene sets that fit the profile -1 0 within the framework of all parent GO BP terms</td>
<td>34</td>
</tr>
<tr>
<td>3.1 Scatter plot of Hartung combined p-values versus Stouffer combined p-values</td>
<td>37</td>
</tr>
<tr>
<td>3.2 Scatter plot of Hartung combined p-values versus Stouffer combined p-values when the estimated covariance is positive</td>
<td>38</td>
</tr>
<tr>
<td>3.3 Scatter plot of Hartung combined p-values versus Stouffer combined p-values when the estimated covariance is negative</td>
<td>39</td>
</tr>
<tr>
<td>3.4 Histogram of observed estimated covariances</td>
<td>40</td>
</tr>
<tr>
<td>3.5 Histogram of simulated estimated covariances</td>
<td>41</td>
</tr>
</tbody>
</table>
Chapter 1

Background

1.1 Motivating Example 1

An experiment was performed to better understand the differences in development between different embryo types [2]. In the original study, there were 3 factors of interest: cell type (C), embryo type (T), and gestation days (G). However, in this paper, only embryo type and gestation days will be emphasized. Embryo type has 2 levels of most interest: \textit{in vivo} fertilization (IVV), and \textit{in vitro} fertilization (IVF). Embryo type also has 2 more levels: somatic cell nuclear transfer (“clone”), and parthenogenetic activation (“maternal clone”). Gestation days has 3 levels: 10 days (when embryos have a spherical structure), 12 days (when embryos have a tubular structure), and 14 days (when embryos have an elongated filament structure). All of the samples being considered here come from the inner cell mass of embryonic stem cells (ICM). This makes 12 combinations of embryo type and gestation days that were sampled, but only 6 combinations that will be the main focus. The experimental design is summarized in Table 1.1.

<table>
<thead>
<tr>
<th>embryo type (T)</th>
<th>gestation days (G)</th>
</tr>
</thead>
<tbody>
<tr>
<td>IVV</td>
<td>1 2 1</td>
</tr>
<tr>
<td>IVF</td>
<td>1 1 1</td>
</tr>
<tr>
<td>NT</td>
<td>1 2 1</td>
</tr>
<tr>
<td>PA</td>
<td>2 2 1</td>
</tr>
</tbody>
</table>

Table 1.1: Summary of design of motivating example 1. The number in each cell of the table is the number of replicates for that combination.

To better understand the differences in embryo development, it is desired to determine which biological processes were more, or less, active for each combination of embryo type and gestation days. Because the level of activity of a biological process can be assessed by
the relative activity, or expression, of the genes that contribute to that biological process, it can be said that the objective of this experiment was to “identify and characterize genes with expression differences between embryo types across gestation days.” [2] In addition to determining which biological processes show expression differences, it is also desired to classify those differences as either up-regulated or down-regulated.

1.1.1 Gene Expression Technology

There are a variety of technologies available for simultaneously measuring gene expression levels of all genes in an organism. In this example the RNA-Seq platform [3] was used. The data produced for each sample contains the total number of RNA fragments detected and the number of RNA fragments that map to each gene, thus allowing the expression level of each gene to be measured relative to all other genes. Comparisons of gene expression levels between factor levels (such as embryo types) can be done by having RNA-Seq data on each sample.

1.1.2 Gene Sets

Extensive studies have been done to identify the functions of genes, or the roles genes play in a given organism. Based on these results, genes can be grouped into sets of genes with common roles. One of the most used representations of gene sets is Gene Ontology (GO) [4]. GO has ontologies for biological processes (BP), molecular functions (MF), and cellular components (CC). This example focused on BP. Each biological process has a gene set that includes all genes that have been shown to contribute to the process, but none that inhibit it. Basically, if the genes in the gene set are expressed, the biological process proceeds. If any genes in the set are not expressed, the process may be “disturbed”. Even as few as one gene in the set not being expressed may be enough to disturb the process [5]. Hill, et al [5] gives the example of the shh gene. When all products of the shh gene are missing (the gene is not expressed at all), the process of heart development is “disturbed”. This is just one example, but it shows the potential for the lack of a single gene’s expression to have a negative effect on an entire biological process. For this reason, when testing gene
sets for association with an experimental condition, it is important to test for a consensus of significant differential expression between conditions rather than just test that at least one of the genes in the set is significantly differentially expressed. If a gene set shows consensus of differential expression, then the corresponding biological process can be said to be differentially active between those conditions.

1.1.3 Statistical Methods Used in Analysis of Motivating Example

Because RNA-Seq data is in counts [3], Poisson regression was used in the motivating example. A Poisson regression model was fit for each gene separately. The model used was

$$
\log(E[Y_{ijkl}]) = \log(N_l) + \mu + T_j + G_k + TG_{jk}
$$

(1.1)

where $N_l$ is the total number of RNA fragments detected in replicate $l$, and $Y_{ijkl}$ is the fragment count for gene $i$ in replicate $l$ of gestation days (G) $k$, for embryo type (T) $j$. There was no evidence of overdispersion that would make the Poisson model inappropriate [2].

1.1.4 Multivariate and Directional Differential Expression

In the motivating example, it is desired to compare *in vivo* fertilization versus *in vitro* fertilization (IVV vs. IVF). This was done for each gene separately, as well as for each level of gestation days. Contrasts were used to get each of the desired $p$-values for each gene ($H_0$: IVV = IVF vs. $H_a$: IVV > IVF for each gestation day individually). There were 3 contrasts tested (IVV vs. IVF at 10/12/14 days) for each gene. The three contrasts for each gene were

$$
\psi^{(1)} = T_1 - T_2 + TG_{11} - TG_{21}
$$

(1.2)

$$
\psi^{(2)} = T_1 - T_2 + TG_{12} - TG_{22}
$$

(1.3)

$$
\psi^{(3)} = T_1 - T_2 + TG_{13} - TG_{23}
$$

(1.4)

where the first two levels of embryo type (T) are IVV and IVF, respectively, and the three levels of gestation days (G) are 10, 12, and 14 days, respectively. In all, 3 one-
sided p-values were produced for each gene. As a result, there were 3 one-sided hypothesis
tests performed on each gene (and on each gene set), one test for each level of gestation
days. For the comparison of IVV vs. IVF, each gene set could be classified into a profile
based on the significance results (activity/expression levels for IVV are either significantly
greater than, significantly less than, or not significantly different than for IVF) at each
of the three gestation days (10, 12, 14). For example, a gene that is significantly less
expressed in IVV than IVF at day 10, but not significantly different at days 12 and 14,
would have the profile -1, 0, 0. This results in $3 \times 3 \times 3 = 27$ different possible profiles in the
motivating example. This motivating example is just one case of how a test for differential
expression between conditions can be made at multiple levels of another factor. In this case,
differential expression between embryo types was tested at each level of gestation days. This
has been termed “multivariate differential expression” [2]. In general, there are $3^n$ possible
profiles, where $n$ is the number of levels in the factor across which the comparisons are made
(gestation days, in this case, because the comparison IVV vs. IVF was made for each level
of gestation days).

It seems worth emphasizing that the tests performed in this example, and in general,
by the $R$ [6] package $mvGST$ [7], which was developed for this MS report, will be one-sided
tests. This differs from other commonly used methods of gene set testing [8] [9] [10] [11]
that only attempt to find any differential expression of genes in the gene set, regardless of
direction.

The gene sets of GO only contain genes that contribute to a specific process, and
none that inhibit it [5]. Because of this, the results of a one-sided hypothesis test do
make sense. In this example, a one-sided test can determine if a specific biological process
is more active for IVV than IVF or vice versa, which is more informative than simply
determining that there is a some significant difference. In the case where some genes in the
corresponding gene set are significantly higher expressed in IVV and some are significantly
lower expressed in IVV, a one-sided test is not only more informative, but more meaningful.
In such a case, a two-sided test may conclude that the corresponding biological process is
differentially expressed, but the lack of consensus does not provide evidence of differential activity between IVV and IVF in either direction.

1.1.5 Desired Summary Format

The desired summary format from the multivariate gene set test in this motivating example is a table showing the number of biological processes that fit into each profile for IVV vs. IVF. It is also desired to be able to look at which biological processes fit each profile.

1.2 Motivating Example 2

An experiment was performed to help understand how obatoclax mesylate, a drug owned by GeminX Pharmaceuticals, treats leukemia [12]. A total of 12 samples were taken from two leukemia cell lines, RS4:11 and SEM-K2 (six samples from each). Within each cell line, three treatments were tested: control (CTL), low-dose obatoclax (LOW), and high-dose obatoclax (HIGH). There were 2 samples for each combination of leukemia cell line and treatment for a full $2 \times 3$ factorial design with 2 replicates.

For demonstration purposes of the mvGST package, there are 2 objectives. The first is to create profiles of all biological processes across all 4 combinations of cell lines and LOW/HIGH vs. CTL. The second objective is to stratify the profiles by cell line. This will create profiles for each of the 2 cell lines separately across the comparisons of LOW vs. CTL and HIGH vs. CTL.

1.2.1 Gene Expression Technology

In this study, Affymetrix GeneChips [13] were used to collect gene expression data. There are numerous statistical issues with the raw data, but they can be handled by RMA preprocessing [14] [15] [16].
1.2.2 Linear Model

A linear model was fit for each gene using the *limma* package [17] in R. The model accounted for leukemia cell line (L), treatment level (T), and their interaction. The model used was

\[ Y_{ijkl} = \mu + T_j + L_k + TL_{jk} + \epsilon_{ijkl} \] (1.5)

where \( Y_{ijkl} \) is the log (base 2) of the expression level for gene \( i \) in replicate \( l \) of treatment level \( T \) \( k \), for leukemia cell line \( L \) \( j \). \( \epsilon_{ijkl} \) follows a normal distribution with variance \( \sigma^2 \).

Four contrasts were tested (HIGH/LOW vs CTL at each cell line) with \( H_0: \) HIGH/LOW = CTL vs. \( H_a: \) HIGH/LOW > CTL. The four contrasts tested for each gene were

\[ \psi = T_1 - T_3 + TL_{11} - TL_{31} \] (1.6)

\[ \psi = T_1 - T_3 + TL_{12} - TL_{32} \] (1.7)

\[ \psi = T_2 - T_3 + TL_{21} - TL_{31} \] (1.8)

\[ \psi = T_2 - T_3 + TL_{22} - TL_{32} \] (1.9)

where the levels of treatment \( T \) are high doses of obatoclax, low doses of obatoclax, and control, respectively, and the two levels of leukemia cell line \( L \) are RS4:11 and SEM-K2, respectively.

1.3 P-value Combination

The contrasts in Sections 1.1.4 and 1.2.2 provide p-values for the differential expression of each gene individually. In order to make a conclusion about the differential activity of each biological process it is desired to have one p-value per contrast for each gene set being tested. There are many ways to do this. Fisher’s p-value combination method [18] and Goeman’s [8] global test have been shown to be powerful [19]. However, Goeman’s global test is only for testing association of a gene set with a single “clinical outcome”, which makes it unsuitable for multivariate differential expression. Fisher’s method uses individual genes’
p-values (as can be calculated for each contrast) and then combines them into a single p-value for the gene set. Both of these methods essentially test $H_0$: “none of the genes are differentially expressed” vs. $H_a$: “at least one of the genes is differentially expressed”. As discussed previously in Sections 1.1.2 and 1.1.4, however, a more meaningful null and alternative would be more appropriate.

1.3.1 Fisher’s Method

The test statistic in Fisher’s method is $\sum_{i=1}^{n} -2\log(p_i)$, which follows a $\chi^2_{2n}$ distribution under $H_0$, where $n$ is the number of p-values being combined and $p_i$ is the $i^{th}$ p-value. The combined p-value is the tail probability [18].

The null hypothesis for Fisher’s method is the same as the individual null hypotheses for the p-values being combined (which implies that the original hypotheses tested must all be the same). The alternative hypothesis is that at least one of the common alternatives tested is true. In gene set testing, this means that at least one of the genes is differentially expressed. However, this may not be enough for the gene set, as a whole, to have any significant change in biological process activity. As discussed in section Section 1.1.2, the lack of just one gene being expressed can “disturb” a biological process, which means a single significantly differentially expressed gene may not be enough to overcome the genes that are not differentially expressed. It is more meaningful to use an alternative hypothesis that indicates “collective support” [20].

1.3.2 Stouffer’s Method

Stouffer’s method [21] was shown by Fridley [19] to be nearly as powerful as Fisher’s method for gene set testing. However, Stouffer’s method tests for consensus and has been shown to be superior to Fisher’s method in cases where consensus is more suitable [22], which is the case for gene set testing (as discussed in Section 1.1.2).

Stouffer’s method calculates a combined z-statistic that follows a standard normal distribution. This test statistic is $Z = \sum_{i=1}^{n} (\Phi^{-1}(p_i))/\sqrt{n}$ where $n$ is the number of p-values being combined, $p_i$ is the $i^{th}$ p-value, and $\Phi$ is the cumulative density function of the
standard normal distribution. The combined p-value is the upper tail probability [21].

1.3.3 Stouffer’s vs Fisher’s

Although Fisher’s method is more powerful, Stouffer’s method has a more useful alternative hypothesis for gene set testing and has the benefit of symmetry.

1.3.3.1 Alternative Hypotheses

As discussed in Sections 1.3.1 and 1.3.2, Fisher’s method is unsuitable for combining p-values in gene sets because the alternative for Fisher’s method is that at least one of the genes is differentially expressed, which may or may not actually be enough to make a difference for the entire gene set (and, consequently, the activity of the corresponding biological process). Stouffer’s method tests for a consensus of activity, which is generally a more meaningful alternative for gene set testing.

1.3.3.2 Symmetry

Because all of the p-values being combined are one-sided p-values, it is important that significantly high p-values and significantly low p-values are given equal weight. This is what symmetry means in this case. Stouffer’s method has symmetry. Fisher’s method, however, gives more weight to p-values near 0 than p-values near 1.

One reason symmetry is important is because the direction of the alternative hypothesis is completely arbitrary. If \( p \) is the one-sided combined p-value for a gene set under the alternative hypothesis \( X > Y \), where \( X \) and \( Y \) are activity levels under two different treatments, then \( 1 - p \) should be the combined p-value for the alternative hypothesis \( X < Y \). If this is true, then the arbitrary decision of which way the inequality points in the alternative hypothesis has no bearing on the results. This is true for Stouffer’s method, but not for Fisher’s method. A simple example can demonstrate this. Suppose the alternative hypothesis tested is \( X > Y \) and 2 genes that make up a gene set have the p-values 0.01 and 0.3. With a significance threshold \( \alpha = 0.05 \), a combined p-value of less than 0.025 or greater than 0.975 is considered significant. For this gene set, Stouffer’s method yields a significant
result \((p = 0.0219)\), as does Fisher’s method \((p = 0.0204)\). Now suppose the alternative hypothesis is flipped to be \(X < Y\), so the gene’s p-values are 0.99 and 0.7, respectively. Notice that the magnitude (or absolute difference from 0.5) of the p-values hasn’t changed; the new p-values are equal to one minus the original p-values. Now Stouffer’s method gives a significant result \((p = 0.9781)\), but Fisher’s method gives a non-significant result \((p = 0.9471)\). Also notice that the new Stouffer’s p-value, 0.9781, is equal to one minus the original Stouffer’s p-value, 0.0219 (i.e., Stouffer’s method is symmetric). Fisher’s method, however, is not symmetric \((0.9471 \neq 1 - 0.0204)\). In this example, the arbitrary choice of alternative hypothesis determined whether or not Fisher’s method gave a significant result.

Another reason symmetry is so important, is because some gene sets have bimodal distributions of p-values. Stevens and Isom showed that this may be very common \([2]\). Because of the lack of symmetry, Fisher’s method may give nonsense conclusions for these bimodal gene sets. Assume \(H_0 : X = Y\) vs. \(H_a : X < Y\) are the hypotheses being tested here. One possibility is a gene set is split down the middle with about half of the p-values near 1 (indicating, for example, \(X > Y\)) and about half near 0 (\(X < Y\)). In this scenario, a Stouffer’s combined p-value near 0.5 makes sense (due to the lack of consensus), but Fisher’s method may give a significant result (the Fisher’s combined p-value would be near 0 because “at least one” of the genes in the set was differentially expressed). Suppose a gene set with 2 genes has the p-values 0.01 and 0.99. Both genes are significant, but in opposite directions (i.e., there is a lack of consensus), which should result in a non-significant combined p-value. Stouffer’s method gives a combined p-value of 0.5, but Fisher’s method gives a p-value of 0.0079 (concluding that at least one of the genes is differentially expressed).

Another possibility is that a gene set has a Stouffer’s consensus p-value near 1, but a small number of p-values near 0. It is possible that Fisher’s method will actually give a significant result (small p-value) in favor of the small group of p-values near 0. Suppose a gene set with 4 genes has the p-values 0.9990, 0.9900, 0.9900, 0.0001. In this case, Stouffer’s method gives a combined p-value of 0.9779 (significant consensus that \(X > Y\), but Fisher’s method gives a p-value of 0.0180 (concluding that at least one gene has \(X < Y\)) even though
there was only one gene that didn’t have a p-value near 1.

1.4 Multiple Hypothesis Testing

In both motivating examples, and in any other similar study, many hypothesis tests are being performed. There are multiple tests done on thousands of gene sets. If no adjustment for multiple hypothesis tests is made, then there will be numerous type I errors made. For example, with 20,000 p-values (perhaps 4 tests performed on 5,000 gene sets) there will be approximately 1000 type I errors made if the per-comparison type I error rate is controlled at 0.05.

One approach is to control the familywise type I error rates (FWER) with methods such as Holm’s [23]. These methods assure that the probability of at least 1 type I error is less than or equal to a predetermined $\alpha$. However, they tend to be overly conservative [24].

Another approach, first proposed by Yoav Benjamini and Yosef Hochberg, is to control the false discovery rate (FDR) [24]. The FDR is defined as the expected proportion of type I errors to the number of rejected null hypotheses. Because controlling the FDR allows for some type I errors, it is a more powerful approach than controlling the familywise type I error rate. Controlling the FDR is more suitable for exploratory studies, where it is not essential that there are no type I errors, and increased power is desired. Controlling the FWER is preferred more for confirmatory studies.

The approach originally proposed by Benjamini and Hochberg, however, assumes that the p-values are independent. Benjamini and Daniel Yekutieli showed that this procedure also controls the FDR for some dependency structure among the p-values [25], but the dependency structure among p-values of gene sets is completely unknown [26]. P-values from parent and child gene sets must be dependent since all of the genes in a child set are also in the parent set [4]. Therefore, it is necessary to use a method that will control the FDR regardless of dependent p-values. The method proposed by Benjamini and Yekutieli achieves this [25].
1.5 Potential for Generalization

It is becoming increasingly common for experiments to test multiple hypotheses for each gene, rather than a single comparison [2], which is not what existing gene set testing methods are designed to do. Existing methods like globaltest [8] and ROAST [11] identify gene sets that are associated with a single comparison. Although the method of performing those hypothesis tests depends on the form of the data (e.g., contrasts in a Poisson model for example 1, or contrasts in a linear model for example 2), mvGST takes the p-values for all hypotheses tested on each gene and characterizes the significance, or lack thereof, of each gene set for the tests performed. Because mvGST takes p-values as input, it can be used with any platform for obtaining gene expression levels. mvGST also offers the advantage of performing meaningful one-sided tests, which differs from existing methods.

1.5.1 Translating Gene Names to GO

Each platform for collecting gene expression data has its own gene naming system. GO is not platform specific, but the gene names do need to be translated to GO. A couple of gene naming systems (Affymetrix, Entrez) can easily be translated to GO with tools from the package topGO [27] in R. However, there are many more systems in use that do not easily convert with current tools. There is an R package that is capable of making the conversion, gProfileR [28], but going directly to GO in this package takes far too much time to ever be practical. Testing showed that each extra gene name that is translated to GO adds anywhere from a couple seconds to several minutes to the overall time (presumably depending on how many gene sets the gene is a part of). This increase in time appears to be a linear increase. Instead, this packages is used to translate other gene naming systems to Entrez ID’s which can then be easily translated to GO.

1.5.1.1 Many to One, One to Many

This translation isn’t always one-to-one, however. Suppose the original gene naming system used in a study is called xyz. Undoubtedly, xyz and Entrez ID’s would not be a perfect match. Some genes in xyz would translate to 2 or more genes in Entrez.
groups of 2 or more genes in xyz would translate to a single gene in Entrez. This problem may be avoided by using either Entrez or Affymetrix ID’s originally, since they can easily be mapped to GO, but some researchers may not have data in that form. For a researcher with data from a different naming system, the only way to avoid the translation issues is to provide their own list of gene sets, but this may not be practical. This mvGST package provides several potential solutions that will be discussed in Sections 2.1.1.1, 2.1.1.2, 2.1.1.3, and 2.1.1.4. Admittedly, all of the solutions have drawbacks and the choice of what solution to use is left to the user.
Chapter 2

Methods

There are 4 main functions in the package \textit{mvGST}: \textit{profileTable}, \textit{pickOut}, \textit{graphCell}, and \textit{go2Profile}. After \textit{profileTable} has been run, the other 3 main functions can be used on the result. Sections 2.1, 2.2, 2.3, 2.4 and 2.5 summarize these functions and their arguments. Then the functions are demonstrated on both examples from Chapter 1.

2.1 \textit{profileTable}

\textit{profileTable} achieves the main goal of the package and sets up the use of the other 3 main functions. \textit{profileTable} takes a matrix (rows representing genes and columns representing contrasts) of one-sided p-values, a vector of gene names (in order so that the gene names correspond to the rows of the matrix of p-values), and a vector of contrasts tested (in order so that the contrasts correspond to the columns of the matrix of p-values; format will be discussed in Section 2.2.1) and produces an object of class \textit{mvGST} with a matrix showing the number of gene sets that fit each significance profile for each stratum, if any. Strata used are determined by the vector of contrasts and will be discussed in Section 2.2.1. An \textit{mvGST} object is a list that also contains other information which will be outlined in Section 2.2.2.

The first thing \textit{profileTable} looks at is whether or not the user has provided a list of gene sets (which would be useful if the user want to avoid potential problems caused by translating to Entrez before mapping to GO (see Section 1.5.1.1). If not, then a list is generated that maps the genes to GO. The list generated depends on the type of gene names provided, as well as the desired ontology: biological processes (BP), molecular functions (MF), or cellular components (CC). If the gene names are Affymetrix, Entrez, Genbank, Alias, Ensemble, Symbol, Genename, or Unigene ID’s, then functions from the package \textit{topGO} [27] are used to generate the list. These functions do not provide a complete mapping (genes that are already
mapped to a child set are not mapped to the parent set), so the appropriate OFFSPRING object from the package GO.db [29] is used to complete the list of gene sets. For example, if BP is selected, then GOBPOFFSPRING is used to complete the list generated from topGO. If the gene naming system is anything else, then the function gconvert from the gProfileR package is used to translate the gene names to Entrez. Then the gene set list is generated in the same way as before. If the gene names do have to be translated, then there will be issues with one-to-many and many-to-one (see Section 1.5.1.1) translation. There are 4 methods available to handle these issues in the mvGST package.

2.1.1 Note on Gene Name Translation Methods

Based on the available literature, there does not seem to be a commonly used way of handling the issues (see Section 1.5.1.1) that arise in translating gene names from one naming system to another. The methods described in the following sections do not attempt to be definitive solutions, rather they are just a few possible ways of handling the problem. How valuable these options are varies from case to case (some translations may have very few issues, others may have many). How big of a problem it is to ignore the issue altogether may also depend on what the minimum size gene set included in the analysis is (a smaller gene set would be more affected by a duplicate p-value). Without further research, it is very difficult to know how often the translation issues have a significant effect on the analysis.

2.1.1.1 Method 1

The first method is to ignore the problem all together. This will cause problems, but those problems may not be bad enough to justify any action.

If one gene name translates to many, then all of those genes will have the same p-values and, certainly, will be in many of the same gene sets. This will cause a problem when the p-values in each set are combined using Stouffer’s method because Stouffer’s method assumes independent p-values. If the sets being tested are small, then having two or more p-values that are completely dependent is a big problem. However, if the gene sets being tested are large, then maybe it’s not much of a problem. It is up to the user to decide what constitutes
a big problem.

When many gene names are translated to just one gene name, there are similar problems. This will result in multiple rows of p-values for the same gene. Most likely, the multiple p-values for that gene will be very dependent, since Entrez considers them to be from the same gene. This will cause problems with the Stouffer combination method used within each gene set. Once again, the user must determine if this is a big problem or not.

Table 2.1 shows an example of 4 Affymetrix ID’s that were translated to Entrez ID’s. One of the Affymetrix ID’s translates to two Entrez ID’s, and two of the Affymetrix ID’s translate to the same Entrez ID. After applying method 1, Entrez ID’s 780 and 100616237 have the same p-value because they were translated from the same Affymetrix ID, and Entrez ID 5594 has two p-values because it was translated from two different Affymetrix ID’s.

<table>
<thead>
<tr>
<th>Affymetrix ID</th>
<th>Entrez ID</th>
<th>P-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1007_S_AT</td>
<td>780 and 100616237</td>
<td>0.5045</td>
</tr>
<tr>
<td>1053_AT</td>
<td>5982</td>
<td>0.4665</td>
</tr>
<tr>
<td>1552263_AT</td>
<td>5594</td>
<td>0.3436</td>
</tr>
<tr>
<td>1552264_A_AT</td>
<td>5594</td>
<td>0.3506</td>
</tr>
</tbody>
</table>

After applying method 1

<table>
<thead>
<tr>
<th>Affymetrix ID</th>
<th>Entrez ID</th>
<th>P-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1007_S_AT</td>
<td>780</td>
<td>0.5045</td>
</tr>
<tr>
<td>1007_S_AT</td>
<td>100616237</td>
<td>0.5045</td>
</tr>
<tr>
<td>1053_AT</td>
<td>5982</td>
<td>0.4665</td>
</tr>
<tr>
<td>1552263_AT</td>
<td>5594</td>
<td>0.3436</td>
</tr>
<tr>
<td>1552264_A_AT</td>
<td>5594</td>
<td>0.3506</td>
</tr>
</tbody>
</table>

Table 2.1: Example of method 1 used on Affymetrix ID’s that are translated to Entrez ID’s

2.1.1.2 Method 2

The second method ignores the one-to-many problem, but provides a solution for the many-to-one problem. When many gene names are translated into just one gene names, that one gene has multiple p-values for each contrast and would be included multiple times in every gene set that includes it. Method 2 combines those multiple p-values into one for each contrast so that the new p-value matrix has just one row for that gene like it should
have. Method 2 uses Stouffer’s method [21] to combine the p-values.

Table 2.2 uses the same genes and p-values as in Table 2.1 to demonstrate method 2. The only difference is that Entrez ID 5594 is now only included once in the results with a p-value that is the Stouffer combination of 0.3436 and 0.3506. The internal functions of profileTable also treat the Affymetrix ID as if it were “other 1”, but that only makes a difference if method 4 is used (Section 2.1.1.4).

<table>
<thead>
<tr>
<th>Affymetrix ID</th>
<th>Entrez ID</th>
<th>P-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1007_S_AT</td>
<td>780 and 100616237</td>
<td>0.5045</td>
</tr>
<tr>
<td>1053_AT</td>
<td>5982</td>
<td>0.4665</td>
</tr>
<tr>
<td>1552263_AT</td>
<td>5594</td>
<td>0.3436</td>
</tr>
<tr>
<td>1552264_A_AT</td>
<td>5594</td>
<td>0.3506</td>
</tr>
</tbody>
</table>

Table 2.2: Example of method 2 used on Affymetrix ID’s that are translated to Entrez ID’s

2.1.1.3 Method 3

The third method provides a solution for the one-to-many problem, but ignores the many-to-one problem. When one gene name is translated to many gene names, the p-values for that one gene are duplicated and, essentially, extra data is created. Usually, more data is a good thing, but not when it is fabricated in this way. Those genes are also likely to be in many of the same gene sets, which will violate the assumption of independence in Stouffer’s method. To treat this problem, the many genes must be converted back into one, and the only way to do that is to just pick one. For that reason, method 3 treats the one-to-many problem by simply eliminating all but the first of the “many”. Using this method, any gene that would be translated to many genes, instead is translated to the first gene that gconvert associates it with. Although this does provide a solution to the one-to-many problem, such an arbitrary solution may not be desirable.

Table 2.3 uses the same genes and p-values as in Table 2.1 to demonstrate method 3.
The only difference from method 1 is that Entrez ID 100616237 is no longer included in the results.

<table>
<thead>
<tr>
<th>Affymetrix ID</th>
<th>Entrez ID</th>
<th>P-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1007_S_AT</td>
<td>780 and 100616237</td>
<td>0.5045</td>
</tr>
<tr>
<td>1053_AT</td>
<td>5982</td>
<td>0.4665</td>
</tr>
<tr>
<td>1552263_AT</td>
<td>5594</td>
<td>0.3436</td>
</tr>
<tr>
<td>1552264_A_AT</td>
<td>5594</td>
<td>0.3506</td>
</tr>
</tbody>
</table>

Table 2.3: Example of method 3 used on Affymetrix ID’s that are translated to Entrez ID’s

2.1.1.4 Method 4

The fourth method is merely a combination of the second and third methods. Method 4 first runs method 2, and then a modified version of method 3 that ensures that any rows with Stouffer combined p-values will not be eliminated by mistake.

Table 2.4 uses the same genes and p-values as in Table 2.1 to demonstrate method 4. Notice that the p-values for Entrez ID 5594 were combined into one p-value, just like in method 2, and Entrez ID 100616237 has been eliminated, just like in method 3.

<table>
<thead>
<tr>
<th>Affymetrix ID</th>
<th>Entrez ID</th>
<th>P-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1007_S_AT</td>
<td>780 and 100616237</td>
<td>0.5045</td>
</tr>
<tr>
<td>1053_AT</td>
<td>5982</td>
<td>0.4665</td>
</tr>
<tr>
<td>1552263_AT</td>
<td>5594</td>
<td>0.3436</td>
</tr>
<tr>
<td>1552264_A_AT</td>
<td>5594</td>
<td>0.3506</td>
</tr>
</tbody>
</table>

Table 2.4: Example of method 4 used on Affymetrix ID’s that are translated to Entrez ID’s
2.1.2 Limitations

There are a couple of situations that are not adequately accounted for in these methods. For example, suppose gene names A and B were translated to gene names $\alpha$, $\beta$ and $\gamma$ as shown in Table 2.5. Running method 4 would first use Stouffer’s method to combine the p-values associated with $\beta$. The combined p-value would be associated with the original name “other 1” as shown in Table 2.6. Notice that the one-to-many problem that existed in Table 2.5 is gone. Thus, when method 4 is complete, there will be 3 rows of p-values even though there were only 2 originally (for A and B). Method 4 is supposed to avoid this fabrication of extra p-values. In testing, this problem seems rare.

<table>
<thead>
<tr>
<th>Original Names</th>
<th>Translated Names</th>
<th>Type of P-value</th>
<th>P-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>$\alpha$</td>
<td>raw</td>
<td>0.1</td>
</tr>
<tr>
<td>A</td>
<td>$\beta$</td>
<td>raw</td>
<td>0.1</td>
</tr>
<tr>
<td>B</td>
<td>$\beta$</td>
<td>raw</td>
<td>0.9</td>
</tr>
<tr>
<td>B</td>
<td>$\gamma$</td>
<td>raw</td>
<td>0.9</td>
</tr>
</tbody>
</table>

Table 2.5: Hypothetical gene name translation with many-to-one and one-to-many problems

<table>
<thead>
<tr>
<th>Original Names</th>
<th>Translated Names</th>
<th>Type of P-value</th>
<th>P-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>$\alpha$</td>
<td>raw</td>
<td>0.1</td>
</tr>
<tr>
<td>other 1</td>
<td>$\beta$</td>
<td>Stouffer</td>
<td>0.5</td>
</tr>
<tr>
<td>B</td>
<td>$\gamma$</td>
<td>raw</td>
<td>0.9</td>
</tr>
</tbody>
</table>

Table 2.6: Hypothetical gene name translation after running method 4

2.2 profileTable continued

Once there is a list of gene sets, an object of class mvGST is created with all of the essential information that needs to be passed through the internal functions.

First, the rows of the provided matrix of p-values are grouped according to gene sets. Within each group, p-values are combined using Stouffer’s method. The results are a matrix of p-values with each row representing a gene set and each column representing a contrast.
Second, the FDR is controlled with a Benjamini-Yekutieli adjustment [25] within each contrast (column of the p-value matrix). If the FDR is controlled within each column, then it is controlled overall [30].

Next, the matrix of adjusted p-values is converted to a matrix of ones, zeroes, and negative ones representing significance. The p-values are compared to thresholds $\alpha/2$ and $1 - \alpha/2$. Because the p-values should be one-sided and come from tests of the form $H_0 : \mu_1 = \mu_2$ vs $H_1 : \mu_1 > \mu_2$, p-values that are less than $\alpha/2$ are converted to 1 (significantly greater than), and p-values that are greater than $1 - \alpha/2$ are converted to -1 (significantly less than). All other p-values are converted to 0.

Finally, a table is created showing how many gene sets fit each profile for each stratum, or just how many gene sets fit each profile overall. The possible profiles and the strata, if any, that are displayed in the table are determined by the contrasts argument. The contrasts must be in either the format: \texttt{Var1.Var2}, or in the format: \texttt{Var1} (in other words, \texttt{Var2} is optional). The number of unique levels of \texttt{Var1} determines the number of dimensions of the profiles. The number of levels of \texttt{Var2} determines the number of strata. After the table is created, all 0 rows are removed and the rows are sorted so that the rows with the most gene sets come first.

### 2.2.1 profileTable Arguments

\texttt{profileTable} has the following 12 arguments:

- \texttt{gene.names}
  
  \texttt{gene.names} must be a character vector containing the gene names that correspond to the rows of the matrix of p-values.

- \texttt{contrasts}
  
  \texttt{contrasts} must be a character vector containing the contrasts that correspond to each column in the matrix of p-values. It must either be in the format: \texttt{Var1} or \texttt{Var1.Var2} (\texttt{Var2} is optional). The number of levels in \texttt{Var1} determines the dimensions of the profiles (e.g., if \texttt{Var1} has 2 levels, then there will be two columns for the profiles in
the returned table). *Var2* determines the number of columns, or strata, that will be reported in the returned table for each profile. If *Var2* is not given, then there will only be one column reported, which will be the ontology chosen.

- **pvals**
  
  *pvals* must be a matrix containing one-sided p-values corresponding to the various genes (rows) and contrasts (columns). It is assumed that the one-sided p-values result from per-gene tests of the form $H_0 : \psi = 0$ vs. $H_a : \psi > 0$ for each contrast $\psi$ (so small p-values suggest $\psi > 0$, and large p-values suggest $\psi < 0$).

- **list.groups**
  
  *list.groups* is an optional list containing user-defined gene sets. The names of the list are the names of the gene sets. Each element of the list is a character vector containing the gene names in each gene set.

- **sig.level**
  
  *sig.level* is the $\alpha$ level that should be used. The default is 0.05.

- **gene.ID**
  
  *gene.ID* is the gene naming system used for the gene names. It is used to generate a list of gene sets mapping genes to Gene Ontology sets. For Entrez ID’s, *gene.ID* should be “Entrez”. For Affymetrix ID’s, *gene.ID* should be “affy”.

- **organism**
  
  *organism* is the organism that the genes come from. It is used to generate a list of gene sets mapping genes to Gene Ontology sets. *organism* should be in the form of the first letter of the scientific name followed by the second word of the scientific name with no space in between. For example, human is “hsapiens”. All allowed values are: “agambiae”, “athaliana”, “btaurus”, “celegans”, “cfamiliaris”, “dmelanogaster”, “drerio”, “ecoliK12”, “ecoliSakai”, “ggallus”, ”hsapiens”, ”mmusculus”, ”mmulatta”, ”pfalciparum”, ”ptroglytes”, ”rnorwegianus”, ”scerevisiae”, ”scoielcolor”, ”sscrofa”, ”tgondii”, and ”xlaevis”.
• **affy.chip**

   *affy.chip* is the type of Affymetrix gene chip used, if *gene.ID* is "affy".

• **ontology**

   *ontology* is the ontology that should be used for generating gene sets: "BP", "MF", or "CC".

• **method**

   *method* is the method for handling gene name translation issues. It can be set to either 1, 2, 3 or 4, corresponding to Sections 2.1.1.1, 2.1.1.2, 2.1.1.3, 2.1.1.4, respectively.

• **minsize**

   *minsize* is the minimum size gene set that will be included in the generated list [31].

• **maxsize**

   *maxsize* is the maximum size gene set that will be included in the generated list [31].

### 2.2.2 profileTable Returns

*profileTable* returns an object of class mvGST. An object of class mvGST is a list containing the following components:

• **results.table**

   This is the primary result. It is a matrix with possible profiles as row names and contrasts as column names. The cells of the matrix show how many gene sets have each profile for each contrast.

• **raw.pvals**

   *raw.pvals* is a matrix of the original p-values provided with gene names as row names and contrasts as column names.

• **grouped.raw**

   *grouped.raw* is a matrix of Stouffer combined p-values. Each row is for a gene set and each column is for a contrast.
• *adjusted.group.pvals*

*adjusted.group.pvals* is the same matrix as in *grouped.raw*, but with a Benjamini-
Yekutieli (BY) adjustment being performed within each column.

• *ones.zeroes*

*ones.zeroes* is a matrix showing the significance results of each of the BY adjusted
p-values. 1 means significantly greater. -1 mean significantly less. 0 means not
significant.

• *ord.lev*

*ord.lev* contains the levels of *Var1* (see Section 2.2.1).

• *contrasts*

*contrasts* contains the strata that are the column names of *results.table*. These are
the levels of *Var2* (see Section 2.2.1).

• *group.names*

*group.names* contains the names of the gene sets, whether they were provided or
generated.

### 2.3 *pickOut*

*pickOut* uses the resulting table from *profileTable*, and allows the users to see which
gene sets contribute to the totals in each cell of the table. The user specifies a row and
column of the table, and pickOut returns the ID’s of the gene sets that have the profile of
the specified row for the strata of the specified column.

#### 2.3.1 *pickOut Arguments*

*pickOut* has the following 3 arguments:

• *mvgst*

*mvgst* is a mvGST object with a final *results.table*, as returned by *profileTable*. 
• row

row is the row number of the desired profile in the table returned by profileTable.

• col

col is the column number of the desire stratum. Column refers to the levels of Var2 (Section 2.2.1), if Var2 was used. It is the number of the column after the 1, 0, -1 columns that show the profiles. Default value is 1.

2.3.2 pickOut Returns

The pickOut function returns a character vector containing the ID’s of the gene sets in the specified row and column.

2.4 graphCell

The graphCell function visualizes the gene sets exhibiting a particular profile. For graphCell to be used, the gene sets must have GO ID’s, which will always be the case if the user does not provide their own list of gene sets. graphCell is similar to pickOut, except, instead of just returning the ID’s, graphCell creates a GO graph of the selected gene sets. This gives the user a visual representation of how those gene sets are related. graphCell makes use of tools from the package Rgraphviz [32] to make the GO graph interactive. Within the GO graph, the nodes of the selected cell (i.e., the GO terms classified to the selected profile) are colored yellow.

2.4.1 graphCell Arguments

graphCell has the following 7 arguments:

• object

object is a mvGST object with a final results.table, as returned by profileTable.

• row

row is the row number of the desired profile in the table returned by profileTable.
• col

`col` is the column number of the desired stratum. Column refers to the levels of `Var2` (Section 2.2.1), if `Var2` was used. It is the number of the column after the 1, 0, -1 columns that show the profiles. Default value is 1.

• ontology

`ontology` is the ontology, within Gene Ontology, that should be used ("BP", "MF", "CC").

• interact

`interact` indicates whether or not the graph should be interactive. Clicking on the interactive graph will display a legend with the GO ID and name of the gene set represented by the node closest to the point of the click. If `interactive` is TRUE, the escape button ends interaction with the graph.

• legend.pos

If `interactive` is TRUE, `legend.pos` indicates the desired position of the legend that shows name and GO ID of selected node. Allowable values are “bottomright”, “bottom”, “bottomleft”, “left”, “topleft”, “top”, “topright”, “right”, and “center”.

• print.legend

`print.legend` indicates if a legend should also be printed separately, showing GO names of all nodes. It is impossible to fit GO names, or even just GO ID’s, into nodes of the graph. Instead, the nodes are numbered and the legend shows which number matches which GO name.

2.5 go2Profile

`go2profile` allows a user to see which profile one or more gene sets fit for each strata. The result is a table, or list of tables, that are similar to the table from `profileTable` except that only one gene set is included.
2.5.1 *go2Profile* Arguments

*go2Profile* has the following 2 arguments:

- **names**
  
  *names* is a character vector with the names, or ID’s, of the gene sets of interest. If the gene set names were not provided by the user, then this should be the GO ID’s of the gene sets of interest.

- **object**
  
  *object* is a mvGST object with a final *results.table*, as returned by *profileTable*.

2.5.2 *go2Profile* Returns

A list of matrices is returned by *go2Profile*. Each matrix has possible profiles as the row names and strata as the column names. Ones in the appropriate cells show which profile the gene set fits for each strata and zeroes elsewhere. The names of the list are the names, or ID’s, provided.

2.6 Demonstration of Package

2.6.1 Motivating Example 1

In the first motivating example, *in vivo* fertilization (IVV) is compared to *in vitro* fertilization (IVF) at gestation days (G) 10, 12, and 14. In the example code below, *gene.names1* is a character vector containing the gene names, which are Entrez ID’s. Also in the code below, the vector *contrasts1* contains:


Notice that *profileTable* interprets this as being in the form *Var1.Var2* with the first period being the divider. Also in the code below, *pvals1* is a matrix of one-sided p-values with rows representing genes and columns representing contrasts. The first 6 rows are displayed below:
Even though the samples in this study came from pigs, due to the custom nature of the experimental lab’s gene-naming database, it was necessary to translate the gene names to homologous human genes. For that reason, the organism is human. Since the scientific name for human is homo sapiens, the organism needs to be “hsapiens”. The ontology is set to “BP” because only biological processes are of interest. Since only gene sets with between 5 and 1000 gene were considered, minsize equals 5 and maxsize equals 1000. Thus, the appropriate function call is:

```r
> example1 <- profileTable(gene.names1, contrasts1, pvals1,
+ gene.ID ="Entrez", organism = "hsapiens",
+ ontology = "BP", minsize = 5, maxsize = 1000)
```

Table 2.7 shows the output from profileTable. The output shows that 5510 gene sets (biological processes) with between 5 and 1000 genes show no significant activity difference between IVV and IVF at any of the days tested. Also, using row 9 as an example, 2 gene sets have the profile that they are significantly more active in IVV than IVF at gestation days 10 and 12, but significantly less active at day 14. Suppose that this profile (1, 1, -1) is of particular interest. To see which 2 gene sets fit that profile, pickOut can be used. In this case, the output from profileTable was saved as example1. The profile of interest is in the 9th row. There is only one stratum that was tested, so the column of interest is 1. The correct function call is:

```r
> gene.sets <- pickOut(example1, 9, 1)
```
Table 2.7: output from profileTable for first example:

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>G10</td>
<td>G12</td>
<td>G14</td>
<td>IVV.IVF</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5510</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>730</td>
<td></td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>243</td>
<td></td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>115</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>22</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>17</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

This gives the GO ID’s of the 2 gene sets that fit the profile of interest. It is also possible to make a GO graph of these 2 sets (and all of their parent nodes) using graphCell. The way graphCell is called is very similar to pickOut:

```r
> graphCell(example1, 9, 1, interact = FALSE, print.legend = FALSE)
```

The only difference, in this case, is that interact and print.legend are both set to FALSE for convenience. Figure 2.1 shows the desired graph. The 2 yellow nodes are the 2 gene sets that fit the profile of interest.

Lastly, suppose the gene set GO:0045916 is of particular interest. The function go2Profile can be used by just providing the gene set ID and the original profileTable output:

```r
> profile <- go2Profile("GO:0045916", example1)
```

This output (Table 2.8) says that for the IVV.IVF contrast, GO:0045916 had a profile of (1, 1, -1) across the gestation days 10, 12, and 14.
Fig. 2.1: GO graph showing GO:0045916 (node 1) and GO:2000258 (node 2) within the framework of all parent GO BP terms

2.6.2 Motivating Example 2a

In the second motivating example, high and low doses of obatoclax are given to both of the RS4:11 and SEM-K2 cell lines. These dosed groups can be compared to a control group within each cell line. In the code below, the object `gene.names2` is a character vector containing the gene names, which are Affymetrix ID’s. The type of Affymetrix gene chip used is hgu133plus2. In the code below, the object `contrasts2a` contains:

[1] "RS4Low" "RS4High" "SEMK2Low" "SEMK2High"

Notice that `profileTable` interprets this as being in the form `Var1` because there are no periods in the names. Also in the code below, `pvals2` is a matrix of one-sided p-values with
Table 2.8: Table showing profile of the gene set GO:0045916

rows corresponding to genes and columns corresponding to contrasts. The first 6 rows of `pvals2` are:

\[
\begin{array}{cccc}
[1,] & 0.5044534 & 0.2717903 & 0.5389786 & 0.68724607 \\
[2,] & 0.4665344 & 0.3121148 & 0.1162036 & 0.53931978 \\
[3,] & 0.8135495 & 0.7929617 & 0.9584846 & 0.65778015 \\
[4,] & 0.3793150 & 0.1299377 & 0.5644299 & 0.38994582 \\
[5,] & 0.1318970 & 0.2182358 & 0.5302079 & 0.30793280 \\
[6,] & 0.1416885 & 0.5124254 & 0.4539475 & 0.09247627 \\
\end{array}
\]

The organism is human, or “hsapiens”. The ontology is set to “BP” again. In this case, all biological process gene sets, regardless of size, are included in the analysis. Thus, the appropriate function call is:

```r
> example2a <- profileTable(gene.names2, contrasts2a, pvals2,
+ gene.ID="affy", organism = "hsapiens",
+ ontology = "BP", affy.chip = "hgu133plus2")
```

Table 2.9 shows the output from `profileTable`. The output shows that 10,714 gene sets (biological processes) show no significant difference between any of the dosed groups and the control for either cell line. Also, using row 15 as an example, 20 gene sets have the profile that, for the RS4 cell line, they are significantly less active in both dosed groups, compared to control, with no significant difference between either dosage level and control for the SEMK2 cell line. Suppose this profile (-1, -1, 0, 0) is of particular interest. To see which 20 biological processes fit that profile, `pickOut` can be used again. The profile of interest is in the 15th row. With only one stratum, the correct function call is:
Table 2.9: output from *profileTable* for second example:

> gene.sets <- pickOut(example2a, 15, 1)

[1] "GO:0001510" "GO:0006400" "GO:0006417" "GO:0006431" "GO:0006437"
[6] "GO:0006564" "GO:0006839" "GO:0007005" "GO:0008614" "GO:0008615"
[16] "GO:0034975" "GO:0042819" "GO:0055129" "GO:0071301" "GO:0071494"

This gives the GO ID's of the 20 gene sets that fit the profile of interest. Next suppose a GO graph of those 20 processes is desired. Figure 2.2 shows the desired graph, generated
by the function call:

```r
> graphCell(example2a, 15, 1, interact = FALSE, print.legend = FALSE)
```

Fig. 2.2: GO graph showing the 20 gene sets that fit the profile -1 -1 0 0 within the framework of all parent GO BP terms

Lastly, suppose the gene sets GO:0001510 and GO:0006171 are of particular interest. `Go2Profile` can gives results for both of those at the same time.

```r
> profiles <- go2Profile(c("GO:0001510", "GO:0006171"), example2a)
```

Tables 2.10 and 2.11 show the resulting tables. In this case, this verifies the result from pickOut that GO:0001510 has the profile -1, -1, 0, 0 (consensus less active in both dosage levels compared to control in the RS4 cell line), and it shows that GO:0006171 has
Table 2.10: Table showing profile of the gene set GO:0001510

<table>
<thead>
<tr>
<th>RS4Low</th>
<th>RS4High</th>
<th>SEMK2Low</th>
<th>SEMK2High</th>
<th>BP</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 2.11: Table showing profile of the gene set GO:0006171

<table>
<thead>
<tr>
<th>RS4Low</th>
<th>RS4High</th>
<th>SEMK2Low</th>
<th>SEMK2High</th>
<th>BP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

the profile 1, 1, 0, 0 (consensus more active in both dosage level compared to control in the RS4 cell line).

### 2.6.3 Motivating Example 2b

Another way to look at the second motivating example is to let the dosages of the drug obatoclax (low and high) be the dimensions of the profiles and let the cell lines be strata. This way, within each strata, or cell line, each biological process fits into one of the profiles. Looking at the problem this way, contrasts2b would contain:

```
> contrasts2b <- c("Low.RS4", "High.RS4", "Low.SEMK2", "High.SEMK2")
```

`gene.names2` and `pvals2` are the same objects used in Section 2.6.2. The correct function call is still the same as before, except using `contrasts2b` instead of `contrasts2a`.

```r
> example2b <- profileTable(gene.names2, contrasts2b, pvals2,
+                           gene.ID = "affy", organism = "hsapiens",
+                           ontology = "BP", affy.chip = "hgu133plus2")
```

Table 2.12 shows the output from `profileTable`. The output shows that 11,398 gene sets of biological processes show no significant difference between either of the dosage levels and the control for the RS4 cell line, and 10,970 biological processes show no significance for the SEMK2 cell line. Also, using row 7 as an example, 33 gene sets have the profile that
they are significantly less active for a low dosage of obatoclax compared to control, with no significant difference for a high dose for the RS4 cell line, and 7 gene sets have that same profile for the SEMK2 cell line.

The function calls for `pickOut` and `graphCell` will also be the same as before, except now the user must specify a stratum: column 1 for the RS4 cell line or column 2 for the SEMK2 cell line. Looking at the -1, 0 profile for the SEMK2 cell line, the correct function calls are (Figure 2.3 shows the desired graph):

```r
> gene.sets <- pickOut(example2b, 7, 2)
[1] "GO:0006813" "GO:0043266" "GO:0043268" "GO:0048745" "GO:0051481"
[6] "GO:0071526" "GO:0090075"

> graphCell(example2b, 7, 2, interact = FALSE, print.legend = FALSE)
```

The call for `go2Profile` doesn’t change at all, but looking at GO:0006813 does provide an interesting example because it fits different profiles for the 2 strata.

```r
> profiles <- go2Profile(c("GO:0006813"), example2b)
```

Table 2.13 shows that this biological process has no significant difference between either dosage level and control in the RS4 cell line. However, it is significantly less active for low dose versus control in the SEMK2 cell line and not significantly different for high dose versus control in the SEMK2 cell line.
Fig. 2.3: GO graph showing the 7 gene sets that fit the profile -1 0 within the framework of all parent GO BP terms

<table>
<thead>
<tr>
<th>$\text{GO:0006813}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>-1</td>
</tr>
</tbody>
</table>

Table 2.13: Table showing profile of the gene set GO:0006813
Chapter 3
Discussion

3.1 Summary

The package \textit{mvGST} is independent of platform (for collecting gene expression data) and experimental design. It offers a novel approach to gene set testing, allowing for multivariate and directional comparisons of GO activity levels (using expression of gene sets as proxy). The package \textit{mvGST} offer tools for summarizing the results and visualizing gene sets that have similar results in the multivariate and directional comparisons.

3.2 Noteworthy Challenges

The most difficult challenges in the construction of this package all had to do with making sure the genes could be mapped to Gene Ontology correctly, regardless of the original gene name database used.

3.2.1 Gene Name Translation

The code for a package, like \textit{mvGST}, needs to be as generalizable as possible. For most aspects of the package, this required no more than using variables instead of constants. However, mapping genes to Gene Ontology was much more complicated. There are many databases of gene names and the tools for mapping to Gene Ontology are only available for a few of them. The package \textit{topGO} provides a mapping from Affymetrix and Entrez, along with a few others [27], but there are many more possibilities that must be accounted for. The package \textit{gProfileR} provides mappings to and from numerous databases [28]. \textit{gProfileR} is capable of mapping directly to Gene Ontology, but testing showed that this took too much time to be practical (see Section 1.5.1). That is why \textit{mvGST} uses \textit{gProfileR} to translate genes to Entrez, and then uses \textit{topGO} to create a list of gene sets that map from Entrez to
Gene Ontology. This translation creates two problems: multiple genes translating to one gene (many-to-one), and one gene translating to multiple genes (one-to-many).

### 3.2.2 Many-to-One

With multiple p-values being combined into one, this seems like a perfect time to use a p-value combination method. Stouffer’s method [21] is an option, but it may be inappropriate in this case. Since the multiple p-values are being combined because Entrez considers them to be from the same gene, it is reasonable to believe that the corresponding genes do not act independently. If two or more genes are so closely connected that they can reasonably be considered to be a single gene (and the reason the many-to-one issue happens is because Entrez considers more than one gene to really just be a single gene), then it is certainly possible that they do not act, or are not expressed, independently. Even though the resulting p-values are statistically independent, the potential underlying biological dependence may cause Stouffer’s method to overstate the significance of the gene (the single Entrez ID). In this case (that the genes are not expressed independently), it seems most likely that a positive dependence structure exists. If this is a real problem, then Hartung’s [33] p-value combination method is a viable option.

#### 3.2.2.1 Hartung’s Method

Hartung’s modified inverse normal method [33] accounts for a constant, positive dependence structure between the p-values. If all of the weights are set to 1, the test statistic is:

\[
t(\hat{\gamma}^*, \kappa) = \frac{\sum_{i=1}^{n} t_i}{\sqrt{n + (n^2 - n)[\hat{\gamma}^* + \kappa \sqrt{2/(n + 1)(1 - \hat{\gamma}^*)]}}
\]

(3.1)

where \( \hat{\gamma}^* = \max\{-1/(n - 1), \hat{\gamma}\} \), \( \hat{\gamma} = 1 - \sum_{i=1}^{n} (t_i - \sum_{j=1}^{n} t_j/n)^2/(n - 1) \), \( \kappa = 0.1 + 1/(n - 1) - \hat{\gamma}^* \), and \( t_i \) is the normal quantile of the \( i^{th} \) one-sided p-value. Under the null hypothesis, this test statistic approximately follows the standard normal distribution. Hartung also suggested the \( \kappa = 0.2 \) could be used. In his simulation, which choice of \( \kappa \) was better depended on the dependency structure of the p-values. In this package, \( \kappa = \)}
0.1(1 + 1/(n − 1)) − γ^* is used because κ = 0.2 tends to be overly conservative at times, based on Hartung’s simulations [33].

Hartung showed, via simulation study, that even when the assumption of constant dependence was violated, his method still performed relatively well and did a better job than Stouffer’s method [33]. In the case of gene translation, if there are only 2 p-values being combined this way (which is the most common scenario), then the dependence can be considered constant. If there are more than 2, there is no way to know if the assumption is met, but Hartung’s method should still perform reasonably well, based on his simulations involving p-values that did not have constant dependence [33].

3.2.2.2 Hartung’s versus Stouffer’s

![Scatter plot of Hartung combined p-values versus Stouffer combined p-values](image)

**Fig. 3.1:** Scatter plot of Hartung combined p-values versus Stouffer combined p-values

In theory, arguments can be made for both Hartung’s and Stouffer’s methods for combining p-values from the many-to-one problem. The fact that the p-values are statistically independent favors Stouffer’s method, but the potential underlying biological dependence favors Hartung’s method. To compare these methods, gene names from the second motivating
example (Section 2.6.2) were translated to Entrez ID’s (this is for the sake of demonstration, since the gene names in the second motivating example were Affymetrix ID’s that could already be easily mapped to GO). In this example, there were 10,933 cases of the many-to-one problem. For each of those cases the Hartung combined p-value and the Stouffer combined p-value were computed, as well as the number of p-values being combined and the covariance estimate (assuming constant covariance), which is used to compute Hartung’s test statistic. Figure 3.1 shows the plot of Hartung combined p-values versus Stouffer combined p-values with lines added at the significance thresholds (0.025 and 0.975) and points colored by local density. Sometimes Hartung’s method is more conservative and sometimes Stouffer’s method is more conservative. Figure 3.2 shows that when the estimated covariance of the p-values being combined is positive (cases when Hartung’s method is most likely to be appropriate), Stouffer’s method tends to produce more extreme p-values (closer to 0 or 1), which may be an overstatement of a gene’s true significance. Figure 3.3, however, shows that when the estimated covariance of the p-values being combined is negative (cases
when Hartung’s method is least likely to be appropriate), Hartung’s method tends to produce more extreme p-values, which may be a result of the negative covariance (which is a violation of the assumptions for Hartung’s method) and not the true significance of the corresponding gene.

Since comparing the resulting combined p-values from Hartung’s method and Stouffer’s method does not strongly favor either method, it may be useful to look at the estimated (constant) covariance (or $\hat{\gamma}^*$) of each group of p-values that was combined. Figure 3.4 shows that most of the estimated covariances are positive, but many of them are negative, too. If Hartung’s method was appropriate (if there was a significant degree of biological dependence between genes whose p-values were being combined), then one would expect almost all of the estimated covariances to be positive. It is also concerning that there are small spikes of covariances at a couple of negative values, most notably near -1. One way to determine if these covariance estimates are likely to be produced by biologically dependent genes is to compare them to covariances that were estimated from simulated p-values that have no

Fig. 3.3: Scatter plot of Hartung combined p-values versus Stouffer combined p-values when the estimated covariance is negative
possible biological dependence. Groups of p-values (following a uniform distribution) were simulated such that the groups of simulated p-values were of the same size as the groups of actual p-values combined (e.g. in the observed data there were 5480 groups with 2 p-values, so in the simulated data there were also 5480 groups with 2 p-values). Figure 3.5 shows that the overall shape of the histogram of simulated covariance estimates is somewhat similar to the shape of the histogram of observed covariance estimates. The spikes at -1 and -0.5 are taller in the histogram produced from simulated data, and the spike at 1 is taller in the histogram produced from the observed data. Overall, the observed data have more positive covariance estimates, which supports the use of Hartung’s method. There are, however, still a sizable number (about 500) of observed covariance estimates near -1, for which Hartung’s method would be inappropriate. Although this data set provides some evidence supporting the idea of biological dependence (and, therefore, the possible overstatement by Stouffer’s method of significance for the corresponding genes), there is also some evidence that using Hartung’s method would not be appropriate for all cases of the many-to-one problem. With a lack of overwhelming evidence in favor either Stouffer’s or Hartung’s method, *mvGST* offers Stouffer’s method as a solution to the many-to-one problem because the platform chosen by the researcher (to measure gene expression levels)
Simulated Covariance Estimates

Fig. 3.5: Histogram of simulated estimated covariances

considers the many genes (of the many-to-one) to be separate, independently expressed genes. If those genes are independently expressed, then Stouffer’s method is the most appropriate way to combine the corresponding p-values.

### 3.2.3 One-to-Many

Ignoring the one-to-many problem artificially creates more p-values than there were tests. Also, the duplicated p-values are likely to end up in the same gene sets which would grossly violate the assumption of independence in Stouffer’s method. The only way to return to the correct number of p-values is to eliminate all but one of the genes that the original gene translated to. In *mvGST*, the first gene listed by `gconvert` is the one that is not eliminated. This forces the translation to be one to one which resolves the above problems. However, bias is possible if there is something systematically different about the first genes. This solution also creates a loss of information. If the original gene name maps to multiple new names, there must be a biological reason for that. It is up to the researcher to decide which is worse: possibly significant loss of information (because genes were arbitrarily ignored), or possibly significant violation of statistical assumptions (because, when the p-values are combined with Stouffer’s method for each gene set, p-values that are 100% dependent will
be treated like they are independent).

3.3 Possible Future Extensions

Two concerns that could be resolved with future research or extensions are: the possible over-conservativeness of the Benjamini-Yekutieli adjustment, and the violation of the assumption of independence in Stouffer’s p-value combination method.

The Benjamini-Yekutieli adjustment [25] for multiple hypothesis tests accounts for any type of dependency in controlling the false discovery rate [24]. The focus level adjustment [26] specifically accounts for the Gene Ontology structure in controlling the family-wise error rate. Even though the focus level adjustment controls the family-wise type I error rate, not the FDR, it may be a preferred adjustment because of the advantage of taking into account the relationship between parent and child gene sets. One disadvantage of the focus level adjustment is the amount of time that it takes to run. It may be possible in the future to incorporate the focus level adjustment in a way that does not take an unreasonable amount of time [34].

Since mvGST starts with p-values provided by the user, there is no current way to account for the unknown dependency structure within those p-values. Because this dependency violates one of the assumptions of Stouffer’s method, this is an area that could be improved in the future. It may be impossible to account for all possible analyses that create those p-values (due to the infinite range of possible experimental designs), but one possible extension of this package could be to provide a few ways of generating those p-values. If the user used mvGST to perform the original analysis, then a monte carlo simulation could be used to account for the dependency. This would be very computationally expensive, but it may be worth the time for some users.
References


Appendix
profileTable <-
function(gene.names, contrasts, pvals, list.groups = NULL, sig.level = .05,
    gene.ID, organism, affy.chip, ontology = "BP", method = 1,
    minsize = 1, maxsize = Inf){
# Takes a matrix of p-values and returns a table with the number of gene sets
# that fall into each possible profile
#
# Args:
# gene.names: A character vector containing the gene names that correspond
# to the rows of the matrix of p-values
# contrasts: A character vector containing the contrasts that correspond to
# each column in the matrix of p-values. Must be in 1 of 2 formats:
# Vari.Var2 or Vari
# pvals: A matrix containing the p-values corresponding to the various genes
# and contrasts
# list.groups: An optional list containing user-defined gene sets
# sig.level: The alpha level that should be used. Default is .05.
# gene.ID: Gene naming system used for the gene names. Used to generate list
# of gene sets mapping genes to Gene Ontology sets
# organism: The organism that the genes come from. Used to generate list
# of gene sets mapping genes to Gene Ontology sets
# affy.chip: The type of affy.chip used, if gene.ID == "affy".
# ontology: The ontology that should be used for gene sets: BP, MF, or CC
# method: The method for handling gene name translation issues
# minsize: The minimum size gene set that will be included in the list
# maxsize: The maximum size gene set that will be included in the list
#
# Returns:
# The primary return is a table containing the number of gene sets that fall
# into each of the possible significance profiles
if (!is.matrix(pvals)){
    stop("pvals must be a matrix")
}
# adding ".BP" to contrasts if in the form Vari
vars <- length(unlist(strsplit(contrasts[1], ".[.]")))
if (vars == 1){
    contrasts <- paste(contrasts, ontology, sep = ".")
}
# accounting for the possibilities of 1's and 0's in the p-values
f.one <- 1 -.Machine$double.eps
f.zero <- .Machine$double.eps
pvals <- ifelse(pvals == 1, f.one, pvals)
pvals <- ifelse(pvals == 0, f.zero, pvals)

if (is.null(list.groups)){
  if (any(gene.ID == c("affy", "entrez", "genbank", "alias", "ensemble",
    "symbol", "genename", "unigene")) != TRUE){
    # Converts gene names to entrez database that can be mapped directly to GO.
    old.names <- character()
    new.names <- character()
    # gene names are converted 1000 at a time because doing more than that
    # may cause an error (vector too large)
    for (i in 0:floor(length(gene.names) / 1000)){
      low <- i * 1000 + 1
      high <- (i + 1) * 1000
      if (high > length(gene.names)){
        high <- length(gene.names)
      }
      gene.names <- toupper(gene.names)
      converted1 <- gconvert(gene.names[low:high], target = "ENTREZGENE_ACC")
      old.names <- c(old.names, as.character(converted1$alias))
      new.names <- c(new.names, as.character(converted1$target))
    }

    # trimming all-numeric ID's down to just the numbers
    if(grep("^[[:digit:]]+$", str_trim(gene.names[1])) == 1){
      all.numeric <- TRUE
    }
    new.genes <- cbind(old.names, new.names)
    new.genes[, 2] <- gsub("ENTREZGENE_ACC:", "", new.genes[, 2])
    if (all.numeric){
      new.genes[, 1] <- substr(new.genes[, 1],
        regexpr("^[[:digit:]]+$", new.genes[, 1]),
        nchar(new.genes[, 1]))
    }

    # eliminating duplicate rows (i.e., both the new and old gene names
    # are the same)
    duplicate <- rep(FALSE, length(new.genes[, 1]))
}
for (i in 2:length(new.genes[, 1])){
    duplicate[i] <- ifelse(all(new.genes[i, ] == new.genes[i-1, ]), TRUE, FALSE)
}
new.genes <- new.genes[!duplicate, ]

converted <- geneNameConvertRows(pvals, gene.names, new.genes, method)
pvals <- converted$p.mat
gene.names <- converted$genes
gene.ID <- "entrez"
}
# generates list of gene sets
list.groups1 <- generateGeneSets(ontology=ontology, species = organism, 
    ID = gene.ID, affy.chip)
offspring <- get("as.list", pos = "package:AnnotationDbi")
    (get(paste("GO", ontology, "OFFSPRING", sep = "")))
list.groups <- sapply(1:length(offspring), 
    function(x) fillInList(list.groups1[[names(offspring[x])]],
        names(offspring)[x],
        offspring, list.groups1))

names(list.groups) <- names(offspring)
size <- sapply(list.groups, length)
keepers <- (size <= maxsize & size >= minsize)
list.groups <- list.groups[keepers]

# Creates MVGST object
pmat <- mvGSTObject(gene.names, contrasts, pvals, list.groups)
# Converts p-values matrix for genes to a p-value matrix for gene sets
grouped.pmat <- separate(pmat, list.groups)

# Perform Benjamini-Yekutieli adjustment for multiple hypothesis testing
by.adjusted <- oneSideBYAdjust(grouped.pmat)
# Creates the final output
one.zero <- changeTO10(by.adjusted, sig.level=.05)
almost.final <- finalResults(one.zero)
near.final <- cut(almost.final)
final <- mvSort(near.final)
return(final)
function(mvgst, row, col = 1) {
    # Return the gene sets that are included in a specific cell of the final
    # results table of a mvGST object.
    #
    # Args:
    #  mvgst: A mvGST object with a final results table.
    #  row: The row of the desired cell.
    #  column: The column of the desired cell.
    #
    # Returns:
    #  A character vector containing the gene sets in the desired cell.
    table <- mvgst$results.table
    raw.profile <- dimnames(table)[[1]][row]
    temp <- substr(raw.profile, 3, nchar(raw.profile) - 1)
    temp2 <- as.integer(strsplit(temp, ",")[[1]])
    the.profile <- matrix(temp2, nrow = 1)
    observed <- profileCombine(mvgst$ones.zeros)[[col]]
    t <- apply(observed, MARGIN = 1, FUN = function(x) all(x == the.profile))
    sets <- mvgst$group.names[!is.na(t) & t]
    return(sets)
}

graphCell <-
function(object, row, col = 1, ontology = "BP", interact = TRUE, legend.pos = "bottomleft",
         print.legend = TRUE) {
    # Graphs the gene sets from a given cell of the final results table
    # of a mvGST object.
    #
    # Args:
    #  object: A mvGST object with a final results table.
    #  row: The row of the desired cell.
    #  column: The column of the desired cell.
    #  ontology: The ontology, within Gene Ontology, that should be used ("BP", "MF", "CC").
    #  interact: Indicates whether or not the graph should be interactive.
    #  legend.pos: If interactive, indicates the desired position of the legend.
    #  print.legend: Indicates if the legend should also be printed separately
    sets <- pickOut(object, row, col)
    if (ontology == "BP") {
        g.sub <- GOGraph(sets, GOBPPARENTS)
    } else {
}
if (ontology == "MF"){
    g.sub <- GOGraph(sets, GOMFPARENTS)
} else {
    if (ontology == "CC"){
        g.sub <- GOGraph(sets, GOCCPARENTS)
    } else {
        stop("invalid ontology")
    }
}

if (ontology == "MF"){
    g.sub <- removeNode("all", g.sub)
}

interactiveGraph(g.sub, sets, interact, legend.pos, print.legend)

# Performs the same operation as go2GeneSet, but for multiple gene sets

# Args:
# names: A character vector with the names, ID's, of the gene sets
# object: A mvGST object with a final results table.

# Returns:
# A list of matrices. Each matrix has possible profiles as the
# row names and contrasts as the column names. Ones in the appropriate
# cells showing which profile the gene set fit for each contrast and
# zeroes elsewhere.

result <- lapply(names, go2GeneSet, object)
list.results <- list()

for (i in 1:length(result)){
    temp <- list(results.table = NULL, ord.lev = NULL)
    temp$results.table <- result[[i]]
    class(temp) <- "mvGST"
    temp1 <- cut(temp)
    list.results[[i]] <- temp1
}

names(list.results) <- names

return(list.results)
separate <- function(pmat, list.groups){
  # Uses Stouffer's method to combine p-values for gene sets
  #
  # Args:
  #  pmat: An MVGST object with a matrix of p-values with corresponding
gene names as the row names
  #  list.groups: A list of character vectors containing the gene sets
  #
  # Returns:
  #  An MVGST object with a matrix of p-values with corresponding gene
  # set names at the row names
  mvgst <- pmat$raw.pvals
  new.ps <- matrix(NA, nrow = length(list.groups), ncol = ncol(mvgst))
  for (i in 1:length(list.groups)){
    t <- is.element(rownames(mvgst), list.groups[[i]])
    if (any(t)){
      new.ps[i, ] <- apply(matrix(mvgst[t, ], nrow = sum(t)), MARGIN = 2, FUN = combinePvalues)
    }
  }

  new.ps <- matrix(new.ps, dimnames = list(rep(1:nrow(new.ps)),
                                         colnames(mvgst)),
                                      nrow = nrow(new.ps))

  pmat$grouped.raw <- new.ps
  return(pmat)
}

finalResults <- function(pmat){
  # Takes a matrix of significance results (-1, 0, 1) and creates a table
  # with the total number of gene sets that fall into each possible
  # profile
  #
  # Args:
  #  pmat: An MVGST object containing the matrix of ones, zeroes, and
  #      negative ones indicating whether each gene set is
# significantly less differentially expressed, not
# significantly differentially expressed, or significantly
# greater differentially expressed for each contrast
#
# Returns:
# An MVGST object containing a matrix with row names that are each
# possible significance profile for the first variable, column names
# that are each combination of the remaining variables, and cell
# values that indicate the number of gene sets that have the
# corresponding significance profile.
mat <- pmat$ones.zeros
# setting up the final results matrix
profile <- profiles(mat)
columns <- tableColumns(mat)
observed <- profileCombine(mat)

profs <- as.list(rep(NA, nrow(profile)))
# counting the number of gene sets that fit in each cell of matrix
for (i in 1:nrow(profile)){
  profs[[i]] <- profile[i,]
}
final <- matrix(NA, nrow = nrow(profile), ncol = length(columns),
    dimnames = list(profs, columns))
for (i in 1:nrow(final)){
  for (j in 1:ncol(final)){
    final[i,j] <- sum(apply(observed[[j]], MARGIN = 1, FUN = function(x) all(x == profile[i,])), na.rm = TRUE)
  }
}
pmat$results.table <- final
pmat$ord.lev <- dimnames(profile)[[2]]
pmat$contrasts <- dimnames(final)[[2]]
return(pmat)
}

profiles<-function(mat){
# Takes a matrix of significance results (-1, 0, 1) and creates a matrix
# with each row being a possible significance profile and the number of
# rows being equal to the number or possible significance profiles.
#
# Args:
# mat: A matrix with column names that are the in the format
# Var1.Var2.Var3...Var(n-1).Var(n) that represent the contrasts being tested.

# Returns:
# A matrix with each row being a possible significance profile and the number of rows being equal to the number of possible significance profiles.

names <- dimnames(mat)[[2]]
prof <- rep(NA, length(names))
for (i in 1:length(names)){
    prof[i] <- unlist(strsplit(names[i], ".["])[[1]]
}
t <- duplicated(prof)
unique <- prof[!t]
nprofiles <- 3 ^ length(unique)
profiles <- matrix(rep(NA), ncol = length(unique),
    nrow = nprofiles,
    dimnames = list(rep(1:nprofiles), unique))
for (i in 1:length(unique)){
    profiles[, i] <- rep(c(rep(-1, nprofiles / 3 ^ i),
        rep(0, nprofiles / 3 ^ i),
        rep(1, nprofiles / 3 ^ i)), 3 ^ (i - 1))
}
return(profiles)

mvGSTObject <- function(gene.names, contrasts, pvals, groups){
    # Creates an object of class mvGST
    #
    # Args:
    # gene.names: A character vector containing the gene names that correspond to the rows of the matrix of p-values
    # contrasts: A character vector containing the contrasts that correspond to each column in the matrix of p-values. Must be in format: Var1.Var2.Var3...Var(n-1).Var(n)
    # pvals: A matrix containing the p-values corresponding to the various genes

    #
# and contrasts
# groups: An optional list containing user-defined gene sets
#
# Returns:
# A mvGST object with most things still empty
object <- matrix(pvals, dimnames = list(gene.names, contrasts),
                 nrow = length(gene.names))
obj <- list(raw.pvals = object,
            results.table = matrix(rep(NA, 4), nrow = 2),
            ord.lev = NA,
            contrasts = NA,
            grouped.raw = NA,
            adjusted.group.pvals = NA,
            ones.zeros = NA,
            group.names = names(groups))
class(obj) <- "mvGST"
return(obj)

tableColumns <- function(mat){
  # Takes a matrix of significance results (-1, 0, 1) and returns the
  # contrasts that need to be the column names for the final results
  # matrix
  #
  # Args:
  # mat: A matrix with column names that are in the format
  #      Vari.Var2 or Var1that represent the contrasts
  #      being tested.
  #
  # Returns:
  # A character vector in the format Var2 or "ontology"
  names <- dimnames(mat)[[2]]
  col <- rep(NA, length(names))
  columns <- rep(NA, length(names))
  for (i in 1:length(names)){
    col[i] <- unlist(strsplit(names[i], "[.]"))[1]
    columns[i] <- substr(names[i], nchar(col[i]) + 2, nchar(names[i]))
  }
t <- duplicated(columns)
unique <- columns[!t]
return(unique)
}

convertPvalues <- function(pvals, relativity, two.sided = TRUE){
  # Converts a matrix of p-values from two-sided to one-sided or vice versa.
  #
  # Args:
  # pvals: A matrix of p-values.
  # relativity: Only used when two.sided == TRUE. Numeric value that is
  # greater than 0 if the one-sided p-value should be less
  # than .5.
  # two.sided: TRUE if pvals contains two-sided pvalues, FALSE if pvals
  # contains one-sided pvalues.

  if(two.sided){
    newP <- ifelse(relativity < 0, 1 - pvals / 2, pvals / 2)
  } else {
    newP <- ifelse(pvals < .5, pvals * 2, (1 - pvals) * 2)
  }
return(newP)
}

combinePvalues <- function(pvals){
  # Uses Stouffer's method to combine p-values
  #
  # Args:
  # pvals: A vector of p-values.
  #
  # Returns:
  # A single combined p-value
comb.p <- pnorm(sum(qnorm(pvals)) / sqrt(length(pvals)))
return(comb.p)
}

profileCombine <- function(mat){
  # Takes a matrix of significance results (-1, 0, 1) and list of matrices

# with each matrix containing the significance profiles for a given
# Var2.Var3...Var(n-1).Var(n)
#
# Args:
# mat: A matrix with column names that are the in the format
# Var1.Var2.Var3...Var(n-1).Var(n) that represent the contrasts
# being tested.
#
# Returns:
# list of matrices with each matrix containing the significance
# profiles for a given Var2.Var3...Var(n-1).Var(n)

names <- dimnames(mat)[[2]]
col <- rep(NA, length(names))
duplicates <- rep(NA, length(names))
columns <- tableColumns(mat)
each <- profiles(mat)
mats <- rep(list(matrix(rep(NA), nrow = nrow(mat),
ncol = ncol(each))), length(columns))
for(i in 1:length(columns)){
  for (j in 1:length(names)){
    col[j] <- unlist(strsplit(names[j], "."))[1]
    duplicates[j] <- substr(names[j], nchar(col[j]) + 2, nchar(names[j]))
  }
  t <- duplicates == columns[i]
  mats[[i]] <- matrix(mat[, t], nrow = nrow(mat))
}
return(mats)

changeTO10 <- function(pvals.mat, sig.level){
  # Converts a matrix of one-sided p-values to matrix of ones, zeroes and
  # negative ones where 1 represents significantly greater, 0 represents
  # no significance, and -1 represents significantly less than
  #
  # Args:
  # pvals.mat: A mvGST object containing a matrix of one-sided p-values.
  # sig.level: Significance level to be used. P-values less than
  # sig.level / 2 are converted to 1. P-values greater
  # than 1 - sig.level / 2 are converted to -1.
  #
# Returns:
# A matrix of ones, zeroes, and negative ones.
pvals <- pvals.mat$adjusted.group.pvals
ind.profile <- matrix(rep(0), nrow = nrow(pvals), ncol = ncol(pvals),
                        dimnames = dimnames(pvals))
ind.profile <- ifelse(pvals <= sig.level / 2, 1,
                      ifelse(pvals >= 1 - sig.level / 2, -1, 0))
pvals.mat$ones.zeros <- ind.profile
return(pvals.mat)
}

oneSideBYAdjust<-function(pval.mat){
  # Converts one-sided p-values to two-sided. Performs a Benjamini-Yekutieli
  # adjustment on the two-sided p-values. Converts the adjusted two-sided
  # p-values back to one-sided p-values.
  # 
  # Args:
  # pval.mat: A mvGST object that contains a matrix of Stouffer combined
  # p-values
  # 
  # Returns:
  # A mvGST object that also contains a matrix of BY adjusted, Stouffer
  # combined p-values.
pvals <- pval.mat$grouped.raw

two.sided <- convertPvalues(pvals, two.sided = FALSE)
two.adjusted <- apply(two.sided, MARGIN = 2, FUN = p.adjust, method = "BY")
relative <- ifelse(pvals < .5, 1, -1)
one.combined <- convertPvalues(two.adjusted, relative)
pval.mat$adjusted.group.pvals <- matrix(one.combined, nrow = nrow(pvals),
                                          dimnames = dimnames(pvals))
return(pval.mat)
}

cut <- function(y){
  # Removes the rows with all zeroes from the final results table in a mvGST object.
  # 
  # Args:
mvSort <- function(y){
  # Sorts the rows in the final results table in a mvGST object from the greatest
  # row total to the least.
  #
  # Args:
  #  y: A mvGST object that contains a final results matrix
  #
  # Returns:
  #  A mvGST object that contains a final results matrix with rows sorted by
  #  row total.
  x <- y$results.table
  temp <- x
  row.sum <- apply(temp, MARGIN = 1, FUN = sum)
  ranked <- rank(1 / row.sum, ties.method = "first")
  for (i in 1:nrow(temp)){
    x[ranked[i], ] <- temp[i, ]
    dimnames(x)[[1]][ranked[i]] <- dimnames(temp)[[1]][i]
  }
  y$results.table <- x
  return(y)
}

# y: A mvGST object that contains a final results matrix
#
# Returns:
# A mvGST object that contains a final results matrix with no all zero rows
x <- y$results.table
temp.mat <- matrix(0, ncol = ncol(x))
temp.names <- NA
for (i in 1:nrow(x)){
  if (sum(x[i,]) != 0){
    temp.mat <- rbind(temp.mat, x[i, ])
    temp.names <- c(temp.names, dimnames(x)[[1]][i])
  }
}

temp.mat <- temp.mat[-1, ]

final.mat <- matrix(temp.mat, ncol = ncol(x),
                     dimnames = c(list(temp.names), list(dimnames(x)[[2]]))
)

y$results.table <- final.mat
return(y)
}
y$results.table <- x
return(y)
}

print.mvGST <- function(x, ...){
  # Prints an object of class mvGST
  
  # Args:
  # x: A mvGST object
  if (is.na(x$results.table[1, 1])){
    print(x$raw.pvals)
  } else {
    y <- x$results.table
    col.names1 <- dimnames(y)[[2]]
    col.names <- ifelse(nchar(col.names1) < 4, paste(col.names1, " "), col.names1)
    row.spaces <- nchar(col.names)
    row.names <- dimnames(y)[[1]]
    col.spaces <- max(nchar(row.names))
    for (i in 1:length(row.names)){
      temp.name <- substr(row.names[i], 3, nchar(row.names[i])-1)
      row.names[i] <- " 
      counter <- 1
      for (j in 1:nchar(temp.name)){
        char <- substr(temp.name, j, j)
        if (char == "-"){
          row.names[i] <- paste(row.names[i], char, sep = "")
        } else {
          if (char == "0"){
            row.names[i] <- paste(row.names[i], char)
            for (k in 1:nchar(x$ord.lev[counter])){
              row.names[i] <- paste(row.names[i], "")
            }
          } if (char == "1"{  
            if (substr(temp.name, j - 1, j - 1) == "-"){
              row.names[i] <- paste(row.names[i], char, sep = "")
            } else {
              row.names[i] <- paste(row.names[i], char, sep = "")
            }
          }
        }  
      }
      counter <- counter + 1
    }
  }
}
else {
    row.names[i] <- paste(row.names[i], char)
}
for (k in 1:nchar(x$ord.lev[counter])){
    row.names[i] <- paste(row.names[i], "")
}

counter <- counter + 1
}
}
}
}
}

row.names <- gsub("", " ", row.names)
col.spaces <- max(nchar(row.names))
cat(" ")
for (i in 1:length(x$ord.lev)){
    cat(x$ord.lev[i], " ")
}

cat(col.names, "\n")
for (i in 1:length(row.names)){
    cat(row.names[i], "")
    for (j in 1:ncol(y)){
        spaces <- 1 + row.spaces[j] - nchar(y[i, j])
        cat(y[i, j], rep(" ", max(spaces, 1)))
    }
    cat("\n")
}
}

summary.mvGST <-
function(object, ...){
    # Creates a summary of the mvGST object, giving number of gene sets tested,
    # levels of the ordered factor, number of other factors, number of possible
    # profiles, number of profiles that have gene sets, and number of contrasts
    # tested.
    #
# Args:
# object: A mvGST object
y <- object$results.table
gene.sets <- sum(y[, 1])
ordered.vars <- length(object$ord.lev)
summ <- c(gene.sets, 3 ^ ordered.vars, nrow(y), ncol(y))
class(summ) <- "summary.mvGST"
return(summ)
}

print.summary.mvGST <- function(x, ...){
# Prints an object of class summary.mvGST
#
# Args:
# x: A summary.mvGST object
cat("", x[1], "gene sets \n", x[4], "possible profiles \n",
x[5], "profiles used \n", x[6], "strata \n")
}

geneNameConvertRows <- function(pvals, gene.names, new.names, method = 1){
# Handles translation from one set of gene names to another.
#
# Args:
# pvals: A matrix of p-values.
# gene.names: A character vector containing the original gene names.
# new.names: A matrix with the first column containing the original gene
# names, and the second column containing the corresponding new
# names.
# method: A number from 1 to 4 that indicates what method should be used to
# handle duplicates in the name translation.
# Method 1 does nothing. As a result, some rows of p-values will be
# duplicated when one name translates to many. Some rows will also
# have the same gene name when many names translate to just one.
# Method 2 uses Hartung's modified inverse normal method to combine
# p-values when many names translate to just one.
# Method 3 accounts for when one name translates to many. Instead of
# duplicating rows of p-values, only the first of the new names is
# used.
# Method 4 combines methods 2 and 3. First method 2 is performed, then
# method 3.
#
# Returns:
# A list containing the new p-value matrix, the new gene names, and the old
gene names if method != 1.

if (method == 1){
  new.pvals <- matrix(NA, nrow = nrow(new.names), ncol = ncol(pvals))
  for (i in 1:nrow(new.names)){
    new.pvals[i, ] <- pvals[new.names[i, 1] == gene.names]
  }
  return(list(p.mat = new.pvals, genes = new.names[, 2]))
}
if (method == 3){
  return(method3(pvals, gene.names, new.names))
}
if (method == 2){
  return(method2(pvals, gene.names, new.names))
}
if (method == 4){
  method.two <- method2(pvals, gene.names, new.names)
  new.new.names <- data.frame(method.two$old.names, method.two$genes)
  return(method4(method.two$p.mat, method.two$old.names, new.new.names))
}
}

hartung <- function(pvals){
  # Uses Hartung's modified inverse normal method to combine a set of
  # p-values
  #
  # Args:
  # pvals: A vector of p-values
  #
  # Returns:
  # A single p-value.
  n <- length(pvals)
  t <- qnorm(pvals)
  rho.hat <- 1 - sum((t - sum(t) / n) ^ 2) / (n - 1)
  rho.hat.star <- max(-1 / (n - 1), rho.hat)

  rho.hat <- 1 - sum((t - sum(t) / n) ^ 2) / (n - 1)
  rho.hat.star <- max(-1 / (n - 1), rho.hat)
kappa <- .1 * (1 + 1 / (n-1) - rho.hat.star)
combined.test.statistic <- sum(t) /
  sqrt(n + (n - 2 - n) * 
    (rho.hat.star + kappa * sqrt(2 / (n+1)) * (1 - rho.hat.star)))
return(pnorm(combined.test.statistic))

method3 <- function(pvals, gene.names, new.names){
  # Accounts for the one-to-many gene name translation issue
  # by only using the first of the many new names that are
  # possible.
  
  # Args:
  # pvals: A matrix of p-values.
  # gene.names: A character vector with the old gene names
  # new.names: A data frame with old gene names in the first
  # column and corresponding new gene names in the
  # second column.
  
  # Returns:
  # A list with the new matrix of p-values and the corresponding
  # gene names.
  new.names <- new.names[order(new.names[, 1]), ]
  index <- rep(TRUE, nrow(new.names))
  for (i in 2:length(new.names[, 1])){
    if (any(new.names[i, 1] == new.names[1:(i - 1), 1])){
      index[i] <- FALSE
    }
  }
  trunc.new.names <- new.names[index, ]
  new.pvals <- matrix(NA, nrow = length(index), ncol = ncol(pvals))
  for (i in 1:length(index)){
    if (index[i]){ 
      new.pvals[i, ] <- pvals[new.names[i, 1] == gene.names]
    }
  }
  new.pvals <- new.pvals[!is.na(new.pvals[, 1]), ]
  return(list(p.mat = new.pvals, genes = trunc.new.names[, 2]))
}
method2 <- function(pvals, gene.names, new.names){
  # Accounts for the many-to-one gene name translation issue
  # by using Stouffer's inverse normal method to combine
  # the p-values
  
  # Args:
  # pvals: A matrix of p-values.
  # gene.names: A character vector with the old gene names
  # new.names: A data frame with old gene names in the first
  # column and corresponding new gene names in the
  # second column.
  
  # Returns:
  # A list with the new matrix of p-values and the corresponding
  # gene names.

  new.names <- new.names[order(new.names[, 2]), ]
  trunc.new.names <- unique(new.names[, 2])
  matches <- rep(0, nrow(new.names))
  temp.names <- c("NOT A GENE", as.character(new.names[, 2]), "NOT A GENE")
  for (i in 2:(length(temp.names) - 2)){
    if (temp.names[i] == temp.names[i + 1] & temp.names[i] != temp.names[i - 1]){ j <- 1
      k <- 1
      while (temp.names[k] == temp.names[k + 1]){ j <- j + 1
        k <- k + 1
      }
      matches[i - 1] <- j
    }
  }
  old.names <- character(length(trunc.new.names))
  new.p.mat <- matrix(NA, ncol=ncol(pvals), nrow=length(trunc.new.names))
  j <- 1
  k <- 0
  for (i in 1:length(matches)){
    if (j > i) {k <- k + 1; next}
    if (matches[i] == 0){
      new.p.mat[i-k,] <- pvals[gene.names == new.names[j, 1]]
    }
  }
}
old.names[i-k] <- new.names[j, 1]
j <- j + 1
} else {
  originals <- new.names[new.names[j,2] == new.names[,2], 1]
  temp <- rep(FALSE, length(gene.names))
  for (l in 1:length(originals)){
    if (any(originals[l] == gene.names)){
      temp[originals[l] == gene.names] <- TRUE
    }
  }
  if (!is.matrix(pvals[temp,])){
    new.p.mat[i-k,] <- combinePvalues(pvals[temp,])
  } else {
    new.p.mat[i-k,] <- apply(pvals[temp,], MARGIN = 2,
                        FUN = combinePvalues)
  }
  old.names[i] <- paste("other", as.character(i))
  j <- j + matches[i]
}
}
old.names <- old.names[!is.na(old.names) & old.names != ""]
return(list(p.mat = new.p.mat, genes = trunc.new.names, old.names = old.names))

method4 <- function(pvals, gene.names, new.names){
  # Does what method3 does, but for a p-value matrix that has already
  # gone through method2.
  #
  # Args:
  # pvals: A matrix of p-values.
  # gene.names: A character vector with the old gene names
  # new.names: A data frame with old gene names in the first
  #       column and corresponding new gene names in the
  #       second column.
  #
  # Returns:
  # A list with the new matrix of p-values and the corresponding
  # gene names.
  temp <- new.names
new.names <- new.names[order(temp[,1]),]
gene.names <- gene.names[order(temp[, 1])]
pvals <- pvals[order(temp[, 1]), ]
if (!is.matrix(pvals)){
pvals <- matrix(pvals)
}
old.names <- c("NOT A GENE", gene.names, "NOT A GENE")
keepers <- logical()
for (i in 2:(length(old.names) - 1)){
  if (old.names[i] != old.names[i - 1] & old.names[i] != old.names[i + 1]){  
    keepers[i - 1] <- TRUE
  } else {
    keepers[i - 1] <- FALSE
  }
}
keepers[length(keepers) + 1] <- TRUE
k <- 0
j <- 1
for (i in 1:(length(keepers) - 1)){
  if (keepers[i] == TRUE){
    next
  } else {
    if (k >= i){
      next
    } else {
      j <- 1
      while (!keepers[i + j]){  
        j <- j + 1
      }
      keepers[i] <- TRUE
      k <- i + j - 1
    }
  }
}
keepers <- keepers[-length(keepers)]
return(list(p.mat = pvals[keepers, ], genes = new.names[keepers, 2]))

generateGeneSets <- function(ontology, species, ID, affy.chip){
# Generates a list of gene sets based on gene ontology.
#
# Args:
# ontology: The specific ontology within to be used. Either
# "BP", "MF", or "CC".
# species: The organism being studied. It is made up of the
# first letter of the scientific name and the last
# word of the scientific name. For example, human is
# "hsapien"
# ID: The naming system being used on the genes
# affy.chip: If ID = "affy", this is the specific chip that was
# used
#
# Returns:
# A list of character vectors. Each vector contains the names
# of the genes in a set. The names of the elements of the list
# are the Gene Ontology ID's for each gene set.
if (ID == "affy"){
  gene.set.list <- annFUN.db(ontology, affyLib = affy.chip)
  return(gene.set.list)
} else {

### define species info with species.db <- ###

if (species == "agambiae") {species.db <- "org.Ag.eg.db"} else
  if (species == "athaliana") {species.db <- "org.At.tair.db"} else
    if (species == "btaurus") {species.db <- "org.Bt.eg.db"} else
      if (species == "celegans") {species.db <- "org.Ce.eg.db"} else
        if (species == "cfamiliaris") {species.db <- "org.Cf.eg.db"} else
          if (species == "dmelanogaster") {species.db <- "org.Dm.eg.db"} else
            if (species == "drerio") {species.db <- "org.Dr.eg.db"} else
              if (species == "ecoliK12") {species.db <- "org.EcK12.eg.db"} else
                if (species == "ecoliSakai") {species.db <- "org.EcSakai.eg.db"} else
                  if (species == "ggallus") {species.db <- "org.Gg.eg.db"} else
                    if (species == "hsapiens") {species.db <- "org.Hs.eg.db"} else
                      if (species == "mmusculus") {species.db <- "org.Mm.eg.db"} else
                        if (species == "mmulatta") {species.db <- "org.Mmu.eg.db"} else
                          if (species == "pfalciparum") {species.db <- "org.Pf.plasmo.db"} else
                            if (species == "ptroglodytes") {species.db <- "org.Pt.eg.db"} else

if (species == "rnorvegicus") {species.db <- "org.Rn.eg.db"} else
if (species == "scerevisiae") {species.db <- "org.Sc.sgd.db"} else
if (species == "scoelicolor") {species.db <- "org.Sco.eg.db"} else
if (species == "sscrofa") {species.db <- "org.Ss.eg.db"} else
if (species == "tgondii") {species.db <- "org.Tgondii.eg.db"} else
if (species == "xlaevis") {species.db <- "org.Xl.eg.db"} else
stop("organism must be one of the following:
agambiae, athaliana, btaurus, celegans, cfamiliaris
dmelanogaster, drerio, ecoliK12, ecoliSakai, ggallus
hsapiens, mmusculus, mmulatta, pfalciparum, ptrogldytes
rnorvegicus, scerevisiae, scoelicolor, ssrofa
tgondii, or xlaevis")

if (any(ID == c("entrez", "genbank", "alias", "enensemble", "symbol", "genename", "unigene"))) {
  gene.set.list <- annFUN.org(ontology, mapping = species.db, ID = ID)
  return(gene.set.list)
}
}

interactiveGraph <- function(GO.Graph, color.nodes, interact = FALSE,
  legend.pos = "bottomleft", print.legend = FALSE) {
  # Creates a graph showing all given gene sets and all parent gene sets
  #
  # Args:
  # GO.Graph: A graphNEL object created by the function GOGraph. It contains
  # all of the gene sets that will be in the graph.
  # color.nodes: A character vector containing the gene sets that are of
  # interest.
  # interact: Indicates whether or not the graph should be
  # interactive.
  # legend.pos: Indicates what position the legend should be in.
  # print.legend: Indicates whether or not a legend should be printed.
  nodes <- buildNodeList(GO.Graph)
  focusnode <- names(nodes) %in% color.nodes
  names(focusnode) <- names(nodes)
  nodefill <- ifelse(focusnode, "yellow", "white")
  nAttrs <- list()
  nAttrs$fillcolor <- nodefill
nAttrs$field <- 1:length(names(nodes))
names(nAttrs$field) <- names(nodes)
pg <- plot(GO.Graph, nodeAttr = nAttrs)
x <- getNodeXY(pg)$x
y <- getNodeXY(pg)$y
ordering <- sort.list(order(-y, x))
nAttrs$field <- ordering
names(nAttrs$field) <- names(nodes)
plot(GO.Graph, nodeAttr = nAttrs)
Terms <- sapply(lookup(names(nodes)[sort.list(ordering)], "GO", "TERM"), Term)
names(Terms) <- NULL
legend <- data.frame(Terms)
if(print.legend){
  print(legend)
}
if(interact){
  repeat {
    p <- locator(n = 1)
    if (is.null(p)) break()
    pg <- plot(GO.Graph, nodeAttr = nAttrs)
x <- getNodeXY(pg)$x
y <- getNodeXY(pg)$y
distance <- abs(p$x - x) + abs(p$y - y)
idx <- which.min(distance)
  legend(legend.pos, legend=c(nAttrs$field[idx], names(focusnode[idx]),
    Term(lookup(names(focusnode)[idx],
      "GO", "TERM")[[1]]), bg = "white")
}
}
go2GeneSet <- function(name, object){
  # Creates a table showing the profile for a single gene set in each of the
  # tested contrasts.
  #
  # Args:
  # name: The name, or ID, of the desired gene set.
  # object: A mVGST object containing a final results table.
# Returns:
# A matrix with possible profiles as the row names and contrasts as the
column names. Ones in the appropriate cells showing which profile the
gene set fit for each contrast and zeroes elsewhere.
table <- object$results.table
single.table <- matrix(0, nrow = nrow(table), ncol = ncol(table), dimnames = dimnames(table))
observed <- profileCombine(object$ones.zeros)
profs <- lapply(observed, function(x) x[object$group.names == name, ])
all.profs <- matrix(NA, nrow = nrow(table), ncol = ncol(observed[[1]]))
for (i in 1:nrow(table)){
  raw.profile <- dimnames(table)[[1]][i]
  temp <- substr(raw.profile, 3, nchar(raw.profile) - 1)
  temp2 <- as.integer(strsplit(temp, ",")[[1]])
  the.profile <- matrix(temp2, nrow = 1)
  all.profs[i, ] <- the.profile
}
for (i in 1:ncol(single.table)){
  row <- apply(t(profs[[i]] == t(all.profs)), MARGIN = 1, all)
  single.table[row, i] <- 1
}
result <- list(results.table = single.table, ord.lev = object$ord.lev)
class(result) <- "mvGST"
return(single.table)

fillInList <- function(group, term, offspring, list.groups){
  # Takes a gene sets in which genes are not associated
  # with offspring terms from the GO ontology and includes all genes
  # from offspring terms.
  
  # Args:
  # group: A character vector with the genes already associated
  # with the set.
  # term: The name of the gene set.
  # offspring: A list showing the offspring sets of each gene set
  # list.groups: A list showing all genes already associated with
  # each gene set.
  if (is.na(offspring[[term]][1])){
    return(as.character(unlist(list.groups[[term]])))
  }
}
else {
    new.group <- unique(unlist(c(group, list.groups[offspring[[term]]])))
    return(new.group)
}