RMF Investigation of GO-ONMF

Robert M Flight rflight79@gmail.com

2015-12-03 11:04:56

GO ONMF

Contact with Authors

As of Dec 1, 2015, a copy of this report was sent to Hwanjo Yu as the corresponding author listed on the manuscript for feedback, either in email, or ideally as issues or pull requests on this `github` repo.

Introduction

Kim, Sael and Yu, "A Mutation Profile for Top-k Patient Search Exploiting Gene-Ontology and Orthogonal Non-negative Matrix Factorization", Bioinformatics, 2015, http://dx.doi.org/10.1093/bioinformatics/btv409

Kim, Sael and Yu recently published the above paper in which they develop a method for creating a mutation-gene-ontology profile. As part of the overall procedure, they described (Section 2.3, *Gene Function Profile*):

> To reduce the term correlation, we use only the most specific terms, i.e. the leaf node term after propagating the scores of the non-leaf terms down to the leaf node terms. This approach also resolves the problem of evaluating genes annotated with general term as the effect of the gene of function identification is spread out over several leaf node terms.

This sounds troubling to my ears, because the Gene Ontology (GO) is a *directed acyclic graph* (DAG), wherein there are specific relationships between the terms, and there is a directionality. There is also an expectation that when a gene product is annotated to a *specific* term, then that gene product is also automatically annotated to all of the *less-specific* **parent** terms in the DAG.

What the procedure quoted above sounds like it is doing is essentially **adding more specific** GO annotations to a gene product, which as far as I know is not allowed.

From The Gene Ontology Consortium, 2001:

> The pathway from a child term to its top-level parent(s) must always be true.

This is why when considering gene-GO annotations, we consider the specific term annotation, and **all** parent terms as well, because of the **true path** rule. However, the converse is not true.

This document is a record of my investigation as to whether **more specific** GO annotations are being added to gene products during the creation of the mutation profiles.

Software

Kim and Yu made an `Octave` version of their software available at https://sites.google.com/site/postechdm/research/implementation/orgos, and a tarball of the Octave version can be downloaded from https://sites.google.com/site/postechdm/research/implementation/orgos/ONMF_octave-simple.zip

This code was used to fetch a copy of the software:

```
wget https://sites.google.com/site/postechdm/research/implementation/orgos/ONMF_octave-simple.zip
mkdir ONMF_source
unzip ONMF_octave-simple.zip -d ONMF_source
```

As part of the software package, a few data files are provided. The ones that are important for our purposes include:

- ONMF_source/brca/go_(merged).csv - the index to GO term file
- ONMF_source/brca/gene_(merged).csv - the index to gene file
- exp_onmf_brca.m - the file that runs the *brca* analysis
- network_gene2go(merged).csv - the gene 2 GO annotations

Running Code

After modifying *exp_onmf_brca.m* to have the correct library path, and then adding some code that will save **gene - GO** indices before and after the optimization step (see below for the actual code run), I `cd` to the `ONMF_source` directory, and then start `Octave`, and then run:

```
cd brca
run exp_onmf_brca_mod.m
```

As this runs, it generates *gene2go.mat*, which is the propagated scores of gene to GO associations, and then saves indices (see file):

```
%% Orthogonal non-negative matrix factorization/Gene-Ontology-based stratification
library_path = '/home/rmflight/Projects/personal/onmf/ONMF_source/ONMF_octave';
addpath(genpath(library_path))
%% Convert somatic mutation cancer data to my data
patient2gene = load('network_patient2gene(merged).csv');
gene2go = load('network_gene2go(merged).csv');
go2go = load('network_allgo2allgo(merged).csv');
patient2gene = getSparse(patient2gene);
gene2go = getSparse(gene2go);
go2go = getSparse(go2go);
patient2gene(find(patient2gene)) = 1;
gene2go(find(gene2go)) = 1;
org_notzero = find(gene2go(1, :));
save -ascii org_notzero org_notzero;
[baseSMData, patient2gene, gene2go_new, go2go_new] = TuneData(patient2gene, gene2go, go2go);
isequal(go2go_new, go2go)
prop_notzero = find(gene2go_new(1, :));
save -ascii prop_notzero prop_notzero;
```

Prerequisites

To continue this analysis, you will need some other `R` packages installed.

```r
install.packages(c("readr", "dplyr"))
library(BiocInstaller)
biocLite(c("GO.db", "graph", "org.Hs.eg.db"))
```

Load Packages

```
library(readr)
library(dplyr)
library(GO.db)
library(org.Hs.eg.db)
```

Check Actual Input Data

Lets check that the actual data being used as input looks right. **Note** that 1 gets added to the map indices so that we easily work with the data from the matrices in `Octave`. The input data includes the `go_map`, that maps GO terms to indices, `gene_map` mapping gene symbols to indices, `gene2go` mapping the GO annotations to genes, and `go2go`, giving the parent-child relationships of the GO terms.

```
go_map <- read_csv("ONMF_source/brca/go_(merged).csv",
                   col_names = c("loc", "GO"))
go_map$loc <- go_map$loc + 1
gene_map <- read_csv("ONMF_source/brca/gene_(merged).csv",
                     col_names = c("loc", "GENE"))
gene_map$loc <- gene_map$loc + 1
gene2go <- read_csv("ONMF_source/brca/network_gene2go(merged).csv",
                    col_names = c("gene", "score", "GO"))
gene2go$gene <- gene2go$gene + 1
gene2go$GO <- gene2go$GO + 1
go2go <- read_csv("ONMF_source/brca/network_allgo2allgo(merged).csv",
                  col_names = c("GO1", "distance", "GO2"))
go2go$GO1 <- go2go$GO1 + 1
go2go$GO2 <- go2go$GO2 + 1
```

And add the actual gene symbols and GO IDs to the `gene2go` to enable easy comparisons with any other data source we can find.

```
gene2go$geneid <- unlist(gene_map[match(gene2go$gene, gene_map$loc), "GENE"],
                         use.names = FALSE)
gene2go$goid <- unlist(go_map[match(gene2go$GO, go_map$loc), "GO"],
                       use.names = FALSE)
```

And then query `org.Hs.eg.db` for the gene-annotations as well.

```
hs_gene2go <- select(org.Hs.eg.db, keys = unique(gene2go$geneid),
                     columns = c("GO", "ONTOLOGY"), keytype = "SYMBOL")
```

```
## 'select()' returned 1:many mapping between keys and columns
```

```
hs_gene2go <- dplyr::filter(hs_gene2go, ONTOLOGY == "BP", EVIDENCE != "IEA")
```

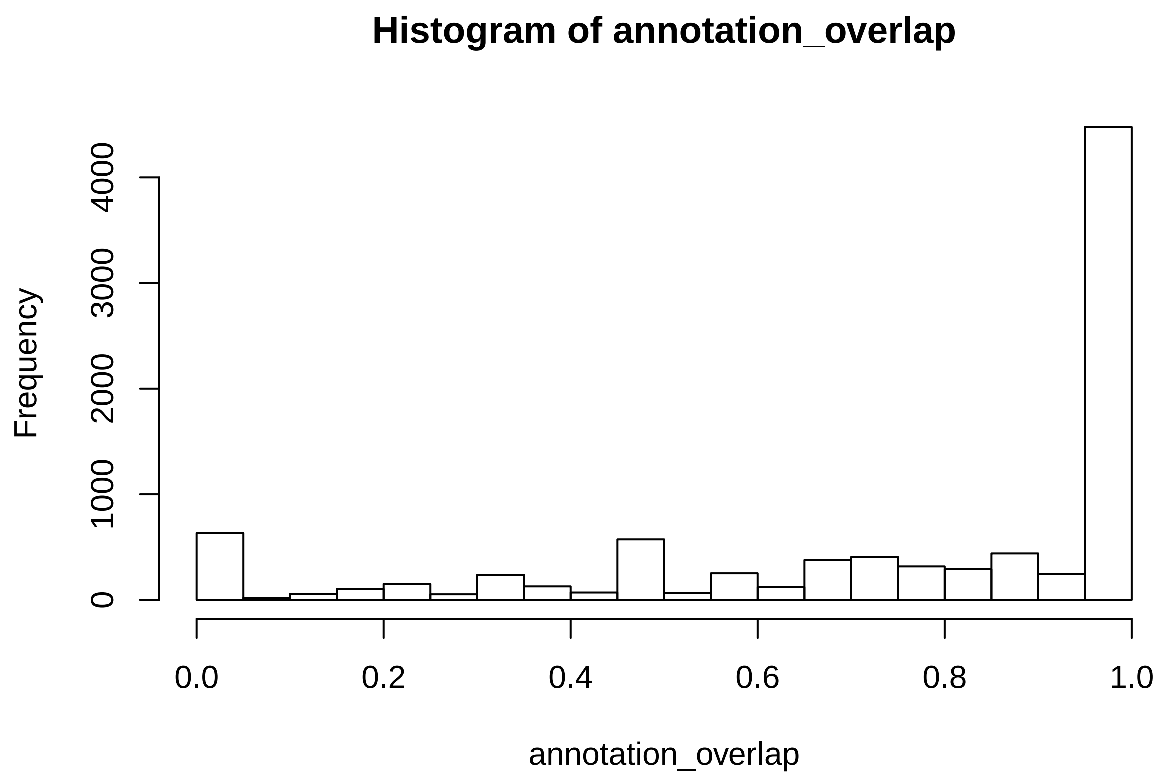
Compare Gene-GO Annotations

And compare the annotations for the genes between the data from Kim and from the Bioconductor database.

```r
annotation_overlap <- vapply(unique(hs_gene2go$SYMBOL), function(in_loc){
  hs_data <- dplyr::filter(hs_gene2go, SYMBOL == in_loc) %>%
    dplyr::select(., GO) %>% unlist()
  onmf_data <- dplyr::filter(gene2go, geneid == in_loc) %>%
    dplyr::select(., goid) %>% unlist()
  sum(hs_data %in% onmf_data) / length(hs_data)
}, numeric(1))
```

And check the distribution of overlap:

```r
hist(annotation_overlap)
```



That is great, over 4000 of the genes have what appear to be very similar terms!

Compare GO-GO Relationships

Lets also check the **GO2GO** data.

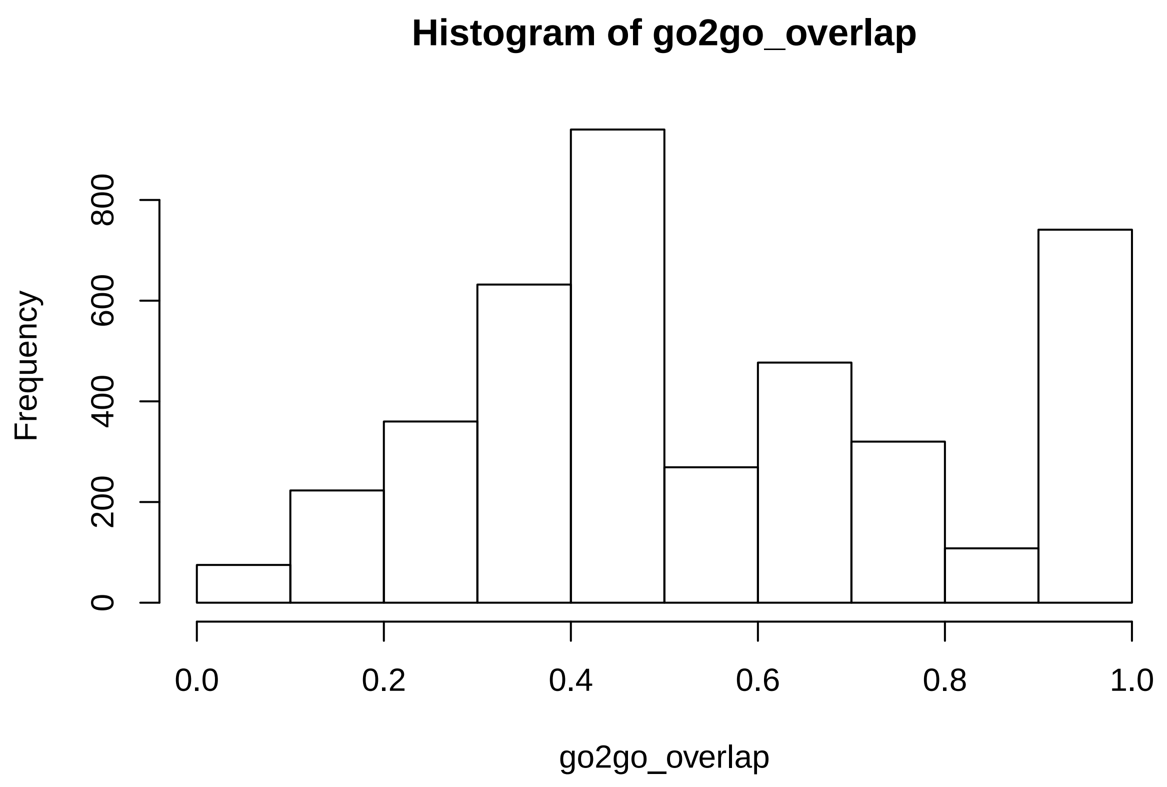
```r
go_notself <- dplyr::filter(go2go, distance != 1)
go_notself$ID1 <- unlist(go_map[match(go_notself$GO1, go_map$loc), "GO"], use.names = FALSE)
go_notself$ID2 <- unlist(go_map[match(go_notself$GO2, go_map$loc), "GO"], use.names = FALSE)

onmf_pc <- split(go_notself$ID2, go_notself$ID1)
go_pc <- mget(names(onmf_pc), GOBPCHILDREN, ifnotfound = NA)
```

```r
go2go_overlap <- vapply(names(onmf_pc), function(in_loc){
  go_data <- go_pc[[in_loc]]
  onmf_data <- onmf_pc[[in_loc]]
  sum(go_data %in% onmf_data) / length(go_data)
}, numeric(1))
```

And plot it:

```r
hist(go2go_overlap)
```



Work With Single Gene (A2M)

Now lets work with a single gene, and see what is going on. We'll keep it simple and play with the first one in the list, *A2M*.

Grab Starting and Propagated Terms

```r
onmf_a2m_start <- dplyr::filter(gene2go, geneid == "A2M") %>%
  dplyr::select(goid) %>% unlist(., use.names = FALSE) %>% sort()

onmf_a2m_indices_org <- scan(file = "ONMF_source/brca/org_notzero", what = numeric(),
                         sep = " ")
onmf_a2m_indices_org <- onmf_a2m_indices_org[!is.na(onmf_a2m_indices_org)]
onmf_a2m_matrix_org <- dplyr::filter(go_map, loc %in% onmf_a2m_indices_org) %>%
  dplyr::select(GO) %>% unlist(., use.names = FALSE) %>% sort()

onmf_a2m_indices_prop <- scan(file = "ONMF_source/brca/prop_notzero",
                              what = numeric(), sep = " ")
onmf_a2m_indices_prop <- onmf_a2m_indices_prop[!is.na(onmf_a2m_indices_prop)]
onmf_a2m_matrix_prop <- dplyr::filter(go_map, loc %in% onmf_a2m_indices_prop) %>%
  dplyr::select(GO) %>% unlist(., use.names = FALSE) %>% sort()
```

Compare To Database

Now, how many of the terms in the starting gene annotation actually in the database annotation?

```
hs_a2m_go <- dplyr::filter(hs_gene2go, SYMBOL == "A2M") %>%
  dplyr::select(GO) %>% unlist(., use.names = FALSE) %>% unique()
sum(onmf_a2m_matrix_org %in% hs_a2m_go) / length(onmf_a2m_matrix_org)
```

```
## [1] 0.8
```

OK, seems we have most of them. Good.

Check if Propagated Are Children of Original

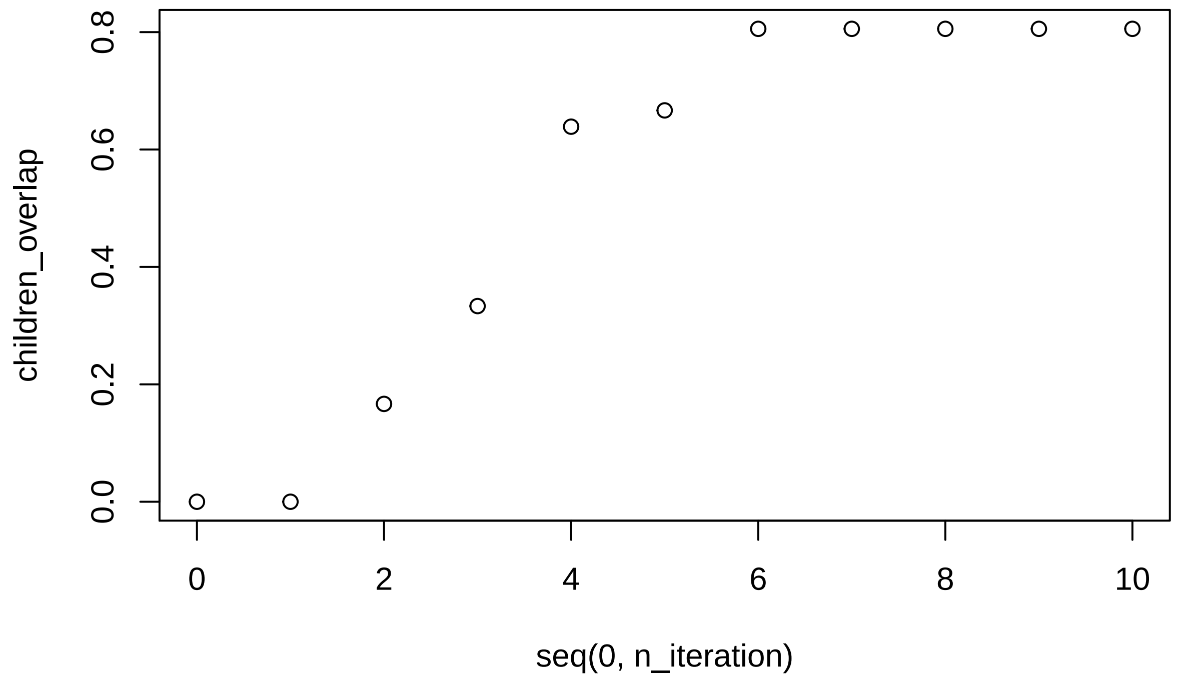
Next, are the propagated terms actually children of the starting terms? We will use multiple iterations of calls to `GOBPCHILDREN` and see if we get progressively more of the terms.

```
n_iteration <- 10
query_list <- onmf_a2m_matrix_org
children_overlap <- numeric(n_iteration + 1)

for (i_iter in seq(1, n_iteration)) {
  children_overlap[i_iter + 1] <-
    sum(onmf_a2m_matrix_prop %in% query_list) / length(onmf_a2m_indices_prop)

  query_child <- mget(query_list, GOBPCHILDREN, ifnotfound = NA) %>%
    unlist(., use.names = FALSE) %>% unique()
  query_child <- query_child[!is.na(query_child)]
  query_list <- unique(c(query_list,
                       query_child))
}
```

```
plot(seq(0, n_iteration), children_overlap)
```



FOUND THEM. The propagated terms have **NO** overlap with the original terms (see 0's at the start of the graph), and are children of the original terms. Just to be sure, I will also compare the propagated terms to the `GOALL` annotation, which would be all of the GO terms that can **technically** be considered as **annotated** to the gene (see *true path rule* from the Introduction), in that all **terms** and their **parent** terms are **annotated** to a gene product.

```
hs_a2m_all <- select(org.Hs.eg.db, keys = "A2M", columns = "GOALL",
                     keytype = "SYMBOL")
```

```
## 'select()' returned 1:many mapping between keys and columns
```

```
head(hs_a2m_all)
```

```
##   SYMBOL      GOALL EVIDENCEALL ONTOLOGYALL
## 1    A2M GO:0001775         TAS          BP
## 2    A2M GO:0001867         IDA          BP
## 3    A2M GO:0001868         IDA          BP
## 4    A2M GO:0001869         IDA          BP
## 5    A2M GO:0002020         IPI          MF
## 6    A2M GO:0002252         IDA          BP
```

```
sum(hs_a2m_all$GOALL %in% onmf_a2m_matrix_org) # the starting annotation
```

```
## [1] 12
```

```
sum(hs_a2m_all$GOALL %in% onmf_a2m_matrix_prop) # propogated annotation
```

```
## [1] 0
```

From this, none of the **propogated** terms can be properly considered as annotations to **A2M**. Effectively, the code has generated a **new set** of GO annotations to **A2M**. I am sure this is the situation for all of the genes.

Back to Parent

On Dec 3, Sael replied to my email (see correspondence), and the crux of his response amounts to propagating back up to parents to get significant / interpretable GO terms (also in the supplemental materials, I have added a PDF of the supplementary materials. This seems rather odd to end up doing two sets of propagation, one down to leaf terms to make the ONMF better (I assume), and then another up to parent terms to get more interpretable / significant results.

I am also concerned that propagating back up will actually result in more terms than were originally annotated to a gene of interest. This is possible because of the DAG nature of the GO and multi-parent / multi-child relationships.

Lets actually check if a round of propagating down to children and then back up to parents results in the same or different set of parent terms than was started with. I will use the same set of terms that were annotated to *A2M* above.

I am going to use a set number of iterations of propagation (10) both down and up to demonstrate the possible problem.

```
children_down <- list(11)
children_down[[1]] <- hs_a2m_go

for (i_iter in seq(2, 11)) {
  tmp_down <- mget(children_down[[i_iter - 1]], GOBPCHILDREN, ifnotfound = NA)
  na_entries <- sapply(tmp_down, function(x){sum(is.na(x)) == 1})
  all_children <- unique(c(unlist(tmp_down, use.names = FALSE), names(na_entries)[na_entries]))
  children_down[[i_iter]] <- all_children[!is.na(all_children)]
}

len_children <- vapply(children_down, length, numeric(1))
len_children
```

```
##  [1]  10  50 100  95  74  69  69  69  69  69  69
```

From this, it looks like all of the children are captured by step *5* of the propagation. So it will be 5 steps of propagation back up to parents.

```
parents_up <- list(6)
parents_up[[1]] <- children_down[[6]]

for (i_iter in seq(2, 6)) {
  tmp_up <- mget(parents_up[[i_iter - 1]], GOBPPARENTS, ifnotfound = NA)
  na_entries <- vapply(tmp_up, function(x){sum(is.na(x)) == 1}, logical(1))
  all_parents <- unique(c(unlist(tmp_up, use.names = FALSE), names(na_entries)[na_entries]))
  parents_up[[i_iter]] <- all_parents[!is.na(all_parents)]
}
len_parents <- vapply(parents_up, length, numeric(1))
len_parents
```

```
## [1]  69 115 172 228 248 221
```

As I suspected, going from the propagated children back up to parents (even a limited number of iterations) results in a huge number of terms, many more than was in the initial list. Granted, this did not seem to happen to such a degree in the manuscript, because they were able to get significant enrichment of parent terms, so only a limited subset had to have been returned by the iterative matrix multiplication.

Concluding Thoughts

However, there is nothing in the manuscript that demonstrates that the initial propagation to child terms is necessary for performance of the method, or that using a GOSlim does not work, or that the same / similar parent terms are returned by propagating back up. I am surprised that the reviewers did not ask for any of these things given the nature of the data.

I would also like to commend the authors for making the implementation available in an open-source language (`Octave`), and for replying to my email questioning the methodology. More of the published methodologies need to provide at least a reference implementation in an open-source programming language.

Commenting

I have left a comment on the Pubmed Commons Page for this manuscript pointing to this analysis.