Digital Populations User Manual

Table of Contents

Digital Populations....1
Digital Populations Tutorial 2
The Relationship File GUI 8
Running Digital Populations....14
Data File Reference....18
Fitting Criteria File....21
Relationship File....25
Region Map File....32
Region Attribute Table....33
Land-Use Map....34
Household Sampling (Archetype) File....35
Population Sampling (Archetype) File....36
Parameters File....37
Household Realizations File....40
Population Realizations File....41
Glossary....42
Digital Populations FAQ....43
How to Start Developing 44
ConflatePumsQueryWithTracts Phases 47
Dp2Kml Tool 50
Specifying Coordinate Reference Systems....53
Bubble Templates 54
Placemark Template 58
Csv2Kml Tool 59
CSV Metadata 61
CsvPlusMap Tool 63

Digital Populations

What is Digital Populations?

Digital Populations is a software suite for synthesizing plausible geo-referenced households and people from census or polling data.

Digital Populations starts with statistics and partial data from an actual census, and generates a complete census that resembles the input data. The generated census contains locations and statistics for every household and individual even though the input data contains no location data and only partial statistics.

The original purpose of Digital Populations was to facilitate the detection of statistical clusters. Cluster detection requires a complete population data set, however most governments don't release that data for privacy reasons. Digital Populations can generate configurations of households that can be used for cluster detection. More recently, Digital Populations is being used to to provide rich contextual information of households and people for agent-based social-cultural behavoir models.

Where Do I Go From Here?

Download the suite.

Visit the project site.

Get started developing with the source code.

Read the documentation:

- Introduction
- The Relationship File GUI
- Running Digital Populations
- Data File Reference
- Glossary
- FAQ

- Phases of the CensusGen Algorithm
- Dp2Kml: Viewing the results from Digital Populations in Google Earth
- Csv2Kml: Viewing general CSV files in Google Earth
- CsvPlusMap: Append values from a map to a table as a new column

Digital Populations Tutorial

What is Digital Populations?

Digital Populations is a software suite for synthesizing a plausible population census.

Digital Populations starts with statistics and partial data from an actual census, and generates a complete census that resembles the input data. The generated census contains locations and statistics for every household and individual even though the input data contains no location data and only partial statistics.

The primary purpose of Digital Populations is to facilitate the detection of statistical clusters. Cluster detection requires a complete population data set, however most governments don't release that data for privacy reasons. Digital Populations can generate configurations of households that can be used for cluster detection.

Required Input Data

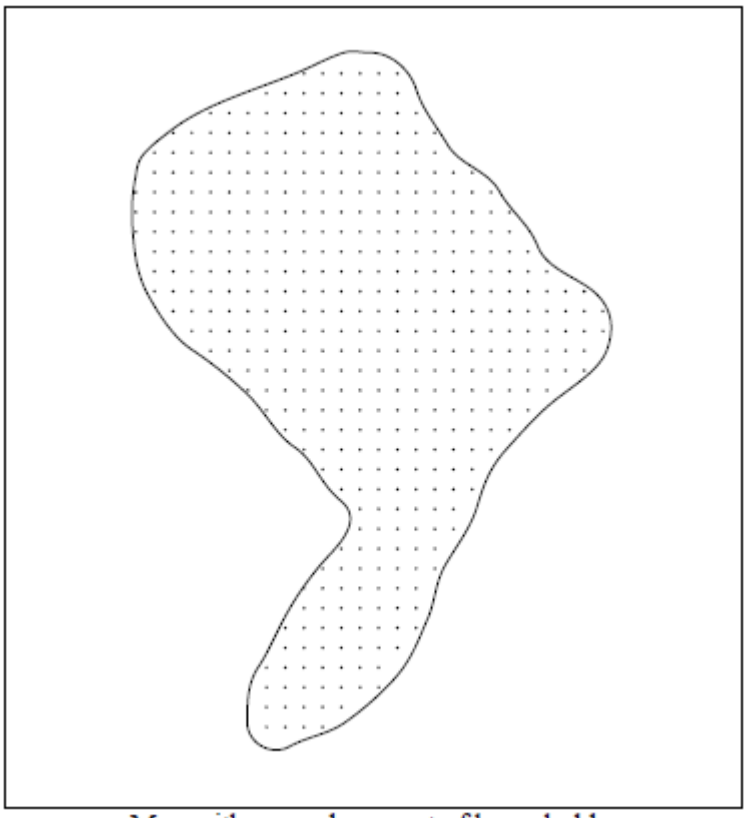
Digital Populations requires input files that describe some portion of an actual census, along with control files that describe the files and specify which statistics are important to the analysis. The files include:

- Tract statistics. Censuses are generally collected by dividing a section of landscape into tracts or regions, then collecting statistics a region at a time. Digital Populations uses this data to guide the generation and placement of households and individuals.
- Partial data sample (aka "public use microdata sample," or PUMS). While governments generally refuse to release the full census data, they will usually release an anonymized subset of the data. Digital Populations clones the households and individuals in this data as needed to produce the full population for the area being analyzed.
- Relationship file. A control file is required to describe the input files and their structure. This file also contains a listing of "traits" which name an interesting characteristic of a households or individual, and specifies how to calculate the characteristic from the region data as well as from the PUMS data. By comparing the two calculations, Digital Populations can determine the quality of its generated census.

Sample Walk Through

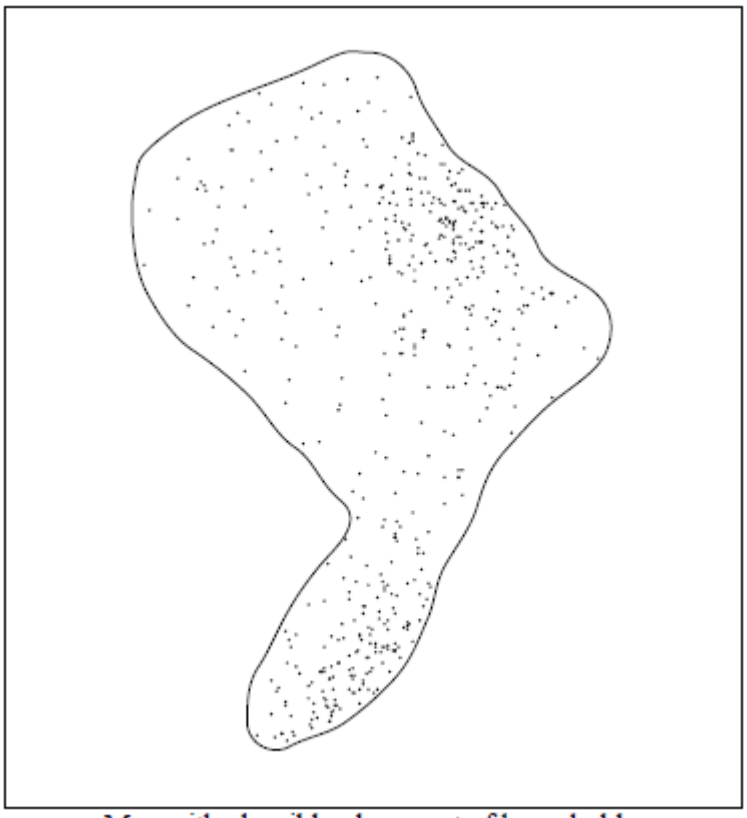
The ultimate goal of Digital Populations is to distribute households across a map in way that resembles an actual census. Since Digital Populations doesn't have complete census data to work with, it starts with random locations, then nudges households into positions that agree with statistical data given as input. This is called a "plausible" arrangement – it's simulated, but resembles an actual census.

Below is a map with a poor placement of households. While it may contain a reasonable number of households, it is not common for them to be uniformly spaced on a grid.



Map with poor placement of households.

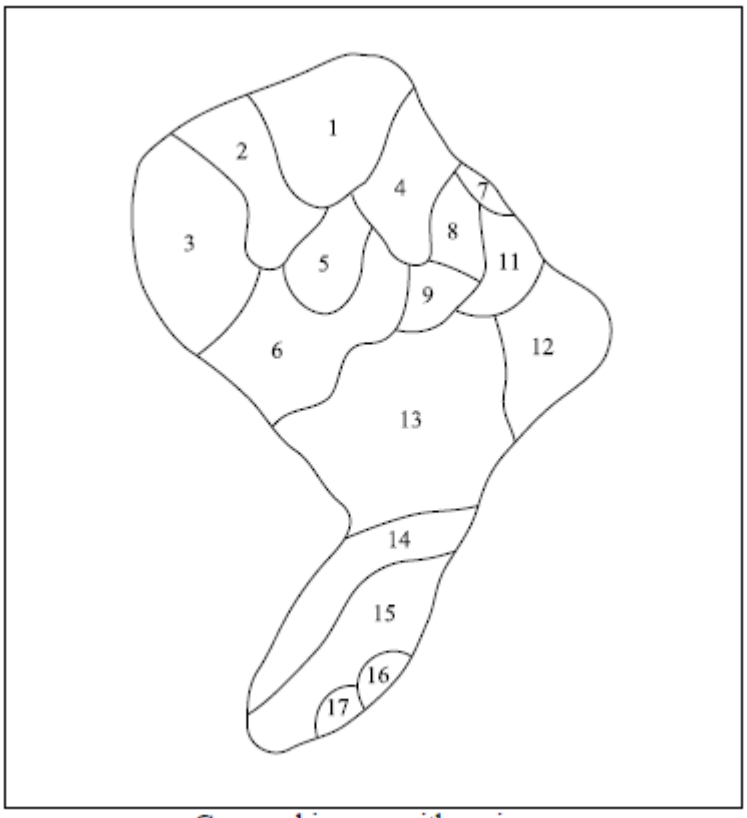
Below is a map with plausible placement of households. While this placement may not be *correct*, it more closely resembles the way houses cluster in the real world.



Map with plausible placement of households.

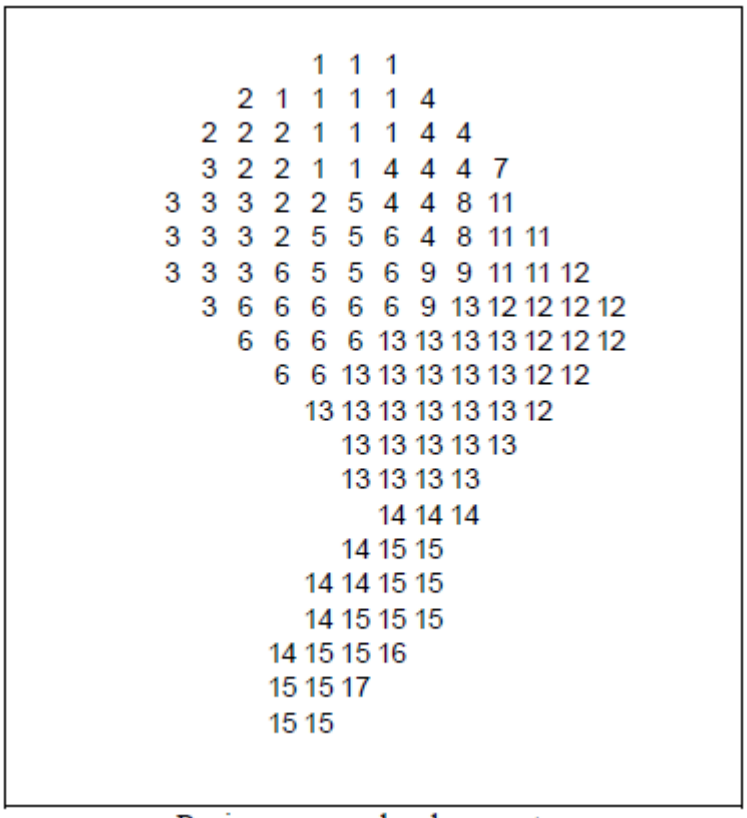
To generate such a result, Digital Populations needs pieces of data from an actual census. Each piece contains only a subset of the data in the census, so Digital Populations computes an arrangement of households that closely matches every input. In essence, the pieces of data collectively form a huge equation, and Digital Populations tries to find the best solution to it.

The first requirement is a set of region statistics. The geographic region to be analyzed must be broken down into small regions, then composite numbers must be provided for each region. Digital Populations works at a resolution of regions – households will be placed into regions so as to best match the statistics, but each household's location within a region will be totally random. Generally, the most convenient regions to use are the tracts defined by a recent government census.



Geographic map with regions.

DP doesn't work with vector maps; it requires a raster version of the region map. The raster map simply contains a region number in each cell.



Region map, rendered as a raster.

A table file provides composite statistics for the regions defined above. Each row in this table defines one region, and lists overall values for the region. DP will use these values as goals, and attempt to place households in a way that best matches them.

Region Number	Rabbits	Mouses	White Animals	Grey Animals
1	23	18	20	21
2	5	15	4	16
3	4	6	5	3
4	104	31	99	36
5	15	23	35	3
etc.				

Partial region statistics for above map.

The second piece of data is a set of archetype households. Generating fake households from scratch is far too complicated, so instead DP takes a set of actual household and population data as input, and uses them as inspiration for the households placed in the final map. The archetype list need not be complete; Digital Populations will duplicate the ones provided as many times as necessary to fully populate the map. The selection and placement of households will be governed by the region statistics

defined above.

Household Number	Dwelling Type	Building Size
1	Apartment	8-unit
2	House	2-story

Sampling of households from the map.

Household Number	Species	Color
1	Rabbit	Grey
1	Rabbit	Grey
2	Mouse	White
2	Mouse	Grey
2	Rabbit	White

Members of above households.

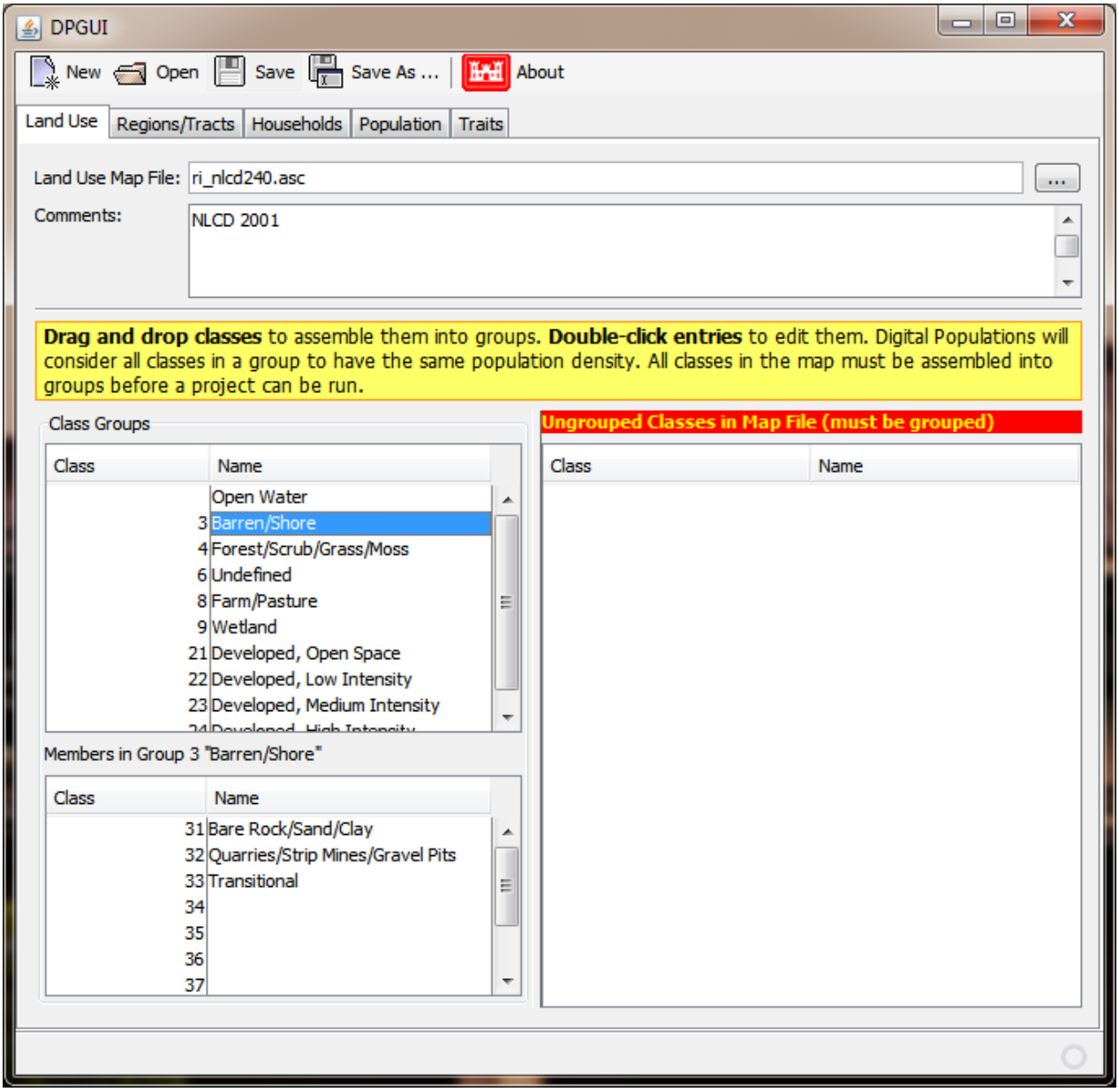
The final piece of data is the control structures that describe the above files, and also specify the exact problem to be solved (i.e. how to compare region goals to copies of the archetypes, and which values are important to a run.) The relationship file lists the files and links them together, and the fitting criteria file selects the important criteria from the relationship file. These files can be created with the GUI, documented below.

The Relationship File GUI

To help create the control files, Digital Populations provides a GUI to help the user select and describe the input files. Operation is as simple as walking through the tabs one at a time, and filling out the fields.

Tab “Land Use”

The Land Use tab specifies a raster map as described above plus simplifying criteria.



Land Use Map File specifies the raster map file itself. Digital Populations doesn't care about the meaning of the groups; it only cares about the population density of each cell in the map. The lower part of this tab is used to create **Class Groups**, where the classes in each group are expected to have similar densities. Digital Populations will compute the actual density value, then try to mimic this value when placing households on the map.

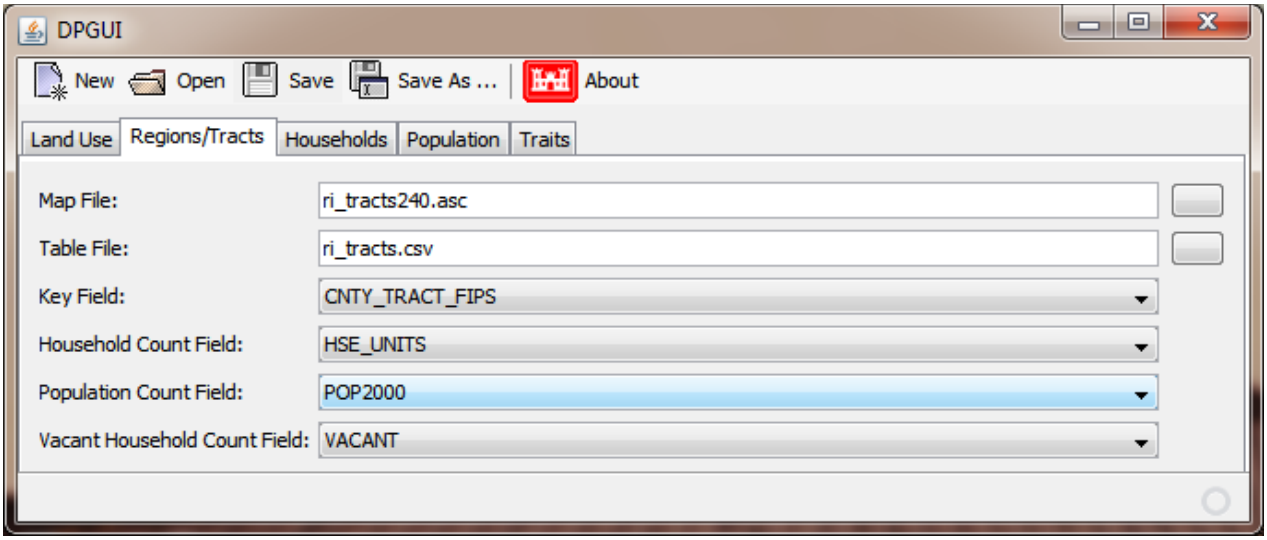
To create a group,simply select one or more classes from the list under **Ungrouped Classes**, then drag them into **Class Groups**. DPGUI will automatically create a group with a default class number and name. Both of these can be freely changed. Selecting a group will display its contents under **Members in Groups**, and classes can be dragged in and out of this list as well.

The first line of Class Groups is the "vacant cell" class, and always implies a total lack of households. Classes that have no residents should be dragged to this group. The name can be changed, but this group has no number.

Digital Populations requires all classes to be grouped before a run, so please ensure the Ungrouped Classes list is empty before using the relationship file.

Tab "Regions/Tracts"

To compute the best location for each household, Digital Populations needs a set of statistics that it can target. It also needs a map defining the regions for which goal statistics are provided. These regions also define the resolution at which Digital Populations will work: households will be placed into regions in a way that closely mimics the statistics, but the exact location of each household within the region will be random.



Map File specifies the raster map that provides the location of each region, and **Table File** provides the composite statistics for each region. Each cell in the map has a number indicating the region it belongs to, and each row in the table provides data for one region. **Key Field** specifies the column within the table file that links the two files, specifying the region for which the row provides data.

Household Count Field specifies the column that gives the total number of households in each region, while **Population Count Field** does the same for individuals. **Vacant Household Count Field** gives the number of households in each region that are unoccupied.

Tab “Households”

Rather than fabricating households from whole cloth, DP instead takes a table of “archetype households” which will be cloned as needed to populate the final map. This table should contain a thorough sampling of real households, as characteristics (combinations or traits of household members) absent from this table will be absent from the output map as well.

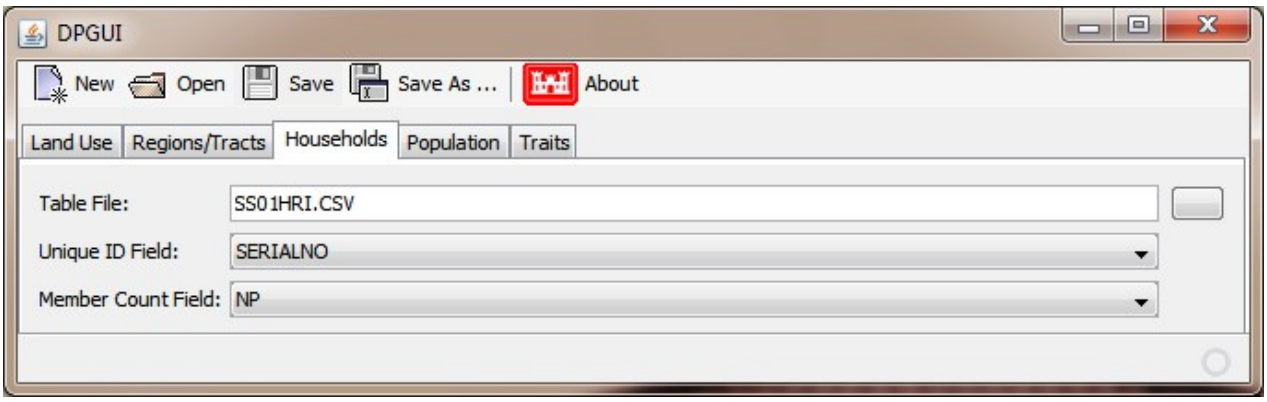


Table File specifies the household table where each row describes one household. **Unique ID Field** gives each household a unique number or code that the population table can refer to. **Member Count Field** specifies the number of people living in each household. DP will compute this count from the population table if provided.

Tab “Population”

Digital Populations also accepts an optional archetype population table that provides specific data for the members of each household.

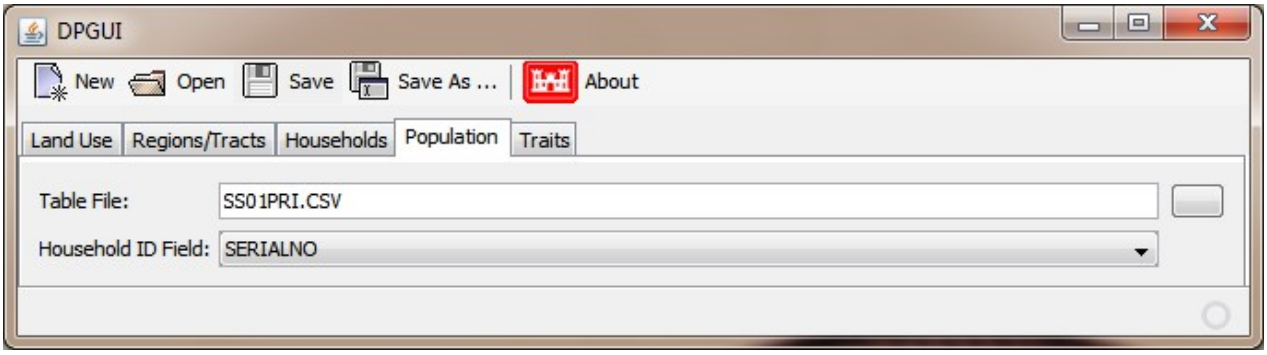


Table File specifies the population table where each row describes one individual. **Household ID Field** provides the value from the household table's Unique ID Field that identifies the specific house

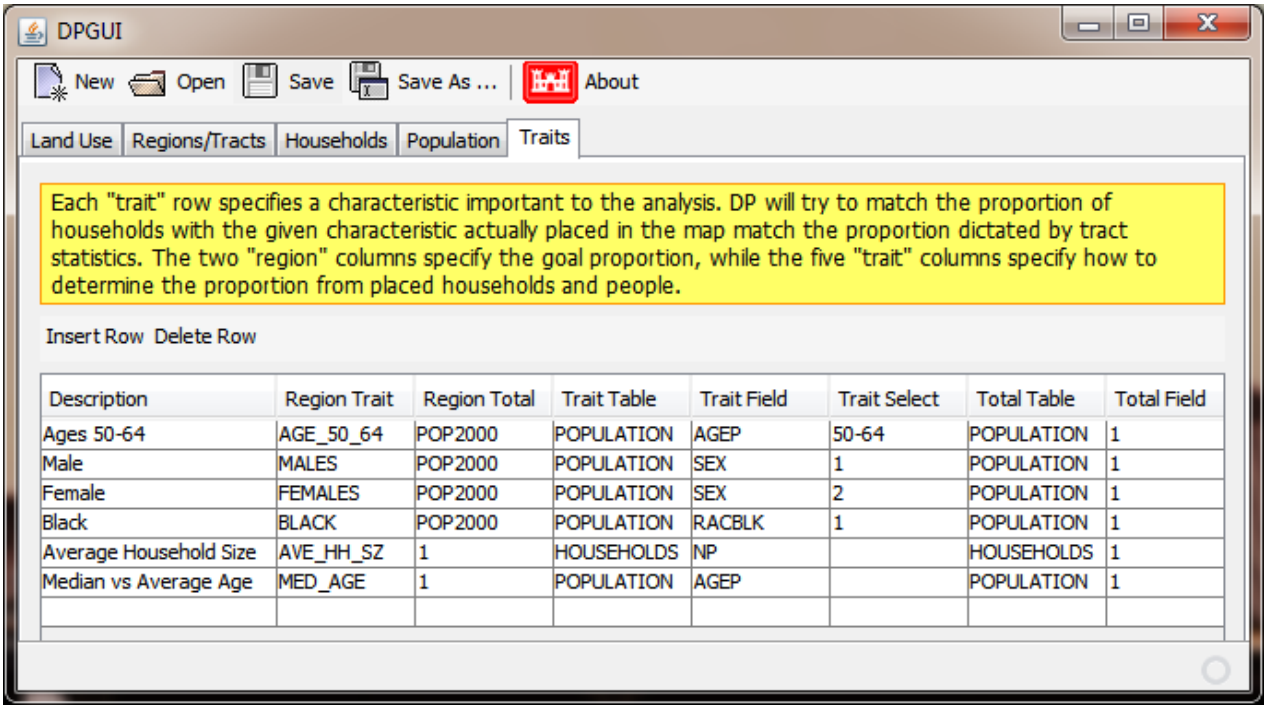
this person lives in. Records without household ID values will be ignored.

Tab "Traits"

The Traits tab is the hardest part of the system to understand, as it describes how the above tables are to be used to generate a new census. Each row here describes a characteristic along with how to calculate the characteristic from the above tables. Digital Populations uses these rows to evaluate the quality of arrangements of households.

Digital Populations starts by internally creating two traits from Household Count Field and Vacant Household Count Field above. These ensure the raw numbers of households are correct, and along with the population density data computed from the land use files, that the households are distributed where the people actually live.

The Traits tab allows the user to specify additional criteria so that the households place actually mirror important characteristics of the actual population being studied. DP does not necessarily use all the criteria listed here; it requires another "fitting criteria" file to select which of these rows will be used in a given run.



Description	Region Trait	Region Total	Trait Table	Trait Field	Trait Select	Total Table	Total Field
Ages 50-64	AGE_50_64	POP2000	POPULATION	AGEP	50-64	POPULATION	1
Male	MALES	POP2000	POPULATION	SEX	1	POPULATION	1
Female	FEMALES	POP2000	POPULATION	SEX	2	POPULATION	1
Black	BLACK	POP2000	POPULATION	RACBLK	1	POPULATION	1
Average Household Size	AVE_HH_SZ	1	HOUSEHOLDS	NP		HOUSEHOLDS	1
Median vs Average Age	MED_AGE	1	POPULATION	AGEP		POPULATION	1

Description is a free-text field that lets the user give each row a useful name.

Region Trait and **Region Total** specify columns from the Regions/Tracts table that together form a target proportion for each region. For example, the second row "Male" specifies the number of males in each region as the numerator, and the total number of people in each region as the denominator, resulting in the percentage of each region that is male. DP will place households in each region so that the proportion of individuals placed that are male is close to this target.

The remaining five columns describe how to compute a proportion from the households placed in a map. **Trait Table** specifies whether the numerator will come from the household table (on the Households tab) or the population table. **Trait Field** specifies which column from that table provides the required value. **Total Table** and **Total Field** work the same way, but provide the denominator.

Trait Select controls the way the values from Trait Field are used. If left blank, Digital Populations will simply add together all the values, and divide by the sum of the Total Field values, effectively providing and average over the households in question. If a list of values or ranges is given here, Digital Populations will instead count the number of objects (households or persons) that have a Trait Field value that is one of the values listed, then divide by the Total Field sum as above.

In the above example, the row labeled Male needs to compute the proportion of a region's population that is male. This is accomplished by counting the number of males, and dividing by the total number of people. The Population table doesn't have a "MALE Y/N" column, but instead provides a "SEX" column containing a code that indicates the gender of each person. So POPULATION/SEX/1 selects all individuals with a SEX value of "1", indicating male. And POPULATION/1 simply counts the number of people in the region.

Trait Field and Total Field can contain a fixed number instead of a field name, and this value will be used instead of the data tables. For example, HOUSEHOLDS/1 will provide a count of the households that DP has placed into a region, and POPULATION/1 will provide a count of the people placed.

Running Digital Populations

The Conflate Command

While Digital Populations provides graphical applications to help edit the control files, the main application must be run from a command prompt. (A tutorial for the command prompt is beyond the scope of this document.) The main application is called "ConflatePumsQueryWithTracts," so named for the technical description of the process used to synthesize a census. The distribution abbreviates this to the command "conflate". Entering this command will produce a brief summary of the command's syntax:

```
C:>CensusGen.bat
Usage:
  ConflatePumsQueryWithTracts [options] fitting-criteria-file

Option                                  Description
------                                  -----------
-c <File>                               configuration (properties file) to
                                          load before parsing command line
                                          (default=last-run.properties)
-f <Integer>                            first realization number (default=1)
-h, --help                              print this help
-l <Integer>                            last realization number (default=value
                                          of -f)
-o <File>                               output directory (default=current dir)
-p, --phase <Integer>                   final phase to execute:  system will
                                          save and exit when this phase
                                          completes (default=3, which skips
                                          phase 4)
--parallel <Integer>                    number of realizations to build
                                          simultaneously (default=1)
-r <Long>                               random seed (default=time-dependent
                                          random)
-s <Double>                             phase 2 skip factor (default=99.5%;
                                          zero is best but slowest)
```

Important points:

- All outputs will be written to the current directory unless "-o" is used to specify another directory.

- Input files can be in any directory. "fitting-criteria-file" must provide the path and name to that file; all other files will be automatically located relative to that file.
- The application can also be in any directory. If not in the current directory, the path to the command must be specified. All of the required pieces of the application will be automatically located relative to that file.
- If "`-h`" appears anywhere on the command line, the above summary will be printed and the application will exit. Running the program with no arguments at all will also print this summary.
- Argument "`-r`" allows the user to specify a custom starting point for the random number generator. Running the program twice with the same seed will produce exactly the same output files.
- Arguments "`-f`" and "`-l`" allow the user to produce multiple realizations in a single run. These can be used to hunt for trends and patterns. Running the program twice with the same realization number and random seed will produce exactly the same output files.
- Argument "--parallel" will run multiple realizations simultaneously, and is useful if the current computer has multiple processors or cores installed. A value of 0 (zero) will use all processors; values greater than zero will use precisely the number of processors given; and values less than zero will reserve that many processor and use only the remainder.

Realizations and Random Numbers

One set of output files is not usually enough for a quality analysis, especially given the quantity of random data that must be injected to produce a synthetic census. A better analysis can be achieved by running the program repeatedly with different random seeds, then examining all the output files for patterns. The user can do this by hand simply by changing the random seed with the "`-r`" argument. But the conflate program contains another feature that helps organize the runs and their output files, and it's known as a "realization number."

The word "realization" is used to refer to a set of concrete outputs derived from some abstract description. In Digital Populations, the inputs form an abstract description of a population (a partial sample plus some general statistics) and the outputs form the concrete realization by providing full details and a precise location for every household.

Since the input files contain no location data at all, Digital Populations uses a random number generator to inject locations for the households. The generator is not truly random; it is fully deterministic, but scrambled in a way that appears random. "Deterministic" means that if two generators are seeded with the same number, they will produce the exact same numbers in the exact same order. Thus if Digital Populations is run twice with the same random seed, the analysis will be the same each time, and the output files will be identical. If conflate is run with the same seed but a

different realization number, then a different set of outputs will be generated. The actual random seed used during a run is a function of the initial seed and the realization number.

To produce a complete set of output files, the user must specify a set of realization numbers to generate. One run of conflate can produce multiple outputs if a range of numbers is requested. Further, multiple computers can be used to produce different realizations simultaneously. Realization numbers need not be consecutive; the user can pick numbers at random if desired.

Some examples:

- "`conflate -r 3 -f 1 -l 10 fitcrit.dpfxml`" will generate 10 sets of output files, numbered one through ten.
- "`conflate -r 3 -f 3 fitcrit.dpfxml`" will generate realization number 3 only, and outputs will be identical to number 3 generated by the above command.
- "`conflate -r 4 -f 3 fitcrit.dpfxml`" will generate another realization number 3, but the outputs will be different from the above commands.

Parallel Operation for Greater Throughput

Computer operating systems support "multi-tasking," which allows a single processor to pretend to run multiple programs simultaneously. But if the more programs a computer runs at once, the slower they all go. This can be solved with "parallel computing," which involves putting multiple physical processors into one computer. And this capability can be expanded through "cluster computing," which uses multiple physical computers connected through a network, working on the same problem. Over the last few years, modern processor chips have gained multiple cores, where each core is a complete processors, but all cores reside on the same chip. This allows a single inexpensive computer to support several times the throughput of the old single-core computers.

Unfortunately, computer algorithms are not so cooperative. Some algorithms can be parallelized to run on several cores while working on one problem. But many algorithms cannot; no matter how many cores are available, only one can be used. This is why the word "throughput" is used instead of "speed" -- two cores can not make one job twice as fast, but if they each run their own copy of the program, they can complete twice as many jobs.

The core algorithms used by Digital Populations fall into the latter group. Conflate can only use one processor at a time. Fortunately, realizations are computed independently from each other. Thus multiple processors can be used to compute multiple realizations simultaneously. And if the files are

copied to other computers or made available through a network, then other computers can be used to compute more realizations.

Making this work in practice is as simple as opening one command prompt for every processor to be used, and running conflate with a unique range of realization numbers. Conflate has been designed so that the output files will not collide with each other, so all runs can write to the same directory.

Note that the system has internal auto-parallelization in the form of the "--parallel" option. This will automatically compute multiple realizations simultaneously according to the above rules. This only works on the current computer; to run on another computer, another copy of the software must be manually started there.

Caveats

Phase 1 of conflate is saved to a file and re-used in later runs. If multiple copies of conflate are started simultaneously, they may all generate their own phase 1. For maximum efficiency, start one copy of conflate, then wait for it to enter phase 2 before starting the others. All other copies can be started simultaneously; only the first one needs special treatment. Note that violating this rule will not introduce errors, it will merely cause the computer(s) involve to perform more work than necessary.

Note also that the result of phase 1 is derived from the random number generator. The file will be regenerated if the seed changes, so it is not recommended to save the results from different seeds to the same directory.

Data File Reference

INPUT FILES

These files must be developed by the user and provided to Digital Populations before it can run.

Fitting Criteria File (fittingcriteria.dpfxml)

Master control file. Specifies dependent file and criteria for one run.

Relationship File (relationship.dprxml)

Specifies a complete set of files for one region and how they relate to each other.

Region Map (tracts.asc)

This raster map provides the location of the regions or tracts that Digital Populations will populate. This is the finest resolution that DP can work with -- a region is selected based on statistical analysis, but the location of a household within a region is random. Coordinate system must be planar (i.e. UTM). DP does not understand coordinate systems, so distance calculations will not be correct in other systems.

Region Attribute Table (tracts.csv)

This table provides aggregate statistics for each region in the region map. The relationship file describes how this table links to the map.

Land Use Map (landuse.asc)

This raster map provides land use data (city, farm, forest, etc.) for the regions in the region map. Digital Populations uses this to guide the random placement of households within a region toward cells where people actually tend to live. Must use exact same coordinate system as the Region Map.

Household Archtype Table (households.csv)

Digital Populations can't create households and people from scratch. Instead, it clones the households from this table in a way that matches the region statistics. The more households provided here, the fewer times any particular archtype needs to be cloned, and the better the result. The relationship file describes how data in this table relates to the statistics in the region table.

Population Archtype Table (population.csv)

This optional table provides residents for the households. Each record identifies which household it resides within, and the residents are cloned when their household is cloned. This allows Digital Populations to compute statistics from personal data.

Parameters File (last-run.properties)

This is a record of all options and settings active the last time Digital Populations was run. It provide access to "hidden" and internal options, and makes it easy to exactly reproduce an older run.

INTERMEDIATE FILES

These files are generated by Digital Populations while it runs. They are not useful by themselves, but they can be used to see how the final results were computed.

Log File (ConflatePumsQueryWithTracts.rzn001.log)

The main Digital Populations application is named "ConflatePumsQueryWithTracts". While running, it dumps a listing of decisions made and progress to a file for the user's later reference. The realization numbers requested are inserted into the file name so that output from separate runs can be save in the same directory. i.e. If realizations five through nine are requested, the file name will contain "rzn#5-9".

Population Density: Solver Matrix (LandcoverPopulationDensity-input.csv)

Digital Populations computes population density per cell of the region map to help guide the placement of households. This file is the matrix derived from the land-use map that is used to compute per-class density data. It's written to this file before solving so that the user can perform their own analysis.

Population Density: Per-Class Density (LandcoverPopulationDensity-landuse.csv)

The result of solving the matrix is written to this file. If this file is specified in the fitting criteria file, Digital Populations will skip the solver and use it directly. Note that class groups (as described in the relationship file) appear here rather than the raw classes. Also, only "good" classes appear in this file; classes that the solver thought were too irregular will be missing, and receive a density of zero.

Population Density: Density Map (LandcoverPopulationDensity-map.asc)

This is the result of applying the per-class density table to the land-use map. If this file is specified in the fitting criteria file, Digital Populations will skip the above processes altogether and use it directly. In this case, the land-use map need not appear in the relationship file at all.

Phase 1 Cache File (`phase1-cache.bin`)

Phase 1 computes the quantity of each household sample that will be required to meet the statistics in the region table. This process can take some time, so the results are written to this file so subsequent runs can use it and skip phase 1 altogether. This file is recomputed whenever the input files change, or the file is missing or corrupt.

Partial Product (`rzn001-households(initial).csv`, `rzn001-population(initial).csv`, `rzn001-signif-households(initial).csv`)

While running, DP will periodically create the output files using the best solution it has at the time. These files are complete, and can be used in place of the final output files, though the result will not be the best possible.

OUTPUT FILES

The final results are written to these files when the application ends.

Household Realizations (`rzn001-households.csv`)

This table provides a precise location for as many households as are required to fully populate the region map. A full set of attributes from the household sample table is copied here, though attributes not relevant to the analysis are zeroed.

Population Realizations (`rzn001-population.csv`)

If a population sample table was provided, the individuals in each household are copied here, along with the location of the household. This table can be used to analyze population distribution directly.

Fitting Criteria File

The fitting criteria file is a custom XML file that points to a relationship file, then provides specific details for a run.

```
<!-- Digital Populations Fitting-Criteria File -->

<FittingCriteria relationshipFile="relationship.dprxml">
  <!--
  | Safety check:  The <traits> section from the relationship file must
  | be copied here exactly.  This allows us to warn the user in case of
  | unexpected changes.
  -->
  <traits>
    <trait id="1"  regionTrait="AGE_50_64" regionTotal="POP2000"
pumsTraitTable="POPULATION" pumsTraitField="AGEP"   pumsTraitSelect="50-64"
pumsTotalTable="POPULATION" pumsTotalField="1" desc="Ages 50-64"/>
    <trait id="2"  regionTrait="MALES"     regionTotal="POP2000"
pumsTraitTable="POPULATION" pumsTraitField="SEX"    pumsTraitSelect="1"
pumsTotalTable="POPULATION" pumsTotalField="1" desc="Male"/>
    <trait id="3"  regionTrait="FEMALES"   regionTotal="POP2000"
pumsTraitTable="POPULATION" pumsTraitField="SEX"    pumsTraitSelect="2"
pumsTotalTable="POPULATION" pumsTotalField="1" desc="Female"/>
    <trait id="4"  regionTrait="BLACK"     regionTotal="POP2000"
pumsTraitTable="POPULATION" pumsTraitField="RACBLK" pumsTraitSelect="1"
pumsTotalTable="POPULATION" pumsTotalField="1" desc="Black"/>
    <trait id="A1" regionTrait="AVE_HH_SZ" regionTotal="1"
pumsTraitTable="HOUSEHOLDS" pumsTraitField="NP"
pumsTotalTable="HOUSEHOLDS" pumsTotalField="1" desc="Average Household Size"/>
    <trait id="A2" regionTrait="MED_AGE"   regionTotal="1"
pumsTraitTable="POPULATION" pumsTraitField="AGEP" pumsTraitContinuous="0-89"
pumsTotalTable="POPULATION" pumsTotalField="1" desc="Median vs Average Age"/>
  </traits>

  <!--
  | Relative importance of traits.  Only relevant trait IDs need be listed here.
  | Each weight must be between 0.0 and 1.0, but don't need to add up to
  | anything in particular.  The attribute "location" provides the weight for
  | the traits that are auto-generated by DP.  Traits can be identified by id,
  | regionTrait (if unique), or desc.
  |
```

```
| Weights may need to be adjusted to get a good output.  It's frequently worth
| tweaking the trait weights to give them an ordering.
|
| location:
|   This is the weight to be used with internally-generated traits, which are
|   generally involved with getting the right numbers of things into regions.
| weight:
|   Each of these assigns a weight value to a trait.
-->
<weights location="1.0">
  <trait id="1" weight="1.0"/>
  <trait id="3" weight="1.0"/>
  <trait id="4" weight="1.0"/>
</weights>

<!--
| Adjust phase 1 results.  Expansion factors are normally computed from
| region data.  If archtype data is more reliable, this can be used to guide
| the numbers in that direction.
|
| trust:
|   Percentage trust in the result of phase 1, in percent. "100" will use the
|   result of phase 1; "0" will use 'factor' and 'std-dev'; and a value
|   between will use a weighted average of the two. (i.e. "43" is interpreted
|   as 0.43 * phase_1 + 0.57 * fixed_factor, where phase_1 is the result of
|   Digital Population's phase 1, and fixed_factor is the value derived from
|   the 'factor' and 'std-dev' attributes.)
| factor:
|   Quantity or name of column to use as expansion factor. If absent, the
|   naive multiplier will be use (e.g. number of households in map divided by
|   number of archtypes in households table.) If a number is provided, that
|   will be the factor for every household. If a name is provided, that
|   column in the households table will provide the factor for each
|   household.
| std-dev:
|   Standard deviation for a random factor. If absent, the default is zero
|   (i.e. the 'factor' attribute will be used as-is.) If present, a random
|   value will be selected whose mean is 'factor' and whose standard
|   deviation is given here.
-->
<expansion-factor trust="50" factor="clones" std-dev="clone-sd"/>

<!--
| Phase 4 computes precise easting/northing locations for each household.
| This section contains the
```

```
  | List of specs that control how we will compute precise locations for each
  | household. Can contain 'cluster' and 'match' elements in any order.
  |
  | <cluster> elements instruct this phase to introduce a degree of clustering
  | by moving "similar" households closer to each other.  The referenced trait
  | provides a definition of "similar", and the remaining attributes specify an
  | amount of clustering.
  |   id:
  |     Identifier for one of the <trait> elements above.  Clustering rules will
  |     be derived from this trait.  Trait must be a normal tract-based trait
  |     (i.e. one with 'regionTrait' and 'regionTotal'.)
  |   reduction:
  |     Percentage reduction in "inertia," which is a measure of the randomness
  |     of an arrangement of households.  A reduction of inertia is an increase
  |     of clustering, and is achieved by moving households closer together.
  |     Goal will be this much lower than the "inertia" values calculated when
  |     phase 4 begins.
  |   distance:
  |     Maximum size of clusters to build. Only houses within this radius will be
  |     considered for clustering.  1000 meters or 0.01 degrees is a good starting
  |     point.  This values is in the same units as the input maps.
  |
  | <match> elements instruct the phase to use a referenced trait exactly as
  | specified.  These elements are considered constraints, and households will
  | only be moved to locations permitted by all <match> elements together.  If
  | the system is over-specified and the elements collectively forbid all
  | locations in a tract, then the 'rank' field is used to find an acceptable
  | solution.

  |   id:
  |     Identifier for one of the <trait> elements above.  Trait must be an
  |     attribute map (i.e. with 'attribute' field) as this is the only kind
  |     of trait that has enough precision to work here.
  |   rank:
  |     Optional integer that specifies the order in which <match> elements are
  |     to be used.  Elements are processed in this order:
  |       * First, elements with 'rank', and in numeric order (low to high).
  |       * After, elements without 'rank', and in the order they appear in
  |         the fitting-criteria file.
  |
  | Note that <match> elements are order-independent -- the same
  | result is produced no matter the order in which these elements are
  | applied.  The 'rank' field is only used when attribute maps are over
specified,
  | and the system cannot find a location that satisfies all <match>
  | elements simultaneously.  In that case, only the top elements will be
```

```
  | used, and evaluation will stop with the first element that breaks.
  -->
  <position-rules>
    <cluster id="1" reduction="50" distance="1000"/>
    <cluster id="3" reduction="50" distance="1000"/>
    <cluster id="4" reduction="50" distance="1000"/>

    <match id="19"/>
  </position-rules>

</FittingCriteria>
```

Relationship File

The relationship file is a custom XML file that lists all the files relevant to a geographical are and describes how they relate to each other. Which bits of data DP will use from this file are given in the fitting criteria file.

```
<!-- Digital Populations Data Relationship File -->
<FileRelationship>
  <!--
  | The land use map guides where households can and cannot be placed.
  | <combination> elements group classes that are expected to have similar
  | population density to simplify analysis.
  |
  | All classes in map must be covered by either <vacancy> or a <combination>
  | element.  This might seem strict, but we have no other data dictionary
  | for the map.
  -->
  <landuse map="ri_nlcd240.asc">
    <comment>NLCD 2001</comment>
    <vacant desc="Open Water" classes="11"/>
    <combination desc="Wetland" target="9" classes="90-99"/>
    <combination desc="Farm/Pasture" target="8" classes="81-89"/>
    <combination desc="Forest/Scrub/Grass/Moss" target="4" classes="41-60,71-80"/>
    <combination desc="Undefined" target="6" classes="61-70"/>
    <combination desc="Barren/Shore" target="3" classes="31-40"/>
    <combination desc="Developed, Open Space" target="21" classes="21"/>
    <combination desc="Developed, Low Intensity" target="22" classes="22"/>
    <combination desc="Developed, Medium Intensity" target="23" classes="23"/>
    <combination desc="Developed, High Intensity" target="24" classes="24"/>
  </landuse>


  <!--
  | DP needs a population density map (aka. population density function, or
  | PDF) to determine where the households need to go.  DP normally computes
  | the map by analyzing the distribution of land-use classes in each region
  | vs. the number of occupants in the region.  The analysisCsv2Kml Tool  is very
  | rudimentary however, thus this element can be used to bypass it and
  | provide density data directly.
```

```
|
| Only one of the attributes can be specified:
|  * 'map' specifies a finished PDF, ready to use.  The map must provide
|    values for every cell in every region covered by the region map.  Each
|    cell in this map specifies a relative density for that patch of ground.
|  * 'landuse' specifies a CSV table that provides a relative density
|    for each land-use class code.  DP will build a PDF by substituting the
|    values into the appropriate cells in the land-use map.
|
| "Relative density" simply means a proportion, using the same scale for
| all cells.  Thus a cell with the value 3.2 will receive twice as many
| households as one with the value 1.6.
|
| <popdensity map="custom-pdf.asc"/>
| <popdensity landuse="LandcoverPopulationDensity-result.csv"/>
-->
<!--popdensity landuse="../LandcoverPopulationDensity-landuse.csv"/-->
<!--popdensity map="../LandcoverPopulationDensity-map.asc"/-->


<!--
| Aggregate table defines total counts of population characteristics
| for each region defined in a map.
|
| NOTE:  Exactly one <regions> element must provide
| households/population/vacancies attributes.  This is considered the
| "primary" region map, and will provide those numbers to the system.  Other
| elements must NOT provide these attributes, though a similar analysis can
| be requested with custom <trait> elements.
|
| 'id' is a unique ID for this map/table pair.  Optional if there is only one
|   <regions> element; mandatory if two or more.
| 'map' is the ESRI ASCII map file that provides the locations of all regions.
| 'table' provides attributes for every region.
| 'key' links the map and the table.
| 'households' gives the number of households in each region.
| 'population' (OPTIONAL) gives the number of people in each region.
| 'vacancies' (OPTIONAL)  gives the number of empty households in each region.
-->
<regions id="tracts"
  map="ri_tracts240.asc" table="ri_tracts.csv" key="CNTY_TRACT_FIPS"
  households="HSE_UNITS" population="POP2000" vacancies="VACANT"/>


<!--
```

```
| Random sampling of households in the area covered by the region map.
|
| 'table' names the file to load, relative to this relationship file.
| 'key' names the column that provides a unique ID for each record.
-->
<households table="SS01HRI.CSV" key="SERIALNO" members="NP"/>


<!--
| OPTIONAL, only required if <traits> need it.
| Random sampling of people in the area covered by the region map.
|
| 'table' names the file to load, relative to this relationship file.
| 'household' names the column that provides a link to a record in
|   the households table.
-->
<population table="SS01PRI.CSV" household="SERIALNO"/>


<!--
| The trait table describes how to relate specific households and people
| to the raster maps.  Households will be randomly placed in a way that
| agrees with map statistics.  Each element describes which maps to use
| and how to use them.
-->
<traits>
  <!--
  | The "trait" element relates households and people to population maps.
  | Statistics are computed over geographic regions, and households are
  | placed into regions to match the statistics as closely as possible.
  |
  | "Aggregate data" comes from per-region numbers in the attribute table
  | attached to the region map.  One column is chosen as "trait", and
  | contains the number of people in each region with an interesting
  | characteristic.  A second column is chosen as "total", and provides
  | the number of people in each region that are candidates for the trait.
  | This is usually just the total number of people in the region, but
  | sometimes a subset is desired.  When "trait" is divided by "total",
  | the result is the target proportion for the region.
  |
  | Household and population data comes from the households that are
  | actually placed in the map.  Again, one column is specified as "trait"
  | and one as "total".  When divided, this provides a statistic that can
  | be compared to the goal proportion computed above.
  |
```

```
| General attributes:
|   id           - Unique identifier that is used in the fitting criteria
|                  file.
|   desc         - Long description for user's reference.
|
| Goal values from region data (if used, do NOT use attribute* fields):
|   regionTable  - Selects the region map (i.e. <regions> element) from which
|                  data will be drawn by this trait.  Optional if there is
|                  only one <regions> element; mandatory if two or more.
|   regionTrait  - Which field in region table gives number of items in
|                  region with trait.  Used as numerator for target
|                  proportion.
|   regionTotal  - Which field in region table gives total number of
|                  candidates for trait.  Used as denominator for target
|                  proportion.  Specify "1" if regionTrait is already the
|                  target proportion.
|
| Goal values from attribute map (if used, do NOT use region* fields):
|   attribute       - ID of attribute map to use as goal data.
|   attributeSelect - Subset of attribute map values considered acceptable.
|                     Format is a comma-separated list of numbers and ranges,
|                     i.e. "1,4,7-9".
|
| Specific numerator:
|   pumsTraitTable      - Which type of object we're analyzing: HOUSEHOLDS
|                         or POPULATION.  "Pums" data fields will come from
|                         the corresponding table.  If POPULATION, data will
|                         be summed per household, then summed per region.
|                         If HOUSEHOLDS, data will be summed per region.
|   pumsTraitField      - Which field contains trait data for each object.
|   pumsTraitSelect     - Switches to 'flag' mode (see below.)
|   pumsTraitContinuous - Enables no-data capability (sele below.)
|
| Specific denominator:
|   pumsTotalTable      - Type of object to draw totals from: HOUSEHOLDS
|                         or POPULATION, as above.  If absent, defaults to
|                         1.0 per region.  i.e.
|                         No proportion will be calculated; the trait value
|                         will be used directly.
|   pumsTotalField      - Which field contains total values for each object.
|
| Use cases:
|   Careful selection of the above attributes controls how digital
|   populations sums and analyzes the data.  There are three major "modes" of
|   use:
```

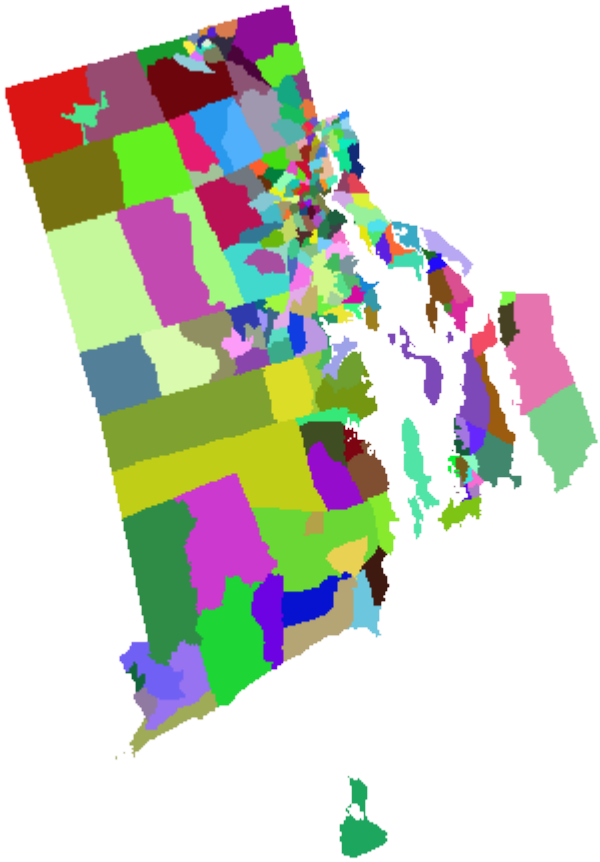
```
    |
    |   CONTINUOUS MODE:  For each object of requested type, the trait fields are
    |     fetched and summed together.  Total fields are seperately fetched and
    |     summed, and dividing the two yields the "proportion" value for the
    |     objects.
    |   CONTINUOUS NO-DATA FEATURE:  pumsTraitContinuous can contain a single
    |     numeric range.  Objects are totaled as above, but only where the trait
    |     is in the given range.  If not, neither the trait sum nor the total is
    |     updated.  In other words, objects with values outside the range are
    |     considered to have "no-data", and are ignored when computing
    |     proportions.
    |   FLAG MODE:  pumsTraitSelect can contain a list of numbers and number
    |     ranges (i.e. "1,2,4,7,9-11,17-29".  For each object, the trait field is
    |     fetched, and if the field has a value in the list, the object is counted
    |     as "1".  If not, it's "0".  These are summed to yield a count of
    |     objects, and this count acts as numerator.  The denominator (total
    |     value) is as above.
    |
    | Automatic traits:
    |   DP automatically generates two traits from the file and field
    |   specifications above.  Even if the <traits> element is empty, DP will
    |   still be able to produce some basic matching.  These traits will appear
    |   in the log file before any of the user traits.
    |    * Automatic Trait #1: Absolute Households Per Region
    |      Ensures the proper number of empty houses appear in each region.
    |    * Automatic Trait #2: Vacant Households
    |      Ensure the proper percentage of vacant households appear in each
region.
    |    * Automatic Trait #3: Absolute Population Per Region
    |      Ensures the proper number of people appear in each region.
    -->
    <trait id="1"  regionTrait="AGE_50_64" regionTotal="POP2000"
pumsTraitTable="POPULATION" pumsTraitField="AGEP"   pumsTraitSelect="50-64"
pumsTotalTable="POPULATION" pumsTotalField="1"  desc="Ages 50-64"/>
    <trait id="2"  regionTrait="MALES"     regionTotal="POP2000"
pumsTraitTable="POPULATION" pumsTraitField="SEX"    pumsTraitSelect="1"
pumsTotalTable="POPULATION" pumsTotalField="1"  desc="Male"/>
    <trait id="3"  regionTrait="FEMALES"   regionTotal="POP2000"
pumsTraitTable="POPULATION" pumsTraitField="SEX"    pumsTraitSelect="2"
pumsTotalTable="POPULATION" pumsTotalField="1"  desc="Female"/>
    <trait id="4"  regionTrait="BLACK"     regionTotal="POP2000"
pumsTraitTable="POPULATION" pumsTraitField="RACBLK" pumsTraitSelect="1"
pumsTotalTable="POPULATION" pumsTotalField="1"  desc="Black"/>
    <trait id="5"  regionTrait="AVE_HH_SZ" regionTotal="1"
pumsTraitTable="HOUSEHOLDS" pumsTraitField="NP"
pumsTotalTable="HOUSEHOLDS" pumsTotalField="1"  desc="Average Household Size"/>
```

```
    <!--
    | Average age example
    | (Note: our current tract table has 'median age', but DP can only
    | calculate averages.)
    -->
    <trait id="6"
      desc="Median vs Average Age"
      regionTrait="MED_AGE" regionTotal="1"
      pumsTraitTable="POPULATION" pumsTraitField="AGEP" pumsTraitContinuous="0-89"
      pumsTotalTable="POPULATION" pumsTotalField="1"/>

    <!--
    | Automatic traits example.
    | These are the traits that will be auto-generated by DP for this file.
    |
    <trait id="Auto-1"
      desc="Absolute Households (AUTO)"
      regionTrait="HSE_UNITS" regionTotal="1"
      pumsTraitTable="HOUSEHOLDS" pumsTraitField="1"/>
    <trait id="Auto-2"
      desc="Vacant Households (AUTO)"
      regionTrait="VACANT" regionTotal="HSE_UNITS"
      pumsTraitTable="HOUSEHOLDS" pumsTraitField="NP" pumsTraitSelect="0"
      pumsTotalTable="HOUSEHOLDS" pumsTotalField="1"/>
    <trait id="Auto-3"
      desc="Absolute Population (AUTO)"
      regionTrait="POP2000" regionTotal="1"
      pumsTraitTable="HOUSEHOLDS" pumsTraitField="NP"/>
    -->

    <!--
    | <forbid> explicitly disallows the placement of certain households into
    | certain map cells.  Households are selected based on their attributes,
    | and map cells are described by a custom map file.
    |
    | Households with attributes matching the household selector will not be
placed
    | in any cells described by map cell selector. Households that do not match
    | household selector will be ignored by this constraint, allowing them to be
    | placed anywhere. Cells that do not match the map selector will be allowed to
    | receive any kind of household. Only selected households will be constrained,
    | and they will be constrained only from the selected cells.
    |
    | All parameters are mandatory.
    |
```

```
    | General attributes:
    |   id           - Unique identifier that is used in the fitting criteria
    |                  file.
    |   desc         - Long description for user's reference.
    |
    | Map cell selector:
    |   map          - Raster map that provides cell attributes.  Value is a
    |                  path and name of an ESRI ASCII file, relative to this
    |                  fitting criteria file.
    |   mapSelect    - Map cell values to match.  Selected households will NOT
    |                  be placed in any cells where given map has this value.
    |                  Format is a comma-separated list of numbers and ranges,
    |                  i.e. "1,4,7-9".
    |
    | Household selector:
    |   pumsTraitTable      - Which type of object we're analyzing: HOUSEHOLDS
    |                         or POPULATION.  Same as for <trait>.
    |   pumsTraitField      - Which field contains trait data for each object.
    |   pumsTraitSelect     - Only households where pumsTraitField has these
    |                         values will be constrained. Format is a comma-
    |                         separated list of numbers and ranges,
    |                         i.e. "1,4,7-9".
    |
    <forbid id="f1"
      desc="Electrified houses require electricity."
      map="electrified_areas.asc" mapSelect="0"
      pumsTraitTable="HOUSEHOLDS" pumsTraitField="typeLight" pumsTraitSelect="1"/>
    -->

  </traits>
</FileRelationship>
```

Region Map File

The region map is a standard ESRI ASCII raster map that provides locations for every region or tract of interest.



Region Attribute Table

The region table provides attributes for every region in the region map. The file is formatted as a comma-separated-value text file, which can be created by any spreadsheet program (i.e. Microsoft Excel.) The first row must contain column names; all subsequent rows contain data. DP mandates a few bookkeeping columns, but other columns are required to perform an analysis. All column names must be specified in the relationship file.

This table acts as the source of goals for the analysis: Households will be moved around until they match the values provided for each region here.

- Map-table linkage column: One of the columns must match the values that appear in the region map. This column is used to link a row to each region in the map.
- Household count: One column must specify the total number of households in each region.
- Population count: One column must specify the total number of people in each region.
- Vacant household count: One column must specify the number of unoccupied households in each region.
- Statistic columns: Other columns can provide aggregate statistics. These columns must be listed in the relationship file as "traits".
- Other columns: Columns not listed in the relationship file are ignored by DP.

Land-Use Map

The land-use map is a standard ESRI ASCII raster map that provides a land-use data for the region being analyzed. This map is used to compute a population density map to guide the placement of households. If a population density map is provided instead, this map is not necessary.

Household Sampling (Archetype) File

The household sample file (also known as archetype or PUMS (Public-Use Microdata Samples)) contains data for the households that will be sprinkled over the regions. This table contains one row for each household, and each row specifies specific attributes for that household.

DP only mandates a few columns, but other columns are required to perform an analysis. All column names must be specified in the relationship file.

- Unique ID column: (optional) Each row may have a unique ID. Any arbitrary string can be used. If absent, the system will generate a unique number for each row, starting from 1.
- Number of members: Each row must specify the number of people in that household. If a population table is provided, this field will be ignored.
- Statistic columns: Other columns can provide aggregate statistics. These columns must be listed in the relationship file as "traits".
- Other columns: Columns not listed in the relationship file are ignored by DP.

All values are stored as integers, with some constraints:

- The NODATA value is hard-coded in the system as -999999. If this value appears in a cell, it will be considered to be undefined. What effect this has depends on the nature of the statistic being evaluated.
- Non-numeric values and numbers outside the range of "integer" are replaced with NODATA, and considered to be undefined.
- Non-integer numbers in this table are rounded to an integer randomly, with a probability equal to the fractional portion of the number. i.e. The value 3.4 has a 60% chance of being rounded to 3, and a 40% chance of being rounded to 4.
- The ID column is a string, and is handled specially by the system.

Population Sampling (Archetype) File

The population sample file (also known as archetype or PUMS (Public-Use Microdata Samples)) contains data for the individual members of each household. The table is optional, and can be skipped if the relationship file only makes use of household statistics.

DP only mandates a few columns, but other columns are required to perform an analysis. All column names must be specified in the relationship file.

- Household ID column: Each row must identify the household to which it belongs. This value must match one from the unique ID column in the household table.
- Statistic columns: Other columns can provide aggregate statistics. These columns must be listed in the relationship file as "traits".
- Other columns: Columns not listed in the relationship file are ignored by DP.

All values are stored as integers, with the same constraints as in the households table.

Parameters File

ConflatePumsQueryWithTracts saves a file named "last-run.properties" every time it runs. This file contains every option that was active when it ran. This serves as a record of precisely how the last run was configured.

It also loads this file when run, to serve as a source of default values. If no other arguments are given, then it will re-run the analysis exactly as it was configured last time. If any command-line arguments are given, these will replace the settings from this file, and the modified configuration will run. This configuration will then be saved as a new "last-run.properties".

Do not edit this file while the program is running, as the program will overwrite your changes when it exits. If you want to keep customized files separate from last-run.properties, save the file under a different name, then use the -c argument to run it.

Order of Loading

ConflatePumsQueryWithTracts loads its configuration store in a particular order.

1. **Defaults.** A set of default values are installed first.
2. **last-run.properties** is then loaded from the current directory, if it exists. Any values found override the defaults installed above.
3. The **-c argument** is processed next, if present. The named file is loaded, and any values found within override those installed above.
4. The **command line** is parsed last. Any options given explicitly will override all of the above sources. Thus the program will always run the job specified on the command line. Only missing options will be drawn from the above sources.

Parameter keys:

- **criteria_file**: path and name of main fitting criteria file, relative to current directory
- **do_dump_number_archtypes**: If true, the log file will receive the results of phase 1.
- **do_dump_statistics**: If true, the log file will receive detailed reports from the quality evaluation objects between phases.
- **do_write_all_hoh_fields**: If false, each record in households.csv will contain coordinates plus

a reference to one of the archtypes. If true, each record will also contain a full copy of the archtype record.

- **do_write_all_pop_fields**: Same, but for population records.
- **first_rzn_num**: (argument -f) Index of first realization to generate. All realizations between 'first' and 'final' will be run and saved. In other words, ConflatePumsQueryWithTracts will run repeatedly, once for each realization, but with different random seeds each time.
- **final_rzn_num**: (argument -l) Index of final realization to generate.
- **initial_seed**: (argument -r) Random seed that will guide the creation of all random numbers. Running the program twice with the same seed will result in the exact same output. Delete from file to make program generate a new random seed.
- **only_one_region**: This is a speed hack that causes algorithm to use only the first census tract. The program runs very quickly, but the output is useless.
- **output_dir**: (argument -o) Target directory for all output files, relative to current directory.
- **phase1_time_limit**: Limit phase 1 to this many minutes.
- **phase2_random_tract_prob**: (argument -s) This fraction of households will be placed randomly, regardless of whether the placement is a good fit or not. Setting to zero will greatly improve the output of phase 2, but also take a great deal longer.
- **phase2_tract_skip_prob_init**: (argument -s) This fraction of tracts will be ignored when phase 2 tries to place a household into a tract. If phase2_random_tract_prob is triggered, then this setting is not used. Setting this to zero will make phase 2 properly evaluate all tracts, but take a great deal longer.
- **phase2_tract_skip_prob_delta**: If phase2_tract_skip_prob_init caused phase 2 to skip every tract, then the fraction will be reduced by this amount, and placement will be tried again.
- **phase3_save_intermediate**: Phase 3 will write a set of files every this many minutes. Phase 4 also uses this timer for the same purpose.
- **phase3_skip**: If true, phase 3 will be skipped altogether.
- **phase3_time_limit**: Limit phase 3 to this many minutes.
- **phase4_skip**: If true, phase 4 will be skipped altogether.
- **phase4_time_limit**: Limit phase 4 to this many minutes.

```
#Last run of ConflatePumsQueryWithTracts
#Fri Oct 01 00:00:22 PDT 2010
criteria_file=data\\fittingcriteria.dpfxml
do_dump_number_archtypes=true
do_dump_statistics=true
do_write_all_hoh_fields=true
do_write_all_pop_fields=true
final_rzn_num=1
first_rzn_num=1
initial_seed=283
only_one_region=false
output_dir=.
phase1_time_limit=7.0
```

```
phase2_random_tract_prob=0.995
phase2_tract_skip_prob_delta=0.05
phase2_tract_skip_prob_init=0.995
phase3_save_intermediate=60.0
phase3_skip=false
phase3_time_limit=10.0
phase4_skip=false
phase4_time_limit=5.0
```

Household Realizations File

This is the final output file from DP. Each row represents a real household in the generated synthetic census. The fields in each row are copied from the household sampling file, and prefixed with four columns:

- "x": East-West Ordinate. Exact location of household, in same units as input maps.
- "y": North-South Ordinate. Exact location of household, in same units as input maps.
- "uid": Unique ID. Each discrete household receives a unique ID derived from the original archtype ID.
- Other columns: All subsequent columns are copied from the archetype referenced by Archetype ID. To conserve memory, unused data from that table was deleted, consequently columns that were not used in the analysis will be zeroed in the output file.

Population Realizations File

This file is constructed like the household file: Each row represents an individual, and the fields are copied from the population sampling file. If none was provided, then this file is not generated. Each row is prefixed with the same four fields as the household output file.

Glossary

Archetype	Prototype from which a specific object can be built. Each record in the household sample table is considered an "archetype" household rather than a physical one. The physical households are copied from the archetypes.
Realization	Complete data record built or computed from some descriptive data. Digital Populations takes a copy of a household archetype and computes a location for it. This complete record (location plus attributes) is a realization of a household.

Digital Populations FAQ

WARNING: Match stat for X eliminated all regions from consideration for PumsHousehold "Y"

The named constraint turns out to be one constraint too many -- the given household cannot be placed anywhere in the map and still satisfy all of the constraints. Constraints are evaluated in the order specified in the fitting criteria file, and each one rules out some of the cells in the region map. The named constraint was the one that ruled out all remaining cells, leaving nowhere for the household to be placed.

This is only a warning as the household will be placed in one of the cells permitted by all the constraints prior to the named one.

How to Start Developing

- Install Java.
  - Use the latest version.
  - Install the JDK, not a JRE. 32- and 64-bit versions both work fine.
  - Bundled versions that include other tools (i.e. GlassFish or NetBeans) won't hurt, but are generally useless for this project.

- Install "Eclipse IDE for Java Developers".
  - Other versions (i.e. J2EE or "plugin development") won't hurt, but are useless for this project.
  - There's no installer, just download and unzip somewhere safe.

- Optionally install NetBeans.
  - Development of the GUI front ends use the visual designer in NetBeans. The source code can be edited and run from Eclipse, however.
  - NetBeans can be installed with Java above, or downloaded and installed separately from: http://netbeans.org/downloads/index.html
  - The small "Java SE" version contains everything required.

- Install the Subclipse plugin for Eclipse.
  - Run the Eclipse installed above.
  - Follow their instructions:

    http://subclipse.tigris.org/servlets/ProjectProcess?pageID=p4wYuA

  - .. or follow mine. For Eclipse 3.5 and 3.6.2:
    - Help -> Install New Software...
    - [Add]

      Name: Subclipse

      Location: http://subclipse.tigris.org/update_1.6.x

      [OK]

    - (wait patiently)
    - select checkbox for Subclipse
    - [Next>]
    - [Next>]
    - accept license, [Finish]
    - (wait patiently)

        - restart when asked

- Configure Subclipse. The default option usually doesn't work, so this switch is necessary.
    - Select Window->Preferences.
    - Select Team/SVN in the tree on the left.
    - Set "Client:" option to "SVNKit".
    - Click [OK].

- Check out the latest version of the RGIK from its repository
    - File -> New -> Other
    - SVN/Checkout Projects from SVN
    - Create a new repository location, [Next>]
    - URL: https://ehlschlaegergik.svn.sourceforge.net/svnroot/ehlschlaegergik
    - (username/password not required to download, only to check in)
        - select Graph2D, Research GIS Kernel, and DPGUI
        - click [Next>]
        - Enter information:

          Check out into the workspace as projects

          Project naming: blank/blank

          Check out HEAD revision ("head" means "latest in repository")

          Depth: Fully recursive

          Ignore externals: NO

          Allow unversioned obstructions: NO
        - [Next>]
    - Use default workspace location: YES
    - [Finish]

RECOMMENDED TOOLS:

WinMerge

http://winmerge.org/

Tortoise SVN

http://tortoisesvn.net/

ConflatePumsQueryWithTracts Phases

Phases of CensusGen (ConflatePumsQueryWithTracts)

ConflatePumsQueryWithTracts operates in phases. Each phase refines the result from the previous phase. Some phases are mandatory, while some can be skipped, though with a loss in output quality.

Note that input files MUST use planar coordinates. Phases 1-3 generally tolerate other systems, but phase 4 will behave funny if not planar. DP does not understand coordinate systems, so distance calculations require a planar system.

Phase 1: Compute Expansion Factors

"Expansion factor" is defined as the number of clones we will create of the records in the households table. These are normally computed from region data, but can be adjusted in the fitting criteria.

Note that only the phase-1 analysis is cached in phase1-cache.bin. Tweaks to this element will NOT cause the analysis to be re-run; the cache is loaded and adjusted based on this element.

Needs region table, households.csv, traits, weights, optional <expansion-factor> section.

Paremeters:

- do_dump_number_archtypes
- phase1_time_limit

Termination Conditions:

- Quits when no improvements can be found. Will take as much time as necessary.

Phase 2: Make Initial Placement

Generates a complete set of households, giving them a plausible initial placement.

Paremeters:

- phase2_random_tract_prob
- phase2_tract_skip_prob_delta
- phase2_tract_skip_prob_init

Termination Conditions:

- None; every household is processed exactly once.

Phase 3: Find Best Placement

Moves households around to find the arrangement that best fits the region statistics.
trys to get as close to goals as possible (matches desired distribution of attribs)

Needs tracts; skipped if only 1 tract

Paremeters:

- phase3_save_intermediate
- phase3_skip
- phase3_time_limit

Termination Conditions:

- Phase is skipped if input data contains only one tract.
- Aborts after phase3_time_limit.
- Quits if no changes can be found (i.e. all households have been probed, but none moved).

Phase 4: Adjust Clustering

Select precise easting/northing values for all household realizations, then nudge them based on clustering criteria. Operates much like phases 2 and 3 -- first households are place, then they are moved around until arrangement is as good as possible.

Needs <position-rules>; skipped if missing or empty. Elements specify goal values. This phase will not try to get "better" than the goal, but get as close as possible to goal values.

Paremeters:

- phase4_skip
- phase4_time_limit

Termination Conditions:

- "Giving up: Too many failures." If phase makes too many consecutive tests without finding a household to move, it will assume no improvements are possible and end the phase. "Too many" is defined as the number of households. So if there are 150,000 households and the phase scores 150,000 failures in a row, it will end.
- "Giving up: Time limit has been reached." Phase will end if processing has taken too much time. "Too much" is defined by variable "phase4_time_limit" in the properties file.

Dp2Kml Tool

The Digital Populations suite includes a program called Dp2Kml that can convert the output of censusgen into maps for Google Earth. The command requires the households file and optionally a matching population file, and will build a Google Earth representation of the data using reasonable internal defaults.

Dp2Kml has reasonable defaults for Digital Populations data, so running the program is *almost* simple.

```
Dp2Kml --crs EPSG:32610 rzn001-households.csv
```

The problem is that Digital Populations files are usually UTM, and such data **requires** a --crs argument to specify where on the Earth this data is located. Without this argument, the results will likely not be visible at all. You can read more about Coordinte Reference Systems.

Bubble Template

Dp2Kml can add descriptive pop-up bubbles to each generated placemark, and can even help you generate a bubble template file. This file is processed once for every data point in the households file, and the result becomes the bubble text in the generated placemark. Note that Dp2Kml has been tuned to handle large bubbles and large data sets, so memory usage *shouldn't* be a problem.

- Open a suitable command prompt.
- Enter this command:

  ```
  Dp2Kml --gentemplate rzn001-households.csv >bubbble.html
  ```

- Change the CSV file name if necessary, and the HTML file name if desired.
- Edit the bubble.html if desired. Lines can be disabled by inserting ## at the beginning of the line. This has already been done for a few unimportant lines.

To run with the bubble template, use this command:

```
Dp2Kml --crs EPSG:32610 --bubble bubble.html rzn001-households.csv rzn001-
population.csv
```

Only add the population file name if you have one. The file is ignored if you're not using bubble templates.

Style Template

Dp2Kml can customize the appearance of icons and labels based on the contents of each data record. A special script file must be hand-written to accomplish this. You can read more about style template files.

Caveats

- The program store households and population records in a database file. This greatly reduces the amount of memory required, but the database file can be extremely large.
- Upon opening the kmz, most placemarks will not be visible. The huge numbers of households generated by Digital Populations can easily choke Google Earth, thus the result has been broken into tiles which will not become visible until you zoom in "close enough". How close that is depends on the numbers of placemarks involved.
- To help visualize the overall result, an "overview" bitmap will appear in red on the landscape. More intense red means more households. Zoom into the red to make the individual placemarks appear.

Usage:

Running the program without parameters will generate the help text:

```
Usage: Dp2Kml [options] <household-file> [<population-file>]
 <household-file> is the table of households generated by Digital Populations
 <population-file> is an optional table of members of the above households

Option                                  Description
------                                  -----------
```

```
--bubble <File>          create bubbles using this template
                           file (default: no bubbles)
--crs                    coord ref system used in file
                           (default: no conversion)
--gentemplate            output simple bubble template from
                           households file and exit
-h, --help               print this help
--name                   format string for name of each point
                           (default: "%{uid}"
--output <File>          dir or file name to receive output
                           (default: household file name, with
                           .kmz extension)
--pix <Integer>          minimum pixels per household for
                           visibility; tiles with less will not
                           be displayed
--style <File>           adjust placemark styles using this
                           template file
--testtemplate           generate HTML from the first 10
                           records then exit
```

Specifying Coordinate Reference Systems

Dp2Kml an Csv2Kml can convert coordinates to the system required by Google Earth. To do this, they need to know what system is being used by the input file. Since coordinate systems are a nightmare to deal with, the programs use GeoTools to provide the conversion as well as definitions of different coordinate systems.

Both program accept a coordinate system specification on their command lines using the syntax "`--crs <file or name>`". The parameter can either be a file name, or a specially coded CRS name.

- **CRS name:** The best option is to supply one of the CRS name strings documented by http://spatialreference.org/ . These names take the form "`EPSG:####`" or "`SR-ORG:####`", and generally contain all the parameters GeoTools needs to perform a conversion.

- **File name:** If the parameter exists as a file, then the file is loaded as a projection file. GeoTools supports a variety of formats, but it's very picky about the file being *complete*. If any data is missing from the file (and some of the parameters are quite obscure) then the program will crash. It's generally much better to find the CRS name, and use GeoTools' internal projection data.

Note that the EPSG name system is the most reliable way to get a successful conversion. If a workable name cannot be found, the next best option is to re-project the input data into a system documented on that site. It can be quite difficult to build a projection file that GeoTools will accept, and we the developers have not been successful. The file needs to look something like these examples (ignore the bug described there, it's not relevant.) ESRI doesn't seem to produce files in that format, though the GeoTools docs say it can. Further, GeoTools requires certain parameters be present in the file, but it's rare to find a projection file that contains these parameters.

If you can convince the tools to accept your projection file, then you can use that. But the EPSG:number system is the best idea. GeoTools works best with those, and it ensures you're using a well-known coordinate system.

Bubble Templates

Google Earth allows detailed descriptions to be attached to placemarks. The text appears in a popup "bubble" or "balloon" when the user clicks on a placemark. The *2Kml tools allow the description text to be generated from a *template file* which produces a block of styled text for each household in the input file.

The template file is an HTML file containing special markers. The file is processed once for each input record, and the markers are replaced with values from the input record. The resulting text is copied into the placemark description.

To create a template file from scratch, follow this sequence:

1. Run Csv2Kml with --gentemplate to create a simple working file to start with.
2. Edit and refine the generated file as needed.
3. Run Csv2Kml with --testtemplate to ensure the template appears correctly.
4. Run Csv2Kml normally and verify the result works correctly in Google Earth.

Testing Support

Dp2Kml provides two tools to help build templates.

- --gentemplate will construct a simple template from the input tables. This template will contain all columns from the households file, as well as all columns from all $members if a population file is specified.

```
dp2kml houses.csv peoples.csv --gentemplate > bubble.html
```

- --testtemplate generates a plain HTML by evaluating the template file over 10 random households. This file can be opened in a browser to verify the template.

```
dp2kml houses.csv peoples.csv --bubble mytemplate.html
--testtemplate
```

The Template File

The template file is a composite of three scripting "languages" which work together to produce styled text:

1. **HTML**, the language of the World Wide Web. Google Earth can display blocks of HTML in bubbles exactly as if they were mini web browsers. Into the HTML is inserted chunks of code using ...
2. **Apache Velocity**, the template processing language. The Velocity engine evaluates the chunks, and replaces them with data from the input file. Velocity gains access to the input file by way of ...
3. **Dp2Kml itself**, which prepares each data record, and calls Velocity to generate the bubble text for the corresponding placemark. Dp2Kml also provides a library of helpful functions that can be called from the template file.

The HTML is plain HTML as it's used on the Internet, with some slight modifications: the <HTML>, <HEAD>, and <BODY> wrapper elements are not necessary, and some elements won't work the same as they do in a browser. However, most elements work exactly the way you'd expect. You can even use a GUI designer to build the HTML file.

Apache Velocity processes the HTML template file, inserting data values as requested. Velocity uses a special language to identify variables and functions that need to be evaluated. The variable is then removed, and replaced with the result of the evaluation. See the documentation for all the tricks available:

http://velocity.apache.org/engine/devel/user-guide.html

Example:

```
<P>There are $point.member_count members in this household.</P>
```

Dp2Kml provides the *context* in which the above template is evaluated. A "context" is a list of variables and functions that can be directly accessed with the $ notation. The *2Kml tools provide one or two variables, depending on input files:

- $point is the data from one household. It holds all the fields from one row of the households table. This will generally be one of the rzn###-households.csv files generated by Digital Populations, but Csv2Kml can process any tabular file.

- $members is only provided by Dp2Kml, and contains a list of the members of the current household. The variable is only available if a population file is being processed with the households file. The value is a list of rows, and each row contains all of the fields from one row of the population table (generally rzn###-population.csv as generated by Digital Populations.)

Dp2Kml also provides some utility functions, documented below.

Individual fields can be accessed by name or by column number. Velocity will remove the reference text, then fetch the referenced column from the current row, and insert that value in its place. Any of these methods will work:

- `$point[index]`
- `$point.Clean_Column`
- `$point["Column Name"]`
- `$point["Clean_Column"]`
- `$point.get("Column Name")`
- `$point.get("Clean_Column")`

"Index" is simply a column number, where the leftmost column has the value 0 (zero). "Column Name" is the value in the first row of the table for a column, where column names are traditionally stored. "Clean_Column" is a *cleaned* version of the column name, since Velocity imposes certain restrictions on names when they are not enclosed in quotes. Dp2Kml applies two transforms to convert column names into clean columns:

1. All characters that are not letters or numbers are replaced with underscores "_". Velocity prohibits a certain selection of characters from appearing in field names, but for simplicity, Dp2Kml replaces everything that isn't a letter or number.
2. Consecutive underscores are replaced with a single underscore.

For example, a column named `$data$_$column$` in the CSV file would be referenced as `"$point._data_column_"`. Rule 1 converts the title to "_data___column_", then rule 2 collapses the underscores to form "_data_column_".

In other words, $point.name works for the "clean" version of the name, while $point["name"] and $point.get("name") work for both the clean name as well as the original table header. And $point.[number] works if you know the index of the column you want (though numbers shouldn't be used as they stop working if the columns are rearranged.) Because of the above, velocity supports tricks like this:

```
#set($n = "Person assessing scores")
<TD>$n</TD><TD>$point[$n]</TD>
```

Utility Functions

The velocity language does not provide any helper functions by default, so the tools provide a few:

- $ConversionTool -- Utility methods from Velocity class ConversionTool.
- $DisplayTool-- Utility methods from Velocity class DisplayTool.
- $MathTool -- Utility methods from Velocity class MathTool.

Helper functions are accessed by name, much like the fields in $point.

Example:

```
$MathTool.div($point.age_of_house, 12)
```

Caveats

Here are a few other notes for working with Velocity:

- `$foreach.count` starts from 1, not zero.
- Math expressions can only be used inside `#set`, although utility functions can be called anywhere.
- *null* is not the same as ""; conditionals must test for both: `#if($v && $v != "")`

Placemark Template

The text in this template is completely ignored. Instead, the template must #set Velocity variables with values that describe the desired appearance of the placemark and its label. Like the bubble template, this file is evaluated once for every household,so each placemark can be customized.

For reference, a #set command takes this form:

```
#set( $icon.color = "red" )
```

The available variables are:

- icon.img = name or file (default "household.png")
- icon.size = multiple of standard size (default 0.4)
- icon.color = name or code (default "white") (acts as color filter for icon)
- label.size = multiple of standard size (default 1.0)
- label.color = name or code (default "white")
- name = text of placemark label
- desc = content of pop-up bubble

WARNING -- Icon file management has NOT been implemented, so the icon image file must be an absolute HTTP URL, or else the file must be inserted into the final KMZ file by hand.

icon.size and label.size are specified as multiples of whatever Google earth considers normal. The only way to find out is to try it, and examine the result.

icon.color and label.color can be one of the options below. These options are tried in the order given, so this method cannot decode a color whose name is 8 characters long and composed entirely of the letters [ABCDEF].

- KML color code as eight hex digits: aabbggrr
- HTML CSS short form: #rgb
- HTML form: #rrggbb
- HTML color name

$name and $desc are pre-initialized with defaults before the script is called, and the script and edit them or generate new ones if desired. The bubble template is a much easier way to generate content for the bubble, but it can instead be generated here if it seems simpler.

Csv2Kml Tool

The Digital Populations suite includes a program called Csv2Kml that can convert CSV tables into maps for Google Earth. This program is identical to Dp2Kml, except for a few differences:

- Dp2Kml supports a second table which provides data on the members of each household. This program only supports a single table.
- Dp2Kml stores intermediate data in database, allowing it to handle much larger projects. This program loads all data into RAM, and is severely constrained because of this. Bubbles in particular generate huge amounts of text, so memory problems are common.
- Dp2Kml includes defaults for <x>, <y>, and --name. These fields must be explicitly provided here.
- This program supports CSV Metadata; Dp2Kml does not.

Usage and caveats for this program are otherwise identical to Dp2Kml.

Usage:

Running the program without parameters will generate the help text:

```
Usage: Csv2Kml [options] <file> <x> <y>
  <file> is csv file to convert.
  <x> is name of col containing horizontal pos of each point
  <y> is name of col containing vertical pos of each point

Option                                  Description
------                                  -----------
--bubble <File>                         create bubbles using this template file
                                          (default: no bubbles)
--crs                                   coord ref system used in file
                                          (default: no conversion)
--gentemplate                           create default bubble template and exit
-h                                      print this help
--name                                  format string for name of each point
                                          (default: "%{householdID}:%
```

```
                                    {realizationID}"
--output <File>                     dir or file name to receive output
                                      (default: household file name, with .
                                      zip extension)
--pix <Integer>                     minimum pixels per household for
                                      visibility; tiles with less will not
                                      be displayed
--style <File>                      adjust placemark styles using this
                                      template file
--testtemplate                      generate HTML from the first 10
                                      records then exit
```

CSV Metadata

CSV tables by default have no metadata, so we've defined a special marker that can be used to identify metadata rows in the table. "Metadata" is data that describes the data, i.e. data that is not specific to any particular field, but rather describes the column as a whole. Common metadata are column data types, descriptive titles, help text, etc. The exact meaning of the metadata is application-dependent.

A metadata row is a normal table row where the first column flags the row as metadata. The text in the first column must have the form "##word" or "##word data". "Word" provides a name for the metadata in this row, and "data" provides the metadata value for the first column of the table. Each additional field provides the metadata for the corresponding column. Fields can be empty.

Metadata is accessed in the Velocity script via the "word" along with the desired column name or number. The metadata system adds a new variable to the template context named "metadata", and $metadata works much like $point.

- Access a metadata field with `$metadata.column.word`, where "column" identifies a column just like it does for $point (i.e. .clean_column or ["column name"] or .get("column name") or [number]). "word" identifies the metadata row of interest with the same syntax [i.e. .clean_word or ["word"] or .get("word")]. So a field can be access simply as `$metadata.Easting.type`, or verbosely as `$metadata.get("Easting").get("type")`.
- The $metadata object always exists, but $metadata[column] or $metadata[column][word] may be null. Use this syntax to test for the existence of a field: `#if($metadata[column] && $metadata[column].word && $metadata[column].word != "")`
- Csv2Kml provides a few metadata rows by default, generated from the structure of the CSV input file:
    - csvIndex contains the column number starting from zero.
    - csvTitle contains the original name of column from the first row of the CSV file.
    - csvField contains the "clean" form of column name.

Some examples:

- $metadata[4].csvIndex == 4
- `$metadata.county_name.csvField` == "county_name"
- `$metadata["county name"].csvTitle` == "county name"
- `$metadata[$metadata[4].csvTitle].csvIndex` == 4

Here is a simple example table with metadata:

```
Name          Easting Northing Size Age Color
##type string double  double   int  int string
Bill          45      35        12  90  blonde
Willhelm      37      22         8  72  alabaster
Billiard      18      89         1   3  ivory
```

Data column 1 will contain {45, 37, 18}, while metadata column 1 contains "double", which can be used to define the type of structure that should hold the data. The metadata field can be accessed as metadata.get(1).get("type") or metadata.get("Easting").get("type").

CsvPlusMap Tool

This utility adds a column to a CSV table, where values are read from a map based on coordinates encoded in each row of the table. Each row must contain an x/y coordinate. This location is examined on the map, and the value found there is inserted into the corresponding row of the new column.

The result will be written to a new file.

Usage:

Running the program without parameters will generate the help text:

```
Usage: CsvPlusMap [options] <csv-file> <map-file>
  <csv-file> is csv to extend
  <map-file> is ESRI ASCII map file that provides values

Option                                  Description
------                                  -----------
-h                                      print this help
-n                                      name of new column (default is base
                                          name of map file)
-o <File>                               output file (default is <csv-file>.
                                          plusmap.csv)
-x                                      name of X/longitude/easting column
                                          (default is "x")
-y                                      name of Y/latitude/northing column
                                          (default is "y")
```