

# Package ‘flowcluster’

May 8, 2026

**Title** Cluster Origin-Destination Flow Data

**Version** 0.2.1

**Description** Provides functionality for clustering origin-destination (OD) pairs, representing desire lines (or flows). This includes creating distance matrices between OD pairs and passing distance matrices to a clustering algorithm. See the academic paper Tao and Thill (2016) <[doi:10.1111/gean.12100](https://doi.org/10.1111/gean.12100)> for more details on spatial clustering of flows. See the paper on delineating demand-responsive operating areas by Mahfouz et al. (2025) <[doi:10.1016/j.urbmob.2025.100135](https://doi.org/10.1016/j.urbmob.2025.100135)> for an example of how this package can be used to cluster flows for applied transportation research.

**License** MIT + file LICENSE

**Encoding** UTF-8

**RoxygenNote** 7.3.2

**URL** <https://hussein-mahfouz.github.io/flowcluster/>,  
<https://github.com/hussein-mahfouz/flowcluster>

**Depends** R (>= 4.1.0)

**Imports** sf, dbscan, dplyr, glue, lwgeom, purrr, rlang, tibble, units, tidy, tidyselect

**LazyData** true

**Suggests** testthat (>= 3.0.0), tmap

**Config/testthat/edition** 3

**NeedsCompilation** no

**Author** Hussein Mahfouz [aut, cre] (ORCID:  
<<https://orcid.org/0000-0003-1706-7802>>),  
Robin Lovelace [aut] (ORCID: <<https://orcid.org/0000-0001-5679-6536>>)

**Maintainer** Hussein Mahfouz <[husseinmahfouz93@gmail.com](mailto:husseinmahfouz93@gmail.com)>

**Repository** CRAN

**Date/Publication** 2025-08-21 09:40:11 UTC

## Contents

add_flow_length . . . . .	2
add_xyuv . . . . .	3
aggregate_clustered_flows . . . . .	3
cluster_flows_dbscan . . . . .	5
dbscan_sensitivity . . . . .	6
distance_matrix . . . . .	7
filter_by_length . . . . .	8
flows_leads . . . . .	9
flow_distance . . . . .	9
weight_vector . . . . .	10
<b>Index</b>	<b>12</b>

---

add_flow_length	<i>Add Length Column to Flow Data</i>
-----------------	---------------------------------------

---

### Description

Also checks that 'origin' and 'destination' columns are present.

### Usage

```
add_flow_length(x)
```

### Arguments

x sf object of flows (LINESTRING, projected CRS)

### Value

sf object with an additional length\_m column (od length in meters)

### Examples

```
flows <- sf::st_transform(flows_leads, 3857)
flows <- add_flow_length(flows)
```

---

add_xyuv	<i>Add Start/End Coordinates &amp; Flow IDs</i>
----------	---

---

**Description**

Add Start/End Coordinates & Flow IDs

**Usage**

```
add_xyuv(x)
```

**Arguments**

x                    sf object of flows

**Value**

tibble with x, y, u, v, flow\_ID columns

**Examples**

```
flows <- sf::st_transform(flows_leeds, 3857)
flows <- add_flow_length(flows)
flows <- add_xyuv(flows)
```

---

aggregate_clustered_flows	<i>Aggregate clustered OD flows into representative lines</i>
---------------------------	---

---

**Description**

This function aggregates flows within clusters and creates a single representative line for each cluster. The start and end coordinates are computed as weighted averages (weighted by flow counts or another variable), or simple means if no weights are provided. Each cluster is represented by one LINESTRING.

**Usage**

```
aggregate_clustered_flows(flows, weight = NULL, crs = sf::st_crs(flows))
```

**Arguments**

flows                An sf object containing OD flows with coordinates for origins (x, y) and destinations (u, v), a cluster column, and optionally a count or other weighting variable.

weight               (optional) Name of a column in flows to use for weighting. If NULL (default), unweighted means are used.

crs                    Coordinate reference system for the output (default: taken from flows).

**Value**

An sf object with one line per cluster, containing:

- `count_total`: total weight (if provided), otherwise number of flows
- `size`: the cluster size (from the input, not recomputed)
- `geometry`: a LINESTRING representing the aggregated OD flow

**Examples**

```
# ----- 1. Basic Usage: A quick, runnable example ---
# This demonstrates the function with minimal, fast data preparation.
flows <- flowcluster::flows_leeds

# Create the required input columns in a single, fast pipeline
flows_clustered <- flows |>
  add_xyuv() |>
  # Manually create 3 dummy clusters for demonstration
  dplyr::mutate(cluster = sample(1:3, size = nrow(flows), replace = TRUE)) |>
  # The function requires a 'size' column, so we add it
  dplyr::group_by(cluster) |>
  dplyr::add_tally(name = "size") |>
  dplyr::ungroup()

# Demonstrate the function
flows_agg_w <- aggregate_clustered_flows(flows_clustered, weight = "count")
print(flows_agg_w)

# ----- 2. Detailed Workflow (not run by default) ---
## Not run:
# This example shows the ideal end-to-end workflow, from raw data
# to clustering and finally aggregation. It is not run during checks
# because the clustering steps are too slow.

# a) Prepare the data by filtering and adding coordinates
flows_prep <- flowcluster::flows_leeds |>
  sf::st_transform(3857) |>
  add_flow_length() |>
  filter_by_length(length_min = 5000, length_max = 12000) |>
  add_xyuv()

# b) Calculate distances and cluster the flows
distances <- flow_distance(flows_prep, alpha = 1.5, beta = 0.5)
dmat <- distance_matrix(distances)
wvec <- weight_vector(dmat, flows_prep, weight_col = "count")
flows_clustered_real <- cluster_flows_dbscan(dmat, wvec, flows_prep, eps = 8, minPts = 70)

# c) Filter clusters and add a 'size' column
flows_clustered_real <- flows_clustered_real |>
  dplyr::filter(cluster != 0) |> # Filter out noise points
  dplyr::group_by(cluster) |>
  dplyr::mutate(size = dplyr::n()) |>
  dplyr::ungroup()
```

```

# d) Now, use the function on the clustered data
flows_agg_real <- aggregate_clustered_flows(flows_clustered_real, weight = "count")
print(flows_agg_real)

# e) Visualize the results
if (requireNamespace("tmap", quietly = TRUE)) {
  library(tmap)
  # This plot uses modern tmap v4 syntax.
  tm_shape(flows_clustered_real, facet = "cluster") +
    tm_lines(col = "grey50", alpha = 0.5) +
  tm_shape(flows_agg_real) +
    tm_lines(col = "red", lwd = 2) +
  tm_layout(title = "Original Flows (Grey) and Aggregated Flows (Red)")
}

## End(Not run)

```

---

cluster\_flows\_dbscan *Cluster Flows using DBSCAN*

---

### Description

See [dbscan](#) for details on the DBSCAN algorithm.

### Usage

```
cluster_flows_dbscan(dist_mat, w_vec, x, eps, minPts)
```

### Arguments

dist_mat	distance matrix
w_vec	weight vector
x	flows tibble with flow_ID
eps	DBSCAN epsilon parameter
minPts	DBSCAN minPts parameter

### Value

flows tibble with an additional cluster column

### Examples

```

flows <- sf::st_transform(flows_leeds, 3857)
flows <- head(flows, 100) # for testing
# Add flow lengths and coordinates
flows <- add_flow_length(flows)
# filter by length

```

```

flows <- filter_by_length(flows, length_min = 5000, length_max = 12000)
flows <- add_xyuv(flows)
# Calculate distances
distances <- flow_distance(flows, alpha = 1.5, beta = 0.5)
dmat <- distance_matrix(distances)
wvec <- weight_vector(dmat, flows, weight_col = "count")
clustered <- cluster_flows_dbscan(dmat, wvec, flows, eps = 8, minPts = 70)

```

---

dbscan\_sensitivity      *Sensitivity analysis of DBSCAN parameters for flow clustering.*

---

### Description

The function allows you to test different combinations of epsilon and minPts parameters for clustering flows using DBSCAN. It can be used to determine what parameter values make sense for your data

### Usage

```

dbscan_sensitivity(
  dist_mat,
  flows,
  options_epsilon,
  options_minpts,
  w_vec = NULL
)

```

### Arguments

dist_mat	a precalculated distance matrix between desire lines (output of distance_matrix())
flows	the original flows tibble (must contain flow_ID and 'count' column)
options_epsilon	a vector of options for the epsilon parameter
options_minpts	a vector of options for the minPts parameter
w_vec	Optional precomputed weight vector (otherwise computed internally from 'count' column)

### Value

a tibble with columns: id (to identify eps and minpts), cluster, size (number of desire lines in cluster), count\_sum (total count per cluster)

**Examples**

```

flows <- sf::st_transform(flows_leeds, 3857)
flows <- head(flows, 1000) # for testing
# Add flow lengths and coordinates
flows <- add_flow_length(flows)
# filter by length
flows <- filter_by_length(flows, length_min = 5000, length_max = 12000)
# Add x, y, u, v coordinates to flows
flows <- add_xyuv(flows)
# Calculate distance matrix
distances <- flow_distance(flows, alpha = 1.5, beta = 0.5)
dmat <- distance_matrix(distances)
# Generate weight vector
w_vec <- weight_vector(dmat, flows, weight_col = "count")

# Define the parameters for sensitivity analysis
options_epsilon <- seq(1, 10, by = 2)
options_minpts <- seq(10, 100, by = 10)
## Run the sensitivity analysis
results <- dbscan_sensitivity(
  dist_mat = dmat,
  flows = flows,
  options_epsilon = options_epsilon,
  options_minpts = options_minpts,
  w_vec = w_vec
)

```

---

distance\_matrix

*Convert Long-Format Distance Tibble to Matrix*


---

**Description**

Convert Long-Format Distance Tibble to Matrix

**Usage**

```
distance_matrix(distances, distance_col = "fds")
```

**Arguments**

distances      tibble with columns flow\_ID\_a, flow\_ID\_b, and distance  
distance\_col    column name for distance (default "fds")

**Value**

distance matrix (tibble with rownames). The matrix has flow\_ID\_a as rownames and flow\_ID\_b as column names. This function converts the output of flow\_distance() into a format suitable for the [dbscan](#) clustering algorithm.

## Examples

```
flows <- sf::st_transform(flows_leeds, 3857)
flows <- head(flows, 100) # for testing
# Add flow lengths and coordinates
flows <- add_flow_length(flows)
flows <- add_xyuv(flows)
# Calculate distances
distances <- flow_distance(flows, alpha = 1.5, beta = 0.5)
dmat <- distance_matrix(distances)
```

---

filter_by_length	<i>Filter Flows by Length</i>
------------------	-------------------------------

---

## Description

Filter Flows by Length

## Usage

```
filter_by_length(x, length_min = 0, length_max = Inf)
```

## Arguments

x	sf object with length_m
length_min	minimum length (default 0)
length_max	maximum length (default Inf)

## Value

filtered sf object. Flows with length\_m outside the specified range are removed.

## Examples

```
flows <- sf::st_transform(flows_leeds, 3857)
flows <- add_flow_length(flows)
flows <- filter_by_length(flows, length_min = 5000, length_max = 12000)
```

---

flows_leeds	<i>Example flow data for Leeds. It is from the 2021 census, and it contains all Origin - Destination flows at the MSOA level. For more info on census flow data, see the <a href="https://www.ons.gov.uk/census/aboutcensus/censusproducts/origindestinationflowdataONS">R hrefhttps://www.ons.gov.uk/census/aboutcensus/censusproducts/origindestinationflowdataONS</a> documentation See data-raw/flows_leeds.R for how this data was created.</i>
-------------	--

---

### Description

Example flow data for Leeds. It is from the 2021 census, and it contains all Origin - Destination flows at the MSOA level. For more info on census flow data, see the [ONS documentation](#) See data-raw/flows\_leeds.R for how this data was created.

### Usage

```
flows_leeds
```

### Format

An object of class `sf` with `LINESTRING` geometry. It has the following columns:

**origin** MSOA code of origin zone

**destination** MSOA code of destination zone

**count** number of people moving from origin to destination

**geometry** desire line between origin and destination

### Source

[https://www.nomisweb.co.uk/sources/census\\_2021\\_od](https://www.nomisweb.co.uk/sources/census_2021_od)

---

flow_distance	<i>Calculate Flow Distance and Dissimilarity</i>
---------------	--

---

### Description

This function calculates flow distance and dissimilarity measures between all pairs of flows based on the method described in @tao2016spatial.

### Usage

```
flow_distance(x, alpha = 1, beta = 1)
```

**Arguments**

x	tibble with flow_ID, x, y, u, v, length_m
alpha	numeric, origin weight
beta	numeric, destination weight

**Value**

tibble of all OD pairs with fd, fds columns

**References**

Tao, R., Thill, J.-C., 2016. Spatial cluster detection in spatial flow data. *Geographical Analysis* 48, 355–372. <https://doi.org/10.1111/gean.12100>

**Examples**

```
flows <- sf::st_transform(flows_leeds, 3857)
flows <- head(flows, 100) # for testing
# Add flow lengths and coordinates
flows <- add_flow_length(flows)
flows <- add_xyuv(flows)
# Calculate distances
distances <- flow_distance(flows, alpha = 1.5, beta = 0.5)
```

---

weight\_vector

*Generate Weight Vector from Flows*


---

**Description**

Generate Weight Vector from Flows

**Usage**

```
weight_vector(dist_mat, x, weight_col = "count")
```

**Arguments**

dist_mat	distance matrix
x	flows tibble with flow_ID and weight_col
weight_col	column to use as weights (default = "count")

**Value**

numeric weight vector. Each element corresponds to a flow in the distance matrix, and is used as a weight in the DBSCAN clustering algorithm.

**Examples**

```
flows <- sf::st_transform(flows_leeds, 3857)
flows <- head(flows, 100) # for testing
# Add flow lengths and coordinates
flows <- add_flow_length(flows)
flows <- add_xyuv(flows)
# Calculate distances
distances <- flow_distance(flows, alpha = 1.5, beta = 0.5)
dmat <- distance_matrix(distances)
wvec <- weight_vector(dmat, flows, weight_col = "count")
```

# Index

## \* datasets

- flows\_leeds, 9
  
- add\_flow\_length, 2
- add\_xyuv, 3
- aggregate\_clustered\_flows, 3
  
- cluster\_flows\_dbscan, 5
  
- dbscan, 5, 7
- dbscan\_sensitivity, 6
- distance\_matrix, 7
  
- filter\_by\_length, 8
- flow\_distance, 9
- flows\_leeds, 9
  
- sf, 9
  
- weight\_vector, 10