UNIVERSITY OF Southampton

Technical summary:

Office for National Statistics 2001 Output Area Production System (OAPS) methodology

Andrew Harfoot, Samantha Cockings, Duncan Hornby
School of Geography, University of Southampton

August 2010

Produced by the Census2011Geog project
(ESRC Census Programme grant RES-348-25-0019)
http://census2011geog.census.ac.uk

Census.ac.uk

E·S·R·C ECONOMIC & SOCIAL RESEARCH COUNCIL

Technical summary:
Office for National Statistics (ONS) 2001 Output Area Production System (OAPS) methodology

Table of Contents

1. Background

This technical summary of the Office for National Statistics (ONS) 2001 Census Output Area Production System (OAPS) methodology was produced by the ESRC-funded Census2011Geog project based on examination of the 2001 OAPS code (as provided by Steve King, ONS), a meeting with Steve King (15/01/09) and subsequent discussions with Steve King and Brian Parry (ONS).

The 2001 OAPS system comprises a hybrid set of software programmes and files, including ArcInfo Arc Macro Language (AML), Powerbuilder scripts, C programmes and various system files/programmes such as Unix/Exceed scripts which link the various components together and pass data between the different programmes.

There are three key stages to the OAPS process:

- Pre Automated Zoning Procedure (AZP) code – this prepares the various datasets and look-up tables and creates the synthetic postcode polygon building blocks
- AZP process code – this is the automated zoning procedure which carries out the iterative aggregation of the building blocks in order to produce an optimised set of output areas thus producing a look-up table of postcodes to output areas
- Output Area (OA) creation code – this set of code uses the look-up table to create the final set of output areas.

Key methodological steps, rules and values are highlighted in bold italics.

2. Pre AZP preparation code

Run_OAPS.aml

A controlling script which executes ALL or some of the steps. Run_OAPS is called by the Powerbuilder graphical user interface (GUI) module W_OAPS. Run_OAPS receives a ward code and starting step number, ward part list and boundary line year as arguments. The ward code defines which ward to work with (each ward is handled in isolation). All steps are run in the same order, but certain circumstances may necessitate re-runs for parts of the processing, hence the starting step number argument.

A small script (set_variables.aml) is run to set the ward code and boundary line year as global variables, amongst other variables. If the year argument is not passed, then the global variable defaults to 1998.

Extract_address.aml

Reads all address points (technically, ***dwelling spaces***[1]) for current district from the FormID table in the Oracle / ArcSDE database by calling external Exceed scripts which in turn call ksh scripts (not available for review). Output is a point coverage called addr.*code*.

Create_addrpt.aml

This AML undertakes further processing on the address point data extracted by Extract_address.aml. Most of the processing is fairly mundane:

- Leading zeroes are trimmed from the eastings and northings fields, this is probably necessary to ensure that coincident grid references are identified as such.
- The list of farpoints (from the farpoints.txt file) is converted into a point coverage with the same structure as the address point data. This allows the two datasets to be merged for the Thiessen operation in Create_Poly.aml

Finally a query is run against the Oracle based FormID table. The query selects rows from the current district, and groups by the eastings, northings, and ward_code fields, counting the number of records. Any groups with a count greater than 250 (greater than twice the household count target) are written to *stacked.code* output. These large collections of coincident address points are classed as tower blocks and need to be broken up before the OAPS processing can continue. The dispersion of the tower blocks is handled by manual intervention using the Tower.aml script.

Tower.aml

This is a user intervention routine that disperses stacks of more than 250 coincident address points. The OAPS methodology insists that coincident points must all be assigned to the same building block polygon. Large stacks could therefore cause problems when OAs are built by being immediately over threshold, and therefore indivisible.

For each identified stack of addresses in turn, the user is prompted to create subsets of the stack by choosing one or more sets of addresses with the same postcode. Each subset must have an address count of between 125 and 250. These limits were defined to ensure that the each subset could constitute an OA in its own right without requiring additional building blocks to merge with. Following the selection of the subset, the user then assigns these addresses a different grid reference, near to the original stack's location.

Brian Parry (ONS) stated that the procedure followed by the operators was to examine the addresses of the postcodes in the stack to ascertain which were at the top of the building, and to then move this upper block to the nearest point in a car park, nearby grassy area or similar.

[1] There are various inconsistencies between the terminology employed in the documentation/annotation of the OAPS code and the real-world features that the terms relate to. For example, reference is frequently made to *address points/counts*, but technically the features being counted/extracted are *dwelling spaces*. No attempt has been made to correct these terms throughout this technical summary, so readers should bear this in mind when interpreting the code.

Situations may exist where it isn't possible to create subsets that fit the criteria specified by the program. In these situations the code returns an error and the assumption is that the problem was handled manually outside the OAPS code process.

Following the completion of this manual intervention, it is assumed that the Create_addrpt.aml is re-run to confirm that all of the 'tower blocks' have been dispersed.

Note that the adjusted locations of the addresses are not written back to the FormID table, but are retained solely for the OAPS processing.

Create_wardpart.aml

Ward parts (WP) are created from the intersection of ward boundaries and parish boundaries to ensure that both are preserved in the OA geography. Ward parts form independent processing groups for subsequent OAPS processes. The overlay operation can produce a number of odd situations; a lot of this script is therefore focussed on identifying these and then flagging or correcting them.

- Following the intersection, 'sliver' polygons with an area less than 10,000 sq m (with an option to reduce this) are associated with the first adjacent polygon that is within the same ward and above the 10,000 sq m threshold. If no qualifying polygons are immediately adjacent, the set of polygons adjacent to these (the next 'shell' out) are considered. This has the potential to produce disconnected groups. Sliver polygons not finding a non-sliver to merge with result in an error being logged.
- After a DISSOLVE command is used to merge the slivers identified in the previous step, all remaining areas less than 10 sq m are merged with the neighbour with the longest shared boundary using the ELIMINATE command.
- Attributes are reworked, removing and recreating WPCODE and PARENT_WP, and defining substrings of WPCODE and OPCS-CODE to extract further information.

The second part of the script assigns ward part codes to the newly created polygons. It can also be called subsequently to rebuild the ward part codes if some ward parts are later found to be sub-threshold and needed to be grouped (and assigned a parent). See Edit_wp.aml.

WP codes are assigned sequentially starting at 400; the county and district code is prefixed to this number. 400 was chosen as the starting point as an older version of the OA coding scheme used a similar format. All objects with the same values of OPCSCode, ParishCode and Parent_WP receive the same WP code. On the first run, the Parent_WP variable is newly created and therefore empty. A text file *(wplookup.txt)* is generated which lists the number of parts for each WPCode.

Finally the individual ward parts are combined using the DISSOLVE command. The field used for the dissolve operation is parent_wp. On the initial run, this field will be empty, however, subsequent user intervention (see Edit_wp.aml) may have added values to this which group sub-threshold ward parts together. If the parent_wp field is empty, then the wp_code value is substituted. This operation can produce split ward parts, where polygons with the same WPCode are not adjacent. A list of WPCodes where this situation occurs is generated and placed in *wpsplit.*

UO_wplookup

This Powerbuilder script is concerned with identifying under-threshold ward parts, which are then passed on for manual correction. As the AZP algorithm is run on a ward part at a time to ensure that ward and parish boundaries are preserved, it is important that there are sufficient households and population in each ward part to allow a single OA to be created that is above the minimum thresholds.

The ward parts are identified by the *wplookup.txt* file generated by Create_wardpart.aml. The initial step is to load this information into a Database Management System (DBMS), and then use this table to calculate the total population and number of households for the ward part. It is assumed that the counts come from a pre-generated database table which aggregates the FormID information by ward and parish codes.

If a ward part's population or household count is below the thresholds defined in *oaps.ini*, an error will be logged. All the ward parts will be processed, but the script will not report success on completion.

The initial run of this script will not encounter any values in the parent_wp field, but subsequent runs may. The parent_wp field allows ward parts to be grouped together so that they are above threshold. The populations of the child ward parts are summed together to check that this is the case, and a further check is made for unnecessary grouping. An error is also logged if all the child ward parts are over-threshold.

If sub-threshold ward parts are found, it is then necessary to run Edit_wp.aml to manually aggregate these into groups that are within-threshold.

Edit_wp.aml

If UO_wplookup identified sub-threshold ward parts this script allows user intervention to solve the problem. Ward parts can be grouped together by assigning a value (the parent wp_code) to the parent_wp field, the group will then be handled as a single unit by subsequent OAPS processing steps. The user can also identify and sub-divide existing groups.

The rules for creating a new ward part group are as follows:

- Only the parent (initial) ward part can be over-threshold
- Only adjacent, ungrouped ward parts can be added to the group.
- The default number of ward parts to group is two

After altering the groupings and updating the parent_wp field accordingly, Create_wardpart.aml is automatically re-run, starting at the point where ward part codes are assigned. It is assumed that the UO_wplookup script is then re-run as well.

Create_oscar.aml

This is a short script that merges the Ordnance Survey Centreline Alignment of Roads (OSCAR) data from a tile-based system into a single coverage for the district being

processed. After appending the data together, the linear features are put through a CLEAN command that builds polygons where possible, but will not delete any unused (or dangling) lines. Whilst not of use initially, the dangling lines may come into play when the coverage is overlain with the other boundary datasets.

Create_Poly.aml

This script moves on from Create_oscar.aml by continuing the creation of building blocks through overlay of various boundary datasets. The script works at the ward part level, looping through a list of the unique parent_wp codes. This means that grouped ward parts are handled together.

At a couple of points in the script, special consideration has to be made for, 'doughnut' polygons' (polygons containing holes), as the topological system employed by ArcInfo will generate an object to fill the hole when the object is extracted into a new dataset. Typically the auto generated doughnut hole polygons are identified on the basis of null attribute values and a positive area (to distinguish them from the 'universe' exterior topological polygon).

All address points (dwelling spaces) are extracted that fall within the ward part. A text file (*add%wpnum%.txt*) containing the address points is created: this is used in the handling of vertical (stacked) postcodes. Thiessen polygons are then generated from the address points along with an additional set of appended 'far points'. The 'far points' sit around the minimum bounding rectangle (MBR) of England and Wales, and ensure that the Thiessen polygons always extend beyond the edge of the ward part being processed so that no gaps are present when the polygons are clipped back to the ward part boundary.

The address point-based Thiessen polygons are then dissolved based on their postcode, and polygons without a postcode are removed by merging with neighbours using the ELIMINATE command.

This process may not remove all polygons without postcodes, particularly if two or more of these polygons are adjacent to one another. Empty polygons may originate initially due to some address points not having postcodes.

Finally the 2001 Enumeration District boundaries (EDLINE2001) and OSCAR road centrelines are intersected with the postcode polygons (specifying the IDENTITY switch to preserve all input information). Very small polygons (less than 10 sq m in size) are merged with their neighbours using ELIMINATE. This may result in minor deviations from the original ward part boundaries.

Text files are generated from the intersected products, which are used in the UO_snap_pc code.

1. *wp%wpnum%polypat.txt* contains the combination of PolyID, postcode, thiessenID, EDLINE code and OSCAR ID for each output polygon
2. *wp%wpnum%polyaat.txt* contains the adjacency information for each polygon coded as a series of left and right poly IDs
3. *wp%wpnum%points.txt* contains a list of PolyIDs with an entry for each instance of an address point in the poly. This is used to identify empty polys by virtue of their IDs not appearing in the list.

List_multiples.aml

This is an initial analysis of the address point list *add%wpnum%.txt* (generated by Create_Poly.aml), with the aim of identifying coincident addresses with differing postcodes – a ***vertical postcode stack***. The script works at the ward part level, looping through a list of the unique parent_wp codes. This means that grouped ward parts are handled together.

The output is a text file *vert%wpnum%list.txt* that gives the coordinates, postcode and number of address points at that location, in that postcode, for all the stacks of addresses that have more than one postcode between them. A second text file *vert%wpnum%worst.txt* is output that counts the number of postcodes at each location and presents the list sorted in descending order – this seems to be for use outside the remaining OAPS processes.

UO_Vertlookup

This script generates a text file called *vert%wpnum%pc.txt* that gives a ***postcode to stack code lookup table***. The aim is to ensure that ***all instances of a given postcode are kept together – an underlying requirement of the 2001 OA creation policy***.

This requirement is complicated by the fact that some postcodes are in vertical stacks so that in the process of building Thiessen polygons from address points and dissolving them by postcode they may be 'lost' if they weren't selected as the dissolving postcode value.

The script works at the ward part level, looping through a list of the unique parent_wp codes. This means that grouped ward parts are handled together.

The maximum allowable postcodes in a stack (single point) is hard-coded to 100. The code will also return an error if the number of postcodes within a ward exceeds 1500 (hardwired value).

The code works with the file *vert%wpnum%list.txt*, generated by List_multiples.aml. The processing steps are as follows:

- Reads each line from vert*wpnum*list.txt and extracts out XY and postcode
- When a new stack grid reference is found it then checks if the postcode(s) in it have been assigned to a previous stack. Based on this it does 1 of 3 things:
    i. None of the postcodes are in any other stack so create a new stack code.

ii. Some of the postcodes are already in a single, existing stack assign the remaining unassigned postcodes to this existing stack code.

iii. Some of the postcodes have already been assigned, but to more than one stack. Take the first existing stack code, and assign all the other postcodes, both unassigned and all postcodes with the other stack codes to this one.

An illustration of the various options is given below:

1. Situation at load – no stacks assigned

Postcodes							
	001		006				009
	002	004	007	004	010	007	002
	003	005	008	009	011	008	012
Stack Code							

2. Three stacks read L to R – new stack codes assigned (option i above)

Postcodes							
	001		006				009
	002	004	007	004	010	007	002
	003	005	008	009	011	008	012
Stack Code	A	B	C				

3. The next stack contains a postcode (004) that has already been assigned a stack code (B). The other postcodes in the new stack have not been assigned a stack code, or have been assigned the same one as 004. All the postcodes in the new stack are assigned the existing stack code (option ii above)

Postcodes							
	001		006				009
	002	004	007	004	010	007	002
	003	005	008	009	011	008	012
Stack Code	A	B	C	B			

4. Two more applications of options i and ii

Postcodes							
	001		006				009
	002	004	007	004	010	007	002
	003	005	008	009	011	008	012
Stack Code	A	B	C	B	D	C	

5. Now a stack is encountered that contains two postcodes that have already been assigned to different stack codes (002 – A & 009 – B). Option iii applies, and the whole stack is assigned to the first stack code found (A). In addition, all stacks assigned to the other stack codes (B) are also reassigned to the first stack code (A)

Postcodes							
	001		006				002
	002	004	007	004	010	007	009
	003	005	008	009	011	008	012
Stack Code	A	A	C	A	D	C	A

This continues until all the stacks have been processed and has the effect of ensuring that ***all instances of a postcode will receive the same stack code***, and that ***all members of a stack will also receive the same stack code***.

Output is written to *vert%wpnum%pc.txt* for each postcode with the format as follows: postcode,STK000*x*.

Note: as the input is a text file which only holds stacked address points the output of this process will always generate a row of data, because the minimum is at least 2 address points with the same postcode which will generate a stack code.

UO_snap_pc

This is a complex script dealing with assigning empty polygons (those not containing address points created during the intersection of layers in Create_Poly.aml) to an appropriate neighbouring polygon.

Two constraints exist: the ***maximum number of polygons in a ward cannot be greater than 10,000*** and the ***maximum number of neighbours a polygon can have is 50***.
Parameters read from OAP.INI are the minimum boundary length, set to 16%, selfmerge = 1 and deftype = 3.

The script works at the ward part level, looping through a list of the unique parent_wp codes. This means that grouped ward parts are handled together.

Information is read from *wp%wpnum%polypat.txt wp%wpnum%polyaat.txt* and *wp%wpnum%points.txt* created in Create_Poly.aml. These describe the fragments created when OSCAR, EDLINE and the postcode Thiessen polygons were overlaid – each fragment has a set of three IDs assigned from the three intersected datasets. In addition, the AAT file contains the adjacency relationships between neighbouring polygons and the common boundary length. *Wp%wpnum%points.txt* contains the IDs of all fragments containing an address point and is used to identify those which don't. These empty fragments are the target of the merging process.

Each boundary between fragments is assigned a type based on which, if any, of the fragment IDs change across it. Using the numbering system below, each boundary is assigned the highest number from the ID changes detected.

The order of assignment is:

1. Thiessen ID
2. EDLINE 2001 Code
3. OSCAR ID

If the fragment was generated from a dangling line (particularly from the OSCAR data), then none of the IDs will change across it, and it is assigned a value of 3.

Each empty polygon is cycled through in turn and its neighbours passed through a series of rules:

1. Calculate shared boundary with the neighbour as a % of empty polygon perimeter
2. Ignore neighbours that have a shared boundary < 16% to avoid creating thin 'necks'
3. Look for pairs of neighbours that have the same postcode and are not themselves contiguous; if they exist, merge the empty polygon with one of them to form a link and avoid splitting postcode polygons.
4. If all neighbours have different postcodes, merge with a non-empty neighbour trying to minimise the boundary type number and then maximise the shared length. If there are no suitable candidates, the criteria are relaxed: initially the program allows 'mutual mergers' with neighbours already having the same merge ID. If this does not work, then the shared boundary length threshold is reduced by one % point, and the neighbours are checked again. The relaxing of criteria alternates back and forth until a suitable candidate is found, or the shared boundary threshold reaches 0, at which point an error is logged.

If the solution found is a mutual merger then an error is logged. Otherwise, all polygons with the same merge ID as the empty polygon are re-assigned to the new merge ID. In this way existing groups of polygons are not broken up by subsequent reassignments.

For each input polygon, the polygon ID and new postcode, as dictated by the merge process, are written to a text file *wp%wpnum%newpc.txt*.

Note that in the program comments it is suggested that allowing mutual mergers may cause empty polygons to arise, though it is unclear how this can happen as merger candidates are never allowed to have an empty postcode and therefore should never be considered. It is possible that there was some manual intervention in this process.

Update_pc.aml

Following on from the merging code UO_snap_pc, this code dissolves the boundaries between empty polygons and neighbours. As the process may be iterative in situations with complex postcode geometries the code is designed to be run at least two times, initially in 'SNAP' mode to generate a set of intermediate data for further checking. The final run is made in 'CLEAN' mode, preparing the datasets for input to AZP.

The script works at the ward part level, looping through a list of the unique parent_wp codes. This means that grouped ward parts are handled together.

The intermediate file generated in SNAP mode is *PC%wpnum%INIT*; this is used as the input to the script when run in CLEAN mode.

The starting geometry is the intersected polygons generated from Create_Poly.aml. To this file is added the modified postcode assignments generated by UO_snap_pc. The new postcode values are then used as the attribute for a DISSOLVE command – hopefully removing the empty polygons.

Various output text files are generated to describe the new dissolved geometry; these are in a very similar format to those generated by Create_Poly.aml:
wp%wpnum%initpat.txt - POLYID, POSTCODE (new values), AREA, X-COORD, Y-COORD
wp%wpnum%initaat.txt - LPOLYID, RPOLYID, LENGTH
wp%wpnum%initlist.txt – POLYID, ADDRESSPT EASTINGS, ADDRESSPT NORTHINGS – one record for each address intersecting a polygon.

In SNAP mode, some reworking of the vertical postcode data is performed that extracts all the postcodes that were not represented in the Thiessen output. This happens when the postcode is part of a stack from which only one value can be used to attribute the Thiessen. If the postcode is not used to attribute any of the Thiessen polygons that it lies within, then it will not be represented. This list of 'hidden' postcodes is used to query out all the stack codes that they have been assigned to from *vert%wpnum%pc.txt*. The resulting list is put into *vert%wpnum%finalpc.txt*.

In 'CLEAN' mode, the 'hidden' postcode steps are not run. Otherwise the steps are very similar apart from the filenames used. At the end of the script, prior to output, there is an additional section dealing with the urban/rural classification of addresses:

The urban / rural classification is INTERSECTed with the address point dataset, which is then RELATEd to the polygons using the new postcodes. Polygons are flagged as urban if they are associated with one or more 'urban' address points, or if they have a stacked postcode (STK*). All other polygons are flagged as rural. In this section a 'group' attribute is added, but this is not used at all throughout the script.

The output files are almost identical to those generated in SNAP mode, with slight changes to the naming, and the inclusion of the urban rural flag:
wp%wpnum%pcpat.txt - POLYID, POSTCODE (new values), AREA, X-COORD, Y-COORD, URTYPE
wp%wpnum%pcaat.txt - LPOLYID, RPOLYID, LENGTH
wp%wpnum%pclist.txt – POLYID, ADDRESSPT EASTINGS, ADDRESSPT NORTHINGS – one record for each address intersecting a polygon

UO_Clean_PC

This, again, is complex code which is another level of error correcting or "cleaning up" of data. It identifies split postcodes with few address points that are a certain distance from the main body of the split postcode.

Two constraints exist, ***the maximum number of polygons in a ward (maxns) cannot be greater than 1500*** and the ***maximum number of neighbours (maxnbs) a polygon can have is 30***. Other parameters are read from the OAP.INI file: the maximum number of address point (*maxpts*) is set to 1, reporting width (*repwidth*) to 1.6449 and error width (*errwidth*)

to 2.3263. The width values are thresholds used in the reporting of erroneous split postcode polygons.

The script works at the ward part level, looping through a list of the unique parent_wp codes. This means that grouped ward parts are handled together.

This is the script in which the vertical postcode stack codes (STK*) are applied to the polygon fragments. The vertical stack data used is the reworked *vert%wpnum%finalpc.txt* file generated by Update_pc.aml in SNAP mode, and only relates to postcodes that have been 'hidden' in the Thiessen generation process.

At the end of this script the Update_pc.aml script is run again in CLEAN mode.

Using the various output files generated by Update_pc.aml in SNAP mode, polygons that still haven't received a postcode can be identified. These are merged with the neighbour with the longest common boundary, ideally with a postcode, and not erroneously split. A special error is raised if the polygon is an island (i.e. a polygon with no neighbours) and cannot be merged.

The second check run by the script is a check for split postcodes. This is done by checking for other polygons with the same postcode, which are referred to as siblings. If this situation arises, then the polygon containing the highest number of address points is identified, and the distance between the centroids of this and all the other sibling polygons is calculated. At the same time, 'erroneous' postcode siblings are identified. These are sibling parts that contain less than or equal to the *maxpts* value (set to 1 in the INI file).

The mean and standard deviation of the distances between the siblings and the main polygon for the split postcode are calculated, and are then used to generate two threshold distances: a reporting distance and error distance. Default values are used if there are only two parts in the split postcode, or if the calculated distance thresholds fall outside 'reasonable' ranges (500 – 2000m).

The distance thresholds are used to identify erroneous sibling parts of the split postcode whose separation from the main part exceeds these. If the reporting distance is exceeded, then the sibling is logged; if the error distance is exceeded, then the sibling is automatically merged with a neighbour with the longest common boundary, and ideally with a postcode, and not erroneously split.

A lot of information is logged by the script, including the maximum distance between postcode siblings, and counts of wards changed (i.e. where merges have occurred) and erroneous siblings merged and reported.

The merging or recoding actions that have been performed by the script are written out to *wp%wpnum%finalpc.txt* which lists the new postcode value for each polygon ID.

Edit_pc.aml

A manual intervention script that facilitates the ***editing of postcodes to 'fix' splits*** or ***group physical islands together, forcing contiguity for the purposes of the AZP***. Fixing split postcodes involves altering the postcode of one or more of the component fragments to another postcode that is adjacent to each. Adjusting contiguity rebuilds the AAT file to re-present the grouped island postcodes as being contiguous, but with a common boundary length of zero.

3. AZP Process

The AZP program gathers the polygons generated in the previous section together in aggregations that will eventually be output areas. The aggregation or clustering process is re-run iteratively in an attempt to optimise the results. The quality of the resulting aggregation is measured using a number of parameters – the population and household counts of the resulting OAs, a measure of the socio-economic homogeneity of the associated population/housing type and the shape of the OA geometry.

OAPS.INI Parameters

AZP_MAIN first attempts to open a log file in append mode then reads in various settings from the *read_ini()* function using two Windows functions *GetPrivateProfileString()* and *GetPrivateProfileInt()*. If it fails to find the entry in the *ini* file it reverts to a default.

Settings are read as follows:

1. Database connection parameters.
2. ***Iterations*** into *iters* which is currently set to ***100*** (default is 0)
3. ***Targetpop***[2] into *targetpop* which is currently set to ***125*** (default is 125)
4. ***MinPop*** into *confpop* which is currently set to ***100*** (default is 50)
5. ***MinHhd*** into *confhhd* which is currently set to ***40*** (default is 25)
6. ***PopWeight*** into *popweight* which is currently set to ***100*** (default is 100)
7. ***BndWeight*** into *bndweight* which is currently set to ***100*** (default is 100)
8. ***HomWeight*** into *homweight* which is currently set to ***100*** (default is 100)
9. ***MinBoundary*** into *minbndpc* which is currently set to ***10*** (default is 0)
10. ***MissedPCs*** into *maxmissct* which is currently set to ***0*** (default is 0)

Data Loading

The program works at the ward part level, with a start and end ward part number being supplied as program parameters. This means that grouped ward parts are handled together.

[2] Note that a revision to the code was made, late in the OAPS development process, which switched from a population-based target to a household-based one. Variable names do not seem to have been changed and hence can be misleading. For example, ***the targetpop variable actually stores a household target.*** We have not attempted to update the documentation accordingly, but this should be borne in mind when reviewing the code.

A series of functions are used to load the building block data into arrays – the information stored is as follows:

- ***Polygon attributes*** (*load_pat*) – from *wp%wpnum%pcpat.txt*. This array stores the assigned postcode, area, centroid and urban / rural class for each polygon.
- ***Adjacency information*** (*load_aat*) – from *wp%wpnum%pcaat.txt*. This contains the adjacent polygons for each polygon in the ward part, and the length of the common boundary.
- ***Address point locations*** (*load_pclist*) – from *wp%wpnum%pclist.txt*. These are not individually stored, but aggregated by the polygon that they fall within to give a count and address weighted centroid.
- ***Split postcodes*** (*calc_orphans*) – the polygon attribute list is read and checked for duplicate postcode values (indicative of a split). Each split part of a postcode is given a pointer to the ID of the next part to create a linked list that allows all siblings to be easily looped over.
- ***Vertical (STK) codes*** (*load_vertlookup*) – from *vert%wpnum%finalpc.txt*. A list of the real postcodes that comprise each STK code are stored.
- ***Population counts and homogeneity variables*** (*load_pcounts*) – from a database table *pcounts*. Values for population, household counts and the eleven homogeneity categories (tenure and accommodation type) are stored for each postcode, including those held in STK codes. The values for STK codes are summed from the constituent postcodes. Totals are calculated for all the variables for the ward part.

OA Generation

The optimum number of OAs to be generated is calculated before the generation process is started. This is defined as the total number of households divided by the target OA household count, rounded up to a whole number.

Doughnut Polygons

Consideration is made for so called doughnut polygons throughout the code. These occur at ward level, when a ward part completely encloses a different ward part and hence contains a hole. The term 'doughnut polygon' is used in the code to refer to the holes as discussed here. Due to the topological geometry system used by ArcInfo coverages, these holes will exist as polygons in the input data, but will have null attributes. This property of the holes is used to identify and flag them to be ignored during processing.

Initial Random Aggregation (IRA)

If the total population of the ward part is sufficiently low, then a single OA can be created – this occurs if the total population is less than 1.35 times the target population[3]. Neither household counts nor the lower thresholds are considered here.

[3] Note that this check seems to have not been altered in line with the late change from a population-based to a household-based target, thus resulting in an invalid comparison between the ward population and the target household count.

If the population is higher than this, then multiple OAs are created by examining and adding postcodes in turn. The start point for the postcode OA assignment is randomly set, but then the postcodes list is worked through in the order in which it was loaded.

The initial pass through the postcode list is concerned only with split postcodes – all parts of a split postcode are required to belong to the same OA. A new OA is started for each split postcode, though these may subsequently be merged with others.

To connect all the siblings of a split postcode, a search is made for the shortest path connecting a sibling back to the emergent OA containing its other part(s). The shortest path is defined as the one spanning the least number of intervening polygons[4]. Connections are rejected to adjacent polygons whose proportions by length of the polygons' boundaries fall below a minimum threshold. This avoids the creation of narrow necks in OAs. If the sibling that is being connected has a different urban/rural flag to the OA so far, then this factor is not considered when searching for a path, otherwise paths that involve polygons of the other type are avoided unless no solution is found, at which point this restriction is removed and the process re-run.

If the path found comprises polygons that have already been assigned to another OA, then that entire OA is subsumed into the current one. No checks are made on population or household thresholds in this operation, however the maximum permitted number of postcodes per OA (1001) may be reached, generating an error.

Following the inclusion of all split postcodes, the remaining unsplit postcode polygons are aggregated together. Again, a postcode is chosen at random to start a new OA, and valid neighbouring postcode polygons are added to it. Valid neighbours are those that have the same urban/rural class, haven't already been assigned to an OA, and have a shared boundary length that is greater than the specified proportion of the initial polygon boundary. Note that this is less stringent than the criteria used in the path searching as there the proportion test is applied from both sides of the merger. Neighbouring postcodes are added to the OA until there aren't any valid ones left, or either the household or population minimum thresholds are reached.

All postcodes should have now been assigned to an OA grouping, but household and population thresholds have not necessarily been observed during this.

The next step is to identify all OAs that fall below the thresholds and try to merge them into adjacent OAs. Merging potential is evaluated at the postcode polygon level, and initially mergers are only considered where the two OAs have the same urban/rural class, and a pair of postcode polygons from each have a common boundary length that is above the minimum threshold (only checked against the first polygon's perimeter, not both ways). If no merge candidates are found then the common boundary threshold is halved, and this is repeated up to three times before the urban/rural class requirement is relaxed. If more than one merging candidate is found, then the one with the lowest household count is chosen.

[4] Would the paths found ever vary between the different random start points? This is quite likely as the path is traced between each sibling and a polygon with the target OA code, the form of which changes with each sibling added and the order in which they are added. In simple cases (of only two siblings), if there were multiple paths between two siblings that had the same number of steps the first encountered is accepted, and this could vary with a number of random factors.

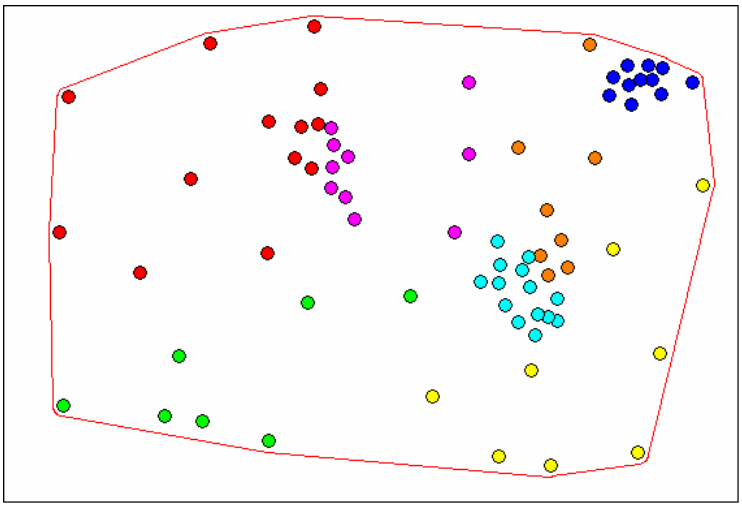
Following the merging of sub-threshold OAs, a second round of merging is performed to reduce the total number of OAs present down to the optimum number to bring the mean household count of the OAs as close to the target as possible. The OAs with the lowest populations are merged with neighbours, following the same rules described above, until the total number of OAs equals that required.

Note that throughout the IRA process, merging between OAs is considered at the postcode polygon level, one polygon at a time, not by using the total boundary length in common between the two OA groups.

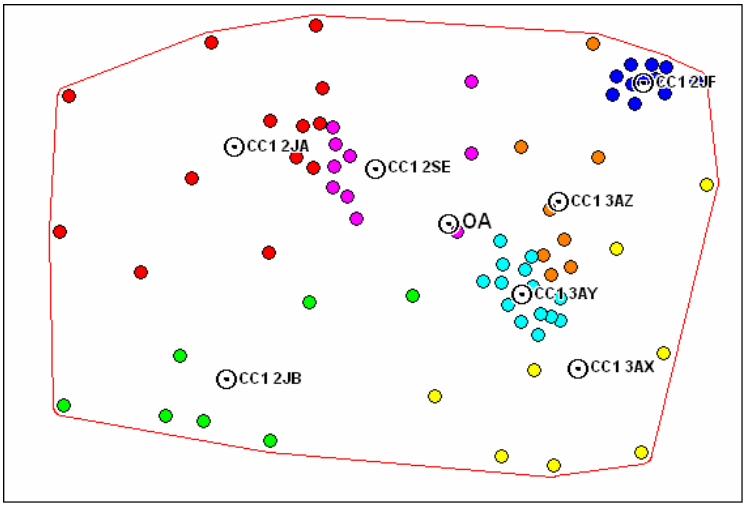
Quality metric calculation

For each OA, the following metrics are calculated. The metrics are used to compare the ***relative quality*** of OA groupings

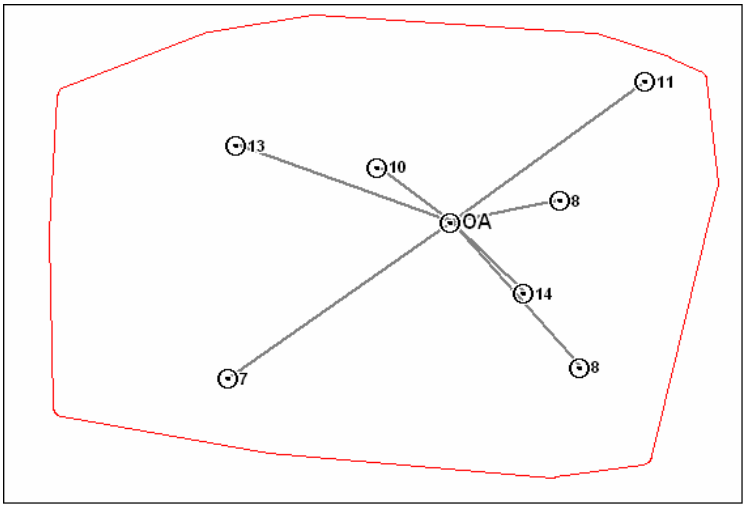
- ***Household Target Score – the squared difference of the OA household count from the target value*** – lower is better, signifying a closer fit to the target population.
- ***Shape (boundary) score – the sum of weighted squared differences between the OA and constituent postcode polygon address-count weighted centroids (the weighting being by the postcode address point count).*** A lower score is better, signifying a less dispersed collection of postcode polygons. A step by step illustration of the method used is given below:



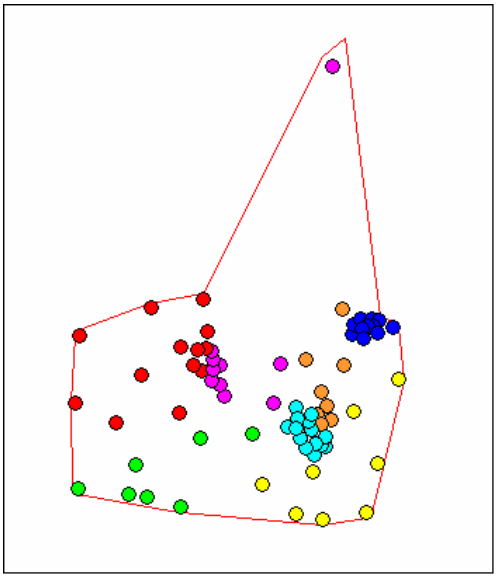
1. OA is formed of address points which belong to one or more different zones



2. Centroids of the OA and each component zone are calculated as the mean of the coordinates of the constituent address points.



3. The distance between each zone centroid and the OA centroid is calculated. The mean distance, weighted by the number of address points in each zone, is used as the shape metric.



4. As with all metrics, it has pros and cons. This metric may not be very sensitive to isolated outliers that can substantially change the shape of the OA. For example, the shifted address point in the diagram above only results in a 3% increase in the metric.

Intra-area correlation (IRA) score – Calculated as defined in Tranmer and Steel (1998) "Using census data to investigate the causes of the ecological fallacy" *Environment and Planning A* 30, 817-31. A higher score is better, signifying that variation in the homogeneity variables is occurring between OAs rather than between the postcode polygons within individual OAs.

Zone swapping

To improve on the quality of the initial random aggregation of postcode polygons into OAs, individual postcode polygons are swapped to adjacent OAs and the effect on the quality metrics examined.

An OA is chosen at random, and all postcode polygons that are valid swaps are found. A valid swap is a postcode polygon adjacent to the OA that isn't split, has the same urban/rural class as the OA, and has a common boundary length proportion greater than the threshold (checked both ways – against the perimeter of both polygons across the boundary). A check is also made to see whether removing the proposed swap from its existing OA will break that OA into unconnected parts (note that an OA is not considered connected across a boundary between two postcode polygons that is less than the threshold proportion, checked both ways).

Once built, the list of available swaps is worked through in the build order, and each swap is evaluated in turn. A swap out of an OA that reduces the population or household count below threshold is not allowed. The quality metrics are re-calculated for the new OAs formed after the swap, and a comparison is made with the original OAs using a weighted combination of the proportional changes in the quality metrics (note that for the final OAPS run to create the 2001 OAs the weightings were all equal, as set in the INI file). If an improvement is seen, then the swap is made immediately, and the properties of the two

OAs are updated. The list of available swaps is worked through with a swap being made every time an improvement is seen.

The swapping process is repeated on each OA in the ward part, in a random order. If any swaps were made throughout this and hence any improvement seen, then the whole process is repeated for all OAs, and this loop continues until no more swaps are made in an iteration.

Repeated Runs

At the completion of a polygon swapping optimisation phase the zone design process is repeated, starting from the Initial Random Aggregation. A total of 100 runs are performed. At the end of each run a relative quality of solution is calculated based on a weighted combination of the proportional changes in the quality metrics between the new solution and the best previous one. The best solution found from all iterations is output as a list of postcodes and their assigned OA codes, along with summary information.

Island Zones

In certain situations the contiguity file can be manually adjusted to force postcodes to be adjacent to one another: this is done using Edit_pc.aml. The common boundary length for these adjusted situations is set to zero. When the AAT file is read, the zero boundary lengths do not receive special handling, however their subsequent treatment in the OAPS code is inconsistent with respect to the minimum boundary threshold checks. In some situations a zero common boundary length is exempt from the minimum boundary threshold checks, in others it is not. A summary of the different situations is give below:

Exempt (zero length common boundaries will not be considered to be too short in these cases):

1. Path building when connecting split postcodes in initial random aggregation
2. Aggregating non-split, unassigned zones to OAs during the initial random aggregation
3. Looking for the neighbouring OA with the smallest population during the consolidation phase of the initial random aggregation
4. Building a list of candidate zones to swap between OAs in the zone swapping phase.

Not exempt (zero length common boundaries will be considered to be too short in these cases):

1. Updating the list of candidate neighbouring zones following the addition of a zone to the OA during the initial random aggregation (this is in contrast to the initial neighbour list generation, covered in item 2 above)
2. Checking contiguity between zones whilst attempting to swap a zone out of an OA during the zone swapping phase.

4. OA Polygon Creation

Build_oas.aml

This AML is used to generate the OAs from the AZP output.

- Spatial operations on the data include dissolving the zone boundaries to form OAs and removing slivers at ward part boundaries when appending child ward parts together.
- OA centroids calculated in the OAPS C code are adjusted so that they fall within the OA boundary by incrementally moving them towards the ARC generated geometric centroid, guaranteed to be within the polygon.
- Reports are generated of large and split postcodes
- OA codes are generated and uploaded to the FormID table using the postcode as the join field.