A gentle start to using the PrixFixe R/Bioconductor package

Murat Taşan - updated by Adrian Pasculescu

September 22, 2021

1 R/Bioconductor

This guide assumes the reader has some basic familiarity with using **R** and has installed the core **Bioconductor** packages. Information on installing both of these can be found on their respective websites:

`http://r-project.org`
`http://bioconductor.org`

Since this package is not yet hosted by CRAN or Bioconductor, its dependencies will also need to be installed prior to its own installation. These required packages are: `IRanges` (included by default as part of Bioconductor), `plyr`, `stringr`, `igraph`, `httr`, and `jsonlite`. Even if these packages are installed, it's worth upgrading them to the most recent versions in case any are outdated. Most R users will be familiar with installing/upgrading packages, but just in case, one way to install these packages from within R is to type:

```
> source("http://www.bioconductor.org/biocLite.R")
> biocLite("IRanges")
> install.packages(c("plyr", "stringr", "igraph", "httr", "jsonlite"))
```

Additional help on installing these packages can be found on the individual package pages hosted on CRAN or Bioconductor.

1.1 Internet access and remote data

This package installs the base software required to set up a "prix fixe" problem (i.e. Selecting causal genes from genome-wide association studies via functionally coherent subnetworks). It then efficiently solves it using a genetic algorithm optimizer. To create a well-defined prix fixe instance, all that's needed is a set of gene sets (preferably disjoint), and a graph (a.k.a. "network") with edges connecting these genes.

The accompanying prix fixe manuscript (eference section [9]) uses data from (i) the International HapMap Project and (ii) published functional-association network papers to help (i) establish the gene sets and (ii) retrieve the network edges. These two data sources are quite large and would make this package swell to many GBs in size. So, two functions are provided in this package that access a public web service querying pre-built databases. These two functions (`getLDRegionsFromSNPs.WS` and `getCandidateEdges.WS`) are described further below, but it is noted here that they both require the user to have a functioning internet connection (and the `.WS` suffix is used to remind the user of the web service dependency).

Also, our prix fixe web service is in continous development and thus access methods should not be considered finalized. Furthermore, since the service is under development, calls should **NOT** be looped, as they'd likely cripple our public servers (so please be kind)!

2 Installation and loading

The PrixFixe R package can be installed from source like any other R package. One such method (from within R) is:

```
> install.packages("/the/path/to/PrixFixe_0.1-3.tar.gz", repos = NULL)
```

If any dependencies are missing, the user will be notified, and they can then be install as needed (see above).

The package can then be loaded like so:

```
>   library(PrixFixe, quietly = TRUE)
```

3 Getting function details

Most of the functions described here have corresponding R help/man pages that describe the options and parameters in greater detail. Within this document, the default settings are used in almost every case to keep this introduction, well, introductory.

To access help pages for functions listed in this vignette, use the following commands:

```
help("getLDRegionsFromSNPs.WS") or ??getLDRegionsFromSNPs.WS
help("getGeneRegionsTable")
help("getCandidateEdges.WS")
help("GPF$PF$new")
help("GPF$GA$run")
help("GPF$GA$scoreVertices")
```

4 Mapping SNPs to LD regions

To convert a collection of dbSNP IDs to candidate gene sets, a web service call may be used. Here, we'll use a list of example dbSNP IDs that are provided with the package (`example_dbsnp_ids`). This example list of SNPs is quite small and is not intended to reflect typical use cases where perhaps hundreds of loci are contributing to a trait, but is rather kept small for the sake of brevity and simplicity within this vignette.

```
>   str(example_dbsnp_ids)

 chr [1:19] "rs498872" "rs2736100" "rs11979158" "rs4295627" "rs6010620" "rs2252586" "rs2157719" "rs49777

>   ld_regions_query_result <- getLDRegionsFromSNPs.WS(example_dbsnp_ids)

.......................
```

The web service returns information about any bad IDs or those which don't appear in either the current 'Ensembl' or 'NCBI' public resources.

```
>   ## these look funny:
>   ld_regions_query_result$invalid_dbsnp_ids

[1] "foo"

>   ## these look all right, but don't appear in the dbSNP database:
>   ld_regions_query_result$unfound_dbsnp_ids

[1] "rs0"

>   ## these are known SNPs, but don't appear in the HapMap database:
>   ld_regions_query_result$unmapped_dbsnps

 [1] "rs2252586"   "rs79335915"  "rs2538069"   "rs79093312"  "rs59010780"  "rs2110263"   "rs6975304"   "
 [9] "rs7793297"   "rs147653122" "rs6946428"   "rs2538067"   "rs77148056"  "rs6945082"   "rs6593195"   "
[17] "rs76843561"  "rs55662545"  "rs111545791" "rs150568098" "rs77767664"  "rs58626582"  "rs11766623"  "
[25] "rs2709275"   "rs73410447"  "rs111517756" "rs7793272"   "rs77799804"  "rs2709274"   "rs2709273"   "
[33] "rs73410456"  "rs564956477" "rs2330759"   "rs11765408"  "rs2538068"   "rs2538066"   "rs7449190"
```

The LD regions themselves are usually best viewed as a single table with the candidate genes.

```
>   candidate_gene_table <- getGeneRegionsTable(ld_regions_query_result$ld_regions)
>   head(candidate_gene_table)
```

```
                                    id     symbol  region_name dbSNPs_count
NCBI_Gene:23187         NCBI_Gene:23187     PHLDB1  chr:11q23.3           17
NCBI_Gene:7015           NCBI_Gene:7015       TERT  chr:5p15.33           22
NCBI_Gene:1956           NCBI_Gene:1956       EGFR   chr:7p11.2           45
NCBI_Gene:137196       NCBI_Gene:137196     CCDC26  chr:8q24.21           32
NCBI_Gene:51750         NCBI_Gene:51750      RTEL1 chr:20q13.33           64
NCBI_Gene:100048912 NCBI_Gene:100048912 CDKN2B-AS1   chr:9p21.3          107
```

```
>   str(candidate_gene_table)
```

```
'data.frame':        12 obs. of  4 variables:
 $ id          : chr  "NCBI_Gene:23187" "NCBI_Gene:7015" "NCBI_Gene:1956" "NCBI_Gene:137196" ...
 $ symbol      : chr  "PHLDB1" "TERT" "EGFR" "CCDC26" ...
 $ region_name : chr  "chr:11q23.3" "chr:5p15.33" "chr:7p11.2" "chr:8q24.21" ...
 $ dbSNPs_count: int  17 22 45 32 64 107 47 11 11 24 ...
```

5 Edges between genes

The prix fixe code can use any collection of edges between the candidate genes. All matching on genes should be done on unique identifiers, and here we use NCBI Gene IDs for that purpose. (If using yeast, for example, SGD IDs would be a natural choice.)

Edges are provided to the prix fixe methods as a simple two-column `data.frame` with the gene IDs. To acquire the human edges used in the accompanying manuscript, a web service call is made (using NCBI Gene IDs):

```
>   candidate_edges <- getCandidateEdges.WS(candidate_gene_table$id)
>   head(candidate_edges)
```

```
       NCBI_Gene1 NCBI_Gene2
156767       1029       1030
157212       1029       1956
158176       1029      50861
158272       1029      51750
159048       1029       7015
159418       1029      84619
```

```
>   str(candidate_edges)
```

```
'data.frame':        17 obs. of  2 variables:
 $ NCBI_Gene1: int  1029 1029 1029 1029 1029 1029 1030 1956 51750 7015 ...
 $ NCBI_Gene2: int  1030 1956 50861 51750 7015 84619 1956 84619 7015 8771 ...
```

This web service call only retrieves those edges connecting candidate genes to other candidate genes. Working with this (very much) reduced set of edges is much faster than working with the full network. These edges will be further reduced to only those satisfying the prix fixe n-partite constraint (as described in the manuscript).

6 Creating a prix fixe instance

Once a set of candidate gene sets has been acquired, and edges between those candidate genes have been supplied, a prix fixe problem instance can be created. The candidate genes should be partitioned into their appropriate loci, and thus specified as an R `list` of `character` vectors:

```
>   candidate_gene_sets <- tapply(candidate_gene_table$id, candidate_gene_table$region_name, I)
>   candidate_gene_sets

$`chr:11q23.3`
[1] "NCBI_Gene:23187"

$`chr:20q13.33`
[1] "NCBI_Gene:51750" "NCBI_Gene:84619" "NCBI_Gene:8771"  "NCBI_Gene:10139" "NCBI_Gene:50861"

$`chr:5p15.33`
[1] "NCBI_Gene:7015"

$`chr:7p11.2`
[1] "NCBI_Gene:1956"

$`chr:8q24.21`
[1] "NCBI_Gene:137196"

$`chr:9p21.3`
[1] "NCBI_Gene:100048912" "NCBI_Gene:1030"      "NCBI_Gene:1029"

>   prix_fixe <- GPF$PF$new(candidate_gene_sets, candidate_edges, VERBOSE = TRUE)

[1] "start: constructing a PrixFixe problem instance."
[1] "0 sanitized FAN vertices"
[1] "candidate gene set sizes = 1, 5, 1, 1, 1, 3"
[1] "15 candidate combinations (over 6 gene sets)."
[1] "8 simplified combinations (over 4 gene sets)."
[1] "done: constructing a PrixFixe problem instance."
```

As seen above, the function is stored in a list called `GPF`... this is just a convenience object that holds many utility functions. You will see it used again below when running the genetic algorithm and scoring genes.

The prix fixe problem instance doesn't score genes itself, but rather just sets up the problem with the appropriate constraints. For instance, An n-partite graph is created (using the `igraph` R package):

```
>   prix_fixe$G

IGRAPH f608061 UN-- 13 12 --
+ attr: name (v/c)
+ edges from f608061 (vertex names):
 [1] NCBI_Gene:1956 --NCBI_Gene:1029 NCBI_Gene:1956 --NCBI_Gene:1030 NCBI_Gene:50861--NCBI_Gene:1029
 [4] NCBI_Gene:51750--NCBI_Gene:1029 NCBI_Gene:51750--NCBI_Gene:1030 NCBI_Gene:7015 --NCBI_Gene:1029
 [7] NCBI_Gene:7015 --NCBI_Gene:1956 NCBI_Gene:51750--NCBI_Gene:7015 NCBI_Gene:50861--NCBI_Gene:7015
[10] NCBI_Gene:84619--NCBI_Gene:1029 NCBI_Gene:84619--NCBI_Gene:1956 NCBI_Gene:8771 --NCBI_Gene:7015
```

PrixFixe objects aren't really all that useful themselves, but rather are to be handed-over to the dense prix fixe subgraph searchers, as described in the next section.

Any gene identifiers and networks (for any species) can be used to create a prix fixe problem instance. The only requirement is that the gene identifiers are consistent between both the candidate gene sets and edge arguments.

7 Scoring genes

Once a prix fixe instance has been created, it can be passed to the genetic algorithm optimizer, which will attempt to find dense prix fixe subnetworks:

```
>   GA_run_result <- GPF$GA$run(prix_fixe, VERBOSE = TRUE)

[1] "generation 0:"
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  3.000   3.000   4.000   3.989   5.000   5.000
[1] "generation 1:"
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  3.000   4.000   5.000   4.483   5.000   5.000
[1] "relative stopping-statistic improvement over previous generation = 0.123884"
[1] "generation 2:"
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  3.000   5.000   5.000   4.752   5.000   5.000
[1] "relative stopping-statistic improvement over previous generation = 0.0599545"
[1] "generation 3:"
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  3.000   5.000   5.000   4.873   5.000   5.000
[1] "relative stopping-statistic improvement over previous generation = 0.0254198"
[1] "generation 4:"
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  3.000   5.000   5.000   4.923   5.000   5.000
[1] "relative stopping-statistic improvement over previous generation = 0.0102196"
[1] "generation 5:"
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  3.000   5.000   5.000   4.944   5.000   5.000
[1] "relative stopping-statistic improvement over previous generation = 0.00434712"
```

Upon termination of the algorithm, genes can now be scored:

```
>   score_table <- GPF$GA$scoreVertices(GA_run_result, VERBOSE = TRUE)

[1] "scoring vertex set chr:20q13.33."
[1] "scoring vertex set chr:5p15.33."
[1] "scoring vertex set chr:7p11.2."
[1] "scoring vertex set chr:9p21.3."
```

Genes not having any connections in the n-partite graph are unscored, and so we can merge the scores back with the original candidate gene table and give a score of zero to those disconnected genes:

```
>   candidate_gene_table$score <-
+   score_table$scaled_score[match(candidate_gene_table$id, score_table$vertex_name)]
>   candidate_gene_table$score[is.na(candidate_gene_table$score)] <- 0
```

We can then sort this table by locus and/or score to see the top-scoring candidate genes within each locus:

```
>   ## we use the plyr package to manipulate data frames
>   library(plyr)
>   candidate_gene_table <- plyr::arrange(candidate_gene_table, region_name, -score)
>   candidate_gene_table[c("id", "symbol", "region_name", "score")]

                   id     symbol  region_name     score
1      NCBI_Gene:23187     PHLDB1  chr:11q23.3 0.0000000
2      NCBI_Gene:51750      RTEL1 chr:20q13.33 0.6666667
3      NCBI_Gene:84619      ZGPAT chr:20q13.33 0.6596000
4      NCBI_Gene:50861      STMN3 chr:20q13.33 0.6596000
5       NCBI_Gene:8771   TNFRSF6B chr:20q13.33 0.3333333
6      NCBI_Gene:10139     ARFRP1 chr:20q13.33 0.0000000
7       NCBI_Gene:7015       TERT  chr:5p15.33 0.8984667
```

```
8       NCBI_Gene:1956       EGFR   chr:7p11.2 0.7611333
9     NCBI_Gene:137196     CCDC26  chr:8q24.21 0.0000000
10      NCBI_Gene:1029     CDKN2A   chr:9p21.3 0.9917333
11      NCBI_Gene:1030     CDKN2B   chr:9p21.3 0.4524000
12 NCBI_Gene:100048912 CDKN2B-AS1   chr:9p21.3 0.0000000
```

```
>         ## reorder candidate gene table by score if needed
>         candidate_gene_table <- candidate_gene_table[order(candidate_gene_table$score, decreasing=TRU
```

In the cases of relative small candidate gene table we can visualy represent it as a graph using the igraph package. For convenience we will first convert the gene ids into gene symbols.

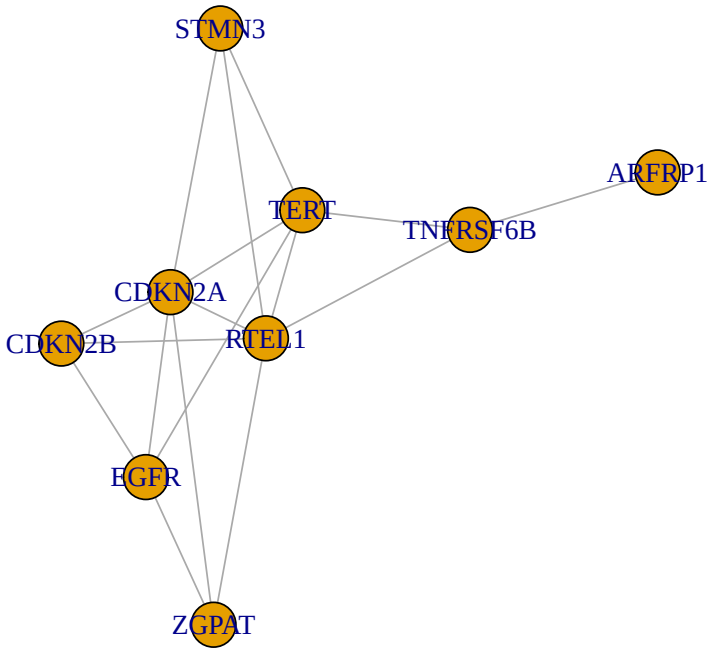
```
>         ## Convert candidate_gene_table to a new table that uses gene symbols
>         rownames(candidate_gene_table) <- candidate_gene_table$id
>         candidate_edges_genes <- cbind(
+             candidate_gene_table[paste('NCBI_Gene:',candidate_edges[,1],sep=''),'symbol']
+                  ,candidate_gene_table[paste('NCBI_Gene:',candidate_edges[,2],sep=''),'symbol'])
>         candidate_edges_genes
```

```
      [,1]     [,2]
 [1,] "CDKN2A" "CDKN2B"
 [2,] "CDKN2A" "EGFR"
 [3,] "CDKN2A" "STMN3"
 [4,] "CDKN2A" "RTEL1"
 [5,] "CDKN2A" "TERT"
 [6,] "CDKN2A" "ZGPAT"
 [7,] "CDKN2B" "EGFR"
 [8,] "EGFR"   "ZGPAT"
 [9,] "RTEL1"  "TERT"
[10,] "TERT"   "TNFRSF6B"
[11,] "ARFRP1" "TNFRSF6B"
[12,] "CDKN2B" "RTEL1"
[13,] "EGFR"   "TERT"
[14,] "STMN3"  "RTEL1"
[15,] "STMN3"  "TERT"
[16,] "RTEL1"  "ZGPAT"
[17,] "RTEL1"  "TNFRSF6B"
```

```
>         # create the graph from the above table that defines the edges of the graph
>         library('igraph')
>   g <- graph_from_edgelist(as.matrix(candidate_edges_genes), directed=FALSE)
>
```

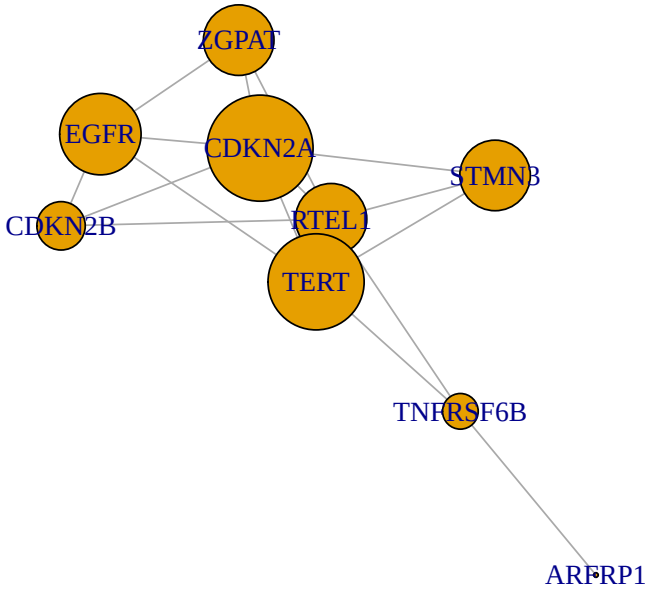
Plot the graph from the Candidate genes

```
>   plot(g)
```



Add the PrixFixe score to the candidate genes graph and show the vertex size proportional to the score.

```
>                 plot(g, vertex.size=candidate_gene_table$score[
+                     match(attr(V(g),'name'),candidate_gene_table$symbol )]*40)
```



8 Next up in development

- Additional plotting functions to visualize the results.
- Improved testing to handle bad-input cases and provide more meaningful error messages.
- Parallelization support for genetic algorithm execution and scoring.

9 References

- M Tasan, G Musso, T Hao, M Vidal, CA MacRae and FP Roth. Selecting causal genes from genome-wide association studies via functionally coherent subnetworks. Nature Methods 12(2):154-159 (2015)