Stride project

User Manual

Version 1.0 (June 20, 2019) .

Centre for Health Economics Research & Modeling of Infectious Diseases, Vaccine and Infectious Disease Institute, University of Antwerp.

Modeling of Systems and Internet Communication, Department of Mathematics and Computer Science, University of Antwerp.

Interuniversity Institute for Biostatistics and statistical Bioinformatics, Hasselt University.

Willem L, Kuylen E & Broeckhove J

Contents

CHAPTER 1

Introduction

This manual provides a brief description of the Stride software and its features. Stride stands for **S**imulation of **tr**ansmission of **i**nfectious **d**is**e**ases. It is an agent-based modeling system for close-contact infectious disease outbreaks developed by researchers at the University of Antwerp and Hasselt University, Belgium. The simulator uses census-based synthetic populations that capture the demographic and geographic distributions, as well as detailed social networks.

Stride is an open source software. License information can be found in the project root directory in the file `LICENSE.txt`. The authors hope to make large-scale agent-based epidemic models more useful to the community.

More info on the project and results obtained with the software can be found in *"Willem L, Stijven S, Tijskens E, Beutels P, Hens N & Broeckhove J. (2015) Optimizing agent-based transmission models for infectious diseases, BMC Bioinformatics, 16:183"* [**?**] and also in *"Kuylen, E Stijven, S, Broeckhove,J, & Willem, L (2017) Social Contact Patterns in an Individual-based Simulator for the Transmission of Infectious Diseases, Procedia Computer Science, 108C:2438"* [**?**], though a lot functionality in the simulator has been improved, extended and added since those publications.

CHAPTER 2

Software

2.1 Source code

The source code is maintained in a GitHub repository https://github.com/broeckho/stride. We use continuous integration via the TravisCI service. Every new revision is built and tested automatically at commit. Results of this proces can be viewed at https://travis-ci.org/broeckho/stride/branches, where you should look for the master branch. The integration status (of the master branch) is flagged in the GitHub repsitory front page for the project.

Stride is written in C++ and is portable over Linux and Mac OSX platforms that have a sufficiently recent version of a C++ compiler. To build and install `Stride`, the following tools need to be available on the system:

- A fairly recent GNU g++ or LLVM clang++
- make
- A fairly recent CMake
- The Boost library
- Python and SWIG (optional, for calling Stride in Python scripts)
- Doxygen and LaTeX (optional, for documentation only)

A detailed list of current versions of operating system, compiler, build and run tools can be found in the project root directory in the file `PLATFORMS.txt`.

2.2 Installation

To install the project, first obtain the source code by cloning the code repository to a directory . The build system for `Stride` uses the `CMake` tool. This is used to build and install the software at a high level of abstraction and almost platform independent (see http://www.cmake.org/). The project also include a traditional `make` front to `CMake` with For those users that do not have a working knowledge of `CMake`, a front end `Makefile` has been provided that invokes the appropriate `CMake` commands. It provide the conventional targets to "build", "install", "test" and "clean" the project trough an invocation of make. There is one additional target "configure" to set up the `CMake/make` structure that will actually do all the work.

More details on building the software can be found in the file `INSTALL.txt` in the project root directory.

2.3 Documentation

The Application Programmer Interface (API) documentation is generated automatically using the Doxygen tool (see www.doxygen.org) from documentation instructions embedded in the code . A copy of this documentation for the latest revision of the code in the GitHub repository can be found online at https://broeckho.github.io/stride/.

The user manual distributed with the source code has been written in LaTeX(see www.latex-project.org).

2.4 Directory layout

The project directory structure is very systematic. Everything used to build the software is stored in the root directory:

- `main`: Code related files (sources, third party libraries and headers, ...)
  - `main/<language>`: source code, per language: cpp, python, R
  - `main/resources`: third party resources included in the project:
- `doc`: documentation files (API, manual, ...)
  - `doc/doxygen`: files to generate reference documentation with Doxygen
  - `doc/latex`: files needed to generate the user manual with Latex
- `test`: test related files (scripts, regression files, ...)

2.5 File formats

The Stride software supports different file formats:

CSV
: Comma separated values, used for population input data and simulator output.

JSON
: JavaScript Object Notation, an open standard format that uses human-readable text to transmit objects consisting of attribute-value pairs. (see www.json.org)

TXT
: Text files, for the logger.

XML
: Extensible Markup Language, a markup language (both human-readable and machine-readable) that defines a set of rules for encoding documents.

Proto
: Protocol Buffers, used for exporting and importing the generated population and geographical grid.

2.6 Testing

Unit tests and install checks are added to Stride based on Google's "gtest" framework and CMake's "ctest" tool. In addition, the code base contains assertions to verify the simulator logic. They are activated when the application is built in debug mode and can be used to catch errors at run time.

2.7 Results

The software can generates different output files:

cases.csv
: Cumulative number of cases per day.

summary.csv
: Aggregated results on the number of cases, configuration details and timings.

person.csv
: Individual details on infection characteristics.

logfile.txt
: Details on transmission and/or social contacts events.

gengeopop.proto
: Generated population and geographical grid.

2.8 Protocol Buffers

Protocol Buffers[1] is a library used for serializing structured data. We use it to read and write the synthethic populations generated by GenGeoPop (see 4.2).

Protocol Buffers uses an interface description language that describes the structure of the data we want to store, in this case the GeoGrid. The file that describes how the GeoGrid is structured is located at: `main/cpp/gengeopop/io/proto/geogrid.proto`. It is used by the Procol Buffer tool to generate the C++ code to read and write our GeoGrid structure. We include this generated code in the project, so it's not necessary to install the `protobuf-c` package[2] in order to compile Stride. These generated source files can be found in the same folder.

If you want to change the structure, for example due to changes in the way the population is generated, you will need to install the `protobuf-c` package and re-generate the necessary code. To make this easier, we provided a CMake parameter and target that will generate the code and copy it to the source directory respectively. In that case, `Stride` needs to be compiled using the "`STRIDE_GEN_PROTO=true`" macro setting. This will generate the code based on the `geogrid.proto` file and use that instead of the version included in the source directory. If you then want to copy this code to the correct location in the source, you can use "`make proto`". If the version of `protobuf-c` you're using is significantly newer than the `protobuf` included in this repository, you might also need to update the files stored `main/resources/lib/protobuf`.

At the time of writing, this is done in the following way, although this may be subject to change:

- Copy all files from https://github.com/protocolbuffers/protobuf/tree/master/src/google/protobuf to `main/resources/lib/protobuf/google/protobuf`.
- Copy the `libprotobuf_files` and `libprotobuf_includes` from https://github.com/protocolbuffers/protobuf/blob/master/cmake/libprotobuf.cmake and https://github.com/protocolbuffers/protobuf/blob/master/cmake/libprotobuf-lite.cmake
- Remove the prefix `$protobuf_source_dir/src/` from these listings.
  For example `$protobuf_source_dir/src/google/protobuf/any.cc` becomes `google/protobuf/any.cc`.
- Edit the `CMakeLists.text` at `main/resources/lib/protobuf` and replace the current values of `libprotobuf_files` and `libprotobuf_includes` with the ones we just obtained.

The classes that are responsible for reading and writing a GeoGrid to an `istream` containing the serialized data are the `GeoGridProtoReader` and `GeoGridProtoWriter` respectively.

[1] https://developers.google.com/protocol-buffers/
[2] https://github.com/protobuf-c/protobuf-c

CHAPTER 3

Simulator

3.1 Workspace

By default, `Stride` is installed in `./target/installed/` inside de project directory. This can be modified by setting the `CMAKE_INSTALL_PREFIX` on the CMake command line (see the `INSTALL.txt` file in the prject root directory) or by using the CMakeLocalConfig.txt file (example file can be found in `./src/main/resources/make`).

Compilation and installation of the software creates the following files and directories:

- Binaries in directory `<install_dir>/bin`
  - *stride*: executable.
  - *gtester*: regression tests for the sequential code.
  - *gengeopop*: generates the population and geographical grid.
  - *wrapper_sim.py*: Python simulation wrapper
- Configuration files (xml and json) in directory `<install_dir>/config`
  - *run_default.xml*: default configuration file for Stride to perform a Nassau simulation.
  - *run_generate_default.xml*: default configuration file for Stride to first generate a population and geographical grid and then perform a Nassau Simulation.
  - *run_import_default.xml*: default configuration file for Stride to first import a population and geographical grid and then perform a Nassau Simulation.
  - *run_miami_weekend.xml*: configuration file for Stride to perform Miami simulations with uniform social contact rates in the community clusters.

  - *wrapper_miami.json*: default configuration file for the wrapper_sim binary to perform Miami simulations with different attack rates.
  - ...
- Data files (csv) in directory `<project_dir>/data`
  - *belgium_commuting*: Belgian commuting data for the active populations. The fraction of residents from "city_depart" that are employed in "city_arrival". Details are provided for all cities and for 13 major cities.
  - *belgium_population*: Relative Belgian population per city. Details are provided for all cities and for 13 major cities.
  - *flanders_cities*: Cities and municipalities in Flanders with coordinates and population figures based on samples. These relative population figures are used for assigning residencies and domiciles based on a discrete probability distribution.
  - *flanders_commuting*: Relative commuting information between cities and communities. Since this data is relative, the total number of commuters is a derived parameter, based on the fraction of the total population that is commuting.
  - *contact_matrix_average*: Social contact rates, given the cluster type. Community clusters have average (week/weekend) rates.
  - *contact_matrix_week*: Social contact rates, given the cluster type. Community clusters have week rates.
  - *contact_matrix_week*: Social contact rates, given the cluster type. Primary Community cluster has weekend rates, Secondary Community has week rates.
  - *disease_xxx*: Disease characteristics (incubation and infectious period) for xxx.
  - *holidays_xxx*: Holiday characteristics for xxx.
  - *ref_2011*: Reference data from EUROSTAT on the Belgian population of 2011. Population ages and household sizes.
  - *ref_fl2010_xxx*: Reference data on social contacts for Belgium, 2011.
- Documentation files in directory `./target/installed/doc`
  - Reference manual
  - User manual

The install directory is also the workspace for `Stride`. The `Stride` executable allows you to use a different output directory for each new calculation (see the next section).

3.2 Running the simulator

From the workspace directory, the simulator can be started using the command "`./bin/stride`". Arguments can be passed to the simulator on the command line:

```
USAGE:

   bin/stride  [-e <clean|dump|sim|genpop>] [-c <CONFIGURATION>] [-o
               <<NAME>=<VALUE>>] ...  [-i] [--stan <COUNT>] [--]
               [--version] [-h]

Where:

   -e <clean|dump|sim|genpop>,  --exec <clean|dump|sim|genpop>
     Execute the corresponding function:

         clean:  cleans configuration and writes it to a new file.

         dump:   takes built-in configuration writes it to a file.

         sim:    runs the simulator and is the default.

         genpop: generates geo-based population to file (no sim)

     Defaults to --exec sim.

   -c <CONFIGURATION>,  --config <CONFIGURATION>
     Specifies the run configuration parameters. The format may be either
     -c file=<file> or -c name=<name>. The first is mostly used and may be
     shortened to -c <file>. The second refers to built-in configurations
     specified by their name.

     Defaults to -c file=run_default.xml

   -o <<NAME>=<VALUE>>,  --override <<NAME>=<VALUE>>  (accepted multiple
      times)
     Override configuration file parameters with values provided here.

   -i,  --installed
     File are read from the appropriate (config, data) directories of the
     stride install directory. If false, files are read and written to the
     local directory.

     Defaults to true.
```

```
--stan <COUNT>
  Stochastic Analysis (stan) will run <COUNT> simulations, each with a
  different seed for the random engine. Only applies in case of -e sim.

--,  --ignore_rest
  Ignores the rest of the labeled arguments following this flag.

--version
  Displays version information and exits.

-h,  --help
  Displays usage information and exits.
```

For example if you run with the default configuration file, but you want to change the logging level, (choices are: `trace`, `debug`, `info`, `warn`, `error`, `critical`, `off`)) execute:

```
stride --override stride_log_level=error
```

3.3 Python Wrapper

A Python wrapper is provided to perform multiple runs with the C++ executable. The wrapper is designed to be used with .json configuration files and examples are provided with the source code. For example:

```
./bin/wrapper_sim --config ./config/wrapper_default.json
```

will start the simulator with each configuration in the file. It is important to note the input notation: values given inside brackets can be extended (e.g., "rng_seeds"=[1,2,3]) but single values can only be replaced by one other value (e.g., "days": 100).

CHAPTER 4

Concepts and Algorithms

4.1 Introduction

The model population consists of households, schools, workplaces and communities, which represent a group of people we define as a "ContactPool". Social contacts can only happen within a ContactPool. When school or work is off, people stay at home and in their primary community and can have social contacts with the other members. During other days, people are present in their household, secondary community and a possible workplace or school.

We use a *Simulator* class to organize the activities from the people in the population. The ContactPools in a population are grouped into ContactCenters (e.g. the different classes of a school are grouped into one K12School ContactCenter). These ContactCenters are geographically grouped into a geographical grid (sometimes called GeoGrid)

The *ContactHandler* performs Bernoulli trials to decide whether a contact between an infectious and susceptible person leads to disease transmission. People transit through Susceptible-Exposed-Infected-Recovered states, similar to an influenza-like disease. Each *ContactPool* contains a link to its members and the *Population* stores all personal data, with *Person* objects. The implementation is based on the open source model from Grefenstette et al. [**?**]. The household, workplace and school clusters are handled separately from the community clusters, which are used to model general community contacts. The *Population* is a collection of *Person* objects.

4.2 Population on a geographic grid

4.2.1 Background

To explain the algorithms used for generating the geography of the countries and their respective population, we have to introduce some background concepts.

ContactPool : A pool of persons that may contact with each other which in turn may lead to disease transmission. We distinguish different a number of types of ContactPools associated with the household, the workplace, the school, ... The Household is a key type because it fixed the home address of a person.

ContactCenter : A group of one or more ContactPools of the same type, at the same Location. This construct allows e.g for a K12School (ContactCenter) to contain multiple classes (ContactCenter). The former has a reference size of 500 (again a configurable reference number) pupils in 25 (again a configurable reference number) classes. A ContactCenter may however also associated one-on-one with a ContactPool; this is the case for ContactCenters of type Household.

Age Brackets : We define age brackets that classify individuals in terms of the type of ContactPool they can be a member of. Everyone is member of a one Houseold ContacCenter and ContactPool. For 6 ¡= age ¡ 18 one is a K12School student, for 18¡= age ¡ 26 one is either a member of a College ContactCenter or member of a Workplace ContactCenter or neither (i.e. not a college student and not employed). Details of all the rules can be found in the code documentation.

K-12 student : Persons from 3 until 18 years of age that are required (at least in Belgium) to attend school. Students that skip or repeat years are not accounted for.

College student : Persons older than 18 and younger than 26 years of age that attend an institution of higher education. For simplicity we group all forms of higher eduction into the same type of ContactCenter, a College. A fraction of college students will attend a college "close to home" and the others will attend a college "far from home". Most higher educations don't last 8 years, but this way we compensate for changes in the field of study, doctoral studies, advanced masters and repeating a failed year of study.

Employable : We consider people of ages 18 to 65 to be potentially employable. A fraction of people between 18 and 26 will attend a college, and the complementary fraction will be employable.

Active population : The fraction of the employable population that is actually working. A fraction of these workers will work "close to home" and the complementary fraction will commute to a workplace "far from home".

Household profile : The composition of households in terms of the number of members and their age is an important factor in the simulation. In this case the profile is not defined by the age of its members or fractions, but through a set of reference households. This set contains a sample of households which is representative of the whole population in their composition.

GeoGrid locations : Our data only allows for a limited geographical resolution. We have the longitude and latitude of cities and municipalities (a distinction we will not make), which we shall use to create GeoGrid locations for the domicile of the households. All households in the same location are mapped to the coordinates of the location's center. These locations with corresponding coordinates will form a grid that covers the simulation area.

4.2.2 Generating the geography

We start by generating the geographical component, a GeoGrid. It contains locations with an id, name, province, coordinates and a reference population count.

The latter requires some comment. When we build a synthetic population, we start by generating households. That household is the assigned a location by drawing from a discrete distribution of locations with weights proportional to the relative population count of the location. As a consequence, each location will have a population count that differs stochastically from the reference count, but will be close to it. When many synthetic populations are generated, the average of a location 's population count will tend to its reference count.

Locations contain multiple ContactCenters, like Schools and Households, which in turn contain ContactPools. This structure is internally generated by several "Generators" and will afterwards be used by "Populators" to fill the ContactPools. The different types of ContactCenters are created by a different partial generator for each type and added separately to the GeoGrid. We construct the following types of ContactCenters:

Households : The number of households is determined by the average size of a household in the reference profile and the total population count. The generated households are then assigned to a location by a draw from a discrete distribution of locations with weights proportional to the relative population count of the location.

K-12 Schools : Schools a reference count of 500 students, with a refernce count of 20 students per class, corresponding to 25 ContactPools per school. The total number of schools in the region is determined by the population count, the fraction of people in the K-12 school age bracket and of course the reference school size. The algorithm for assignment of schools to locations is similar to that of households.

Colleges : Colleges have a reference count of 3000 students with 150 students per ContactPool. Colleges are exclusively assigned to the 10 locations with highest reference population count (cities). For these 10 locations we use a discrete distribution with weights proportional to the population count of the city relative to the total population of the 10 cities.

Workplaces : The algorithm for assignment of workplaces to locations is analogous to that of households, but here we factor in commuting information to determine the actual number of workers at a location. That is, the reference count of active persons (i.e. persons assigned to a workplace) is the number of active persons living at a location (i.e. their household is located there) plus the active persons commuting out of that location minus the number of active persons commuting into that location. The reference count for persons at a workplace is 20.

Communities : We create both primary and secondary communities. Each has a reference count of 2000 persons. Communities consist of persons from all ages. The assignment to locations is again similar to that of households.

4.2.3 Generating the population

After creating the structures that will allow people to come in contact with each other, we have create the population itself and determine the different ContactPools they will be in. The persons are created based on the Household profiles in the HouseholdPopulator. Similar to the approach the ContactCenters, we have a partial populators that each populates ia type of ContactPool:

Households : To create the actual persons, we randomly draw a household from the list of reference households and use that as a template for the number of household members and their ages. We simply do this for each household in the GeoGrid, since we already determined the locations and number of households while generating the geography.

K-12 Schools : We start by listing all schools within a 10km radius of the household location. If this list is empty, we double the search radius until it's no longer empty. We then randomly select a ContactPool from those in the schools of the list, even if this ContactPool now has more students than average.

Colleges : Students who study "close to home" are assigned to a college with an algorithm similar to the assignment to K-12 schools. Students that study "far from home" we first determine the list of locations where people from this locations commute to. We randomly select one of these locations using a discrete distribution based on the relative commuting information. We then randomly choose a ContactPool at a college in this location and assign it to the commuting student.

Workplaces : We first decide whether the person is active (correct age, fraction of the age bracket that is a student, fraction of the age bracket that is active). We assign a workplace to an active person that works "close to home" in an analogous way as the assignment of K-12 schools to students. For the commuting workers we use an algorithm analogous to that of the commuting college students.

Communities : The communities we can choose from at a location is determined in an analogous way to the K-12 schools. For primary communities we randomly select, for each person in a household, a ContactPool from the list of

Communities within a 10km radius. If the list empty, ... see the K-12 algoritm. In secondary communities however, we assign complete households to the ContactPools instead of each person in the household separately.

CHAPTER 5

Additional Features

5.1 Introduction

For our bachelor thesis, we were asked to implement a couple of additional features to the original Stride project. We were requested to add two additional contact pools, being Daycare and PreSchool, and two additional data formats, being JSON and HDF5, where JSON would be used for GeoGrid's and households, and HDF5 would be just for GeoGrid's, a visualization tool which will allow the user to get a visual overview of an instance, demographic profiles, for more specific households and finally workplace size distribution, to distinguish different workplace sizes and their chance of occurring.

Each of these expansions came along with their respective implementations and tests. The following implementations were added to the system, and they are used, if not automatically, in the following manner.

5.2 Demographic Profile

5.2.1 Idea

The initial idea for this feature was that we were previously incorrectly assuming that every household is the same in the entire country. We therefore wanted to

be able to read, whenever given, multiple configurations for households, so that we would be able to generate the household in a more specific manner, rather than using a general set that is the same for the entire population.

A secondary issue was that not all city types have the same household data, the major cities might have a very different ratio for young and old (which is what the differences between the household configurations are based on) than for example a smaller village might have. Since it is very difficult to determine what each city's type is, we considered the major cities versus all other cities.

5.2.2 Input

The way this data is given to stride is via the configuration file for the population. There already was a required field in the geopop_gen section (if it exists) which was household_file, this field will still be used, and will still be required, it will now hold the data which will be used if there is no more specific configuration at hand.

The following input fields were added, and will be used as their respective household configuration, if given. The final file is the file used to determine which cities are considered major:

Antwerp :
antwerp_household_file

Flemish Brabant :
flemish_brabant_household_file

West Flanders :
west_flanders_household_file

East Flanders :
east_flanders_household_file

Limburg :
limburg_household_file

Major Cities Households :
major_cities_file

Major Cities :
major_cities

5.2.3 New implementations for generators

A few decisions had to be made in order to make this work. The basis was simple, we had to make a reader that read the corresponding configuration files which would store these into the memory. What was more difficult, was determining whether or not these changes should be applied to certain pools and why. The following generators are altered in the following manner.

Households Generator : The *household generator* will iterate over the different regions, it will consider all locations within that region and will, for each of these locations, generate the corresponding households. It will use the given household configuration, if it was available, or the default configuration if not.

College Generator : The *college generator* will iterate over the different regions and will generate colleges where needed, it will however not consider any specific household data, as this is irrelevant to the existence of colleges. People go to colleges all over the country, so it is more dependent on how accessible they are and how many people live in an area, rather than the possible configurations of set households in that area.

K12School Generator : The *K12school generator* will iterate over the different specified regions and will use the possibly available data for that region. It is logical for a k12school to be dependent on the population in the surrounding environment since most of the students will generally go to a nearby k12 school.

PreSchool Generator : The *preschool generator* will follow the same logic as the k12school did in which it will use the specific province's/major city's household data, if available.

DayCare Generator : The *daycare generator* will follow the same logic as the k12school did in which it will use the specific province's/major city's household data, if available.

Workplace Generator : The *workplace generator* is a bit more difficult, in the sense that bigger workplaces will behave like the college pools did, in which they would attract people from all over the country, and other factors would become more important, while the small workplaces would behave just like K12school did, in which it would attract people from the neighboring area most of the time. We made the decision to make workplaces dependent on the household configurations of the specific areas, since almost all of the workplaces would be of a small size.

Primary- and Secondary Community : These are not affected by household factors, and are therefore left unaltered.

5.3 Workplace Size Distribtution

5.3.1 Idea

The initial idea for this feature was based on the fact that not all workplaces are equal in size, and not all of the sizes are equally as common. In order to resolve this we added an extra input file which specifies the different sizes, along with the chance that they occur.

5.3.2 Input

This addition adds one extra, optional, input tag, $workpalce\_file$, in the geopop gen section of the populations config file which will be used to generate the workplaces.

5.3.3 Workplace Generator

Rather than creating a workplace for every 20 persons, we will now iterate for as long as we do not have sufficient space to allow all people that should work, to work. In each iteration we will generate a random size (taking in account the given chances for each size) for a new workplace pool. We will add these sizes to the geogrid configuration.

5.3.4 Workplace Populator

The Workplace populator will select a random workplace pool for each of the workers. As long as there are places left in the workplaces, these places will get priority over the other workplaces. As soon as all pools are filled to their intended size, we will randomly select pools out of all the pools within a certain range. The pools will not always get perfectly filled to their intended size, even though we intended to create an exact fit since randomness is used in the populator.

5.4 Data Formats

5.4.1 JSON: GeoGrid

This feature offers the user to create a GeoGrid file in JSON format. These files are handled by a reader and a writer based by the concepts of "JSON for Modern C++".

If a JSON file is prefered to be used as configuration for the GeoGrid population then it is necessary to alter some attributes in the XML configuration file:
Change the filename of the tag `population_file` to a file written in the JSON format.

A JSON formatted file for a GeoGrid configuration has the following attributes:

- `locations` : A list containing objects with:
  - `id` (number)
  - `name` (string)
  - `province` (string)
  - `population` (number)

  - `coordinate`
    - `latitude` (number)
    - `longitude` (number)
  - `commutes` : A list with objects formatted as:
    - `to` (number) : ID of a location.
    - `proportion` (number) : Percentage of people commuting to this location.
  - `contactPools` : List of pools per class containing the attributes:
    - `class` (string defining one of the contact types)
    - `pools` : Each pool defines the persons belonging to it:
      - `id` (number)
      - `people` (list of numbers)

- `persons` : A list cintaining objects with:
  - `id` (number)
  - `age` (number)
  - `daycare` (number) : ID of pool in which this person is located.
  - `preSchool` (number) : ID of pool in which this person is located.
  - `k12School` (number) : ID of pool in which this person is located.
  - `household` (number) : ID of pool in which this person is located.
  - `workplace` (number) : ID of pool in which this person is located.
  - `primaryCommunity` (number) : ID of pool in which this person is located.
  - `secondaryCommunity` (number) : ID of pool in which this person is located.
  - `college` (number) : ID of pool in which this person is located.

5.4.2 JSON: Household

Besides reading household information in CSV format, we offered a way to provide this data in JSON format. We applied the same concept of "JSON for Modern C++" to implement a Reader that can handle such files.

If you want to use this feature and give a JSON formatted as configuration for the households then you need to change the following in the configuration XML file: When the tag `population_type` is in generating mode then you need to change the tag `household_file` (within the `geopop_gen` tag) to the name of the JSON formatted file you wish to use.

A JSON formatted file for a GeoGrid configuration has the following attributes:

- `householdsName` (string)
- `householdsList` (list of lists of numbers) : A list with lists containing the ages within households.

5.4.3 HDF5: GeoGrid

Another supported dataformat is the HDF5 format. Hierachical DataFormat (HDF5) is a file format designed to store and organize a large amount of data. HDF5 files are defined by a .h5 extension.

If a HDF5 file is prefered to be used as configuration for the GeoGrid population then it is necessary to alter some attributes in the XML configuration file:
Change the filename of the tag `population_file` to a file written in the HDF5 format.

A Geogrid formatted file written in HDF5 has the following format:

- `Locations` : Group
  - `Location`i : Group
    - `ContactPools` : Group
      - `ContactPool`i : Dataset of unsigned ints
      - `Commutes` : Dataset of Commute types
- `Persons` : Dataset or Person types

`Locations`

- `size` : unsigned long

`Location`i

- `id` : unsigned int
- `name` : variable length c-style string
- `province` : unsigned int
- `population` : unsigned int
- `longitude` : double
- `latitude` : double
- `size` : unsigned long

`ContactPools`

- `size` : unsigned long

`ContactPool`i

- `id` : unsigned int

- `size` : unsigned long
- `type` : variable length c-style string

`Commutes`

- `size` : unsigned int

`Persons`

- `size` : unsigned int

5.5 Daycares and PreSchools

The current implementation in stride generates contact pools for different ContactTypes and their associated AgeBrackets. There are two added ContactTypes for 0-3 years old and 3-6 years old, Daycare and Preschool. They have fixed participation rates that can't be altered via config. The participation rate for Daycare is 0.45 and the participation rate for Preschool is 0.99.

5.6 Seperating Geograpic and Demographic Layer

5.6.1 Idea

In Stride the classes *GeoGrid* and *Location* both held the geographical and demographical information. The idea was to split these functionalities into seperate layers. A geographical layer which holds the geographical information like coordinates and place names, while the demographic layer holds other information namely the population. This would allow for the reuse of the geographical layer in the GUI.

5.6.2 Implementation

The seperation was done via inheritance. The geographical parts of *GeoGrid* and *Location* were moved to two new classes *Region* and *GeoLocation*. *Location* would inherit from *GeoLocation* and *GeoGrid* from *Region*. This allowed for new classes like *EpiGrid* and *EpiLocation* to inherit from *Region* and *GeoLocation* as well.

5.7 Epidemiological output

As input for the GUI, an epidemiological output was proposed. This **epi_output** contains for every saved timestep the epidemiological situation. For every location the percentage of people per *HealthStatus* per *ContactType* is saved. To write this file during the simulation an extra viewer was created, the *EpiViewer*. It will write the epi-output to the desired file.
The controller was not yet adapted to fully function with this feature. The *ControlHelper* will be changed so it will check for an **epi_output** filename to write to.

5.8 Data Visualization

Currently only the model of the GUI has been implemented. An *EpiReader* will read the **epi_output** and create an *EpiGrid*. This class will hold *EpiLocations* which will hold the coordinates of the location and for every timestep the a *PoolStatus*. This *PoolStatus* will hold the *HealthPool* for every *ContactType*. The *HealthPool* will in turn have the percentage of people in every *HealthStatus*.
The GUI will be called by a commandline which will tell where the **epi_output** is located.