CODE MANUAL

Simulation and estimation of Exponential Random Partition Models

ARTICLE HISTORY
Compiled September 29, 2020

1. File structure

All relevant R files are available in the repository https://github.com/marion-hoffman/ERPM. An example script displaying all possible uses of the code is found in the Rmarkdown file Example_script.Rmd. The functions necessary to estimation, simulation, and diverse calculations are found in the following files:

- **functions_estimate.R**: contains the parent function for the estimation *estimate_ERPM*,
- **functions_exactcalculations.R**: contains functions to calculate average sizes and likelihood functions whenever it is possible,
- **functions_phaseX.R**: contains the functions *run_phaseX* for the first, second, or third phase of the algorithm,
- **functions_Metropolis.R**: contains the function *draw_Metropolis* used for the Metropolis sampler, and auxiliary functions,
- **functions_change_statistics.R**: contains the function *computeStatistics* that calculates for a given sufficient statistic the value of this statistic for a given partition,
- **functions_loglikelihood.R**: contains the functions used for estimation of the log-likelihood and AIC, including the calculation of the log likelihood for a simple Dirichlet model (only one effect being the number of groups),
- **functions_output.R**: contains the function *draw_Metropolis* used for the Metropolis sampler, and auxiliary functions,
- **functions_utility.R**: contains the function *draw_Metropolis* used for the Metropolis sampler, and auxiliary functions,

2. Model specification

2.1. Objects definition

- Node attributes should be contained in a data.frame, with each column being an attribute (integer or character).
- Networks should be simple full matrices.
- Effects should be a list with first element called "names" including a list of effect names, and a second "objects" element containing the name of the object needed to calculate the effects (for example, column name in the nodeset if it's an attribute effect, or name of a matrix).

- We add objects (matrices) into another list, with "name"and "object" attribute (to recover which matrix to use in a network effect for example).

2.2. Parametrization

For now the following structural effects are implemented:

- **isolates**: number of isolates
- **num_groups**: number of groups
- **num_groups_3**: number of groups of size 3
- **num_groups_4**: number of groups of size 4
- **num_groups_5**: number of groups of size 5
- **sizes_squared**: sum of squared sizes
- **product_sizes**: product of all group sizes
- **sum_log_factorials**: sum of logarithms of group sizes
- **num_ties**: number of group intra-ties (ties are between individuals in the same group, not related here to another network)
- **num_triangles**: number of group intra-triangles
- **num_fours**: number of group intra-four-cliques
- **num_fives**: number of group intra-five-cliques
- **alt_cliques**: alternated weighted sum of intra-tie-cliques (solves near-degeneracy)
- **degree2**: sum of intra-tie degrees, squared
- **av_degree**: average intra-tie degree
- **av_degree2**: average intra-tie degree, squared

And the (dyadic or individual) attribute effects are:

- **tie**: sum of group intra-ties for a given network
- **attisolation**: sum of the attributes of isolated people
- **attgroups**: sum of the attributes of in-groups people
- **alter**: sum of individual attributes multiplied by their group size
- **attisolation**: sum of the attributes of isolated people
- **same**: sum of the number of dyads in groups having the same attribute
- **diff**: sum of the absolute differences in attributes for people in the same groups
- **number_attributes**: sum of different attributes in a group, summed over all groups

3. Simulation

- The main version of the algorithm uses the option for mini-steps "normalized", meaning that we always propose a new partition, but we multiply the Hastings ratio with the ratio of possible neighbor partitions between the old and the newly sampled partition. If the option is set to "self-loops" we have the option of staying in a same partition to keep the number of possible neighbours equal. Right now this second one might have bugs and seems less efficient.
- When the options for allowed sizes and simulated sizes are given, the Metropolis sampler goes through the partitions allowed by the simulated sizes option, but samples only the ones that have sizes in the allowed sizes. This has to be done when the space of allowed partitions is disconnected or ill-connected to cover better the space. It can be done by adding the sizes just smaller or bigger than the limits, and it often helps to allow singletons.
- Right now only size restrictions with a minimum size and a maximum size work!!
- The neighborhood option is set to 1, 2, or 3. Neighborhood 1 means we move from one partition to the next by swapping two actors in different groups (possibly isolates). Neighborhood 2 means we move from one partition to the next by merging two groups, or by dividing one group into two. Neighborhood 3 is a 50/50 mix of neighborhoods 1 and 2. For now the second alone seems more efficient, the first shouldn't be used alone, and maybe the third is nice if there are difficulties browsing the space of possible distributions for attribute effects.
- Sometimes, it will be difficult to sample if the model favours partitions having sized within the simulated sizes option but not within the allowed sizes option. In this case it can useful to change the option of simulated sizes.

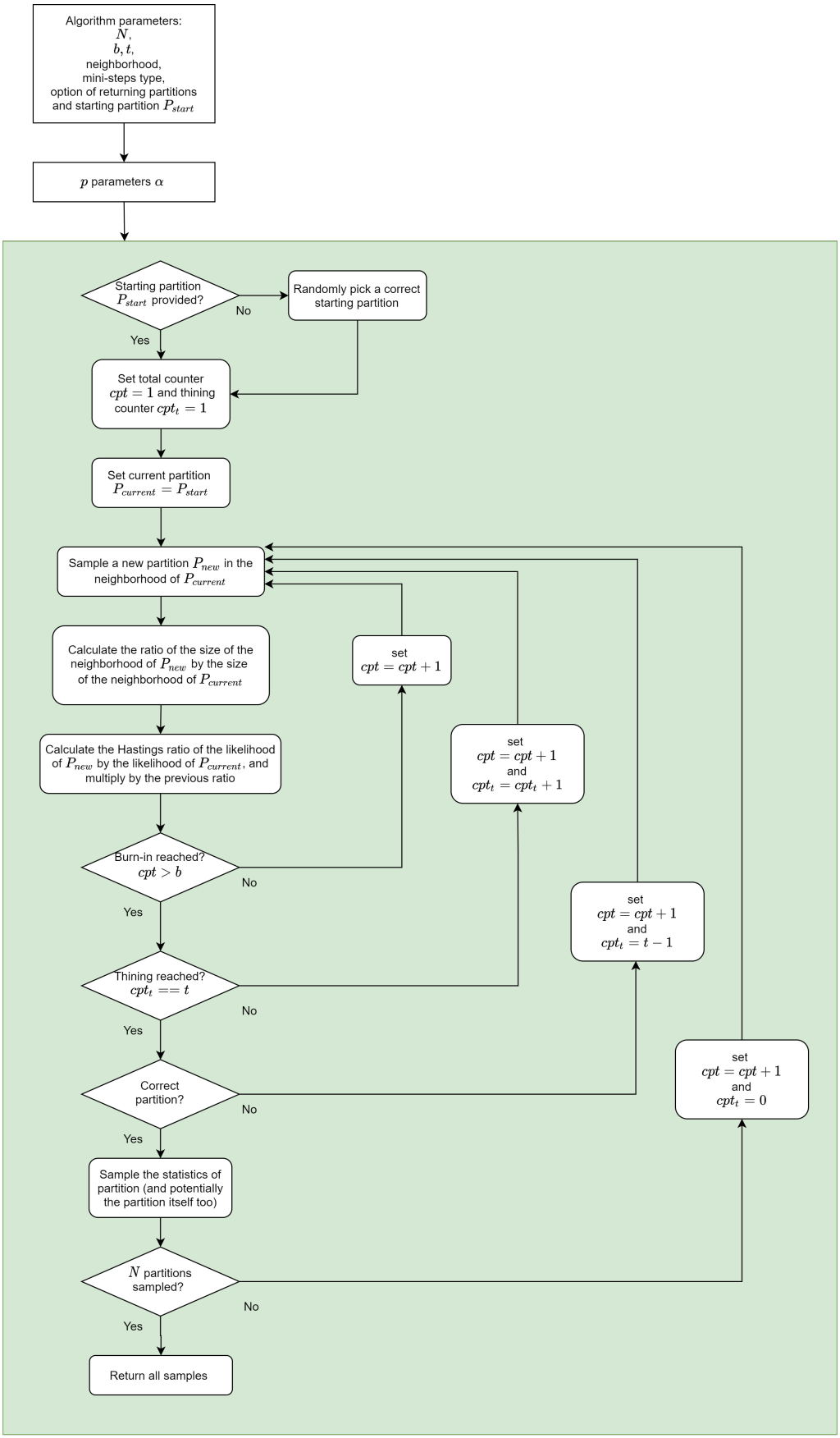

Figure 1. Flowchart for sampling partitions with the Metropolis-Hastings algorithm.

4. Estimation

4.1. General algorithm

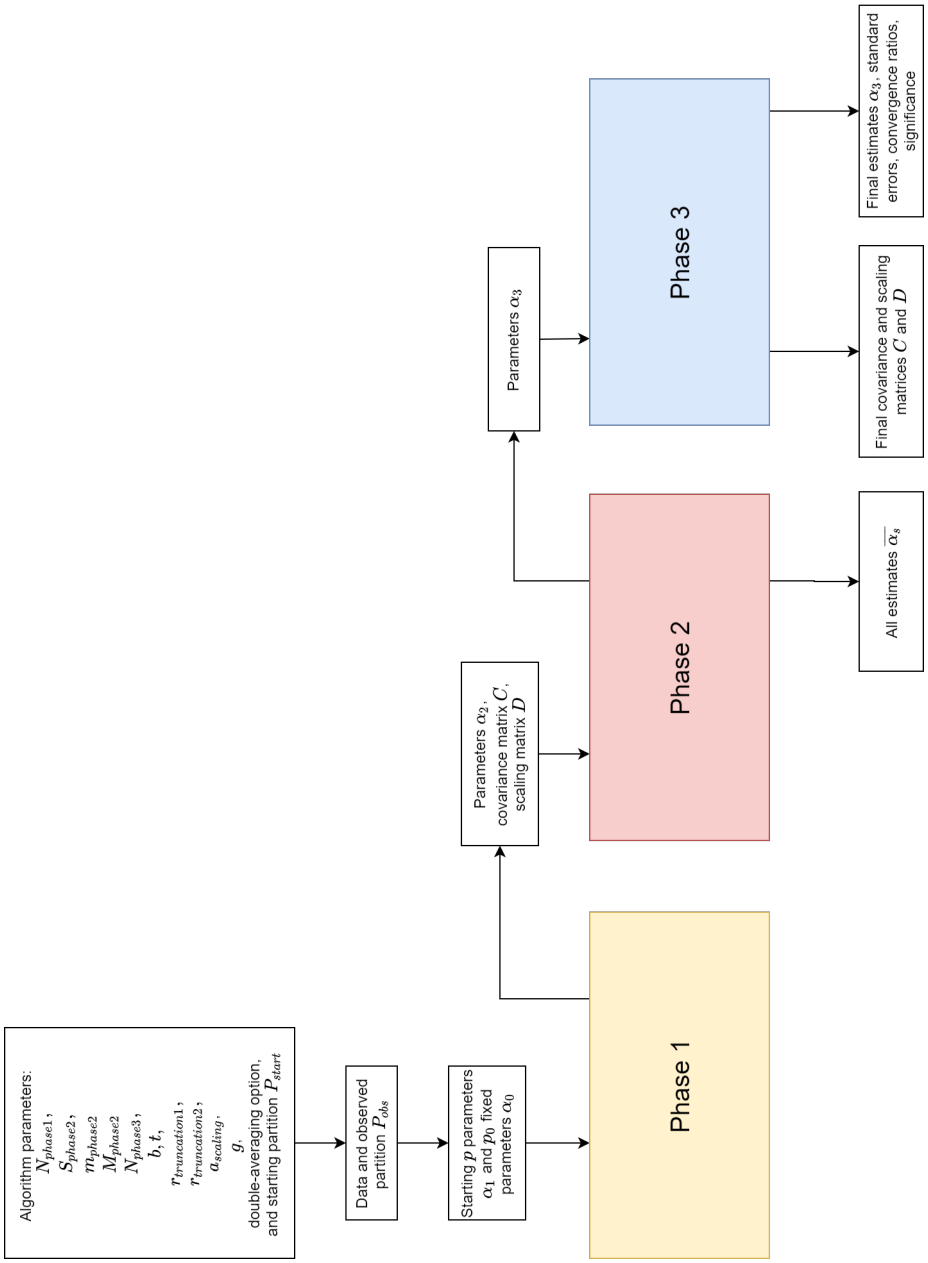

Figure 2. Flowchart for the general estimation algorithm.

4.2. Algorithm for phase 1

- The starting partition has to respect the allowed sizes given to the estimation.
- The length of phase 1 does not need to be very long, the calculation of the co-variance is not very sensitive.
- The variable $a_{scaling}$ is used to reduce the influence of non diagonal elements of the scaling matrix (before inverting it!). If it's zero, the estimation in phase 2 might fluctuates less around the parameter space, so it helps keeping things under control, but it might not converge exactly to what we want.
- The truncating factor $r_{truncation1}$ helps avoiding making too large steps when estimating the parameter in case some elements in the co-variance matrix are too large; the convergence will be slower but safer.
- Co-variance and scaling matrices are inverted once for all here.
- It's better to skip this phase if we've already ran the estimation once and have better estimates and co-variance matrix from a previous phase 3 anyway.

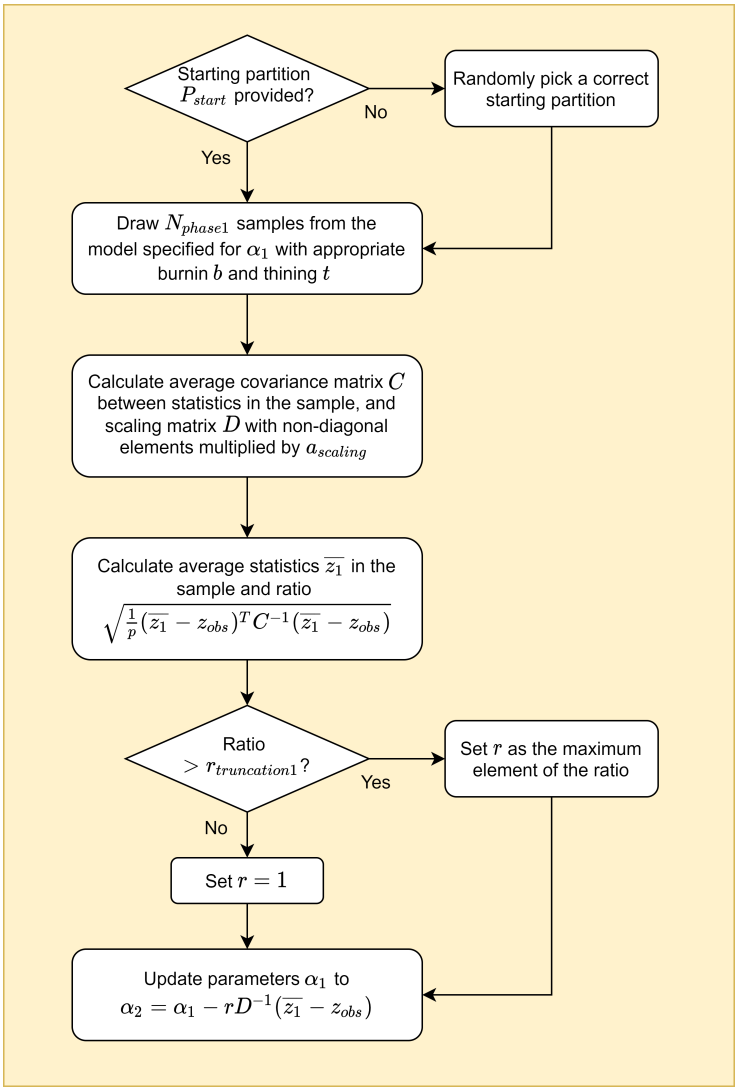

Figure 3. Flowchart for Phase 1 of the main algorithm.

4.3. Algorithm for phase 2

- For each step we start from a random matrix (unless we provided a starting partition), but from one sub-step to the next we keep a partition generated by the previous chain.
- Again the truncating factor $r_{truncation2}$ helps avoiding making too large steps when estimating the parameter in case some elements in the co-variance matrix are too large; the convergence will be slower but safer. This is always calculated from the co-variance and note the scaling matrix!
- Double-averaging can be used to average the estimates over a step AND over all sub-steps of one step. It can sometimes stabilize the estimation procedure. Double-averaging is only when there are no fixed parameters for now.
- A step ends when along all sub-steps we sampled statistics going above AND below the observed statistics (that's what we call statistics "crossing"). Additionally, the parameters m_{phase2} help controlling that each step lasts a bit but not too long even if statistics do not cross. The minimum length is $m_{phase2}(2.52)^k$ and $m_{phase2}(2.52)^k+200$ (so the subphases get longer and longer, without threatening the convergence of the Robbins-Monro procedure).
- When we have a bad starting estimates for the parameters, it can help to have many steps, but when we are close we can reduce to a couple of them, potentially with long sub-steps.

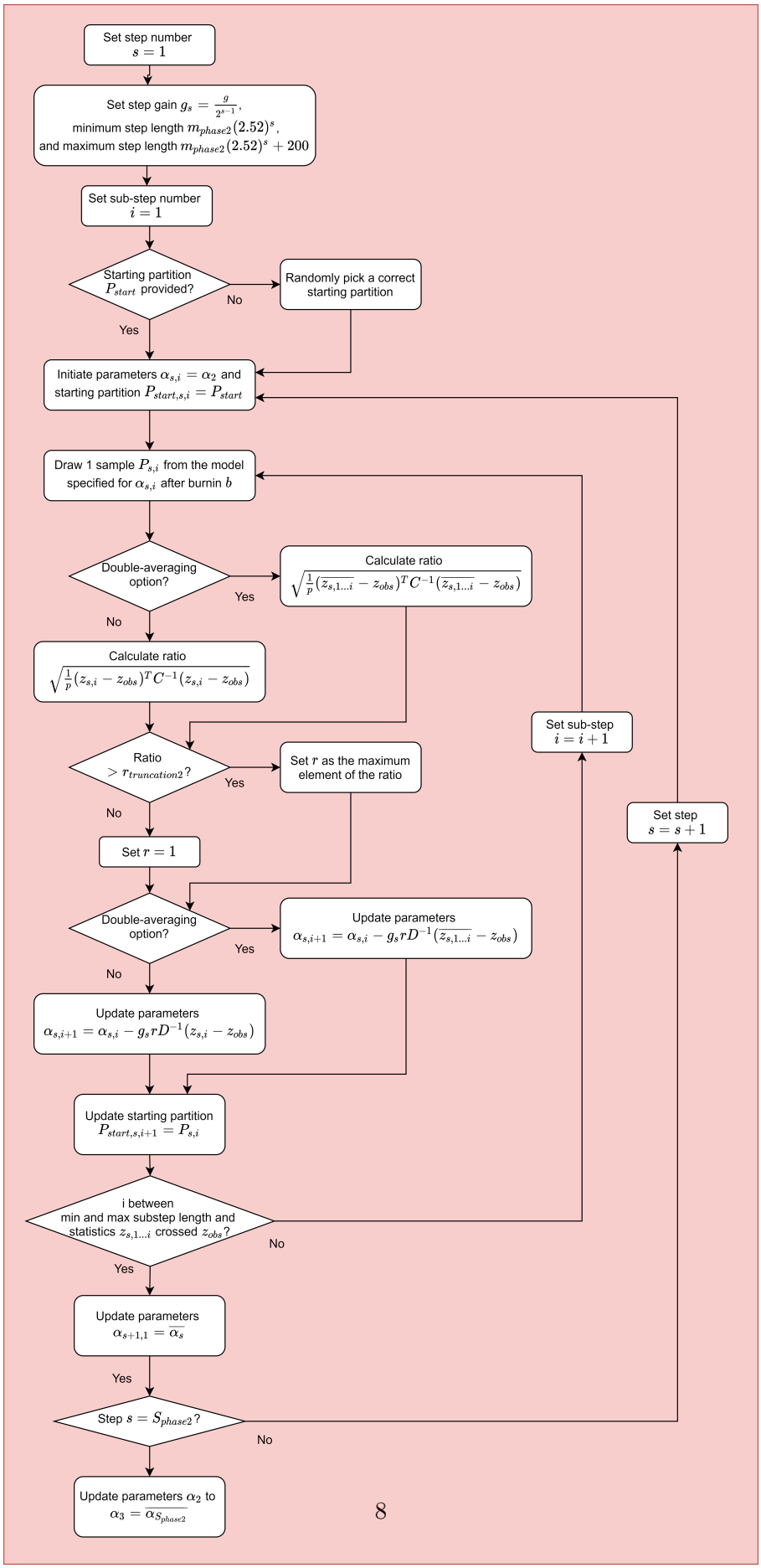

Figure 4. Flowchart for Phase 2 of the main algorithm.

4.4. Algorithm for phase 3

- It helps to keep the co-variance and scaling calculated from this step to a next estimation.
- When we are close to a good estimate for the parameters, it helps to have a very large sample in this phase, to estimate correctly standard errors and convergence ratios.

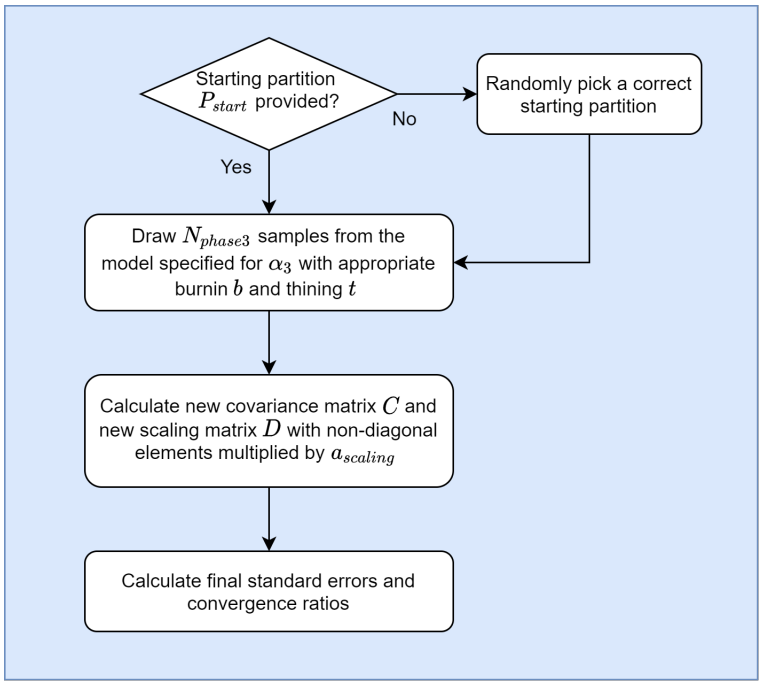

Figure 5. Flowchart for Phase 3 of the main algorithm.

5. Model diagnostics

5.1. Goodness of fit

The example script shows a few examples on how to calculate and visualize the fit of the model regarding certain statistics. It makes use of the simulation engine detailed earlier.

5.2. Estimation of a log-likelihood and AIC

- This procedure can take a while, because each step requires a good sample (just like phase 3), and in the case of restricted sizes sampling is not that easy.
- This can only work if the model include the statistic number of groups, because we can calculate easily the exact likelihood of a model including only this statistic (basically a Dirichlet partition model).
- The α_0 parameters should be a vector of zeros, except for the number of groups effect that should contain the exact estimate (easily found with an optimisation function and using the utility functions to calculate the denominator of Dirichlet partition models).
- It helps to plot the distributions of sampled statistics in the different steps, and follow the "frog-leaping"strategy: the algorithm only works well if the jump from one step to the next is reasonable, meaning if the distributions of statistics have a sufficiently large overlap.

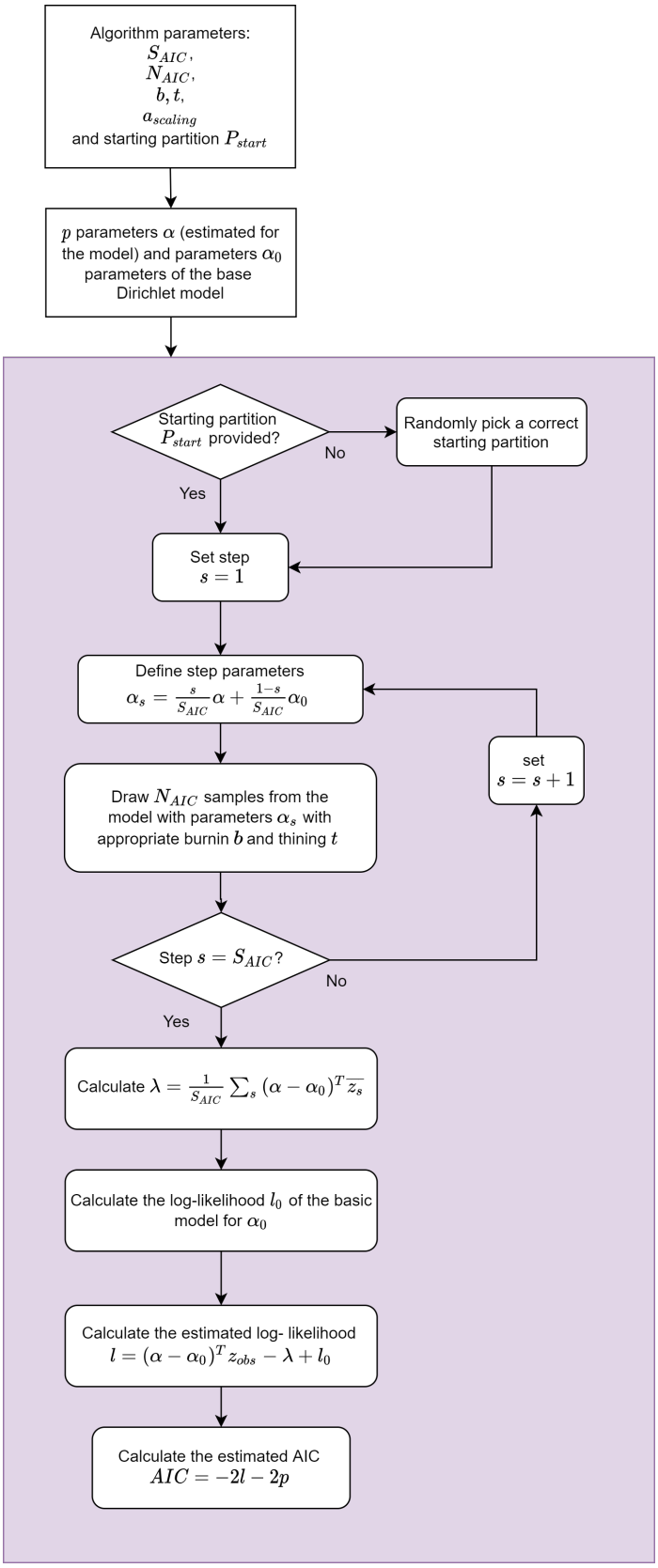

Figure 6. Flowchart for the path-sampling algorithm to estimate the log-likelihood of a given model.

6. Notes

The procedure is heavily inspired from procedures used for Exponential Random Graph Models and the three main references below, and relies on a large amount of great advice from Tom A.B. Snijders.

References

Hunter, D. R., Goodreau, S. M., & Handcock, M. S. (2008). Goodness of fit of social network models. *J. Am. Stat. Assoc.*, *103*(481), 248–258.

Lusher, D., Koskinen, J., & Robins, G. (2013). *Exponential random graph models for social networks: Theory, methods, and applications*. Cambridge University Press.

Snijders, T. A. B. (2002). Markov chain monte carlo estimation of exponential random graph models. *J. Soc. Biol. Struct.*, *3*(2), 1–40.