P4_code

Load libraries

```
# Set Java 21 rather than Java 24
Sys.setenv(JAVA_HOME = "/Library/Java/JavaVirtualMachines/jdk-21.jdk/Contents/Home")
library(rJava)
options(java.parameters = '-Xmx4G')

library(tidyverse)
library(here)
library(knitr)
library(tigris)
library(stringr)
library(maptiles)
library(tidyterra)
library(r5r)
library(sf)
library(leaflet)

here("P4_trip_distribution", "grvty_balancing.R") |> source()
```

Select a study area

There are 939 core-based statistical areas (CBSAs) in the U.S. → 393 are metropolitan areas (MSAs), and 542 are micropolitan areas (μSAs)

```
all_cbsas <- core_based_statistical_areas(progress_bar = FALSE,
                                          year = 2024) |>
  select(NAMELSAD) |>
  mutate(type = ifelse(!is.na(str_match(NAMELSAD, "Metro")), "Metro", "Micro")) |>
  mutate(type = as.character(type))

table(all_cbsas$type) |>
  kable()
```

Var1	Freq
Metro	393
Micro	542

The sweet spot is to select a large micro area or a small metro area.

```
salem <- all_cbsas |>
  filter(NAMELSAD == "Salem, OH Micro Area") |>
  st_transform("WGS84")

base_map <- get_tiles(salem,
                      provider = "CartoDB.Positron",
                      zoom = 9,
                      crop = TRUE)

ggplot(salem) +
  geom_spatraster_rgb(data = base_map) +
  geom_sf(fill = NA,
          color = "orange") +
  theme_void()
```

Load job data

The LEHD Origin-Destination Employment Statistics (LODES) dataset provides the number of workers who live and work between any pair of census blocks in a given state in a given year.

Counties in the Salem, OH Micro Area

- Columbiana County, OH (FIPS code: 39029)

Since the Youngstown-Warren, OH Metro Area spans both OH and PA, we'll need to download and process the LODES data for both states, then combine the results and filter for the counties of interest.

```
# Set year and FIPS codes
year <- "2021"
salem_counties_5_digit <- c("39029")

# Function to load and filter data for a given state
load_state_data <- function(state_abbrev, counties_5_digit, year) {
  url <- paste0("https://lehd.ces.census.gov/data/lodes/LODES8/",
                state_abbrev,
                "/od/",
                state_abbrev,
                "_od_main_JT00_",
                year,
                ".csv.gz")

  read_csv(url, col_types = cols()) |>
    mutate(w_county = substr(w_geocode, 1, 5),
           h_county = substr(h_geocode, 1, 5)) |>
    filter(h_county %in% counties_5_digit & w_county %in% counties_5_digit) |>
    mutate(w_geocode = as.character(w_geocode),
           h_geocode = as.character(h_geocode))
}

# Load the OH LODES data
oh_data <- load_state_data("oh", salem_counties_5_digit, year)

head(oh_data)
```

```
## # A tibble: 6 × 15
##   w_geocode      h_geocode  S000  SA01  SA02  SA03  SE01  SE02  SE03  SI01  SI02
##   <chr>          <chr>     <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 3902995010010… 39029950…     1     1     0     0     0     1     0     0     0
## 2 3902995010010… 39029950…     1     0     0     1     0     0     1     0     0
## 3 3902995010010… 39029950…     1     0     1     0     1     0     0     0     0
## 4 3902995010010… 39029950…     1     0     1     0     0     1     0     0     0
## 5 3902995010010… 39029950…     1     0     0     1     0     0     1     0     0
## 6 3902995010010… 39029950…     1     0     1     0     0     0     1     0     0
## # ℹ 4 more variables: SI03 <dbl>, createdate <dbl>, w_county <chr>,
## #   h_county <chr>
```

Aggregate data to zone totals

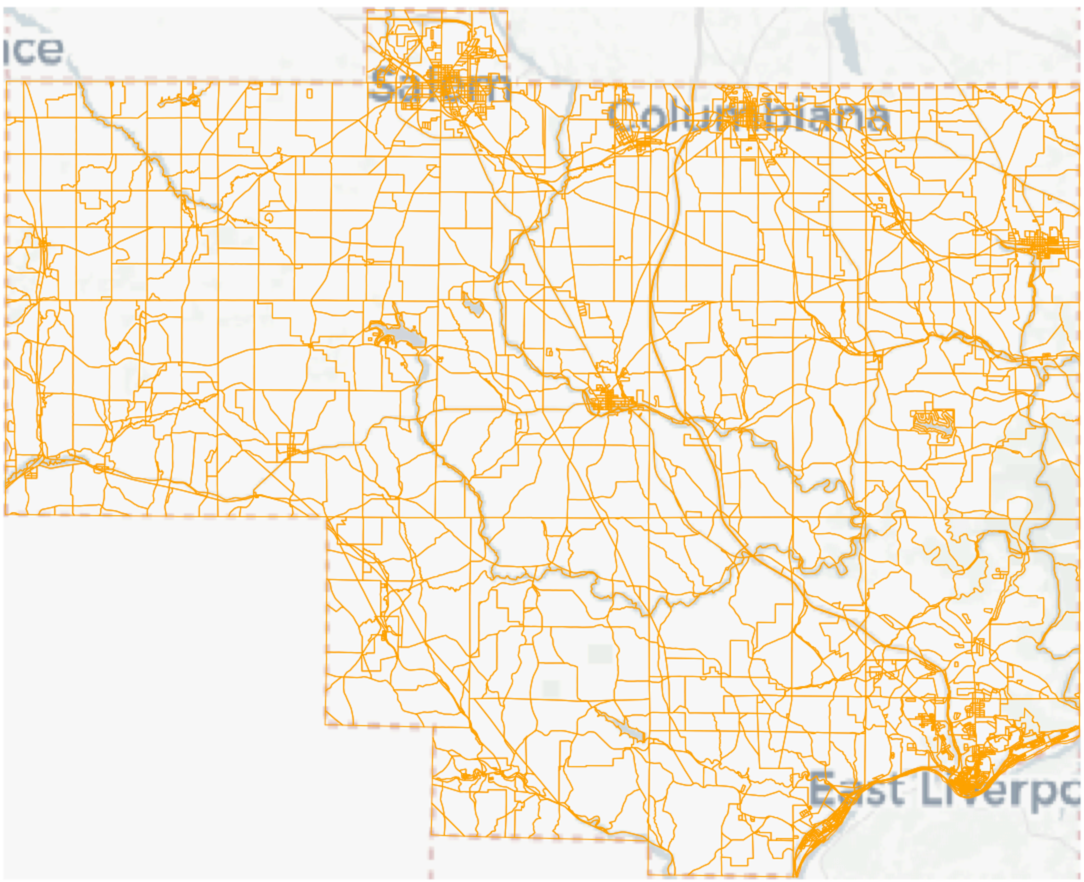
There are three industry categories: goods, trade, and services. We want to create a trip generation table that shows the number of workers produced (by living in a zone) and attracted to (by working in a zone) for each industry category.

```
# Calculate the total number of trip productions
total_prod <- oh_data |>
  group_by(h_geocode) |> # Group by home census block
  summarise(goods_p = sum(SI01), # Goods
            trade_p = sum(SI02), # Trade
            serve_p = sum(SI03), # Service
            total_p = sum(S000)) |> # Total
  rename(geocode = h_geocode)

# Calculate the total number of trip attractions
total_attr <- oh_data |>
  group_by(w_geocode) |> # Group by home census block
  summarize(goods_a = sum(SI01), # Goods
            trade_a = sum(SI02), # Trade
            serve_a = sum(SI03), # Service
            total_a = sum(S000)) |> # Total
  rename(geocode = w_geocode)

# Calculate trip generations (trip productions + trip attractions)
trip_gen <- full_join(total_prod,
                      total_attr) |>
  replace_na(list(goods_p = 0,
                  goods_a = 0,
                  trade_p = 0,
                  trade_a = 0,
                  serve_p = 0,
                  serve_a = 0,
                  total_p = 0,
                  total_a = 0))

head(trip_gen)
```

```
## # A tibble: 6 × 9
##   geocode         goods_p trade_p serve_p total_p goods_a trade_a serve_a total_a
##   <chr>             <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
## 1 3902995010010…       1       1       4       6       0       0       0       0
## 2 3902995010010…       0       0       1       1       0       0       0       0
## 3 3902995010010…       3       0       2       5       0       0       0       0
## 4 3902995010010…       0       0       1       1       0       0       0       0
## 5 3902995010010…       0       0       2       2       0       0       0       0
## 6 3902995010010…       0       0       1       1       0       0       0       0
```

Load spatial data

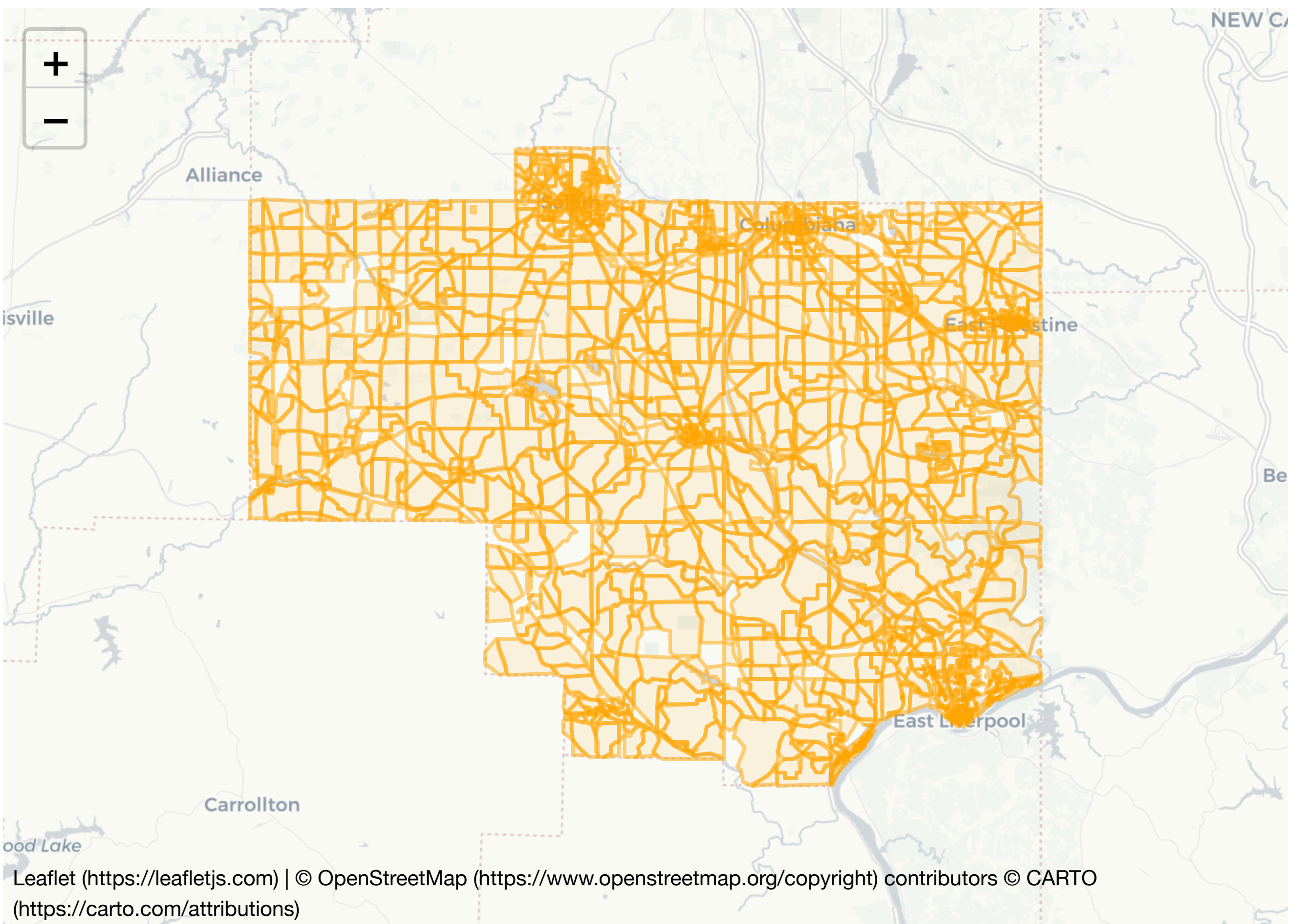
```
# Download the census block boundaries for the counties of interest

# Define county FIPS (3-digit codes)
oh_counties_3_digit <- c("029")

# Load block shapefiles
msa_blocks <- blocks(state = "OH",
                    county = oh_counties_3_digit,
                    progress_bar = FALSE)

# Plotting the map
ggplot(msa_blocks) +
  geom_spatraster_rgb(data = base_map) +
  geom_sf(fill = NA, color = "orange") +
  theme_void()
```



Creating `trip_gen_locs`

We will right join the trip generation table to the census blocks of interest (only include the ones in the trip generation table)

```
trip_gen_locs <- msa_blocks |>
  rename(geocode = GEOID20) |>
  right_join(trip_gen) |>
  select(geocode,
         goods_p,
         trade_p,
         serve_p,
         total_p,
         goods_a,
         trade_a,
         serve_a,
         total_a) |>
  st_transform("WGS84")

leaflet(trip_gen_locs) |>
  addProviderTiles(provider = "CartoDB.Positron") |>
  addPolygons(weight = 2,
              color = "orange",
              fillColor = "orange",
              fillOpacity = 0.1,
              highlightOptions = highlightOptions(weight = 3,
                                                  fillOpacity = 0.5),
              label = trip_gen_locs$geocode)
```



```
nrow(trip_gen_locs) # This number of blocks is indicative of the computational expense a
nd how long it will take to skim the network
```

```
## [1] 2661
```

Load the network

We will use BBBike (a cycle route planner) to extract/download the OpenStreetMap network

Details

- Area: ‘Salem, Columbiana County, Ohio, 44460, United States’ covers 80 square km
- Format: Protocolbuffer (PBF)

I downloaded the network .pbf file from BBBike, saved it in a folder called `network` → in the code below, we will use r5r to save the .pbf file as two shape files in an empty folder I created called `data`

Note: I had to manually delete the cache from my laptop in order to regenerate the shape files after redoing the BBBike step

It seems like the goal is to extract the smallest street network that overlaps with the trip generation zones because if the street network area is too large, I start getting some errors when trying to save the .pbf file as shape files

```
# Setting to eval = FALSE so that we don't repeat this process again
salem_core <- setup_r5(
  data_path = here("P4_trip_distribution", "network"),
  overwrite = TRUE,
  verbose = TRUE
)

street_vis <- street_network_to_sf(salem_core)

street_lines <- street_vis$edges
street_pts <- street_vis$vertices

# Write/save as two shape files
st_write(street_lines,
         here("P4_trip_distribution",
              "data",
              "street-lines.shp"))

st_write(street_pts,
         here("P4_trip_distribution",
              "data",
              "street-pts.shp"))

stop_r5()
```

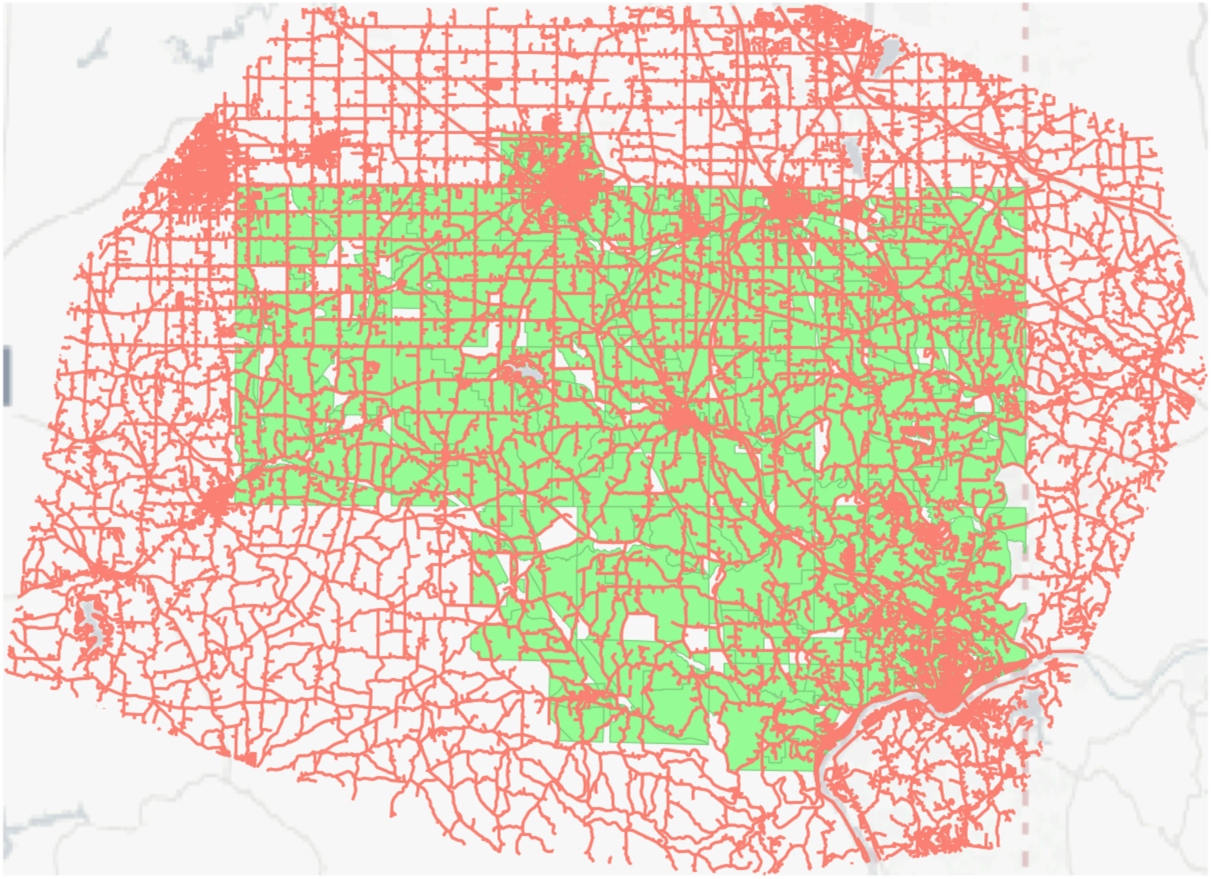
```
# Reading the two network shape files now that they're created
street_lines <- here("P4_trip_distribution",
                     "data",
                     "street-lines.shp") |>
  st_read()

street_pts <- here("P4_trip_distribution",
                     "data",
                     "street-pts.shp") |>
  st_read()
```

Overlaying the street network and trip generation zones

We want to make sure that the street network (red) covers at least the full trip generation zones (green)

If this is not the case, we need to shift the target area and re-download the BBBike OpenStreetMap network (I had to do this)

```
base_map <- get_tiles(street_lines,
                      provider = "CartoDB.Positron",
                      zoom = 8,
                      crop = TRUE)

ggplot() +
  geom_spatraster_rgb(data = base_map) +
  geom_sf(data = trip_gen_locs,
          color = "palegreen3",
          fill = "palegreen") +
  geom_sf(data = street_lines,
          color =  "salmon") +
  theme_void()
```



Skim the network

First, we will represent the block locations as zone centroid points, then we will calculate the travel time by car from every census block to every other census block

We use the `st_point_on_surface()` function to ensure that all points are on the inside of the polygon (rather than in a lake for instance) to prevent any points from being located outside the zone

```
# Setting eval = FALSE to prevent skimming the network every single time
trip_gen_loc_ids <- trip_gen_locs |>
  st_point_on_surface() |>
  st_nearest_feature(street_pts)

trip_gen_pts <- street_pts[trip_gen_loc_ids,] |>
  mutate(id = trip_gen_locs$geocode) |>
  select(id)

# Calculating travel time matrix (skimming the network)
salem_core <- here("P4_trip_distribution", "network") |>
  setup_r5()

skim <- travel_time_matrix(salem_core,
                           origins = trip_gen_pts,
                           destinations = trip_gen_pts,
                           mode = "CAR",
                           max_trip_duration = 180)

stop_r5()

# Write the csv file
write_csv(skim, file = here("P4_trip_distribution",
                            "data",
                            "salem-skim.csv"))
```

```
# Read the data
skim <- read_csv(here("P4_trip_distribution",
                      "data",
                      "salem-skim.csv"),
                 col_types = "ccn")

head(skim) |>
  kable()
```

from_id	to_id	travel_time_p50
390299509003046	390299509003046	0
390299509003046	390299518004000	65
390299509003046	390299511002021	36
390299509003046	390299521002059	58
390299509003046	390299504001024	39
390299509003046	390299514011052	67

```
paste0("Number of blocks in the study area: ", nrow(trip_gen_locs))
```

```
## [1] "Number of blocks in the study area: 2661"
```

```
paste0("Total number of pairs in the skim: ", nrow(skim))
```

```
## [1] "Total number of pairs in the skim: 7080921"
```

```
paste0("Does the number of blocks in the study area squared equal the total number of pa
irs in the skim? ", nrow(trip_gen_locs)^2 == nrow(skim))
```

```
## [1] "Does the number of blocks in the study area squared equal the total number of pa
irs in the skim? TRUE"
```

Yes, since the number of blocks in the study area squared equals the total number of pairs in the skim, we can move forward. This means that the r5r package was able to find a path through the network between every possible pair of census blocks.

Apply a gravity model

We will apply a gravity model to distribute trips from each production zone among all possible attraction zones.

Gravity model

- The attractiveness of a zone to production increases with the number of attractions in it and decreases with the cost of travel time
- Travel time vs. attractiveness can be modeled with a decay function
- $f(c_{ij}) = exp(-\beta c_{ij})$
    - $f(c_{ij})$ is the friction between zones i and j
    - c_{ij} is the travel time between zones i and j
    - β is a parameter that quantifies how sensitive travelers are to travel time
        - Positive β implies greater sensitivity to travel time → people are less attracted to longer travel times
        - Negative β implies lower sensitivity to travel time → people are more attracted to longer travel times (counterintuitive and problematic)

```
flow_tt <- oh_data |>
  rename(from_id = h_geocode,
         to_id = w_geocode) |>
  right_join(skim) |>
  rename(flow_total = S000,
         flow_goods = SI01,
         flow_trade = SI02,
         flow_serve = SI03) |>
  replace_na(list(flow_total = 0,
                  flow_goods = 0,
                  flow_trade = 0,
                  flow_serve = 0))

avg_tts <- tibble(`Worker sector` = c("Goods", "Trade", "Service", "Total"),
                  `Average travel time (observed)` = c(
                    sum(flow_tt$flow_goods * flow_tt$travel_time_p50) /
                      sum(flow_tt$flow_goods),
                    sum(flow_tt$flow_trade * flow_tt$travel_time_p50) /
                      sum(flow_tt$flow_trade),
                    sum(flow_tt$flow_serve * flow_tt$travel_time_p50) /
                      sum(flow_tt$flow_serve),
                    sum(flow_tt$flow_total * flow_tt$travel_time_p50) /
                      sum(flow_tt$flow_total)))

kable(avg_tts, digits = 1)
```

Worker sector	Average travel time (observed)
Goods	17.1
Trade	16.9
Service	16.4
Total	16.6

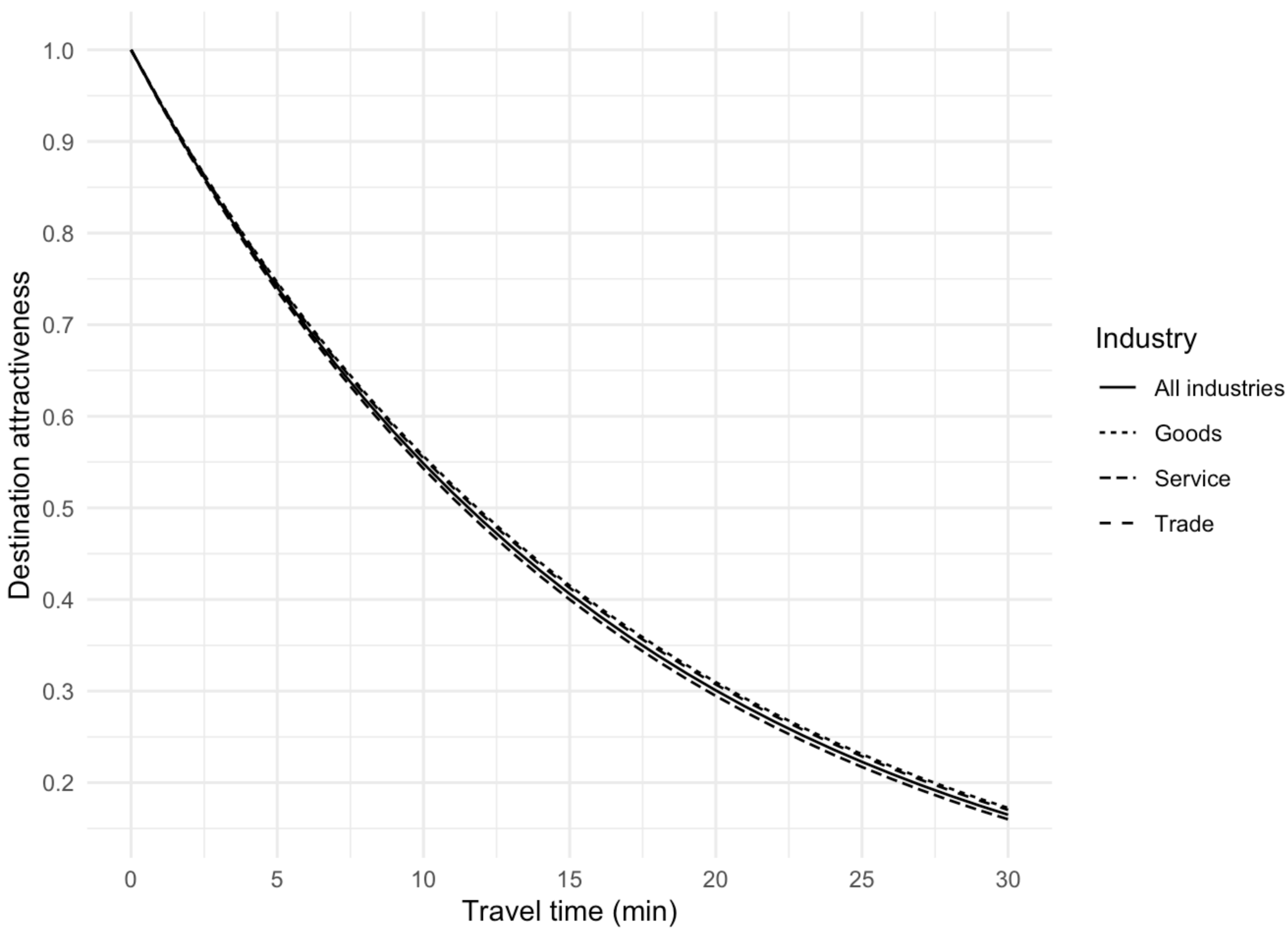
On average, workers in goods-producing industries live 17.1 minutes from work, workers in trade and retail live 16.9 minutes from work, and workers in service live 16.4 minutes from work. On average, among all workers from all industries, the average travel time from home to work is 16.6 minutes.

```
betas <- 1/avg_tts$`Average travel time (observed)`
names(betas) <- c("Goods", "Trade", "Service", "Total")

initial_betas <- tibble(`Worker sector` = names(betas),
                  `Initial β value` = betas)

kable(initial_betas, digits = 3)
```

Worker sector	Initial β value
Goods	0.059

Worker sector	Initial β value
Trade	0.059
Service	0.061
Total	0.060

```
friction <- tibble(`Travel time (min)` = seq(0, 30, by=1)) |>
  mutate(Goods = exp(-1 * betas["Goods"] * `Travel time (min)`),
         Trade = exp(-1 * betas["Trade"] * `Travel time (min)`),
         Service = exp(-1 * betas["Service"] * `Travel time (min)`),
         `All industries` = exp(-1 * betas["Total"] * `Travel time (min)`)) |>
  pivot_longer(cols = -`Travel time (min)`,
               names_to = "Industry") |>
  rename(`Destination attractiveness` = value)

ggplot(friction) +
  geom_line(aes(x = `Travel time (min)`,
                y = `Destination attractiveness`,
                linetype = Industry)) +
  scale_x_continuous(breaks = seq(0, 30, by=5)) +
  scale_y_continuous(breaks = seq(0, 1.1, by=0.1)) +
  theme_minimal()
```



Above is a plot of the sensitivity to travel time (beta values) for workers across the three industries and overall on average

We can apply these calculated beta values to calculate the friction factors between zones. Friction factors quantify the effect of travel time on trips between zones; higher friction factors indicate easier travel with lower travel time.

```
# Calculating friction factors
flow_tt <- flow_tt |>
  mutate(friction_goods = exp(-1 * betas["Goods"] * travel_time_p50),
         friction_trade = exp(-1 * betas["Trade"] * travel_time_p50),
         friction_serve = exp(-1 * betas["Service"] * travel_time_p50),
         friction_total = exp(-1 * betas["Total"] * travel_time_p50))
```

Estimating the initial trip matrix

The number of trips between any two zones can be calculated as $T_{ij} = A_i O_i B_j D_j f_{ij}$

- T_{ij} is the number of trips between zones i and j
- O_i is the number of origins/productions from zone i
- D_j is the number of destinations/attractions to zone j
- A_i and B_j are balancing factors to ensure that the total number of attractions and productions remain equal

```
# Setting eval = FALSE to prevent us from running this every time
flow_goods_est <- grvty_balancing(od_zones = trip_gen,
                                  friction = flow_tt,
                                  zone_id = "geocode",
                                  zone_o = "goods_p",
                                  zone_d = "goods_a",
                                  friction_o_id = "from_id",
                                  friction_d_id = "to_id",
                                  friction_factor = "friction_goods",
                                  tolerance = 0.001,
                                  max_iter = 100)

flow_trade_est <- grvty_balancing(od_zones = trip_gen,
                                  friction = flow_tt,
                                  zone_id = "geocode",
                                  zone_o = "trade_p",
                                  zone_d = "trade_a",
                                  friction_o_id = "from_id",
                                  friction_d_id = "to_id",
                                  friction_factor = "friction_trade",
                                  tolerance = 0.001,
                                  max_iter = 100)

flow_serve_est <- grvty_balancing(od_zones = trip_gen,
                                  friction = flow_tt,
                                  zone_id = "geocode",
                                  zone_o = "serve_p",
                                  zone_d = "serve_a",
                                  friction_o_id = "from_id",
                                  friction_d_id = "to_id",
                                  friction_factor = "friction_serve",
                                  tolerance = 0.001,
                                  max_iter = 100)

flow_total_est <- grvty_balancing(od_zones = trip_gen,
                                  friction = flow_tt,
                                  zone_id = "geocode",
                                  zone_o = "total_p",
                                  zone_d = "total_a",
                                  friction_o_id = "from_id",
                                  friction_d_id = "to_id",
                                  friction_factor = "friction_total",
                                  tolerance = 0.001,
                                  max_iter = 100)

write_csv(flow_goods_est$flows,
          file = here("P4_trip_distribution",
                      "data",
                      "init-goods-flow.csv"))

write_csv(flow_trade_est$flows,
          file = here("P4_trip_distribution",
```

```
                      "data",
                      "init-trade-flow.csv"))

write_csv(flow_serve_est$flows,
          file = here("P4_trip_distribution",
                      "data",
                      "init-serve-flow.csv"))

write_csv(flow_total_est$flows,
          file = here("P4_trip_distribution",
                      "data",
                      "init-total-flow.csv"))
```

```
# Reading the saved data
flow_goods <- here("P4_trip_distribution",
                   "data",
                   "init-goods-flow.csv") |>
  read_csv(col_types = "ccn") |>
  rename(from_id = o_id,
         to_id = d_id,
         goods_flow_est = flow)

flow_trade <- here("P4_trip_distribution",
                   "data",
                   "init-trade-flow.csv") |>
  read_csv(col_types = "ccn") |>
  rename(from_id = o_id,
         to_id = d_id,
         trade_flow_est = flow)

flow_serve <- here("P4_trip_distribution",
                   "data",
                   "init-serve-flow.csv") |>
  read_csv(col_types = "ccn") |>
  rename(from_id = o_id,
         to_id = d_id,
         serve_flow_est = flow)

flow_total <- here("P4_trip_distribution",
                   "data",
                   "init-total-flow.csv") |>
  read_csv(col_types = "ccn") |>
  rename(from_id = o_id,
         to_id = d_id,
         total_flow_est = flow)
```

```
flow_tt <- flow_tt |>
  left_join(flow_goods) |>
  left_join(flow_trade) |>
  left_join(flow_serve) |>
  left_join(flow_total)

# Comparing the average estimated travel time to the average observed travel time
avg_tts <- avg_tts |>
  mutate(`Average travel time (estimated)` = c(
    sum(flow_tt$goods_flow_est * flow_tt$travel_time_p50) /
      sum(flow_tt$goods_flow_est),
    sum(flow_tt$trade_flow_est * flow_tt$travel_time_p50) /
      sum(flow_tt$trade_flow_est),
    sum(flow_tt$serve_flow_est * flow_tt$travel_time_p50) /
      sum(flow_tt$serve_flow_est),
    sum(flow_tt$total_flow_est * flow_tt$travel_time_p50) /
      sum(flow_tt$total_flow_est)))

avg_tts |>
  kable(digits = 1)
```

Worker sector	Average travel time (observed)	Average travel time (estimated)
Goods	17.1	10.0
Trade	16.9	13.5
Service	16.4	12.8
Total	16.6	12.4

Evaluating the model accuracy via RMSE and visually

The two average travel times observed vs. estimated are very different, so we should calculate the RMSE

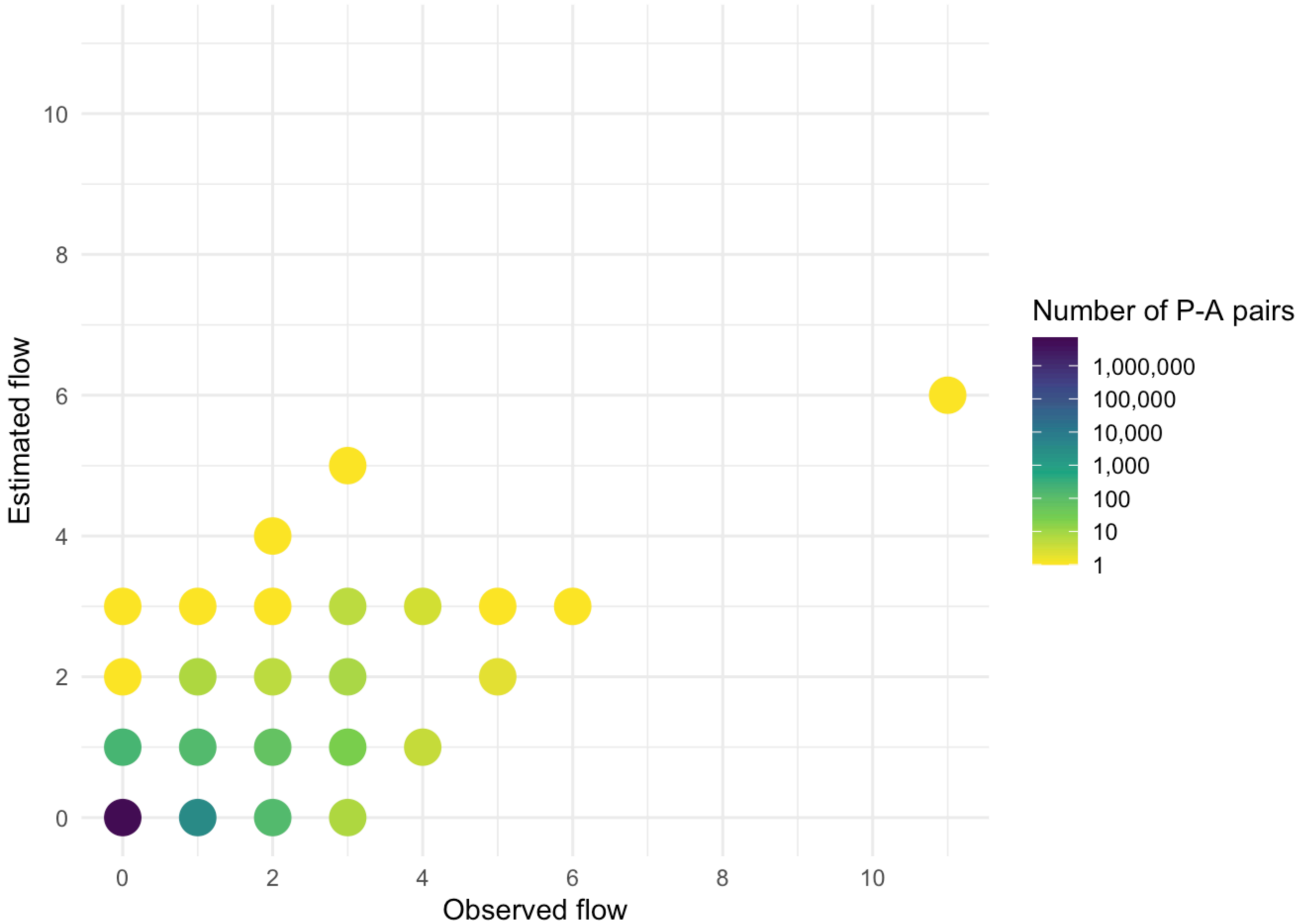
```
avg_tts <- avg_tts |>
  mutate(rmse = c((mean((flow_tt$flow_goods - flow_tt$goods_flow_est)^2))^0.5,
                  (mean((flow_tt$flow_trade - flow_tt$trade_flow_est)^2))^0.5,
                  (mean((flow_tt$flow_serve - flow_tt$serve_flow_est)^2))^0.5,
                  (mean((flow_tt$flow_total - flow_tt$total_flow_est)^2))^0.5))

kable(avg_tts, digits = 2)
```

Worker sector	Average travel time (observed)	Average travel time (estimated)	rmse
Goods	17.07	9.96	0.03
Trade	16.95	13.52	0.02
Service	16.37	12.79	0.04

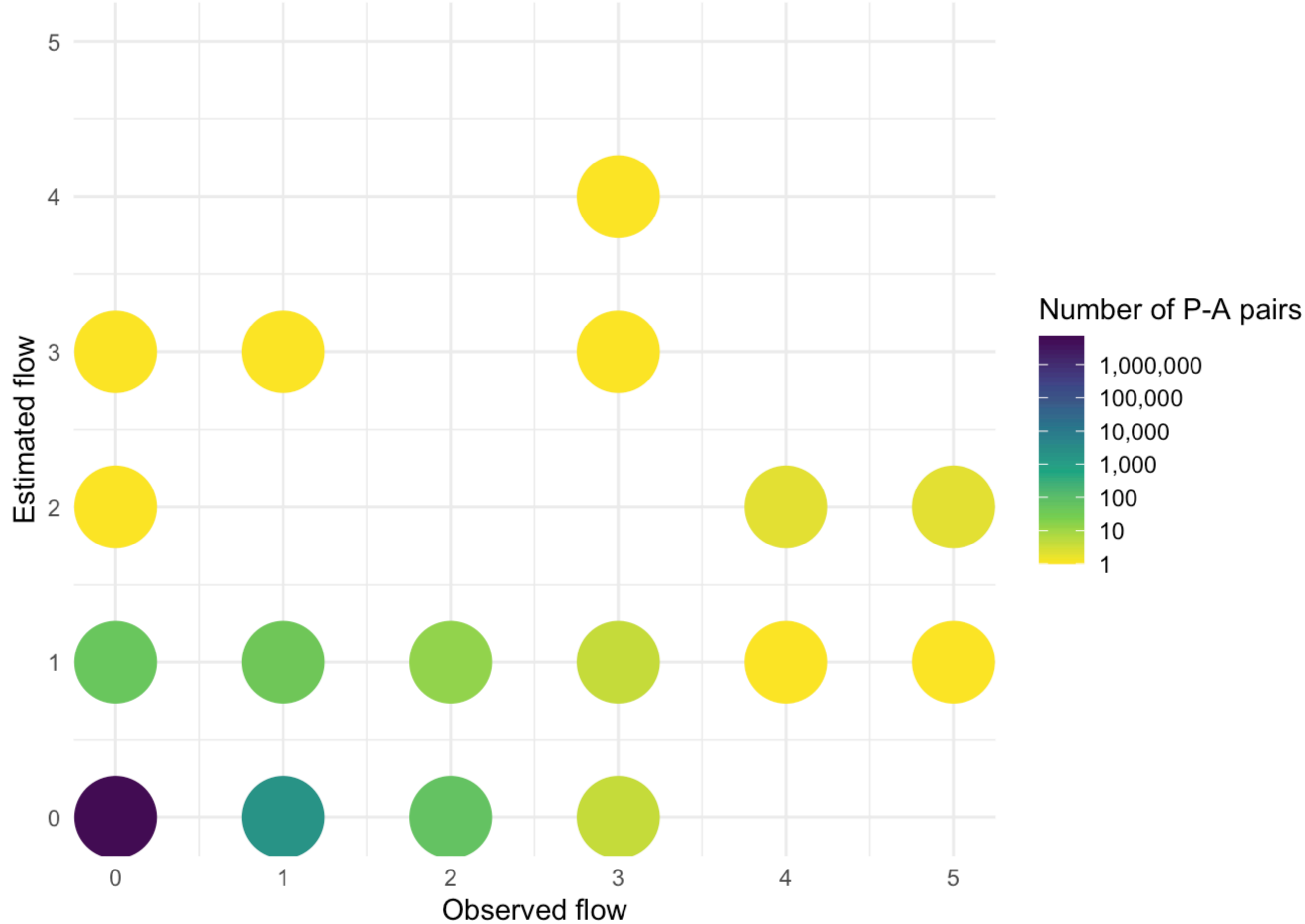
Worker sector	Average travel time (observed)	Average travel time (estimated)	rmse
Total	16.65	12.40	0.05

```
# Plotting function to visualize the estimated vs. observed number of production attract
ion pairs
plot_flows <- function(flow_df,
                       obs_col_name,
                       est_col_name) {

  summary <- flow_df |>
    rename(obs = all_of(obs_col_name),
           est = all_of(est_col_name)) |>
    group_by(obs, est) |>
    summarize(n = n())

  max_scale <- max(summary$obs, summary$est)
  my_interval <- ceiling(max_scale / 10)
  dot_size <- floor(70 / max_scale)

  max_n_exp = round(log10(max(summary$n)))

  ggplot(summary) +
    geom_point(aes(x = obs,
                   y = est,
                   color = n),
               size = dot_size) +
    scale_x_continuous(name = "Observed flow",
                       limits = c(0, max_scale),
                       breaks = seq(0, max_scale, by=my_interval)) +
    scale_y_continuous(name = "Estimated flow",
                       limits = c(0, max_scale),
                       breaks = seq(0, max_scale, by=my_interval)) +
    scale_color_viridis_c(transform = "log",
                          breaks = my_breaks <- c(10^seq(-1,
                                                         max_n_exp,
                                                         by=1)),
                          labels = formatC(my_breaks, format = "d",
                                           big.mark = ","),
                          direction = -1,
                          name = "Number of P-A pairs") +
    theme_minimal()


}
```

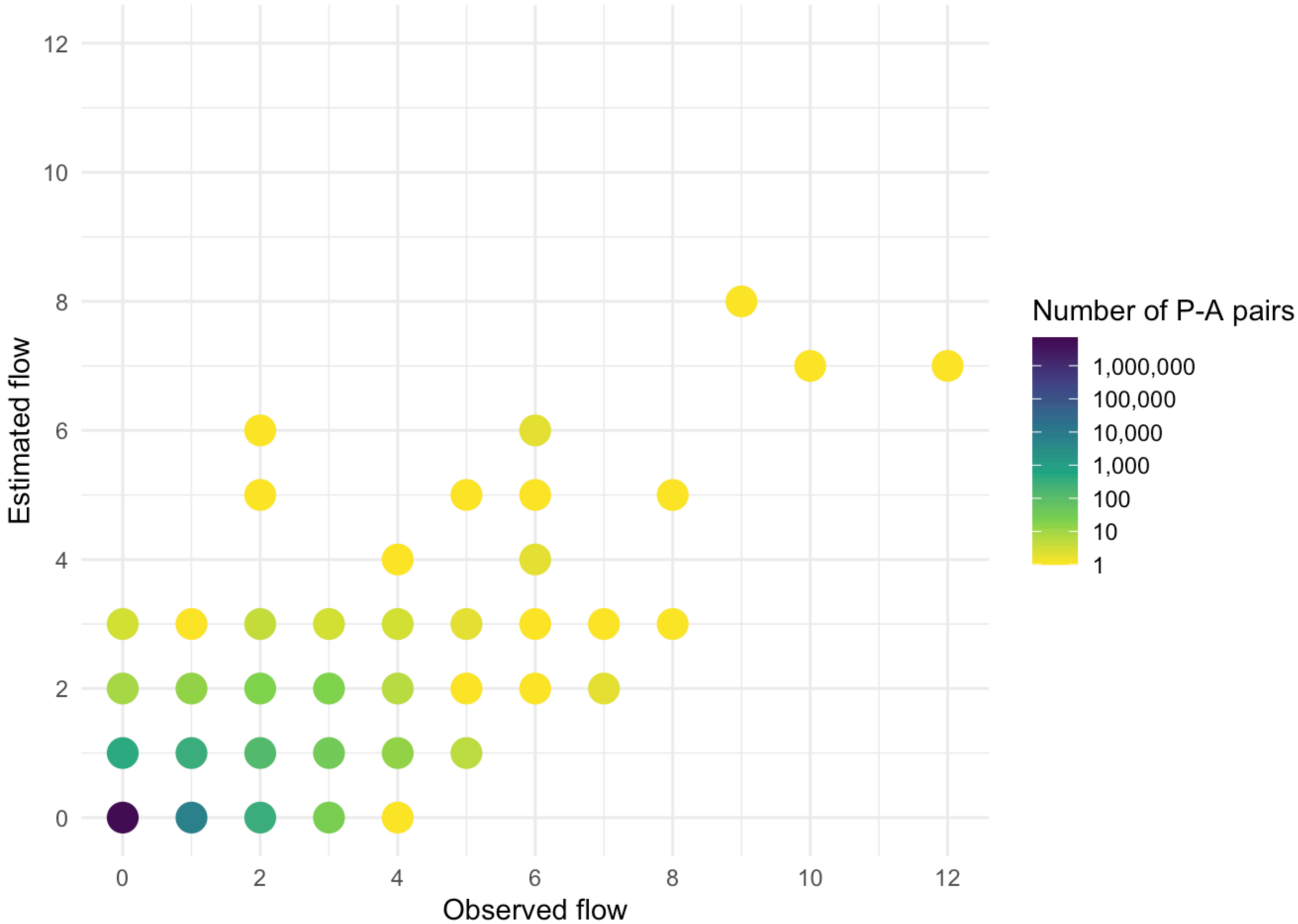
```
# Visualizing estimated vs. observed for the goods sector
plot_flows(flow_tt,
           obs_col_name = "flow_goods",
           est_col_name = "goods_flow_est")
```



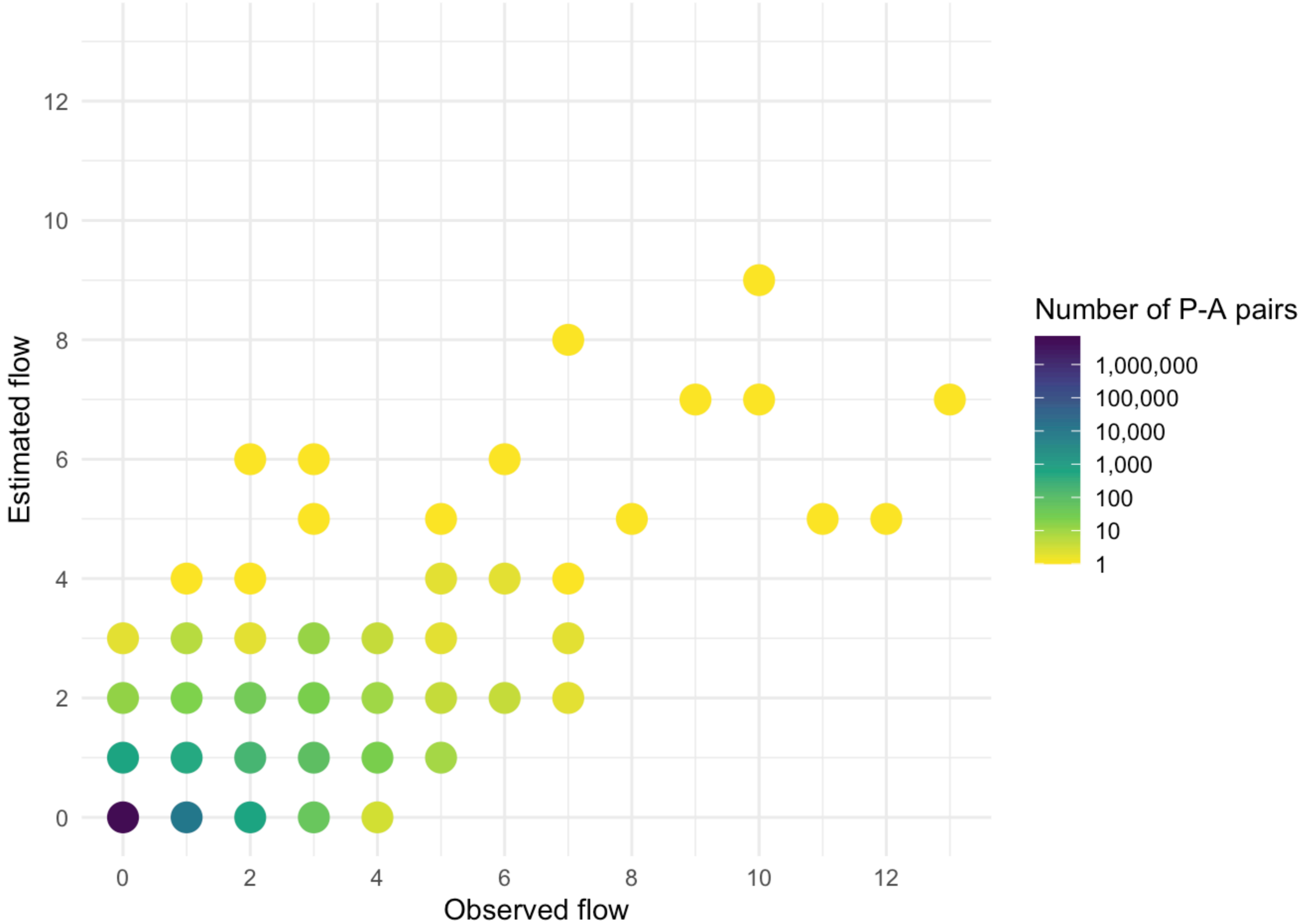

```
# Visualizing estimated vs. observed for the trade sector
plot_flows(flow_tt,
           obs_col_name = "flow_trade",
           est_col_name = "trade_flow_est")
```




```
# Visualizing estimated vs. observed for the service sector
plot_flows(flow_tt,
           obs_col_name = "flow_serve",
           est_col_name = "serve_flow_est")
```




```
# Visualizing estimated vs. observed for all sectors combined
plot_flows(flow_tt,
           obs_col_name = "flow_total",
           est_col_name = "total_flow_est")
```



Calibrate the gravity model

Our initial guess of $\beta = \frac{1}{c^*}$ was ok, but we can calibrate/improve the beta values further

```
# Setting eval = FALSE to prevent from running every time
# Drop old estimates
flow_tt <- flow_tt |>
  select(-goods_flow_est, -trade_flow_est, -serve_flow_est, -total_flow_est)

# ---- Calibrate goods sector ----
calibrated_flows_goods <- grvty_calibrate(
  obs_flow_tt = flow_tt,
  o_id_col = "from_id",
  d_id_col = "to_id",
  obs_flow_col = "flow_goods",
  tt_col = "travel_time_p50",
  tolerance_balancing = 0.0001,
  max_iter_balancing = 30,
  tolerance_calibration = 0.2,
  max_iter_calibration = 30
)

beta_goods <- calibrated_flows_goods$beta

goods_flow_est <- calibrated_flows_goods$flows |>
  rename(from_id = o_id, to_id = d_id, goods_flow_est = flow_est)

# ---- Calibrate trade sector ----
calibrated_flows_trade <- grvty_calibrate(
  obs_flow_tt = flow_tt,
  o_id_col = "from_id",
  d_id_col = "to_id",
  obs_flow_col = "flow_trade",
  tt_col = "travel_time_p50",
  tolerance_balancing = 0.0001,
  max_iter_balancing = 30,
  tolerance_calibration = 0.2,
  max_iter_calibration = 30
)

beta_trade <- calibrated_flows_trade$beta

trade_flow_est <- calibrated_flows_trade$flows |>
  rename(from_id = o_id, to_id = d_id, trade_flow_est = flow_est)

# ---- Calibrate service sector ----
calibrated_flows_serve <- grvty_calibrate(
  obs_flow_tt = flow_tt,
  o_id_col = "from_id",
  d_id_col = "to_id",
  obs_flow_col = "flow_serve",
  tt_col = "travel_time_p50",
  tolerance_balancing = 0.0001,
  max_iter_balancing = 30,
  tolerance_calibration = 0.2,
  max_iter_calibration = 30
```

```r
)

beta_serve <- calibrated_flows_serve$beta

serve_flow_est <- calibrated_flows_serve$flows |>
  rename(from_id = o_id, to_id = d_id, serve_flow_est = flow_est)

# ---- Calibrate total sector ----
calibrated_flows_total <- grvty_calibrate(
  obs_flow_tt = flow_tt,
  o_id_col = "from_id",
  d_id_col = "to_id",
  obs_flow_col = "flow_total",
  tt_col = "travel_time_p50",
  tolerance_balancing = 0.0001,
  max_iter_balancing = 30,
  tolerance_calibration = 0.2,
  max_iter_calibration = 30
)

beta_total <- calibrated_flows_total$beta

total_flow_est <- calibrated_flows_total$flows |>
  rename(from_id = o_id, to_id = d_id, total_flow_est = flow_est)

# ---- Join calibrated estimates ----
flow_tt <- flow_tt |>
  left_join(goods_flow_est, by = c("from_id", "to_id")) |>
  left_join(trade_flow_est, by = c("from_id", "to_id")) |>
  left_join(serve_flow_est, by = c("from_id", "to_id")) |>
  left_join(total_flow_est, by = c("from_id", "to_id"))

# ---- Store calibrated beta values ----
betas_table <- tibble(
  Industry = c("Goods", "Trade", "Service", "Total"),
  beta_initial = betas,
  beta_calibrated = c(beta_goods, beta_trade, beta_serve, beta_total)
)

# ---- Save calibrated flows and betas ----
write_csv(flow_tt, here("P4_trip_distribution", "data", "calib-flows.csv"))
write_csv(betas_table, here("P4_trip_distribution", "data", "calib-betas.csv"))
```

```
# Read the saved data
flow_tt <- here("P4_trip_distribution",
                "data",
                "calib-flows.csv") |>
  read_csv()

betas_table <- here("P4_trip_distribution",
                "data",
                "calib-betas.csv") |>
  read_csv()
```

Calculating average travel time again after model calibration

```
avg_tts <- avg_tts |>
  select(-rmse) |>
  mutate(`Average travel time (estimated)` = c(
    sum(flow_tt$goods_flow_est * flow_tt$travel_time_p50) /
      sum(flow_tt$goods_flow_est),
    sum(flow_tt$trade_flow_est * flow_tt$travel_time_p50) /
      sum(flow_tt$trade_flow_est),
    sum(flow_tt$serve_flow_est * flow_tt$travel_time_p50) /
      sum(flow_tt$serve_flow_est),
    sum(flow_tt$total_flow_est * flow_tt$travel_time_p50) /
      sum(flow_tt$total_flow_est)))

avg_tts |>
  kable(digits = 1)
```

Worker sector	Average travel time (observed)	Average travel time (estimated)
Goods	17.1	17.1
Trade	16.9	16.9
Service	16.4	16.4
Total	16.6	16.7

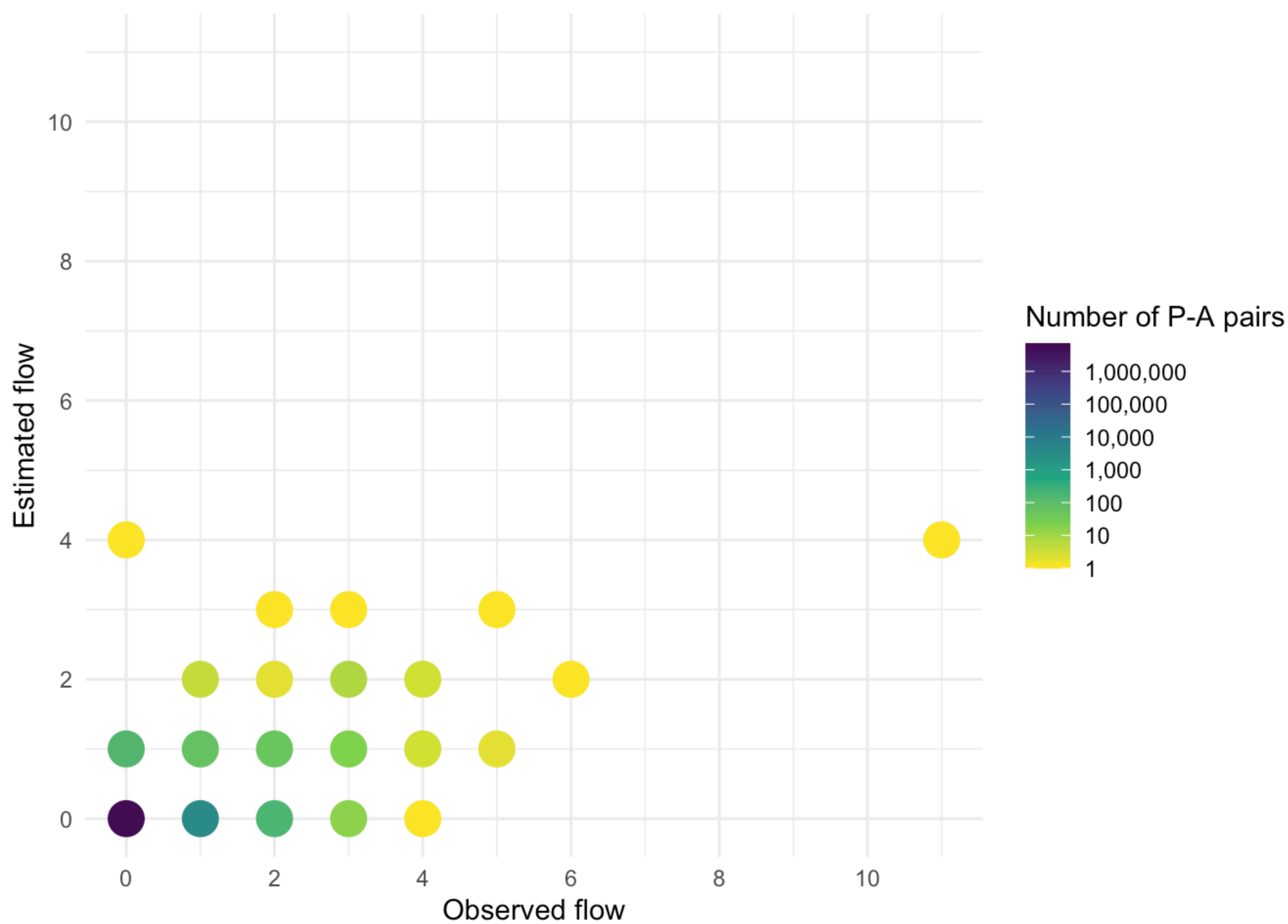
The average travel times observed vs. estimated seem closer

Reevaluating after model calibration

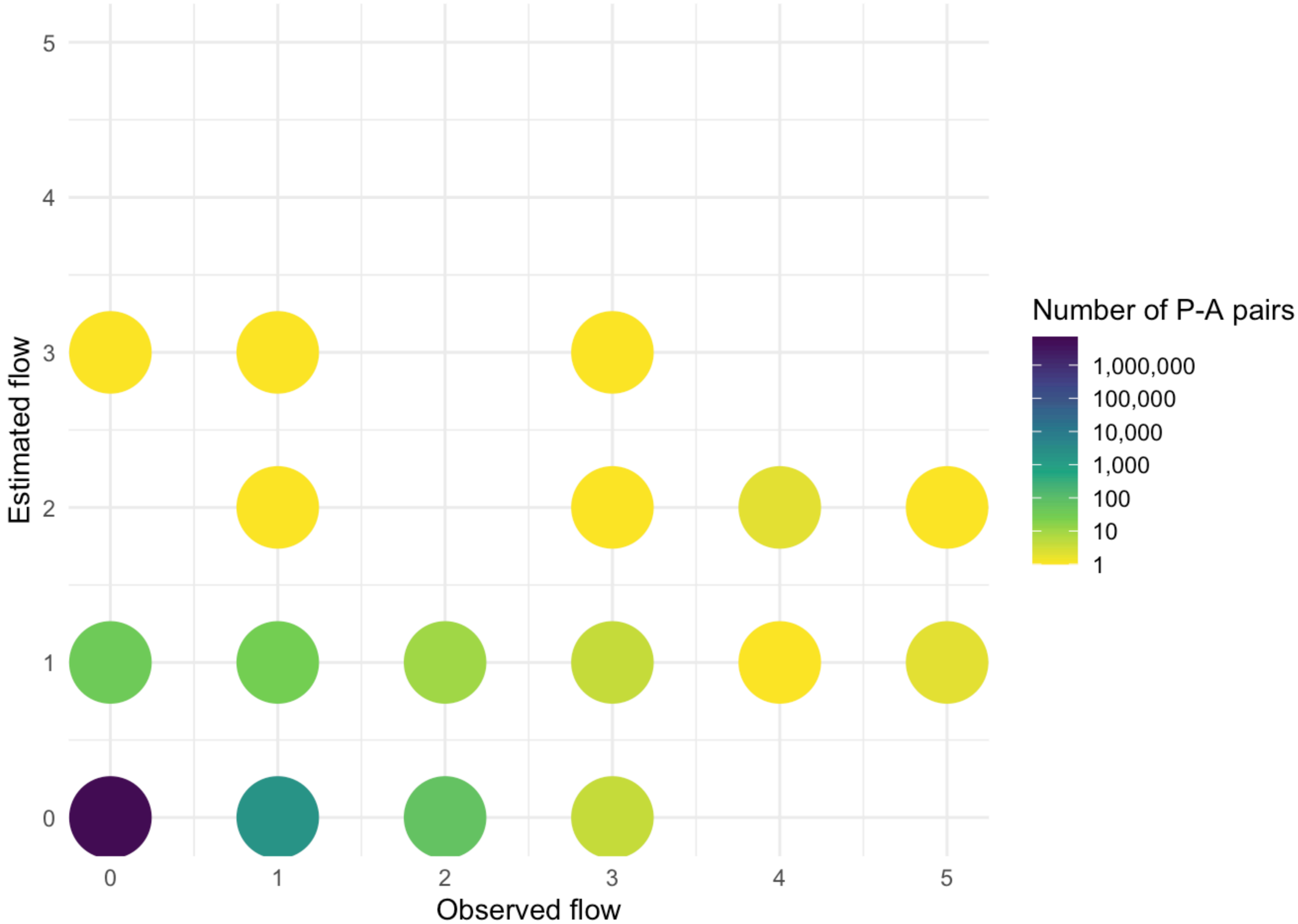
```
# Calculating RMSE
avg_tts <- avg_tts |>
  mutate(rmse = c((mean((flow_tt$flow_goods - flow_tt$goods_flow_est)^2))^0.5,
                  (mean((flow_tt$flow_trade - flow_tt$trade_flow_est)^2))^0.5,
                  (mean((flow_tt$flow_serve - flow_tt$serve_flow_est)^2))^0.5,
                  (mean((flow_tt$flow_total - flow_tt$total_flow_est)^2))^0.5))

kable(avg_tts, digits = 2)
```

Worker sector	Average travel time (observed)	Average travel time (estimated)	rmse
Goods	17.07	17.06	0.03
Trade	16.95	16.85	0.02
Service	16.37	16.37	0.04
Total	16.65	16.66	0.05

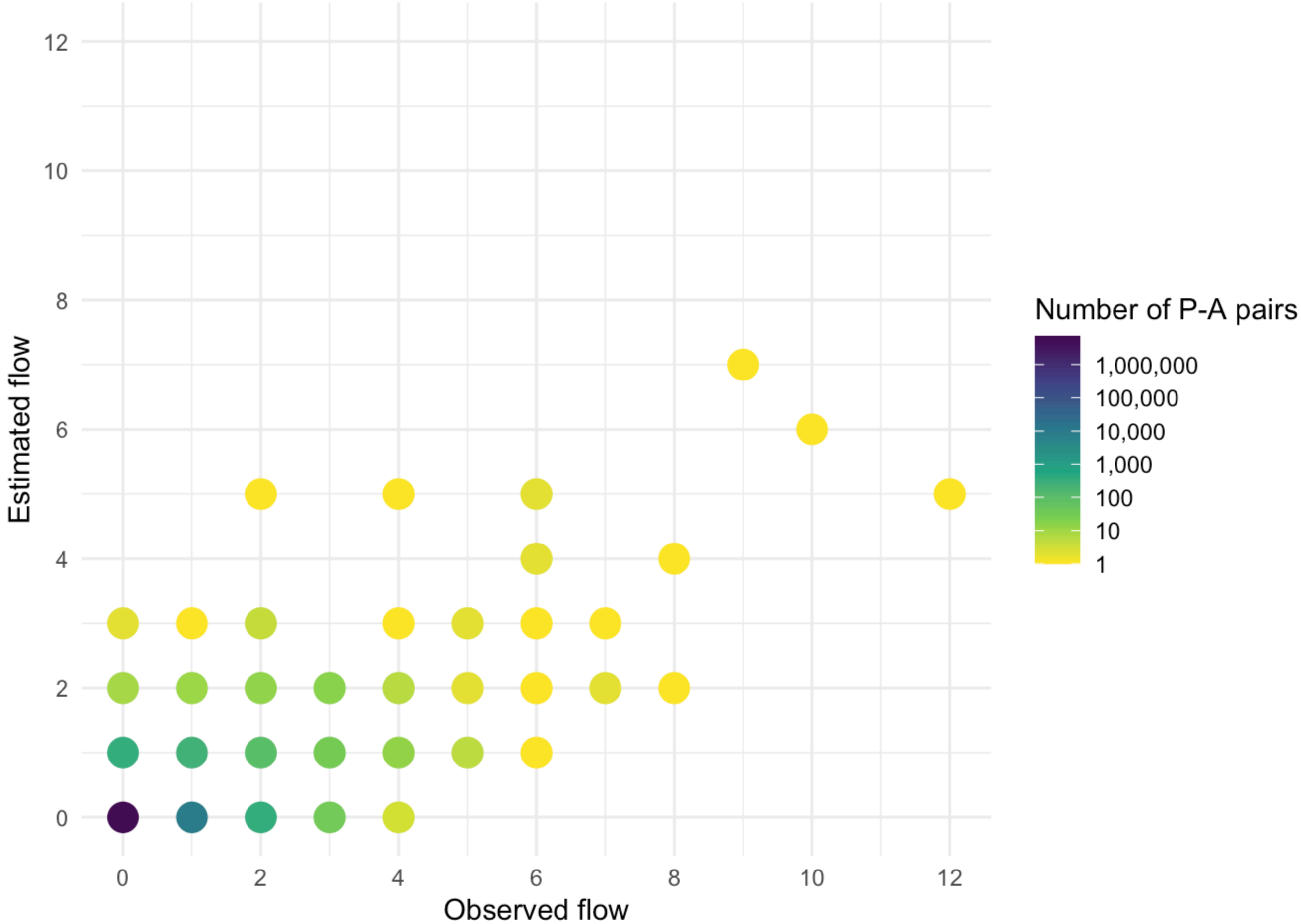
```
# Visually comparing estimated vs. observed

# Goods sector
plot_flows(flow_tt,
           obs_col_name = "flow_goods",
           est_col_name = "goods_flow_est")
```



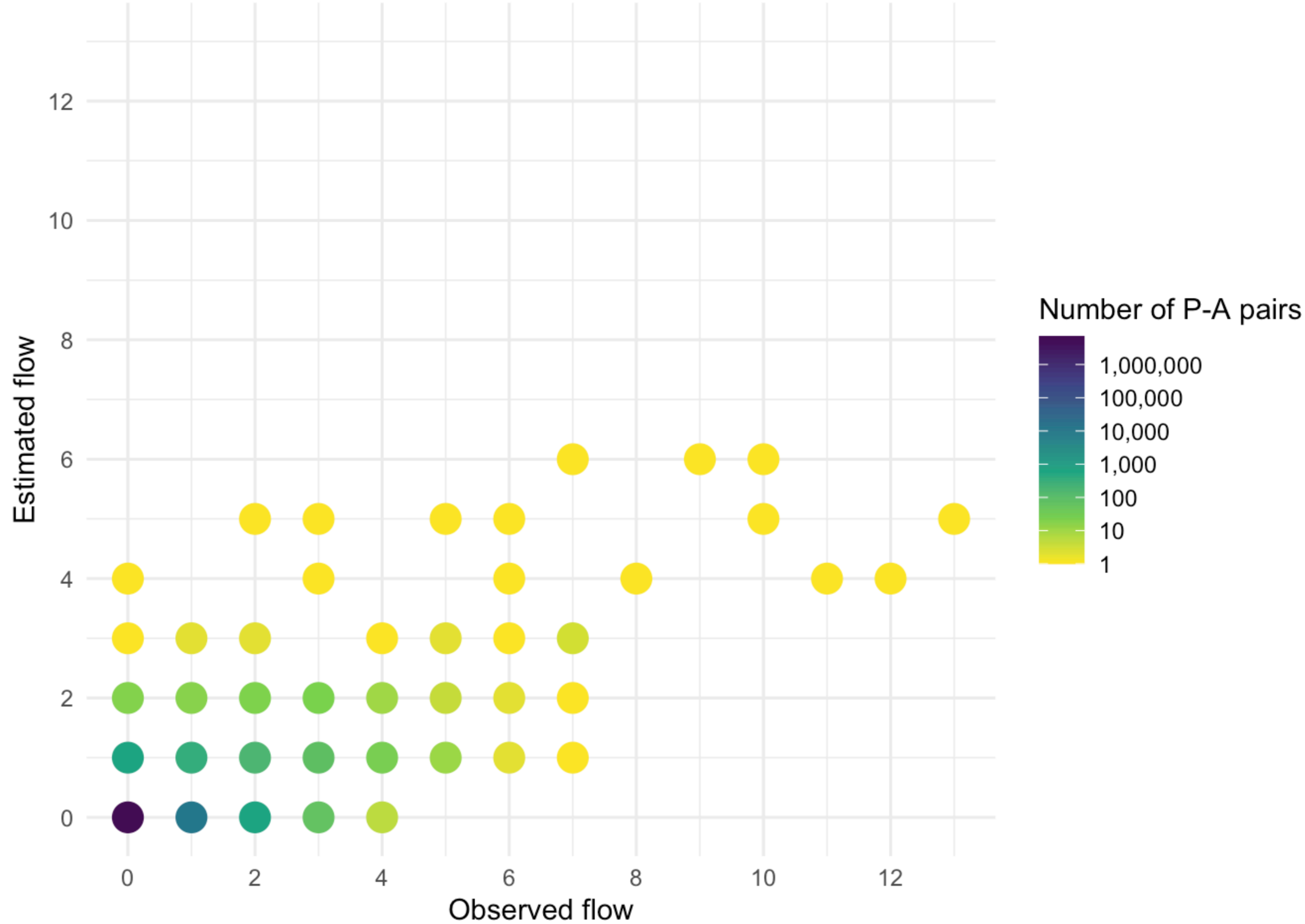
```
# Trade sector
plot_flows(flow_tt,
           obs_col_name = "flow_trade",
           est_col_name = "trade_flow_est")
```




```
# Service sector
plot_flows(flow_tt,
           obs_col_name = "flow_serve",
           est_col_name = "serve_flow_est")
```



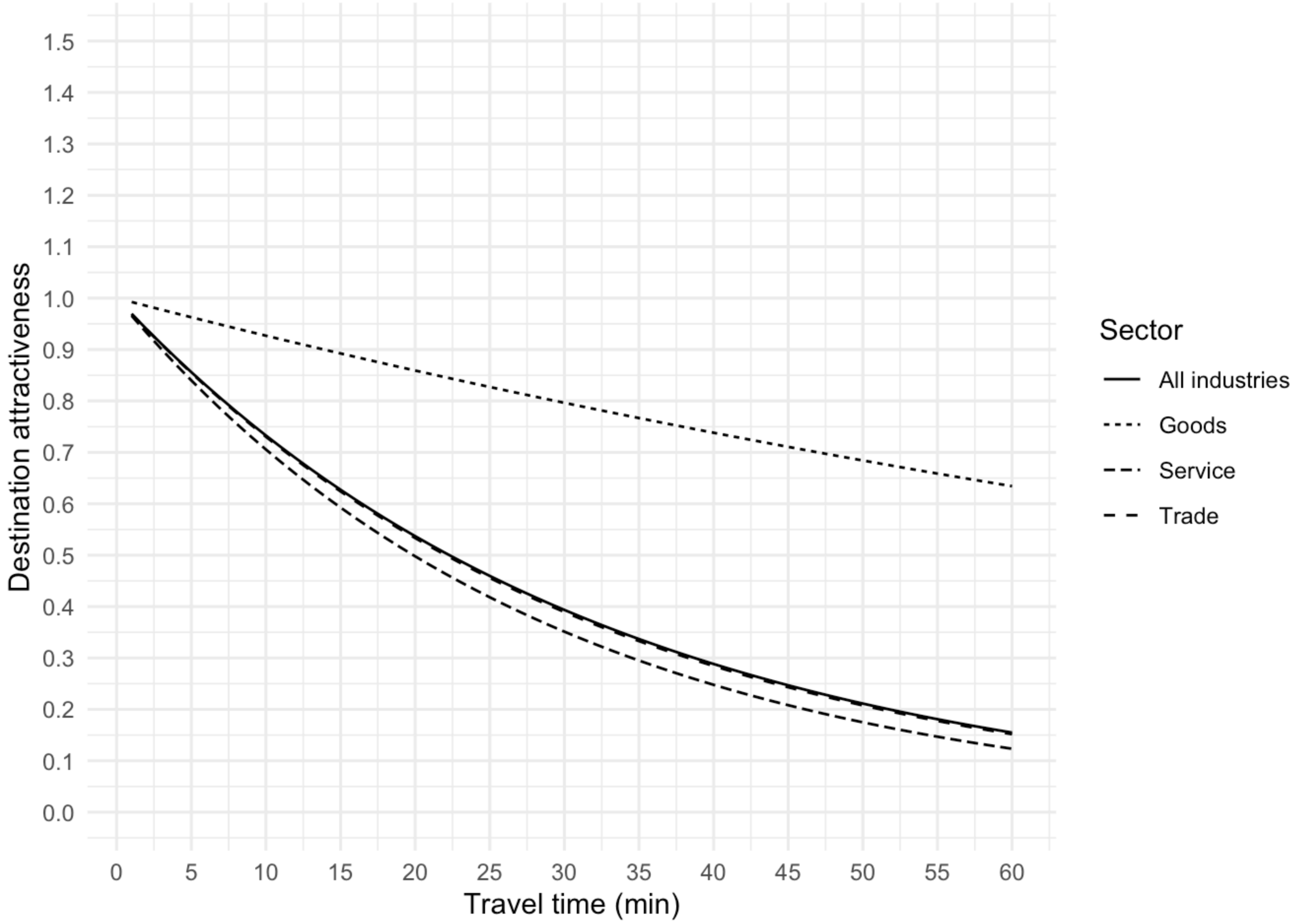

```
# All workers
plot_flows(flow_tt,
           obs_col_name = "flow_total",
           est_col_name = "total_flow_est")
```



Interpreting calibrated parameters

Calibrating the beta parameter resulted in an improved model fit. What does the difference in model results tell us about peoples' travel choices?

```
friction <- tibble(`Travel time (min)` = seq(1, 60, by=1)) |>
  mutate(Goods = exp(-1 * betas_table$beta_calibrated[1] * `Travel time (min)`),
         Trade = exp(-1 * betas_table$beta_calibrated[2] * `Travel time (min)`),
         Service = exp(-1 * betas_table$beta_calibrated[3] * `Travel time (min)`),
         `All industries` =
           exp(-1 * betas_table$beta_calibrated[4] * `Travel time (min)`)) |>
  pivot_longer(cols = -`Travel time (min)`,
               names_to = "Sector") |>
  rename(`Destination attractiveness` = value) |>
  filter(`Destination attractiveness` < 2)

ggplot(friction) +
  geom_line(aes(x = `Travel time (min)`,
                y = `Destination attractiveness`,
                linetype = Sector)) +
  scale_x_continuous(breaks = seq(0, 60, by=5)) +
  scale_y_continuous(breaks = seq(0, 2, by=0.1),
                     limits = c(0, 1.5)) +
  theme_minimal()
```




```
# Printing the betas table
betas_table
```

```
## # A tibble: 4 × 3
##   Industry beta_initial beta_calibrated
##   <chr>           <dbl>           <dbl>
## 1 Goods          0.0586         0.00759
## 2 Trade          0.0590         0.0314
## 3 Service        0.0611         0.0349
## 4 Total          0.0601         0.0311
```