The Journal of Open Source Software

yenpathy: An R Package to Quickly Find K Shortest Paths Through a Weighted Graph

Toph Allen[1] **and Noam Ross**[1]

1 EcoHealth Alliance, New York, NY, USA

DOI: 10.21105/joss.01766

Software

- Review
- Repository
- Archive

Submitted: 05 September 2019
Published: 26 September 2019

Summary

Finding the shortest paths through a network is a task important to many problems, from transportation routing to social network analysis. **yenpathy** provides a method to find the *k* shortest paths through a network using Yen's Algorithm (Yen, 1971), previously not available in the R Langauge (R Core Team, 2019).

yenpathy provides the function `k_shortest_paths()`, which returns *k* shortest paths between two nodes in a graph, ordered from shortest to longest, with length determined by the sum of weights of consecutive edges. The function wraps a C++ implementation of Yen's algorithm created by Yan Qi (Qi, 2017). It works correctly with stand-alone edge lists, as well as with objects from the **igraph** (Csardi & Nepusz, 2006) and **tidygraph** (Pedersen, 2019) packages.

Usage

To find shortest paths through a network, pass `k_shortest_paths()` a data frame with rows representing the edges of a graph (`graph_df`), as well as supplying `from` and `to` nodes in `graph_df`. The function will return a list containing up to `k` (default 1) vectors representing paths through the graph:

```
library(yenpathy)

small_graph <- data.frame(
start = c(1, 4, 5, 1, 1, 8, 1, 2, 7, 3),
end = c(4, 5, 6, 6, 8, 6, 2, 7, 3, 6),
weight = c(1, 1, 1.5, 5, 1.5, 2.5, 1.5, 0.5, 0.5, 0.5)
)

k_shortest_paths(small_graph, from = 1, to = 6, k = 4)
```

```
## [[1]]
## [1] 1 2 7 3 6
##
## [[2]]
## [1] 1 4 5 6
##
## [[3]]
## [1] 1 8 6
```

```
##
## [[4]]
## [1] 1 6
```

You can also pass an `edge_penalty`, a number which is added to all of the edge weights, effectively penalizing paths consisting of more edges.

```
k_shortest_paths(small_graph, from = 1, to = 6, k = 4, edge_penalty = 1)
```

```
## [[1]]
## [1] 1 6
##
## [[2]]
## [1] 1 8 6
##
## [[3]]
## [1] 1 4 5 6
##
## [[4]]
## [1] 1 2 7 3 6
```

API

`k_shortest_paths()` takes a data frame as its first argument (`graph_df`). In place of a data frame, you can provide an **igraph** (Csardi & Nepusz, 2006) or **tidygraph** (Pedersen, 2019) graph object, which will be converted to a data frame using `igraph::as_data_frame()`.

By default, the first and second columns are assumed to contain the names of the the start and end nodes, respectively. These may be either integers or character vectors. If there is a column named "weights", it is used for edge weights. You can change these defaults by passing column numbers or names to the `col_from`, `col_to`, and `col_weights` arguments.

An example using flight data

A common application for Yen's algorithm is in transportation planning, when one wants to find the shortest routes between stops such as airports. Here we use the `USairports` dataset from the **igraphdata** package to demonstrate how to find the k shortest routes between airports in Bangor, ME (BGR) and Omaha, NE (OMA).

```
library(igraph)
library(igraphdata)
data("USairports")
paths <- k_shortest_paths(USairports, "BGR", "OMA",
                          k = 20, col_weights = "Distance")
paths[1:5]
```

```
## [[1]]
## [1] "BGR" "DTW" "OMA"
##
## [[2]]
## [1] "BGR" "DTW" "ORD" "OMA"
##
```

```
## [[3]]
## [1] "BGR" "DTW" "ORD" "DSM" "OMA"
##
## [[4]]
## [1] "BGR" "DTW" "DSM" "OMA"
##
## [[5]]
## [1] "BGR" "DTW" "AZO" "ORD" "OMA"
```

The result shows the airport *nodes*, but we can use the `paths` object to look up the *edges*, or flights, that make up these itineraries. In this case, we will select just the carriers with the most seats for each leg, as `USAairports` includes multiple edges for each carrier operating flights between an airport pair:

```
usa <- as_data_frame(USairports)
itineraries <-
  lapply(paths, function(.x) {
    n_stops <- length(.x)
    legs <- cbind(.x[1:(n_stops - 1)], .x[2:n_stops])
    flights <- apply(legs, 1, function(.z) {
      fl <- subset(usa, from == .z[1] & to == .z[2])
      fl <- subset(fl, Seats == max(Seats))
    })
    do.call(rbind, flights)
  })
itineraries[1:5]
```

```
## [[1]]
##       from  to                      Carrier Departures Seats Passengers
## 8404   BGR DTW      Pinnacle Airlines Inc.         29  1450       1287
## 18340  DTW OMA Atlantic Southeast Airlines         71  4615       3350
##       Aircraft Distance
## 8404       629      750
## 18340      631      651
##
## [[2]]
##       from  to                       Carrier Departures Seats Passengers
## 8404   BGR DTW       Pinnacle Airlines Inc.         29  1450       1287
## 20147  DTW ORD American Eagle Airlines Inc.        144  7200       6344
## 21492  ORD OMA        United Air Lines Inc.         48  6798       5077
##       Aircraft Distance
## 8404       629      750
## 20147      675      235
## 21492      694      416
##
## [[3]]
##       from  to                       Carrier Departures Seats Passengers
## 8404   BGR DTW       Pinnacle Airlines Inc.         29  1450       1287
## 20147  DTW ORD American Eagle Airlines Inc.        144  7200       6344
## 22514  ORD DSM        Shuttle America Corp.         73  5110       2759
## 17379  DSM OMA                Allegiant Air          1   130         31
##       Aircraft Distance
## 8404       629      750
## 20147      675      235
```

```
## 22514      677      299
## 17379      654      117
##
## [[4]]
##       from  to               Carrier Departures Seats Passengers Aircraft
## 8404   BGR DTW Pinnacle Airlines Inc.         29  1450       1287      629
## 8534   DTW DSM Pinnacle Airlines Inc.         42  2100       1518      629
## 17379  DSM OMA          Allegiant Air          1   130         31      654
##       Distance
## 8404       750
## 8534       534
## 17379      117
##
## [[5]]
##       from  to                     Carrier Departures Seats Passengers
## 8404   BGR DTW       Pinnacle Airlines Inc.         29  1450       1287
## 8514   DTW AZO       Pinnacle Airlines Inc.         66  3299       2093
## 19876  AZO ORD American Eagle Airlines Inc.         54  2700       1451
## 21492  ORD OMA        United Air Lines Inc.         48  6798       5077
##       Aircraft Distance
## 8404       629      750
## 8514       629      113
## 19876      675      122
## 21492      694      416
```

We can calculate the total distance of each itinerary:

```
sapply(itineraries, function(.x) sum(.x[["Distance"]]))
```

```
##  [1] 1401 1401 1401 1401 1401 1401 1402 1402 1402 1402 1404 1405 1405 1407
## [15] 1407 1408 1411 1414 1416 1416
```

Performance

A number of packages, such as **igraph** and **dodgr**, provide functions to find the shortest path between two nodes on a network, or *all* "simple" (non-looping) paths between nodes. For the *single shortest* path, these packages can be faster than **yenpathy**:

```
library(bench)
set.seed(42)
network <- sample_smallworld(1, 20, 4, .05)
bench::mark(
  shortest_paths(network, from = 1, to = 10),
  k_shortest_paths(network, from = 1, to = 10, k = 1),
  check = FALSE
)
```

```
## # A tibble: 2 x 6
##   expression                                          min median
##   <bch:expr>                                        <bch> <bch:>
## 1 shortest_paths(network, from = 1, to = 10)        143us  161us
## 2 k_shortest_paths(network, from = 1, to = 10, k = 1) 533us  572us
## # ... with 3 more variables: `itr/sec` <dbl>, mem_alloc <bch:byt>,
## #   `gc/sec` <dbl>
```

However, to calculate *k* shortest paths, one generally has to calculate *all* such paths, and later subset. Doing this with `igraph::all_simple_paths()` can be considerably slower, even on a very simple network, and impossible due to memory or time constraints for large, densely-connected graphs.

```
network <- sample_smallworld(1, 8, 3, .05)
bench::mark(
  all_simple_paths(network, from = 1, to = 5, mode = "all"),
  k_shortest_paths(network, from = 1, to = 5, k = 1),
  check = FALSE
)
```

```
## # A tibble: 2 x 6
##   expression                                                    min  median
##   <bch:expr>                                                 <bch:t> <bch:t>
## 1 all_simple_paths(network, from = 1, to = 5, mode = "all")  49.2ms  49.2ms
## 2 k_shortest_paths(network, from = 1, to = 5, k = 1)        427.6us 482.5us
## # ... with 3 more variables: `itr/sec` <dbl>, mem_alloc <bch:byt>,
## #   `gc/sec` <dbl>
```

Acknowledgment of Financial Support

yenpathy was developed in part with financial support from the U.S. Dept. of Homeland Security (70RSAT18CB0031001). The package is currently in use at EcoHealth Alliance for an ongoing project creating a flight network model to estimate the possible spread of disease outbreaks.

References

Csardi, G., & Nepusz, T. (2006). The igraph software package for complex network research. *InterJournal*, *Complex Systems*, 1695. Retrieved from http://igraph.org

Pedersen, T. L. (2019). *Tidygraph: A tidy api for graph manipulation*. Retrieved from https://CRAN.R-project.org/package=tidygraph

Qi, Y. (2017). *An implementation of k-shortest path algorithm (cpp version)*. Retrieved from https://github.com/yan-qi/k-shortest-paths-cpp-version

R Core Team. (2019). *R: A language and environment for statistical computing*. Vienna, Austria: R Foundation for Statistical Computing. Retrieved from https://www.R-project.org/

Yen, J. Y. (1971). Finding the k shortest loopless paths in a network. *Management Science*, *17*(11), 712–716. doi:10.1287/mnsc.17.11.712