Synthetic Populations and Ecosystems of the World
Version 1.2.0

Bill Eddy[1], Shannon Gallagher[1], Lee Richardson[1], and Sam Ventura[1]

[1]Department of Statistics, Carnegie Mellon University
[2]MIDAS Informatics Services Group

October 7, 2016

Contents

1 Introduction

This document provides specific details on how we generate our synthetic ecosystems. As the data and methodology are updated, this document will be updated to reflect the changes. The software used to generate our ecosystems is in the R package, `spew`. The release of ecosystems will be synchoronized with versions of the software. Thus, our version 1.1.0 ecosystems were created with `spew` version 1.1.0. One caveat is that lots of code is written to collect and transform the data for `spew`. This is available online at:

`https://github.com/leerichardson/spew_olympus`

In future versions, we'll consolodate these two code bases into one location.

In this document, the term SPEW, or `spew`, will be used often. By SPEW, we mean the final synthetic ecosystems. By `spew`, we mean the software used to generate the ecosystems.

The current version of SPEW has 124 total synthetic ecosystems, which can be grouped into three categories. We have 52 **United States** ecosystems, one for each state. The major addition to the United States ecosystems is that locations are samlpled using road data. Second, we have 71 ecosystems using **IPUMS** data. This is more than previous versions of SPEW, as various speed, storage, and data issues have been resolved. Finally, we have one **custom** synthetic ecosystem from Canada, which is the same as in previous versions.

The R package `spew` is designed to abstract the process of raw data sources to final ecosystem. For example, different data sources were used for the United States, Canada, and IPUMS ecosystems, but `spew` created them all. This abstraction enhances our ecosystems in many ways. For one, `spew` requires standardized data. Standardization specifies the exact format of input data to be usable by `spew`. This expedites the process of moving from raw data to final ecosystem. In addition, `spew` makes our ecosytems more reliable. Since `spew` generates each ecosystem, fixing bugs in one place propogates to all other ecosystems. Finally, `spew` is flexible and provides a straighforward way to add new elements to the ecosystem (eg: restaraunts, hospitals, etc...), as new data and methodology is discovered.

For interested users, we provide the `spew` source code. This allows users to track down the exact details in generation of their ecosystems. In the future, we hope `spew` will be independent of us, and will be helpful to parties interested in their own synthetic ecosystems. Our software is publically accesible on Github at:

`https://github.com/leerichardson/spew`

2 Data Sources

This section describes the data sources we used used to generate our ecosystems. Each ecosystem represents a **region**. Typically, each region is partitioned into **sub regions**, the union of which makes up a region. Generating an ecosystem for a region `spew` requires three data sources:

1. **Population counts**: How many people are in the region. This tells us the number of either people or households is each sub-region.
2. **Geographies**: Geographic boundaries of the regions, typically a shapefile. [1]. Each polygon in the shapefile corresponds to a sub-region, and these must align with the population counts.
3. **Microdata**: Individual records. This is typically a data-set in which each row is an individual, and each column is a characteristic of that individual. Some example columns are age, sex, and income.

More data can be included (eg: schools and workplaces) when available. But all `spew` ecosystems use these three data sources.

In the introduction, we mentioned that this version of SPEW has three different data groups: United States, IPUMS, and Canada. The following sections detail the data sources used to generate ecosystems for each group.

2.1 United States

The United States synthetic ecosystem is our most detailed. Data comes largely from the US census, with supplementary data on schools and workplaces. Because the US census data is so detailed, ecosystems are created at the tract level. This means sub-region ecosystems have population sizes ranging from $500 - 10,000$. Users can aggregate these tracts together to construct larger synthetic ecosystems.

2.1.1 Population Counts

American Community Survey Summary Tables (2006-2010)

- Available at: `https://www.census.gov/programs-surveys/acs/technical-documentation/summary-file-documentation.html`
- Total number of households by Tract

[1] From ArcGis: "A shapefile is an Esri vector data storage format for storing the location, shape, and attributes of geographic features"

2.1.2 Geographies

US Census Topologically Integrated Geographic Encoding and Referencing (TIGER) Shapefiles (2010)

- Available at `https://www.census.gov/geo/maps-data/data/tiger.html`
- Geographies at the Census tract level [2]
- Roads at the County level

2.1.3 Microdata

1-Year American Community Survey (2013)

- Available at: `http://www2.census.gov/acs2013_1yr/pums/`
- Corresponds to 2010 defined Census geography
- Both household and people populations
- See appendix for variables used

2.1.4 Schools

National Center for Education Statistics School Data (2011-2013)

- Available at: `http://nces.ed.gov/ccd/elsi/tableGenerator.aspx`
- Public Schools (2013) have latitude/longitude information. Private schools (2011) only have county level information.
- More information at: `http://data.olympus.psc.edu/syneco/west/north_america/united_states/schools/`

2.1.5 Workplaces

ESRI Workplace Data (2009)

- Available with a license from ESRI
- ID, employee counts, and county of different businesses in the US
- More information at `http://data.olympus.psc.edu/syneco/west/north_america/united_states/workplaces/`

[2] See: `http://www2.census.gov/geo/pdfs/reference/geodiagram.pdf` for an image of the Census Geographic Hierarchy

2.2 IPUMS

IPUMS synthetic ecosystems combine three global data sources. The most important of which is the Minnesota Population Center's International Public Microdata Sample (IPUMS). Not only does IPUMS provide microdata, but also corresponding shapefiles, which allow us to sample the sub-region synthetic ecosystems more accurately. Population counts come from another global source, Geohive. In this version, we omit data collected from the GADM project, although we hope to incorporate this greater geographic detail in future versions.

2.2.1 Population Counts

Geohive

- Available at: `http://www.geohive.com/`
- Compiles population statistics from various statistical agencies throughout the world (The list can be seen here `http://www.geohive.com/earth/statorgz.aspx`)
- Population counts from over 150 countries at various administrative levels.

2.2.2 Geography

IPUMS Shapefiles

- Available at: `https://international.ipums.org/international/`
- Shapefiles corresponding to IPUMS microdata.
- Available at administrative level 1

2.2.3 Microdata

International Public Use Microdata Sample (IPUMS)

- Available at: `https://international.ipums.org/international/`
- Microdata from 82 different countries
- See the appendix for the variables used

2.3 Canada

Our Canadian synthetic ecosystem is similar to the United States, since it's also at the tract level. However, detailed data coming from roads, schools, and workplaces is not included. Our Canadian inpur data comes from Statistics Canada.

2.3.1 Population Counts

Statistics Canada Census Profile (2011)

- Available at: `https://www12.statcan.gc.ca/census-recensement/2011/dp-pd/prof/details/download-telecharger/comprehensive/comp-csv-tab-dwnld-tlchrgr.cfm?Lang=E` specifying the Census Tracts option.
- Total population for every tract

2.3.2 Geographies

Statistics Canada Boundary File (2011)

- Available at: `https://www12.statcan.gc.ca/census-recensement/2011/geo/bound-limit/bound-limit-2011-eng.cfm` specifying the English, ArcGIS, and Census tract option.
- Geographies at the Census Tract level

2.3.3 Microdata

Public Use Microdata File (2011)

- Obtained with special permissions from Statistics Canada
- Variables defined in the appendix

3 Methods

This section describes the methodology used to generate SPEW ecosystems. The first step is always to standardize the data sources. Our standard for `spew` input data is:

1. There must be population counts, geographies, and microdata.
2. The **sub-regions** in each data source must have identical names.

For example, Canada uses three different sources: Population counts, geographies, and microdata. Each one of these sources has a geographic division of Canada. Before `spew` can run, each source must have non contradicting geographic divisions. Our package `spew` includes a series of checks to verify their data is standardized.

At a high level, `spew` splits a **region** into **sub-regions**, then generates an ecosystem for each sub-region, see Algorithm 1 for an overview. After `spew` recognizes the input data is valid, it splits the regions into sub regions (eg: Census Tracts, states, provinces), the union of which is the entire region. Then `spew` generates a synthetic ecosystem for each sub-region. Finally, the sub-region synthetic ecosystems are put back together to form a region synthetic ecosystem.

input : Population counts, geographies, microdata, other sources
1. Check for required data sources
2. Check that data sources have aligned geographic divisions

for *Every Sub region* **do**
    1. Sample Households
    2. Sample Locations
    3. Sample People
    4. Assign other data if available (schools, workplaces, etc...)

end
output: Synthetic Ecosystem

Algorithm 1: Pseudocode for spew synthetic ecosystems

A piece of geographic information we often is the Public Use Microdata Area (**PUMA**). This is used when we have data-sources at different levels. A **PUMA** groups together various sub-regions. For example, the United States provides microdata at the PUMA level. Each PUMA is made up of 30-60 tracts, or around 100,000 people. To generate an ecosystem for a particular tract, we use microdata corresponding to whichever PUMA the tract is a part of.

The next sections provide detials on each component of the `spew` process. We'll explain what we've done, and what we're planning to include in future versions.

3.1 Sample Households

From Algorithm 1, the step of `spew` is to sample households. This is because microdata typically comes in two pieces: Household and Individual data. Households are sampled first so we can link individual people to their respective household, verify each household has the correct number of people. Sampling entire household units also avoids issues such as generating a household full of 6 year olds.

This version of `spew` uses a basic approach: Simple Random Sampling. In addition to SRS, we are hoping to include more sophisticated sampling methods in future versions. Some of these include:

- Moment Matching (MM)
- Iterative Proportional Fitting (IPF)

3.1.1 Simple Random Sampling

The rationale for using SRS is that it was the easiest to implement, and allowed us to focus on other issues (eg: collecting and formatting the data) to put together populations. In a sense, we used SRS as a placeholder while putting together other components of SPEW, and always intented to revisit it later and think about a more sophisticated sampling approach. The idea behind SRS is as follows. Let's say we want to generate x households and y people for a given region. First, we uniformly sample x records with replacement from the household microdata, which produces x synthetic households. For the synthetic people, we use the ID variable linking the household and person level microdata, and the people whose households were sampled make up the synthetic people.

A few things to point out here. First, since we are sampling households with replacement, identical households can be seen inside the population. This also implies that people may be duplicated within the population. However, usually the records aren't exactly identical, since the sampling of locations also assigns a latitude and longitude to each household. Next, we point out that the data sometimes comes with the total number of households, and other times as the total number of people. If we know the total number of households to sample, we use this. If we only know the total number of people, we need to estimate the average household size, and usually divide the number of people by this to obtain x. Here we see the importance of the Household Size variable, which we will re-visit in more detail when describing MM.

3.1.2 Moment Matching

The second approach we propose is denoted as Moment Matching (MM). We initially used this method as a way to match the household sizes of West African countries. For example, we may have had microdata for a given country, but the average household size in the microdata does not match the average household size known from other sources. This method assigns weights to household

records such that sampling according to these weights gives a synthetic population with the appropriate moments.

First, let's set the notation. Assume we have access to the first moment of region R's household size, but the distribution of household size is unknown. Denote the first moment of the household size of region R by M_R. Next, let's say that the microdata has N distinct household sizes $\mathbf{n} = (n_1, \ldots, n_N)^T$, where each n_i indicates that there is at least one household within the PUMS of size n_i. Denote weights by $\mathbf{x} = (x_1, \ldots, x_N)^T$. Let $x_i \geq 0$ for $i = 1, \ldots, N$, $\sum_{i=1}^N x_i = 1$, and $\sum_{i=1}^N x_i n_i = M_R$ denote the constraints. This formulation alone has infinitely many solutions. To settle on a particular value, we form a quadratic program and minimize

$$f(x) = \frac{1}{2}||\mathbf{x}||_2^2,$$

where $||\cdot||_2$ is the L^2 norm. We note that one reason for this proposed method is due to is simplicity and computational tractability.

Our objective function f and the constraints h_i for $i = 1, \ldots N$, ℓ_1 and ℓ_2 are as follows,

$$\begin{aligned} f(\mathbf{x}) &= \frac{1}{2}||\mathbf{x}||^2 \\ h_i(\mathbf{x}) &= -x_i \leq 0 \text{ for } i = 1, \ldots, N \\ \ell_1(\mathbf{x}) &= \sum_{i=1}^N x_i - 1 = 0 \\ \ell_2(\mathbf{x}) &= \sum_{i=1}^N n_i x_i - M_R = 0 \end{aligned}$$

$$\mathbf{A}\mathbf{x} = \mathbf{b}, \text{ where} \tag{1}$$

$$\mathbf{A} = \begin{bmatrix} 1 & \ldots & 1 \\ n_1 & \ldots & n_N \end{bmatrix}, \mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}, \mathbf{b} = \begin{bmatrix} 1 \\ M_C \end{bmatrix}$$

$$\min_{\mathbf{x} \in \mathbb{R}^N} \frac{1}{2}\mathbf{x}^T\mathbf{x}, \text{ with } \mathbf{A}\mathbf{x} = \mathbf{b}, \mathbf{x} \geq 0, \tag{2}$$

which is in the form of a quadratic program. We use the R Package `quadprog` to solve.

We hope to include MM in situations when matching a single variable is critical.

3.1.3 Iterative Proportional Fitting

Iterative proportional fitting (IPF) [2] [3] estimates individual cell values of a contingency table with known marginal totals. Beckmam [1] created a sampling scheme based on IPF to generate synthetic populations for transportation simulations. The IPF sampling scheme is widely adopted, for instance, RTI [4] used IPF to generate their United States synthetic population. The IPF sampling scheme is a two step method:

1. Generate contingency table of population demographics with IPF
2. Sample households using contingency table probabilities as weights

Step 1 estimates an m-dimensional contingency table using marginal totals. For our purposes, each dimension represents a **demographic**. Each demographic is made up of **categories**. An example demographic is gender, which has two categories: male and female. Following [1], our notation is:

- n : total number of observations in the table
- m : number of demographics (dimensions of the contingency table)
- n_j : number of categories for the j^{th} demographic. $j = 1, 2, ..., m$
- i_j : value of the j^{th} demographic. $i_j = 1, 2, ..., n_j$
- $p_{i_1,i_2,...,i_m} = \frac{n_{i_1,i_2,...,i_m}}{n}$: proportion of observations in an individual cell
- $T_k^{(j)}$: marginal totals for k^{th} category of j^{th} demographic. $k = 1, 2, ..., n_j$

So for all j, we have:

$$n = \sum_{k=1}^{n_j} T_k^{(j)} \tag{3}$$

IPF **updates** the contingency table until the marginal totals are within a **tolerance** of the known marginals. Each update is called an **iteration**. Let $p^{(t)}_{i_1,i_2,...,i_m}$ denote the estimation of the cell $(i_1, i_2, ..., i_m)$ during iteration t.
The initial contingency table for IPF is:

$$p^{(0)}_{i_1,1_2,...i_m} = p_{i_1,i_2,...,i_m} \tag{4}$$

In practice, the raw counts from PUMS are used, without incorporating the marginals. For instance if the PUMS data has four males heading three person household aged 30-34, earning $ 100,000 dollars a year, then:

$$p_{i_{gender},i_{hhsize},i_{age},i_{income}} = 4$$

Each iteration goes through each margin, and updates the estimated proportion $\hat{p}_{i_1,i_2,\ldots,i_m}$. Specifically, for each of the j margins, we update the k^{th} category by:

$$p^{(t)}_{i_1,i_2,\ldots,i_j=k,\ldots,i_m} = p^{(t-1)}_{i_1,i_2,\ldots,i_j=k,\ldots,i_m} \frac{T^{(j)}_k/n}{\sum_{i=1}^{n_1}\sum_{i=1}^{n_2}\cdots\sum_{i=1}^{n_m} p^{(t-1)}_{i_1,i_2,\ldots,i_j=k,\ldots,i_m}} \quad (5)$$

We continue iterations until the tolerance is reached. Beckman [1] reports that the procedure typically converges in 10-20 iterations.

Step 2 samples households in proportion to the probabilities in the contingency table. The probabilties determine how many households of each demographic combination should be sampled. For each demographic combination, probabilities are assigned to each PUMS household, based on how "close" they are, in terms of demographics. The closeness of each household is determined by the distance function:

$$D(p,c) = w_p \prod_{i\in J}(1 - |\frac{d^p_i - d^c_i}{r_i}|^k) \times \prod_{i\notin J}(1 - (\delta(d^p_i, d^c_i)) \quad (6)$$

With the following notation:

- p : household from PUMS
- c : cell from contintency table
- J : set of ordinal variables. $\notin J$ is the set of categorical variables.
- d^p_i : value of i^{th} demographic for household p
- d^c_i : value of i^{th} demographic of cell type c
- r_i : range of demographic i in the PUMS
- w_p : weight from household p
- $\delta(d^p_i, d^c_i) = \begin{cases} \alpha & d^p_i = d^c_i \\ 1-\alpha & d^p_i \neq d^c_i \end{cases}$

Note that when $\alpha = 0$ and $k \to 0$, $D(p,c)$ is a 0-1 loss function, which means we're only sampling the subset of exact matching characteristics.

Finally, each record is sampled according to the following weights:

$$\mathbb{P}(\texttt{Select Household p}) = \frac{D(p,c)}{\sum_j D(j,c)} \quad (7)$$

3.2 Sample Locations

Once households are sampled, we need to assign them a location. This is why the geographies of a region are a required data source. In previous versions, we always assigned locations uniformly. Similar to household sampling, this was used as placeholder until more realistic method could be implemented. In this version, we've incorporated road based sampling for United States populations. The goal of all our approaches, of course, is to make our populations as realistic as possible.

3.2.1 Uniform Location Sampling

Given the polygon of sub region, we'd assign a household to a random point in the polygon. Each point is given equal probability to contain the household. To implement uniform sampling, we use the `spsample` from the `sp` package.

One note here is that this function was particular slow for large regions, such as the regions of China and India, where we needed to assign tens of millions of locations. In some situations, we would only take 100,000 samples, resample tens of millions of these, and add noise to the results. This was strictly to increase the performance of `spew`, not for accuracy purposes.

3.2.2 Road Location Sampling

The United States road data came at the county level, whereas the polygons come at the tract level. To resolve this difference, we use the `gIntersection` function from the `rgeos` package. This function takes two shapes (in our case, the roads were lines, and the tracts were polygons) and returns only the geographies in one shape. In our case, this function gave us all the roads within a given tract.

With the tract level roads, we once again used the `spsample` function to assign locations. In this case, `spsample` sampled from the lines, and added standard normal random noise, to ensure households weren't exactly in the streets

While the `spsample` has proved very useful, we've ran into consistency and speed issues. In future versions of `spew`, we will implement our own sampling methods, in ways that will speed up the process and make it more reliable.

3.3 Sample People

Now that we have households and locations assigned, the next step is to sample people for each household. Note that the household and people level microdata always contain an identifier which can be used to link them. Using this variable, to assign people to households we perform a left join [3] operation using the sampled households and people microdata. This means that our synthetic people

[3] Left Join comes from SQL Joins, which combine records for two tables. Left Join's return all rows from the table 1 along with matching rows from table 2

are simply the people corresponding to the households which were already sampled. We take this approach to ensure the consistency between our household and person ecosystems. Note that the location attached to each person will be identical to the location of their household.

Like our other sampling methods, we believe there is much room to improve here. In particular, we see that that both persons and households are frequently duplicated. To get around this for people, we could estimate the conditional density of people corresponding to household sizes, and sample from this. We hope to include methods like this in future versions of the program.

3.4 School Assignments

Note that for schools, we only have data corresponding to the United States. For that reason, we will explain our method for assigning schools in the US here. We have access to school data that includes:

- enrollment totals
- latitude and longitude for public schools
- county location for private schools.

In addition, the synthetic people have features for school enrolment (`SCH`), grade level (`SCHG`), and age (`AGEP`).Using these three variables, we can find which people need to be assigned to a public school or private school. Note that our school dataset only contains elementary, middle, and high schools, so we are not assigning preschool, college, or professional schools.

The algorithm is as follows. For each person, determine whether the person should be sent to school. Use the county of the person to identify the schools (Note that if there are no schools in this particular, county, we use the entire state). Then using `SCH`, determine whether we should use public or private schools. Using the `SCHG` variable, further subset the schools to find those with the proper grade. For private schools, weight the subsetted schools by the enrollment total, assigning more weight to schools with more enrolled students. For public schools, weight the subsetted schools by both the enrollment total and the haversine distance [4]between the person and the schools. Schools with more people are given more weight as are schools that are physically closer to the person. Finally, sample a school with the weights calculated above. In the case where there are no schools in the county, subset the schools only by state.

In this way, students will not cross county borders when being assigned a school. The enrollment totals and distances are used to assign probabilities to various schools, as opposed to hard restrictions. The process is summarized in Algorithm 2.

[4]Distance between two points on a sphere using longitude and latitude

```
input  : synthetic people, schools data
for Every person do
    Determine whether child should be sent to school if no school then
        school ID ← NA
    else
        if coordinates of school exist then
            weight schools based on distance from child and enrollment
            totals
        else
            weight schools based on enrollment totals
        end
        sample school from county (use state if there are no schools in a
        county) based on weights school ID ← sampled school ID
    end
end
output: School IDs
```

Algorithm 2: Pseudo code for generating schools

3.5 Workplace Assignments

Workplaces are assigned im a similar way as schools. The synthetic people have a feature: employment status recode (`ESR`), which tells us if the person is working. The workplace data we have contains the number of employees as well as the state and county of the workplace. We first subset the workplaces to match the state and county of the person in question. Then we weight the workplaces by the number of employees, with more employees given more weight. Finally, we sample a workplace for each person. The pseudo code for this assignment is shown in Algorithm 3.

```
input  : synthetic people, workplace data
for Every person do
    Determine whether person is in workforce
    if no employment then
        work ID ← NA
    else
        weight workplaces in county based on number of employees
        sample a workplace based on weights
        workplace ID ← sampled workplace ID
    end
end
output: Workplace IDs
```

Algorithm 3: Pseudo code for generating workplaces

4 Data Organization

SPEW synthetic ecosystems are available online at:

`http://data.olympus.psc.edu/syneco/`.

Synthetic ecosystems are stored in a geographic hierarchy, based on the hierarchy of the United Nations Statistics division. This hierarchy is available at:

`http://unstats.un.org/unsd/methods/m49/m49alpha.htm`).

The lowest level of our geographic hierarchy is a country. Each country has a corresponding ISO3 code, invaluable for matching data accross sources. Sometimes, we have data at lower levels than country. In this case, we extend the geographic hierarchy to include data at lower levels within the country. An example is the United States, where we have data at the state level, so we include a state level in the hierarchy, underneath the US country.

The hierarchy is as follows:

1. Region
2. Sub-region
3. Country
4. Lower level data (if available)

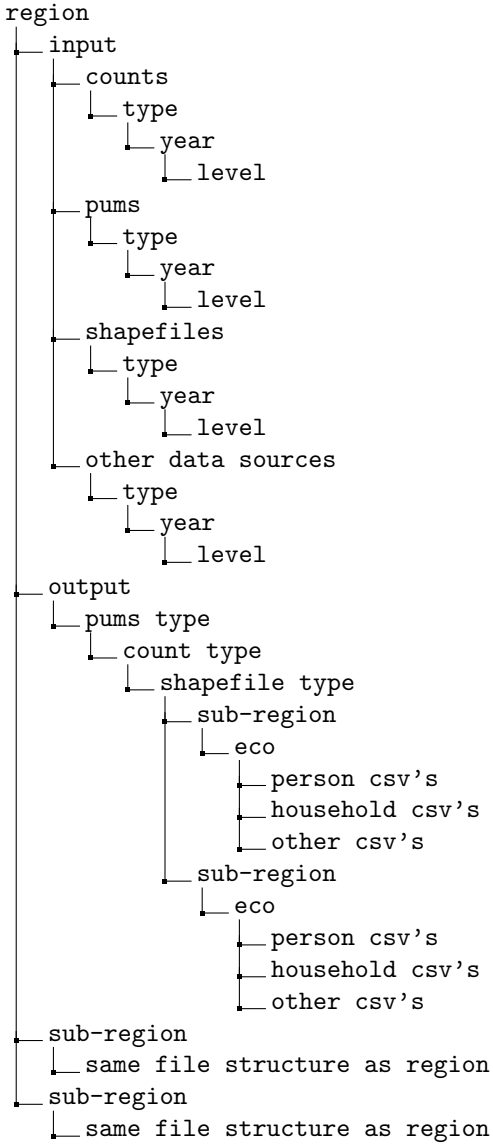
Below are example file-paths for China and Califoria, within the hierarchy:

- `spew_1.2.0/asia/eastern_asia/chn`
- `spew_1.2.0/americas/northern_america/usa/06`

4.1 Directory Structure

Each synthetic ecosystem is contained within its own directory. In this directory, the input data goes in the `input/` folder, and the output data in the `output/` folder. The `input/` directory organizes input data by type (counts, shapefiles, etc), year, and geographic level. The geographic levels won't necesarily match accross different data types, as they come from entirely different sources.

The specific directory structure is as follows. In this case, the **region** is a country, and the **sub-region** at the bottom represents a state.



4.2 Diagnostics

We're currently building a suite of diagnostic checks, which run automatically for each ecosystems. These will be included in future versions.

References

[1] Richard J Beckman, Keith A Baggerly, and Michael D McKay. "Creating synthetic baseline populations". In: *Transportation Research Part A: Policy and Practice* 30.6 (1996), pp. 415–429.

[2] W Edwards Deming and Frederick F Stephan. "On a least squares adjustment of a sampled frequency table when the expected marginal totals are known". In: *The Annals of Mathematical Statistics* 11.4 (1940), pp. 427–444.

[3] Stephen E Fienberg. "An iterative procedure for estimation in contingency tables". In: *The Annals of Mathematical Statistics* (1970), pp. 907–917.

[4] William D Wheaton et al. "Synthesized population databases: A US geospatial database for agent-based models". In: *Methods report (RTI Press)* 2009.10 (2009), p. 905.

A Codebook

We've generated ecosystems using three different sources of microdata. The source of microdata establishes the columns used for our ecosystems. The following links provide the codebooks for each source of microdata:

A.1 United States: American Community Survey

See https://usa.ipums.org/usa/resources/codebooks/DataDict2013.pdf

A.2 Canada: Public Use Microdata File

See http://data.olympus.psc.edu/syneco/west/north_america/canada/docs/pums/2011%20PUMF_FMGD/Hierarchical%20file/English/Documentation%20and%20user%20guide/2011%20NHS%20Hierarchical%20PUMF%20User%20Guide.pdf

A.3 IPUMS: International Public Use Microdata Sample

See https://usa.ipums.org/usa/resources/codebooks/DataDict0610.pdf

B SPEW Process Documentation

This section describes the steps to generate our ecosystems on the Olympus supercomputer. In paricular, we will walk through how to load `spew` on Olympus, how we're running it, and how `spew` read data and turns them into ecosystems. The goal is if we were all hit by busses, there would be enough details for others to smoothly generate new ecosystems. Hopefully, this makes it easier to integrate with other software, and will give enough detail so others can understand and help make the process more efficient.

B.1 Installing SPEW on Olympus

To install `spew` on Olympus, you must be using R version 3.2.1. [5]. It may work with older versions, but to be safe we have been using 3.2.1. Load this version of R with:

```
module load r/3.2.1
```

Next, we need the `spew` R package. This is located on Github at:

`https://github.com/leerichardson/spew`

We're installing `spew` in our personal R libraries on Olympus, opposed to being maintained by system administrators. The reason is that `spew` is under development, and we want to be able to make changes and check them on Olympus quickly. Our process is to test new features locally, and when they're ready, update the package in our personal libraries on Olympus. The following commands load `spew` in your personal library:

```
library(devtools)
library(httr)
library(curl)
personal_ind <- grep("leerich", .libPaths())
personal_lib <- .libPaths()[personal_ind]
remove.packages(pkgs = "spew", lib = personal_lib)
withr::with_libpaths(new = personal_lib, install_github("leerichardson/spew"))
library(spew)
```

This will install the most recent version of `spew` in your personal library on Olympus. Note that if you're installing `spew` for the first time, then you will only need to omit line 4, which removes an older version of `spew` from your personal R libraries.

B.2 Calling SPEW

Once `spew` is installed, you can use it on Olympus. This section explains how we run `spew`. However, we often run it interactively to test new features. The

[5] R/3.3.1 is probably fine, but this version of ecosystems was generated using R/3.2.1

code to call spew is on Olympus at:

```
/mnt/beegfs1/data/shared_group_data/syneco/spew_olympus
```

The inputs for our `spew` are located in a lookup table. The file that creates the lookup table is called `spew_olympus/create_lookup.R`, and the output of this is a lookup table saved at `input/spew_lookup.csv`. Our current process is that whenever we obtain new data, we update the `spew_olympus/create_lookup.R` file so that lookup table has correct inputs. In the future, we hope to get away from the lookup table, and generate the inputs from the file structure.

We call spew with `spew_olympus/run_spew.sh`. This shell script takes one argument, which is an R query in the form:

```
run_spew.sh which(lookup$[geo_level] == ‘‘[geo_level_name]’’)
```

Where `geo_level` is either `hemisphere`, `continent`, `country`, or `state_dirs`, and `geo_level_name` is the corresponding name. For example, to run the uruguay, one would use:

```
run_spew.sh which(lookup$country == ‘‘uruguay’’)
```

In Lee's home directory, there is a file `/home/leerich/call_spew.sh`) which takes two inputs, `geo_level` and `geo_level_name`, and constructs the query for you. So you can run with `call_spew.sh country uruguay`.

The purpose of the query argument is that we are using it to subset the lookup table and obtain the appropriate inputs for a given location. Specifically, `spew_olympus/run_spew.sh` calls the file `spew_olympus/subset_lookup.R`, which takes the query and subsets the lookup table to create a file `input/tmp_subset_lookup.csv`.

Once the temporary lookup table is created, `spew_olympus/run_spew.sh` loops through each row and submits a job, using the columns as the parameters. The submission script is `spew_olympus/spew.sh`.

In addition, note that `spew_olympus/run_spew.sh` does two more takss things before submission. First, it removes pre-existing outputs in the output directory. This means that every directory starting with either output or eco will be removed from the output folder. Next, it construct the path for the logfile to be written to corresponding directory, and creates a `logfiles/` directory to store this inside.

The final call of `spew_olympus/run_spew.sh` is to `spew_olympus/spew.sh`, seen here:

```
qsub -o $output_log -e $output_log -v data_group=$data_group,
input_dir=$input_dir,output_dir=$output_dir,
convert_count=$convert_count,pop_table=$pop_table,
shapefile=$shapefile,pums_h=$pums_h,pums_p=$pums_p
-N $state_dirs
/mnt/beegfs1/data/shared_group_data/syneco/spew_olympus/spew.sh
```

Now we're running `spew_olympus/spew.sh`. Since this is called using qsub, the first lines are PBS commands. After verifying that the pandoc, io, and r/3.2.1 modules are loaded on the compute node for this run of spew, spew.sh calls the final function, `spew_olympus/run_spew.R`:

```
Rscript /mnt/beegfs1/data/shared_group_data/syneco/spew_olympus/run_spew.R
${data_group} ${input_dir} ${output_dir} ${convert_count} ${pop_table}
${shapefile} ${pums_h} ${pums_p}
```

The `spew_olympus/run_spew.R` script prepares the R session, then calls the `spew` function `generate_spew`. By preparing the session, we mean it loads in the packages, parses the inputs, and determines the variables to generate. This is part is inconsistent: some of the inputs parameters come from the table, while others are generated in this file. We're hoiping to synchronize this in the future.

The final line of `spew_olympus/run_spew.R` calls the function `generate_spew` using all of the inputs generated from both the lookup table and this file, and we are now running SPEW with these inputs to generate the synthetic ecosystems.

B.3 How SPEW works

The previous section describes how to run `spew` on olympus. Here, we explain how `spew` converts input data into ecosystems. `spew` is split into three main functions:

1. **Read**: Loads input data
2. **Format**: Verifies data is in standard form
3. **Make**: Converts input data into synthetic ecosystem. [6]

The `generate_spew` function is wrapper function which calls, `read_data`, `format_data`, and `make_data` in sequence.

`read_data` function takes a base directory, an R list containing the folder names, and a data group. It then uses the data group variable to determine how to read in the data. Currently, we have functions to read in USA, IPUMS, and Custom data. The USA and IPUMS data groups rely on the input data having an identical file-structure, whereas the custom data group provides filepaths for each source. As output, `read_data` provides a list with an element for each data-type. We require that the list contains shapefile, pop_table, and pums elements. It can also include other optional inputs, such as schools, workplaces, etc...

Next, `format_data` function takes the data list and data group, and returns the data list in standard form. The key feature of properly formatted data is an element in the data list called `data_list$pop_table`, which is a data frame with 3 columns:

1. place_id

[6] We want to change the name of this function, but haven't gotten around to it yet

2. puma_id

3. n_house

The place_id column corresponds to the unique id for all of the regions within a location. For example, the place ID could correspond to tracts within a state, districts within a country, etc...Note that the place_id will be identical for both the shapefile and pop table, as well as any other data inputs being used. The puma_id variable is used for subsetting the PUMS data in order to obtain more accurate samples, and each place ID is contained within a puma ID.

For the USA and Canada, the puma_id variable is used to save the tract level synthetic ecosystems into groups corresponding to their puma_id, whereas for IPUMS data the lowest level of geography we have is the puma_id. For IPUMS microdata, the column corresponding to the puma_id is `GEOLEV1`, which is a numeric. The IPUMS shapefiles have an element for both the `GEOLEV1` name (which is a character in R), and the `GEOLEV1` ID, (a numeric in R). Since the population counts are obtained from Geohive as characters, we match the names these with the IPUMS `GEOLEV1` name, and this becomes the place_id. Similarly, the puma_id is a numeric since the because this is what was available to match with the IPUMS shapefile. In principle, outputting the final formatted table should give the all of the correct lookup information in order to either display the information as an ID variable or the character name. This should be available as a standard SPEW output very soon.

Finally, the `make_data` function takes in the properly formatted data-list and outputs a a synthetic ecosystem for each one of the place_ids. By default, we create a directory for each puma_id, and the place_id synthetic ecosystem is stored in its corresponding puma_id folder. For the USA and Canada, this means we have many tract_ids stored within a puma directory, and for IPUMS, we store the synthetic ecosystem for each puma/place_id within in it's own directory.

The `make_data` function is the key function which implements the spew algorithm. The other two functions are primarily there for formatting purposes. In the future, we hope to make this distrinction more clear, so people only need to function to impement the `spew` algorithm.

C Acknowledgements

This work was supported by the Models of Infectious Disease Agency Study (MIDAS) from the National Institute of General Medical Sciences (NIGMS), Cooperative Agreement NIH 1 U24 GM110707-01. The content is solely the responsibility of the authors and does not necessarily represent the official views of the NIGMS or the National Institutes of Health (NIH).