# `netrb`: Network Robustness Simulation Functions

Nicholas "Kiko" Whiteley

## Purpose

Though a network's attributes can reveal much about its robustness, direct experimentation on a given graph may also provide useful information in light of, for example:

- Niche metric(s) of interest.
- Non-applicability of abstract robustness considerations, or lack of familiarity thereof.
- Verification of abstract inferences.
- Curiosity.

`netrb` aims to facilitate this experimentation with minimal user-end code.

Its main simulation function, `netrb::metric_vs_del()`, records the effect of iterated *structural changes* (currently, vertex deletions) on the value of a *metric of interest* in response to such deletions, simulated via an `igraph` object. Multiple vertex deletion orders ("VDOs") can be simulated concurrently, and the metric of interest is passed in as any function compatible with an `igraph` object.

Simulations on larger networks entail potential limitations to keep runtimes reasonable, in light of which `netrb` provides customizability, parallelization, and progress monitoring.

## Simulation overview

`metric_vs_del()` returns a `base::data.frame` containing:

- A single "X" column denoting the fraction of total vertices deleted.
- One "Y" column for each VDO, containing response metric values recalculated after each corresponding fraction of vertex deletions according to the column's VDO.
- One row for each "chunk" of deletions (passed as `nchunks`), plus an initial row before deletions.

Results can also be plotted with `netrb::plot_Ys_vs_X()`. #### Additional parameters

| Name | Default | Description | Definition |
|---|---|---|---|
| `del_min` | `0` | Minimum deletion level | Starting fraction or number of nodes deleted. |
| `del_max` | `0.5` | Maximum deletion level | Ending fraction or number of nodes deleted. |
| `nchunks` | `20` | Number of deletion chunks | Total number of sets of deletions to occur in the simulation. |
| `recalc_VDOs` | `FALSE` | | Whether or not to recalculate VDOs after each set of deletions. |

### Keeping runtimes reasonable with large graphs

**Be aware of distant-dependent functions** Many graph operations have constant or linear time complexity with respect to vertex and/or edge count. For example, retrieving the total number of vertices, each vertex's degree, or even more sophisticated vertex attributes like Google's PageRank, complete within a few seconds

at most, even with millions of vertices.

Other operations, however, have quadratic or cubic time complexity with respect to vertex and/or edge count – most notably, those involving direct calculation of many or all shortest paths – and can become unreasonably slow for vertex counts well under a hundred thousand.

### Suggested limitations

- A distance-independent response metric such as largest component size, number of components, or mean degree.
- Distance-independent VDOs, determined, for example, by degree, PageRank, or randomly.
- One-time calculation of VDOs, by passing `recalc_VDOs == FALSE`. This requires a `simulator` object.
- A low number of deletion chunks, low maximum deletion level, or narrow total deletion window (max - min).

To help monitor and reduce runtimes, `netrb` also implements, by default, parallelization via `parallel::mclapply()` and progress bar printing via the `pbapply` package.

## Running simulations

### Simulator setup (optional; highly recommended)

Though `metric_vs_del()` can be called without passing a `netrb::simulator` instance, creating a simulator (S3 class) via `netrb::simulator()` and passing it into simulation calls has several potential benefits, including consistency/reproducibility, convenience, and potential performance improvements,

### Other considerations

**Unconnected distance** With extensive or even moderate vertex deletions, the original graph will generally fracture into smaller components, leaving many once-connected vertices unconnected.

Thus, if the response metric involves distance, *unconnected distance* (i.e., the value assigned to denote the "distance" between two unconnected vertices), though ultimately arbitrary, becomes more important.

One approach, as in `igraph::mean_distance(..., unconnected = TRUE)` is to ignore all unconnected distances from the mean calculation, generally leading to lower mean distance as the graph fractures into smaller components.

Passing `unconnected = FALSE`, or calculating distances without averaging them, as in `igraph::distances()`, assigns an unconnected distance of `INF`. This makes the mean distance also `INF`, which may not be very helpful in the context of `metric_vs_del()`.

For example, when designing or modifying a road network, one may wish to maximize connectivity between vertices (which typically represent intersections) in the face of vertex deletions (which could represent temporary or permanent blocking of intersections), and thus seeks to minimize the graph's mean distance therein.

Ignoring unconnected distances would not be helpful, since it would incentivize many unconnected road clusters, contradicting the goal of connectivity, and an unconnected distance of `INF` would render the response metric useless once the graph has two or more components.

One alternative approach is to assign an unconnected distances of the graph's diameter$+1$, as it disincentivizes unconnected components while leaving mean distance calculations possible.

Thus, the default unconnected distance in `netrb::distances()` and `netrb::mean_distance()` is the original graph's diameter $+ 1$, which is stored to in `simulator` if used to remain constant during `metric_vs_del()`. All `netrb` functions allow a custom unconnected distance to be specified.