Tutorial

Aleksei Krasikov

2018-06-22

This vignette document sets the following goals:

1. point out the discrepancy between the descriptions of algorithms in the original article and in the package
2. provide several examples of working with the library
3. provide a workflow to reproduce results of the original article (since several function was slightly changed in version 3.0)

Note, that here you do not find detailed explanation of all algorithms. For that purposes, see the original article.

Steiner tree problem in graphs

The Steiner tree problem on unweighted graphs seeks a minimum subtree (i.e. subtree with minimal number of edges), containing a given subset of the vertices (terminals). This problem is NP-complete. This package provides several heuristic and one exact approach for finding Steiner trees, as well as tools for analyzing resultant trees and comparing different algorithms. This R package was originally applied to analyzing biological networks.

Heuristic approaches:

1. shortest paths based approximation (SP)
2. minimum spanning tree based approximation (KB)
3. randomized all shortest paths approximation (RSP)
4. all shortest paths between terminals (ASP)
5. (SPM)

Before we start, let's attach several packages.

```
library(igraph)
library(SteinerNet)
```

As an example of graph, we are going to take well-known "Cubical" graph. Also let's randomly pick 4 terminals using `generate_st_samples`. We also specify `prob` variable, so at each step of random walk procedure node is accepted with probability `prob = 0.1`.

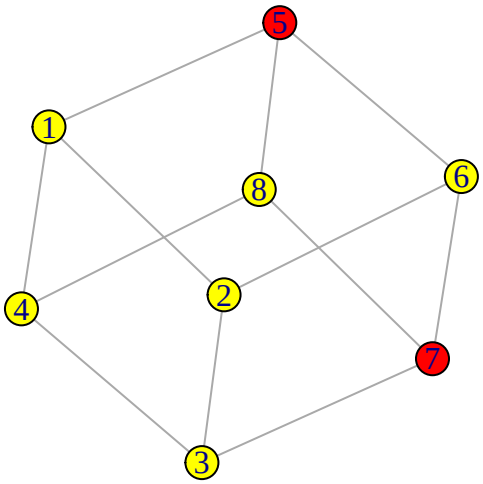
```
g <- graph("Cubical")

set.seed(4)
terminal_nodes <- generate_st_samples(graph = g, ter_number = 2, prob = 0.1)
terminal_nodes
#> [[1]]
#> [1] 5 7
```

As we can see, there is only one element in a list and it contains ids of selected vertices. Of course, we can generate more sets of terminals (e.g. pass a vector to ter_number variable), but we do not need it now.

```
V(g)$color <- "yellow"
V(g)[terminal_nodes[[1]]]$color <- "red"

plot(g)
```



Shortest paths based approximation (SP)

`repeattimes`, and `merge` variables are ignored for "SP".

```
steinertree(type = "SP", terminals = terminal_nodes[[1]],
            graph = g, color = FALSE, merge = FALSE)
#> [[1]]
#> IGRAPH c15fe09 UN-- 3 2 -- Cubical
#> + attr: name (g/c), color (v/c), name (v/c)
#> + edges from c15fe09 (vertex names):
#> [1] 5--6 6--7
```

Note, that in 3.0 version (as well as in 2.0 version) Steiner trees with different randomly chosen start nodes are not repetitively constructed, as it is written in article. This can be done by hand as follows:

```
tree_list <- c()

for (i in 1:20)
        tree_list[[i]] <- steinertree(type = "SP", terminals = terminal_nodes[[1]],
                                      graph = g, color = FALSE, merge = FALSE)

# calculate sizes of trees
tree_list_len <- unlist(lapply(tree_list, function (x) length(E(x[[1]]))))

# select trees with minimal size
index <- which(tree_list_len == min(tree_list_len))
```

Minimum spanning tree based approximation (KB)

`repeattimes`, and `merge` variables are ignored for "KB".

```
steinertree(type = "KB", terminals = terminal_nodes[[1]],
            graph = g, color = FALSE, merge = FALSE)
#> [[1]]
#> IGRAPH cec83e3 UN-- 3 2 -- Cubical
```

```
#> + attr: name (g/c), color (v/c), name (v/c)
#> + edges from cec83e3 (vertex names):
#> [1] 5--8 7--8
```

Randomized all shortest paths approximation (RSP)

`merge` variable is ignored for "RSP".

```
steinertree(type = "RSP", terminals = terminal_nodes[[1]],
            graph = g, color = FALSE, merge = FALSE)
#> [[1]]
#> IGRAPH aedc020 UN-- 3 2 -- Cubical
#> + attr: name (g/c), color (v/c), name (v/c)
#> + edges from aedc020 (vertex names):
#> [1] 5--8 7--8
```

All shortest paths between terminals (ASP)

This method is provided **only for comparison reasons**. In version 2.0 it was hidden, however it is now available for the sake of convenience.

`repeattimes`, and `merge` variables are ignored for "ASP".

```
steinertree(type = "RSP", terminals = terminal_nodes[[1]],
            graph = g, color = FALSE, merge = FALSE)
#> [[1]]
#> IGRAPH 0754784 UN-- 3 2 -- Cubical
#> + attr: name (g/c), color (v/c), name (v/c)
#> + edges from 0754784 (vertex names):
#> [1] 5--6 6--7
```

Exact algorithm (EXA)

Note, that this method can find multiple exact solutions, if they exist. In the following example we specify `merge` variable equals to `FALSE`, because we want to get a list of steiner trees.

`repeattimes` and `optimize` variables are ignored for "EXA".

```
steinertree(type = "EXA", terminals = terminal_nodes[[1]],
            graph = g, color = FALSE, merge = FALSE)
#> [[1]]
#> IGRAPH bd95f26 UN-- 3 2 -- Cubical
#> + attr: name (g/c), color (v/c), name (v/c)
#> + edges from bd95f26 (vertex names):
#> [1] 5--6 6--7
#>
#> [[2]]
#> IGRAPH dbcb3b0 UN-- 3 2 -- Cubical
#> + attr: name (g/c), color (v/c), name (v/c)
#> + edges from dbcb3b0 (vertex names):
#> [1] 5--8 7--8
```

As we can see, two steiner trees are found.

Sub-graph of merged steiner trees (SPM)

"SPM" algorithm can return multiple graphs. Note, that in article this method is called "STM".

```
steinertree(type = "SPM", terminals = terminal_nodes[[1]],
            graph = g, color = FALSE, merge = FALSE)
#> [[1]]
#> IGRAPH 108f771 UN-- 3 2 -- Cubical
#> + attr: name (g/c), color (v/c), name (v/c)
#> + edges from 108f771 (vertex names):
#> [1] 5--8 7--8
#>
#> [[2]]
#> IGRAPH db858c5 UN-- 3 2 -- Cubical
#> + attr: name (g/c), color (v/c), name (v/c)
#> + edges from db858c5 (vertex names):
#> [1] 5--6 6--7
```

Optimization of resultant tree

Optimization means returning the minimum spanning tree on resultant graph and removing all non-terminal nodes of degree one. **Note, that in version 2.0 this function was runned by default for several algorithms.** Again it is explicitly available in version 3.0 for the sake of convenience.

If you want to reproduce results of the article, you need to specify `optimize = TRUE` for the following algorithms:

- "SP"
- "KB"
- "RSP"
- "SPM"

For "EXA" algorithm this option is ignored. **Note, that in version 2.0 for "ASP" algorithm optimization was not avaliable at all.** It means, that experiments was condacted without further optimization.

Article results reproducibility

For experiments terminal sets of the following size are used:

- 50 sets with 5 randomly selected terminals
- 50 sets with 8 randomly selected terminals
- etc.

All vertices was selected with 0.5 probability. Therefore in `generate_st_samples` you probably passed the following:

```
# in version 2.0
# eval = FALSE
listofterminaltest <- c(5, 8, 15, 50, 70)
repetition <- rep(x = 0.5, 50)
```

However, simultaneous work with the number of terminals selected and size of sets makes an output of some function too wired. So now you need to specify:

- `ter_number`. Each element indicates the number of terminals to be selected and length of vector indecates the number of terminal sets to be picked.

- `prob`. prob[i] defines a propability with which each next node accepted or rejected while selecting ter_number[i] terminals. Usually this probability is the same for all terminals, so you may write something like this `prob = rep(0.5, #len of ter_number#)`

The main difference is that you can now operate with **only one value** of number of terminal sets. So to reproduce results of the article, firstly, you need, for example, generate 50 terminal sets with 5 terminals in each.

```
# in version 3.0
# eval = FALSE
generate_st_samples(graph = #your graph#,
                    ter_number = rep(x = 5, 50),
                    prob = rep(x = 5, 50))
```

After running some simulations, secondly, you may want to generate 50 terminal sets with 8 terminals in each.

```
# in version 3.0
# eval = FALSE
generate_st_samples(graph = #your graph#,
                    ter_number = rep(x = 8, 50),
                    prob = rep(x = 8, 50))
```