Journal of Statistical Software

MMMMMM YYYY, Volume VV, Issue II. *doi: 10.18637/jss.v000.i00*

cubble: An R Package for Structuring Spatio-temporal Data

H. Sherry Zhang
Monash University

Dianne Cook
Monash University

Ursula Laa
University of Natural Resources and Life Sciences

Nicolas Langrené
CSIRO Data61

Patricia Menéndez
Monash University

Abstract

The abstract of the article.

Keywords: spatio-temporal data, R.

1. Introduction

Spatio-temporal data record changes of variables in spatially separated regions across time. In this article, we consider spatio-temporal vector data, which are recorded in a fixed interval and are point based, characterised by longitude and latitude, in the spatial aspect. Examples of this type of data include the house price of a city or county, climate measures from weather stations in a country, and river level data from electronic gauges.

Analysing this type of data requires less considerations on the geographical geometry type and map projection but more on how measures in these fixed locations changes across the time domain and whether these changes are related for adjacent locations. For example, when nearby areas show patterns that are regular enough, visualising spatio-temporal data can 1) discover regional time series features, i.e. trend and seasonality, 2) find the Waldo sites from the crowd, and 3) see how correlation of nearby sites changes across time.

The main difficulty in visualising this type of data is to show information in both space and time dimension with the proper level of details without information overflow. This would sometimes require aggregating the time dimension into the proper level or slicing the data into a reasonable number of subset for display. In this sense, a data structure that regulates the manipulation spatio-temporal data will benefit the analysis workflow. While many implementations focus on manipulating and visualising pure spatial or temporal data, there are not sufficient tools to deal with spatio-temporal data. The purpose of this paper is to introduce a spatio-temporal vector data structure for data analysis in R.

The rest of the paper will be divided as follows: Section 2 reviews the existing data structure for spatio, temporal, and spatio-temporal data. Section 3 presents a new data structure for spatio-temporal data: cubble. Then the paper introduces the workflow of data manipulation and visualisation with the cubble structure in Section 4. Section 5 gives some examples on how common spatial and temporal manipulations are performed with cubble and how static and interactive visualisation help to understand climate and [. . .] data.

2. Existing data structure for spatio and temporal data

Below we review some structure for spatial, temporal, and spatio-temporal data.

Many spatial and spatio-temporal data structures have been developed by the R-spatial team for both raster and vector spatial data. For vector spatial data, which is the focus of this paper, `sf` (Pebesma 2018) represents spatial vector information with simple features: points, lines, polygons and their multiples. Various `st_` function are designed to manipulate these features based on their geometric relationships. For spatio-temporal data, `stars` (Pebesma 2021) can represent both raster and vector data using multi-dimensional array. However, the underlying array structure can be difficult to operate for data analysts who are more familiar with a flat 2D data frame structure used by the tidyverse ecosystem.

In the temporal aspect, the `tsibble` (Wang, Cook, and Hyndman 2020) structure and its tidyverts ecosystem have provided a [. . .] workflow to work with temporal data. In a tsibble structure, temporal data is characterised by `index` and `key` where `index` is the temporal identifier and `key` is the identifier for multiple series, which could be used as a spatio identifier. However, a tsibble object, by construction, always requires the `index` in its structure. This makes it less appealing for spatio-temporal data since the output of calculated spatio-specific

variables (i.e. features of each series) don't have the time dimension. Analysts will either need to have an additional step to join this output to the original tsibble or operate with variables stored in two separate objects. In addition, the long form structure of a tsibble object means spatio variables (i.e. longitude, latitude, and features of each series if joined back to the tsibble) of each spatio identifier will be repetitively recorded at each timestamp. This repetition is unnecessary and would inflate the object size for long series.

Under the third tidy data principle [@tidydata], these are two type of observational unit and should form different tables. An example (Table 8) is also given to demonstrate the billboard dataset where there are song variables and rank variables that changes each week. The other has mentioned the lack of tool to work directly with relational data and analysis will usually need to merge the dataset into one table, which could causes errors. These days more implementations have looked at relational data structure and in R, several good examples include, 'tidygraph'[@tidygraph] for graph manipulation, 'dm' [@dm] for relational data model. In the domain of spatio-temporal data, @spacetime has considered four types of spatio-temporal layout. 'Spatstat' [@Spatstat] an implementation of point pattern.

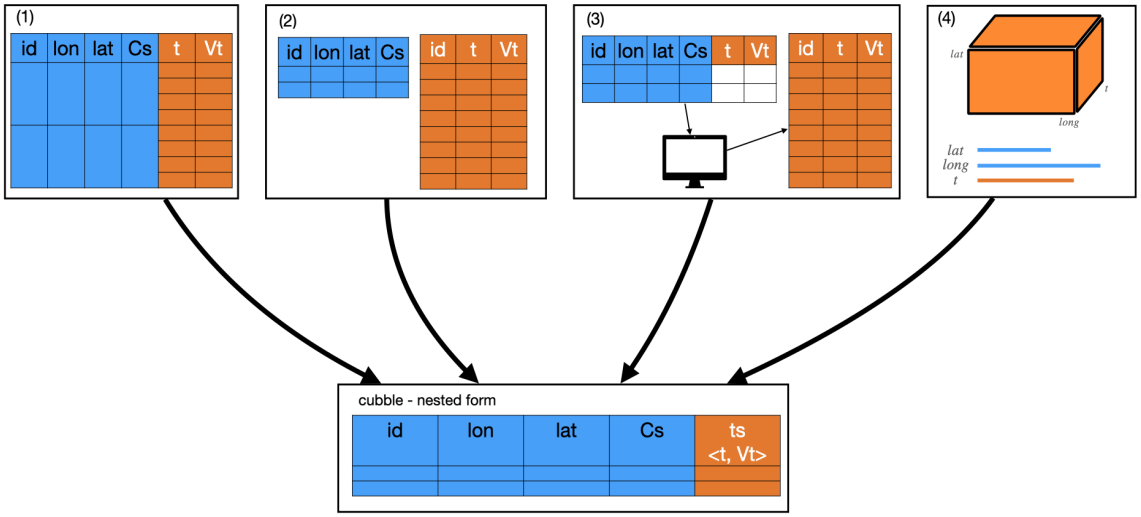

Figure 1: Cubble diagram

3. A new data structure for spatio-temporal data

Spatio-temporal data don't usually come to the analysts as a whole piece. A way to look at these data is to divide it into spatial and temporal dimension with an ID that links between the two. The first row in Figure 1 illustrates this representation where in the spatial dimension, the data is characterised by `id`, `lat`, `long`. V_s in the last column represents all the other site-wise variables, for example, elevation and full name etc. The temporal dimension, on the other hand, can be characterised by `id` and `t` with V_t representing all the time-wise variables. In climate data, this could include precipitation, maximum or minimum temperature, and wind speed etc.

To work with spatio-temporal data, analysts can choose to either work separately on each dimension or join the two sets together, however, each approach has its own problem: While is is natural to work separately on each sheet (since spatial and temporal operations usually don't overlap), analysts will need to manually keep the other data frame up to date. For example, the following pseudo code illustrates the scenario where once the spatial dataset is filtered for those within Victoria, the temporal dataset needs to be manually updated to reflect this spatial filter.

```
R> spatial_new <- spatial %>% filter(SITES_IN_VICTORIA)
R> temporal_new <- temporal %>% filter(id %in% spatial_new$id)
```

If analysts choose to join the spatial and temporal data together, the joined dataset could be too large since each spatial variable will be repeated at each time stamp for each site. Also, recordings of the site ID from different data sources can be slightly different from each other, causing a painful checking and cleaning of site IDs before the join.

A cubble, in essence, wires both dimensions in the spatio-temporal data into one object while provide two forms for manipulation the spatial and temporal dimension separately.

When manipulating the spatial dimension it uses a nest form that:

- defines each group in a row,
- displays the group-related variables in columns, and
- nests all the time-related variables into a column called `ts`.

When manipulating the temporal dimension, it uses the long form that:

- each combination of group and timestamp occupies a row
- time-related variables are displayed, and
- group-related variables are not explicitly displayed but can be accessed through the `meta` attribute.

4. Create a cubble

The creation of a cubble requires the site identifier (`key`), as well as the spatial (`coords`) and temporal (`index`) identifier. `climate_flat` is already a tibble and it uses `id` to identify each station, `date` as the time identifier, and `c(long, lat)` as the spatial identifier. To create a cubble for this data, use:

```
R> climate_flat %>% as_cubble(key = id, index = date, coords = c(long, lat))

# cubble:   id [5]: nested form
# bbox:     [115.97, -32.94, 133.55, -12.42]- check gap on long and lat
# temporal: date [date], prcp [dbl], tmax [dbl], tmin [dbl]
  id            lat  long  elev name           wmo_id ts
  <chr>       <dbl> <dbl> <dbl> <chr>           <dbl> <list>
1 ASN00009021 -31.9  116.  15.4 perth airport   94610 <tibble [366 x 4]>
2 ASN00010311 -31.9  117. 179   york            94623 <tibble [366 x 4]>
3 ASN00010614 -32.9  117. 338   narrogin        94627 <tibble [366 x 4]>
4 ASN00014015 -12.4  131.  30.4 darwin airport  94120 <tibble [366 x 4]>
5 ASN00015131 -17.6  134. 220   elliott         94236 <tibble [366 x 4]>
```

Most of the time, spatio-temporal data doesn't come into this form and analysts need to query the climate variables based on station metadata. This is also a problem illustrated in Section 3.5 in @tidydata. Here we provide a structured way to query this data based on the row-wise operator and nested list. For this type of task, one can structure a metadata into a tibble and use row-wise operator to query the climate variables into a nested list. As an example here we demonstrate the workflow to find the 5 closest stations to Melbourne. We first create a station data frame with the 5 target stations.

```
# A tibble: 5 x 8
  id            lat  long  elev name                 wmo_id  dist city
  <chr>       <dbl> <dbl> <dbl> <chr>                 <dbl> <dbl> <chr>
1 ASN00086038 -37.7  145.  78.4 essendon airport      95866  10.8 melbourne
2 ASN00086282 -37.7  145. 113.  melbourne airport     94866  20.1 melbourne
3 ASN00086077 -38.0  145.  12.1 moorabbin airport     94870  21.9 melbourne
4 ASN00088162 -37.4  145. 528.  wallan (kilmore gap)  94860  48.1 melbourne
5 ASN00087113 -38.0  144.  10.6 avalon airport        94854  48.8 melbourne
```

We can query the climate information into a nested list named `ts` for each station with the `rowwise()` operator. To create a cubble, supply the same identifiers as with the first example.

```
R> sydmel_climate <- stations %>%
+   rowwise() %>%
+   mutate(ts = list(meteo_pull_monitors(id,
+                                        date_min = "2020-01-01",
+                                        date_max = "2020-12-31",
+                                        var = c("PRCP", "TMAX", "TMIN")) %>%
+                      select(-id))) %>%
+   as_cubble(key = id, index = date, coords = c(long, lat))
```

```
# cubble:   id [5]: nested form
# bbox:     [144.47, -38.03, 145.1, -37.38]
# temporal: date [date], prcp [dbl], tmax [dbl], tmin [dbl]
  id            lat  long  elev name                wmo_id  dist city   ts
  <chr>       <dbl> <dbl> <dbl> <chr>                <dbl> <dbl> <chr>  <list>
1 ASN00086038 -37.7  145.  78.4 essendon airport      95866  10.8 melbo~ <tibbl~
2 ASN00086282 -37.7  145. 113.  melbourne airport     94866  20.1 melbo~ <tibbl~
3 ASN00086077 -38.0  145.  12.1 moorabbin airport     94870  21.9 melbo~ <tibbl~
4 ASN00088162 -37.4  145. 528.  wallan (kilmore gap)  94860  48.1 melbo~ <tibbl~
5 ASN00087113 -38.0  144.  10.6 avalon airport        94854  48.8 melbo~ <tibbl~
```

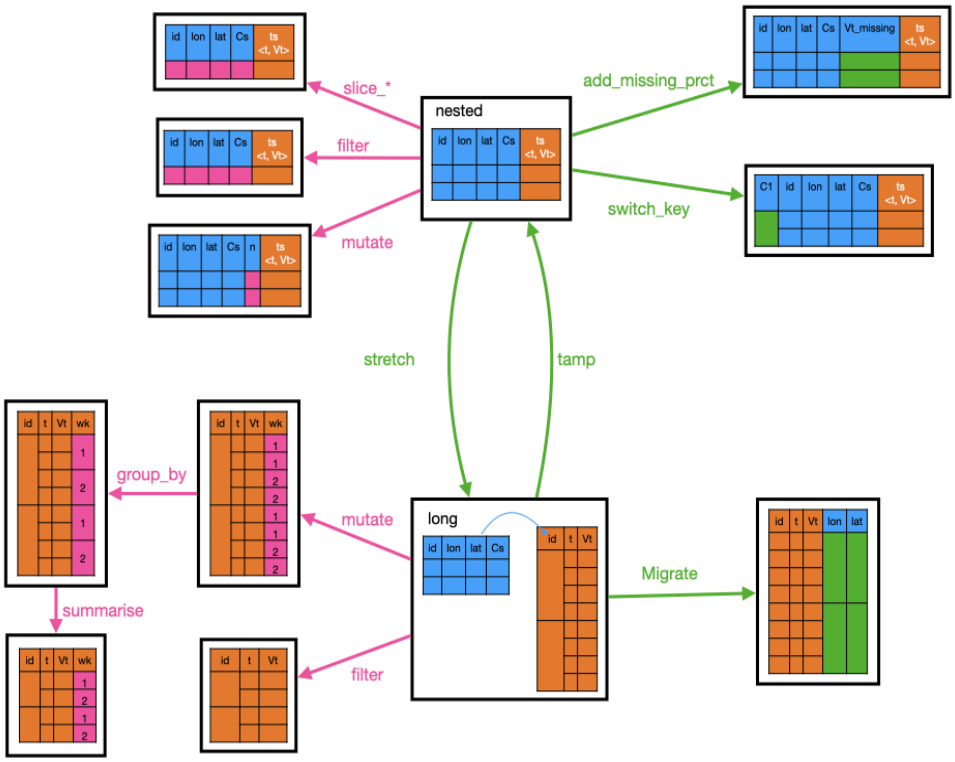

Figure 2: Cubble operations

4.1. Cubble operations

Basics

- `stretch`: nest to long form
- `tamp`: long to nest form
- `migrate`: move selected spatial variables to the long form.
- `add_dscrb_prct`: summary stats for missingness

dplyr compatibility:

- mutate, filter, summarise, select, arrange
- group and ungroup: group_by, ungroup
- slice family

Combine two cubbles

- match river and weather gauges data

- involve combining two cubbles
- join operations combine the two together by appending more rows but what we really want is to bind rows.
- bind rows also doesn't work since we want to bind only when there' s a matching????
- introduce bind_join

Hierarchical structure in cubble

- hierarchical is common.
- Given examples.
- Essence: switch between different levels
- introduce `switch_key`

5. Examples

Daily climate data (prcp, tmax, and tmin) from RNOAA - lots of stations across Australia
An exploratory data analysis questions: What's the climate profile look like in Australia

- General features: Any general trend/ fluctuation in prcp, tmax, and tmin?
- Local features: Any station stands out from the crowd?

5.1. Manipulation

Mutate and filter

In the first example, we want to only keep the stations that have `tmax` recorded in 2020. This requires first narrow down the records to those in 2020, determine if `tmax` is missing for each station, and then retain those stations that have `tmax` recorded. The year filtering is an operation on the time axis, so we start with the long form. Whether each station has `tmax` recorded is a result of each station, rather than of each time point, hence we need to switch to the nested form with `tamp()`. To calculate whether `tmax` is recorded, we mutate a column `tmax_missing` that takes `TRUE` if all the `tmax` in the nested list column `ts` are `NA` and `FALSE` otherwise. To get the stations that we want, we need another filter on `tmax_missing`.

```
R> aus_climate %>%
+   stretch() %>%
+   filter(year(date) == 2020) %>%
+   tamp()

# cubble:   id [85]: nested form
# bbox:     [115.97, -43.32, 153.13, -12.42]- check gap on long
# temporal: date [date], prcp [dbl], tmax [dbl], tmin [dbl]
   id            lat  long  elev name                   wmo_id ts
   <chr>       <dbl> <dbl> <dbl> <chr>                   <dbl> <list>
 1 ASN00009021 -31.9  116.  15.4 perth airport           94610 <tibble [366 x 4~
 2 ASN00010311 -31.9  117. 179   york                    94623 <tibble [366 x 4~
 3 ASN00010614 -32.9  117. 338   narrogin                94627 <tibble [366 x 4~
 4 ASN00014015 -12.4  131.  30.4 darwin airport          94120 <tibble [366 x 4~
 5 ASN00015131 -17.6  134. 220   elliott                 94236 <tibble [366 x 4~
 6 ASN00016065 -30.4  137.  76   andamooka               95660 <tibble [366 x 4~
 7 ASN00023034 -35.0  139.   2   adelaide airport        94672 <tibble [366 x 4~
 8 ASN00023373 -34.5  139. 275   nuriootpa viticultural  94681 <tibble [366 x 4~
 9 ASN00024024 -34.4  141.  30.1 loxton research centre  94682 <tibble [366 x 4~
10 ASN00024048 -34.2  141.  31.5 renmark aero           95687 <tibble [366 x 4~
# ... with 75 more rows
```

Join

Now we want to select the stations that have been registered with world meteorological organisation (WMO) and the dataset `station` has a column `wmo_id` that records this information. To do this task, we first need to join the `station` dataset with our climate dataset and then filter out those stations that don't have the WMO id. Since the join is by station rather than by time, we start with the nested form and write the exact same syntax of join and filter as with tidyverse.

```
R> # join wmo_id for each station
R> # to_join <- station %>% select(id, wmo_id)
R> # out <- climate_small %>%
R> #   left_join(to_join, by = c("station" = "id")) %>%
R> #   filter(!is.na(wmo_id))
R> # out
```

Sometimes, we would like to have station-wise and time-wise variables in the same form (i.e. when plotting glyph maps). This can also be seen as a joining task, on the long form, with the dataset to join being the metadata. `migrate()` is a verb introduced as the shortcut for `left_join()` with a cubble's metadata and below is the comparison of the two syntaxes.

```
R> aus_climate %>%
+   stretch() %>%
+   migrate(id, lat, long)

# cubble:  date, id [85]: long form
# bbox:    [115.97, -43.32, 153.13, -12.42]- check gap on long
# spatial: lat [dbl], long [dbl], elev [dbl], name [chr], wmo_id [dbl]
   id          date        prcp  tmax  tmin   lat  long
   <chr>       <date>     <dbl> <dbl> <dbl> <dbl> <dbl>
 1 ASN00009021 2020-01-01     0  31.9  15.3 -31.9  116.
 2 ASN00009021 2020-01-02     0  24.9  16.4 -31.9  116.
 3 ASN00009021 2020-01-03     6  23.2  13   -31.9  116.
 4 ASN00009021 2020-01-04     0  28.4  12.4 -31.9  116.
 5 ASN00009021 2020-01-05     0  35.3  11.6 -31.9  116.
 6 ASN00009021 2020-01-06     0  34.8  13.1 -31.9  116.
 7 ASN00009021 2020-01-07     0  32.8  15.1 -31.9  116.
 8 ASN00009021 2020-01-08     0  30.4  17.4 -31.9  116.
 9 ASN00009021 2020-01-09     0  28.7  17.3 -31.9  116.
10 ASN00009021 2020-01-10     0  32.6  15.8 -31.9  116.
# ... with 31,100 more rows
```

- data quality check: filter out stations have variables not properly recorded
- data summary:
  - daily -> monthly/ weekly,
  - summarise by mean for tmax/ tmin, sum for prcp
-

5.2. Graphics

Static + interactive -> tooltip to show additional information upon hovering

- Where are those stations on the map?
  - Mention mostly aero, airport, and lighthouse

Summary

Pebesma E (2021). *stars: Spatiotemporal Arrays, Raster and Vector Data Cubes*. R package version 0.5-2, URL https://CRAN.R-project.org/package=stars.

Pebesma EJ (2018). "Simple features for R: standardized support for spatial vector data." *R J.*, **10**(1), 439.

Wang E, Cook D, Hyndman RJ (2020). "A new tidy data structure to support exploration and modeling of temporal data." *Journal of Computational and Graphical Statistics*, **29**(3), 466–478. doi:10.1080/10618600.2019.1695624. URL https://doi.org/10.1080/10618600.2019.1695624.

Affiliation: