FROM COMPUTER CARTOGRAPHY TO SPATIAL VISUALIZATION: A NEW CARTOGRAM ALGORITHM

BIBLIOGRAPHIC SKETCH

Daniel Dorling currently holds a British Academy Fellowship at the University of Newcastle upon Tyne. His research interests include studying the geography of society, politics and housing; visualization, cartography and the analysis of censuses. After the completion of his PhD (entitled 'the visualization of spatial social structure') in 1991, he was awarded a Rowntree Foundation Fellowship. He graduated from Newcastle University in 1989 with a first class degree in Geography, Mathematics and Statistics.

Daniel Dorling
Department of Geography
University of Newcastle upon Tyne
England, NE1 7RU

ABSTRACT

Computer cartography is developing into spatial visualization, in which researchers can choose what they wish to see and how they wish to view it. Many spatial distributions require new methods of visualization for their effective exploration. Examples are given from the writer's work for the preparation of a new social atlas of Britain, which not only uses new statistics but employs radically different ways of envisioning information to show those statistics in a new light - using area cartograms depicting the characteristics of over ten thousand neighbourhoods simultaneously.

INTRODUCTION

Suppose that one could stretch a geographical map so that areas containing many people would appear large, and areas containing few people would appear small . .
Tobler, 1973, p.215

Suppose, now, that one could stretch a geographical map showing the characteristics of thousands of neighbourhoods such that each neighbourhood became visible as a distinct entity. The new map would be an area cartogram (Raisz 1934). On a traditional choropleth map of a country the shading of the largest cities can be identified only with difficulty. On an area cartogram every suburb and village becomes visible in a single image, illuminating the detailed geographical relationships nationally. This short paper presents and illustrates a new algorithm to produce area cartograms that are suitable for such visualization; and argues why cartograms should be used in the changing cartography of social geography.

Equal population cartograms are one solution to the visualization problems of social geography. The gross misrepresentation of many groups of people on conventional topographic maps has long been seen as a key problem for thematic cartography (Williams 1976). From epidemiology to political science, conventional maps are next to useless because they hide the residents of cities while massively overemphasising the characteristics of those living in the countryside (Selvin et al 1988). In mapping social geography we should represent the population equitably.

Visualization means making visible what can not easily be imagined or seen. The spatial structure of the social geography of a nation is an ideal subject for visualization as we wish to grasp simultaneously the detail and the whole picture in full. A population cartogram is the appropriate base for seeing how social characteristics are distributed spatially across people rather than land. Although the problems of creating more appropriate projections have emerged in many other areas of visualization (see Tufte 1990, Tukey 1965).

THE ALGORITHM

Cartograms have a longer history than the conventional topographic maps of today, but only in the last two decades have machines been harnessed to produce them (see for instance Tobler 1973, Dougenik et al 1985). Most cartograms used today are still drawn by hand because the cartographic quality of automated productions was too poor or could not show enough spatial detail. A key problem for visualization is that the maintenance of spatial contiguity could result in cartograms where most places were represented by strips of area too thin to be seen. This paper deals with non-continuous area cartograms (following Olson 1976) where each place is represented by a circle. The area of each circle is in proportion to the place's population and each circle borders as many of the place's correct geographical neighbours as possible (see Härö 1968).

The Pascal implementation of the algorithm is included as an appendix so that detailed cartograms can be produced for countries other than Britain. The algorithm begins by positioning a circle at the centroid of each place on a land map and then applies an iterative procedure to evolve the desired characteristics. All circles repel those with which they overlap while attracting those with whom they share a common border. Many more details are given in Dorling (1991). ***Figure 1*** shows the evolution of a cartogram of the 64 counties and regions of Britain using this algorithm - the areas, as circles, appear to spring into place. ***Figures 2 to 6*** illustrate various graphical uses to which the cartogram can be put, ranging from change and flow mapping, to depicting voting swings by arrows or the social characteristics of places with a crowd of Chernoff faces (the cartogram is also useful when animated, see Dorling 1992).

The true value of this new algorithm is not in producing cartograms of a few hundred areas, as manual solutions and older computer programs can already achieve this. A projection has never been drawn before, however, which can clearly make visible the social structure of thousands of neighbourhoods on a few square inches of paper. ***Figures 7 and 8*** use an equal land area map to show administrative boundaries while ***Figures 9 and 10*** show the same boundaries on a population cartogram. Each of the ten thousand local neighbourhoods (called wards) are visible on the cartogram and there is enough space to name the cities which can only be shown by dots on a conventional map of British counties.

Figures 11 and 12 show the ward cartogram being used to illustrate the spatial distribution of ethnic minorities in Britain. On the ward map it appears that almost everyone is white, with the most significant feature being two *ghettos* in the mountains of Scotland. This map is completely misleading, as are all maps of social geography based on an equal land area projection. Most people in Britain live in neighbourhoods which contain residents belonging to ethnic minorities. Their most significant concentrations are in Birmingham, Leicester, Manchester, Leeds and three areas of London, where "minorities" comprise more than a quarter of some inner city populations. Conventional maps are biased in terms of whose neighbourhoods they conceal.

The new algorithm has been used to create cartograms of over one hundred thousand areal units. To show social characteristics effectively upon these requires more space than is available here and also the use of colour (see Dorling 1992). ***Figures 13 and 14*** have used such a cartogram as a base to illustrate the spatial distribution of people in Britain following the method used by Tobler (1973) for the United States. Once a resolution such as this has been achieved, the cartogram can be viewed as a continuous transform and used for the mapping of incidences of disease or, for instance, the smooth reprojection of road and rail maps. At the limit — were each areal unit to comprise of the space occupied by a single person — the continuous and non-continuous projections would become one and the same.

Population area cartograms illuminate the most unlikely of subjects. Huge flow matrices can be envisioned with ease using simple graphics programming. ***Figure 15*** shows over a million of the most significant commuting flows between wards in England and Wales. The vast majority of flows are hidden within the cities. ***Figure 16*** reveals these through reprojecting the lines onto the ward cartogram. On the cartogram movement is everywhere and so the map darkens with the concentration of flows. Just as all that other commuters can see is commuters, so too that is all we can see on the cartogram. Equal population projections are not always ideal.

The algorithm used to create these illustrations is included as a two page appendix. The author hopes that it will be used by other researchers to reproject the maps of other countries - using the United States's counties or the communes of France for example. The program requires the contiguity matrix, centroids and populations of the areas to be reprojected. It produces a transformed list of centroids and a radius for the circle needed to represent each place (its area being in proportion to that place's population). The cartograms shown here were created and drawn on a microcomputer costing less than $800.

CONCLUSION

The creation and use of high resolution population cartograms moves computer cartography towards spatial visualization. The age old constraints that come from conventional projections are broken as we move beyond the paper map to choose what and how we wish to view the spatial structure of society (Goodchild 1988). Conventional projections are not only uninformative, they are unjust — exaggerating the prevalence of a few people's lifestyles at the expense of the representation of those who live inside our cities, and hence presenting a bias view of society as a whole. If we wish to see clearly the detailed spread of disease, the wishes of the electorate, the existence of poverty or the concentration of wealth, then we must first develop a projection upon which such things are visible. The algorithm presented here creates that projection.

REFERENCES

Dorling, D. (1991) The visualization of spatial social structure, unpublished PhD thesis, Department of Geography, University of Newcastle upon Tyne.

Dorling, D. (1992) Stretching space and splicing time: from cartographic animation to interactive visualization, Cartography and Geographic Information Systems, Vol.19, No.4, pp.215-227, 267-270.

Dougenik, J.A., Chrisman NR. & Niemeyer, D.R. (1985) An algorithm to construct continuous area cartograms, Professional Geographer, Vol.37, No.1, pp.75-81.

Goodchild, M.F. (1988) Stepping over the line: technological constraints and the new cartography, The American Cartographer, Vol.15, No.3, pp.311-319.

Härö, A.S. (1968) Area cartograms of the SMSA population of the United States, Annals of the Association of American Geographers, Vol.58, pp.452-460.

Olson, J. (1976) Noncontiguous area cartograms, The Professional Geographer, Vol.28, pp.371-380.

Selvin, S., Merrill, D.W., Schulman, J., Sacks, S., Bedell, L. & Wong, L. (1988) Transformations of maps to investigate clusters of disease, Social Sciences in Medicine, Vol.26, No.2, pp.215-221.

Raisz, E. (1934) The rectangular statistical cartogram, The Geographical review, Vol.24, pp.292-296.

Tobler, W.R. (1973) A continuous transformation useful for districting, Annals of the New York Academy of Sciences, Vol.219, pp.215-220.

Tufte, E.R. (1990) Envisioning information, Graphics Press, Cheshire, Connecticut.

Tukey, J.W. (1965) The future process of data analysis, Proceedings of the tenth conference on the design of experiments in army research development and testing, report 65-3, Durham NC., US army research office, pp.691-725.

Williams, R.L. (1976) The misuse of area in mapping census-type numbers, Historical Methods Newsletter, Vol.9, No4, pp.213-216.

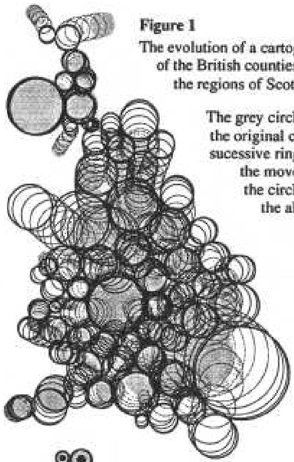


Figure 1
The evolution of a cartogram of the British counties and the regions of Scotland.

The grey circles lie on the original centroids, sucessive rings depict the movement of the circles under the algorithm.

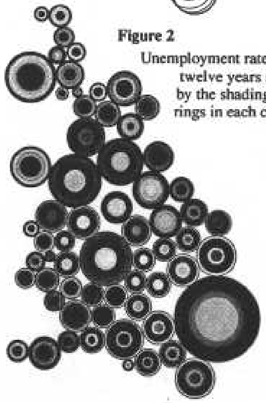


Figure 2
Unemployment rates over twelve years shown by the shading of 12 rings in each county.

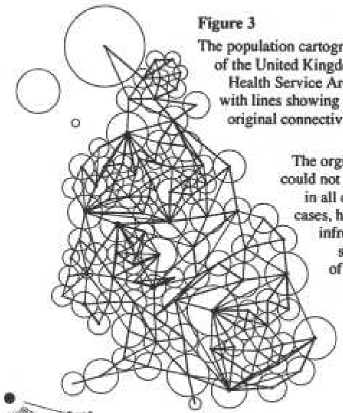


Figure 3
The population cartogram of the United Kingdom Health Service Areas with lines showing the original connectivity.

The orginal contiguity could not be maintained in all cases; difficult cases, however, occur infrequency given sufficient areas of roughly equal populations.

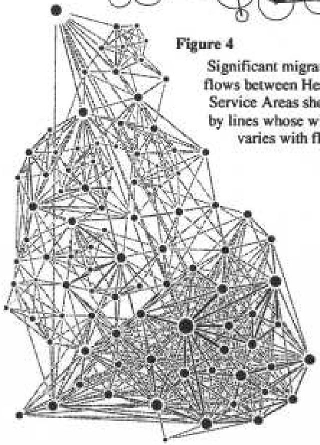


Figure 4
Significant migration flows between Health Service Areas shown by lines whose width varies with flow.

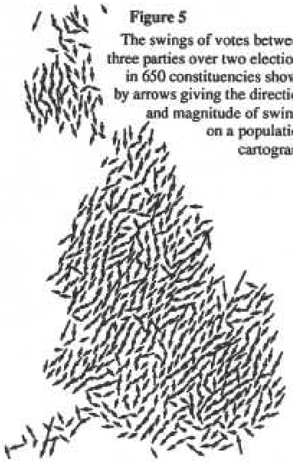


Figure 5
The swings of votes between three parties over two elections in 650 constituencies shown by arrows giving the direction and magnitude of swings on a population cartogram.

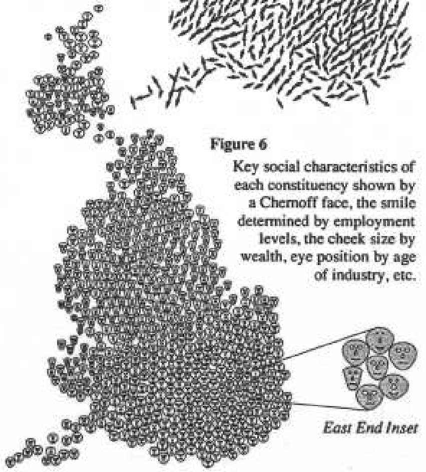
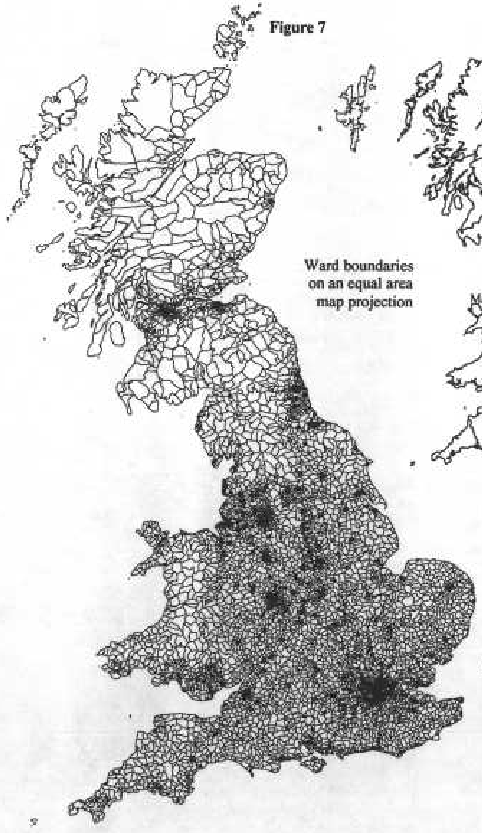


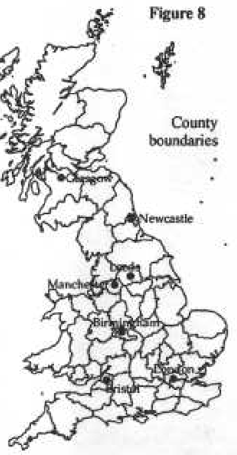
Figure 6
Key social characteristics of each constituency shown by a Chernoff face, the smile determined by employment levels, the cheek size by wealth, eye position by age of industry, etc.

Figure 7



Ward boundaries on an equal area map projection

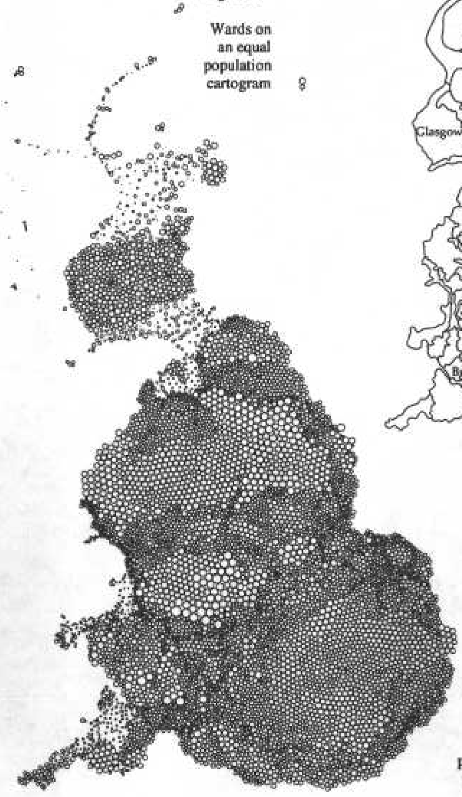
Figure 8



County boundaries

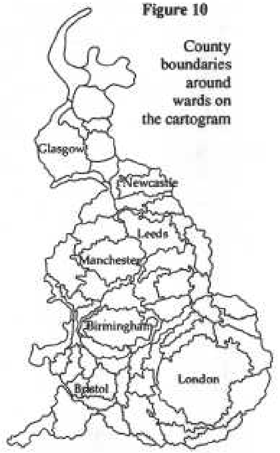
The vast majority of wards are simply not visible on a conventional map no matter how large it is drawn or how many insets are included. Those areas which can be seen usually contain the least people while those which are not visible conceal the largest populations.

Figure 9



Wards on an equal population cartogram

Figure 10



County boundaries around wards on the cartogram

On the equal popualtion cartogram each ward is shown by a circle the area of which is in proportion to its population. The reprojected county boundaries that result are shown in Figure 10. City wards tend to contain more people and hence are larger. The circles do not coalesce in the least populated areas of Scotland, Wales and the South West.

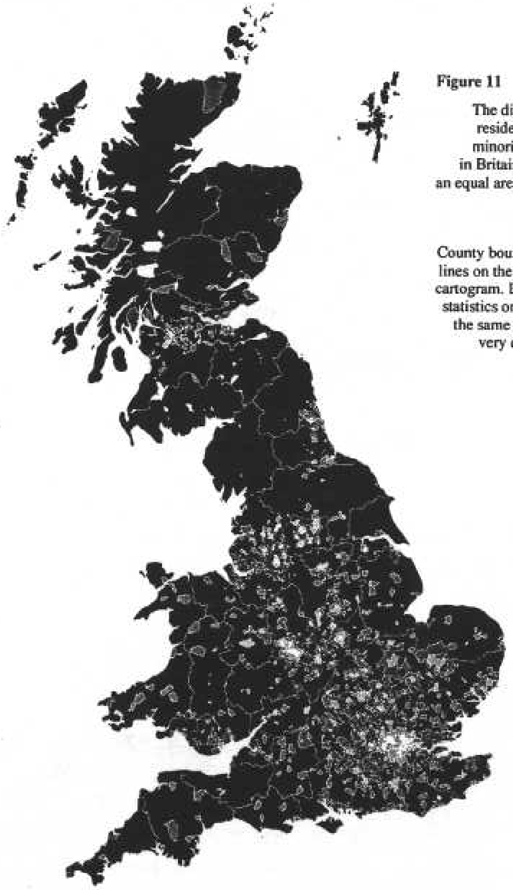


Figure 11

The distribution of residents in ethnic minorities by ward in Britain in 1991 on an equal area projection.

County boundaries are shown by white lines on the map and black lines on the cartogram. Both images show the same statistics on the same areal units using the same shading classes but present very different impressions of the ethnic diversity of Britain because of the different projections upon which each is based.

Scale

☐ = 1,000 km²

residents in ethnic minorities by ward

☐	over 25%
▨	5% to 25%
▓	1% to 5%
■	under 1%

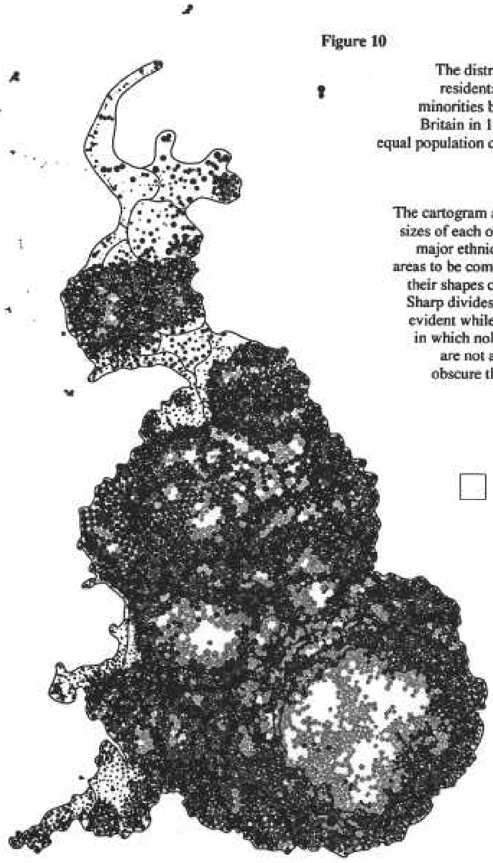


Figure 10

The distribution of residents in ethnic minorities by ward in Britain in 1991 on an equal population cartogram.

The cartogram allows the sizes of each of Britain's major ethnic minority areas to be compared and their shapes contrasted. Sharp divides are made evident while the areas in which nobody lives are not allowed to obscure the image.

Scale

☐ = 250,000 people

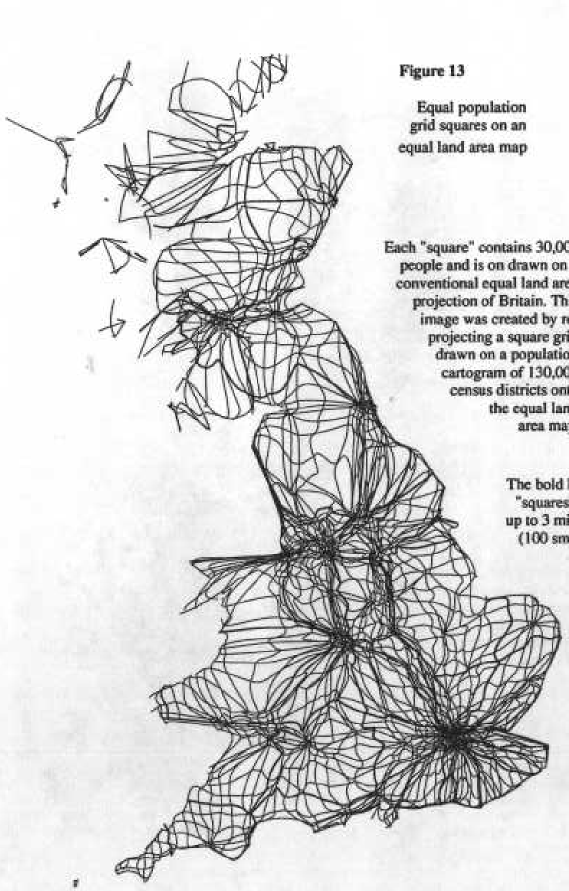


Figure 13

Equal population grid squares on an equal land area map

Each "square" contains 30,000 people and is on drawn on a conventional equal land area projection of Britain. This image was created by re-projecting a square grid drawn on a population cartogram of 130,000 census districts onto the equal land area map.

The bold lines delimit "squares" containing up to 3 million people (100 small squares).

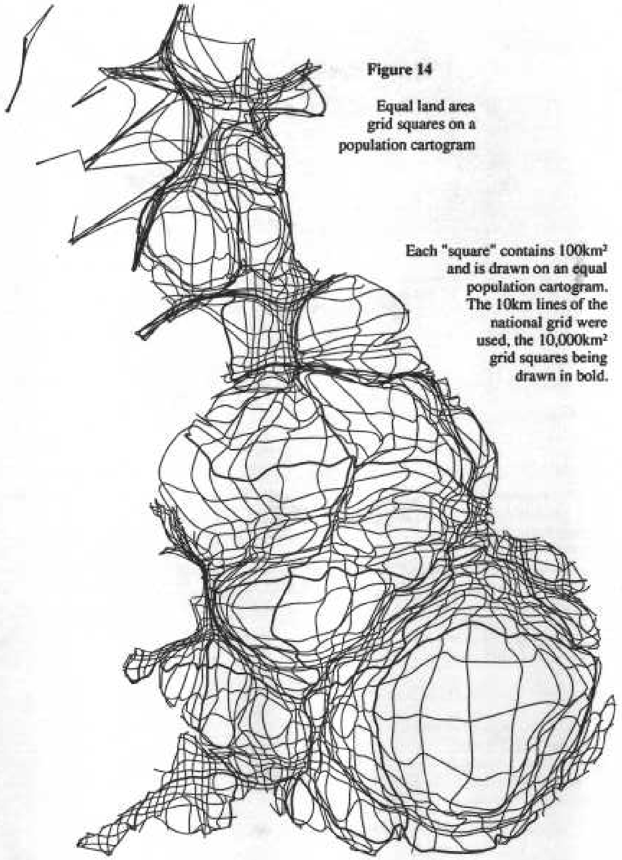


Figure 14

Equal land area grid squares on a population cartogram

Each "square" contains 100km^2 and is drawn on an equal population cartogram. The 10km lines of the national grid were used, the 10,000km^2 grid squares being drawn in bold.

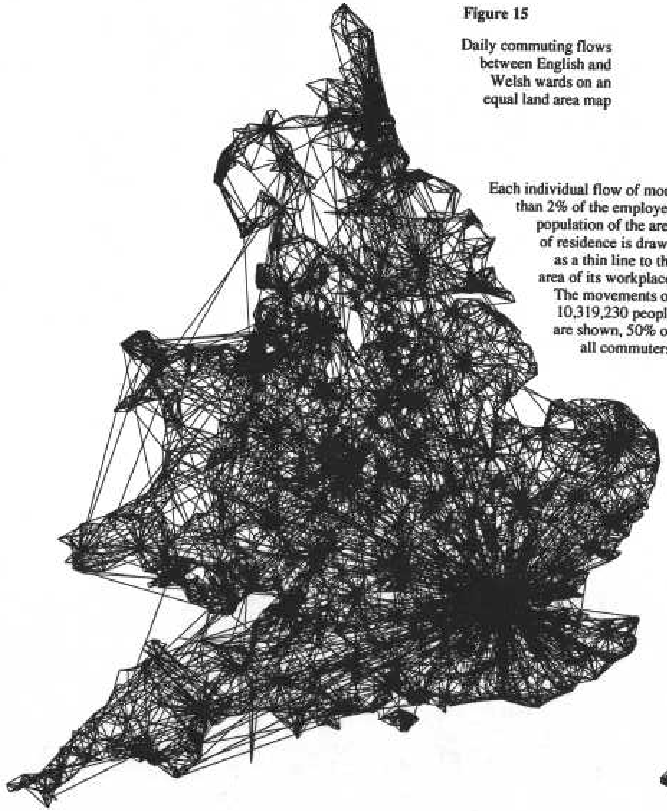


Figure 15

Daily commuting flows between English and Welsh wards on an equal land area map

Each individual flow of more than 2% of the employed population of the area of residence is drawn as a thin line to the area of its workplace. The movements of 10,319,230 people are shown, 50% of all commuters.

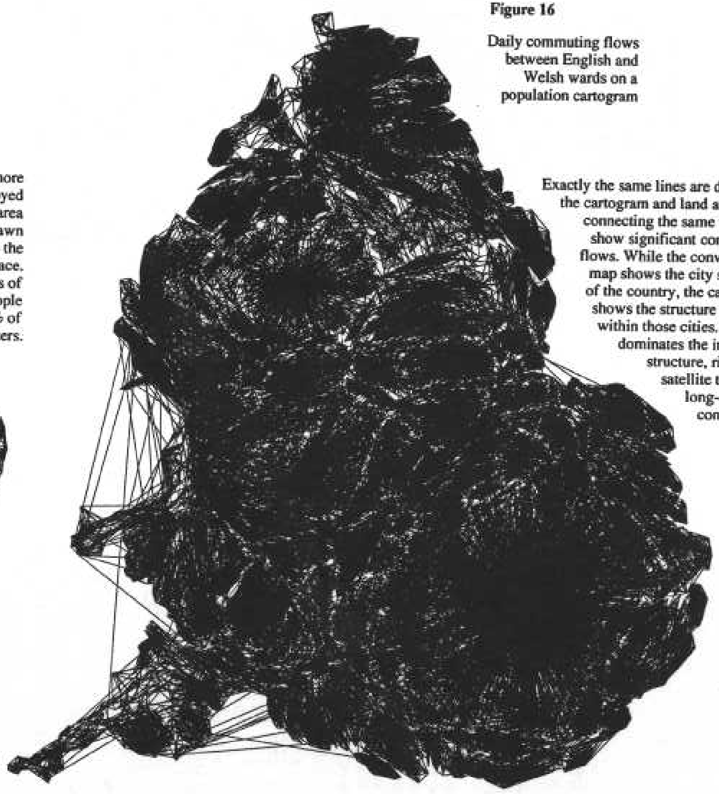


Figure 16

Daily commuting flows between English and Welsh wards on a population cartogram

Exactly the same lines are drawn on the cartogram and land area map, connecting the same wards to show significant commuting flows. While the conventional map shows the city structure of the country, the cartogram shows the structure of flows within those cities. London dominates the industrial structure, ringed by satellite towns of long-distance commuters.

Appendix: Cartogram Algorithm

Being distributed for free academic use only.

```
program cartogram (output);

{Pascal implementation of the cartogram algorithm}

{Expects, as input, a comma-separated-value text
 file giving each zone's number, name, population,
 x and y centroid, the number of neighbouring zones
 and the number and border length of each neighbouring
 zone. Outputs a radius and new centroid for each zone.
 The two recursive procedures and a tree structure are
 include to increase the efficiency of the program.}

{Constants are currently set for the 10,444 1981 census
 wards of Great Britain and for 15,000 iterations of the
 main procedure- exact convergence criteria are unknown.
 Wards do actually converge quite quickly - there
 are no problems with the algorithm's speed -
 it appears to move from O(n^2) to O(n log n)
 until other factors come into play when n
 exceeds about 100,000 zones.}

const
  iters = 15000;
  zones = 10444;
  ratio = 0.4;
  friction = 0.25;
  pi = 3.141592654;

type
  vector  = array [1..zones] of real;
  index   = array [1..zones] of integer;
  vectors = array [1..zones, 1..21] of real;
  indexes = array [1..zones, 1..21] of integer;
  leaves  = record
              id        : integer;
              xpos      : real;
              ypos      : real;
              left      : integer;
              right     : integer;
  end;
  trees = array [1..zones] of leaves;

var
  infile, outfile             : text;
  list                        : index;
  tree                        : trees;
  widest, dist                : real;
  closest, overlap            : real;
  xrepel, yrepel, xd, yd      : real;
  xattract, yattract          : real;
  displacement                : real;
  atrdst, repdst              : real;
  total_dist                  : real;
  total_radius, scale         : real;
  xtotal, ytotal              : real;
  zone, nb                    : integer;
  other, itter                : integer;
  end_pointer, number         : integer;
  x, y                        : index;
  xvector, yvector            : vector;
  perimeter, people, radius   : vector;
  border                      : vectors;
  nbours                      : index;
  nbour                       : indexes;

 {Recursive procedure to add global variable 'zone' to}
 {the "tree" which is used to find nearest neighbours}
 procedure add_point(pointer,axis :integer);
     begin
       if tree[pointer].id = 0 then
         begin
           tree[pointer].id    := zone;
           tree[pointer].left  := 0;
           tree[pointer].right:= 0;
           tree[pointer].xpos  := x[zone];
           tree[pointer].ypos  := y[zone];
         end
     else
       if axis = 1 then
         if x[zone] >= tree[pointer].xpos then
           begin
             if tree[pointer].left = 0 then
               begin
                 end_pointer := end_pointer +1;
                 tree[pointer].left := end_pointer;
               end;
             add_point(tree[pointer].left,3-axis);
           end
         else
           begin
             if tree[pointer].right = 0 then
               begin
                 end_pointer := end_pointer +1;
                 tree[pointer].right := end_pointer;
               end;
             add_point(tree[pointer].right,3-axis);
           end
       else
         if y[zone] >= tree[pointer].ypos then
           begin
             if tree[pointer].left = 0 then
               begin
                 end_pointer := end_pointer +1;
                 tree[pointer].left := end_pointer;
               end;
             add_point(tree[pointer].left,3-axis);
           end
         else
           begin
             if tree[pointer].right = 0 then
               begin
                 end_pointer := end_pointer +1;
                 tree[pointer].right := end_pointer;
               end;
             add_point(tree[pointer].right,3-axis);
           end
     end;

{Procedure recursively recovers the "list" of zones}
{within "dist" horizontally or vertically of the "zone",}
{from the "tree". The list length is given by the integer}
{"number". All global variables exist prior to invocation}
 procedure get_point(pointer, axis :integer);
   begin
     if pointer>0 then
       if tree[pointer].id > 0 then
         begin
           if axis = 1 then
             begin
               if x[zone]-dist < tree[pointer].xpos then
                 get_point(tree[pointer].right,3-axis);
               if x[zone]+dist >= tree[pointer].xpos then
                 get_point(tree[pointer].left,3-axis);
             end;
           if axis = 2 then
             begin
               if y[zone]-dist < tree[pointer].ypos then
                 get_point(tree[pointer].right,3-axis);
               if y[zone]+dist >= tree[pointer].ypos then
                 get_point(tree[pointer].left,3-axis);
             end;
           if (x[zone]-dist < tree[pointer].xpos)
             and (x[zone]+dist>=tree[pointer].xpos) then
             if (y[zone]-dist < tree[pointer].ypos)
               and(y[zone]+dist>=tree[pointer].ypos) then
               begin
                 number := number +1;
                 list[number] := tree[pointer].id;
               end;
         end;
   end;
```

```
{The main program}
   begin
     reset(infile,'FILE=ward.in');
     rewrite(outfile,'FILE=ward.out');
     total_dist :=0;
     total_radius := 0;

     for zone := 1 to zones do
       begin
         read(infile,people[zone],x[zone],y[zone],nbours[zone]);
         perimeter[zone] := 0;
         for nb := 1 to nbours[zone] do
           begin
             read(infile,nbour[zone,nb], border[zone,nb]);
             perimeter[zone]:=perimeter[zone]+border[zone,nb];
             if nbour[zone,nb] > 0 then
               if nbour[zone,nb] < zone then
                 begin
                   xd := x[zone]- x[nbour[zone,nb]];
                   yd := y[zone]- y[nbour[zone,nb]];
                   total_dist := total_dist + sqrt(xd*xd+yd*yd);
                   total_radius := total_radius +
          sqrt(people[zone]/pi)+sqrt(people[nbour[zone,nb]]/pi);
                 end;
           end;
        readln(infile);
      end;
   writeln ('Finished reading in topology');

   scale := total_dist / total_radius;
   widest := 0;
   for zone := 1 to zones do
     begin
       radius[zone] := scale * sqrt(people[zone]/pi);
       if radius[zone] > widest then
         widest := radius[zone];
       xvector[zone] := 0;
       yvector[zone] := 0;
     end;
   writeln ('Scaling by ',scale,' widest is ',widest);

{Main iteration loop of cartogram algorithm.}
   for itter := 1 to iters do
     begin


       for zone := 1 to zones do
         tree[zone].id := 0;
       end_pointer := 1;
       for zone := 1 to zones do
         add_point(1,1);

       displacement := 0.0;

{Loop of independent displacements- could run in parallel.}
       for zone := 1 to zones do
         begin
           xrepel   := 0.0;
           yrepel   := 0.0;
           xattract := 0.0;
           yattract := 0.0;
           closest := widest;
 {Retrieve points within widest+radius(zone) of "zone"}
 {to "list" which will be of length "number".}
           number := 0;
           dist := widest + radius[zone];
           get_point(1,1);
{Calculate repelling force of overlapping neighbours.}
           if number > 0 then
             for nb := 1 to number do
               begin
                 other := list[nb];
                 if other <> zone then
                   begin
                     xd := x[zone]-x[other];
                     yd := y[zone]-y[other];
                     dist := sqrt(xd * xd + yd * yd);
                     if dist < closest then
                       closest := dist;
                     overlap:=radius[zone]+radius[other]-dist;
                     if overlap > 0.0 then
                      if dist > 1.0 then
                       begin
                        xrepel:=xrepel-
                               overlap*(x[other]-x[zone])/dist;
                        yrepel:=yrepel-
                               overlap*(y[other]-y[zone])/dist;
                       end;
                   end;
               end;
 {Calculate forces of attraction between neighbours.}
           for nb := 1 to nbours[zone] do
             begin
               other := nbour[zone,nb];
               if other <> 0 then
                 begin
                   xd := x[zone]-x[other];
                   yd := y[zone]-y[other];
                   dist := sqrt(xd * xd + yd * yd);
                   overlap:=dist-radius[zone]-radius[other];
                   if overlap > 0.0 then
                    begin
                     overlap := overlap*
                             border[zone,nb]/perimeter[zone];
                     xattract:=xattract+
                             overlap*(x[other]-x[zone])/dist;
                     yattract:=yattract+
                             overlap*(y[other]-y[zone])/dist;
                    end;
                 end;
             end;
{Calculate the combined effect of attraction and repulsion.}
           atrdst := sqrt(xattract*xattract+yattract*yattract);
           repdst := sqrt(xrepel*xrepel+yrepel*yrepel);
           if repdst > closest then
             begin
               xrepel := closest * xrepel / (repdst + 1);
               yrepel := closest * yrepel / (repdst + 1);
               repdst := closest;
             end;
           if repdst > 0 then
             begin
               xtotal:=(1-ratio)*xrepel+
                    ratio*(repdst*xattract/(atrdst+1));
               ytotal:=(1-ratio)*yrepel+
                    ratio*(repdst*yattract/(atrdst+1));
             end
           else
             begin
               if atrdst > closest then
                 begin
                   xattract := closest*xattract/(atrdst+1);
                   yattract := closest*yattract/(atrdst+1);
                 end;
               xtotal := xattract;
               ytotal := yattract;
             end;
{Record the vector.}
           xvector[zone]:= friction *(xvector[zone]+xtotal);
           yvector[zone]:= friction *(yvector[zone]+ytotal);
           displacement := displacement+
                        sqrt(xtotal*xtotal+ytotal*ytotal);
         end;

{Update the positions.}
       for zone := 1 to zones do
         begin
           x[zone] := x[zone] + round(xvector[zone]);
           y[zone] := y[zone] + round(yvector[zone]);
         end;
       displacement := displacement / zones;
       writeln('Iter: ', iter, ' disp: ', displacement);
     end;
{Having finished the iterations write out the new file.}
   for zone := 1 to zones do
     writeln(outfile,radius[zone]:9:0,',',x[zone]:9,
                                         ',',y[zone]:9);
 end.
```