Data structures and methods for reproducible street network analysis: overview and implementations in R

Andrea Gilardi, Robin Lovelace and Mark Padgham

Abstract

Emerging datasets, new methods and shifting transport planning priorities have sparked a renewed interest in the analysis of street networks, linear geometries and attributes representing roads and other transport ways at local, city and regional scales. This paper outlines data structures, vital pre-processing steps and analysis methods using open source software. While the concepts and datasets presented are 'language agnostic', we implement them in `R`, an increasingly popular language for geographical and network data analysis. Two approaches are compared: one based on the `stplanr` package, which contains `igraph` and `sf` objects, and the other based on the `dodgr` package, which defines the `dodgr_streetnet` class, extending `R`'s mature `data.frame` class. A range of datasets, from a roundabout to an entire city network, are used to highlight the main features of each approach. We find that both enable street network analysis tasks, including shortest path and centrality measures. We conclude that the merits and potential pitfalls of the 'graph-based' and 'data.frame-based' approaches should be considered before embarking on street network analysis or software development. Finally, the discussion provides a basis for research using these data structures.

Keywords: Network analysis, R, Street networks

1 Introduction

Spatial networks, such as power grids, railways or rivers, are entities that can simultaneously be represented as spatial and graph objects [6]. From a spatial perspective, they consist of features embedded in a (two or three-dimensional) spatial domain while, from a graph perspective, they consist of a set of vertices connected by a set of edges. Both edges and vertices are associated with spatial geometries, typically points, lines, or polygons. *Street networks* represent a particular type of spatial network, with notable distinctive traits. Their abstract representation can be created in several ways (see, e.g., [29]), but, in this paper, we will focus on the most common approach: each road is associated to the edges

of a graph, while the vertices[1] correspond to the intersections, typically at road junctions, although potentially also in between.

The first examples of road network analysis can be traced back to Euler's solution to the Seven Bridges of Königsberg problem [18] and John Snow's map of Cholera in London [40], two seminal studies that represented the foundational points for *graph theory* and *epidemiology*, respectively. Starting from the 1960s, several authors investigated the characteristics of spatial networks using a more systematic approach [21, 12], while, more recently, other authors focused on the comparisons and the analysis of structural characteristics of urban street networks [25, 11, 22, 14].

The paper has two broad objectives. First, we want to demonstrate the particular properties of street networks and explain the problems that make them unsuitable to be analysed using traditional network analysis software. Then, we will present the main functionalities, the classes and the methods of two `R` packages that focus on street network analysis: `stplanr` and `dodgr`. Rather than focusing on specialised GIS platforms, such as `QGIS` or `GRASS`, we decided to present data-structures and pre-processing operations implemented in `R`, an increasingly popular programming language for geographical data and network analysis [37]. Although other languages, including `Python`, `C++`, and `Julia`, have emerging projects for spatial network analysis, such as `osmnx` [10], `networkx` [20], and `networkit` [41], the choice of `R` was based on the authors' experience and the wide range of statistical methods implemented in the language. Moreover, even if the procedures shown in this paper are specific to `R`, the concepts and the approaches demonstrated here can be implemented using any other software. `stplanr` and `dodgr` may enable reproducible research, similar to the studies mentioned before, and within a popular open-source command-line drive computational environment.

As described at the outset, the two packages offer functions and analytic capabilities which are difficult to implement using more general softwares, including most equivalent packages in `python`. `stplanr` and `dodgr`, along with their respective class systems, are introduced in Sections 2 and 3, in the context of `R`'s evolving capabilities for handling spatial networks. The particularities of street network data are outlined in Sections 4 and 5, which cover Open Street Map data and illustrate concrete examples of street network entities, from roundabouts to citywide networks. Section 6 demonstrates how street networks can be analysed using the two `R` packages, and the final section discusses the strengths and limitations of each approach, with a view to informing future development and research efforts.

[1] The examples that we present in the following Sections are based on *Open Street Map* (OSM) data. The basic structure of OSM data is composed of three *elements* named *nodes*, *ways* and *relations* [32]. For this reason, when we use the term *node* we are referring to the OSM elements, while the term *vertices* is used for the generic graph element.

Listing 1: R code used to generate Figure 1. The five matrices store several pairs of coordinates.

```
street_net_matrix_list <- list(
matrix(c(0, 0, 1, 0, 2, 0, 3, 0, 4, 0, 5, 0), ncol = 2, byrow = TRUE),
matrix(c(5, 0, 5, -1, 5, -2, 5, -3, 5, -4, 5, -5), ncol = 2, byrow = TRUE),
matrix(c(5, 0, 5, 1, 5, 2, 5, 3, 5, 4, 5, 5), ncol = 2, byrow = TRUE),
matrix(c(5, 5, 6, 5, 7, 5, 8, 5, 9, 5, 10, 5), ncol = 2, byrow = TRUE),
matrix(c(5, 0, 6, 0, 7, 0, 8, 0, 9, 0, 10, 0), ncol = 2, byrow = TRUE)
)
street_net_df <- as.data.frame(do.call(rbind, street_net_matrix_list))
plot(street_net_df, xlab = "",ylab = "",pch = as.character(rep(1:5, each = 6)))
```

2 R packages and classes for spatial networks

Spatial networks can be represented using various approaches in R and, in this section, we try to present the most important ones. In theory, any package that is capable of representing spatial coordinates and graphs can also represent street networks, and we do not attempt to cover all options. Instead, this section focuses on packages that can represent the spatial and graph components of street networks *simultaneously*. Before describing some of the key approaches, however, it is worth considering that base-R already supports `lists` and `matrices`, two of the essential data structures needed for representing street networks, as demonstrated in Listing 1 (which generates Figure 1, a crude representation of the same street network depicted in Figures 2 and 3).

The object `street_net_matrix_list` is a `list` composed by five `matrices` that represent, schematically and straightforwardly, the streets of a minimal urban network. Each street is represented using a sequence of points, and each pair of coordinates define one point.

The list of matrices representation of street networks in base-R, demonstrated with stylised data in the code chunk above and converted into a data frame for plotting, is of limited use. Such base-R representations are impractical because they lack a formal class system for performing commonly needed operations such as plotting, subsetting, network analysis and shortest-path calculations. One could build additional components on the structure represented in `street_net_matrix_list`. However, for most tasks, it is likely that using pre-existing representations — encoded in class definitions of several R packages that support street networks — will be more effective. A selection of such packages, in ascending order of their first release on CRAN, is outlined below.

spatstat First released in 2002, it was developed for analysing spatial point patterns. It was not initially designed with street networks in mind but, because point patterns on a linear network represent a more and more stimulating application with its problems and solutions [4], several methods for working with linear networks have been

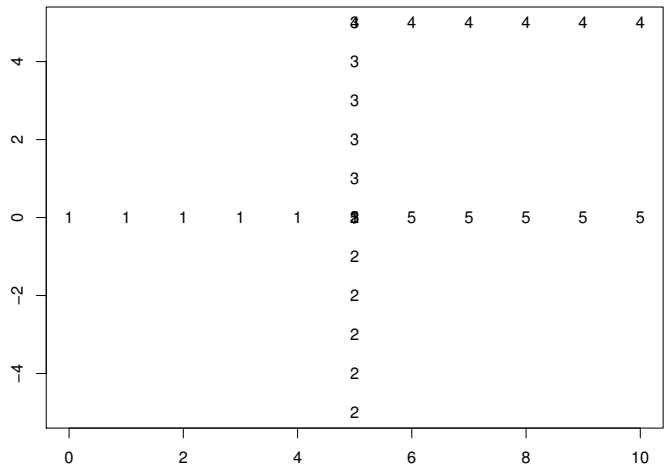


Figure 1: Plot of a simple street network stored as a list of matrices in base-R. Each matrix store several pairs of coordinates, and each number from 1 to 5 represents a distinct segment. There is an overlap of labels on junction points

developed in the package since `spatstat` version `1.22-0`, released in 2011. The authors defined the class `linnet` for representing *a connected network of line segments, such as a road network*, and several ad-hoc functions for Kernel Density Estimation (KDE) and other statistical techniques [3]. This work has recently been extended in the package `spatstat.Knet`, which improves the computational efficiency of statistical methods on route networks [38]. The main limitation of this approach is its flexibility since `spatstat` was developed to perform spatial statistical analysis, and it offers limited support for routing and other street network operations outside of the `spatstat` ecosystem.

shp2graph Before the release of the R package `sf` in 2016, the `Spatial` class system was the most prominent approach to handling spatial data in `R`, with the package `sp` that defined it, and imported by 73 other packages by 2013 [8]. Building on the `Spatial` class system `shp2graph`, first published in 2014, provides functions for converting `SpatialLinesDataFrame` objects into a list representing the components of a spatial network including nodes, edges, weights and other edge attributes. The package enables a variety of common street network operations, including shortest path calculation and network connectivity [28]. The main disadvantage of this package is its reliance on `sp`, which has become largely superseded by `sf`. Moreover, it has not been updated for more than two years.

stplanr This is a package designed to support evidence-based transport planning, with a focus on geographic *desire lines* and route data. The package also provides functions

for working with street network data, including `overline2()`, which converts overlapping lines into a non-overlapping network of lines, and `SpatialLinesNetwork()`, which creates an S4 spatial network object comprising spatial (`sp` or `sf`) and graph (`igraph`) sub-objects [39].

dodgr This `R` package focuses on routing and distances, with a primary focus on directed graphs, and many functions dedicated to analyses of street networks. It performs efficient calculation of many-to-many pairwise distances on dual-weighted directed graphs, aggregation of flows throughout networks, and highly realistic routing through street networks (including time-based routing considering incline, turn-angles and surface quality). It defines classes and functions to represent and manipulate street networks [33].

A couple of packages that have not been published on CRAN, which explore data structures for street networks in `R`, are also worthy of mention. `spnetwork` (see `https://github.com/edzer/spnetwork`) represents an alternative approach to converting `Spatial` objects into `igraph` objects. `sfnetworks` (see `https://github.com/luukvdmeer/sfnetworks`) is a recently developed package that uses `tidygraph` as the basis of the `sfnetwork` class, which is described in detail in a blog post on the subject hosted at `r-spatial.org` (see `https://www.r-spatial.org/r/2019/09/26/spatial-networks.html`) and a *vignette* (see `https://luukvdmeer.github.io/sfnetworks/articles/intro.html`) on the package website.

The remainder of this paper focuses on street network representations in the packages `stplanr` and `dodgr`, which provide distinct data structures for the representation and analysis of street networks.

3 `sfNetwork` and `dodgr_streetnet` objects

As outlined in the Introduction, a defining feature of street networks is their duality: they are simultaneously spatial and graph objects. As spatial objects they are embedded in (typically two dimensional) space; as graphs, their vertices and edges correspond to geographical elements, such as roads or junctions. This section expands on this broad definition and describes the specific data structures that enable street networks to be represented in `stplanr` and `dodgr`.

The `stplanr` representation of a street network is typically created starting from an `sf` object [35], which constitutes the spatial dimension. The graph structure is inferred using an algorithm which is detailed below, and it is stored as an `igraph` object [15]. The `SpatialLinesNetwork()` function combines the two worlds defining an S4 object with class `sfNetwork`. More precisely, the input given to `SpatialLinesNetwork()` is an `sf` object[2] with a `LINESTRING` geometry [1], while the output is an `S4` object having four slots named:

[2]Note that `SpatialLinesNetwork()` can also take `SpatialLinesDataFrame` objects from the `sp` package. In this paper, we focus on the `sf` representations because the `sp` method will no longer be updated.

sl: the `sf` object that was passed as input with an additional column, named `length`, which measures the geographical length of each `LINESTRING` geometry.

g: an `igraph` object. We will explain what the vertices and the edges of the graph are in the next paragraphs. The slots `sl` and `g` represent, respectively, the spatial and the graph dimensions.

nb: a list that summarises the connectivity of the graph;

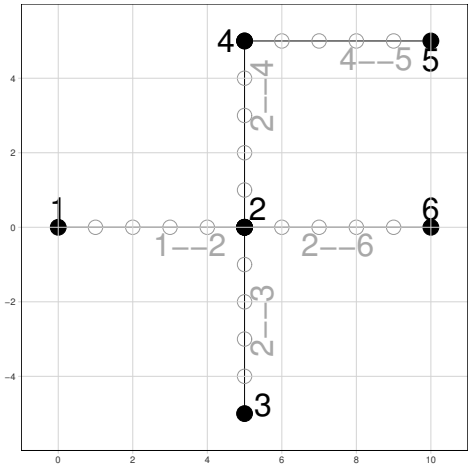
weightfield: a character vector that identifies the weighting profile. The default weights are the lengths of each road segment, but they can be modified using the `weightfield()` function.

The other two parameters of the `SpatialLinesNetwork()` function, i.e. `uselonglat` and `tolerance`, are used to 1) control the Coordinate Reference System (CRS) of the input object and 2) set a numerical value indicating a tolerance threshold to be used when creating the graph structure.

The `igraph` component of an `sfNetwork` object is created using the following algorithm (which is also illustrated in Figures 2 and 4(a)): the edges of the network are the `LINESTRING` geometries that were passed as input, while the vertices of the graph are the first and last points of each geometry (removing the duplicates with identical coordinates if necessary). The connectivity of the graph is determined as follows: two vertices are connected if they belong to the same `LINESTRING` geometry and, by the same reasoning, two edges are connected if they share one boundary point. Moreover, by construction, the `igraph` object has several attributes, named `x`, `y`, `n` and `weight`, that store the spatial coordinates (i.e. `x`, `y`), the connectivity of each vertex, and the `weight` associated to each edge (which is equal to the length of the corresponding spatial line). This algorithm, despite being natural and intuitive, has a few pitfalls that are typical of street network data. The objective of our paper is to present four examples that exhibit how and why it can fail.

On the other side, `dodgr` adopts a different philosophy for representing street networks since it merges the spatial and the graph dimensions, defining a unique object of class `dodgr_streetnet` as an extension of regular `data.frame`. Therefore, the authors of the package defined several functions and methods for working with `dodgr_streetnet` objects, while `stplanr` package adopts `igraph` as a backend for managing its graph structure. They also coded several ways to translate `dodgr_streetnet` objects into and from other formats, like `sf` or `igraph`, which enhances its interoperability with other packages.

The main function for creating a `dodgr_streetnet` object is `weight_streetnet()`, and the only mandatory input parameter is an `sf` data structure having a `LINESTRING` geometry, with coordinates expressed using the WGS84 reference ellipsoid (EPSG code 4326). Data expressed using other CRS must first be transformed to `EPSG:4326` (using, for example, the `sf` function `st_crs()`) before submitting to `weight_streetnet`. One of the primary advantages of `dodgr` is its support for *dual-weighted directed graphs*, which are



Spatial Component

```
Simple feature collection with 5 features
geometry type: LINESTRING
dimension: XY
1 LINESTRING (0 0, 1 0, ...)
2 LINESTRING (5 0, 5 -1, ...)
3 LINESTRING (5 0, 5 1, ...)
4 LINESTRING (5 5, 6 5, ...)
5 LINESTRING (5 0, 6 0, ...)
```

Graph Component

```
6 vertices
[1] 1 2 3 4 5 6
5 edges
[1] 1--2 2--3 2--4 4--5 2--6
```

Figure 2: Graphical example of the data structure used by `stplanr` for representing street networks in the `sfNetwork` class. Left: Map of a street network. The black dots represent the starting and ending point of each `LINESTRING` geometry. The grey dots represent the internal points which are ignored (but they will be important for `dodgr` representation). Right: Summary of the spatial and graph dimension of the street network.

typical of street networks and also necessary to generate realistic routes reflecting mode-specific preferences: pedestrians prefer quiet walkways removed from busy roads, cyclists prefer dedicated bicycle infrastructure, and car drivers prefer fast routes along motorways and large roads. Each edge in a dual-weighted graph has two distances, one representing the *true* geodesic distance, and the other representing a weighting preference (or profile) such that, for example, the weighted length for a pedestrian along an edge of a multi-lane motorway would be considerably longer than the actual distance. The `wt_profile` parameter can be used to select the preferred mode of transport when building a new street network with `weight_streetnet()`. The default value is `bicycle`, with other possibilities including `foot` and `motorcar`, as described in the help page of `weighting_profiles()` function, which also includes details of how to implement custom weighting profiles. The dual weights are also determined by a character label that defines the characteristics of each of the network, such as *motorway*, *primary*, *residential* or *pedestrian*. All road segments are mapped to a set of coefficients according to the chosen mode of transport, and these coefficients determine the dual-weights that define all `dodgr` functions. Their usefulness is exemplified in Section 6.

One further important advantage of `dodgr` package is its ability to *contract* a road network down to only those edges connecting street junctions through the `dodgr_contract_graph()` function. Road networks are commonly represented by points which are effectively arbitrarily located such that, for example, a curved way between two junctions might be

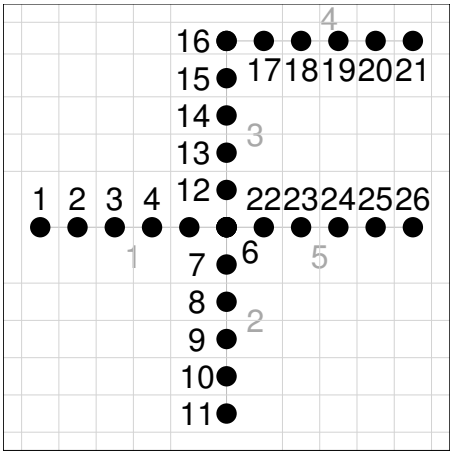
represented by ten intermediate points, while a straight way might not have any intermediate points. These intermediate points are arguably representational artefacts, rather than intrinsic components of the network geometry [23, 31]. It is particularly important for analyses of networks (such as shortest paths or network centrality, described below) to removes these artefacts throughout contracting a network down to only those edges directly connecting junctions, which is precisely what the `dodgr_contract_graph()` function does. The computation efficiency of routing on street networks increases non-linearly with N, the numbers of vertices, with efficiencies commonly scaling $\sim O(N^2)$). Graph contraction is the single most important step necessary for efficient routing. The contracted version of the network considered in the final examples of this paper has over 3 times fewer vertices, with typical values generally ranging between 5 and 10. We refer to the help page of `dodgr_contract_graph()` for more details.

Although the `stplanr` and `dodgr` packages may start with the same spatial objects, they diverge in the construction of street networks. More precisely, `dodgr` divides each `LINESTRING` geometry into its minimal components (also called `Line Segments` [1]), creating a vertex for each node and an edge for every subsequent pair of points belonging to the same `LINESTRING` geometry.

The output of `weight_streetnet()` function is a `dodgr_streetnet` dataframe of edges with several columns including:

1. a unique ID for each `LINESTRING` geometry, called `geom_num`;
2. a unique ID for each edge, called `edge_id`, and for its corresponding vertices, called `from_id` and `to_id`. Duplicated vertices always share the same ID. If the input `sf` data is created using the `osmdata` R package (see below), then `dodgr` checks the uniqueness of the vertices by comparing their Open Street Map ID(s) instead of the spatial coordinates. We will see in Section 5 why this may be important and why this is a peculiar characteristic of street networks;
3. the coordinates of each vertex using the WGS84 reference ellipsoid in four columns named `from_lon`, `from_lat`, `to_lon` and `to_lat`;
4. the spatial length, in meters, of each edge, stored in a column called `d`;
5. the weighted length of each edge, which is estimated using the coefficients determined by the chosen weighting profile and summarised in a column named `d_weighted`.

We include an exemplification of `dodgr` algorithm in Figure 3, where, for simplicity, we report only the first columns, and we omit the dual-weights, typical of `dodgr` objects. Another example is reported in Figure 4(b). Some of the missing columns are reported in Table 1, and the dual-weights will be extensively covered in Section 6.



geom_num	edge_id	from_id	to_id	...
1	1	1	2	...
1	2	2	1	...
1	3	2	3	...
1	4	3	2	...
...	...	...	...	...
5	47	24	25	...
5	48	25	24	...
5	49	25	26	...
5	50	26	25	...

Figure 3: Graphical example of the algorithm used by `dodgr` for the creation of a `dodgr_streetnet` object. Left: Geographical map of the input data. The grey lines are the `LINESTRINGS` while the black dots are the `POINTS` (or *nodes* in OSM jargon) composing each `LINESTRING`. Right: Conversion into a `dodgr_streetnet` object. Each row of the new data frame is an edge of the network and it is linked with a pair of vertices. We created an undirected graph, so each edge is repeated two times. See Section 5 for more details.

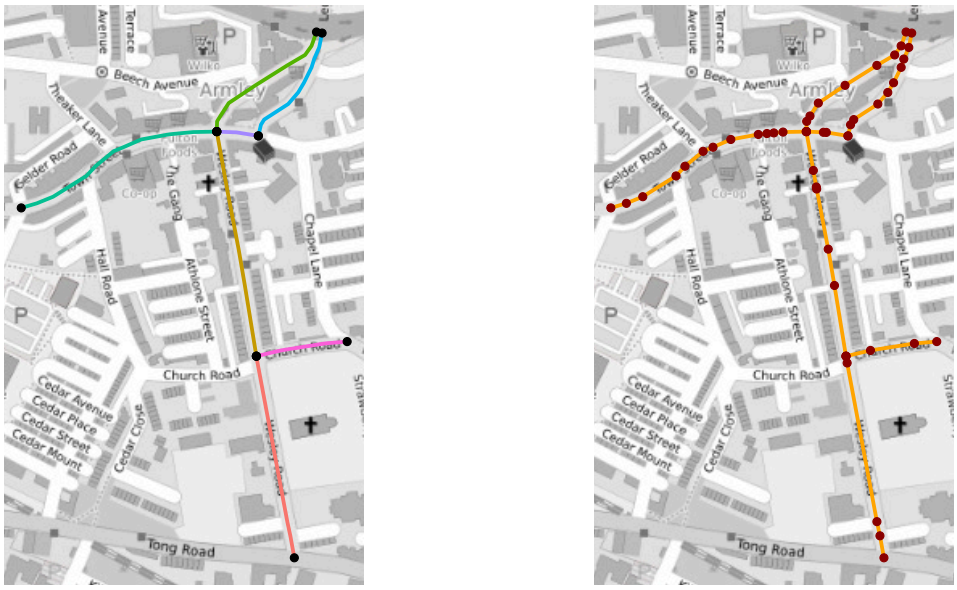
4 Open Street Map

Both packages can work with any type of street data (as long as they are coded in the right way) but, in this paper, we are going to focus on Open Street Map (OSM) data [32]. Open Street Map is the largest, openly available source of spatial network data, and it provides a continuously evolving and future proof basis for research [5]. It's rapid evolution certainly helped the development of street network analysis [2].

There are several `R` packages for downloading data from Open Street Map servers but, at the moment, the most important one is probably `osmdata` [34], which is also perfectly integrated with `dodgr`. In fact, the authors of `dodgr` created the `dodgr_streetnet()` function (which is a wrapper around several other routines defined in `osmdata`) to download and format OSM road data for a given location, which can be expressed either as a string (and processed by *Nominatim geocoding servers*) or a numeric matrix of coordinates that define a bounding box. Then, the output of `dodgr_streetnet()` can be passed to `weight_streetnet()` to create a `dodgr_streetnet` object that represents the street network of a particular area.

The main benefits of this integration are the following:

1. The `osmdata` functions retain the Open Street Map ID(s) of each node hidden within



(a) `stplanr` approach.

(b) `dodgr` approach.

Figure 4: *Left*: `stplanr`-representation of a network. The black dots represent the nodes and the colored lines the edges. *Right*: `dodgr`-representation of a network. The red dots represent the nodes and the orange lines the underlying street lines. The nodes and the edges correspond to several different street segments located in the Armley District of Leeds (UK)

the actual `LINESTRING` structure[3], which enables vertex identification through ID(s) instead of coordinates. We present an example of why this distinction is important in the Subsection named *Overpasses, underpasses and other types of intersections*.

2. The label column, which is used for the estimation of the dual weights, is automatically detected and formatted according to a pre-specified set of weights, as explained in the help page of `weighting_profiles()` function.

3. There are several ancilliary `osmdata` functions which can be used to refine street network construction in `dodgr`, including the `trim_osmdata()` function which can trim a network within a bounding polygon rather than a simple rectangle, and the `osm_poly2line()` function which can convert all `POLYGON` objects into `LINESTRING` objects suitable for routing.

The previous list describes three problems, i.e. node identification, dual weights and

[3]Please note that you can use the `osmdata::unname_osmdata_sf()` function to remove these ID(s), which can be problematic for some plotting routines.

Table 1: First five rows and first eight columns of dodgr representation of the spatial network depicted in Figure 4(b). The missing columns represent other characteristics of the edges like their highway-type and the dual-weights.

	geom_num	edge_id	from_id	from_lon	from_lat	to_id	to_lon	to_lat
1	1	1	0	-1.5886	53.7973	1	-1.5885	53.7969
2	1	2	1	-1.5885	53.7969	0	-1.5886	53.7973
3	1	3	1	-1.5885	53.7969	2	-1.5885	53.7968
4	1	4	2	-1.5885	53.7968	1	-1.5885	53.7969
5	1	5	2	-1.5885	53.7968	3	-1.5885	53.7967

spatial filters, that are typical of street networks and require ad-hoc solutions that are implemented in the corresponding functions and approaches.

5 Street network data types

Street networks are complex phenomena with a range of sizes, shapes and interrelations, ranging from a simple network of mud paths in a remote village to a complex city containing dozens of separate but touching roads for walking, cycling, and motorised modes. In this context, it is important that software for representing and analysing street networks can handle a range of data types. To that end, this section introduces four scenarios that highlight common issues encountered when working with spatial networks representing transport systems. The first three examples are a *roundabout* (which is represented as a circular geometry), an *overpass* (in which intersecting streets are not connected due to a vertical grade of separation) and a *oneway road* (in which vehicles are prohibited from travelling in one direction). They are simple, but they highlight tricky problems that should be taken into account when working with street network data. The final example is a citywide graph containing multiple instances of each of the previous entities, approximating objects that are encountered in applied research and showing how `stplanr` and `dodgr` can work on real-world datasets.

Before proceeding with these examples, it is worth taking a step back to consider the minimal requirements that input datasets must meet before they can be classified as street networks. These requirements apply to data from any source, but, as we said in the previous section, we focus on Open Street Map, and we present a small part of its mature and open set of guidelines. The first requirement is that two or more intersecting streets, or *ways* in OSM nomenclature, must share at least one point, or *node* in OSM nomenclature, otherwise the corresponding edges can not be considered to intersect. A more subtle assumption is that streets that are not truly intersecting due to a vertical degree of separation, such as overpasses and underpasses, should not share any point with identical ID (even though

those points may share identical geographical coordinates). These minimal requirements are documented in the *Editing Standards and Conventions page on the Open Street Map wiki* (see `https://wiki.openstreetmap.org/wiki/Editing_Standards_and_Conventions`).

These are not strict assumptions, and we never found a situation where they are not easily met. There are also several tools for checking these hypotheses, such as `OSM Inspector` (see `https://wiki.openstreetmap.org/wiki/OSM_Inspector`), and fixing the problems, such as `v.clean` tool in `GRASS`[4]. We do not add more details on spatial networks preprocessing, and we refer to Section 8 of sDNA open-source manual, hosted at `https://sdna.cardiff.ac.uk/sdna/wp-content/downloads/documentation/manual/sDNA_manual_v3_4_7/step_by_step_guides.html`. Nevertheless, the importance of *cleaning* the data before building the road network should never be overlooked.

5.1 Roundabouts

Roundabouts are saved by Open Street Map as circular geometries composed by one or more connected `LINESTRING`. In the previous sections, we introduced the algorithm used by `stplanr` for inferring the graph structure of an `sf` data and we said that the connectivity of the graph is determined according to the presence or absence of shared boundary points in the `LINESTRING` geometries. This algorithm implies that the `stplanr`-representation of a roundabout may be unroutable since circular geometries have only one boundary point, which is not always shared with another `LINESTRING`. We present an example of this problem in Figure 5(a). The grey line represents the roundabout, and the black dot is its boundary point. The coloured points are the boundaries of the other streets, which, according to `stplanr` algorithm, are not connected to the roundabout.

This problem can be solved by splitting the circular `LINESTRING`, and this procedure is implemented in the `rnet_breakup_vertices()` function. We refer to its help page for a more detailed description of the algorithm, which is similar to the procedures illustrated in [23]. The result is a routable street network, illustrated in Figure 5(b).

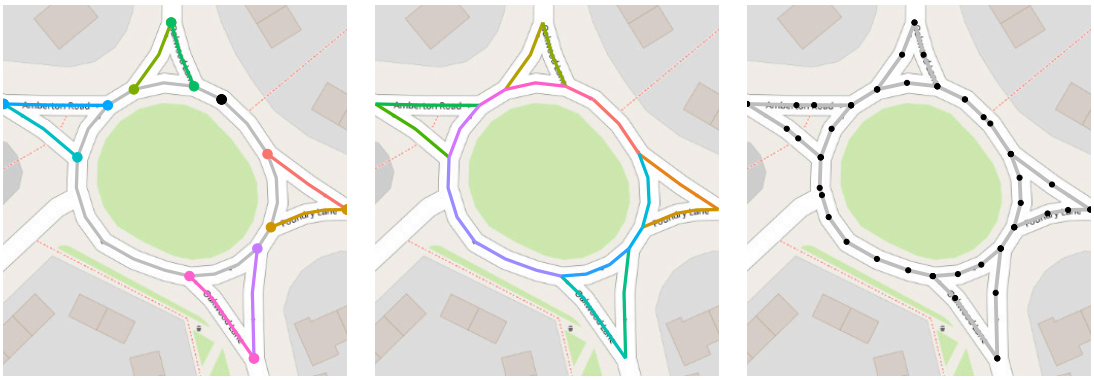
The `dodgr` approach immediately solves this problem by decomposing each `LINESTRING` into its minimal segments, while OSM itself ensures that each junction is represented by a shared vertex. This is clear looking at Figure 5(c).

5.2 Overpasses, underpasses and other types of intersections

The second problem that we analyse is strictly related to roundabouts and it concerns overpasses, underpasses and other types of street intersections.

Overpasses and underpasses could create a challenge for street network software since they represent a crossing of two highways located at different heights, where clearance to traffic on the lower level is obtained by elevating the higher level. From a routing perspective, this particular structure of the network must be taken into account since, even

[4]The `GRASS` tools can be accessed from `R` via *bridges* [26] such as `qgisprocess` [17] and `rgrass7` [7].



(a) Circular `LINESTRING`. (b) Routable `LINESTRING`. (c) `dodgr` approach.

Figure 5: *Left*: The network is unroutable using `stplanr` since the black dot, which is the unique boundary point of the roundabout, is not shared with any other street. *Center*: Now the network is routable since every boundary point that is shared between two different roads is also a node of the network. *Right*: The `dodgr` approach bypasses any problem dividing each `LINESTRING` into its underlying segments.

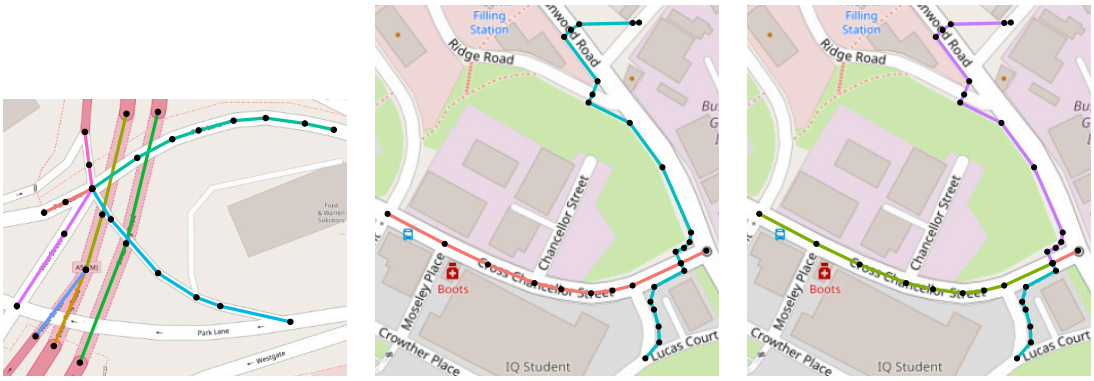
if we are always working in a two-dimensional space, it should never be possible to pass from one street-level to another. The algorithms behind `stplanr` and `dodgr` were designed taking care of this problem. An example of an overpass, located in the south of Leeds (UK), is reported in Figure 6(a). We overlayed a coloured map of the corresponding `stplanr` graph representation where there are connections only between streets belonging to the same level.

Another problem, strictly related to overpasses and roundabouts, is the following. There exist some streets in Open Street Map data that intersect each other, lie at the same level, but do not share any point in their boundaries. This implies that, from `stplanr`perspective, those streets are not connected. See, for example, Figure 6(b). This problem can also be solved (see Figure 5(c)) using the `rnet_breakup_vertices()` function.

The `dodgr` approach for representing street networks immediately solves both problems. Moreover, if the input `sf` data is built using `osmdata` package, then the assumption about the absence of shared nodes between roads at different levels can be removed. If fact, as we said in the previous sections, the comparisons between the vertices of the network are performed according to their Open Street Map ID(s), which are always unique, even where they share identical coordinates.

5.3 Oneway streets

Oneway streets prohibit certain transport modes (typically only motor traffic) from travelling in one direction. They are used in many cities to free up space for other land uses or dedicated cycleways or walkways, with a textbook example being Torrington Place in Lon-



(a) `stplanr` representation of an overpass.

(b) `stplanr` representation of an unroutable junction.

(c) The result of breaking up the street network. Each edge is depicted with a different colour.

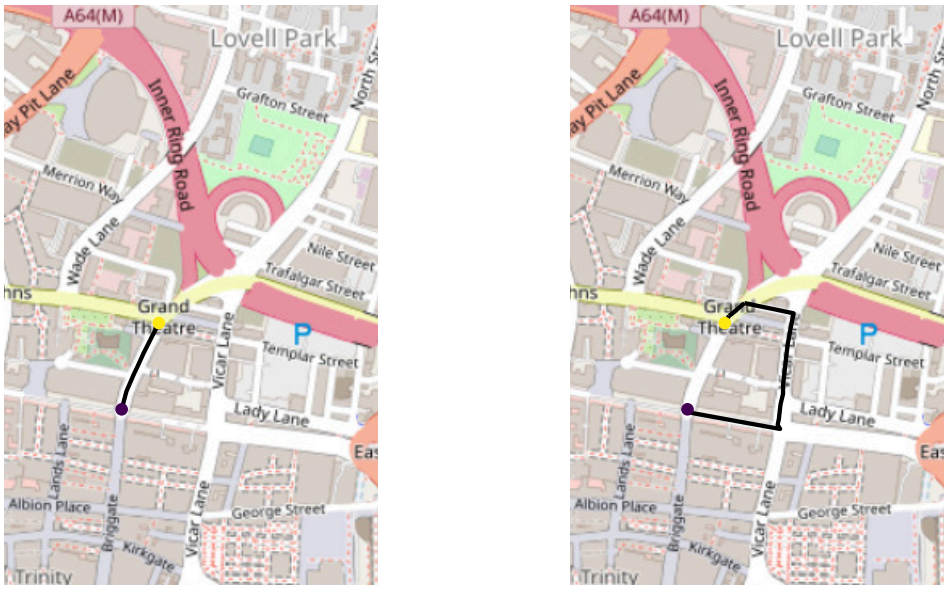
Figure 6: *Left*: `stplanr`-representation of one overpass where there are connections only between roads at the same level. *Center*: `stplanr`-representation of an unroutable junction. *Right*: The result obtained after splitting the junction.

don, where one carriageway was converted into a bidirectional cycleway[5]. Oneway streets pose a challenge from a routing perspective since, by definition, they allow vehicles to travel only in one direction.

At present, support for oneway streets has not been implemented in `stplanr`, meaning that for every two vertices in the network, the shortest path between them is symmetrical, even if that implies going against traffic in the real world. Oneway streets are supported by `dodgr`.

To illustrate the real-world consequences of this, Figures 7(a) and 7(b) demonstrate shortest paths calculated between the end points of an important oneway street in Leeds (UK). The path represented in the first figure, based on a shortest path calculation on the `sfNetwork` class, is shorter but against the law. The path represented in the second figure takes a longer route but follows the law. To activate oneway routing in `dodgr_streetnet` objects, there must be a column in the input `sf` object named *oneway*, typically derived from the *oneway tag* (see `https://wiki.openstreetmap.org/wiki/Key:oneway`) in Open Street Map. This column must be a logical vector, containing `TRUE` for streets that are oneway only and `FALSE` otherwise. This feature can be deactivated (e.g. when working with pedestrian or bicycle routing where directionality is generally unimportant) by removing the *oneway* column from the input data and then rebuilding the `dodgr_streetnet` object.

[5]A consultation report by Camden City Council proposing a oneway street on the Torrington Place / Tavistock Place Corridor, which has been implemented, provides a detailed introduction to oneway streets from a transport planning perspective. See `https://www.camden.gov.uk/documents/20142/3452947/Consultation+leaflet+FINAL.pdf/f628d6e8-c47b-82f1-cb40-8e24db78bea6` for details.



(a) Path in the contra-flow direction (New Briggate, Leeds, UK)

(b) Path in the right direction (New Briggate, Leeds, UK)

Figure 7: Examples of routing with oneway streets. The two paths are not identical since they include a oneway street.

5.4 A transport network

Having seen several types of street network components in the previous examples and the corresponding peculiar problems, this final example demonstrates what a citywide street network looks like, and how to create `sfNetwork` and `dodgr_streetnet` objects starting from an `sf` object with a `LINESTRING` geometry named `chapeltown_leeds`. We took highway data from *Geofabrik*[6] servers, considering a 5 km radius of the Chapeltown neighbourhood of Leeds (UK), an area the authors are familiar with, as the basis for the following examples.

The conversion between the pure `sf` format and the `stplanr` and `dodgr` representations is performed using the following commands:

```
street_network_stplanr <- SpatialLinesNetwork(rnet_breakup_vertices(chapeltown_leeds))
street_network_dodgr <- weight_streetnet(chapeltown_leeds, wt_profile = "bicycle")
```

A few notes:

1. The `rnet_breakup_vertices()` function was applied to the input `sf` data in the `stplanr` representation for the reasons explained in the previous examples. This

[6]Geofabrik is a website that provides extracts of Open Street Map data that are updated daily

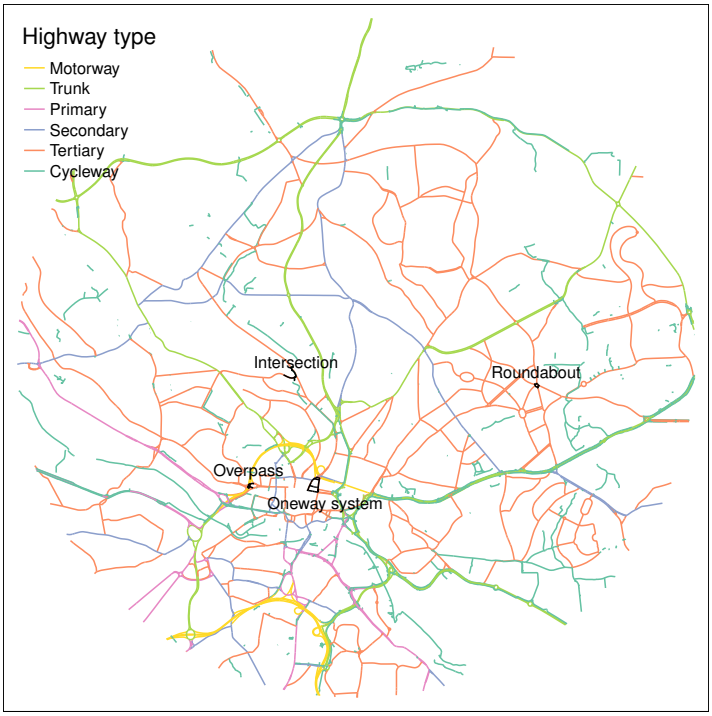


Figure 8: Illustration of a citywide street network, in Leeds (UK). For simplicity, we report only the most important highways.

function fixes several possible routing problems. It runs in approximately 3 seconds considering a road network of 12000 segments.

2. The `SpatialLinesNetwork()` function returns a warning message, and we are going to explain its meaning in the next section.

3. The `wt_profile` parameter in `weight_streetnet` is used to specify the preferred weighting profile, which is going to determine the dual weights.

Neither packages provides explicit tools for visualisation, other than a basic `plot()` method to show the geometry for `sfNetwork` objects. However, a wide range of visualisation packages can be used on the spatial versions of the data, that can be exported from the `@sl` slot of `sfNetwork` objects and via the `dodgr_to_sf()` function. See, for example, Figure 8. All the examples shown in the previous subsections were also taken from within this case study area.

6 Analysing street networks

In this Section, we present how to perform a few common operations on street networks with `stplanr` and `dodgr` packages. The examples are based on the road network built in the last example.

6.1 Connectivity

The first functions that we introduce are related to the identification of the *components* of a street network. The *connectivity* of a network is usually defined as follows: a graph G is said to be *connected* if there exists a sequence of edges going from each vertex to every other vertex. The *components* of the network are its sub-graph(s) formed by all connected vertexes [24].

The package `dodgr` defines a function, named `dodgr_components()`, which is used to identify the clusters of connected edges in a `dodgr_streetnet` object. The output is the same graph object in input with an additional column, called `component`, that identifies the component of each edge, sequentially numbering them starting from one (the largest component).

On the other side, `stplanr` package does not define any explicit function for a direct identification of the components of a street network. Nevertheless, the function `sln_clean_graph()` can be used for an automatic selection of all the edges belonging to the largest component, creating a fully connected graph. Moreover, as we saw with the previous example, `SpatialLinesNetwork()` may raise a warning every time its output is formed by two or more components, suggesting the adoption of `sln_clean_graph()` function. The same result can be obtained using the `dodgr` approach by filtering all the edges where the *component* column is equal to one. As we can see from Figure 8, there are several small clusters of roads in the Chapeltown road network, and we can exclude them with the following commands:

```
street_network_stplanr <- sln_clean_graph(street_network_stplanr)
street_network_dodgr <- street_network_dodgr[street_network_dodgr$component == 1, ]
```

6.2 Shortest paths

We present now several functions that can be used for the estimation of shortest (in terms of geographical distance) and fastest (according to the dual weights set by `dodgr` package) paths between two locations: Monk Bridge, which lies in the south-west of Leeds City Center, and Chapeltown neighbourhood, located in the north-east. The first step is the geo-coding of the coordinates of the two points through `stplanr::geo_code()` via the *Nominatim* service:

```
leeds_monk_bridge <- geo_code("leeds monk bridge")
chapeltown <- geo_code("Leeds Chapeltown")
```

The `route_local()` function can be used for the evaluation of the shortest path according to the `stplanr` approach. The inputs of the function are an `sfNetwork` object, representing the street network, and the coordinates of the start and end points, either as numeric values or text strings. The following command estimates the shortest path between Monk Bridge and Chapeltown, given the road network built in the previous examples.

```
stplanr_shortest <- route_local(street_network_stplanr, leeds_monk_bridge, chapeltown)
```

The output of `route_local()` is an `sf` object, which is created as a subset of the original input data, containing only the edges connecting the two points through the shortest path.

The `dodgr` package offers multiple options for estimating shortest and fastest paths on a street network. The main function is called `dodgr_paths()` and it works as follows:

```
dodgr_fastest_ids <- dodgr_paths(
street_network_dodgr, leeds_monk_bridge, chapeltown, vertices = FALSE
)
```

The inputs are a `dodgr_streetnet` object, representing the street network, and two matrices (or vectors) of numeric coordinates for the start and end points. The output is a list of paths tracing the connections between the points, either as a list of vertices or edges ID(s) (according to the `vertices` parameter). `dodgr` functions are optimized for many-to-many pairwise distances on dual-weighted graphs, so the element `dodgr_fastest_ids[[i]][[j]]` contains the path from the ith starting point to the jth ending point. The examples presented in this paper are based on just two points, so the indexes of the vertices composing the best route between Monk Bridge and Chapeltown can be extracted with `dodgr_fastest_ids[[1]][[1]]`. The fastest path can be reconstructed as a `dodgr_streetnet` dataframe of edges as follows:

```
dodgr_fastest_path <- street_network_dodgr[dodgr_fastest_ids[[1]][[1]], ]
```

Both routes are reported in Figure 9(a). The shortest path, suggested by `route_local()` and coloured in dark-green, is different from the fastest path, suggested by `dodgr_paths()` and coloured in dark-red. We can see that the route chosen by `stplanr` is going through several trunks and motorways, while `dodgr` path prefers tertiary roads and cycleways. This difference is due to the fact that the fastest way is optimised for bicycle routing using a dual-weights system.

More precisely, as we mentioned in the previous sections, the real advantage of `dodgr` approach is the ability to calculate paths on *dual-weighted graphs*. The fastest path is calculated according to one set of weights, while the resultant distances are calculated by accumulating a different set of weights along the resultant path. This is particularly important for realistic transport routing, where different kinds of ways are more or less suitable for different kinds of transport. The true shortest path for a pedestrian may be along an eight-lane highway, but they are never likely to actually traverse that way. Instead, that eight-line highway should be weighted to yield an effective length that is longer than the actual geographical length. `dodgr` offers a range of *weighting profiles*, listed using the following

R command, and detailed in the help page of the included `dodgr::weighting_profiles` data set.

```
wp <- weighting_profiles
unique(wp$weighting_profiles$name)
#> "foot" "horse" "wheelchair" "bicycle" "moped" "motorcycle" "motorcar" "goods"
#> "hgv" "psv"
```

Street networks can be weighted for transport of any of the associated types, by entering the name as the `wt_profile` parameter. The following code demonstrates what one profile actually looks like.

```
head(wp$weighting_profiles [wp$weighting_profiles$name == "foot", ])
#>     name    way           value    max_speed
#> 1   foot    motorway        0.0           NA
#> 2   foot    trunk           0.4           NA
#> 3   foot    primary         0.5            5
#> 4   foot    secondary       0.6            5
#> 5   foot    tertiary        0.7            5
#> 6   foot    unclassified    0.8            5
```

We can check the differences between two weighting profiles by estimating the fastest path between Monk Bridge and Chapeltown according to a *motorcar* type of transport. We do not repeat the commands since they are analogous to the previous example, but the fastest path is reported in Figure 9(b). We can see that the routes suggested by `stplanr` and `dodgr` are almost identical, but for a few minor differences in the City Center due to the fact that the shortest path estimated by `route_local()` is going against the flow.

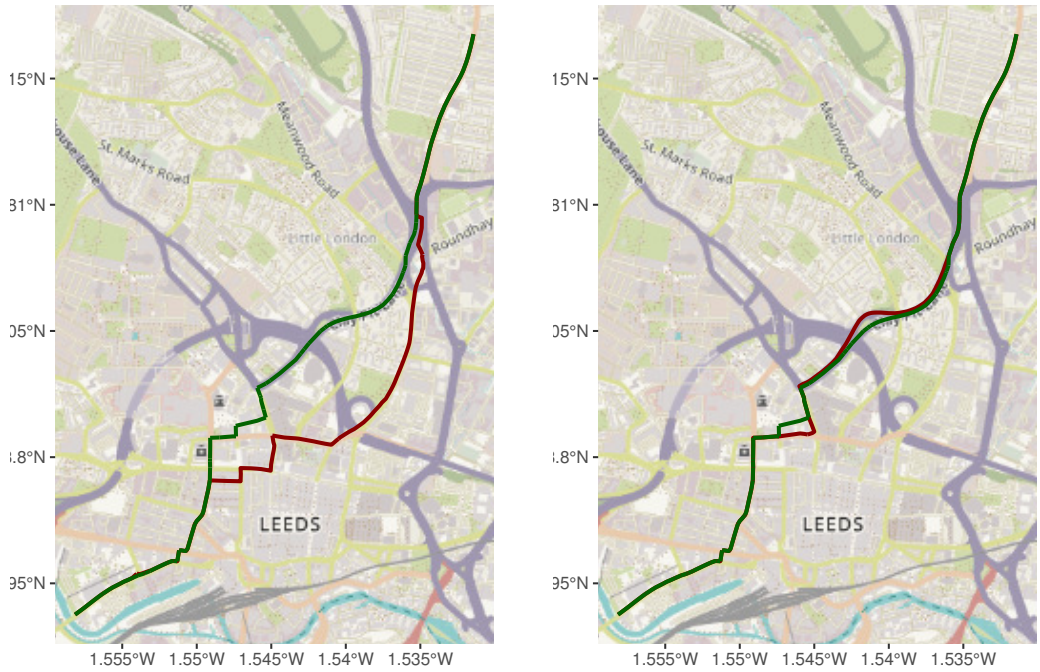
The same ideas can be tested using the `dodgr_dists()` function. The following code estimates the geographical distance between Monk Bridge and Chapeltown according to a bicycle weighting profile using the fastest path:

```
dodgr_dists(street_network_dodgr , leeds_monk_bridge , chapeltown , shortest = FALSE)
#> 8023
```

or the shortest path:

```
dodgr_dists(street_network_dodgr , leeds_monk_bridge , chapeltown , shortest = TRUE)
#> 3704
```

`dodgr` also offers the ability to incorporate elevation data, and to take account of the effect of elevation changes on travel times. While elevation has little or no effect on motorised transport, it has very important effects on human-powered modes of transport (walking and bicycling). Elevation effects can currently only be incorporated when the underlying network is represented in is represented in `silicate` (sc) format (see `https://github.com/hypertidy/silicate`), and we refer to the corresponding `R` package repository for more details. This effectively only involves replacing the `dodgr_streetnet()` function with `dodgr_streetnet_sc()`. The final `dodgr` network will appear largely the same but will incorporate effects of elevation changes, as well as waiting times at traffic



(a) Shortest and fastest paths according to a bicycle weighting profile.

(b) Shortest and fastest paths according to a motorcar weighting profile.

Figure 9: Graphical example of shortest paths between Monk Bridge and Chapeltown (UK). The dark green route is the path suggested by `stplanr`, while the dark red route is the path suggested by `dodgr`. The route on the left is optimized according to a bicycle mode of transport, while the route on the right is optimized according to a motorcar transport.

lights, and time penalties for waiting to turn across oncoming traffic. Time-based routing using these `dodgr` networks will reflect highly realistic transit behaviour.

Finally `dodgr` also offers the functions `dodgr_isodists()` and `dodgr_isochrones()` to calculate the respective isocontours from a given set of starting points. Like all other `dodgr` functions, these are computationally optimised for highly efficient parallel calculation of isocontours from large numbers of starting points.

6.3 Graph metrics

A strength of the `stplanr` approach to street networks analysis is that its `sfNetwork` class contains an `igraph` object, opening-up a wide range of algorithms provided in the `igraph` package [24]. A basic characteristic of any graph is whether or not it is simple, meaning all nodes are connected by only one edge (or either one or two edges for directed graphs). We

can use the igraph function is.simple() to test the street_network_stplanr citywide street network:

```
is.simple(street_network_stplanr@g)
```

The graph representation of a street network is typically not simple. Another example is provided in the following command, which estimates the edge betweenness measure of centrality that indicates the number of shortest paths that pass through the edges, considering all nodes or some subset of them:

```
edge_betweenness(street_network_stplanr@g)
```

Other igraph functions can calculate other network characteristics related to the vertices, such as their *closeness centrality* (which is a measure of the distance from a vertex to all others) or their *degree* (the number of edges incident to each vertex). The following command, to provide another example, estimates the vertex *strength* (defined as the sum of the lengths of all edges incident to a given vertex). Note that, by default, the weights of the edges are the lengths of the corresponding LINESTRING objects.

```
strength(street_network_stplanr@g)
```

At present, there is no functionality in stplanr for linking such vertex metrics to the corresponding spatial coordinates (an issue we plan to address in future work). Still, vertex metrics can be summarised visually, to provide an overview of the structure of the street network that can be compared with street networks from other cities, to explore concepts such as interdependence and resilience [30]. A graphical summary of the edge betweenness and vertex strength metrics is reported in Figures 10(a) and 10(b).

The authors of stplanr recently extended the integration between igraph and sfNetworks objects, defining a new function named rnet_group that can be used to explore the spatial distribution of several algorithms for *community detection* [19]. The following code can be used to estimate the *community* structure of the spatial network according to an algorithm defined in [9].

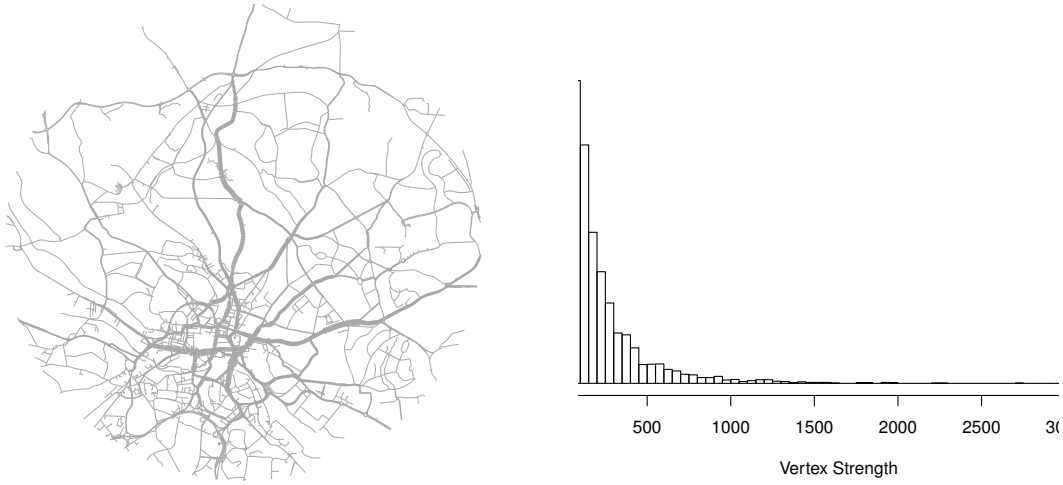
```
rnet_group(street_network_stplanr, igraph::cluster_louvain)
```

The output is an sfNetwork object with an extra column in the sl slot named *rnet_group* that summarizes the partitions.

dodgr also supports network analysis. The function dodgr_centrality() calculates betweenness centrality, with an implementation that can be computed using parallel processing capabilities of modern computers and which enables centrality to be estimated with respect to weighted measures of distance or time. The following command calculates edge betweenness centrality, resulting in a dodgr_streetnet object that has an additional column called centrality.

```
dodgr_edges_betweennes <- dodgr_centrality(street_network_graph)
```

Vertex betweenness centrality measures can be calculated by setting the edges argument to FALSE in the dodgr_centrality() function. The output is a data.frame object with



(a) igraph: edge betweenness centrality metric.

(b) igraph: vertices strength metric.

Figure 10: Graphical summary of two network metrics estimated using `igraph` algorithms. *Left*: We can see how the most important highways in Leeds are highlighted by the betweenness centrality measure. *Right*: Histogram of vertex strength. Several vertices are linked to only a few short edges.

a column called `centrality`. A graphical comparison of edge and vertex betweenness is reported in Figures 11(a) and 11(b).

A characteristic of Open Street Map data is that vertex locations are, to some extent, arbitrary, which can lead to overestimates of centrality near road segments with arbitrarily high numbers of vertices per unit distance. `dodgr` addresses this issue by enabling betweenness centrality measures to be calculated with *contracted* street networks using the `contracted` argument (which defaults to `TRUE`). We refer to the help page of `dodgr_centrality()` for details, including the `dist_threshold` argument which constrains path lengths on which centrality estimates are based. Passing suitable values to `dist_threshold` (and the equivalent `cutoff` argument in `igraph` functions) can improve the computational efficiency of street network centrality calculations.

7 Discussion and Conclusion

This paper has demonstrated that street networks, a particular type of spatial network representing transport systems, can be encoded in classes that build on pre-existing data structures that are available in the statistical programming language `R`. We explored the

(a) dodgr: edge betweenness centrality metric.

(b) dodgr: vertices centrality metric..

Figure 11: Graphical summary of edge and vertex betweenness centrality measure estimated using `dodgr`. We can see that both metrics highlight the most important roads of Leeds city area.

representation of a range of street network features in `stplanr` and `dodgr`, two recently developed `R` packages that provide explicit support for street networks building on pre-existing geographical/graph and simple data frame class definitions, respectively. Overall, we found that both approaches can be used for analysing street networks, with support for shortest path calculation, characterisation of networks, nodes and edges, and the ability to modify networks. Such capabilities can be (and to some extent are already being) implemented in other languages for interactive data analyisis, as demonstrated by the established `Python` package `osmnx` [10] and the recently developed `Julia` package `OpenStreetMap.jl`.

Perhaps more important than languages of implementation are the underlying concepts and real-world applications. We found that concepts of network pre-processing, including the vital stage of breaking-up linestrings at junction intersections for approaches based on geographic data structures and weighting profiles/contraction, are vital for effective use of street network data, regardless of language of implementation or the data structures used to represent the street network. Although the approaches have been applied in this paper on datasets designed to highlight the techniques rather than to answer applied research questions, it seems clear that both approaches can help answer important research questions, including:

- What is the relative circuity (divergence factor) for motorised and non-motorised modes in different areas? - What are the relationships between street network characteris-

tics and travel behaviour (e.g. are some network forms associated with more walking and cycling)? - What are the optimal places to intervene on the road network to improve transport system performances, e.g. based on environmental, health or journey time performance metrics?

Answers to these and many other research questions have real-world applications, particularly in transport planning. Perhaps key test of the packages will therefore be whether they see widespread uptake, beyond niche applications [27], of the type that spatial `R` packages such as `sf` have seen.

`stplanr` and `dodgr` each have strengths and limitations that make them more or less appropriate for different tasks. Building on established spatial and graph packages, `stplanr`'s `sfNetwork` can be analysed using a wide range of spatial and graph functions. Its graph structure can be linked and analysed with several algorithms implemented in `igraph`. A downside of `stplanr`, at the time of writing, is that it lacks support for routing features such as in-built weighting profiles for different modes of transport and the representation of one-way streets. Adding these features, either directly in `stplanr` or other packages that build on the geographical/graph data structures it uses, could represent a promising direction of future development that would address this limitation. `dodgr`'s class system, by contrast, is more specifically focussed on Open Street Map data and routing, with support for a variety of modes and oneway streets being key strengths. The examples presented in Sections 5 and 6 exhibit the relevance of dual-weights for realistic routing. These relative strengths and weaknesses raise the question of priorities for future development, which could go in various directions including better support for routing in `stplanr` to integration with other spatial classes in `dodgr` (which can already be translated to established graph and spatial classes).

An article on the *r-spatial* blog (see `https://www.r-spatial.org/r/2019/09/26/spatial-networks.html`) reports an alternative way of representing dual `sf`/`igraph` objects that uses `tidygraph` as the basis for a data frame-like representation of spatial networks [36]. This approach offers some potential advantages in terms of usability and could be adopted as the basis of future street network classes, but it is not yet mature enough to compare with `stplanr` and `dodgr`. A range of alternative approaches, as yet unexplored, may also be advantageous, for example using `R` as an interface to high performance graph libraries such as `sDNA` [13]. While such options remain relatively unexplored, `stplanr` and `dodgr` have been tested, are actively maintained, and provide class structures that open-up street network analysis to reproducible analysis. Although the paper has focussed on the implementation of classes and techniques in one language, the approaches are language agnostic. We hope that this paper motivates further development of open source software for reproducible and command-line driven street network analysis and applied research building on the techniques demonstrated in this paper to answer questions to support evidence-based interventions on transport networks in cities worldwide.

Acknowledgements

The `ggspatial` [16] maps are created using tiles from © OpenStreetMap Contributors Tiles.

References

[1] (OGC) Open Geospatial Consortium Inc. *OpenGIS Implementation Specification for Geographic information - Simple feature access - Part 1: Common architecture*. Tech. rep. OGC 06-103r4. (OGC) Open Geospatial Consortium Inc., 2011. URL: `https://www.ogc.org/standards/sfa`.

[2] Jennings Anderson, Dipto Sarkar, and Leysia Palen. "Corporate Editors in the Evolving Landscape of OpenStreetMap". In: *ISPRS International Journal of Geo-Information* 8.5 (May 2019), p. 232. ISSN: 2220-9964. DOI: `10.3390/ijgi8050232`. URL: `http://dx.doi.org/10.3390/ijgi8050232`.

[3] Adrian Baddeley, Ege Rubak, and Rolf Turner. *Spatial point patterns: methodology and applications with R*. CRC press, 2015.

[4] Adrian Baddeley et al. "Analysing point patterns on networks—A review". In: *Spatial Statistics* (2020), p. 100435.

[5] Christopher Barrington-Leigh and Adam Millard-Ball. "The world's user-generated road map is more than 80% complete". In: *PLOS ONE* 12.8 (Aug. 2017). ISSN: 1932-6203. DOI: `10.1371/journal.pone.0180698`. (Visited on 08/14/2017).

[6] Marc Barthelemy. "Spatial networks". In: *Physics Reports* 499.1-3 (Feb. 2011), pp. 1–101. ISSN: 0370-1573.

[7] Roger Bivand. *rgrass7: Interface Between GRASS 7 Geographical Information System and R*. R package version 0.2-1. 2019. URL: `https://CRAN.R-project.org/package=rgrass7`.

[8] Roger S. Bivand, Edzer Pebesma, and Virgilio Gómez-Rubio. *Applied Spatial Data Analysis with R*. en. 2nd ed. Springer Science & Business Media, June 2013.

[9] Vincent D Blondel et al. "Fast unfolding of communities in large networks". In: *Journal of statistical mechanics: theory and experiment* 2008.10 (2008), P10008.

[10] Geoff Boeing. "OSMnx: New methods for acquiring, constructing, analyzing, and visualizing complex street networks". In: *Computers, Environment and Urban Systems* 65 (2017), pp. 126–139.

[11] Alessio Cardillo et al. "Structural properties of planar graphs of urban street patterns". In: *Physical Review E* 73.6 (2006), p. 066107.

[12] Richard J Chorley and Petercoed Haggett. *Models in geography*. Methuen and Co., 1967.

[13] Crispin HV Cooper and Alain JF Chiaradia. "sDNA: 3-d spatial network analysis for GIS, CAD, Command Line & Python". In: *SoftwareX* 12 (2020), p. 100525.

[14] Paolo Crucitti, Vito Latora, and Sergio Porta. "Centrality measures in spatial networks of urban streets". In: *Physical Review E* 73.3 (2006), p. 036125.

[15] Gabor Csardi and Tamas Nepusz. "The igraph software package for complex network research". In: *InterJournal* Complex Systems (2006), p. 1695. URL: `https://igraph.org`.

[16] Dewey Dunnington. *ggspatial: Spatial Data Framework for ggplot2*. R package version 1.1.4. 2020. URL: `https://CRAN.R-project.org/package=ggspatial`.

[17] Dewey Dunnington. *qgisprocess: Use 'QGIS' Processing Algorithms*. R package version 0.0.0.9000. 2020. URL: `https://github.com/paleolimbot/qgisprocess`.

[18] Leonhard Euler. "Solutio problematis ad geometriam situs pertinentis". In: *Comment. Acad. Sci. Petropolitanae 8* (1741), pp. 128–140.

[19] Santo Fortunato. "Community detection in graphs". In: *Physics reports* 486.3-5 (2010), pp. 75–174.

[20] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. "Exploring Network Structure, Dynamics, and Function using NetworkX". In: *Proceedings of the 7th Python in Science Conference*. Ed. by Gaël Varoquaux, Travis Vaught, and Jarrod Millman. Pasadena, CA USA, 2008, pp. 11–15.

[21] Peter Haggett and Richard J Chorley. *Network analysis in geography*. Vol. 1. Hodder Education, 1969.

[22] Bin Jiang. "A topological pattern of urban street networks: universality and peculiarity". In: *Physica A: Statistical Mechanics and its Applications* 384.2 (2007), pp. 647–655.

[23] Alireza Karduni, Amirhassan Kermanshah, and Sybil Derrible. "A protocol to convert spatial polyline data to network formats and applications to world urban road networks". In: *Scientific Data* 3.1 (2016), pp. 1–7.

[24] Eric D. D. Kolaczyk and Gabor Csardi. *Statistical Analysis of Network Data with R*. English. 2014 edition. New York, NY: Springer, May 2014. ISBN: 978-1-4939-0982-7.

[25] Stefan Lämmer, Björn Gehlsen, and Dirk Helbing. "Scaling laws in the spatial structure of urban road networks". In: *Physica A: Statistical Mechanics and its Applications* 363.1 (2006), pp. 89–95.

[26] Robin Lovelace, Jakub Nowosad, and Jannes Muenchow. *Geocomputation with R*. CRC Press, 2019. ISBN: 1-138-30451-4. URL: `http://robinlovelace.net/geocompr` (visited on 10/05/2017).

[27] Robin Lovelace et al. "The Propensity to Cycle Tool: An open source online system for sustainable transport planning". en. In: *Journal of Transport and Land Use* 10.1 (Jan. 2017). ISSN: 1938-7849. DOI: 10.5198/jtlu.2016.862. URL: https://www.jtlu.org/index.php/jtlu/article/view/862 (visited on 06/01/2017).

[28] Binbin Lu et al. "Shp2graph: Tools to Convert a Spatial Network into an Igraph Graph in R". In: *ISPRS International Journal of Geo-Information* 7.8 (2018), p. 293. URL: https://doi.org/10.3390/ijgi7080293.

[29] Stephen Marshall et al. "Street network studies: from networks to models and their representations". In: *Networks and Spatial Economics* 18.3 (2018), pp. 735–749.

[30] Richard G. Morris and Marc Barthelemy. "Spatial Effects: Transport on Interdependent Networks". en. In: *Networks of Networks: The Last Frontier of Complexity*. Ed. by Gregorio D'Agostino and Antonio Scala. Understanding Complex Systems. Cham: Springer International Publishing, 2014, pp. 145–161. ISBN: 978-3-319-03518-5. DOI: 10.1007/978-3-319-03518-5_7. URL: https://doi.org/10.1007/978-3-319-03518-5_7 (visited on 04/21/2020).

[31] Open Street Map. *Open Street Map data - Accuracy*. Accessed: 2020-12-05. 2020. URL: https://wiki.openstreetmap.org/wiki/Accuracy#Topology.

[32] OpenStreetMap contributors. *Planet dump retrieved from https://planet.osm.org*. 2017. URL: https://www.openstreetmap.org.

[33] Mark Padgham. "dodgr: An R package for network flow aggregation". In: *Transport Findings* (Feb. 2019). DOI: 10.32866/6945.

[34] Mark Padgham et al. "osmdata". In: *The Journal of Open Source Software* 2.14 (June 2017). DOI: 10.21105/joss.00305. URL: https://doi.org/10.21105/joss.00305.

[35] Edzer Pebesma. "Simple Features for R: Standardized Support for Spatial Vector Data". In: *The R Journal* 10.1 (2018), pp. 439–446. DOI: 10.32614/RJ-2018-009. URL: https://doi.org/10.32614/RJ-2018-009.

[36] Thomas Lin Pedersen. *tidygraph: A Tidy API for Graph Manipulation*. R package version 1.2.0. 2020. URL: https://CRAN.R-project.org/package=tidygraph.

[37] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria, 2020. URL: https://www.R-project.org/.

[38] Suman Rakshit, Adrian Baddeley, and Gopalan Nair. "Efficient Code for Second Order Analysis of Events on a Linear Network". In: *Journal of Statistical Software* 90.1 (2019), pp. 1–37. DOI: 10.18637/jss.v090.i01.

[39] Robin Lovelace and Richard Ellison. "stplanr: A Package for Transport Planning". In: *The R Journal* 10.2 (2018). URL: https://doi.org/10.32614/RJ-2018-053.

[40] John Snow. *On the Mode of Communication of Cholera*. John Churchill, 1855.

[41] Christian L. Staudt, Aleksejs Sazonovs, and Henning Meyerhenke. "NetworKit: A tool suite for large-scale complex network analysis". en. In: *Network Science* 4.4 (Dec. 2016), pp. 508–530. ISSN: 2050-1242, 2050-1250. DOI: 10.1017/nws.2016.20. (Visited on 04/30/2020).