graphBAM and MultiGraph classes.

N. Gopalakrishnan

October 26, 2021

1 graphBAM class

1.1 Introduction

The *graphBAM* class has been created as a more efficient replacement for the *graphAM* class in the *graph* package. The adjacency matrix in the *graphBAM* class is represented as a bit array using a `raw` vector. This significantly reduces the memory occupied by graphs having a large number of nodes. The bit vector representation also provides advantages in terms of performing operations such as intersection or union of graphs.

We first load the *graph* package which provides the class definition and methods for the *graphBAM* class.

```
> library(graph)
```

One of the arguments `df` to the *graphBAM* constructor is a `data.frame` containing three columns: "from","to" and "weight", each row in the `data.frame` representing an edge in the graph. The `from` and `to` columns can be character vectors or factors, while the `weight` column must be a numeric vector. The argument `nodes` are calculated from the unique names in the `from` and `to` columns of the `data.frame`. The argument `edgeMode` should be a character vector, either "directed" or "undirected" indicating whether the graph represented should be directed or undirected respectively.

1.2 A simple graph represented using graphBAM class

We proceed to represent a simple graph using the *graphBAM* class. Our example is a directed graph representing airlines flying between different cities. In this example, cities represent the nodes of the graph and each edge represents a flight from an originating city (`from`) to the destination city (`to`). The weight represents the fare for flying between the `from` and `to` cities.

```
> df <- data.frame(from = c("SEA", "SFO", "SEA", "LAX", "SEA"),
+                  to = c("SFO", "LAX", "LAX", "SEA", "DEN"),
+                 weight = c( 90, 96, 124, 115, 259),
+                 stringsAsFactors = TRUE)
> g <- graphBAM(df, edgemode = "directed")
> g

A graphBAM graph with directed edges
Number of Nodes = 4
Number of Edges = 5
```

The cities (nodes) included in our *graph* object as well as the stored fares(`weight`) can be obtained using the `nodes` and `edgeWeights` methods respectively.

```
> nodes(g)

[1] "DEN" "LAX" "SEA" "SFO"

> edgeWeights(g, index = c("SEA", "LAX"))

$SEA
DEN LAX SFO
259 124  90

$LAX
SEA
115
```

Additional nodes or edges can be added to our graph using the `addNode` and `addEdge` methods. For our example, we first add a new city "IAH" to our graph. We then add a flight connection between "DEN" and "IAH" having a fare of $120.

```
> g <- addNode("IAH", g)
> g <- addEdge(from = "DEN", to = "IAH", graph = g, weight = 120)
> g

A graphBAM graph with directed edges
Number of Nodes = 5
Number of Edges = 6
```

Similarly edges and nodes can be removed from the graph using the `removeNode` and `removeEdge` methods respectively. We proceed to remove the flight connection from "DEN" to "IAH" and subsequently the node "IAH".

```
> g <- removeEdge(from ="DEN", to = "IAH", g)
> g <- removeNode(node = "IAH", g)
> g

A graphBAM graph with directed edges
Number of Nodes = 4
Number of Edges = 5
```

We can create a subgraph with only the cities "DEN", "LAX" and "SEA" using the `subGraph` method.

```
> g <- subGraph(snodes = c("DEN","LAX", "SEA"), g)
> g

A graphBAM graph with directed edges
Number of Nodes = 3
Number of Edges = 3
```

We can extract the `from-to` relationships for our graph using the `extractFromTo` method.

```
> extractFromTo(g)

  from  to weight
1  SEA DEN    259
2  SEA LAX    124
3  LAX SEA    115
```

1.3 Mice gene interaction data for brain tissue (SAGE data)

The C57BL/6J and C3H/HeJ mouse strains exhibit different cardiovascular and metabolic phenotypes on the hyperlipidemic apolipoprotein E (Apoe) null background. The interaction data for the genes from adipose, brain, liver and muscle tissue samples from male and female mice were studied. This interaction data for the various genes is included in the *graph* package as a list of `data.frame`s containing information for `from-gene`, `to-gene` and the strength of interaction `weight` for each of the tissues studied.

We proceed to load the data for male and female mice.

```
> data("esetsFemale")
> data("esetsMale")
```

We are interested in studying the interaction data for the genes in the brain tissue for male and female mice and hence proceed to represent this data as directed graphs using *graphBAM* objects for male and female mice.

```
> dfMale <- esetsMale[["brain"]]
> dfFemale <- esetsFemale[["brain"]]
> head(dfMale)
```

```
         from          to weight
1 10024402938 10024393150  0.835
2 10024415240 10024393156  0.667
3 10024403128 10024393162  0.312
4 10024409968 10024393162  0.482
5 10024393260 10024393163  0.997
6 10024394731 10024393165  0.714
```

```
> male <- graphBAM(dfMale, edgemode = "directed")
> female <- graphBAM(dfFemale, edgemode = "directed")
```

We are interested in pathways that are common to both male and female graphs for the brain tissue and hence proceed to perform a graph intersection operation using the `graphIntersect` method. Since edges can have different values of the weight attribute, we would like the result to have the sum of the weight attribute in the male and female graphs. We pass in `sum` as the function for handling weights to the `edgeFun` argument. The `edgeFun` argument should be passed a list of named functions corresponding to the edge attributes to be handled during the intersection process.

```
> intrsct <- graphIntersect(male, female, edgeFun=list(weight =  sum))
> intrsct
```

```
A graphBAM graph with directed edges
Number of Nodes = 2117
Number of Edges = 473
```

If node attributes were present in the `graphBAM` objects, a list of named function could be passed as input to the `graphIntersect` method for handling them during the intersection process.

We proceed to remove edges from the `graphBAM` result we just calculated with a weight attribute less than a numeric value of 0.8 using the `removeEdgesByWeight` method.

```
> resWt <- removeEdgesByWeight(intrsct, lessThan = 1.5)
```

Once we have narrowed down to the edges that we are interested in, we would like to change the color attribute for these edges in our original `graphBAM` objects for the male and female graphs to "red". Before an attribute can be added, we have to set its default value using the `edgedataDefaults` method. For our example, we set the default value for the color attribute to white.

We first obtain the from - to relationship for the `resWt` graph using the `extractFromTo` method and then make use of the `edgeData` method to update the "color" edge attribute.

```
> ftSub <- extractFromTo(resWt)
> edgeDataDefaults(male, attr = "color") <- "white"
> edgeDataDefaults(female, attr = "color") <- "white"
> edgeData(male, from = as.character(ftSub[,"from"]),
+         to = as.character(ftSub[,"to"]), attr = "color") <- "red"
> edgeData(female, from = as.character(ftSub[,"from"]),
+         to = as.character(ftSub[,"to"]), attr = "color") <- "red"
>
```

2 MultiGraphs

2.1 Introduction

The *MultiGraph* class can be used to represent graphs that share a single node set and have have one or more edge sets, each edge set representing a different type of interaction between the nodes. An `edgeSet` object can be described as representing the relationship between a set of from-nodes and to-nodes which can either be directed or undirected. A numeric value (weight) indicates the strength of interaction between the connected edges.

Self loops are permitted in the *MultiGraph* class representation (i.e. the from-node is the same as the to-node). The *MultiGraph* class supports the handling of arbitrary node and edge attributes. These attributes are stored separately from the edge weights to facilitate efficient edge weight computation.

We shall load the *graph* and *RBGL* packages that we will be using. We will then create a *MultiGraph* object and then spend some time examining some of the different functions that can be applied to *MultiGraph* objects.

```
> library(graph)
> library(RBGL)
```

2.2 A simple MultiGraph example

We proceed to construct a *MultiGraph* object with directed `edgeSets` to represent the flight connections of airlines Alaska, Delta, United and American that fly between the cities Baltimore, Denver, Houston, Los Angeles, Seattle and San Francisco. For our example, the cities represent the nodes of the *MultiGraph* and we have one `edgeSet` each for the airlines. Each `edgeSet` represents the flight connections from an originating city(`from`) to the destination city(`to`). The weight represents the fare for flying between the `from` and `to` cities.

For each airline, we proceed to create a *data.frame* containing the originating city, the destination city and the fare.

```
> ft1 <- data.frame(
+         from = c("SEA", "SFO", "SEA", "LAX", "SEA"),
+           to = c("SFO", "LAX", "LAX", "SEA", "DEN"),
+       weight = c( 90, 96, 124, 115, 259),
+       stringsAsFactors = TRUE)
> ft2 <- data.frame(
```

```
+         from = c("SEA", "SFO", "SEA", "LAX", "SEA", "DEN", "SEA", "IAH", "DEN"),
+           to = c("SFO", "LAX", "LAX", "SEA", "DEN", "IAH", "IAH", "DEN", "BWI"),
+        weight= c(169, 65, 110, 110, 269, 256, 304, 256, 271),
+        stringsAsFactors = TRUE)
> ft3 <- data.frame(
+    from = c("SEA", "SFO", "SEA", "LAX", "SEA", "DEN", "SEA", "IAH", "DEN", "BWI"),
+     to  = c("SFO", "LAX", "LAX", "SEA", "DEN", "IAH", "IAH", "DEN", "BWI", "SFO"),
+  weight = c(237, 65, 156, 139, 281, 161, 282, 265, 298, 244),
+  stringsAsFactors = TRUE)
> ft4 <- data.frame(
+     from = c("SEA", "SFO", "SEA", "SEA", "DEN", "SEA", "BWI"),
+      to  = c("SFO", "LAX", "LAX", "DEN", "IAH", "IAH", "SFO"),
+   weight = c(237, 60, 125, 259, 265, 349, 191),
+   stringsAsFactors = TRUE)
```

These data frames are then passed to *MultiGraph* class constructor as a named `list`, each member of the list being a `data.frame` for an airline. A logical vector passed to the `directed` argument of the *MultiGraph* constructor indicates whether the `MultiGraph` to be created should have directed or undirected edge sets.

```
> esets <- list(Alaska = ft1, United = ft2, Delta = ft3, American = ft4)
> mg <- MultiGraph(esets, directed = TRUE)
> mg
```

```
MultiGraph with 6 nodes and 4 edge sets
 edge_set directed edge_count
   Alaska     TRUE          5
   United     TRUE          9
    Delta     TRUE         10
 American     TRUE          7
```

The nodes (cities) of the *MultiGraph* object can be obtained by using the `nodes` method.

```
> nodes(mg)
```

```
[1] "BWI" "DEN" "IAH" "LAX" "SEA" "SFO"
```

To find the fares for all the flights that originate from SEA for the Delta airline, we can use the `mgEdgeData` method.

```
> mgEdgeData(mg, "Delta", from = "SEA", attr = "weight")
```

```
$`SEA|DEN`
[1] 281

$`SEA|IAH`
[1] 282

$`SEA|LAX`
[1] 156

$`SEA|SFO`
[1] 237
```

We proceed to add some node attributes to the `MultiGraph` using the `nodeData` method. Before node attributes can be added, we have to set a default value for each node attribute using the `nodeDataDefuault` method. For our example, we would like to set a default value of square for the node attribute shape.

We would like to set the node attribute "shape" for Seattle to the value `"triangle"` and that for the cities that connect with Seattle to the value `"circle"`.

```
> nodeDataDefaults(mg, attr="shape") <- "square"
> nodeData(mg, n = c("SEA", "DEN", "IAH", "LAX", "SFO"), attr = "shape")  <-
+     c("triangle", "circle", "circle", "circle", "circle")
```

The node attribute shape for cities we have not specifically assigned a value (such as BWI) gets assigned the default value of "square".

```
> nodeData(mg,  attr = "shape")

$BWI
[1] "square"

$DEN
[1] "circle"

$IAH
[1] "circle"

$LAX
[1] "circle"

$SEA
[1] "triangle"

$SFO
[1] "circle"
```

We then update the edge attribute `color` for the Delta airline flights that connect with Seattle to "green". For the remaining Delta flights that connect to other destination in the MultiGraph, we would like to assign a default color of "red".

Before edge attributes can be added to the MultiGraph, their default values must be set using the `mgEdgeDataDefaults` method. Subsequently, the `megEdgeData<-` method can be used to update specific edge attributes.

```
> mgEdgeDataDefaults(mg, "Delta", attr = "color")  <- "red"
> mgEdgeData(mg, "Delta", from = c("SEA", "SEA", "SEA", "SEA"),
+         to = c("DEN", "IAH", "LAX", "SFO"), attr = "color") <- "green"

> mgEdgeData(mg, "Delta", attr = "color")

$`DEN|BWI`
[1] "red"

$`IAH|DEN`
[1] "red"
```

```
$`SEA|DEN`
[1] "green"

$`DEN|IAH`
[1] "red"

$`SEA|IAH`
[1] "green"

$`SEA|LAX`
[1] "green"

$`SFO|LAX`
[1] "red"

$`LAX|SEA`
[1] "red"

$`BWI|SFO`
[1] "red"

$`SEA|SFO`
[1] "green"
```

We are only interested in studying the fares for the airlines Alaska, United and Delta and hence would like to create a smaller *MultiGraph* object containing edge sets for only these airlines. This can be achieved using the `subsetEdgeSets` method.

```
> g <- subsetEdgeSets(mg, edgeSets = c("Alaska", "United", "Delta"))
```

We proceed to find out the lowest fares for Alaska, United and Delta along the routes common to them. To do this, we make use of the `edgeSetIntersect0` method which computes the intersection of all the edgesets in a MultiGraph. While computing the intersection of edge sets, we are interesting in retaining the lowest fares in cases where different airlines flying along a route have different fares. To do this, we pass in a named list containing the `weight` function that calculates the minimum of the fares as the input to the `edgeSetIntersect0` method. (The user has the option of specifying any function for appropriate handling of edge attributes).

```
> edgeFun <- list( weight = min)
> gInt <- edgeSetIntersect0(g, edgeFun = edgeFun)
> gInt

MultiGraph with 6 nodes and 1 edge sets
            edge_set directed edge_count
 Alaska_United_Delta     TRUE          5
```

The edge set by the `edgeSetIntersect0` operation is named by concatenating the names of the edgeSets passed as input to the function.

```
> mgEdgeData(gInt, "Alaska_United_Delta", attr= "weight")

$`SEA|DEN`
[1] 259
```

```
$`SEA|LAX`
[1] 110

$`SFO|LAX`
[1] 65

$`LAX|SEA`
[1] 110

$`SEA|SFO`
[1] 90
```

2.3 MultiGraph representation of mice gene interaction data. (SAGE)

The C57BL/6J and C3H/HeJ mouse strains exhibit different cardiovascular and metabolic phenotypes on the hyperlipidemic apolipoprotein E (Apoe) null background. The interaction data for the genes from adipose, brain, liver and muscle tissue samples from male and female mice were studied. This interaction data for the various genes is included in the *graph* package as a list of `data.frame`s containing information for `from-gene`, `to-gene` and the strength of interaction `weight` for each of the tissues studied.

We proceed to load the data for male and female mice.

```
> data("esetsFemale")
> data("esetsMale")
> names(esetsFemale)

[1] "adipose" "brain"   "liver"   "muscle"

> head(esetsFemale$brain)

         from          to weight
1 10024404688 10024393150  0.853
2 10024406215 10024393156  0.513
3 10024411796 10024393163  1.000
4 10024415608 10024393167  0.727
5 10024399302 10024393196  0.342
6 10024399912 10024393196  0.555
```

The `esetsFemale` and `esetsMale` objects are a named `list` of data frames corresponding to the data obtained from adipose, brain, liver and muscle tissues for the male and female mice that were studied. Each data frame has a from, to and a weight column corresponding to the from and to genes that were studied and weight representing the strength of interaction of the corresponding genes.

We proceed to create *MultiGraph* objects for the male and female data sets by making use of the *MultiGraph* constructor, which directly accepts a named list of data frames as the input and returns a MultiGraph with edgeSets corresponding to the names of the data frames.

```
> female  <- MultiGraph(edgeSets = esetsFemale, directed = TRUE)
> male  <- MultiGraph(edgeSets = esetsMale, directed = TRUE )
> male

MultiGraph with 7081 nodes and 4 edge sets
 edge_set directed edge_count
```

```
  adipose     TRUE       1601
    brain     TRUE       2749
    liver     TRUE       3690
   muscle     TRUE       3000
```

```
> female
```

```
MultiGraph with 7072 nodes and 4 edge sets
 edge_set directed edge_count
  adipose     TRUE       2108
    brain     TRUE       2789
    liver     TRUE       3584
   muscle     TRUE       2777
```

We then select a particular gene of interest in this network and proceed to identify its neighboring genes connected to this gene in terms of the maximum sum of weights along the path that connects the genes for the brain edge set.

We are interested in the gene "10024416717" and the sum of the weights along the path that connects this genes to the other genes for the brain tissue. Since the algorithms in the *RBGL* package that we will use to find the edges that are connected to the gene "10024416717" do not work directly with *MultiGraph* objects, we proceed to create `graphBAM` objects from the male and female edge sets for the brain tissue.

MultiGraph objects can be converted to a named list of `graphBAM` objects using the `graphBAM` method.

```
> maleBrain <- extractGraphBAM(male, "brain")[["brain"]]
> maleBrain
```

```
A graphBAM graph with directed edges
Number of Nodes = 7081
Number of Edges = 2749
```

```
> femaleBrain <-  extractGraphBAM(female, "brain")[["brain"]]
```

We then identify the genes connected to gene "10024416717" as well as the sum of the weights along the path that connect the identified genes using the function `bellman.ford.sp` function from the *RBGL* package.

```
> maleWt <- bellman.ford.sp(maleBrain, start = c("10024416717"))$distance
> maleWt <- maleWt[maleWt != Inf & maleWt != 0]
> maleWt
```

```
10024409301 10024409745
      0.636       1.389
```

```
> femaleWt <- bellman.ford.sp(femaleBrain, start = c("10024416717"))$distance
> femaleWt <- femaleWt[femaleWt != Inf & femaleWt != 0]
> femaleWt
```

```
10024393904 10024409503
      0.789       0.866
```

For the subset of genes we identified, we proceed to add node attributes to our original `MultiGraph` objects for the male and female data. The node "10024416717" and all its connected nodes are assigned a color attribute "red" while the rest of the nodes are assigned a color color attribute of "gray".

```
> nodeDataDefaults(male, attr = "color") <- "gray"
> nodeData(male , n = c("10024416717", names(maleWt)), attr = "color" ) <- c("red")
> nodeDataDefaults(female, attr = "color") <- "gray"
> nodeData(female , n = c("10024416717", names(femaleWt)), attr = "color" ) <- c("red")
```

Our `MultiGraph` objects now contain the required node attributes for the subset of genes that we have narrowed our selection to.

For the `MultiGraph` objects for male and female, we are also interested in the genes that are common to both `MultiGraph`s. This can be calculated using the `graphIntersect` method.

```
> resInt <- graphIntersect(male, female)
> resInt

MultiGraph with 5699 nodes and 4 edge sets
 edge_set directed edge_count
  adipose     TRUE         88
    brain     TRUE        473
    liver     TRUE        455
   muscle     TRUE        370
```

The operations we have dealt with so far only deal with manipulation of *MultiGraph* objects. Additional functions will need to be implemented for the visualization of the *MultiGraph* objects.