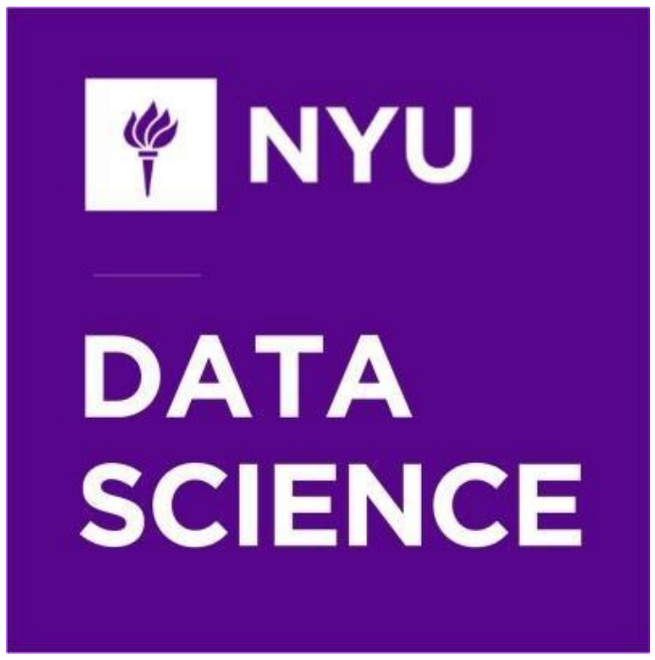
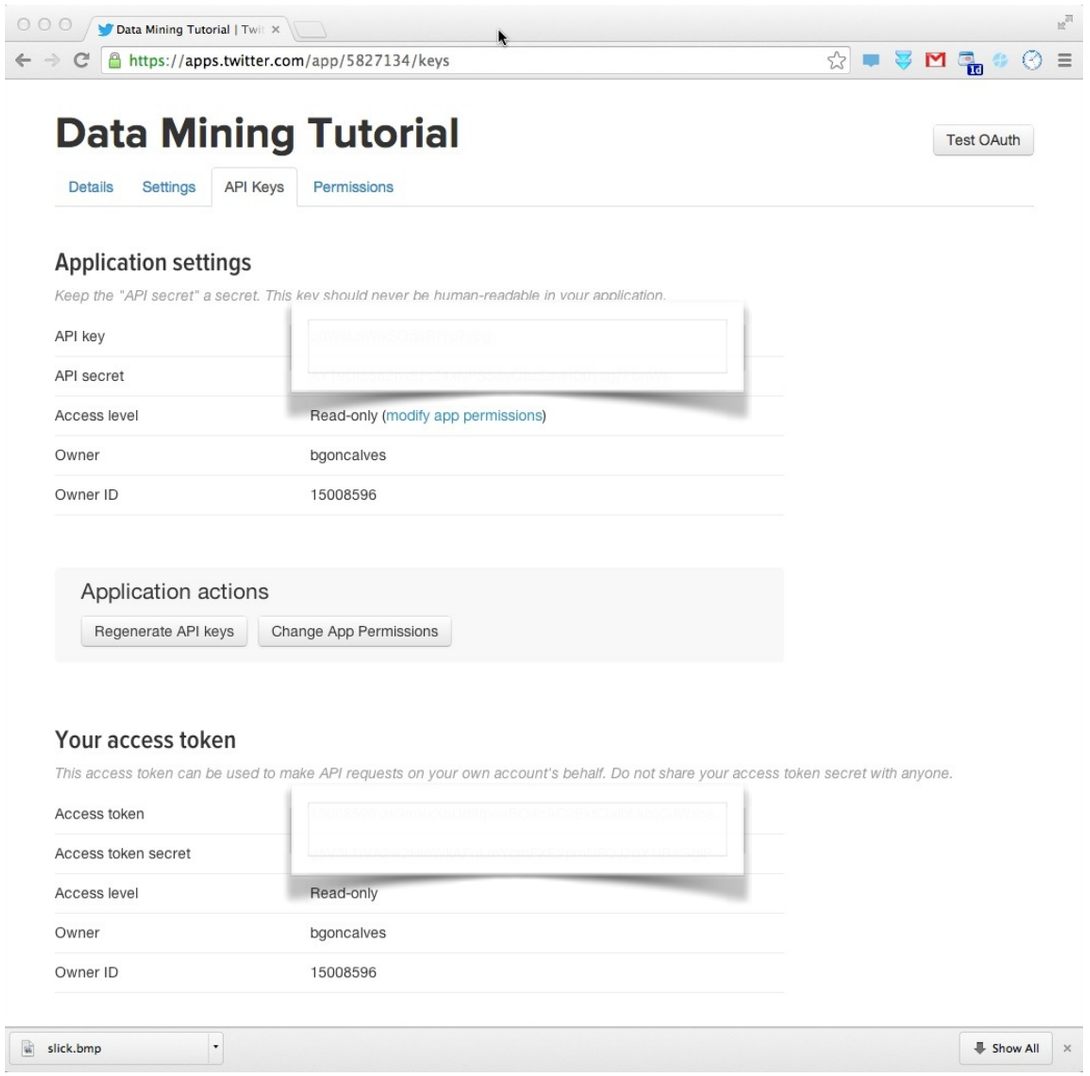
Online Social Networks and Mobility - Practical

Bruno Gonçalves

www.bgoncalves.com



Registering an Application



API Basics

https://dev.twitter.com/docs

- The twitter module provides the oauth interface. We just need to provide the right credentials.

- Best to keep the credentials in a dict and parametrize our calls with the dict key. This way we can switch between different accounts easily.

- .Twitter(auth) takes an OAuth instance as argument and returns a Twitter object that we can use to interact with the API

- Twitter methods mimic API structure

- 4 basic types of objects:
  - Tweets
  - Users
  - Entities
  - Places

Authenticating with the API

```
import twitter
from twitter_accounts import accounts

app = accounts["social"]

auth = twitter.oauth.OAuth(app["token"],
                           app["token_secret"],
                           app["api_key"],
                           app["api_secret"])

twitter_api = twitter.Twitter(auth=auth)
```

- In the remainder of this course, the accounts dict will live inside the twitter_accounts.py file
- 4 basic types of objects:
  - Tweets
  - Users
  - Entities
  - Places

twitter_authentication.py

Searching for Tweets

https://dev.twitter.com/docs/api/1.1/get/search/tweets

- .search.tweets(query, count)
  - query is the content to search for
  - count is the maximum number of results to return
- returns dict with a list of "statuses" and "search_metadata"


```
{u'completed_in': 0.027,
u'count': 15,
u'max_id': 438088492577345536,
u'max_id_str': u'438088492577345536',
u'next_results': u'?max_id=438088485145034752&q=soccer&include_entities=1',
u'query': u'soccer',
u'refresh_url': u'?since_id=438088492577345536&q=soccer&include_entities=1',
u'since_id': 0,
u'since_id_str': u'0'}
```


- search_results["search_metadata"]["next_results"] can be used to get the next page of results

Searching for Tweets

https://dev.twitter.com/docs/api/1.1/get/search/tweets

```
query = "instagram"
count = 200

search_results = twitter_api.search.tweets(q=query, count=count)

statuses = search_results["statuses"]
tweet_count = 0

while True:
    try:
        next_results = search_results["search_metadata"]["next_results"]

        args = dict(parse.parse_qsl(next_results[1:]))

        search_results = twitter_api.search.tweets(**args)
        statuses = search_results["statuses"]

        print(search_results["search_metadata"]["max_id"])

        for tweet in statuses:
            tweet_count += 1

            if tweet_count % 10000 == 0:
                print(tweet_count,file=sys.stderr)

            print (tweet["text"])
    except:
        break
```

twitter_search.py

User Timeline

https://dev.twitter.com/docs/api/1.1/get/statuses/user_timeline

- .statuses.user_timeline() returns a set of tweets posted by a single user
- Important options:
  - include_rts='true' to Include retweets by this user
  - count=200 number of tweets to return in each call
  - trim_user='true' to not include the user information (save bandwidth and processing time)
  - max_id=1234 to include only tweets with an id lower than 1234
- Returns at most 200 tweets in each call. Can get all of a users tweets (up to 3200) with multiple calls using max_id

User Timeline

https://dev.twitter.com/docs/api/1.1/get/statuses/user_timeline

```
import twitter
from twitter_accounts import accounts

app = accounts["social"]

auth = twitter.oauth.OAuth(app["token"],
                           app["token_secret"],
                           app["api_key"],
                           app["api_secret"])

twitter_api = twitter.Twitter(auth=auth)
screen_name = "BarackObama"

args = { "count" : 200,
         "trim_user": "true",
         "include_rts": "true"
        }

tweets = twitter_api.statuses.user_timeline(screen_name = screen_name, **args)
tweets_new = tweets

while len(tweets_new) > 0:
    max_id = tweets[-1]["id"] - 1
    tweets_new = twitter_api.statuses.user_timeline(screen_name = screen_name, max_id=max_id, **args)
    tweets += tweets_new

print("Found", len(tweets), "tweets")
```


twitter_timeline.py

Streaming Geocoded data

https://dev.twitter.com/streaming/overview/request-parameters#locations

- The Streaming api provides realtime data, subject to filters

- Use TwitterStream instead of Twitter object (.TwitterStream(auth=twitter_api.auth))

- .status.filter(track=q) will return tweets that match the query q in real time

- Returns generator that you can iterate over

- .status.filter(locations=bb) will return tweets that occur within the bounding box bb in real time

- bb is a comma separated pair of lon/lat coordinates.
  - -180,-90,180,90 - World
  - -74,40,-73,41 - NYC

Streaming Geocoded data

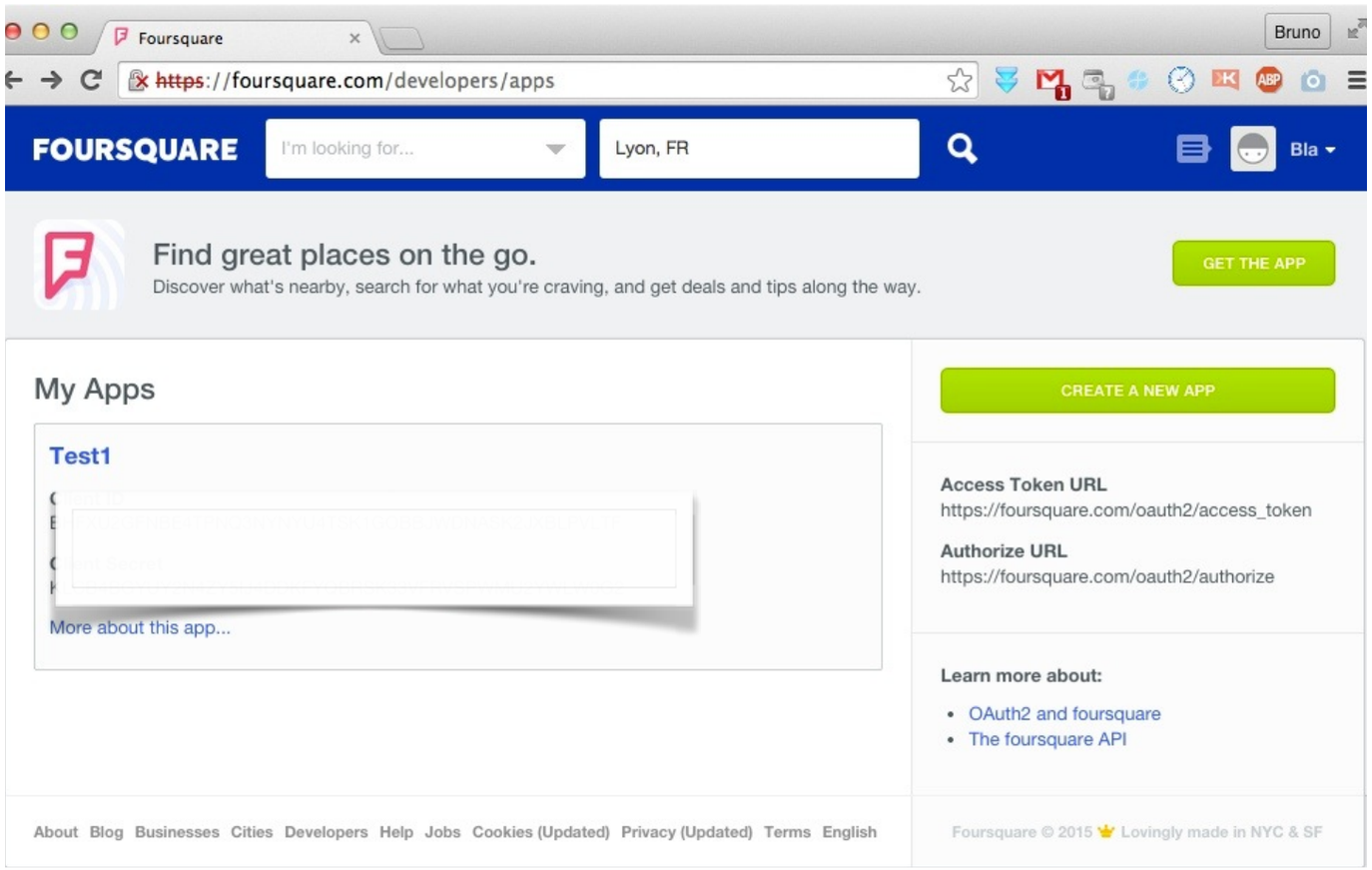
https://dev.twitter.com/streaming/overview/request-parameters#locations

```
import twitter
from twitter_accounts import accounts
import sys
import gzip

app = accounts["social"]

auth = twitter.oauth.OAuth(app["token"],
                           app["token_secret"],
                           app["api_key"],
                           app["api_secret"])

stream_api = twitter.TwitterStream(auth=auth)

query = "-74,40,-73,41"  # NYC
stream_results = stream_api.statuses.filter(locations=query)
tweet_count = 0

fp = gzip.open("NYC.json.gz", "a")

for tweet in stream_results:
    try:
        tweet_count += 1
        print (tweet_count, tweet["id"])
        print(tweet, file=fp)
    except:
        pass

    if tweet_count % 10000 == 0:
        print(tweet_count, file=sys.stderr)
        break
```

twitter_location.py

Foursquare

Registering An Application



Registering An Application

- with this code we can now request an auth_token which will allow us to authenticate with the Foursquare API

```
access_token = client.oauth.get_token('CODE')
```

- This will return the OAuth2 access_token that we can then use directly.

```
import foursquare
from foursquare_accounts import accounts

app = accounts["tutorial"]

client = foursquare.Foursquare(client_id=app["client_id"],

client_secret=app["client_secret"])

client.set_access_token(app["access_token"])
```

- Much simpler and intuitive
- Less prone to mistakes
- Automatically takes care of all the dirty details

foursquare_authentication.py

Venues

https://developer.foursquare.com/overview/venues.html

- .venues(venue_id)
  Returns a venue object

- .venues.similar(venue_id)
  Returns a list of similar venues (abbreviated)

- .venues.search({“query”:query, “near”:location})
  Searches for places matching the query
  (“pizza”, “Eiffel Tower”, etc) near location (“Paris”, etc).

```
[u'city',
 u'cc',
 u'country',
 u'state',
 u'address',
 u'lat',
 u'lng']

[u'count',
 u'groups',
 u'summary']

[u'rating',
 u'reasons',
 u'likes',
 u'mayor',
 u'createdAt',
 u'verified',
 u'id',
 u'shortUrl',
 u'pageUpdates',
 u'location',
 u'tips',
 u'listed',
 u'canonicalUrl',
 u'tags',
 u'photos',
 u'attributes',
 u'stats',
 u'dislike',
 u'hereNow',
 u'categories',
 u'name',
 u'like',
 u'phrases',
 u'specials',
 u'contact',
 u'popular',
 u'timeZone']
```

Similar Venues

https://developer.foursquare.com/overview/venues.html

```
import foursquare
from foursquare_accounts import accounts

app = accounts["tutorial"]

client = foursquare.Foursquare(client_id=app["client_id"],
                               client_secret=app["client_secret"])

client.set_access_token(app["access_token"])

venue_id = "43695300f964a5208c291fe3"
venue = client.venues(venue_id)
similar = client.venues.similar(venue_id)

print("Similar venues to", venue["venue"]["name"], "(", venue["venue"]["hereNow"]["summary"], ")")

for venue in similar["similarVenues"]["items"]:
    print(venue["name"])
```

foursquare_venues.py

Venue Search

https://developer.foursquare.com/overview/venues.html

```
import foursquare
from foursquare_accounts import accounts

app = accounts["tutorial"]

client = foursquare.Foursquare(client_id=app["client_id"],
                               client_secret=app["client_secret"])

client.set_access_token(app["access_token"])

results = client.venues.search({"query": "candy", "ll": "45.507297, -73.565469"})

for result in results["venues"]:
    print(result["name"])
```

foursquare_search.py

GPW

http://sedac.ciesin.columbia.edu/data/collection/gpw-v4

Population Density, v4: Gridde

Bruno

sedac.ciesin.columbia.edu/data/set/gpw-v4-population-density

EARTHDATA | Data Discovery | DAACs | Community | Science Disciplines

SOCIOECONOMIC DATA AND APPLICATIONS CENTER (SEDAC)

A Data Center in NASA's Earth Observing System Data and Information System (EOSDIS) — Hosted by CIESIN at Columbia University

Search SEDAC | Data | Login

DATA | MAPS | THEMES | RESOURCES | SOCIAL MEDIA | ABOUT | HELP

Gridded Population of the World (GPW), v4

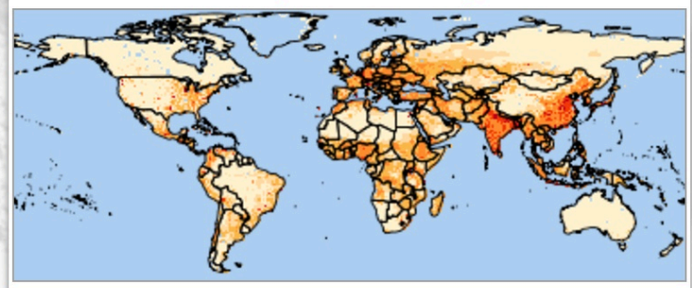
Follow Us: | Share:

- Collection Overview
- Methods
- Data Sets (8)
  - *Population Density, v4 (2000, 2005, 2010, 2015, 2020)*
  - Show All...
- Map Gallery (27)
- Map Services (26)
- Citations
- FAQs
- What's New in GPWv4?
- Documentation
- What is UN-adjusted

Population Density, v4 (2000, 2005, 2010, 2015, 2020)

Set Overview | Data Download | Maps | Map Services | Documentation | Metadata

GPWv4: Population Density - 2000



1 of 5

Purpose:

To provide estimates of population density for the years 2000, 2005, 2010, 2015, and 2020, based on counts consistent with national censuses and population registers, as raster data to facilitate data integration.

Abstract:

Gridded Population of the World, Version 4 (GPWv4) Population Density consists of estimates of human population density based on counts consistent with national censuses and population registers, for the years 2000, 2005, 2010, 2015, and 2020. A proportional allocation gridding algorithm, utilizing approximately 12.5 million national and sub-national administrative units, is used to assign population values to 30 arc-second (~1 km) grid cells. The population density grids are created by dividing the population count grids by the land area grids. The pixel values represent persons per square kilometer.

Recommended Citation(s)*:

ASCII Grid

- Perhaps the simples raster file
- ASCII text based
- A small header

```
ncols         246
nrows         119
xllcorner     -126.500000000000
yllcorner     22.750000000000
cellsize      0.250000000000
NODATA_value  -9999
```

- Followed by rows of numbers
- Very convenient to Read and Write

US_pop.asc

ASCII Gri

```python
import numpy as np
import matplotlib.pyplot as plt

def map_points(xllcorner, yllcorner, cellsize, nrows, x, y):
    x = int((x-xllcorner)/cellsize)
    y = (nrows-1)-int((y-yllcorner)/cellsize)

    return x, y

fp = open("US_pop.asc")
ncols, count = fp.readline().split()
ncols = int(count)
nrows, count = fp.readline().split()
nrows = int(count)
xllcorner, value = fp.readline().split()
xllcorner = float(value)
yllcorner, value = fp.readline().split()
yllcorner = float(value)
cellsize, value = fp.readline().split()
cellsize = float(value)

NODATA_value, value = fp.readline().split()
NODATA_value = float(value)

data = []
for line in fp:
    fields = line.strip().split()
    data.append([float(field) for field in fields])

data = np.array(data)
data[data==NODATA_value] = 0

x = -74.243251
y = 40.730503

coord_x, coord_y = map_points(xllcorner, yllcorner, cellsize, nrows, x, y)
print(data[coord_y, coord_x])
```

asc_load.py

ASCII Gri

```python
import numpy as np
import matplotlib.pyplot as plt

def map_points(xllcorner, yllcorner, cellsize, nrows, x, y):
    x = int((x-xllcorner)/cellsize)
    y = (nrows-1)-int((y-yllcorner)/cellsize)

    return x, y

fp = open("US_pop.asc")
ncols, count = fp.readline().split()
ncols = int(count)
nrows, count = fp.readline().split()
nrows = int(count)
xllcorner, value = fp.readline().split()
xllcorner = float(value)
yllcorner, value = fp.readline().split()
yllcorner = float(value)
cellsize, value = fp.readline().split()
cellsize = float(value)

NODATA_value, value = fp.readline().split()
NODATA_value = float(value)

data = []
for line in fp:
    fields = line.strip().split()
    data.append([float(field) for field in fields])

data = np.array(data)
data[data==NODATA_value] = 0

x = -74.243251
y = 40.730503

coord_x, coord_y = map_points(xllcorner, yllcorner, cellsize, nrows, x, y)
print(data[coord_y, coord_x])
```

asc_load.py

ASCII Grid

asc_load.py

US Flight dataset

https://www.transtats.bts.gov/databases.asp?Mode_ID=1&Mode_Desc=Aviation&Subject_ID2=0

RITA | BTS | Transtats

Bruno

Secure https://www.transtats.bts.gov/databases.asp?Mode_ID=1&Mode_Desc=Aviation&Subject_ID2=0

United States Department of Transportation

Ask a Research Question | A-Z Index

Bureau of Transportation Statistics

Explore Topics and Geography | Browse Statistical Products and Data | Learn About BTS and Our Work | Newsroom

OST-R > BTS

TranStats

Search this site:

Go

Advanced Search

Resources

- Database Directory
- Glossary
- Upcoming Releases
- Data Release History

Data Finder

By Mode

- Aviation
- Maritime
- Highway
- Transit
- Rail
- Pipeline
- Bike/Pedestrian
- Other

By Subject

- Safety
- Freight Transport
- Passenger Travel
- Infrastructure

Data Library: Aviation

Databases | Summary Tables | Glossary | Filter Subject: All Subjects | Go

<<Prev Rows 1 to 15 of 26 Next>>

Database Name	Description	
Air Carrier Financial Reports (Form 41 Financial Data)	Form 41 Financial Schedule consists of financial information on large U.S. certified air carriers--includes balance sheet, cash flow, employment, income statement, fuel cost and consumption, aircraft operating expenses, and operating expenses. **Note:** Numbers presented on B1, B11 Balance Sheet and P11, P12 Statement of Operations now follow the format of common public financial documents. This format reverses signs from the accounting format in which numbers appeared prior to 10/18/2006 (Examples).	Profile
Air Carrier Statistics (Form 41 Traffic)- U.S. Carriers	Monthly data reported by certificated U.S. air carriers on passengers, freight and mail transported. Also includes aircraft type, service class, available capacity and seats, and aircraft hours ramp-to-ramp and airborne.	Profile
Air Carrier Statistics (Form 41 Traffic)- All Carriers	Monthly data reported by certificated U.S. and foreign air carriers on passengers, freight and mail transported. Also includes aircraft type, service class, available capacity and seats, and aircraft hours ramp-to-ramp and airborne.	Profile
Air Carrier Summary Data (Form 41 and 298C Summary Data)	Summary data of the non-stop segment and on-flight market data reported by air carriers on Form 41 and Form 298C	Profile
Airline On-Time Performance Data	Monthly data reported by US certified air carriers that account for at least one percent of domestic scheduled passenger revenues--includes scheduled and actual arrival and departure times for flights.	Profile
Airline Origin and Destination Survey (DB1B)	Origin and Destination Survey (DB1B) is a 10% sample of airline tickets from reporting carriers. Data includes origin, destination and other itinerary details of passengers transported.	Profile
American Travel Survey (ATS) 1995	National data on the nature and characteristics of long-distance personal travel, from a household survey conducted by BTS about every five years.	Profile

US Flight dataset

bit.ly/BTS_Airports

RITA | BTS | Transtats

Bruno

Secure https://www.transtats.bts.gov/FieldInfo.asp?Field_Desc=Origin%20Airport%2C%20Airport%20ID.%20An%20identification%20number%20as...

United States Department of Transportation

Ask a Research Question | A-Z Index

Bureau of Transportation Statistics

Explore Topics and Geography | Browse Statistical Products and Data | Learn About BTS and Our Work | Newsroom

OST-R > BTS

TranStats

Search this site:

Go

Advanced Search

Resources

- Database Directory
- Glossary
- Upcoming Releases
- Data Release History

Data Finder

By Mode

- Aviation
- Maritime
- Highway
- Transit
- Rail
- Pipeline
- Bike/Pedestrian
- Other

By Subject

- Safety
- Freight Transport
- Passenger Travel
- Infrastructure

Air Carriers : T-100 Domestic Market (All Carriers)

Field: Origin Airport, Airport ID. An identification number assigned by US DOT to identify a unique airport. Use this field for airport analysis across a range of years because an airport can change its airport code and airport codes can be reused.

Format results for printing | Download Lookup Table | Databases | Data Tables | Table Contents

<<Prev Rows : 1 - 100 of 1974 Next>>

Code	Description
10001	Afognak Lake, AK: Afognak Lake Airport
10003	Granite Mountain, AK: Bear Creek Mining Strip
10004	Lik, AK: Lik Mining Camp
10005	Little Squaw, AK: Little Squaw Airport
10006	Kizhuyak, AK: Kizhuyak Bay
10008	Elizabeth Island, AK: Elizabeth Island Airport
10009	Homer, AK: Augustin Island
10010	Hudson, NY: Columbia County
10011	Peach Springs, AZ: Grand Canyon West
10012	Blairstown, NJ: Blairstown Airport
10013	Crosbyton, TX: Crosbyton Municipal
10014	Fairbanks/Ft. Wainwright, AK: Blair Lake
10015	Deadmans Bay, AK: Deadmans Bay Airport
10016	Hallo Bay, AK: Hallo Bay Airport
10017	Red Lake, AK: Red Lake Airport
10020	Selawik, AK: Roland Norton Memorial

US Airports

```
import googlemaps
from google_accounts import accounts

app = accounts["torino"]
gmaps = googlemaps.Client(key=app["api_key"])

line_count = 0
for line in open("flights/L_AIRPORT_ID.csv"):
    fields = line.strip()[:-1].split(',')

    line_count += 1

    if line_count > 1:
        try:
            airport_id = fields[0]
            airport_name = ", ".join(fields[2:])

            geocode_result = gmaps.geocode(airport_name)

            print(airport_id[1:-1], geocode_result[0]["geometry"]["location"]["lat"],\
                  geocode_result[0]["geometry"]["location"]["lng"])
        except:
            pass
```


google_airports.py

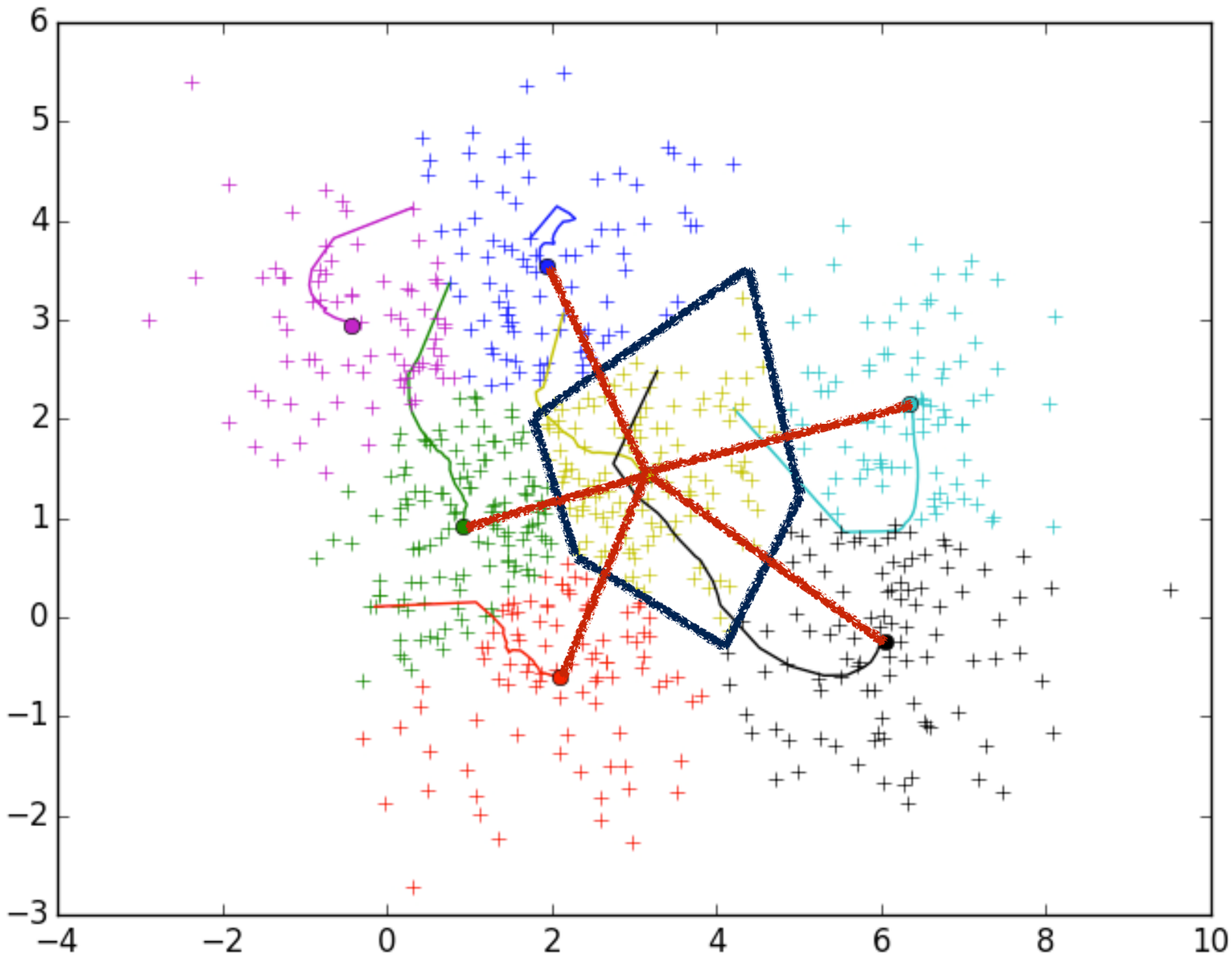
Distance Matrix

```
from geopy import distance

airports = {}

for line in open("airport_gps.dat"):
    fields = line.strip().split()

    airport_id = int(fields[0])
    lat = float(fields[1])
    lon = float(fields[2])

    airports[airport_id] = (lat, lon)

airport_list = airports.keys()

for airport_i in airport_list:
    for airport_j in airport_list:
        try:
            if airport_i != airport_j:
                dist = distance.distance(airports[airport_i], airports[airport_j]).km

                if dist < 20:
                    print(airport_i, airport_j, dist)
        except:
            pass
```


airport_matrix.py

K-Means: Structure

Voronoi Tesselation



Voronoi cells

```
pop, xllcorner, yllcorner, cellsize = load_asc('US_pop.asc')
airports = load_airports('airport_gps.dat')
coords = np.array(list(airports.values()))
nrows = pop.shape[0]
ncols = pop.shape[1]

seeds = []

for i in range(coords.shape[0]):
    x, y = map_points(xllcorner, yllcorner, cellsize, nrows, coords[i][1], coords[i][0])

    if x <0  or x>= ncols:
        continue

    if y <0 or y >= nrows:
        continue

    if pop[y, x] > 0:
        seeds.append((x,y))

seeds = np.array(seeds)

positions = np.zeros((nrows*ncols, 2), dtype='int')

count = 0
for i in range(nrows):
    for j in range(ncols):
        positions[count][0] = j
        positions[count][1] = i
        count += 1
```

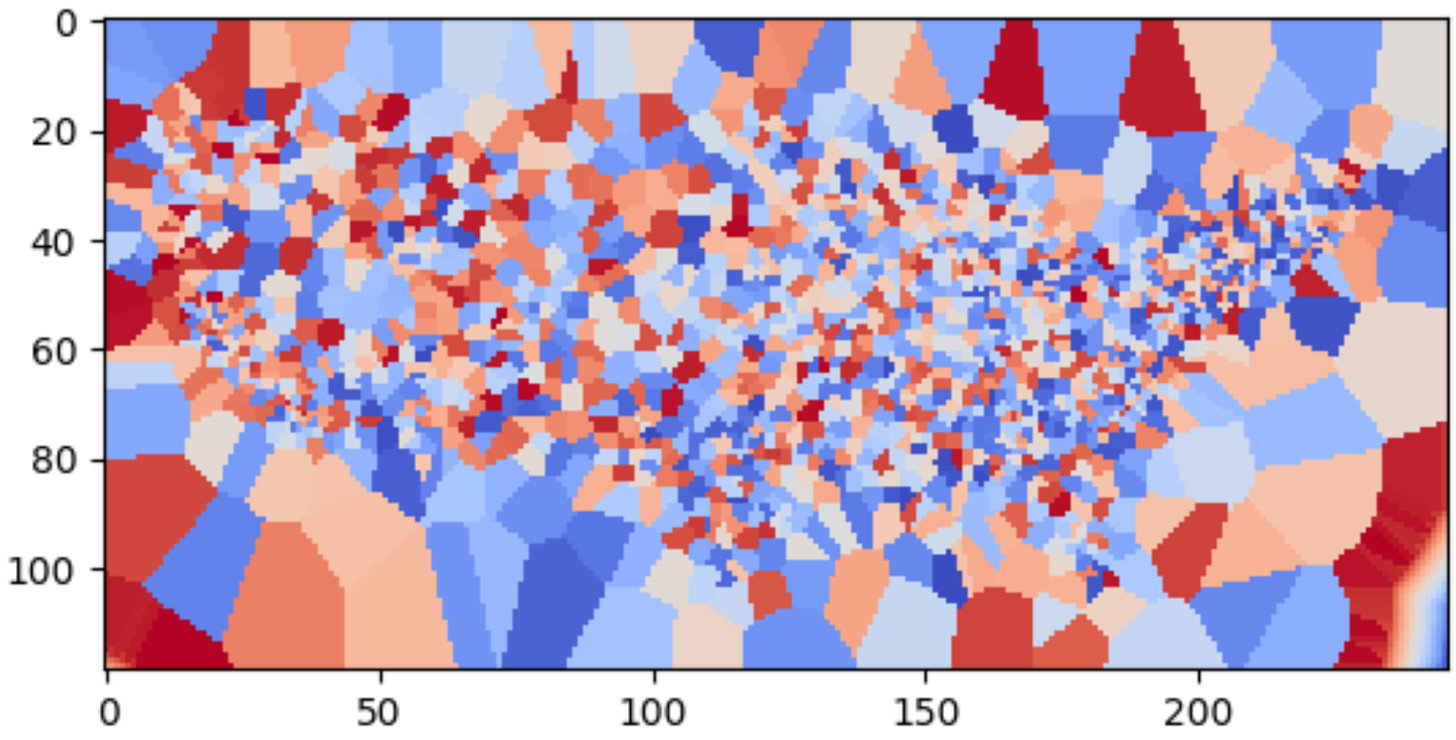

voronoi.py

Voronoi cells

```
kmeans = KMeans(n_clusters=seeds.shape[0], init=seeds, n_init=1, max_iter=1)
kmeans.fit(positions)

basins = np.zeros((nrows, ncols), dtype='int')

for i in range(len(kmeans.labels_)):
    basins[positions[i][1]][positions[i][0]] = kmeans.labels_[i]

plt.imshow(basins, cmap=cm.coolwarm)
plt.savefig('basins.png')
```


voronoi.py

Voronoi cells

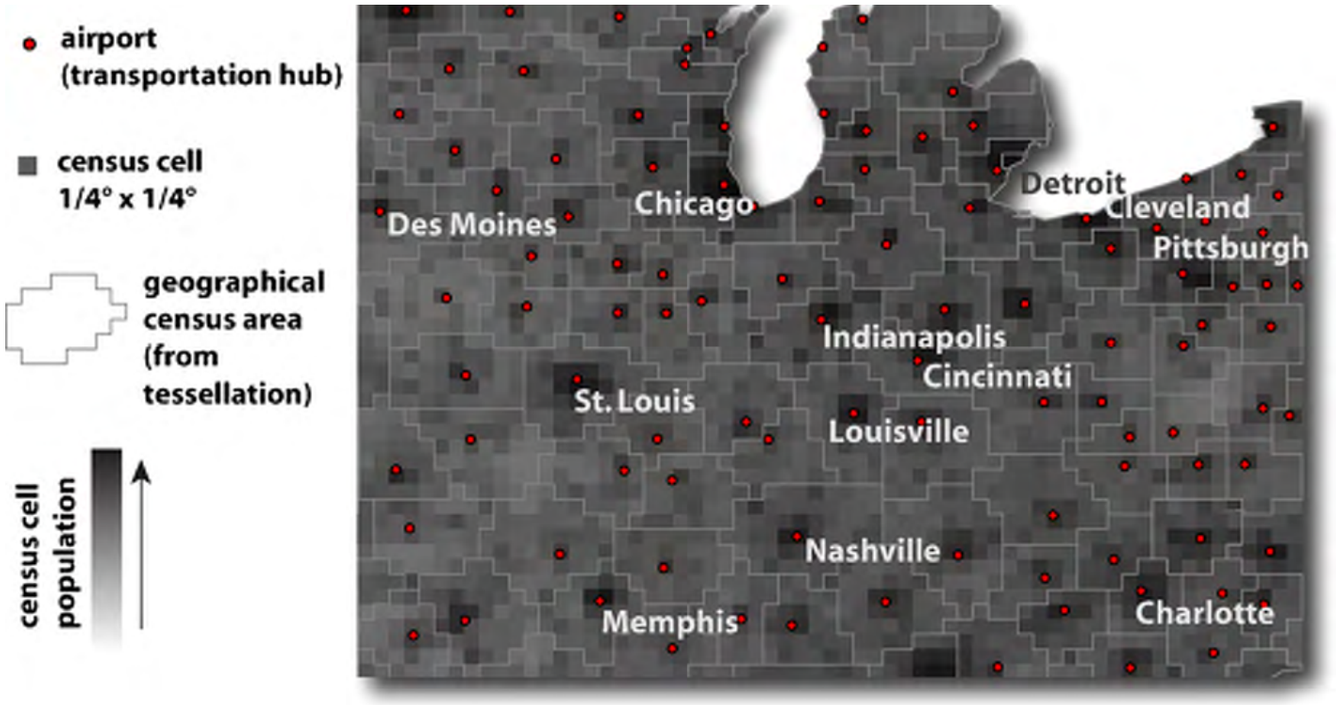


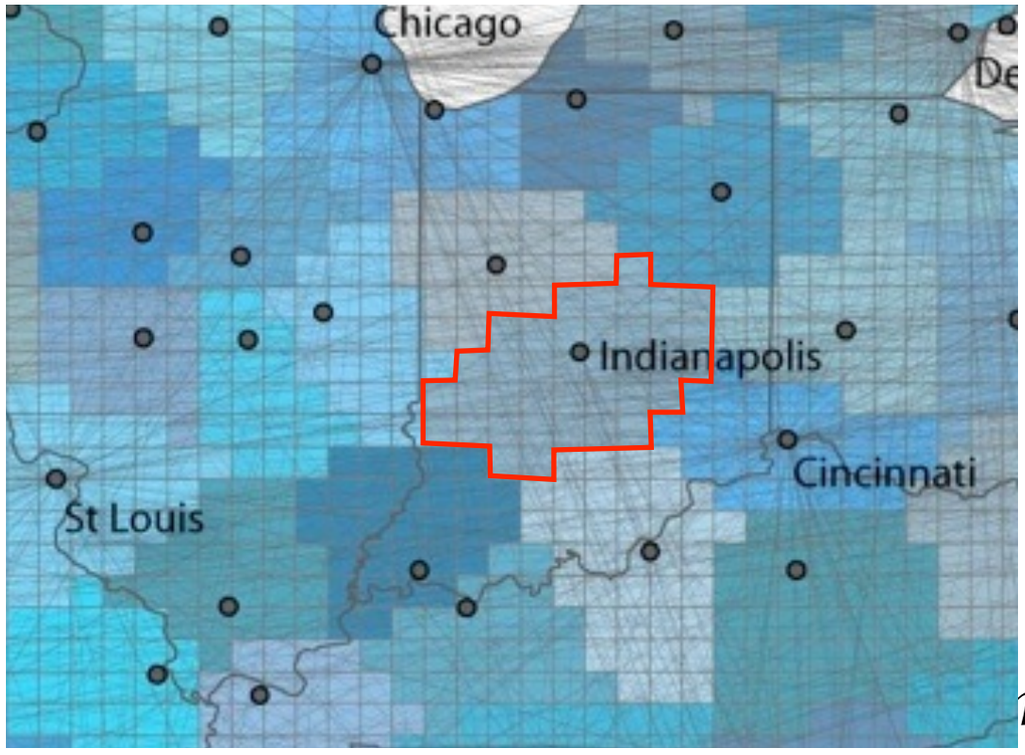
@bgoncalves

voronoi.py

Population Distribution

PNAS 106, 21484 (2009)



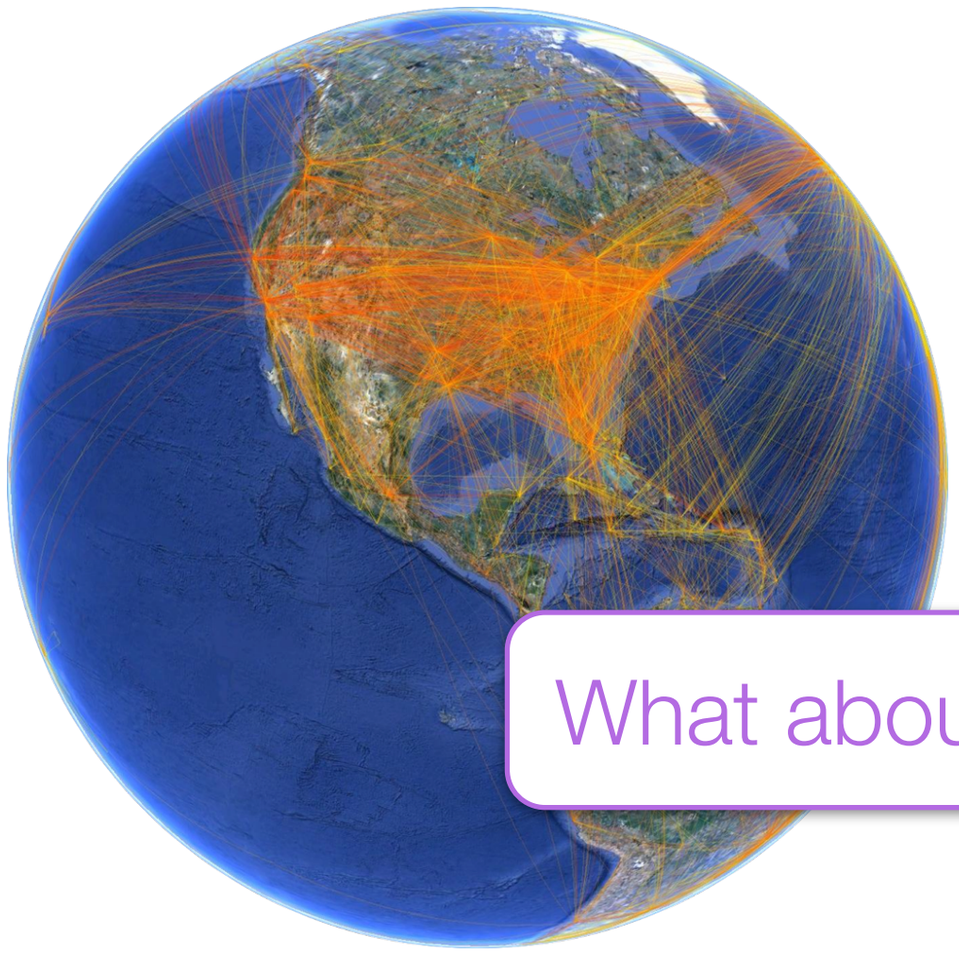


Complete IATA and OAG databases:

3362 airports worldwide
220 countries

Long Range Mobility

PNAS 106, 21484 (2009)



$$p_{jl} = \frac{w_{jl}}{N_j} \Delta t$$

- Probability that any individual in class X travel from j→l

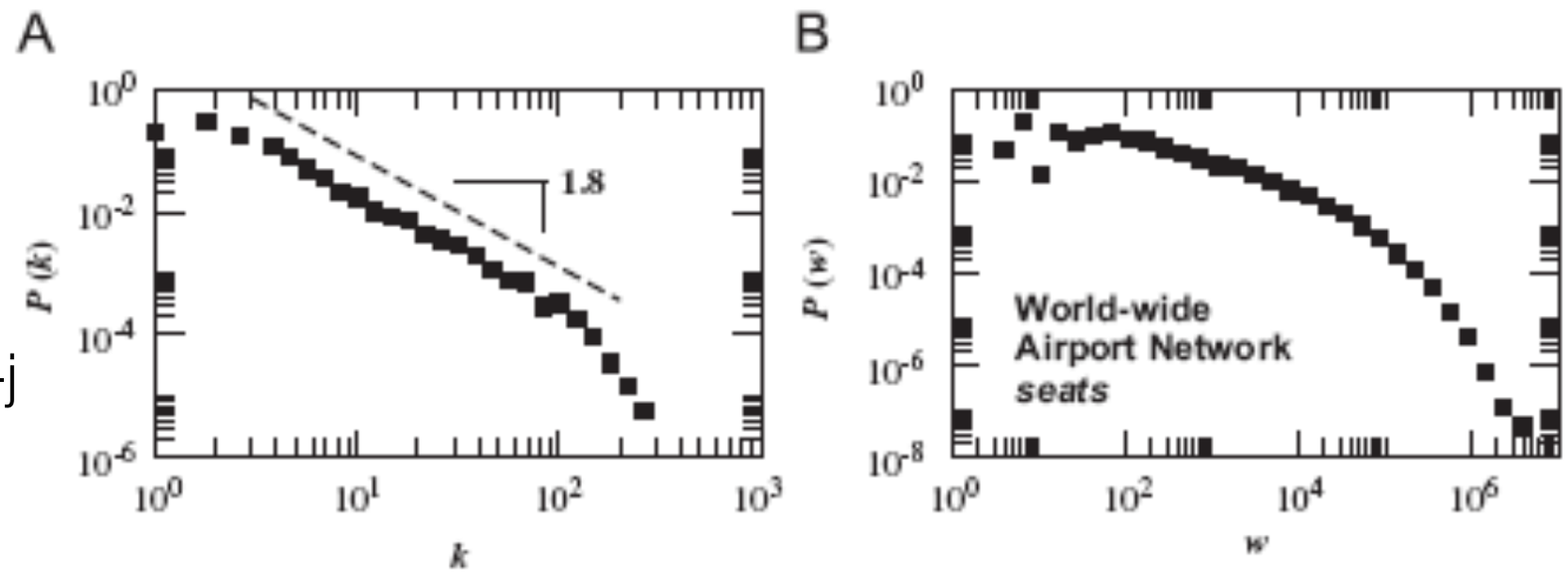
w

population

What about Short Range Mobility?

Complete IATA and OAG databases:

3362 airports worldwide
220 countries
> 20,000 connections
w_{ij} #passengers on connection i-j
>99% total traffic



Gravity-Law of Commuting

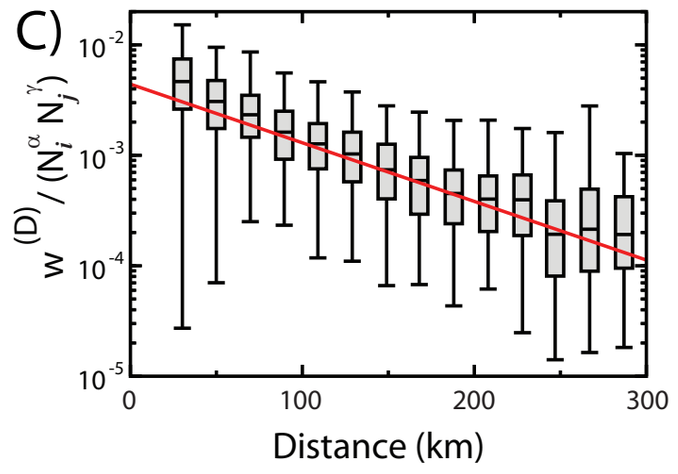
PNAS 106, 21484 (2009)

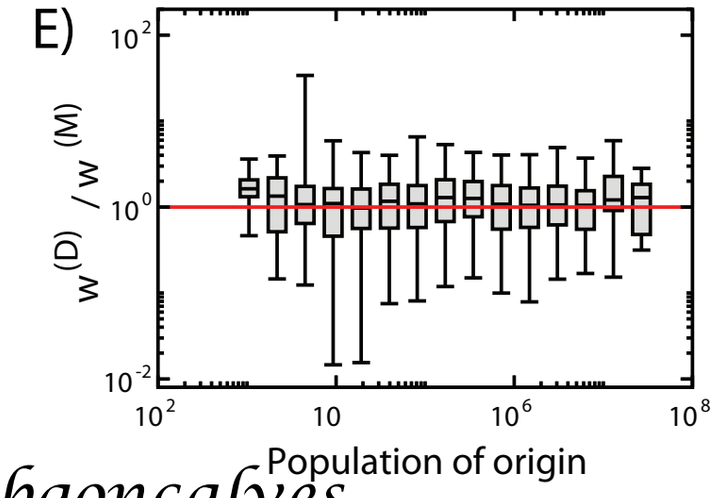
Table 1
Commuting networks in each continent. Number of countries (N), number of administrative units (V) and inter-links between them (E) are summarized.

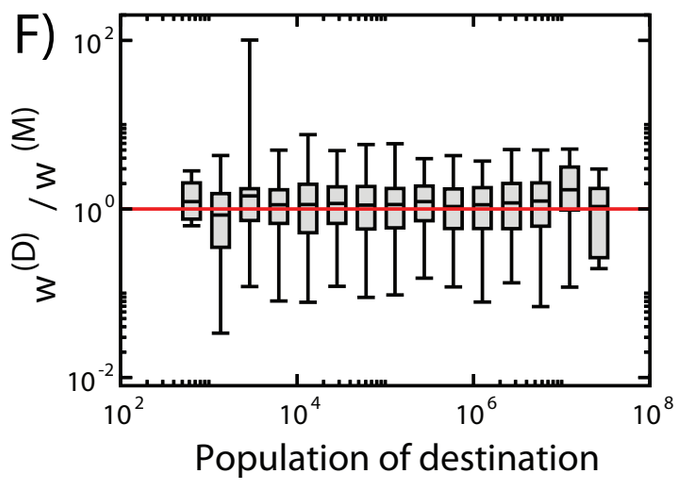
Continent	N	V	E
Europe	17	65,880	4,490,650
North America	2	6986	182,255
Latin America	5	4301	102,117
Asia	4	4355	380,385
Oceania	2	746	30,679
Total	30	82,268	5,186,186

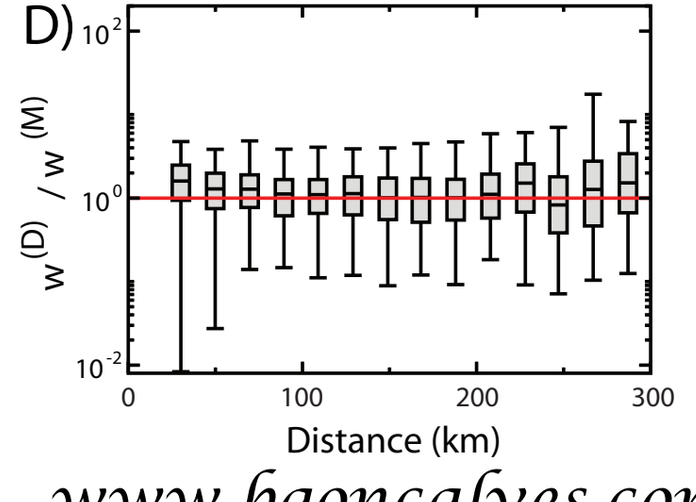
d (km)	Parameter	Estimate	Standard Error	p-value	R^2
≤ 300	α	0.46	0.01	$< 2E-16$	0.7972
	γ	0.64	0.01	$< 2E-16$	
	β	0.0122	0.0002	$< 2E-16$	
> 300	α	0.35	0.06	$6.91E-09$	0.5369
	γ	0.37	0.06	$2.12E-09$	

$$w_{ij} = C\, N_i^{\alpha}\, N_j^{\gamma}\, e^{-\beta\, d_{ij}}$$







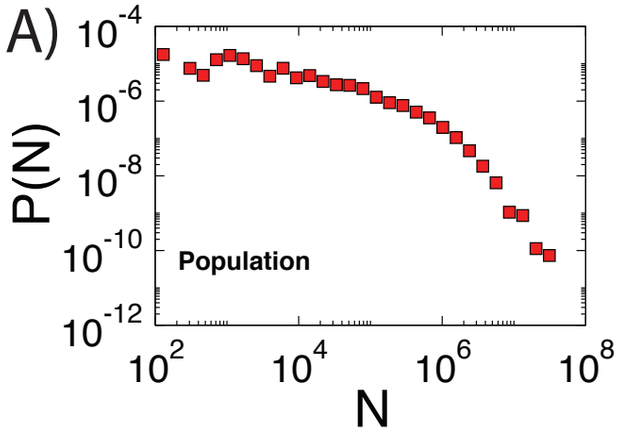


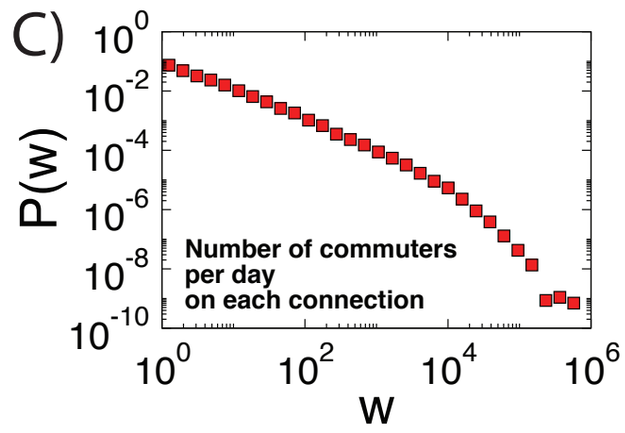
Short Range Mobility

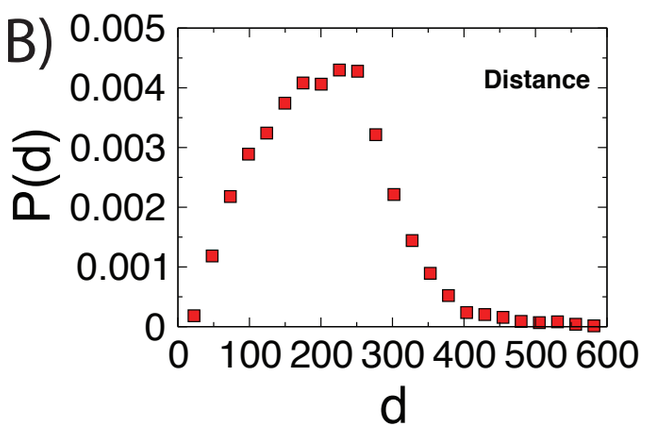
PNAS 106, 21484 (2009)

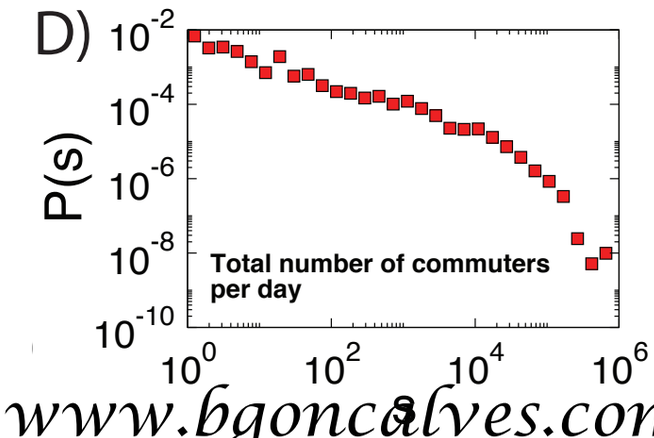
Table 1
Commuting networks in each continent. Number of countries (N), number of administrative units (V) and inter-links between them (E) are summarized.

Continent	N	V	E
Europe	17	65,880	4,490,650
North America	2	6986	182,255
Latin America	5	4301	102,117
Asia	4	4355	380,385
Oceania	2	746	30,679
Total	30	82,268	5,186,186



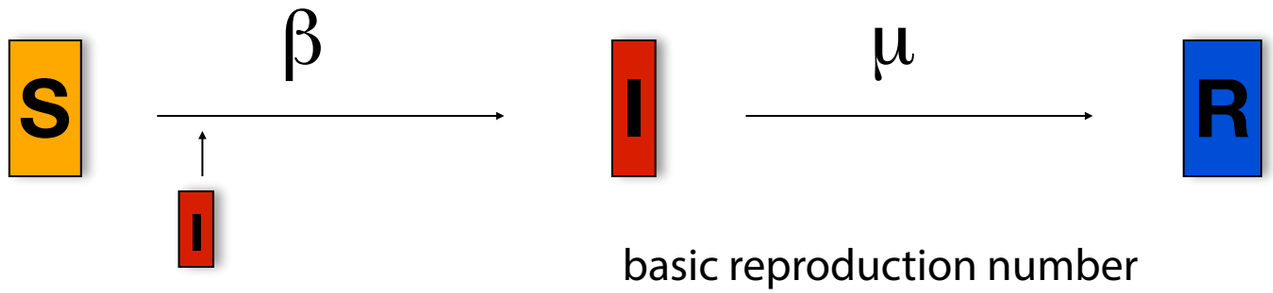






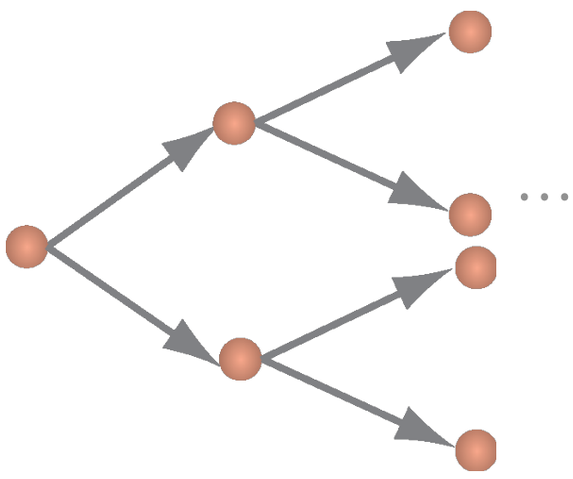
Intra-population: The SIR model

PNAS 106, 21484 (2009)



basic reproduction number

$R_0 > 1$ ⟹ exponential growth

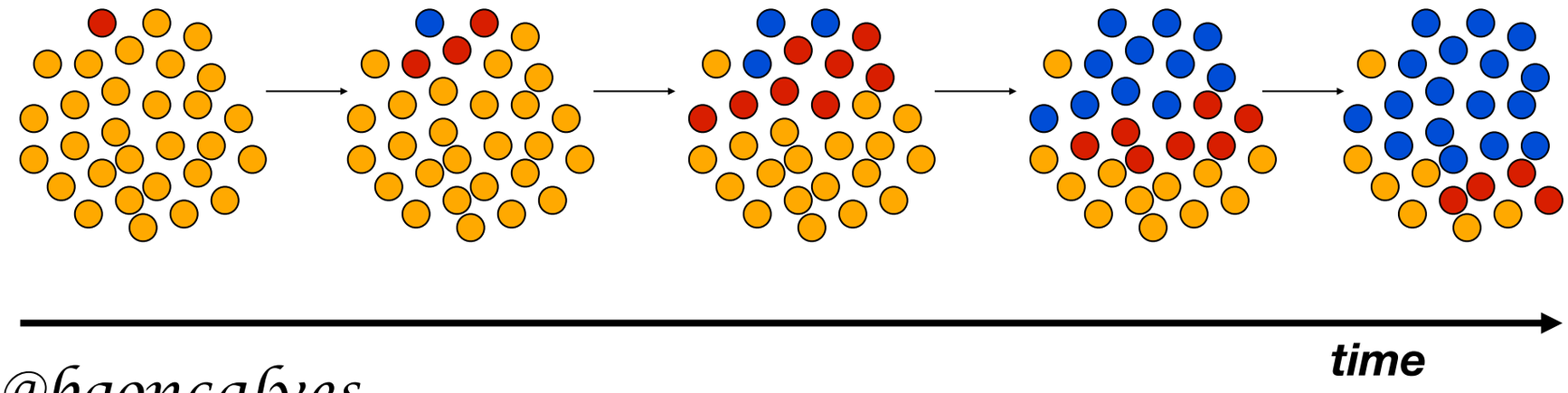


Stochastic coupling terms

$$S_{j,t+\Delta t} = S_{j,t} - Binom_j(S_{j,t}, \beta \Delta t\, I_{j,t}/N) + \Omega_j(S)$$

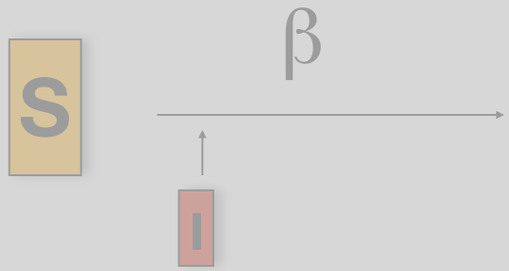
$$I_{j,t+\Delta t} = I_{j,t} + Binom_j(S_{j,t}, \beta \Delta t\, I_{j,t}/N) - Binom_j(I_{j,t}, \mu \Delta t) + \Omega_j(I)$$

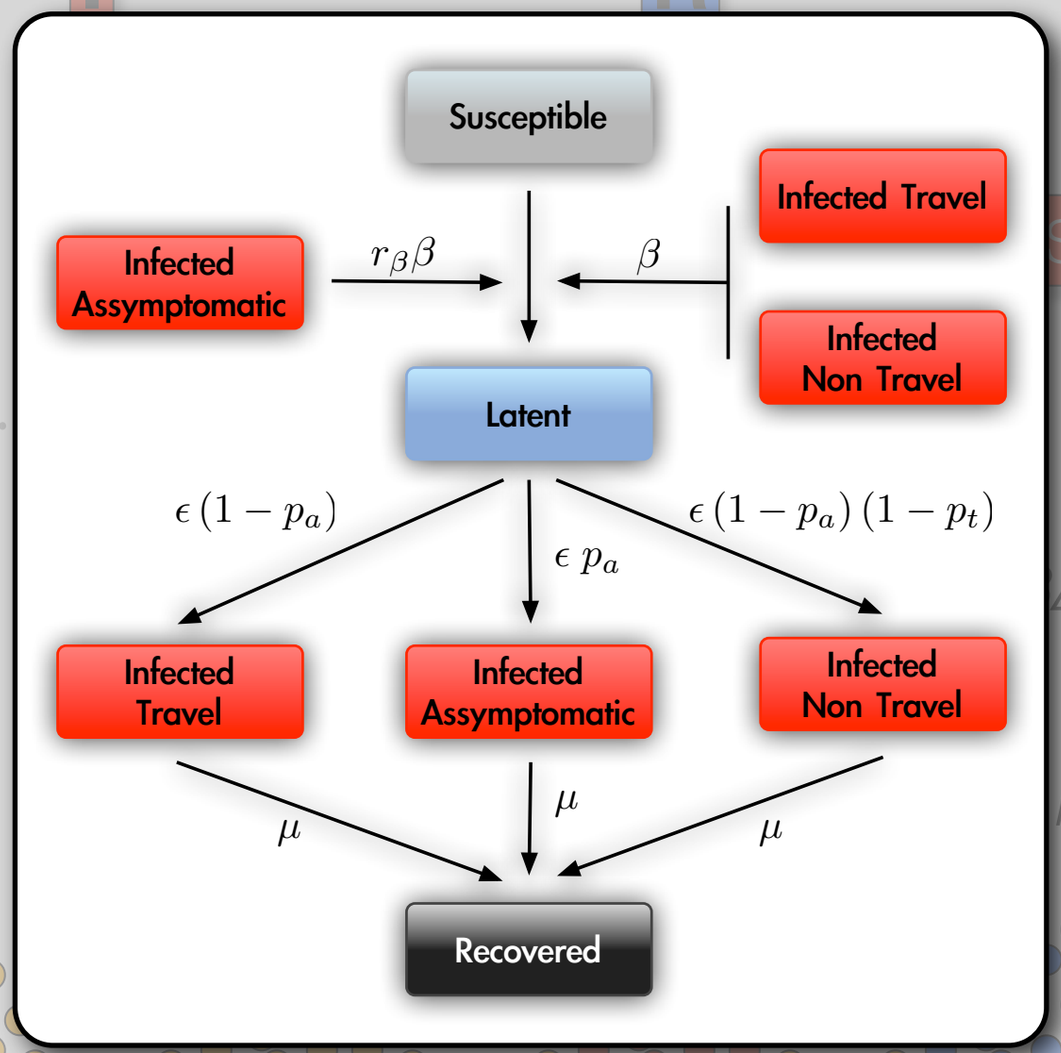
$$R_{j,t+\Delta t} = R_{j,t} + Binom_j(I_{j,t}, \mu \Delta t) + \Omega_j(R)$$



Intra-population: The SIR model

PNAS 106, 21484 (2009)



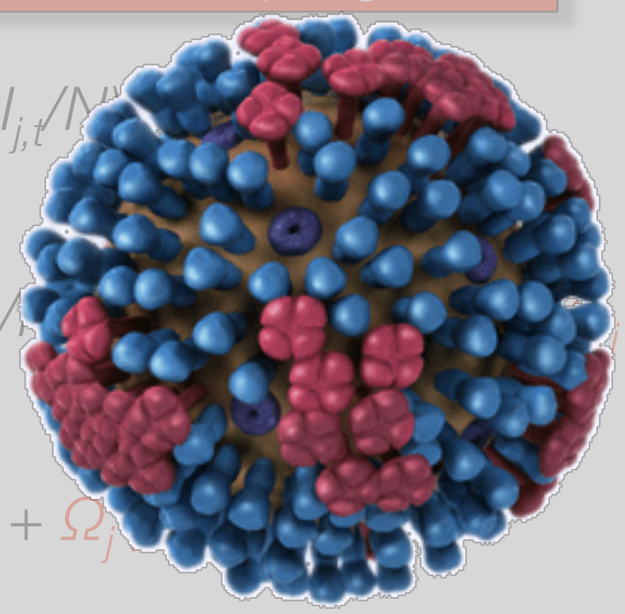


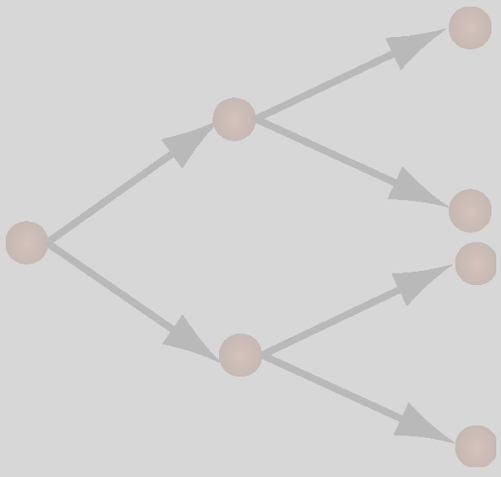
Stochastic coupling terms

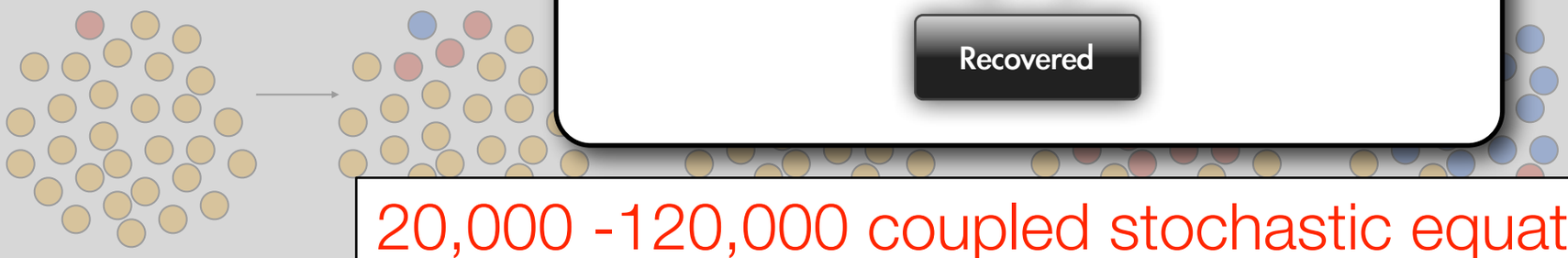
$\beta \Delta t\ I_{j,t}/N$

$\Delta t\ I_{j,t}/$ (I)

$\mu \Delta t) + \Omega_j$



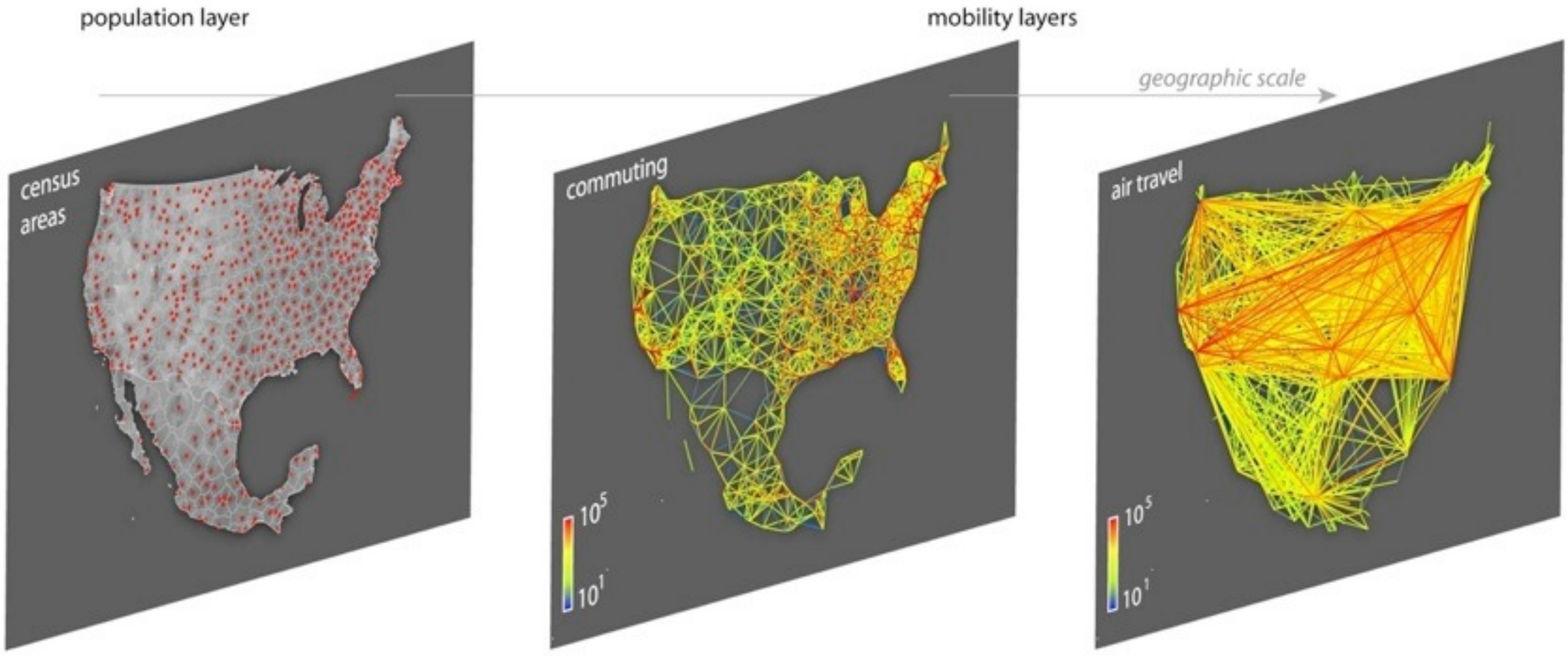


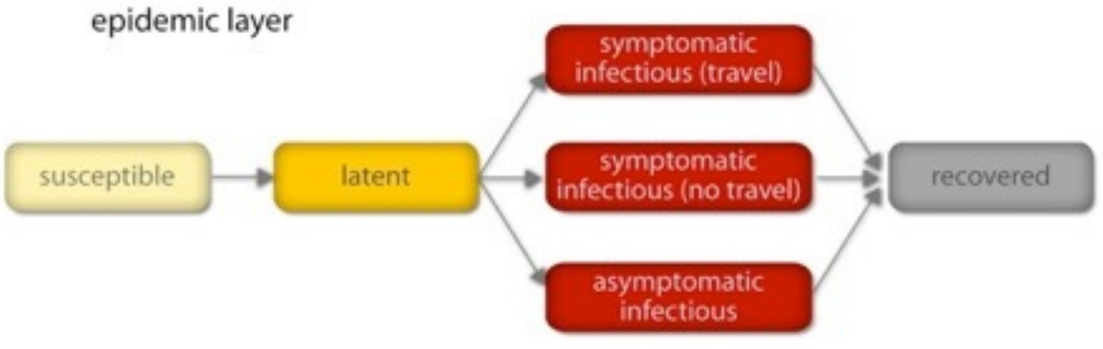


20,000 -120,000 coupled stochastic equations

time

Global Epidemic and Mobility Modeller Platform





Parameter	Value	Description
β	from R_0	transmission probability
ε^{-1}	1.9 [1.1-2.5] d	average latency period
μ^{-1}	3 [3-5] d	average infectious period
p_t	50%	probability of traveling for infectious individuals
p_a	33%	probability of being asymptomatic
r_β	50%	relative infectiousness of asymptomatic infectious individuals

Invasion Tree

GLEaMviz.org

GLEAMVIZ

GLEaMViz

Global Epidemic and Mobility Modeler Visualization

May 3, 2009

MoBS