SoftwareX 9 (2019) 265–270

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

"rsppfp": An R package for the shortest path problem with forbidden paths

Melina Vidoni *, Aldo Vecchietti

Institute of Design and Development (INGAR CONICET-UTN), Avellaneda 3657, Santa Fe, Argentina

ARTICLE INFO

Article history:
Received 6 June 2018
Received in revised form 16 January 2019
Accepted 6 March 2019

Keywords:
R package
Shortest path
Forbidden paths
Network flows

ABSTRACT

The Shortest Path Problem with Forbidden Paths (SPPFP) is a variant of the original shortest path problem, where the constraints come from a set of forbidden arc sequences that cannot be part of any feasible solution. Though this problem is addressed in the academic literature and has numerous applications, there are no open-source implementations of algorithms that solve it. This article proposes "rsppfp", an R package that offers functionalities that solve the SPPFP by transforming it into the traditional shortest path problem. Its main strengths are its parallel processing capability, and it is high compatibility with packages for other network research. In this paper, we describe the design and functionality of "rsppfp", report an evaluation made with different graph structures, and provide guidelines and examples for its use.

Code metadata

Current code version	V1.0.4
Permanent link to code/repository used of this code version	https://github.com/ElsevierSoftwareX/SOFTX_2018_71
Legal Code License	GPL
Code versioning system used	Git
Software code languages, tools, and services used	Language: R v3.4 R Packages: `stringr`, `dplyr`, `foreach`, `doParallel`, `igraph`, `tidyr`
Compilation requirements, operating environments & dependencies	R Packages: `stringr`, `dplyr`, `foreach`, `doParallel`, `igraph`, `tidyr`
If available Link to developer documentation/manual	https://melvidoni.github.io/rsppfp
Support email for questions	melinavidoni@santafe-conicet.gov.ar

Software metadata

Current software version	V1.0.4
Permanent link to executables of this version	https://CRAN.R-project.org/package=rsppfp
Legal Software License	GPL
Computing platforms/Operating Systems	Linux, Windows
Installation requirements & dependencies	R Packages: `stringr`, `dplyr`, `foreach`, `doParallel`, `igraph`, `tidyr`
If available, link to user manual – if formally published include a reference to the publication in the reference list	https://melvidoni.github.io/rsppfp
Support email for questions	melinavidoni@santafe-conicet.gov.ar

1. Motivation and significance

The shortest path problem with forbidden paths (SPPFP) has been introduced by Villeneuve and Desaulniers [1] . On this problem, given a graph $G = (N, A)$, where N is the nodes set and A the arcs set, there is also a list F of forbidden paths in G that cannot be part of any solution path. In this version of the problem, F is known beforehand; however, each forbidden path may be of different length with a minimum of three nodes. This problem is often found on optical networks [2], window networks [3], logistics and routing [4], among others.

* Corresponding author.
E-mail addresses: melinavidoni@santafe-conicet.gov.ar (M. Vidoni), aldovec@santafe-conicet.gov.ar (A. Vecchietti).

https://doi.org/10.1016/j.softx.2019.03.004

Most academic proposals solve the SPPFP by transforming the original graph G, along with its F, to an enlarged graph $G^* = (N^*, A^*)$ such that every path in G^* does not contain any forbidden path $f \in F$. This approach has two advantages:

- In most cases, G and F remain unchanged for long periods of time. Thus, the transformation is completed only once, and G^* can be stored along with the original graph. A new conversion is required only on the rare cases where G or F change.
- The problem can be solved on G^* using a traditional shortest path with algorithms that efficiently manage time and resource constraints. These algorithms are implemented in a plethora of programming languages and frameworks.

As a result, Fig. 1 showcases this solving process, using a paper notation to indicate input and output data.

R is a highly effective software environment for statistical analysis and data processing that provides robust support for network research. Examples are known packages such as igraph [5], WGCNA [6], and others. However, to the authors' knowledge, there are no R Packages that provide solutions for the SPPFP.

"rsppfp" is motivated from undergoing research in traffic networking and routing, in which there are forbidden paths that represent specific restrictions such as, for example, u-turns or left-turns in two-ways streets. Therefore, there is a need to transform the current SPPFP, derived from a city street-map, to solve the shortest path problem.

"rsppfp" manipulates input and output in standard R data frame formats, maximising its compatibility with other packages, in this way the results can be used with other networking tools. Its contributions are highlighted in Fig. 1 while describing the process of solving an SPPFP. Even more, the algorithms included in this package are prepared to be executed in parallel, using R's distinctive capabilities. This configuration is straightforward for the user since only the number of cores is required as an input. "rsppfp" includes different transformation algorithms, to respond to varying constraints inside the problem; some of them are Villeneuve and Desaulniers's [1] transformation, Hsu et al. [7] backward construction, translations to convert a sequence of N^* back to N, and to use these functions with non-directed graphs.

Also, graphs are formatted as data frames, the transformations manipulate graphs where nodes are represented by their names, and the arcs have an undisclosed number of attributes. Even more, these attributes can be of any type; the transformation processes them without any additional configuration from the user.

2. Software description

"rsppfp" is a package that enables the transformation of graphs, and their sets of forbidden paths, to use them later with other tools. It implements algorithms proposed and accepted from an academic point of view [1,7]. It is written in R, and it is available in a GitHub repository, making it possible to clone or fork.

2.1. Input data & format

The primary input data consists of two different data frames and an additional parameter for parallel processing.

First, G represents the original graph as a data frame of arcs. Arcs are mainly represented as origin–destination pairs, with the column names from and to, respectively. These two columns need to be in positions 1:2 of the data frame. Each arc may or not may have additional attributes, and this is handled internally. The node names may be of any simple type, such as character, integer, double, and others; however, the returned graph will have nodes of character type. This is discussed further in Section 1.3.

Second, F represents the data frame of forbidden paths, where each row of the data frame is a path $f \in F$. The paths may be of different lengths, but the unused columns must be filled with NA values. Also, all nodes used in F must be part of G; they can be of any data type, but they are internally manipulated as characters. Columns names are irrelevant on this data frame.

It is assumed that forbidden paths involve at least three nodes. This is because any path of two nodes is a simple arc, and should be removed from G as they are forbidden.

Third, "cores" is an optional parameter that enables the use of parallel processing in these algorithms. It is recommended that for a computer with c cores, the maximum possible value for this parameter should be $c - 1$, to allow one core to take care of the operative system's functions. If no value is received, "cores" defaults to a 1L value, removing the parallelism.

Code fragment (1) provides the structure to create a sample graph, with a $F = \{[s, u, v, t], [u, v, y, u]\}$. This graph can be seen on Fig. 2. It is worth noting that in the SPPFP, a path cannot contain any complete $f \in F$, but can include arcs that are part of any given forbidden path; for example, for the graph seen in Fig. 2, the path $p = [s, u, w, v, y]$ is acceptable: it contains two arcs used in different $f \in F$ ($[s, u]$ and $[v, y]$, respectively), but it does not include any full forbidden path.

```
f <- data.frame( V1 = c("s", "u"), V2 = c("u", "v"),
                 V3 = c("v", "y"), V4 = c("t", "u"),
                 stringsAsFactors = FALSE)

g <- data.frame(from = c("s", "s", "s", "u", "u", "w",
                         "w", "x", "x", "v", "v", "y", "y"),
                to = c("u", "w", "x", "w", "v", "v",
                       "y", "w", "y", "y", "t", "t", "u"),
                cost = c(1L, 1L, 1L, 1L, 1L, 1L, 1L,
                         1L, 1L, 1L, 1L, 1L, 1L),
                stringsAsFactors = FALSE)
```
(1)

2.2. Available functions

"rsppfp" implements two transformation algorithms. In both cases, F must be known beforehand, and each forbidden path on the set can be of a different length –i.e., include a dissimilar number of nodes. However, the minimum length for a forbidden path is three nodes.

It is worth mentioning that these algorithms are non-trivial since the implementation of its logic is not straightforward. However, it is not the goal of this article to describe that logic, as their authors discuss it extensively.

The first one is Villeneuve and Desaulniers' [1] transformation. This algorithm is restricted by one rule, limiting that no $f_i \in F$ must be, or contain, a sub-path of another $f_j \in F$, where $i \neq j$; the meaning is that a forbidden path cannot contain a sub-path of another forbidden path for the same graph. This transformation process is implemented on a specific function, whose declaration is seen in code fragment (2):

```
modify_graph_vs ← function(g, f, cores = 1L) {...}
```
(2)

The second transformation algorithm is the one developed by Hsu et al. [7], named backward construction. This algorithm differentiates from the previous one; no restrictions are limiting the structure of forbidden paths. This transformation process is implemented in an individual function. Its declaration can be seen in code fragment (3):

```
modify_graph_hsu ← function(g, f, cores = 1L) {...}
```
(3)

As a limitation these algorithms only work with digraphs; i.e., graphs where each arc is directed from a specific node to

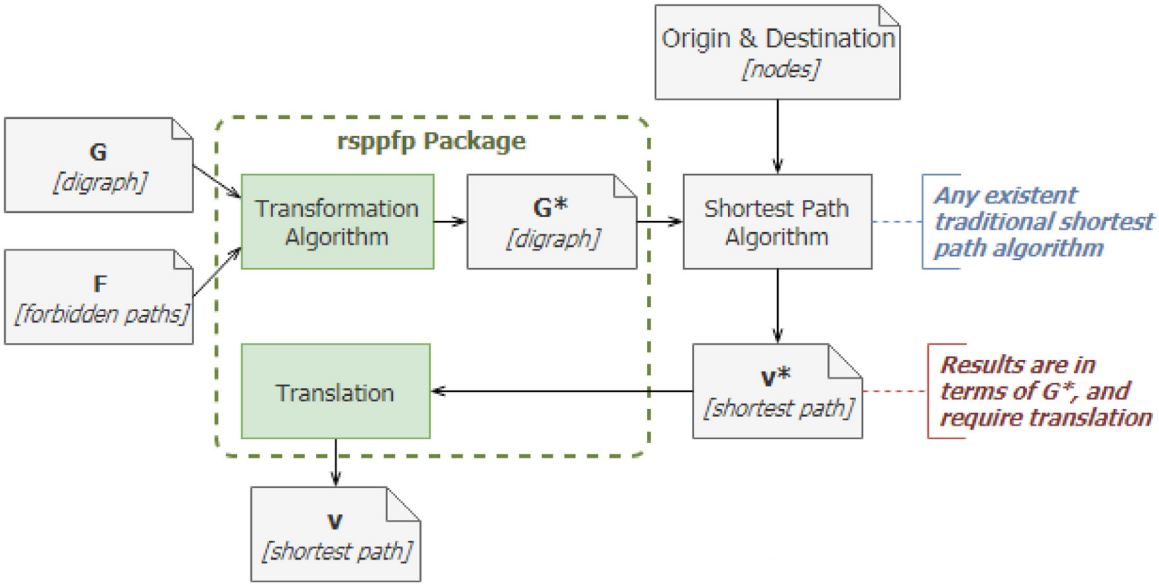

Fig. 1. Flow diagram showcasing an SPPFP solving process, and highlighting "rsppfp" package's contribution.

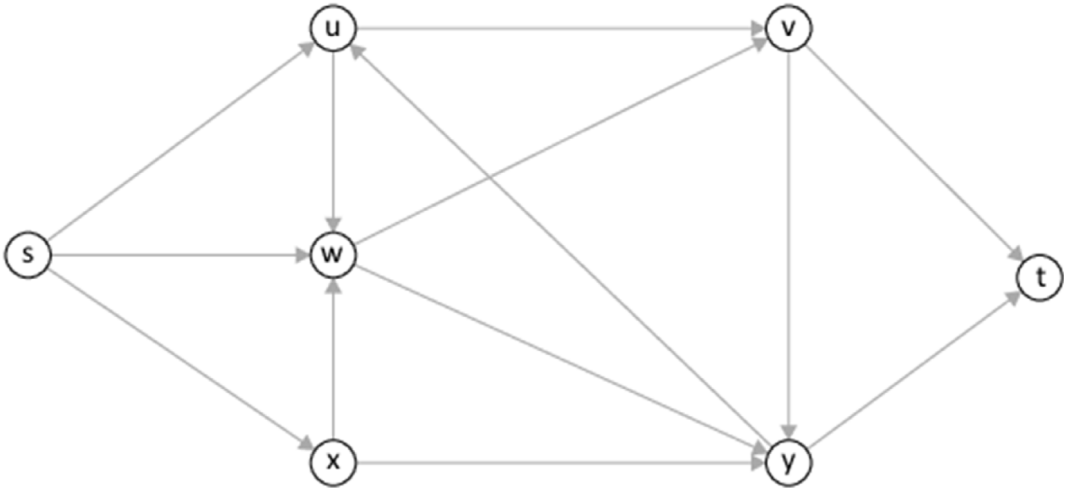

Fig. 2. Structure of the graph generated using code snippet (1).

another node, and can just be travelled in one particular direction. To cover this limitation, "rsppfp" provides an additional function direct_graph(g, cores = 1L) that converts an undirected graph into a digraph. This is done by duplicating each arc and inverting the duplicate's origin–destination pair. This function also supports parallel processing.

2.3. Algorithms' evaluation

To evaluate the performance of the implemented algorithms, several illustrative examples are solved. A total of 27 graphs are generated using igraph's erdos.renyi.game() , and the forbidden paths are constructed with their random_walk() function [5]; results are limited to prevent walks of less than three nodes. These graphs result from the combination of the following values: a set of nodes of $size(N) = \{100, 300, 500\}$, the amount of forbidden paths varying between $size\,(F) = \{50, 250, 500\}$, and a varying density of $size\,(d) = \{0.1, 0.5, 0.9\}$. In graph theory, a dense graph is a graph in which the arcs count is close to the maximal number of possible arcs.

The functions are evaluated when transforming these graphs, using the R base function system.time(). The code, as well as the generated graphs, is available as part of "rsppfp"'s package, on the GitHub repository and website. This is executed on a computer with an Intel i7-4790 CPU at 3.6 Ghz, with 4GB of RAM, using Linux Ubuntu 16.04 as the operating system; three cores out of four are used in all the transformations. Fig. 3 shows the results of the test, faceted by arc density.

As can be seen, in all situations the more general algorithm (Hsu's) takes longer to complete, though this difference increases alongside the number of forbidden paths and density. It is worth mentioning that this is an expected behaviour, as the transformation logic iterates on the forbidden paths F –each $f \in F$ needs to be individually evaluated to add new nodes and arcs to the graph. Because of its logic, Hsu's backward construction always produces a smaller G^* with less added nodes and arcs, while Villeneuve's algorithm is slightly faster; this difference is noticeable for larger values of F.

2.4. Output data & formats

Both types of transformation functions return a graph G^* as a data frame where each row represents an arc. It maintains

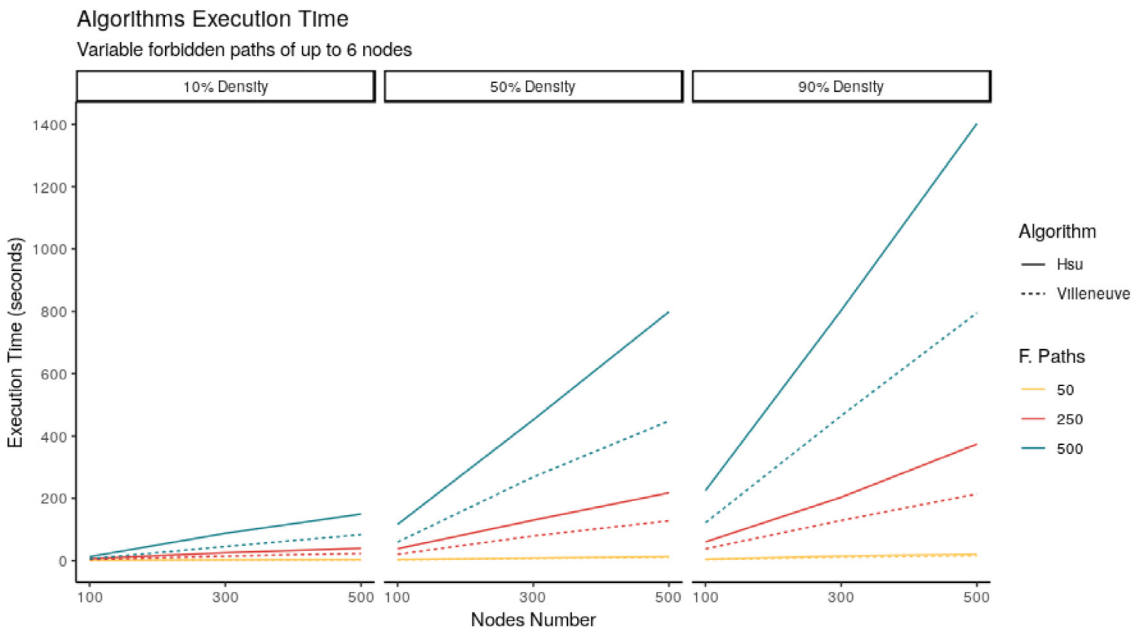

Fig. 3. The execution time of Villeneuve's and Hsu's algorithms in "rsppfp" in regards to nodes number. It includes variable forbidden paths numbers with a path length of up to 6 nodes, with facets per density.

the main columns from and to representing the arcs in terms of origin and destination nodes; additional columns represent other G arcs' attributes, if corresponds. However, regardless of the data type used for N in the original graph, N^* – the nodes of the resulting graph – are always of character type. This is because the generation of the new nodes names follows the proposal made by Villeneuve and Desaulniers' [1]: as both algorithms loop through each node on each forbidden path, these are generated by incrementally concatenating the original names, split by pipe.

For example, for a given $f_1 \in F = \{f, c, p, t, n\}$, with the nodes named as letters, the new nodes are: $f|c, f|c|p, f|c|p|t$, and so on. In a second example, for $f_2 \in F = \{1988, 1985, 1958, 1963\}$, with nodes as integer values, the new nodes names are: 1988|1985, 1988|1985|1958 and 1988|1985|1958|1963. As the package allows transforming graphs with multiple attributes, its conversion towards G^* works as follows:

- For arcs between existing nodes, their attributes remain the same.
- For arcs between a new node (e.g., $f|c$) and an existing node (e.g., s), the attributes of the arc are the same as those of the original arc between the traditional nodes c and s.
- For arcs between new nodes, the attributes are the same as those from the original arc between the last traditional node of each new node. For example, for an arc $\{f|c|p, k|m|t\}$ the attributes are those from the original arc {p, t}.

However, any paths calculated in G^* make use of these names. As a result, "rsppfp" provides an additional function, parse_vpath(vpath) that translates a path with N^* to one compatible with N nodes. The path must be passed as a vector of ordered nodes (or vertexes). Related to this, the package also offers a function to obtain every N_i^*nodes that are equivalent to an original $N_i$node; this can be used to obtain paths involving said node.

3. Illustrative examples

This section presents an example to showcase how "rsppfp" can be used. The full code is accessible from "rsppfp"'s GitHub repository in the examples directory, and in the website.

The first step is to define the graph and its forbidden paths. This is done in code fragment (4), with a set of forbidden paths defined as $F = \{f_1, f_2, f_3, f_4\}$, where $f_1 = \{u, v, y, u\}$, $f_2 = \{u, w, y, u\}$, $f_3 = \{w, v, y\}$ and $f_4 = \{x, w, v, y, t\}$. The graph structure is the same as the one presented in Fig. 1. Though this example is defined arbitrarily, and the input data is hard-coded, it is worth noting that the input can be obtained from different sources such as databases, spreadsheet files, and others; however, that process is outside of "rsppfp"'s scope.

```
# 1a. Load the set of forbidden paths
f <- data.frame( V1 = c("u", "u", "w", "x"),
                 V2 = c("v", "w", "v", "w"),
                 V3 = c("y", "y", "y", "v"),
                 V4 = c("u", "u", NA, "y"),
                 V5 = c(NA, NA, NA, "t"),
                 stringsAsFactors = FALSE)
```
(4)

```
# 1b. Load the graph as a data frame of arcs
g <- data.frame( from = c("s", "s", "s", "u", "u", "w", "w",
                          "x", "x", "v", "v", "y", "y"),
                 to = c("u", "w", "x", "w", "v", "v", "y", "w",
                        "y", "y", "t", "t", "u"),
                 stringsAsFactors = FALSE)
```

Once the graph and the forbidden path are read, it is possible to use "rsppfp"'s functions to transform the original graph, into G^*. In this case, some $f \in F$ have sub-paths that are part of other$f$. As a result, the example makes use of Hsu's backward construction function. This can be seen on (5):

```
# 2. Transform the graph into gStar
gStar <- modify_graph_hsu(g, f, cores = 3L)
```
(5)

The resulting data frame G^* (named in the code as gStar) can be transformed to other data types, specific of particular libraries. For example, it is possible to use the function graph_from_data_frame() provided by igraph [5], to convert G^* to use igraph shortest-path functionalities. Using said function also allows plotting the graph. This was done for Fig. 4, which shows the transformed graph using tkplot().

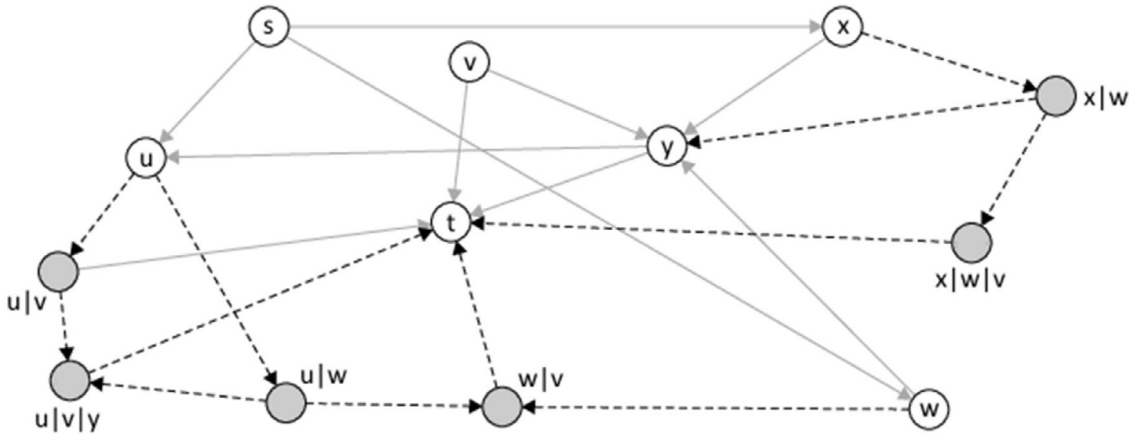

Fig. 4. The modified graph, produced with Hsu's backward construction.

In Fig. 4, grey vertexes indicate the new nodes, and black dashed arcs represent the new links. As "rsppfp" aims to maximise compatibility with existent packages, these visualisations can also be achieved using other libraries, such as ggplot2 [8].

From here, it is possible to use any shortest-path algorithm to obtain the desired path. However, since those should be applied to G^* to avoid the forbidden paths, additional steps need to be carried. First, a node $n \in G$ can have equivalent nodes in G^*. For example, in Fig. 4 the nodes w, $x|w$ and $u|w$ are both equivalents, as they all represent the same original node w, but arriving from different places. Therefore, in order to search for a shortest-path, it needs the consider the starting node, towards all of the equivalences of the destination node. The equivalences can be obtained with the "rsppfp" function get_all_nodes(), as seen on code fragment (6):

```
# 3. Get all nodes where "w" is the destination
dest <- get_all_nodes(gStar, "w")
```
(6)

After that, any shortest path function could be applied. For instance, to use igraph [5] capabilities, gStar needs to be converted to an igraph class; then it is required to obtain the shortest path towards each equivalent destination point, and keep only the one with less weight. The code required to do this is seen at fragment (7):

```
# 4. Convert to igraph in order to use "shortest_paths"
     from igraph
gStar.igraph <- graph_from_data_frame(gStar)

# 5. Find shortest paths from "s" to all N* corresponding to "w"
sp <- shortest_paths(gStar.igraph, from = "s", to = dest,
                     weights = gStar$weight, output = "both")

# 6. Find shortest of these paths
dist <- vapply(sp$epath, function(path) sum(path$weight),
               numeric(1))
shortestPath <- sp$vpath[[which.min(dist)]]
```
(7)

However, since any path calculated in G^* is given in terms of its N^*nodes, it is required to parse the final path back to the nodes belonging to G. This can be done using "rsppfp"'s function parse_vpath(), as seen on fragment (8).

```
# 7. Convert path with nodes from N* to path with nodes
     from N
parse_vpath(names(shortestPath))
```
(8)

"rsppfp" provides a convenience function named get_shortest_path() that summarises the code from snippets (6, 7, 8), simplifying the process of obtaining the shortest path without incurring into forbidden paths. However, this function requires having igraph installed.

4. Impact

The decision to create and publish "rsppfp" package is influenced by the fact that the SPPFP has several applications [2–4]; this becomes relevant, as this package is part of undergoing research in logistics, which requires an implementation of the SPPFP in R. Nonetheless, the SPPFP is still a problem with unexplored questions. Even more, current transformation algorithms are presented in academic journals [9], but their implementation is not public.

As a result, this article presents a package that aims to bring some of these transformation algorithms, implemented in R. This decision is made based on R's increasing popularity and statistics abilities, as well as its widespread use in the data science community [10]. Furthermore, it is a language that can be used throughout the whole research cycle, from obtaining and analysing the data, to displaying it even through interactive websites.

In particular, "rsppfp" objectives are:

- To present a free and easy to use tool for this type of graph problems.
- To build a common platform with several transformation algorithms, each one suited to different needs.
- Open-source, open to the possibility of implementing own functions to incorporate into the package.

A significant advantage is that "rsppfp" can be used out-of-the-box, it is quickly installed from the GitHub repository, and it is highly compatible with other packages, as it uses standardised data types from R. The main contributions are:

- The publication of a ready-to-use package for transforming digraphs with known forbidden paths, into graphs that can be solved using traditional and efficient shortest-path algorithms.
- The use of R's parallel processing capabilities, providing a simple configuration for users.
- The existence of functionalities to translate paths, as well as the package's ability to apply the transformations to undirected graphs.
- Open-source resource, with avenues available to add more algorithms, and able to include contributions from the community.

5. Conclusions

This article presents "rsppfp", a flexible, compatible and user-friendly R-package to transform digraphs for the *shortest path problem with forbidden paths*. As a result, this transformation process allows using traditional shortest path functions on the resulting graphs. Its input and output data is produced in native R data types, maximising the package's interoperability with other networking libraries. Even more, the package makes use of R's parallel processing, offering the users a straightforward way of configuring its use. Time and resources constraints have been analysed, and the test graphs are available in the GitHub repository.

"rsppfp" is an ongoing project. The authors intend to add new features and functionalities by implementing different solutions offered by other academic authors. Examples of these are the elementary version of the SPPFP, and algorithms for when the set of forbidden paths is not known beforehand.

Acknowledgments

The authors gratefully acknowledge the financial support for the work presented in this article to Universidad Tecnológica Nacional (UTN), Argentina through PID EIUTIFE0003974TC. Also, the authors appreciatively recognise the extraordinary efforts of the anonymous reviewers for their suggestions for improving the read of the article and the quality of the code.

Conflict of interest

The authors declare that there is no conflict of interest.

References

[1] Villeneuve D, Desaulniers G. The shortest path problem with forbidden paths. European J Oper Res 2005;165(1):97–107. http://dx.doi.org/10.1016/j.ejor.2004.01.032.

[2] Ahmed M, Lubiw A. Shortest path avoiding forbidden subpaths. In: Symposium on theoretical aspects of computer science. Freiburg, Germany; 2009. p. 63–74. https://hal.archives-ouvertes.fr/STACS2009/inria-00359710.

[3] Yang H-H, Chen Y-L. Finding k shortest looping paths with waiting time in a time–window network. Appl Math Model 2006;30(5):458–65. http://dx.doi.org/10.1016/j.apm.2005.05.005.

[4] Yang H-H, Chen Y-L. Finding K shortest looping paths in a traffic-light network. Comput Oper Res 2005;32(3):571–81. http://dx.doi.org/10.1016/j.cor.2003.08.004.

[5] Csárdi G, Nepusz T. The igraph software package for complex network research. Inter J Complex Syst 2006;1–9.

[6] Langfelder P, Horvath S. WGCNA: an R package for weighted correlation network analysis. BMC Bioinformatics 2009;9. http://dx.doi.org/10.1186/1471-2105-9-559.

[7] Hsu C-C, Chen D-R, Ding H-Y. An efficient algorithm for the shortest path problem with forbidden paths. In: International conference on algorithms and architectures for parallel processing. LNCS, vol. 5574. Taipei, Taiwan; 2009. p. 638–50. https://doi.org/10.1007/978-3-642-03095-6_60.

[8] Wickham H. Ggplot2: Elegant graphics for data analysis. 2nd ed. Switzerland: Springer International Publishing; 2016, http://dx.doi.org/10.1111/j.1541-0420.2011.01616.x.

[9] Pugliese LDP, Guerriero F. A survey of resource constrained shortest path problems: Exact solution approaches. Netw Int J 2013;62(3):183–200. http://dx.doi.org/10.1002/net.21511.

[10] Tippmann S. Programming tools: Adventures with R. Int Wkly J Sci 2015;517:109–10. http://dx.doi.org/10.1038/517109a.