A Light Introduction to `lighthergm`

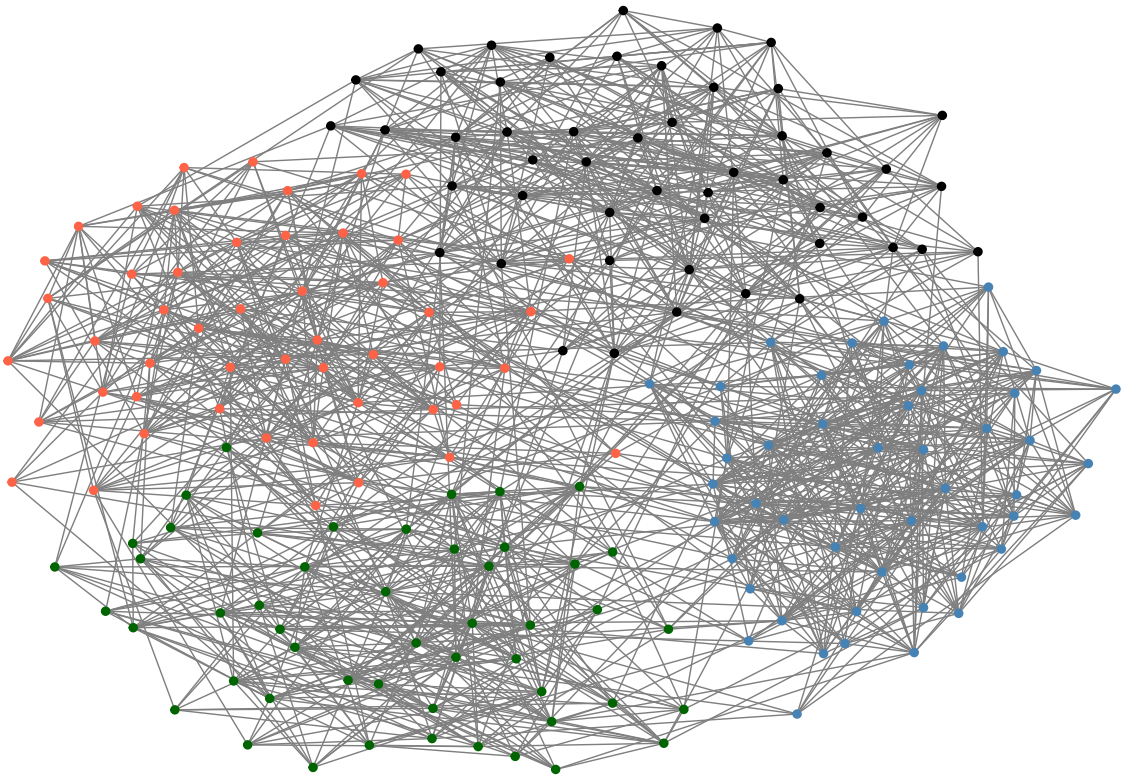
This vignette provides a brief introduction on how to use the R package `lighthergm`, which estimates Hierarchical Exponential-Family Random Graph Models (HERGMs, Schweinberger and Handcock 2015). `lighthergm` is built upon the R package `hergm` (Schweinberger and Luna 2018) and applies scalable algorithms and computational techniques. See Martínez Dahbura et al. (2021) for further information.

A simple example

```
library(lighthergm)
```

`lighthergm` has a toy network to test-drive the package with Let's load the network data and plot it.

```
# Load an embedded network object.
data(toyNet)
# Draw the network.
library(ggplot2)
library(magrittr)
library(GGally)
g <- ggnet2(toyNet,
            size = 1,
            color = rep(c("tomato", "steelblue", "darkgreen", "black"),
                        each = toyNet$gal$n/4),
            mode = "kamadakawai")
plot(g)
```



As you can see, this network has a clear cluster or community structure. Although this is a fake network, we often observe such community structures in real social networks. Exploiting this stylized fact, we model the way agents in a network get connected differently for connections across and within communities:

- Connections across communities happen by luck, influenced by homophily
- Connections within communities also consider interdependencies among links. For example, the probability that agent i and j gets connected may be influenced by a friend in common k.

To estimate the latent community structure of a network and agents' preferences for connection, `lighthergm` implements the following two-step procedure.

1. Recover the community structure by applying a scalable minorization-maximization algorithm.
2. Given the estimated community structure in Step 1, estimate agents' between- and within-community payoffs by maximum pseudo-likelihood estimation.

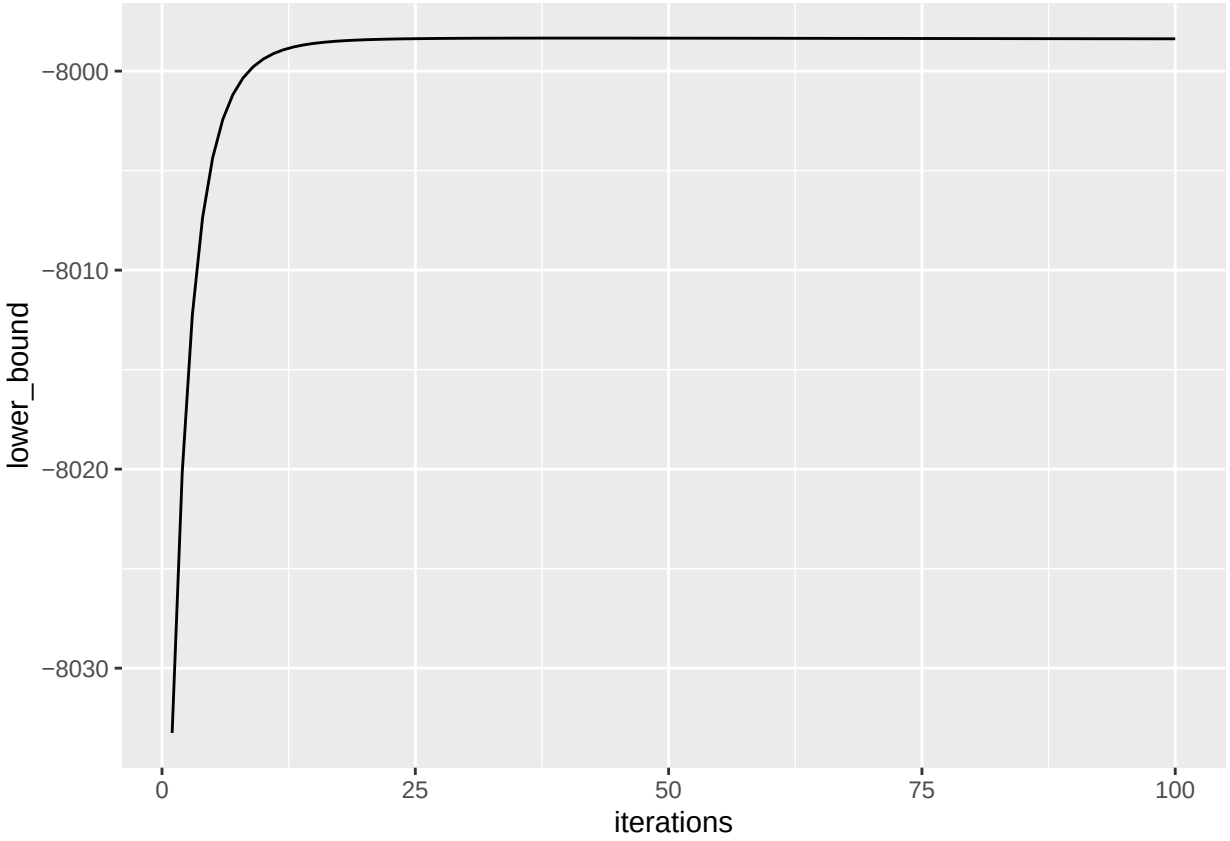
Seeing is believing. Let's perform an estimation using the toy network. (If you would like to see the progress, set `verbose = 1`.)

```
model_formula <- toyNet ~ edges + nodematch("x") + nodematch("y") + triangle

hergm_res <-
  lighthergm::hergm(
    # The model you would like to estiamte
    object = model_formula,
    # The number of blocks
```

```
    n_clusters = 4,
    # The maximum number of EM algorithm steps
    n_em_step_max = 100,
    # Perform parameter estimation after the block recovery step
    estimate_parameters = TRUE,
    # Indicate that clustering must take into account nodematch on characteristics
    clustering_with_features = TRUE,
    # Keep track of block memberships at each EM iteration
    check_block_membership = TRUE
)
```

To see whether the first step (recovering the latent community structure) has converged, we can plot the estimated lower bound of the objective function over iterations.

```
g_LB <-
  ggplot(data = data.frame(iterations = 1:length(hergm_res$EM_lower_bound),
                           lower_bound = hergm_res$EM_lower_bound),
         aes(x = iterations, y = lower_bound)) +
  geom_line()

plot(g_LB)
```



This indicates that the clustering step converged at the early stage. Note that the number of iterations that you need to perform (`n_em_step_max`) varies depending on the size of a network, whether it has a clear community structure, etc.. You need trial and error on how many iterations are at least necessary in your case. Plotting the lower bound may help check the convergence of the clustering step.

You can check the clustering result

```
# Number of nodes per recovered block:
table(hergm_res$partition)
#>
#>  1  2  3  4
#> 50 50 50 50
```

and estimated parameters.

```
# For the between networks
summary(hergm_res$est_between)
#> Call:
#> ergm(formula = between_formula, estimate = "MPLE")
#>
#> Iterations:  NA
#>
#> Maximum Likelihood Results:
#>             Estimate Std. Error MCMC % z value Pr(>|z|)
#> edges       -4.50671    0.07465      0 -60.373   <1e-04 ***
#> nodematch.x  0.79350    0.16086      0   4.933   <1e-04 ***
#> nodematch.y  0.40233    0.18379      0   2.189   0.0286 *
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> For this model, the pseudolikelihood is the same as the likelihood.
#>
#>      Null Deviance: 27587  on 19900  degrees of freedom
#>  Residual Deviance:  2695  on 19897  degrees of freedom
#>
#> AIC: 2701    BIC: 2725    (Smaller is better.)
```

```
# For the within networks
summary(hergm_res$est_within)
#> Call:
#> ergm(formula = formula, constraints = ~blockdiag("block"), offset.coef = offset_coef,
#>     estimate = method_second_step)
#>
#> Iterations:  NA
#>
#> Maximum Pseudolikelihood Results:
#>             Estimate Std. Error MCMC % z value Pr(>|z|)
#> edges       -1.72212    0.06542      0 -26.326   <1e-04 ***
#> nodematch.x  0.54183    0.10602      0   5.111   <1e-04 ***
#> nodematch.y  0.63428    0.10574      0   5.998   <1e-04 ***
#> triangle     0.14674    0.01721      0   8.524   <1e-04 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Warning:  The standard errors are based on naive pseudolikelihood and are suspect.
#>
#>      Null Pseudo-deviance: 6793  on 4900  degrees of freedom
#>  Residual Pseudo-deviance: 5242  on 4896  degrees of freedom
```

```
#>
#> AIC: 5250    BIC: 5276    (Smaller is better.)
```

Currently, the only supported way to include covariates in the model is via `nodematch()`.

Goodness-of-fit

You can evaluate the goodness-of-fit of the model with the `lighthergm::gof_lighthergm()` function:

```
# Prepare a data frame that contains nodal id and covariates.
nodes_data <-
  tibble::tibble(
    node_id = network::network.vertex.names(toyNet),
    block = hergm_res$partition,
    x = network::get.vertex.attribute(toyNet, "x"),
    y = network::get.vertex.attribute(toyNet, "y")
    )

# The feature adjacency matrices
list_feature_matrices <- lighthergm::get_list_sparse_feature_adjmat(toyNet, model_formula)

# The MCMC settings
sim_ergm_control <- ergm::control.simulate.formula(
  MCMC.burnin = 1000000,
  MCMC.interval = 100000
)

# The feature adjacency matrices
list_feature_matrices <- lighthergm::get_list_sparse_feature_adjmat(toyNet, model_formula)

gof_res <- lighthergm::gof_lighthergm(
  toyNet,
  # The feature adjacency matrices
  list_feature_matrices = list_feature_matrices,
  # A dataframe containing the nodes data.
  data_for_simulation = nodes_data,
  # The name of the nodes_data column containing the node IDs
  # which are used within the network g
  colname_vertex_id = 'node_id',
  # The name of the nodes_data column containing the block ID.
  colname_block_membership = 'block',
  # The object returned by lighthergm::hergm()
  lighthergm_results = hergm_res,
  # The MCMC settings
  ergm_control = sim_ergm_control,
  # The number of simulations to use
  n_sim = 100
)
```

Currently, gof is evaluated on the following metrics:

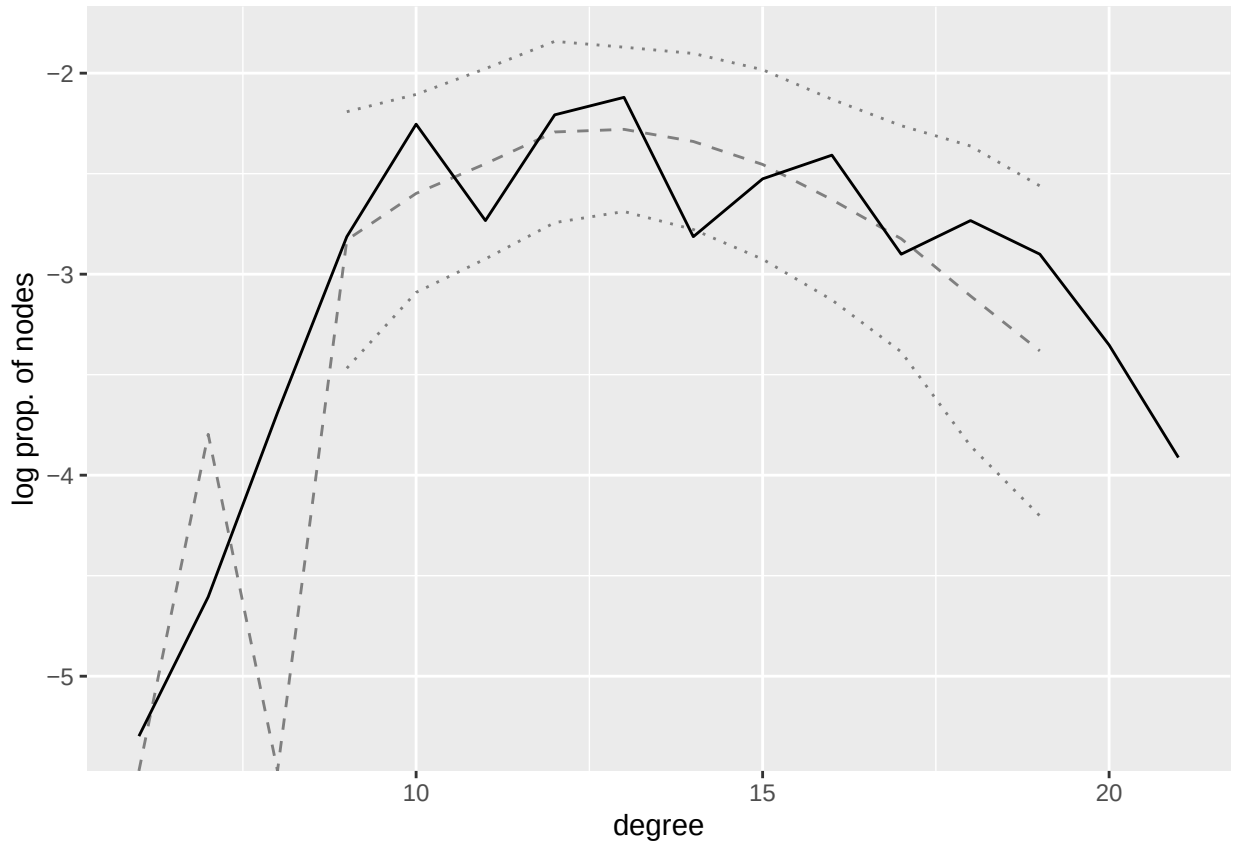
1. the network statistics (the counts you obtain when you use summary on an ergm formula, such as the number of edges, triangles, nodematches, etc.),

2. degree distribution
3. geodesic distance, and
4. edgewise shared partners.

`lighthergm::gof_lighthergm()` returns a list of data frames for these matrices instead of creating plots as `ergm::gof()` does. This allows you to flexibly create gof plots that match your needs.

Below is a example gof plot on degree distribution.

```
degree_gof <-
  gof_res$simulated$degree_dist %>%
  dplyr::group_by(degree) %>%
  dplyr::summarise(log_mean_share = mean(log(share)),
                   log_sd_share = sd(log(share))) %>%
  dplyr::ungroup()

plot_degree <-
  ggplot(data = degree_gof %>%
           dplyr::filter(degree < 20 & degree >= 6),
         aes(x = degree, y = log_mean_share)) +
  geom_line(aes(y = log_mean_share + 1.96 * log_sd_share),
            colour = "grey50",
            linetype = "dotted") +
  geom_line(aes(y = log_mean_share - 1.96 * log_sd_share),
            colour = "grey50",
            linetype = "dotted") +
  geom_line(colour = "grey50",
            linetype = "dashed") +
  geom_line(data = gof_res$original$degree_dist %>%
              dplyr::filter(share > 0 & degree < 22),
            aes(y = log(share))) +
  ylab("log prop. of nodes")

plot(plot_degree)
```



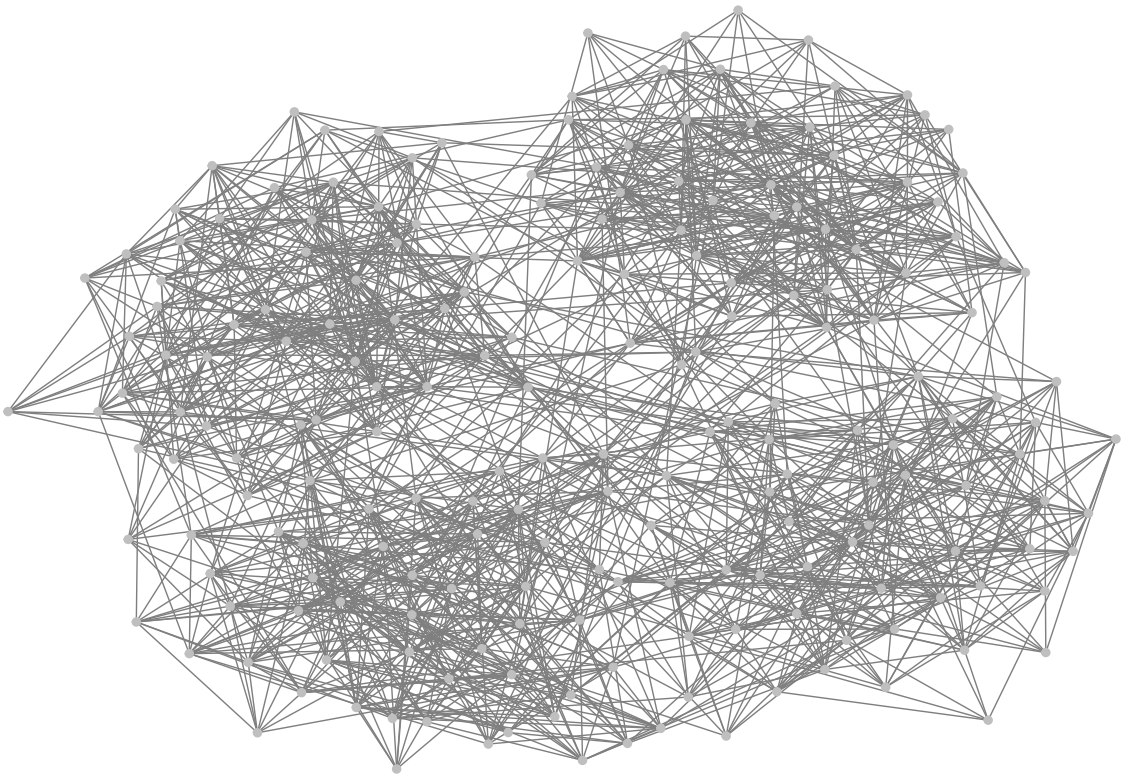
Simulation

You can simulate networks with local dependence using the `lighthergm::simulate_hergm()`.

```
# Estimated coefficients for the between-community connections
coef_between_block <- hergm_res$est_between$coef

# Estimated coefficients for the within-community connections
coef_within_block <- hergm_res$est_within$coef

sim_net <- lighthergm::simulate_hergm(
  # Formula for between-blocks
  formula_for_simulation = model_formula,
  # Same as for gof, a dataframe containing nodes attributes
  data_for_simulation = nodes_data,
  # Name of the column containing node IDs
  colname_vertex_id = "node_id",
  # Name of the column containing block IDs
  colname_block_membership = "block",
  # The coefficients for the between connections
  coef_between_block = coef_between_block,
   # The coefficients for the within connections
  coef_within_block = coef_within_block,
  # The MCMC settings
```

```
  ergm_control = sim_ergm_control,
  # Number of simulations to return
  n_sim = 1,
  # If `stats` a list with network statistics
  # for the between and within connections is returned
  output = "network",
  # Simulates between connections by drawing from a logistic distribution.
  # If FALSE, draws between connections by MCMC.
  use_fast_between_simulation = TRUE,
  # The feature adjacency matrices
  list_feature_matrices = list_feature_matrices
)
```

```
ggnet2(sim_net, size = 1, mode = "kamadakawai")
```



When you work with large networks

If you would like to estimate an HERGM with a large network (say, when the number of nodes $\geq$ 50,000):

- Select features sparse enough to fit into memory. Covariates such as gender or race will be too dense to construct feature matrices. This is a non-negligible limitation of our algorithm and will be solved in the future.

- Prepare a list of multiplied feature adjacency matrices by `lighthergm::compute_multiplied_feature_matrices()`, and pass it to `lighthergm::hergm()` by `list_multiplied_feature_matrices`. Once calculated and stored, it can be used in models with the same network and the same features.
- Use Python's infomap to initialize clusters. This is because it is much faster to implement cluster initialization than R functions such as `igraph::cluster_infomap()`. To install it, run `system("pip3 install infomap")` and check if it is effective by `system("infomap --version")`. If `system("infomap --version")` yields an error, consider using `{reticulate}`.
- If successfully installed Python's infomap, set `use_infomap_python = TRUE` in `lighthergm::hergm()`.
- When the EM estimation does not seem to have converged by inspecting the lower bound plot, you can further continue iterating by passing the `lighthergm` class object to `lighthergm::hergm()` as follows (all parameters such as the number of EM iterations will be inherited from the previous estimation unless specified).

```
hergm_res_second <-
  lighthergm::hergm(object = hergm_res)
```

References

Martínez Dahbura, Juan Nelson, Shota Komatsu, Takanori Nishida, and Angelo Mele. 2021. "A Structural Model of Business Card Exchange Networks," Working Paper. Available at https://arxiv.org/abs/2105.12704.

Schweinberger, Michael, and Mark S Handcock. 2015. "Local Dependence in Random Graph Models: Characterization, Properties and Statistical Inference." *Journal of the Royal Statistical Society B* 77 (3): 647–76.

Schweinberger, Michael, and Pamela Luna. 2018. "Hergm: Hierarchical Exponential-Family Random Graph Models." *Journal of Statistical Software* 85 (1): 1–39.