The generation, visualization, and analysis of link communities in arbitrary networks with the R package **linkcomm**

Alex T. Kalinka

March 11, 2020
alex.t.kalinka@gmail.com

Abstract

Identifying communities of related nodes in networks is an essential tool enabling researchers to make sense of complex data sets. The `linkcomm` package greatly facilitates this process by extracting communities of links from networks of arbitrary size and type. By clustering links, as opposed to nodes, it is possible for nodes to belong to multiple communities thereby revealing the overlapping and nested structure of the network and uncovering the key nodes that form connections across several communities. In addition to this, `linkcomm` provides extensive functionality for visualizing and analysing the resulting link communities. `linkcomm` also provides tools for generating, visualizing, and analysing clusters generated by the Overlapping Cluster Generator (OCG) algorithm.

Contents

1 Introduction

The representation of systems of interacting elements as networks has brought fresh perspectives and insights to the analysis of complex phenomena from the biological to the social sciences. When analysing the structure, function, and dynamics of these networks it is extremely useful to identify sets of related nodes, known as communities (Radicchi *et al.*, 2004).

This approach is greatly enhanced by clustering the links between nodes, rather than the nodes themselves (Evans and Lambiotte, 2009; Ahn *et al.*, 2010). With this method it is possible for nodes to belong to multiple communities, and this in turn reveals the overlapping and nested structure of the network while simultaneously identifying key nodes with membership across several communities.

The R package `linkcomm` (Kalinka and Tomancak, 2011) implements the algorithm proposed by Ahn *et al.* (2010). Similarities between links that share a node are assigned using the Jaccard coefficient and these similarities are then used to hierarchically cluster the links. The resulting dendrogram is cut at a point that maximises the density of links within the clusters after normalizing against the maximum and minimum numbers of links possible in each cluster.

2 Quick start

To install `linkcomm` run the following command within R:

```
> install.packages("linkcomm")
```

To load `linkcomm` into your current R session:

```
> library(linkcomm)
```

To run an interactive demonstration of `linkcomm` within R:

```
> demo(topic = "linkcomm", package = "linkcomm")
```

Four networks are provided as part of the package so that users can see how the functions work before using their own data:

1. A co-appearance network for the stage musical of *Les Misérables* (`lesmiserables`).
2. A social network of friendships between the members of a karate club (`karate`).
3. A sub-network of protein interactions from the yeast interactome (`pp_rnapol`).
4. A sub-network of protein interactions from the human interactome (`human_pp`).

Each network is arranged as an edge list and stored as a data frame. Other acceptable formats for input networks are character or integer matrices.

```
> head(lesmiserables)
```

```
              V1             V2
1       Napoleon         Myriel
2 MlleBaptistine         Myriel
3    MmeMagloire         Myriel
4    MmeMagloire MlleBaptistine
5   CountessDeLo         Myriel
6       Geborand         Myriel
```

3 Example session

To illustrate the package we will extract communities from a densely-connected sub-network from the yeast protein interactome (Yu *et al.*, 2008). This sub-network includes 56 proteins and 449 interactions involved in transcription (see Ahn *et al.* (2010), SI, p. 16). The network is undirected and unweighted (see section 3.4 for dealing with directed and weighted networks).

3.1 Extracting link communities

When extracting communities from a network, the input data must be arranged as a simple edge list.

```
> yeast_pp <- read.table("pp_rnapol.txt", header = FALSE)
> head(yeast_pp)

       V1      V2
1 YBR198C YGL112C
2 YCL010C YDR176W
3 YDR167W YML015C
4 YDR167W YML114C
5 YDR176W YDR448W
6 YDR190C YPL235W
```

In the above edge list, each row corresponds to an interaction between the elements in each of the columns. The interacting elements can be character names, as above, or integer numbers. We will extract link communities from this network using the "single" hierarchical clustering method, specified using the `hcmethod` argument (several clustering methods are available for networks that can be handled in the memory, see `?getLinkCommunities` - the default is `average`).

```
> lc <- getLinkCommunities(yeast_pp, hcmethod = "single")

   Removing loops...
   Removing edge duplicates...
   Calculating edge similarities for 449 edges... 100.00%
   Hierarchical clustering of edges...
   Calculating link densities... 100.00%
   Partition density maximum height =  0.33333
   Finishing up...4/4... 100%
   Plotting...
   Colouring dendrogram... 100%
```

The algorithm returns an object of class `linkcomm` and plots a summary of the results (Fig. 1). Included in this object are the communities that were extracted together with additional data required for plotting and further analysing the communities (see `?getLinkCommunities` for the elements of the `linkcomm` object).

To view a summary of this object:

```
> print(lc)

   *** Summary ***
   Number of nodes =  56
   Number of edges =  449
   Number of communities =  11
   Maximum partition density =  0.7960981
   Number of nodes in largest cluster =  22
   Directed:  FALSE
   Bi-partite:  FALSE
   Hclust method:  single
```

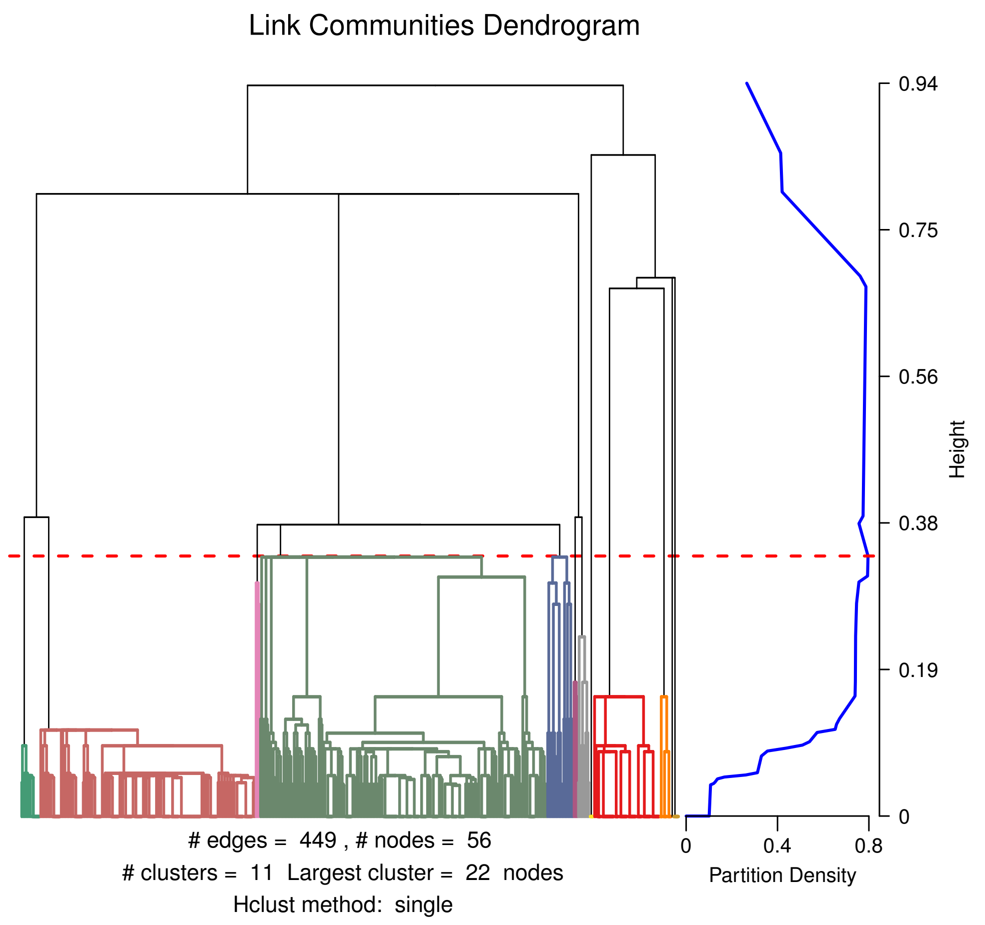

Figure 1: *Example output from extracting link communities from a yeast protein interaction network involved in transcription.*

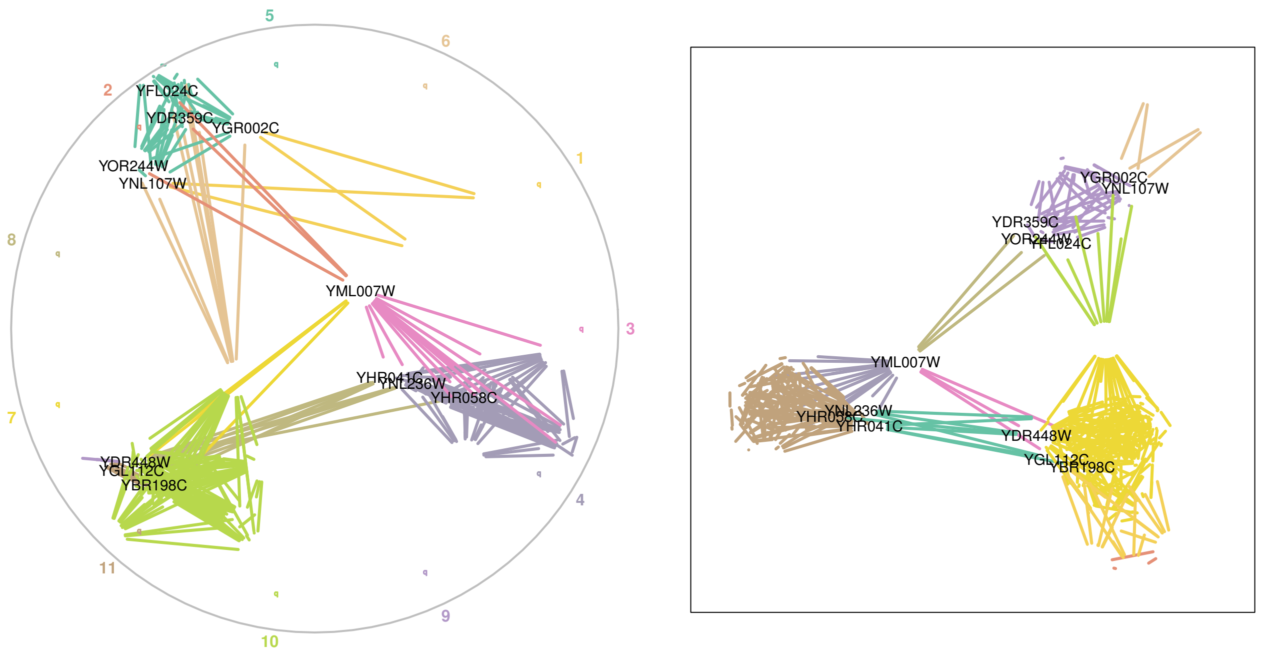

Figure 2: *Visualization of link communities using graphs. The panel on the left shows a Spencer circle layout (Spencer, 2010), while the panel on the right shows a Fruchterman-Reingold layout. In both graphs, we show only the nodes that belong to 3 or more communities. Numbers around the circumference of the circle refer to community IDs.*

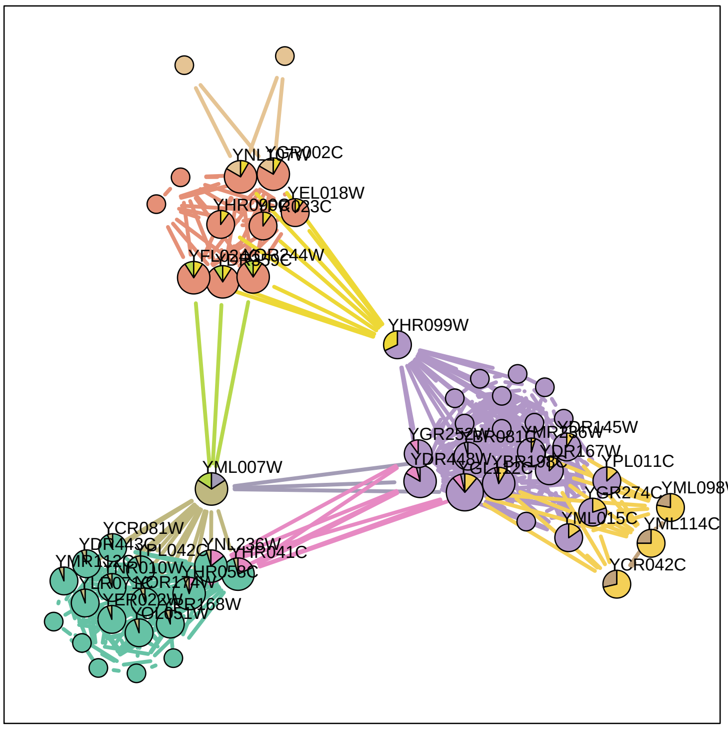

Figure 3: *Visualization of link communities using node pies. The fraction of the total number of edges that a node has in each community is depicted using a pie chart.*

3.2 Visualizing link communities

We can visualize link communities using coloured edges in a graph layout.

```
> plot(lc, type = "graph", layout = layout.fruchterman.reingold)
> plot(lc, type = "graph", layout = "spencer.circle")
```

To aid the visualization of key nodes we can limit the display of nodes using the `shownodesin` parameter. The following displays only the nodes that belong to 3 or more communities (Fig. 2):

```
> plot(lc, type = "graph", layout = "spencer.circle", shownodesin = 3)
```

It's also possible to visualize node membership to different communities using node pies, in which the relative fraction of edges a node has in each community is depicted using a pie chart (Fig. 3):

```
> plot(lc, type = "graph", shownodesin = 2, node.pies = TRUE)
```

We can also visualize node community membership for the top-connected nodes using a community membership matrix (Fig. 4):

```
> plot(lc, type = "members")
```

We can also display a summary of the results of the link communities algorithm (Fig. 1):

```
> plot(lc, type = "summary")
```

Additionally, we can simply plot the dendrogram on its own with coloured community clusters:

```
> plot(lc, type = "dend")
```

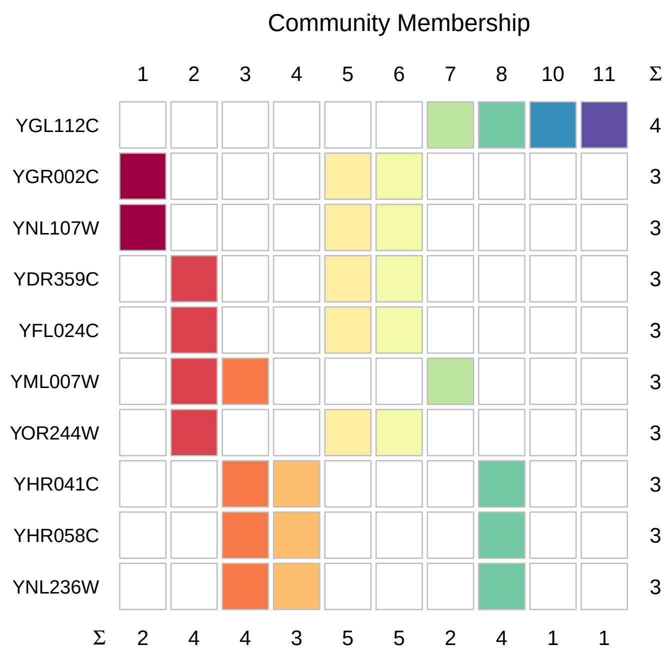

Figure 4: *Visualizing community membership for nodes that belong to the most communities. Colours indicate community-specific membership.*

3.3 Analysing link communities

3.3.1 Nested communities

Nodes can belong to multiple link communities, and so it is possible to discover sets of nodes that belong to a community that is entirely nested within a larger community of nodes:

```
> getAllNestedComm(lc)
```

```
$`9`
[1] 11
```

This result indicates that there is one nested community; the nodes in community 9 are entirely nested within the nodes of community 11. We can verify this by plotting these communities as a graph (Fig. 5):

```
> getNestedHierarchies(lc, clusid = 9)
```

or

```
> plot(lc, type = "graph", clusterids = c(9,11))
```

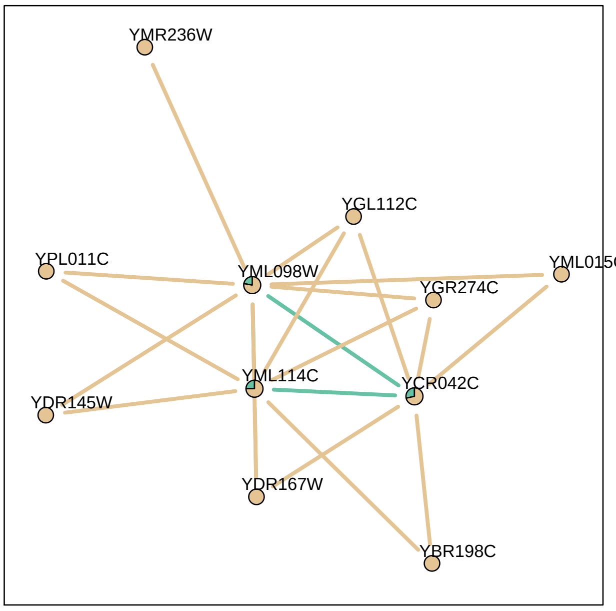

Figure 5: *Nested community structure revealed by* `linkcomm`*. YML098W, YCR042C, and YML114C belong to both communities.*

3.3.2 Relationships between communities

We can also explore the relationships between communities based on the numbers of shared nodes. To do this we can hierarchically cluster the communities after scoring the pair-wise similarities between them using the Jaccard coefficient (based on the number of nodes that they share).

```
> cr <- getClusterRelatedness(lc, hcmethod = "ward.D")
```

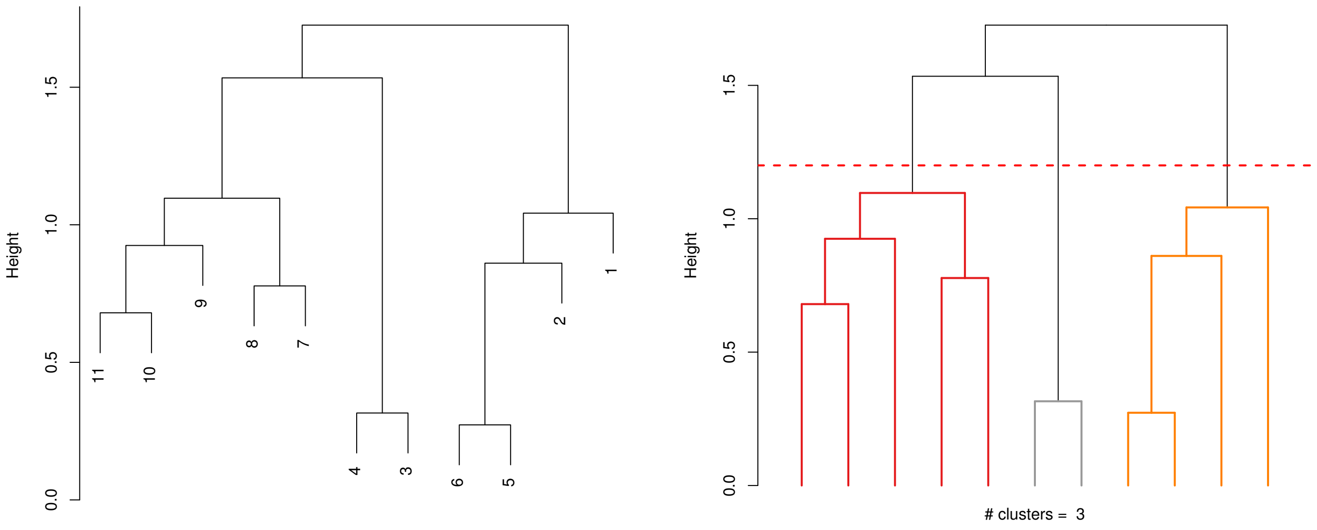

Figure 6: *Dendrograms showing, on the left, the similarity between link communities, and, on the right, clustering between these communities to produce 3 meta-communities.*

This function returns a hierarchical clustering object of class "hclust" and plots the dendrogram (Fig. 6). After inspecting this dendrogram, we can select a height at which to cut the dendrogram and extract the resulting meta-communities.

```
> cutDendrogramAt(cr, cutat = 1.2)
```

This returns the meta-communities (clusters of community IDs). Alternatively, we could have cut the dendrogram using the `cutat` option in the original clustering function, `getClusterRelatedness`. We can verify that these meta-communities make sense by inspecting the communities in the Spencer circle (Fig. 2). Note that nodes may still belong to multiple meta-communities.

When working with large networks, the algorithm will typically identify large numbers of link communities (hundreds or more), which can be difficult to handle. Therefore, it is useful to collapse these larger numbers of communitites into smaller numbers of meta-communities. This can be achieved with the `meta.communities()` function, which utilises an algorithm to automatically identify clusters from a hierarchical clustering dendrogram (Langfelder et al, 2008):

```
> mc <- meta.communities(lc, hcmethod = "ward.D", deepSplit = 0)
```

Where the `deepSplit` argument, taking values from 0-4, indicates how fine-grained the clustering should be, with higher values producing more clusters. The function outputs an object of class `linkcomm` but with a smaller set of larger communities.

3.3.3 Community centrality

In link communities, nodes may belong to several communities, and so it is possible to measure the importance of a node in a network based on the number of communities to which it belongs. To do this, we weight the membership of a node in a community by how distinct that community is from the other communities to which the same node belongs

$$C_c(i) = \sum_{i \in j}^{N} \left(1 - \frac{1}{m} \sum_{i \in j \cap k}^{m} S(j,k) \right), \tag{1}$$

where the main sum is over the N communities to which node i belongs, and $S(j,k)$ refers to the similarity between community j and k, calculated as the Jaccard coefficient for the number of shared nodes between each community pair, and this is averaged over the m communities paired with community j and in which node i jointly belongs.

Nodes that belong to a lot of different communities will get the largest community centrality scores, whereas nodes that belong to overlapping, nested or few communities will get the lowest scores.

We can calculate this value for a set of nodes that have been assigned to link communities:

```
> cc <- getCommunityCentrality(lc)
```

3.3.4 Community modularity and connectedness

We can also calculate the modularity of communities - the relative number of links within the community versus links outside of the community - and its inverse, community connectedness. The modularity of community i can be written as

$$M_i = \left(\frac{e_w(i)}{n_i(n_i - 1)/2} \right) \cdot \left(\frac{e_b(i)}{n_i \bar{d}} \right)^{-1}, \tag{2}$$

where $e_w(i)$ and $e_b(i)$ are the number of links within and without community i respectively, n_i is the number of nodes in community i, and $\bar{d}$ is the average degree of nodes in the network.

We can calculate and plot the modularity of the communities in our network (Fig. 7):

```
> cm <- getCommunityConnectedness(lc, conn = "modularity")
> plot(lc, type = "commsumm", summary = "modularity")
```

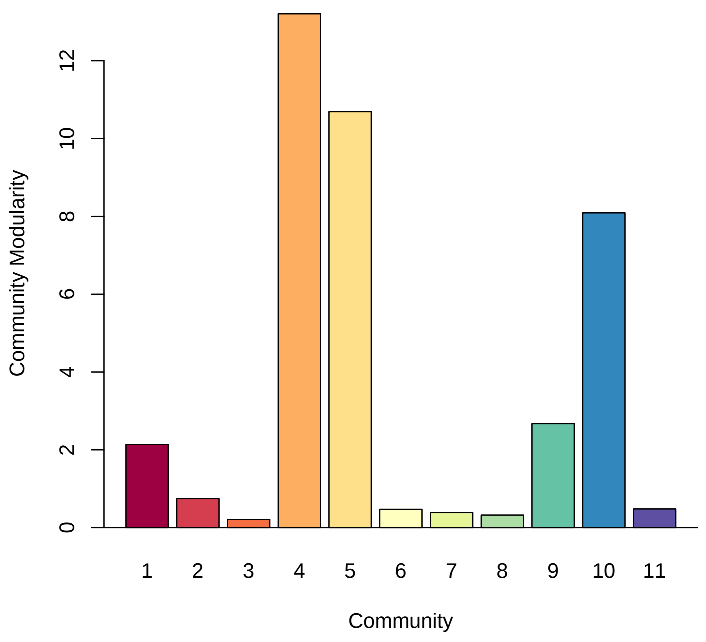

Figure 7: *The modularity of different link communities.*

We can verify that these measures make sense by inspecting the communities in the Spencer circle (Fig. 2).

3.3.5 User-defined link communities

It is also possible for the user to extract link communities that are not defined by the maximisation of the link partition density. Instead, the user can inspect the clustering dendrogram and select a different height at which to extract communities:

```
> lc2 <- newLinkCommsAt(lc, cutat = 0.4)
```

Now the number and composition of the communities has changed, but we use the clustering method from the original clustering:

```
> print(lc2)
```

```
   *** Summary ***
   Number of nodes =  56
   Number of edges =  449
   Number of communities =  7
   Maximum partition density =  0.7960981
   Number of nodes in largest cluster =  25
   Directed:  FALSE
   Bi-partite:  FALSE
   Hclust method:  single
```

3.3.6 Community membership of nodes

We can extract the nodes from single or multiple communities:

```
> getNodesIn(lc, clusterids = c(4,5))

 [1] "YDR308C" "YER022W" "YGL127C" "YNR010W" "YBL093C" "YCR081W" "YDL005C"
 [8] "YDR443C" "YGR104C" "YHR041C" "YHR058C" "YLR071C" "YMR112C" "YNL236W"
[15] "YOL051W" "YOR174W" "YPL042C" "YPR168W" "YFL024C" "YDR359C" "YEL018W"
[22] "YGR002C" "YHR090C" "YJR082C" "YNL107W" "YNL136W" "YOR244W" "YPR023C"
```

We can also find nodes that are shared by sets of communities:

```
> get.shared.nodes(lc, comms = c(3,4))

 [1] "YER022W" "YCR081W" "YDR443C" "YHR041C" "YHR058C" "YLR071C" "YMR112C"
 [8] "YNL236W" "YNR010W" "YOL051W" "YOR174W" "YPL042C" "YPR168W"
```

3.4 Directed and weighted networks

When analysing directed networks, we need to specify that the network is directed, and/or choose a weight for links that share nodes yet are in the opposite orientation (the default value is 0.5):

```
> lc <- getLinkCommunities(yeast_pp, directed = TRUE, dirweight = 0.8)
```

For weighted networks, the input data must be an edge list with an additional third column of numerical weights, for example:

```
> head(weighted)

        node1       node2            weight
1 FBgn0000575 FBgn0037290               0.5
2 FBgn0000575 FBgn0004880               0.5
3 FBgn0000575 FBgn0037375 0.333333333333333
4 FBgn0000575 FBgn0040281              0.25
5 FBgn0000575 FBgn0035101 0.333333333333333
6 FBgn0000575 FBgn0027111               0.4
```

For both directed and weighted networks, the algorithm will score the similarities between links that share a node using the Tanimoto coefficient:

$$S(e_{ik}, e_{jk}) = \frac{\mathbf{a}_i.\mathbf{a}_j}{|\mathbf{a}_i|^2 + |\mathbf{a}_j|^2 - \mathbf{a}_i.\mathbf{a}_j}, \tag{3}$$

where $\mathbf{a}_i$ refers to a vector describing the weights of links between node i and the nodes in the first-order neighbourhoods of both nodes i and j (equal to 0 in the event of an absent link). For directed networks, links to nodes shared by both node i and j are given a user-defined weight below 1 if they are in the opposite orientation.

3.5 Handling large networks

The `linkcomm` package can handle networks of any size. To do so, the upper triangular dissimilarity matrix used by the hierarchical clustering algorithm is compressed and written to disk as a temporary file. This matrix is then read, modified, and re-written (by a compiled C++ function) as clustering proceeds until the file size is 0 bytes. The speed at which large networks are processed will depend on the power of the computer being used. The size of the file that holds the compressed matrix may also be very large, so hard disk space could be a limiting factor for extremely large networks. In these cases, the hierarchical clustering method will always be "single" to enhance performance.

We make no assumptions about the computer resources available to end users, and so we provide a parameter, `edglim`, which can be modified by the user and which determines the maximum permissible size of the input network in terms of links for it to be handled in memory - above this size the dissimilarity matrix will be handled on the disk. The default value is 10^4 links, but this can be modified:

```
> lc <- getLinkCommunities(yeast_pp, edglim = 10)
```

As a guide, a network with 10^4 links will require $((10^4)^2) * 8 = 800$ MB to be handled in an uncompressed format in the memory.

3.6 Exporting link communities to Cytoscape

Cytoscape is an open source platform for complex-network analysis and visualization [1]. We can export our link communities into an edge attribute file that can be imported into Cytoscape:

```
> linkcomm2cytoscape(lc, interaction = "pp", ea = "linkcomms.ea")
```

This will save an edge attribute file to "linkcomms.ea" in the current directory. Communities can also be exported to the Clust&See Cytoscape plug-in which can be used for visualizing and manipulating the clusters produced by various network clustering algorithms (Spinelli *et al*, 2013):

```
> linkcomm2clustnsee(lc, file = "temp.cns", network.name = "network")
```

This will save a Clust&See compatible file to the current directory.

[1] http://www.cytoscape.org/

4 Customizing network visualizations

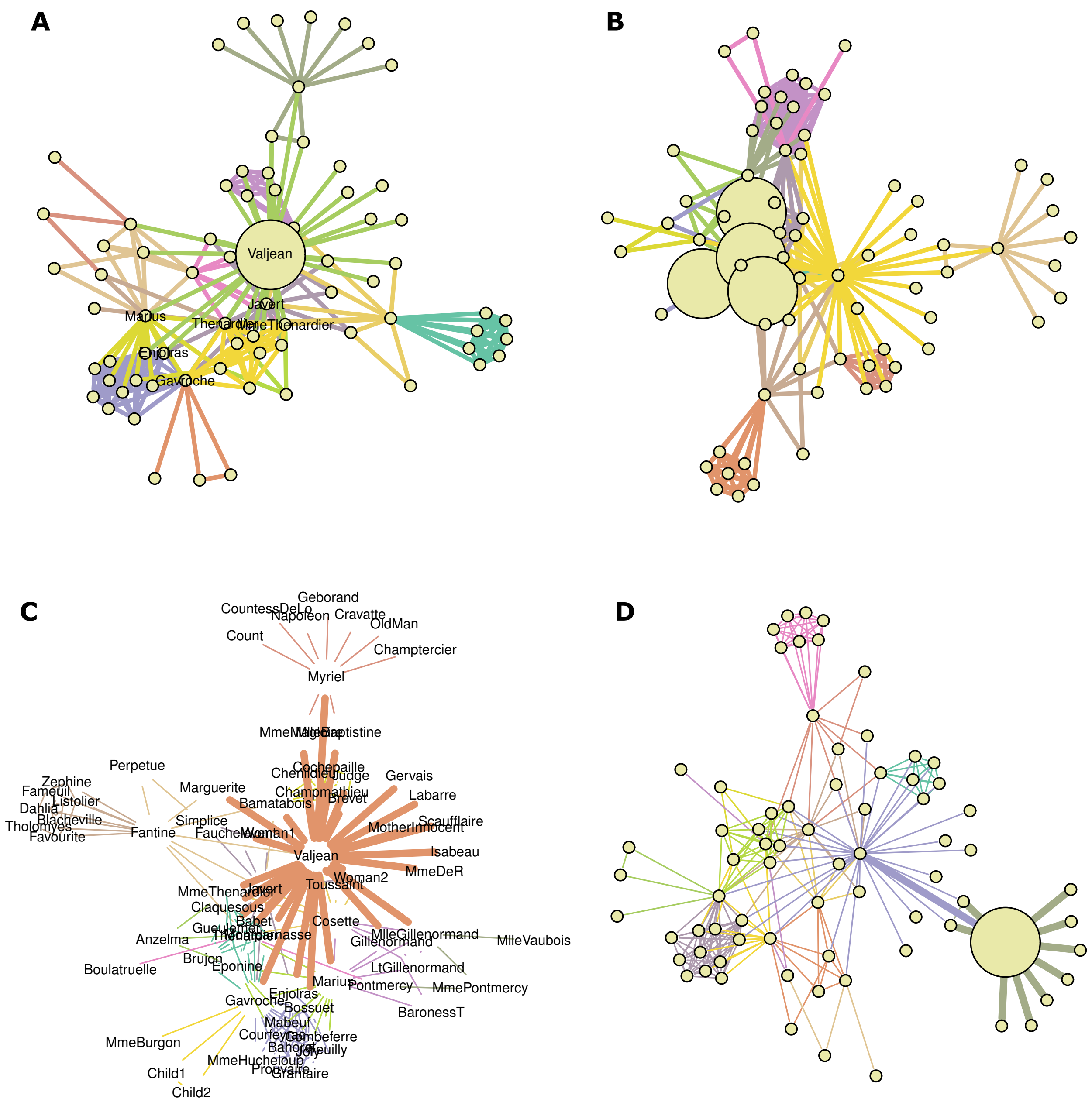

Figure 8: *Customized network visualizations.*

While default network visualization styles are set in the package, full control of how networks are visualized is possible. To demonstrate this functionality, we will work with the co-appearance network of *Les Misérables*.

```
> lm <- getLinkCommunities(lesmiserables, plot = FALSE)
```

If we wish to visualize the link communities using node shapes and with an enlarged node size for a particular node (in this case “Valjean”) then we can achieve this using the `graph.feature` function to generate a vector of node sizes which will be passed to the `plot` function (Fig. 8A). This function requires that the indices of the nodes or edges that are to be altered is provided together with the new values (`features`) and the values held by the rest of the nodes or edges in the network (`default`):

```
> nf <- graph.feature(lm, type = "nodes", indices = which(V(lm$igraph)$name == "Valjean"),
+                      features = 30, default = 5)
> plot(lm, type = "graph", vsize = nf, vshape = "circle", shownodesin = 4)
```

We may also plot larger nodes for all of the nodes belonging to a particular community or several communities. Here we enlarge the nodes belonging to community 1 using the `getNodesIn` function, and we drop node labels using the `vlabel` argument (Fig. 8B):

```
> nf <- graph.feature(lm, type = "nodes", indices = getNodesIn(lm, clusterids = 1,
+                      type = "indices"), features = 30, default = 5)
> plot(lm, type = "graph", vsize = nf, vshape = "circle", vlabel = FALSE)
```

In addition, we may wish to make particular edges thicker. We can make all of the edges in community 14 thicker using the `getEdgesIn` function (Fig. 8C):

```
> ef <- graph.feature(lm, type = "edges", indices = getEdgesIn(lm, clusterids = 14),
+                      features = 5, default = 1)
> plot(lm, type="graph", ewidth = ef)
```

We can also make the edges incident upon a particular node thicker. Here we make all of the edges incident upon "Myriel" thicker, and also make this node larger (Fig. 8D):

```
> ef <- graph.feature(lm, type = "edges", indices = getEdgesIn(lm, nodes = "Myriel"),
+                      features = 5, default = 1)
> nf <- graph.feature(lm, type = "nodes", indices = which(V(lm$igraph)$name == "Myriel"),
+                      features = 30, default = 5)
> plot(lm, type = "graph", vsize = nf, ewidth = ef, vshape = "circle", vlabel = FALSE)
```

Several other options are available and are described in `?plotLinkCommGraph` and also in `?igraph.plotting` in the `igraph` R package.

5 Overlapping Cluster Generator (OCG)

Becker *et al.* (2012) developed the OCG algorithm to generate clusters of nodes that may belong to more than one cluster. The algorithm iteratively fuses clusters of nodes according to an overlapping extension of Newman's modularity measure until no further gains in overlapping modularity can be attained, thus producing a set of optimally overlapping clusters (Becker *et al.*, 2012). This algorithm is integrated into `linkcomm` and in what follows we demonstrate how it can be used.

5.1 OCG clusters

OCG clusters can be extracted from networks arranged as an edge list, either directly from a file or from a data frame or matrix within R.

```
> oc <- getOCG.clusters(lesmiserables)
```

```
Calculating Initial class System....Done
Nb. of classes 43
Nb. of edges not within the classes 19
Number of initial classes 43
Running....
Remaining classes: None
Reading OCG data...
Extracting cluster sizes... 100%
```

The algorithm returns an object of class `OCG`. Contained within this object are the clusters together with all the information necessary for visualizing and analysing them further (see `?getOCG.clusters` for the elements of the `OCG` object). A summary of this object can be printed to the screen:

```
> print(oc)
```

```
   *** Summary ***
   Number of nodes =  77
   Number of edges =  254
   Number of communities =  38
   Number of nodes in largest cluster =  10
   Modularity =  90407
   Q =  0.6769
```

In the OCG algorithm, nodes and not edges belong to communities, and these can be visualized by depicting node community membership according to the number of edges a node shares within each community (see Figure 9):

```
> plot(oc, type = "graph", shownodesin = 7, scale.vertices = 0.1)
```

The option `scale.vertices` allows us to increase the size of nodes by a specified fraction for each additional community to which it belongs. By setting this to `NULL`, all nodes will be the same size (see `?plotOCGraph` for all the options available for visualizing `OCG` clusters).

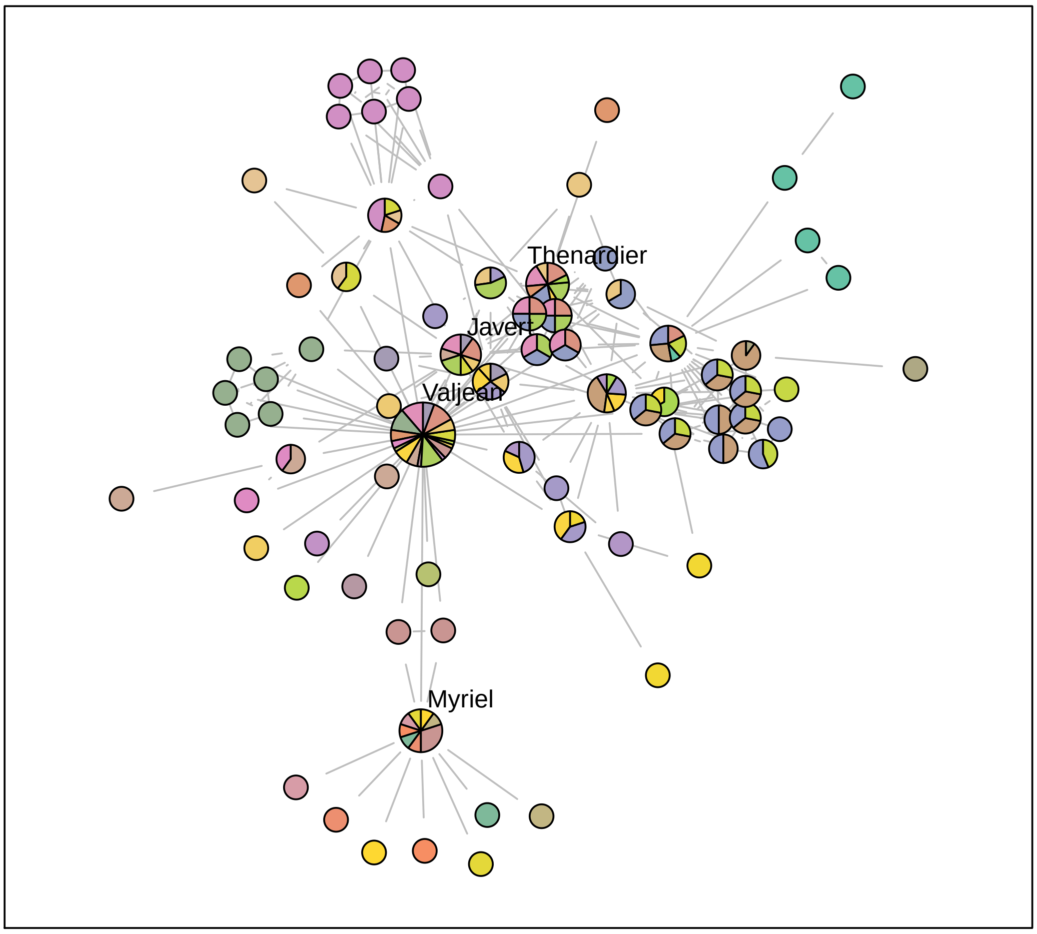

Figure 9: *Visualizing OCG clusters.*

References

Ahn, Y.Y., Bagrow, J.P., and Lehmann, S. (2010) Link communities reveal multiscale complexity in networks. *Nature*, **466**, 761-764.

Becker, E., Robisson, B., Chapple, C.E., Guenoche, A, and Brun, C. (2012) Multifunctional proteins revealed by overlapping clustering in protein interaction network. *Bioinformatics*, **28**, 84-90.

Evans, T.S., and Lambiotte, R. (2009) Line graphs, link partitions and overlapping communities. *Phys. Rev. E.*, **80**, 016105.

Kalinka, A.T., and Tomancak, P. (2011) linkcomm: an R package for the generation, visualization, and analysis of link communities in networks of arbitrary size and type. *Bioinformatics*, **27**, 2011-2012.

Langfelder, P., Zhang, B., and Horvath, S. (2008) Defining clusters from a hierarchical cluster tree: the Dynamic Tree Cut package for R. *Bioinformatics* 24, 719-720.

Raddichi, F., *et al.* (2004) Defining and identifying communities in networks. *Proc. Natl Acad. Sci USA*, **101**, 2658-2663.

Spencer, R., (2010). http://scaledinnovation.com/analytics/communities/comlinks.html.

Spinelli, L., Gambette, P., Chapple, C. E., Robisson, B., Baudot, A., Garreta, H., Tichit, L., Guenoche, A., and Brun, C. (2013). Clust&See: a Cytoscape plugin for the identification, visualization and manipulation of network clusters. *BioSystems* 113, 91-95.

Yu, H., *et al.* (2008) High-quality binary protein interaction map of the yeast interactome network. *Science*, **322**, 104-110.