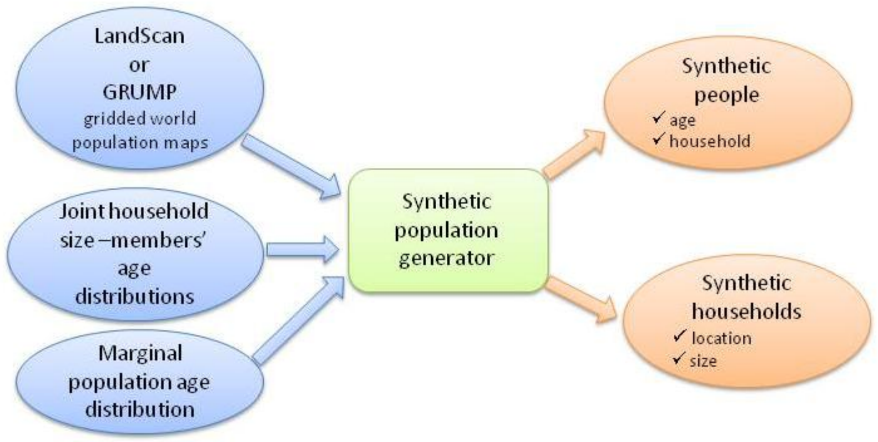
Brief description of population synthesis algorithm

Household structure:

Household sizes are sampled using household size distributions. This data as a rule is easily available from various statistical sources.

Joint household members' age – household size distributions are used for straightforward age sampling. Wherever this type of data is not available, we use averaged distributions. After the sampling, we arrive at the marginal age distribution slightly different from the demographic data. To correct for the mismatch, we use the difference between the model marginal age distribution and statistical data.

The input and output dataflow for the household synthesis task are summarized in the diagram:


LandScan or GRUMP gridded world population maps
Joint household size –members' age distributions
Marginal population age distribution
Synthetic population generator
Synthetic people
✓ age
✓ household
Synthetic households
✓ location
✓ size

Establishment structure:

Population in a given country is divided into a few age groups, so as each age group is associated with a different type of establishment, e.g.:

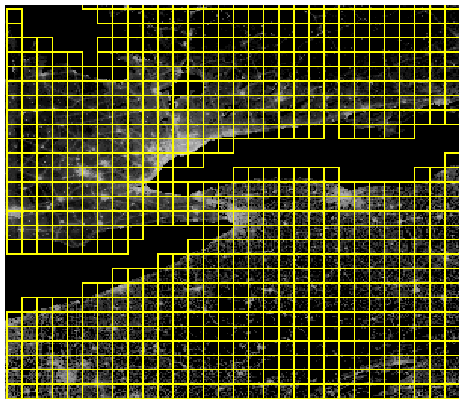
- ✓ 0-4 years: nurseries
- ✓ 5-11 years: primary schools
- ✓ 12-17 years: secondary schools
- ✓ >18 years: workplaces (incl. staff for the above three establishment types)

Each establishment may contain hosts (non-staff) from one age group (e.g. pupils in a school) and staff members drawn from a different age group (e.g. teachers and administrators, etc.)

Establishments are spatially distributed according to the population density data as georeferenced datasets are not available.

Every host chooses an establishment in accordance with a choice kernel and the resulting distance distribution must be matched to available commute survey data. The following choice kernels are used in the current code version:

- ✓ $f(r) = 1/(1+(r/a_0)^{b_0}) + c_1/(1+(r/a_1)^{b_1})$
- ✓ $f(r) = 1/(1+r/a_0)^{b_0} + c_1/(1+r/a_1)^{b_1}$



The algorithm that creates establishments and associates hosts (non-staff) with them combines the rejection-acceptance technique with dynamic establishment allocation. It operates with a probability matrix defined with the choice kernel on a square grid overlay covering the territory of the country (see the figure). This matrix can be considered as a crude representation of the host-to-establishment commute pattern: $P_{kk'} = {}^{1}\!/_{\Sigma_k} \cdot f\left(R_{kk'}\right) N_{k'}$, where $R_{kk'}$ is the distance between patches *k* and *k'* on the grid, $f()$ is the choice kernel, $N_{k'}$ is the number of hosts in patch *k'*, and Σ_k is the normalization factor.

Basic steps of the rejection-acceptance algorithm:

- ❑ For a host *i* in a patch *k*, sample a patch *k'* with $P_{kk'}$
- ❑ Sample an establishment *j* within the patch *k'*
- ❑ Generate a random number rnd in the range $(0,1)$ and check the acceptance condition: $rnd < f\left(r_{ij}^{kk'}\right) / f\left(R_{kk'}\right)$, where $r_{ij}^{kk'}$ is the distance between the host *i* in the patch *k* and the establishment *j* in the patch *k'*.
- ❑ If the acceptance condition is satisfied, assign the host to the establishment. Otherwise, discard the results and start back from step 1.

Establishments are created dynamically to satisfy the demand for vacancies. As an indicator of demand for vacancies within a given patch, we use the number of hits on it while sampling with $P_{kk'}$.

Formal algorithm description:

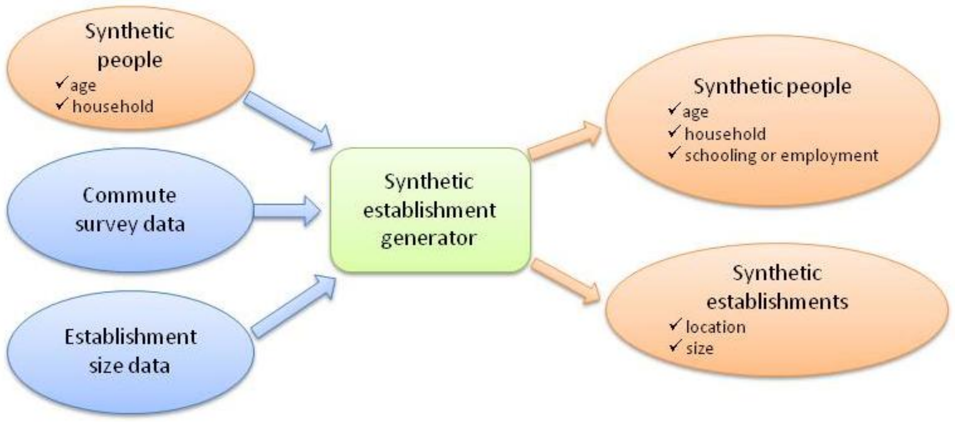
```
while(true)

{

    /* choose random host  from table */

    Host = ChooseRandomHost(HostTable);


    /* if there are no more hosts, terminate loop */
```

```
if( Host == EMPTY )
     break;

/* determine geographical patch where host is located */
GeoPatchHost = WhereIsHost(Host);

while(true)
{
        /* for this patch, sample a geographical patch */
        /*  where establishment may be located */
        GeoPatchEstablishment = SampleGeoPatch(GeoPatchHost );
        if( GeoPatchEstablishment.NumEstablishments == EMPTY )
        {
                 /* if geographical patch contains no establishments, increment hit counter  */
                /* (it represents demand for vacancies in given patch) */
                IncrementHitCounter(GeoPatchEstablishment );

                /* if hit counter does not exceed threshold level, continue */
                /* otherwise set it to zero and create new establishment */
                if(GeoPatchEstablishment.HitCounter < THRESHOLD )
                   continue;
                else
                   Establishment = CreateEstablishment(GeoPatchEstablishment);
        }

        /* determine distance between host and establishment */
        r = HostToEstablishmentDistance(Host, Establishment );

        /* determine distance between patches */
        R = GeoPatchDistance(GeoPatchHost , GeoPatchEstablishment );
```

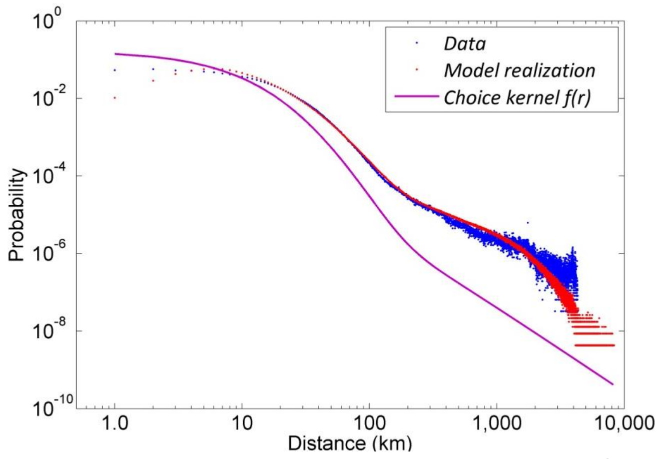
```
            /* check acceptance condition */

            /* if it is satisfied, attempt to assign host to establishment */

            if( Random() < ChoiceKernel(r) / ChoiceKernel(R) )

            {

                        if( Establishment.AssignHost(Host)  == SUCCESS )

                           break;

            }

        }

}
```

The algorithm assigning staff to establishments uses the rejection-acceptance technique similar to the just described one, but without dynamic establishment allocation as all establishments are already in place.

Generating establishments is the final stage in synthesizing population. The input and output dataflow that it involves are portrayed in the diagram:



It is important to note that the resulting commute statistics for the population will be different from the function $f(r)$ that describes a choice of an individual (see the figure on the right). The first reason why it happens is the inhomogeneity in spatial distribution of the population. The second reason is the limited number and capacity of establishments, which leads an individual to the necessity of finding a reasonable compromise between establishment location and its availability while choosing one.



Command line syntax

The command line syntax is:

PopulSynth.exe <LandScan/GRUMP header file name> <LandScan/GRUMP binary dataset file name> <country border raster dataset header file name> <country border raster binary dataset file name> <household size distribution matrix file name> <joint household size-household members' age distribution matrix file name> <marginal age distribution file name> <host-to-establishment travel distribution parameter file name> <establishment size distribution file name> <country code> <scale-down factor> <output file name for the table of synthesized population> <output file name for the table of establishments>

where the command line parameters are:

- ***LandScan/GRUMP header file name***: name of the population density data file header (either LandScan or GRUMP .bil)
- ***LandScan/GRUMP binary dataset file name***: population density data file (in binary format, either LandScan or GRUMP .bil)
- ***country border raster dataset header file name***: self-explanatory (built from the GRUMP data)
- ***country border raster binary dataset file name***: self-explanatory (built from the GRUMP data)
- **country code**: code of a country
- **scale-down factor**: factor to scale down the population size (used to run a pandemic simulation for a smaller model of a country)
- **output file name for the table of synthesized population**: self-explanatory
- **output file name for the table of establishments**: self-explanatory

For the description of the rest of the command line input, see the input file format below.

Input file format

Household size distribution matrix *(represents how probable a given household size is):*

- column1: vector of household sizes
- column2: vector of appropriate cumulative probability distribution values
- column3: vector of appropriate probability distribution values

Joint household size-household members' age distribution matrix *(gives distributions of household members' age stratified by the household sizes):*

- column1: vector of upper age boundaries
- column2: vector of appropriate cumulative probability distribution values for one person households
  ...
- columnN: vector of appropriate cumulative probability distribution values for the household size (N-1)
  ...

Marginal population age distribution matrix *(gives the marginal age distribution for the whole population):*

- column1: vector of upper age boundaries
- column2: vector of appropriate probability distribution values
- column3: vector of absolute population numbers (not currently used)

Host-to-establishment travel distribution parameter file *(list of parameters characterizing commute of people to establishments such as schools, workplaces etc.; note: comments are in grey):*

[0]
num_of_age_groups num_of_param_sets
[1]
age_boundary_0 age_boundary_1 ... age_boundary_max
[2]
for age group 0:
choice kernel function parameters and flags that approximate the real data *(not actually used for population synthesis, but just to compare against simulation results)*
choice kernel function parameters and flags for population synthesis *-- parameter set #0*
...
choice kernel function parameters and flags for population synthesis *-- parameter set #(num_of_param_sets-1)*
[3]
for age group 1:
choice kernel function parameters and flags that approximate the real data *(not actually used for population synthesis, but just to compare against simulation results)*
choice kernel function parameters and flags for population synthesis *-- parameter set# 0*
...
choice kernel function parameters and flags for population synthesis *-- parameter set #(num_of_param_sets-1)*
... ...
[num_of_age_groups+2]
for age group (num_of_age_groups-1):

choice kernel function parameters and flags that approximate the real data *(not actually used for population synthesis, but just to compare against simulation results)*
choice kernel function parameters and flags for population synthesis *-- parameter set #0*
...
choice kernel function parameters and flags for population synthesis *-- parameter set #(num_of_param_sets-1)*

Description of the parameters:

- **num_of_age_groups**: number of age groups the population is divided into
- **num_of_param_sets**: number of different parameter sets (see ***choice kernel function parameters and flags***, the same for each age group)
- **age_boundary_0 age_boundary_1 ... age_boundary_max**: list of age boundaries that determine the population age groups
- ***choice kernel function parameters and flags*** line has the following format:

  choice_kernel_function_code a0 b0 a1 b1 c1 output_flag

  where **choice_kernel_function_code** denotes one of a few predefined choice kernel functions (code 0 corresponds to $f(r) = 1/(1+(r/a_0)^{b_0}) + c_1/(1+(r/a_1)^{b_1})$ and code 1 is for $f(r) = 1/(1+r/a_0)^{b_0} + c_1/(1+r/a_1)^{b_1}$);
  a0, **b0**, **a1**, **b1**, and **c1** are the function parameters;
  output_flag determines whether output files will be produced (if set to 1) or just statistical data will be written in a log file (if set to 0)

Establishment size distribution file *(gives information about establishment size distribution, staff ratio, average workgroup/class size, and fill ratio for each population age group in the model):*

[0] - data block for age group 1
Establishment size distribution matrix:

- column1: **vector of establishment sizes**
- column2: **vector of appropriate probability distribution values**
- column3: **vector of appropriate probability distribution values**

[1]
staff_age_group_indx
staff_ratio
avrg_group_size
fill_ratio
... ...
[n*2 + 0] - data block for age group n
Establishment size distribution matrix:

- column1: **vector of establishment sizes**
- column2: **vector of appropriate probability distribution values**
- column3: **vector of appropriate probability distribution values**

[n*2 + 1]
staff_age_group_indx
staff_ratio
avrg_group_size
fill_ratio
... ...

Description of the parameters:

- Establishment size distribution matrix: self-explanatory
- **staff_age_group_indx**: index of the population age group from which staff members for a given type of establishments are drawn
- **staff_ratio**: number of non-staff/staff ratio for a given establishment type
- **avrg_group_size**: average group size within an establishment (e.g. classes in schools and workgroups in workplaces)
- **fill_ratio**: proportion of hosts associated with establishments in a given age group (e.g. because of the unemployment, not everyone is associated with some workplace)

Output file name convention and format

Name convention:

Files that contain establishment tables have the following name template (one file generated per each age group):

[file name prefix]_[file ID]__[input parameter set ID] (e.g. 'est__USA_0__1'),

where:

- **file name prefix:** can be any!
- **file ID**: file ID (establishment table ID); ID = 0 ... [number of age groups - 1]
- **input parameter set ID**: number of the input parameter set (see the description of ***Host-to-establishment travel distribution parameter file*** above)

Files containing household tables have names fitting the template (one file generated per given parameter set):

[file name prefix]_[input parameter set ID] (e.g. 'popul__USA_1'),

where:

- **file name prefix**: can be any!
- **input parameter set ID**: number of the input parameter set (see the format of ***Host-to-establishment travel distribution parameter file*** above)

Establishment table file format:

The file begins with the header:

```
class EstablishmentTableFileHeader
{
        unsigned short ID1;                    // ID of the file
        unsigned int NumEstablishments;        // total number of establishments
};
```

The header is followed by an array of Establishment records (the number of records, **NumEstablishments**, is given in the header) of the following format:

```
class Establishment
{
        double Lat;                   // latitude establishment coordinate
        double Lon;                   // longitude establishment coordinate
        unsigned int LndScnX;         // Landscan/GRUMP X coordinate of the cell containing the
                                      // establishment
        unsigned int LndScnY;         // Landscan/GRUMP Y coordinate of the cell containing the
                                      // establishment
        unsigned int NumHosts;          // number of hosts associated with the establishment
        unsigned int NumStaff;          // number of staff in the establishment (the total amount of
                                        // individuals associated with the given establishment is then
                                        //(NumStaff + NumHosts)!)
        unsigned int Dummy;             // dummy field; set to -1; (0xFFFFFFFFh)
};
```

Synthetic population table file format:

The file begins with the header:

```
class SynthPopulationFileHeader
{
        unsigned short AgeGroupNum;          // number of age groups in the model
        float AgeGroupLB[AgeGroupNum];       // array of lower age group boundaries
        unsigned int NumHouseholds;          // total number of households
        unsigned int NumHosts;               // total number of hosts
};
```

The header is followed by an array of Household records (the number of records, **NumHouseholds**, is given in the header) of the following format:

```
class Household
{
        double Lat;                          // latitude household coordinate
        double Lon;                          // longitude household coordinate
        unsigned short NumHosts;             // number of hosts in household
        unsigned int Dummy;                  // dummy field; set to -1; (0xFFFFFFFFh)
        Host[NumHosts] HostDescr;            // array of Host records
};
```

where **Host** is:

```
class Host
{
        unsigned char AgeGroupIndex;         // host's age group index
        float Age;                           // host's age
                                             // identifier of an establishment that a host is associated
                                             // with, consists of two parts, ID1 and ID2:
        unsigned short ID1;                  // ID1 (ID of a file that contains a table of  establishments)
        unsigned int ID2;                    // ID2 (index in the table of establishments)
        unsigned short GroupID;              // ID of a group within an establishment that a host
                                             // belongs to (e.g. class in a school or work group in a
                                             // firm); members of staff all have group ID set to -1
                                             // (0xFFFFFFFFh)
};
```