
Stride User Manual Documentation

Release

Willem L, Kuylen E, Broeckhove J, Janssens S, Hermans A, Mylle

Jun 27, 2017

Contents:

1	Introduction	3
2	Software	5
2.1	System Requirements	5
2.2	Installation	5
2.3	Documentation	6
2.4	Directory layout	6
2.5	File formats	6
2.6	Testing	6
2.7	Results	7
3	Simulator	9
3.1	Workspace	9
3.2	Run the simulator	10
3.3	Configuration File	10
4	Checkpointing	13
4.1	Configuring checkpointing	13
4.2	HDF5 file format	14
5	Population Generator	17
5.1	Using the generator	17
6	Visualization	21
6.1	Configuration	21
6.2	Using visualization	21
6.3	Overview features	22
7	Bibliography	27
	Bibliography	29

The following research groups or institutes have provided contributions:

- 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.

Apart from this User Manual, there's also a Reference Manual available, generated by Doxygen.

This manual provides a brief description of the Stride software and its features. Stride stands for **S**imulation of **t**ransmission of **i**nfectious **d**iseases and is an agent-based modeling system for close-contact disease transmission 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. 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 [\[WST+15\]](#).

The model population consists of households, schools, workplaces and communities, which represent a group of people we define as a “cluster”. Social contacts can only happen within a cluster. 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 an `Area`. The `Area` class has a `Population`, different `Cluster` objects and a `Contact Handler`. The `Contact Handler` 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 `Cluster` 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. [\[GBR+13\]](#). 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.

System Requirements

Stride is written in C++ and portable over all platforms that have the GNU C++ compiler. The software has no dependencies on external libraries. The following tools need to be installed:

- g++
- make
- CMake
- Boost
- Python (optional, for automatization)
- Doxygen (optional, for documentation)
- LaTeX (optional, for documentation)
- Sphinx (optional, for documentation)
- Nodejs, npm, electron and electron-packager (optional, for visualization)

Installation

To install the project, first obtain the source code by cloning the repository to a directory or download a zip file with all project material from the Bitbucket website and de-compress the archive. 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 includes the conventional make targets to “build”, “install”, “test” and “clean” the project. There is one additional target “configure” to set up the CMake/make structure that will actually do all the work. 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. More details on building the software can be found in “INSTALL.txt” in the source folder.

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 .

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

Directory layout

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

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

File formats

The Stride software supports different file formats:

CSV

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

HDF5

Hierarchical Data Format 5, designed to store and organize large amounts of data.

JSON

JavaScript Object Notation, an open standard format that uses human-readable text to transmit objects consisting of attribute-value pairs.

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.

Testing

Unit tests and install checks are added to Stride based on Google's *gtest* framework. 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.

Results

The software can generates different output files:

cases.csv

Cumulative number of cases per day.

person.csv

Individual details on infection characteristics.

logfile.txt

Details on transmission and/or social contacts events.

Workspace

By default, Stride is installed in `./target/installed/` inside the project directory though this can be modified using the `CMakeLocalConfig.txt` file (example is given in `./src/main/resources/make`). Compilation and installation of the software will create the following files and directories:

Binaries in directory `<project_dir>/bin`

- `stride`: executable.
- `gtester`: regression tests for the sequential code.
- `wrapper_sim.py`: Python simulation wrapper, this wrapper is not updated and will not work with the current configuration setup of the project.

Configuration files (xml and json) in directory `<project_dir>/config`

- `run_default.xml`: default configuration file for Stride to 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.
- `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 week-end rates, Secondary Community has week rates.
- `disease_xxx`: Disease characteristics (incubation and infectious period) for xxx.
- `holidays_xxx`: Holiday characteristics for xxx.
- `pop_xxx`: Synthetic population data extracted from the 2010 U.S. Synthetic Population Database (Version 1) from RTI International for xxx [\[Int14\]](#), [\[Whe14\]](#).
- `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

Run the simulator

From the workspace directory, the simulator can be started with default configuration using the command `./bin/stride`. Settings can be passed to the simulator using one or more command line arguments:

- `-c` or `--config`: The configuration file.
- `-m` or `--mode`: The simulation run mode (defaults to `Extend` mode).
- `-f` or `--hdf5_file`: The HDF5 file (only used for mode ‘extract’)
- `-o` or `--override`: Override configuration file options in the command line.
- `-t` or `--timestamp`: The timestep from which `Replay` mode is replayed.

The usage of a configuration file is necessary unless you choose the `extract` mode. From this configuration file all the necessary files will be read.

Overrides can be used to quickly override a certain value in the configuration. Multiple `-o` flags can be given. This is based on the API of the Boost property tree, with the following exceptions:

- The first element is always `run`, so this is implicit.
- To get an XML attribute, you can use `@` instead of `<xmlattr>..`

Some typical examples:

- `-o @name=other_name` to change the name of the current run (and therefore the output directory)
- `-o disease.config=other_disease.xml` to try another disease.

Output can be found in `output/<name>`. Mind the fact that you will overwrite your previous run if you don’t change the name.

Configuration File

```
<?xml version="1.0" encoding="utf-8"?>
<run name="default">
  <r0>11</r0>
  <start_date>2017-01-01</start_date>
  <num_days>50</num_days>
  <holidays>holidays_none.json</holidays>
  <age_contact_matrix_file>contact_matrix_average.xml</age_contact_matrix_file>
  <track_index_case>1</track_index_case>
  <num_threads>1</num_threads>
  <information_policy>Global</information_policy>
  <outputs>
    <log level="Transmissions"/>
    <person_file/>
    <participants_survey num="10"/>
    <visualization/>
    <checkpointing frequency="1"/>
  </outputs>
  <disease>
    <seeding_rate>0.002</seeding_rate>
    <immunity_rate>0.8</immunity_rate>
    <config>disease_measles.xml</config>
  </disease>
  <regions>
    <region name="Belgium">
      <rng_seed>1</rng_seed>
      <raw_population>pop_nassau.csv</raw_population>
    </region>
    <region name="">
      <rng_seed>1</rng_seed>
      <population>pop.xml</population>
    </region>
  </regions>
</run>
```

The population, as referenced in a <region> can be either a <raw_population> or a <population>. The first option is a simple csv, the second one an XML file with the following format:

```
<?xml version="1.0" encoding="utf-8"?>
<population>
  <people>people.csv</people>
  <districts>cities.csv</districts>
  <sphere_of_influence speed="100" size="20" min="5"/>
  <clusters>clusters.csv</clusters>
  <households>households.csv</households>
  <cities>
    <city name="Antwerp" pop="5000" lat="51.123" lon="4.567">
      <airport name="ANR"/>
    </city>
    <city name="Brussels" pop="10000" lat="50.850" lon="4.348">
      <airport name="BRU"/>
    </city>
  </cities>
</population>
```

Here, the <people> tag refers to the same kind of file as a <raw_population>.

You can use multiple regions for the multi region feature. The output tags <visualization/> and <checkpointing_frequency/> enable the saving of hdf5 or visualization files.

Configuring checkpointing

Checkpointing is configured using command line options and/or specifying certain parameters in the configuration file. These options are specified in the chapters above. For more detailed information on how to configure checkpointing, read the Simulator chapter (Run the simulator part).

Checkpointing enables the ability to save the state of the simulator multiple times during the simulation itself. The simulator state is saved in a binary format, based on a HDF5 storage format. The format of this file is specified below. Checkpointing is configured using 3 parameters: the checkpointing frequency, the checkpointing file and the simulator run mode.

Checkpointing frequency

How frequently the simulator will be saved, can be set by the checkpointing frequency parameter. This parameter can be set by using a commandline argument or specifying the parameter in the xml configuration file. This are the possible values for the parameter:

- 0 - Only save the last timestep of the simulation
- x - Save the simulator state every x timesteps

Checkpointing file

This parameter specifies the name of the checkpointing file. The use for the file depends on the simulator run mode parameter.

Replay timestep

This parameter is used when the run mode is `Replay`. It specifies the timestep from which you want to start the simulation.

Simulator run mode

The simulator can be run in different modes. Currently, the following run modes are supported:

No starting from checkpointed file in initial mode.

- Initial - The simulator is built from scratch using the configuration file, and is saved every x timesteps according to the checkpointing frequency.
- Extend - The simulation is extended from the last saved checkpoint in the checkpointing file.
- Replay - The simulation is replayed from a specified timestep.
- Extract - The configuration files are extracted from the checkpointing file. This mode will not actually run the simulator itself.

HDF5 file format

Table structure

/Configuration/configuration	rank	1
	dims	1
	dtype	ConfigDatatype
/amt_timesteps	rank	1
	dims	1
	dtype	H5T_NATIVE_UINT
/last_timestep	rank	1
	dims	1
	dtype	H5T_NATIVE_UINT
/person_time_independent	rank	1
	dims	amt_persons
	dtype	PersonTIDatatype
/Timestep_n/randomgen	rank	1
	dims	1
	dtype	StrType
/Timestep_n/person_time_dependent	rank	1
	dims	amt_persons
	dtype	PersonTDDatatype
/Timestep_n/calendar	rank	1
	dims	1
	dtype	CalendarDatatype
/Timestep_n/travellers	rank	1
	dims	amt_travellers
	dtype	TravellerDataType
/Timestep_n/household_clusters	rank	1
	dims	amt_persons
	dtype	H5T_NATIVE_UINT
/Timestep_n/primary_community_clusters	rank	1
	dims	amt_persons
	dtype	H5T_NATIVE_UINT
/Timestep_n/secondary_community_clusters	rank	1
	dims	amt_persons

Continued on next page

Table 4.1 – continued from previous page

	dtype	H5T_NATIVE_UINT
/Timestep_n/work_clusters	rank	1
	dims	amt_workers
	dtype	H5T_NATIVE_UINT
/Timestep_n/school_clusters	rank	1
	dims	amt_students
	dtype	H5T_NATIVE_UINT

Custom datatypes

ConfigDatatype

- StrType - config_content
- StrType - disease_content
- StrType - holidays_content
- StrType - age_contact_content

PersonTIDatatype (time independent)

- H5T_NATIVE_UINT - ID
- H5T_NATIVE_DOUBLE - age
- H5T_NATIVE_CHAR - gender
- H5T_NATIVE_UINT - household_ID
- H5T_NATIVE_UINT - school_ID
- H5T_NATIVE_UINT - work_ID
- H5T_NATIVE_UINT - prim_comm_ID
- H5T_NATIVE_UINT - sec_comm_ID
- H5T_NATIVE_UINT - start_infectiousness
- H5T_NATIVE_UINT - time_infectiousness
- H5T_NATIVE_UINT - start_symptomatic
- H5T_NATIVE_UINT - time_symptomatic

PersonTDDatatype (time dependent)

- H5T_NATIVE_HBOOL - participant
- H5T_NATIVE_UINT - health_status
- H5T_NATIVE_UINT - disease_counter
- H5T_NATIVE_UINT - on_vacation

CalendarDatatype

- H5T_NATIVE_HSIZE - day
- StrType - date

TravellerDataType

This type consists of person data from original simulator, as well as data from the new simulator. Person data which is similar over both simulators is only saved once (such as gender data).

Other than that, the data type also contains metadata information:

- H5T_NATIVE_VARIABLE - home_sim_name
- H5T_NATIVE_VARIABLE - dest_sim_name
- H5T_NATIVE_UINT - home_sim_index
- H5T_NATIVE_UINT - dest_sim_index
- H5T_NATIVE_UINT - days_left

Elaboration

First of all, the configuration files are saved. This allows for independent runs for later simulations, by using the stored configurations.

In terms of person data, the time independent data is saved once. The time dependent data is stored at each save.

The order of person id's in the different cluster types is saved as well. This, in combination with the saving of the rng state, guarantees that the run can be resumed exactly similar to the state in which it was saved. This also allows the exact same end results when running the simulation without multithreading.

As part of the multi region extension, travellers are saved too. This allows for a reconstruction of the simulation with multi region travellers present.

Using the generator

Command line interface

We provided the population generator with a command line interface (TCLAP). This interface contains a help flag which gives more information about the specific arguments. In order to display this help you must execute the following command:

```
./pop_generator -h
```

Input files

The population generator needs two files:

- The xml configuration
- A file with the configuration of families

An example of an xml configuration file can be found here: `src/main/resources/templates/PopGenerator.xml`

Meaning of the attributes in this file

- `population::total`: The total size of the population, the result may contain a small difference
- `population::provinces`: The amount of provinces (currently has no effect in stride)
- `population.family::file`: The file containing the family configurations
- `population.commutingdata::start_radius`: The start radius for when a person is commuting, this person will be assigned to a cluster within this radius

- `population.commutingdata::factor`: If there are no clusters within the start radius, multiply it by this factor and search for other clusters
- `population.cities.city::name`: The name of a city, this city will be generated
- `population.cities.city::pop`: The amount of people in this city
- `population.cities.city::lat`: The latitude of the city
- `population.cities.city::lon`: The longitude of the city
- `population.cities.city.airport::name`: The name of an airport situated in this city
- `population.villages::radius`: When calculating the position of a village, the generator first calculates the weighted middle (average latitude and longitude) of the cities. Then it calculates the maximum distance between this middle and any city. The radius is then used as a factor to determine the maximum distance a village is located from the middle.
- `population.villages.village`: This contains a template of a village. When a village is needed, the generator will pick a random template
- `population.villages.village::min`: The minimum size of the village
- `population.villages.village::max`: The maximum size of the village
- `population.villages.village::fraction`: The chance of this village template being chosen, all fractions must add up to one
- `population.education.mandatory`: Contains the configuration for mandatory schools
- `population.education.mandatory::total_size`: Size of a mandatory school
- `population.education.mandatory::cluster_size`: Size of a group within a mandatory school
- `population.education.optional`: It's attributes and purpose are the same as mandatory schools
- `population.education.optional::total_size`: Size of an optional school
- `population.education.optional::cluster_size`: Size of a group within an optional school
- `population.education.optional.far::fraction`: Fraction of students that goes to a school that is further away from his home
- `population.work::size`: The size of a workplace
- `population.work.far::fraction`: The fraction of working people that goes to workplaces that are located far away from their homes
- `population.community::size`: The size of a community
- `population.community::average_per_person`: Currently not used
- `population.school_work_profile.mandatory::min`: The minimum age for students on a mandatory school
- `population.school_work_profile.mandatory::max`: The maximum age for students on a mandatory school
- `population.school_work_profile.employable::fraction`: Fraction of people that is employed. The others are either students on an optional school, or unemployed.
- `population.school_work_profile.employable.young_employee::min`: Minimum age for students at an optional school
- `population.school_work_profile.employable.young_employee::max`: Maximum age for students at an optional school

- `population.school_work_profile.employable.young_employee::fraction:` Fraction of people within the age range of students that have a job
- `population.school_work_profile.employable.employee::min:` Minimum age of a working person (that is too old for any school)
- `population.school_work_profile.employable.employee::max:` Maximum age of a working person (that is too old for any school)

Restrictions / options

Some min-max pairs should not overlap. E.g. min-max pairs of villages or the maximum age of a young employee should be smaller than the minimum age of a young employee. Also, the `factor` in `population.commutingdata` should be greater than 1.0. Cities aren't required to have airports, and they can have as many airports as you'd like.

Family configuration file

This contains the possible configurations of each family (based on age). Every row is a family. The Population Generator randomly chooses configuration.

Random generators

The population generator uses the mt19937 random generator by default. Besides the mt19937 random generator, you can choose one of the following:

- `default_random_engine`
- `mt19937_64`
- `minstd_rand0`
- `minstd_rand`
- `ranlux24_base`
- `ranlux48_base`
- `ranlux24`
- `ranlux48`
- `knuth_b`

For more information about these generators please go to <http://www.cplusplus.com/reference/random/>.

Configuration

Installing

Building the visualization plugin requires a few libraries/packages. The list of packages can be found in the Software chapter (part System Requirements).

You can install the visualization post processing tool by executing the command `make install_vis` in the root directory of the project. This will install the binary visualization executable in the install directory, `vis/visualization`.

Run configuration

Visualization is configured via a parameter in the run config file. The existence of the tag `run.outputs.<visualization/>` is enough to enable the output of visualization data.

Using visualization

In order to use the tool, first install it by following the instructions in the section above. Make sure that the visualization config run option `run.outputs.<visualization/>` is enabled as well.

Run Stride as you would normally do. After the simulator has ran its course, it has generated the necessary cluster data to CSV files in the `output` directory. You can now start the visualization tool by executing `vis/visualization <OUTPUT_DIR>`. The output directory is the directory relative from the install directory, for example `output/default/vis_Belgium`. The tool will display the cluster data generated by the simulator.

Overview features

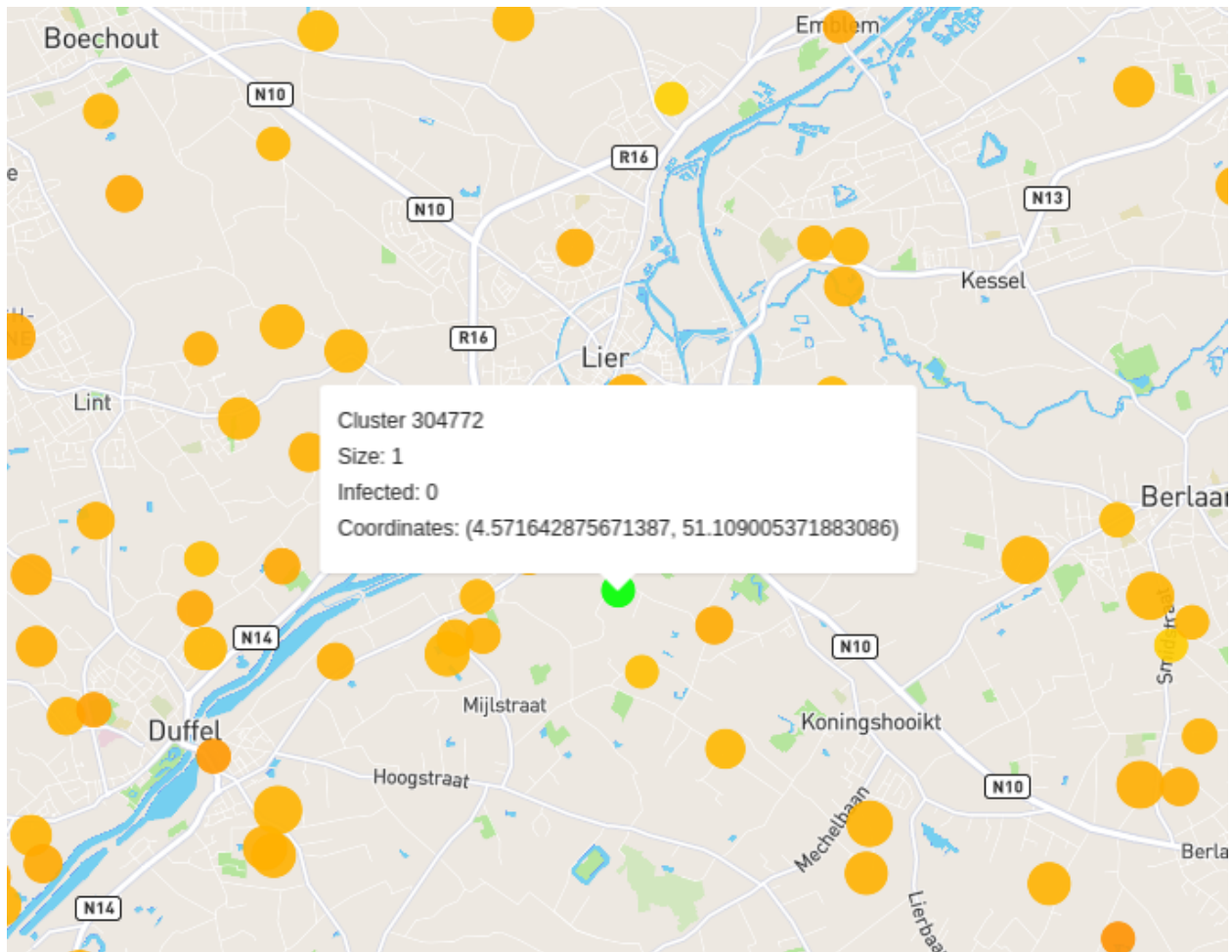
World map

The first thing you see, when running the tool, is a world map. On this map you can see several circles, which represent the clusters saved by the simulator.

The color of the circles represents the percentage of infected people in the cluster. The following color values are used by default:

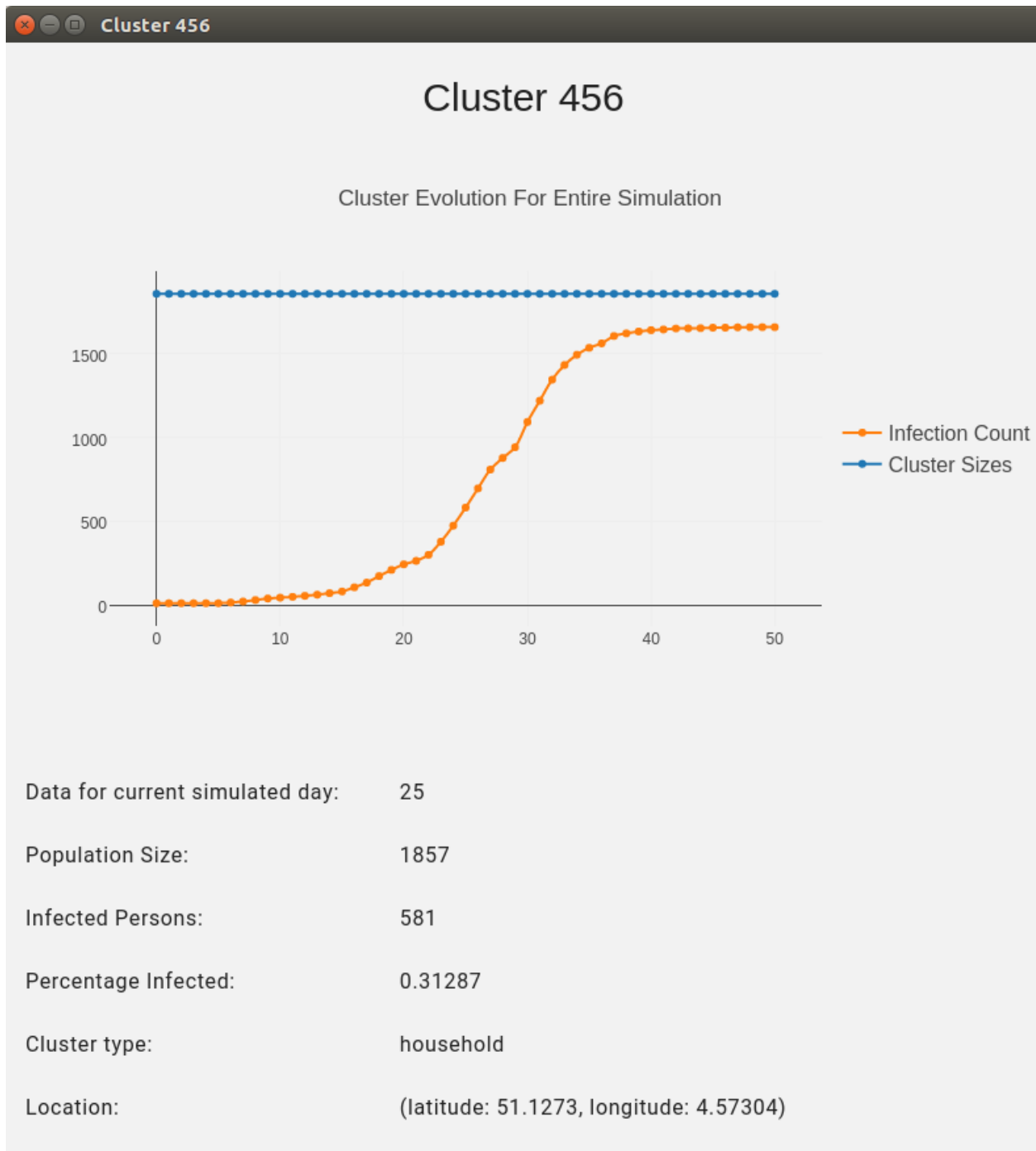
- Green: No people are infected in this cluster.
- Yellow: A low percentage of people are infected in this cluster.
- Red: A high percentage of people are infected in this cluster.

Hovering over a cluster displays some useful information, namely the cluster size, the infected count, the cluster ID and its coordinates.



Cluster panel

The cluster panel can be opened by clicking on a cluster. The cluster panels display a graph showing the evolution of infected persons in that specific cluster, over the simulation. It also displays general information of the cluster.



If multiple clusters are located on the same coordinates, multiple panels get opened with all the overlapping clusters.

Sidebar

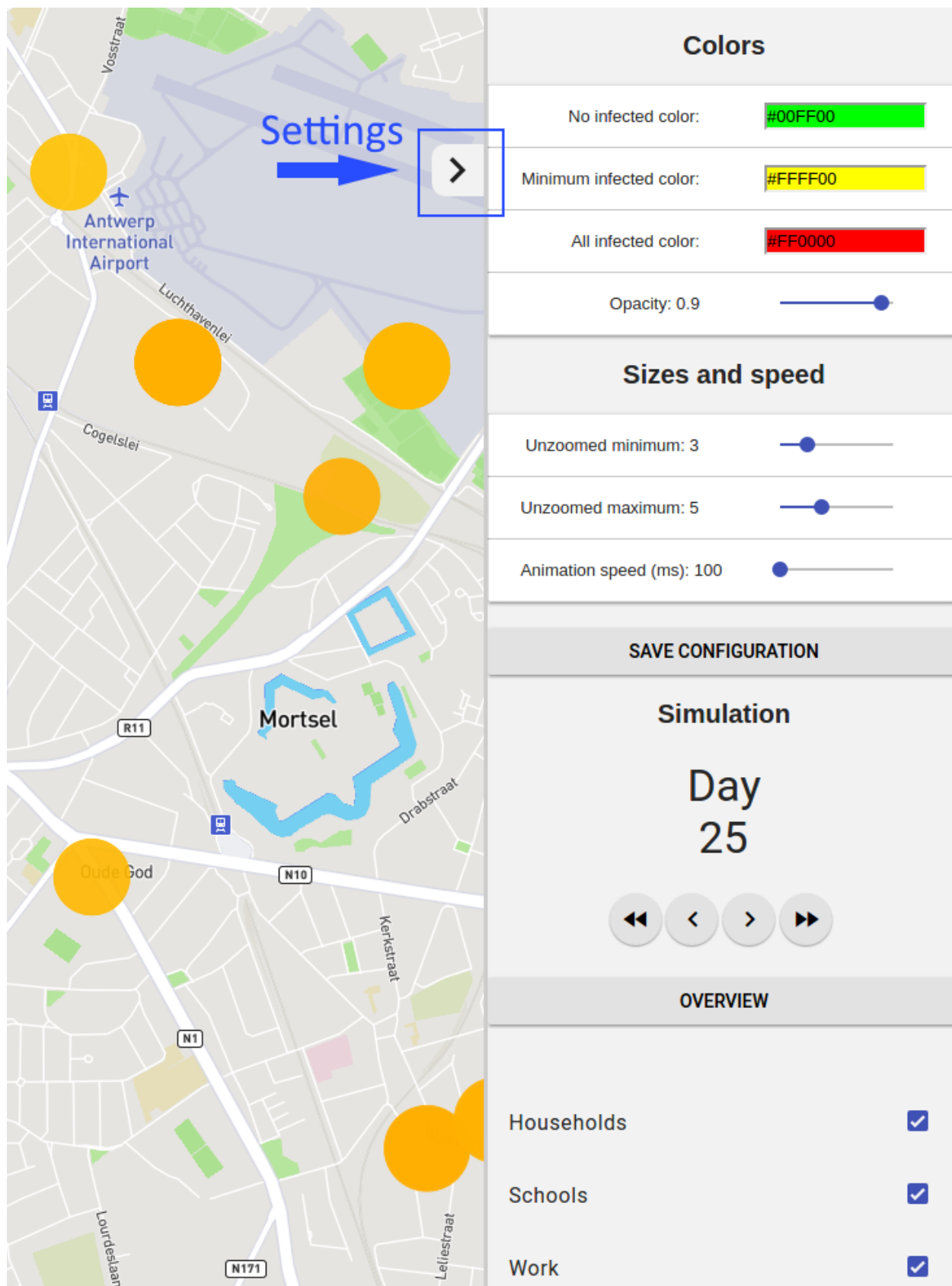
The sidebar can be accessed by clicking on the arrow icon in the top right. The sidebar contains configuration options for the tool and options to display/show different days of the simulation.

The configuration options include:

- The cluster display options such as colors and opacity.

- The sizes of the circles used to display the clusters and the zooming size.
- The animation speed when displaying the different days of the simulation.
- Display options in order to display certain cluster types.

You can save the configuration options after adjusting them, by clicking `SAVE CONFIGURATION`. The saved options will be reused for further uses of the tool.



The screenshot displays the Stride User Manual interface, featuring a map on the left and a settings panel on the right. The map shows a region around Antwerp International Airport, with several orange circles indicating infected locations. A blue arrow points from the 'Settings' label to a button with a right-pointing chevron. The settings panel is divided into several sections:

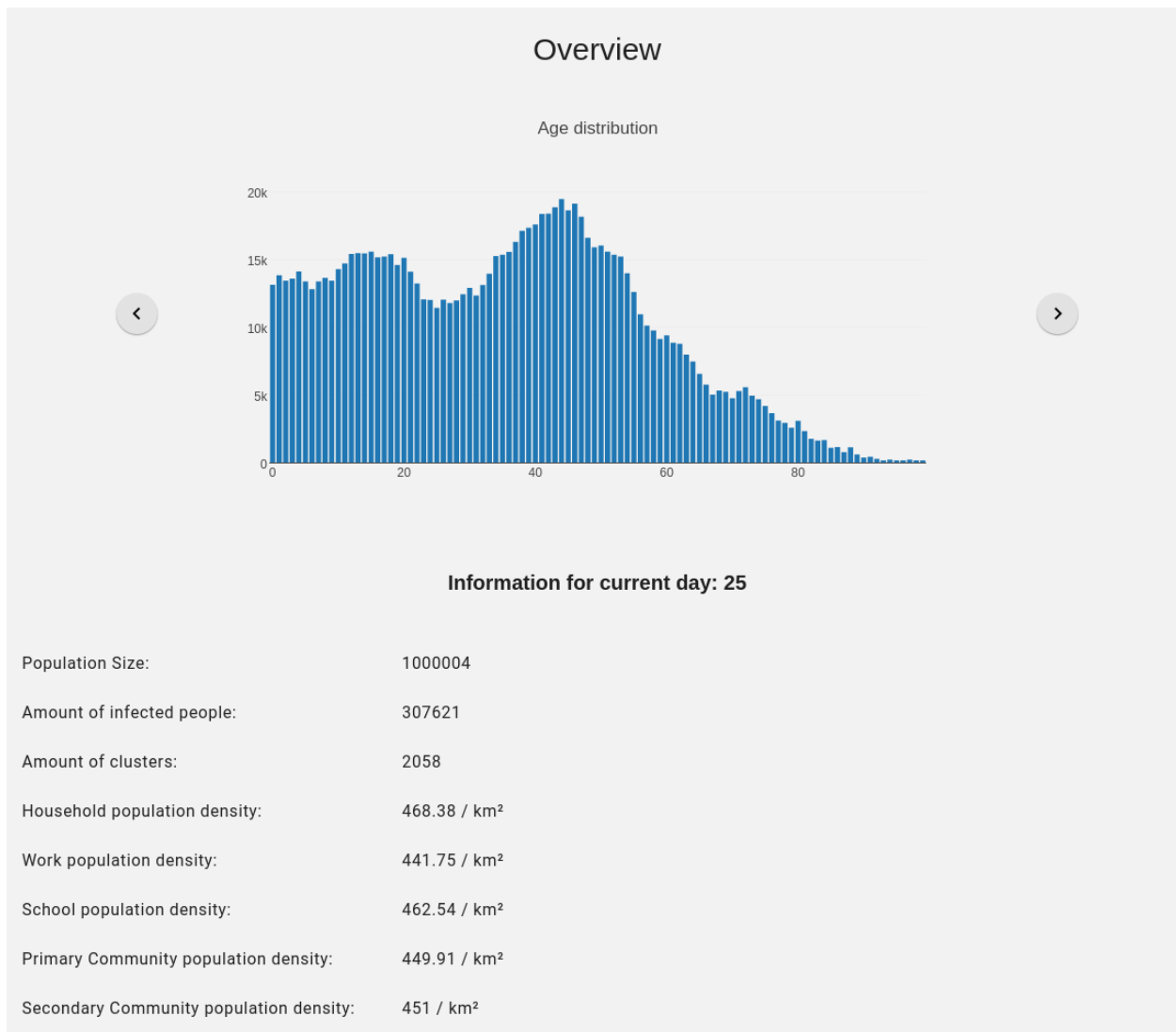
- Colors**
 - No infected color: #00FF00
 - Minimum infected color: #FFFF00
 - All infected color: #FF0000
 - Opacity: 0.9
- Sizes and speed**
 - Unzoomed minimum: 3
 - Unzoomed maximum: 5
 - Animation speed (ms): 100
- SAVE CONFIGURATION**
- Simulation**
 - Day 25
 - Navigation buttons: <<, <, >, >>
- OVERVIEW**
 - Households ☒
 - Schools ☒
 - Work ☒

Airports

If airports are available in your simulation these can also be displayed. Airports show the amount of people that arrived in the last x days or on the previous day. Airports also show the area of influence. This is represented by a circle showing the boundary of the area. The area of influence may shrink or increase during the execution of a simulation.

Overview

The overview can be opened in the sidebar. In the overview you can view specific cluster information, as well as view general information about the population. The general information includes graphs of the age distribution, graphs of the cluster sizes, population density, ...



CHAPTER 7

Bibliography

Bibliography

- [CHOLJ10] Dennis L Chao, M Elizabeth Halloran, Valerie J Obenchain, and Ira M Longini Jr. Flute, a publicly available stochastic influenza epidemic simulation model. *PLoS Comp Biol*, 6(1):e1000656, 2010.
- [GBR+13] John J Grefenstette, Shawn T Brown, Roni Rosenfeld, Jay DePasse, Nathan TB Stone, Phillip C Cooley, William D Wheaton, Alona Fyshe, David D Galloway, Anuroop Sriram, Hassan Guclu, Thomas Abraham, and Donald S Burke. Fred (a framework for reconstructing epidemic dynamics): an open-source software system for modeling infectious diseases and control strategies using census-based populations. *BMC public health*, 13(1):940, 2013.
- [Int14] RTI International. 2010 rti u.s. synthetic population ver. 1.0. *Downloaded from internet URL: <http://www.epimodels.org/midas/pubsyntdata1.do>*, 2014.
- [Whe14] WD Wheaton. 2010 u.s. synthetic population quick start guide. rti international. *Retrieved from http://www.epimodels.org/midasdocs/SynthPop/2010_synth_pop_ver1_quickstart.pdf*, 2014.
- [WST+15] Lander Willem, Sean Stijven, Engelbert Tijskens, Philippe Beutels, Niel Hens, and Jan Broeckhove. Optimizing agent-based transmission models for infectious diseases. *BMC Bioinformatics*, 16:183, 2015.