GeoInformatica 2:4, 383–407 (1998)

Conflict Reduction in Map Generalization Using Iterative Improvement

J. MARK WARE AND CHRISTOPHER B. JONES
School of Computing, University of Glamorgan, Pontypridd CF37 1DL, Wales, U.K.
E-mail: jmware@glam.ac.uk

Received February 19, 1998; Revised July 1, 1998; Accepted August 12, 1998

Abstract

Map data are usually derived from a source that is based on a particular scale of representation and hence are subject to a particular degree of map generalization. Attempts to display data at scales smaller than the source can result in spatial conflict, whereby map symbols become too close or overlap. Several map generalization operators may be applied to resolve the problem, including displacement. In this paper we address the problem of displacing multiple map objects in order to resolve graphic conflict. Each of n objects is assigned k candidate positions into which it can possibly move, resulting in a total of k^n map realizations. The assumption is that some of these realizations will contain a reduced level of conflict. Generating and evaluating all realizations is however not practical, even for relatively small values of n and k. We present two iterative improvement algorithms, which limit the number of realizations processed. The first algorithm adopts a steepest gradient descent approach; the second uses simulated annealing. They are tested on a number of data sets and while both are successful in reducing conflict while limiting the number of realizations that are examined, the simulated annealing approach is superior with regard to the degree of conflict reduction. The approach adopted is regarded as generic, in the context of map generalization, in that it appears possible in principle to employ several map generalization operators combined with more sophisticated evaluation functions.

Keywords: conflict removal, scale reduction, search algorithms, gradient descent, simulated annealing, trial positions

1. Introduction

In the field of computer cartography, automation of the process of map generalization has come to assume increasing importance. The problem is one of selecting and adjusting the symbols on a map to suit the purpose of the map and the scale of the required output [15]. This task has traditionally been performed by cartographers and it is reflected in the widespread publication of maps that have a nominal scale that affects the level of detail and the degree of abstraction that may be expected on the map. The now widespread use of geographical information systems (GIS), with their integral capacity for producing maps, has introduced a requirement to include a facility for map generalization within these systems.

Much of the digital map data in use in GIS and in map production systems is in some sense pre-generalized, having in many cases been derived from cartographic products that are based on a particular scale of representation. Problems arise, however, when the user

requires a map at a scale significantly smaller than that associated with the source data. Typical consequences of displaying pre-generalized data at a smaller scale than was originally intended are that the detail of individual map features may become too small to be legible, while neighboring symbols that should be clearly discriminated may become too close, or they may overlap each other. These latter problems of graphic conflict arise in particular when the map symbols are no longer a true scale representation of the feature they represent. For example a road symbol may be much wider, when map scale is taken into account, than the width of the road on the ground.

In recent years considerable efforts have been put into automating map generalization (collections of papers dedicated to map generalization are to be found in [4], [12], [21]). Most of this effort has, however, been directed at the automation of individual operators required to perform generalization. In summary, these operators carry out tasks that include the following [18]:

- Reduction of the detail in linear features and boundaries.
- Elimination of features that may be too small to discern.
- Collapse in the dimensionality of areal features to lines or points.
- Amalgamation of adjacent features of the same or similar category.
- Exaggeration of important features otherwise too small to represent.
- Typification (or caricature) of the form of features as part of the process of detail reduction.
- Displacement of adjacent features that are in graphic conflict with each other.

In high quality map production the need for full automation is open to question, as it may be argued that the mapmaker, a cartographer, must be able to exercise judgment in creating an effective map design. Nevertheless, many GIS applications, and the increasing market for maps on the Internet, have created a demand for automated map generalization that adapts to the requirements of the user, in the absence of a cartographer. This motivates the pursuit of automation of the entire process of map generalization.

Automation of some of the individual generalization operators can be found in commercial GIS systems such as Intergraph's Map Generalizer and Laser-Scan's Lamps2. Use of the operators, however, still requires manual process control that involves the user in deciding which operators to apply, in which order, and how they should be applied in terms of relevant distance tolerances and other control parameters. Automating process control is essential if map generalization is to graduate from an interactive user-controlled procedure to one that is fully automated [1].

There are considerable difficulties in automating process control. This is because an effective map is one in which care has been taken to address the interactions between all map symbols, rather than treating them in isolation. These interactions may give rise to obvious graphic conflicts of proximity and overlap. They may also determine whether important messages, regarding the structure and form of the mapped features, are effectively communicated. The latter interaction could be expressed for example in the alignment of buildings, clustering of woods and lakes, and parallelism between neighboring rivers and roads. It may be that the problems of graphic conflict can be

addressed by a combination of possible actions such as elimination, displacement, amalgamation and boundary simplification, combined with appropriate techniques for evaluating the quality of the result. However, a problem with application of individual operators is that each time one of them is applied it may have an effect on a map symbol that was not previously in conflict, resulting in propagation of conflict within the map space. Thus an important aspect of process control is the need for effective strategies for conflict resolution in combination with appropriate quality evaluation.

The purpose of this paper is to present the results of some experiments concerned with resolving graphic conflict between multiple map objects. We consider the particular problem of areal map objects, such as buildings that may be too close to, or overlapping, each other. As indicated above, there are several types of action that may be taken to resolve conflict. Here we confine ourselves to the use of a displacement operator applied to rigid objects (figure 1). We use the operator to generate for each object a set of candidate or trial positions, each of which may be regarded as a possible state for that object. The approach adopted is notable for applying experience gained in developing conflict resolution procedures in the sub-problem of automated text placement [6], [7], [8], [23]. Just as in that context, automation of map generalization can be regarded as an attempt to meet a set of constraints [16]. Provided the constraints can be quantified, then an evaluation function (or objective function) can be constructed to determine the degree to which the constraints are met, thereby selecting map solutions that are superior to others

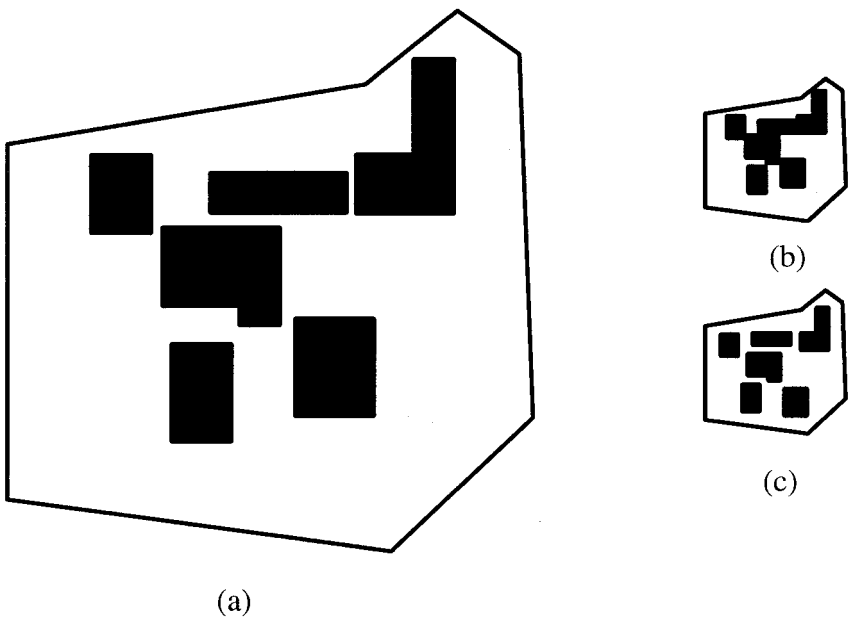

Figure 1. Resolving conflict by object displacement. (a) Map at original scale. (b) Map at approximately 1/4 its original scale—some objects become indistinguishable. (c) Map after object displacement—conflict has been removed.

with regard to the constraints. In demonstrating the validity of the approach to the limited problem considered here, it is envisaged that it may be extended with multiple generalization operators and with more complex evaluation functions.

We examine the use of two optimization techniques, steepest gradient descent and simulated annealing, and present results of the use of the techniques. A demanding aspect of the application of such search procedures is the need to evaluate numerous alternative states of the map. This requires efficient procedures to perform tasks such as finding and measuring distances to nearest neighbors of a given map object and detecting overlap between map objects. Here we exploit triangulation-based procedures that have been used previously to implement individual map generalization operators [9], [19].

In the remainder of the paper we start in Section 2 by presenting map conflict resolution as a search problem and introducing some relevant optimization procedures. We then summarize in Section 3 the characteristics of the generalization toolkit and the associated triangulation-based data structures that we use to experiment with the conflict resolution procedures. Both techniques that we apply make use of a conflict detection procedure, which is summarized in Section 3. In Section 4 we describe the main properties of the gradient descent and simulated annealing procedures, focusing on their application to map object conflict reduction. This is followed in Section 5 by details of experimental results and related discussion. The paper concludes with some closing remarks, which include the potential for future related work.

2. Map conflict resolution as a search problem

When map generalization is performed, each map object at the source scale is subject to a set of possible transformations that alter the state of the object, through the individual operations enumerated in the Introduction. Some of these operations may be largely predetermined, given a specific target scale, in that there may be fixed levels of sinuosity and fixed sizes for certain map symbols, such as a line representing a particular class of road. Other operations may not be entirely predetermined in that they depend upon the interaction between multiple map symbols. Thus, there is a choice of possible states for individual objects, the most appropriate only being determined by the combined states of multiple objects. From an artificial intelligence perspective, the set of all possible states in which the map could find itself as a consequence of considering each individual object state (including those that resulted in objects being deleted or merged to form new objects) constitutes a search space. If we also note that each possible map state is subject to evaluation with regard to the extent that it satisfies the cartographic objectives, then it is possible to envisage applying the various optimization procedures that are available (e.g., [17], [22]) to find the map state that best meets the objectives.

In the case of map generalization it is easy to see that the problem is a demanding one in that there could be a very large number of possible states to consider. If each of n objects was subject to k possible states then there are a total of k^n alternative realizations of the map. Christensen et al. [6] have presented the cartographic name placement problem in terms of a search space, an objective function and various possible optimization methods

that can be used to find a satisfactory solution. The possible states of each label may be generated from a set of positions relative to the labeled features, each of which may be prioritized. Evaluation of an objective function may be carried out using factors such as the number and degree of overlaps between labels, and between labels and map features, in combination with the associated individual priorities. As Christensen et al. [6] have pointed out, the problem for any realistic number of point-labeled features only is NP-hard. Attempts to solve the problem by an exhaustive search of all possible states are therefore impractical and it is necessary to consider sub-optimal strategies.

In map generalization the number of possible states can be expected to be considerably larger than within the sub-problem of name placement, due to the range of possible operations that could be applied to each object. The cost of evaluating the objective function can also be expected to be greater due to the range of possible constraints that could be applied. Thus evaluation could require not just measuring separation distances and testing for overlap, but also examining the shape of map features and analyzing patterns based on multiple objects. The cost of generating each state is also potentially large (as it can be in name placement for linear and area objects) since it could involve complex computational geometry procedures, such as those for medial axis transformation and for amalgamation of neighboring features.

2.1. Iterative improvement algorithms

A well-established approach to solving large optimization problems is to adopt an iterative improvement algorithm. The concept of an iterative algorithm is illustrated by Russell and Norvig [17] by considering all states (i.e., in our case, all map realizations) to be laid out on the surface of a landscape. The elevation at any point on the landscape represents the quality measure returned by the evaluation function for the particular state at that point. An iterative improvement algorithm will move around the landscape in an attempt to find the lowest troughs, which correspond, to optimal states. Iterative improvement algorithms fall into two main categories, namely, gradient descent and simulated annealing (see [13], [14], [17] for good reviews of these, and other, search algorithms). Gradient descent algorithms always make changes that improve the current state (i.e., movement is always downhill), whereas simulated annealing algorithms can sometimes make changes that make things worse (i.e., movement is sometimes uphill).

Simple gradient descent. Figure 2 describes a simple gradient descent implementation, based on the simple hill climbing algorithm given by Rich and Knight [14]. The algorithm is quite straightforward, but is not guaranteed to find an optimal solution since it is possible to arrive at a non-optimal current state from which no better state can be reached. This occurs when the search descends into a local minimum, from which all moves appear to generate a worse state. To use the landscape analogy once more, a local minimum can be thought of as a trough in the landscape that happens to be higher than the lowest point on the landscape. Several ways of trying to deal with the problem of local minima are

```
Algorithm Simple_Gradient_Descent(IN: initial_state; OUT: final_state)
Evaluate initial_state
If (initial_state = solution) then
  final_state ← initial_state
Else
  current_state ← initial_state
  Do
    Select operator that has not yet been applied to current_state
    Apply operator to produce a new_state
    Evaluate new_state
    If (new_state is better than current_state) then
      current_state ← new_state
    Endif
  Until (current_state = solution) or (no new operators left to apply)
  final_state ← current_state
Endif
```

Figure 2. Simple gradient descent algorithm.

available (e.g., random-restart, backtracking and multiple-moves). However, the exponential nature of most realistic search spaces can make such remedies impractical.

Simulated annealing. Searches based on simulated annealing [11] attempt to overcome the problem of getting caught in local minima. They achieve this by sometimes allowing uphill moves (and also neutral moves) to be made. In other words, the current state is sometimes allowed to get worse (or move to some other equivalent state). As can be seen from the algorithm outline (figure 3), the structure of the simulated annealing algorithm is quite similar to that of the gradient descent algorithm. As with gradient descent, simulated annealing always accepts a new state if it is better than the current state. However, in cases where the new state is, in evaluation function terms, worse than or equal to the current state then simulated annealing will sometimes accept the new state with some probability P less than 1. This probability is defined as

$$P = e^{-\Delta E/T}.$$

ΔE represents the ''badness'' of the new state (i.e., the amount by which the evaluation function is worsened). P decreases exponentially as ΔE increases (i.e., a slightly worse new state is more likely to be accepted than a much worse one). T, called the temperature, decreases over time according to an annealing schedule. At higher values of T ''bad'' moves are more likely to be accepted. When $T = 0$, negative steps are always rejected and the algorithm behaves very much like gradient descent. In practice, the probability P is usually tested against a random number $R(0 \leq R \leq 1)$. A value of $R < P$ results in the new state being accepted. For example, if $P = 1/3$, then we would expect, on average, for every third worse new state to be accepted. The initial choice of T and the rate at which it is

```
Algorithm Simulated_Annealing(IN: initial_state; OUT: final_state)
Evaluate initial_state
If (initial_state = solution) then
  final_state ← initial_state
Else
  current_state ← initial_state
  Initialise T according to annealing schedule
  Do
    Select operator that has not yet been applied to current_state
    Apply operator to produce a new_state
    Evaluate new_state
    Compute ΔE
    If (new_state is better than current_state) then
      current_state ← new_state
    Else
      P ← e^(-ΔE/T)
      Generate random number R between O and 1
      If (R < P) then
        current_state ← new_state
      Endif
    Endif
    Revise T according to annealing schedule
  Until (current_state = solution) or (no new operators left to apply)
  final_state ← current_state
Endif
```

Figure 3. Simulated annealing algorithm.

decreased has an effect on how well the algorithm works. Generally, the slower the rate of change, the better the result. However, the processing overheads associated with the algorithm will increase as the rate of change in T becomes more gradual. In practice, a suitable annealing schedule is usually decided upon after some preliminary experimentation.

3. The simplicial data structure

Before describing the application of iterative improvement algorithms we will first summarize the spatial data structure and some relevant functions that are used to implement conflict detection procedures necessary to the evaluation functions used in our experiments. The methods presented in this paper represent part of an on-going development of the Map Authoring and Generalization Expert (MAGE) system [2], [3]. The generalization functions within MAGE make extensive use of a data structure based

on constrained Delaunay triangulation called a simplicial data structure (SDS) [2], [3], [9], [19]. We provide here a brief description of aspects of the SDS relevant to this paper.

The SDS is made up of four primary entities, namely, map features, triangles, edges and vertices. We deal here with a map M made up from a collection of three types of map feature. The feature types are:

- Planar polygonal objects O.
- Free space regions F, which lie between, or are contained within, polygonal objects.
- Linear objects L.

The SDS models M by means of a constrained Delaunay triangulation CDT [5] in which the edges of all polygonal objects O and all linear objects L act as constraints (figure 4). Each object in O and each free space region in F is defined in terms of references to those triangles of CDT lying within its boundary. Each triangle of CDT references its three constituent edges, three Boolean flags which indicate the direction (clockwise or anticlockwise) of each of its edges relative to itself, plus the polygonal object or free space region to which it belongs. Linear objects are defined by reference to their constituent edges. Each SDS edge is described by references to its start and end vertices, and adjacency information in the form of references to the triangles to which it belongs. It is this adjacency information that provides the basis for rapid movement through the SDS during search operations. In the case of an edge forming part of a linear object, that edge

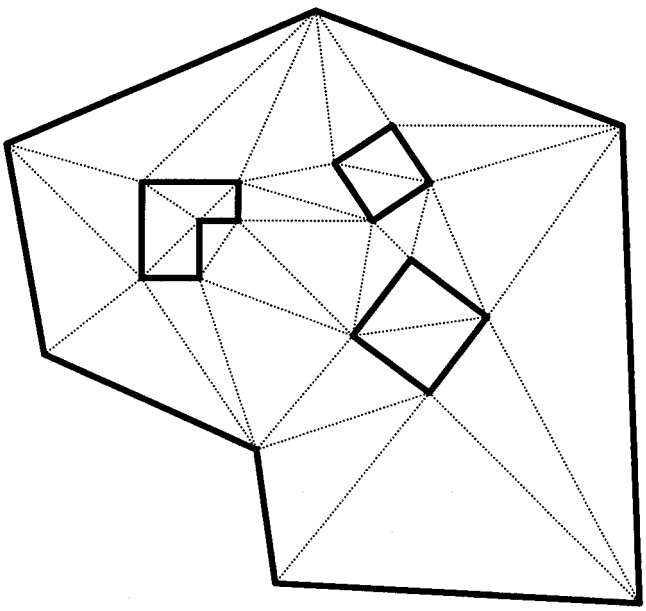

Figure 4. A constrained Delaunay triangulation of a simple map. Bold triangle edges correspond to map feature edges.

will also maintain a pointer to the object in L to which it belongs. Vertices simply consist of a reference to x, y co-ordinate values.

3.1. *Proximal conflict detection*

Taking a single polygonal object o_i in M, we are concerned with finding the set C_i consisting of all other polygonal objects and all linear objects in M which lie within a distance D_{tol} of o_i. We present here a SDS-based algorithm for finding C_i. Its efficiency is dependent on the explicit proximity properties inherent to constrained Delaunay triangulation. The algorithm, *Detect_Conflict*, is based on techniques presented in [10], [20], and is illustrated in figure 5.

```
Algorithm Detect_Conflict(IN: o_i, D_tol; OUT: C_i)

Initialize C_i to empty
object_edges ← GetObjectEdges(o_i)
For each object_edge in object_edges
  Initialize Queue to empty
  Add object_edge to Queue
  Do while (Queue not empty)
    queue_edge ← Dequeue(Queue)
    For each unprocessed adjacent_triangle of queue_edge
      object ← GetObject(adjacent_triangle)
      If (object ≠ o_i) then
        If (object ∈ O) then
          Add object to C_i
        Else
          object ← GetObject(queue_edge)
          If (object ∈ L) then
            Add object to C_i
          Endif
        Endif
        For each unprocessed triangle_edge of adjacent_triangle
          If (GetDistance(object_edge,triangle_edge) < D_tol) then
            Add triangle_edge to Queue
          Endif
        Endfor
      Endif
    Endfor
  Enddo
Endfor
```

Figure 5. Algorithm *Detect_Conflict*.

Detect_Conflict works by detecting the conflict associated with each edge of o_i in turn. For a particular *object_edge* this initially involves placing that edge on a previously empty *Queue*. Edges are subsequently removed from this queue and processed. For any such *queue_edge* this processing involves a series of actions upon each of its adjacent triangles. The first of these actions entails a test to see if the *adjacent_triangle* belongs to o_i. If it does then no further processing of *adjacent_triangle* is required. Alternatively, if *adjacent_triangle* does not belong to o_i, then more work is needed. First, we carry out a check to see if *adjacent_triangle of queue-edge* belongs to an *object* (other than o_i). If this is the case then *object* is added to C_i. Processing of *adjacent_triangle* concludes with an examination of each of its edges. If any such *triangle_edge* lies within a distance D_{tol} of *object_edge* then it is added to the *Queue*. It follows that the *Queue* will always only contain edges that are within a distance D_{tol} of the *object_edge* currently being processed. The algorithm makes use of the following functions:

- *GetObjectEdges* returns a list of edges making up the boundary of any polygonal object.
- *GetObject* returns the object to which a given triangle or edge belongs (if such an object exists).
- *GetDistance* returns the minimum distance between two edges.

4. Map object conflict reduction by iterative improvement

As indicated earlier, iterative improvement algorithms may have the potential for addressing a range of tasks that fall within the scope of process control in map generalization. The intention here is to explore their use within the confines of the sub-problem of resolving graphic conflict due to violation of constraints of proximity and topology. The assumption is that it should be possible to add further generalization operators and to develop evaluation functions that take account of a wider range of constraints.

We now describe two map object conflict reduction algorithms based on the iterative improvement algorithms described previously. The first uses a steepest gradient descent approach, similar to the discrete gradient descent algorithm presented in [6] for point-feature label placement (PFLP). The second adopts a simulated annealing approach, and, again, is similar to a PFLP algorithm presented in [6]. Before describing the algorithms individually, we discuss three elements common to both. These are the search space, the evaluation function and the change state function.

Search space. The search space consists of all possible realizations of the map, and is characterized by the set of trial positions associated with each map object. For a map consisting of n objects, each with k trial positions, the search space will consist of k^n elements. The trial positions associated with a particular object represent a discrete approximation to the continuous space into which it is permissible for that object to move.

The continuous space corresponds to a region that extends from the object by a distance equal to the maximum distance D_{disp} that the object can be displaced. In the case of a ''rigid'' areal object, such as a building, an object can be shifted into a trial position by application of a displacement vector to each of its vertices. It is possible to define a set of k displacement vectors for each object from which a series of k trial positions can be generated. The way in which displacement vectors are generated governs the distribution of trial positions about a given object. To ensure an even distribution of trial positions over the region we make use of a fixed template of displacement vectors, as illustrated in figure 6, with trial position 1 designated as being the object's initial location.

Evaluation function. The purpose of the evaluation function is to assign to any given element of the search space (i.e., any map realization) a value that represents the relative quality of that element. The measure of quality we adopt is based on minimizing the total number of conflicts within a particular element (i.e., the fewer the conflicts, the better the solution). Conflict is evaluated using the function *Evaluate_Conflict*. The viability of any iterative improvement algorithm depends heavily on it having an efficient evaluation function. In order to meet this requirement, *Evaluate_Conflict* makes extensive use of the SDS-based conflict detection procedure, *Detect_Conflict*, described previously. *Evaluate_Conflict* allows for multiple distance tolerances and differentiates between various types of conflict. This means that each class of object can be assigned a distance tolerance appropriate to that class, and ''seriousness'' weightings can be applied to conflicts involving objects of various class pairings. For example, the class of object representing buildings is assigned a minimum separating distance tolerance value of 10 units, whereas the class representing region boundaries is given a value of 5 units.

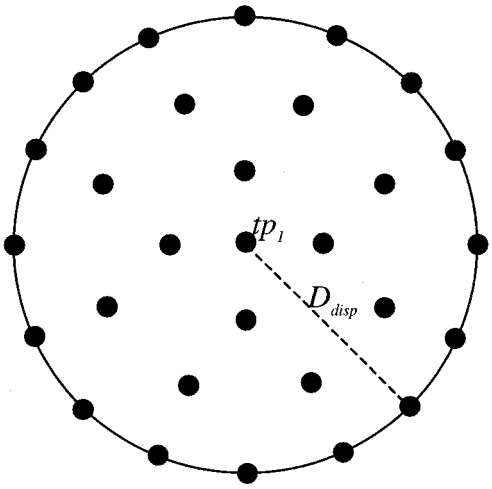

Figure 6. Displacement vector template for generating trial positions. tp_1 = trial position 1. There are 29 trial positions in total.

Furthermore, a conflict between two building objects is deemed less serious than conflict between a building object and a region boundary object (since the region boundary is thought to be a more important map feature than any object positioned within that region).

Change state function. Movement from one state to another (i.e., from one map realization to another) is made possible by the function *Change_State*, which moves a single object from one of its trial positions to another.

4.1. Conflict reduction using steepest gradient descent

Our steepest gradient descent algorithm (*SGD*) is presented in figure 7. The initial state consists of a map with each of its objects in its original position (i.e., trial position 1). The conflict evaluation function is then applied. In the event of there being no conflict the algorithm terminates at this point. Alternatively, if there is conflict, the algorithm enters its

```
Algorithm SGD(IN: initial_state; OUT: final_state)
a ← Evaluate_Conflict(initial_state)
If (a = 0) then
  final_state ← initial_state
Else
  current_state ← initial_state
  Do
    improvement ← False
    For each object in current_state
      For each trial_position of object
        new_state ← Change_State(current_state, object, trial_position)
        b ← Evaluate_Conflict(new_state)
        If (b < a) then
          move_object ← object
          move_position ← trial_position
          a ← b
          improvement ← TRUE
        Endif
      Endfor
    Endfor
    If (improvement) then
      current_state ← Change_State(current_state, move_object, move_position)
    Endif
  While (improvement)
  final_state ← current_state
Endif
```

Figure 7. Algorithm *SGD*, a steepest gradient descent algorithm for conflict reduction.

main loop. Each time through the loop an attempt is made to find the single change in state that results in the most immediate improvement (it is here where steepest gradient descent differs from simple gradient descent in that the latter accepts the first change in state that offers any improvement). If found, this change in state is applied. In other words, each time through the loop we implement the single object repositioning that results in most improvement. The loop is repeated until no further improvement can be made (i.e., the current state is a global or local minimum). Note that in their implemented form, *SGD* and *SA* (see next section) maintain up to date descriptions of the conflict associated with each object. This means that apart from its initial call, when all conflict must be detected, *Evaluate_Conflict* only has to recompute the conflict associated with the object undergoing displacement (although the conflict descriptions of objects with which this object was or is in conflict might also need updating).

4.2. Conflict reduction using simulated annealing

The simulated annealing algorithm for conflict reduction, *SA*, is shown in figure 8. As with *SGD* this algorithm begins by evaluating the initial conflict. Again, if no conflict exists the algorithm terminates, otherwise the algorithm enters its main loop. A significant difference between *SA* and *SGD* is that here, only one object move is tested per loop cycle. The object concerned, and the trial position to which it is moved, are picked at random. If the new position results in a new state that is better than the current state (i.e., $\Delta E < 0$) then the change is accepted. If the new state represents a worse or equivalent solution, then it is still accepted with probability P (i.e., if the random number $R < P$). As indicated earlier, the key factor determining the success or failure of *SA* is the choice of annealing schedule. Our schedule follows the format used in [6]. This involves setting T to an initial value V. At each temperature a maximum of Wn object repositionings (successful or unsuccessful) are allowed, where n is the number of map objects. After every Wn repositionings T is decreased by $X\%$. Also, if more than Yn successful repositionings are made at any one temperature then T is immediately decreased. If no successful repositionings are made at a particular temperature then the algorithm terminates. Finally, a limit on the maximum number of temperature stages allowed is set to Z.

4.3. Detecting topological error

A problem with adopting the trial position approach as described so far is that we run the risk of introducing topological error into the map. This is particularly so when dealing with situations involving object classes of differing importance, and also in situations where negative moves are allowed. In such circumstances it is possible to force map objects to overlap each other (and, in extreme circumstances, to force an object to move from one side of a neighboring object to another).

It has been shown previously that monitoring the relative edge direction of SDS triangles can assist detection of topological error of this kind [9]. It is always the case that,

```
Algorithm SA(IN: initial_state; OUT: final_state)
a ← Evaluate_Conflict(initial_state)
If (a = 0)
  final_state ← initial_state
Else
  current_state ← initial_state
  T ← V
  number_of_temps ← 0
  positions_tested ← 0
  successfully_tested ← 0
  continue ← True
  Do
    Randomly choose object
    Randomly choose trial_position
    new_state ← Change_State(current_state, object, trial_position)
    b ← Evaluate_Conflict(new_state)
    ΔE ← b - a
    Add 1 to positions_tested
    If (ΔE < 0) then
      current_state ← new_state
      a ← b
      Add 1 to successfully_tested
    Else
      P ← e^(-ΔE/T)
      Generate random number R between O and 1
      If (R < P) then
        change_state ← new_state
        a ← b
        Add 1 to successfully_tested
      Endif
    Endif
    If (positions_tested > Wn) or (successfully_tested > Yn) then
      If (number_of_temps > Z) or (successfully_tested = 0) then
        continue ← False
      Else
        Decrease T by X%
        positions_tested ← 0
        successfully_tested ← 0
        Add 1 to number_of_temps
      Endif
    Endif
  While (a ≠ 0) and (continue)
  final_state ← current_state
Endif
```

Figure 8. Algorithm *SA*, a simulated annealing algorithm for conflict reduction.

should topological error have occurred, one or more of the triangles immediately surrounding the objects concerned will have become inverted (i.e., relative edge directions will have changed). Note, however, that the presence of inverted triangles does not necessarily indicate the presence of topological error. One approach to resolving the problem of topological error would be to perform a check for inverted triangles subsequent to each object displacement. Then, if any inversions were found, geometric processing would be carried out to prove or disprove the presence of topological error. If it were present, then the object displacement would be reversed. However, in practice, it is noted that inverted triangles that are not the result of topological error usually signal neighboring object pairs that have become significantly misaligned (figure 9). Because of this, *Evaluate_Conflict* has been modified to reject all object displacements that result in inverted triangles. This has the positive effects of ensuring topological consistency and helping to maintain alignment between objects, but can sometimes result in acceptable object displacements being rejected.

5. Experimental results

The two iterative improvement algorithms described have been implemented using the C programming language. The test data sets used consist of two types of feature, namely, movable polygonal objects and fixed, linear region boundaries. It should be remarked that

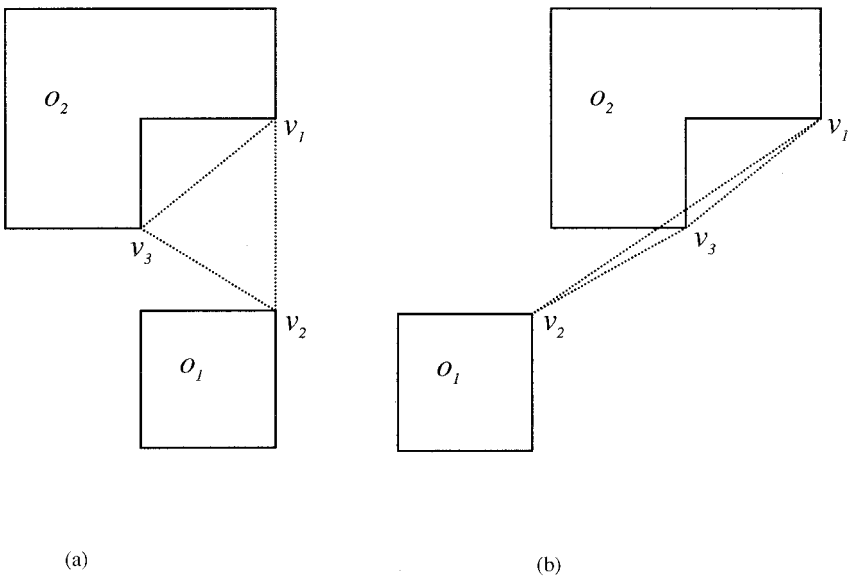

Figure 9. Object misalignment caused by object displacement. (a) Part of a map region, prior to displacement, showing two objects o_1 and o_2. For the sake of clarity, only a single SDS triangle, defined by vertices (v_1, v_2, v_3), is shown. (b) Subsequent to displacement of o_1, the triangle becomes inverted.

immovable boundaries are employed here to demonstrate some flexibility with regard to types of map object and their associated differing tolerances. They are not required as part of the procedure. All experiments were carried out on a Sun Enterprise 2 model 2200 (2×200 MHz Ultrasparc processors).

5.1. *Initial experiments*

Initial experiments involved the use of 5 hand-generated test data sets (figure 10). These experiments were carried out mainly to get a feel for how well the algorithms performed, and in the case of *SA*, to discover what sort of annealing schedules worked best. Each data set consists of a small number of polygonal objects contained within a single region boundary. For all experiments, conflict between a polygonal object and the region boundary was deemed more serious than conflict involving polygonal objects only. Minimum separating distance tolerances values were chosen so as to provide reasonable numbers of initial polygon/boundary and polygon/polygon conflict, while maximum polygonal object displacement values were set so as to give the algorithms a reasonable chance of success.

In the case of the *SA* experimental results shown, the annealing schedule was as follows:

- The initial temperature value was set to 3.0 (i.e., $V = 3.0$).
- At each change in temperature, T was decreased by 10% (i.e., $X = 10$).
- A decrease in temperature occurred after every $100n$ repositionings or every $30n$ successful repositionings (i.e., $W = 100$, $Y = 30$).
- The maximum number of temperature stages was set to 50 (i.e., $Z = 50$).

Each algorithm was executed 100 times for each data set. The results illustrated in figures 11 and 12, and described in tables 1, 2 and 3, show that application of either algorithm reduces the amount of conflict. Note that for *SA*, the conflict reduction achieved can differ for a particular data set from one program execution to the next. The reason for this is that

Table 1. Tolerance values and initial conflict values. Tolerance values for BDTopo data are given in metres. P/P—polygon/polygon conflict, P/B—polygon/boundary conflict.

Data set	*P/P tol*	*P/B tol*	D_{disp}	*Initial P/P Conflict*	*Initial P/B Conflict*
1	2.0	1.5	2.0	10	4
2	3.0	3.0	3.0	6	4
3	2.0	1.5	2.0	12	5
4	3.0	3.0	3.0	12	7
5	2.0	1.5	3.0	26	1
BDTopo	7.5	7.5	7.5	236	36

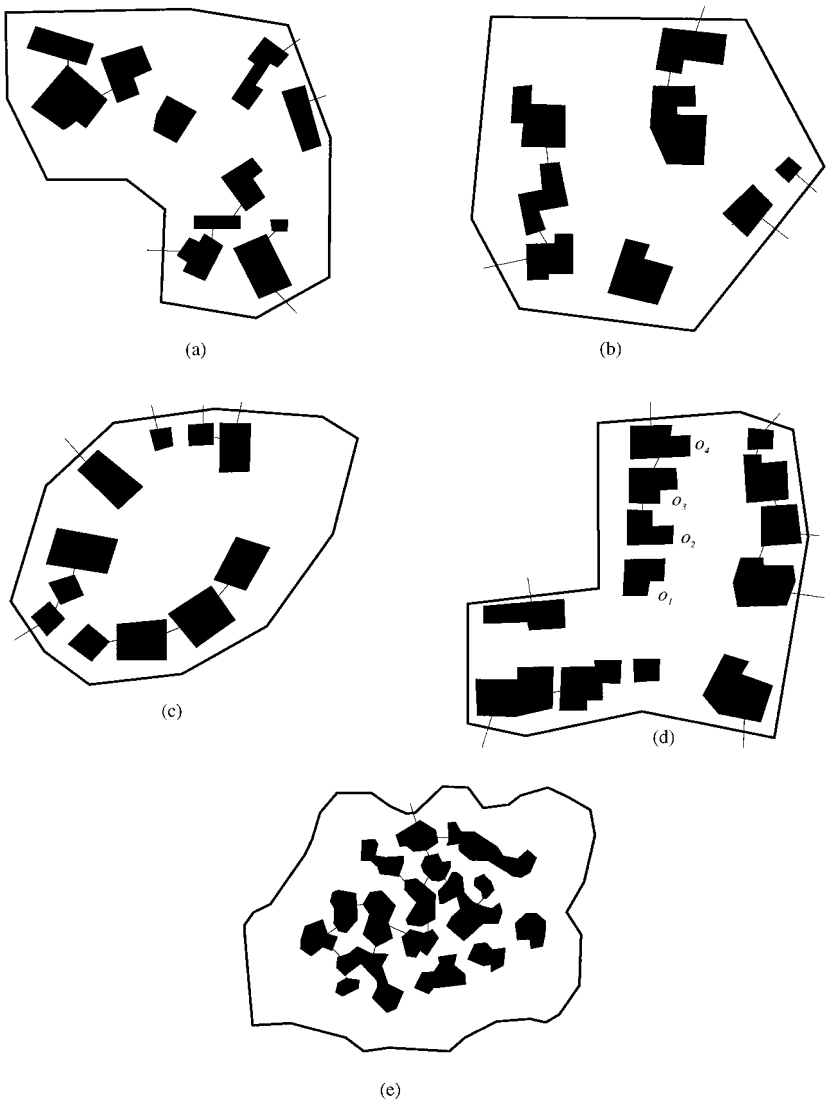

Figure 10. The five hand-generated test data sets. An edge between a pair of map features indicates that those features are in conflict with each other. (a) Data set 1. (b) Data set 2. (c) Data set 3. (d) Data set 4. (e) Data set 5.

simulated annealing involves a degree of randomness; we cannot expect the same result each time the algorithm is run. The *SA* results given are representative of the complete set of results and are consistently better than SGD with regard to conflict reduction.

The superior performance of *SA* is due to its ability to escape from local minima, as predicted. This can be explained to some extent by reference to figures 10d, 11d and 12d, and in particular, to object o_1. As can be seen, initially o_1 is not in conflict with any other

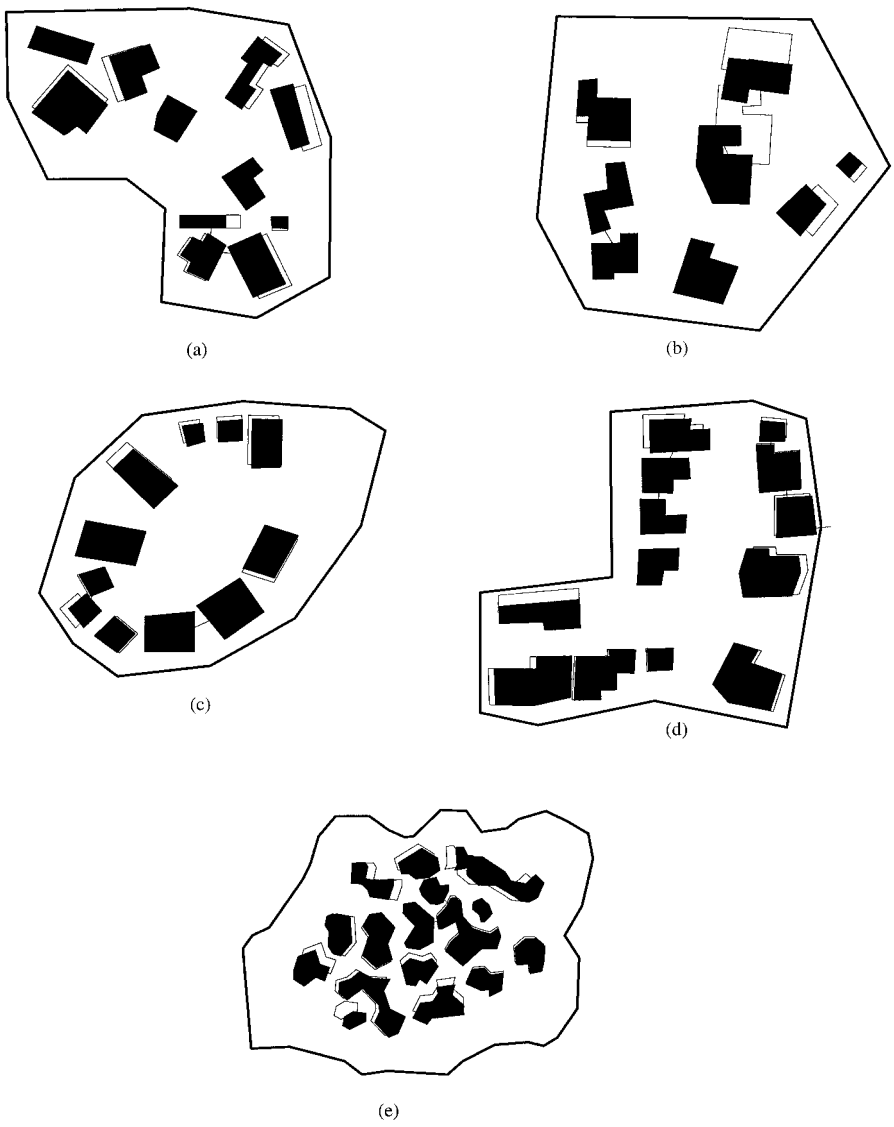

Figure 11. Data sets 1–5 after conflict reduction using *SGD*.

map feature. This means that throughout execution of *SGD*, o_1 might (and in this case does) remain in its original position (since negative and neutral moves are disallowed). However, the three objects above o_1 (i.e., o_2, o_3 and o_4) are in conflict. Unfortunately, because of the presence of o_1, they have no space to move into. In other words, o_1 is acting as a block to conflict resolution. Unlike *SGD*, *SA* sometimes permits negative and neutral

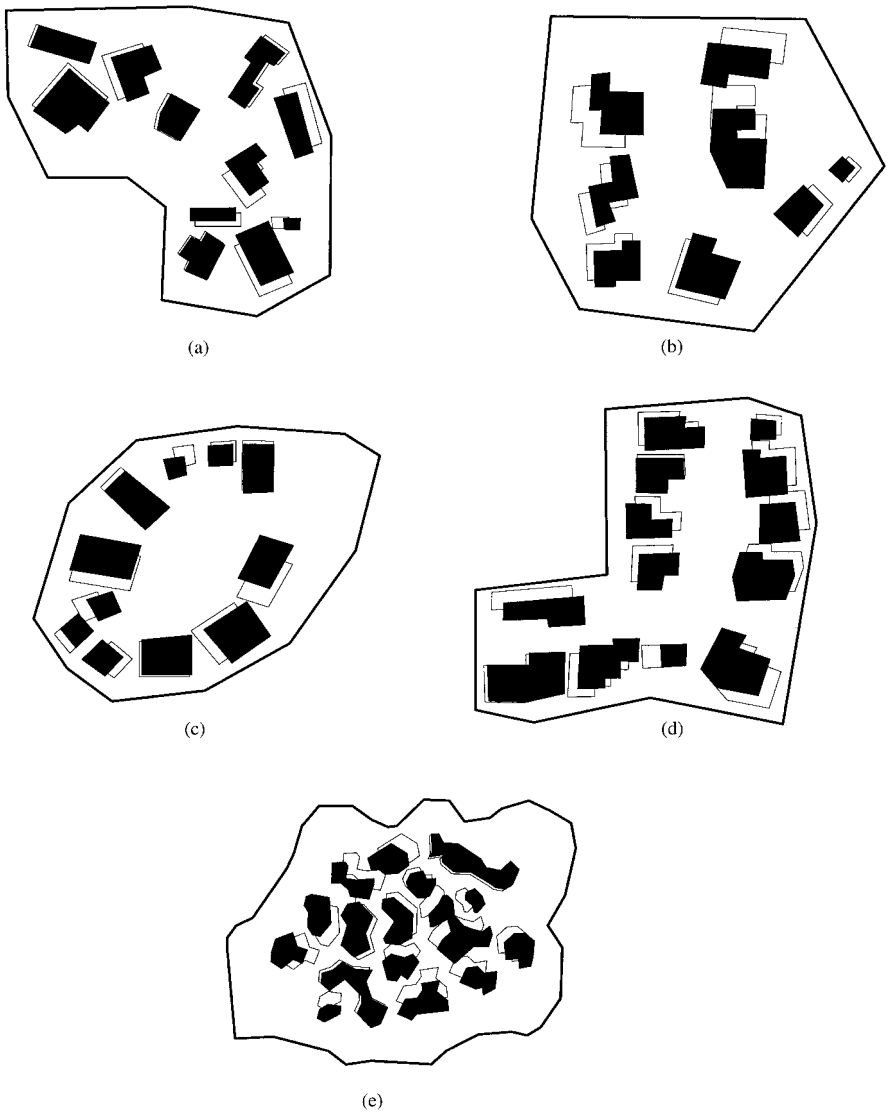

Figure 12. Data sets 1–5 after conflict reduction using *SA*.

moves and, as can be seen, o_1 undergoes such a move. This has the effect of freeing up space for o_2, o_3 and o_4 to move into. In this case, the success of *SA* is dependent on:

- The object o_1 being chosen at random from the set of all objects.
- A particular trial position being chosen at random from the set of all trial positions.
- The random number R being less than the probability P.

Table 2. *SGD* results. All times are given in seconds. s.d.—standard deviation, P/P—polygon/polygon conflict, P/B—polygon/boundary conflict.

Data Set	*Final P/P Conflict*	*Final P/B Conflict*	*Realizations Tested*	*Time Taken (s)*	
				Average	*s.d.*
1	4	0	3509	0.191	(0.001)
2	2	0	2088	0.154	(0.001)
3	4	0	3828	0.142	(0.001)
4	10	1	3016	0.144	(0.001)
5	6	0	4640	1.228	(0.004)
BDTopo	81	1	1045856	103.649	(0.390)

It is clear that the chances of all three conditions being met will increase or decrease according to the choice of annealing schedule.

5.2. *Experiments with topographic data*

Further experiments made use of IGN-France BDTopo data (1:25,000) (figure 13a), consisting of 321 polygonal objects contained within 16 free space regions. The minimum separating distance tolerances used assume a visual perception threshold of 0.15 mm and a map scale reduction to 1:50,000. As before, each test was repeated 100 times. It can be seen from the results in tables 1, 2 and 3 that, while both algorithms significantly reduce conflict, *SA* is a clear winner, both in terms of conflict reduction and execution time. The *SGD* result is shown in figure 13b and a single *SA* result is shown in figure 13c. The better conflict reduction performance of *SA* is again attributed to its ability to escape local minima. This can be illustrated by reference to Region 1 in figure 13. Prior to conflict reduction, 11 of the 23 polygonal objects contained within the region are involved in conflict of some sort. Application of *SGD* results in only 3 of these 11 objects becoming completely free of conflict, whereas application of *SA* brings about the removal of all

Table 3. *SA* results. All times are given in seconds. s.d.—standard deviation, P/P—polygon/polygon conflict, P/B—polygon/boundary conflict.

	Final P/P Conflict		*Final P/B Conflict*		*Realizations Tested*		*Time Taken (s)*	
Data Set	*Average*	*s.d.*	*Average*	*s.d.*	*Average*	*s.d.*	*Average*	*s.d.*
1	0.0	(0.0)	0.0	(0.0)	13999.8	(175.5)	0.833	(0.009)
2	0.0	(0.0)	0.0	(0.0)	10672.6	(36.3)	0.723	(0.012)
3	0.0	(0.0)	0.0	(0.0)	6323.8	(3389.0)	0.273	(0.139)
4	0.0	(0.0)	0.0	(0.0)	8234.2	(1450.5)	0.391	(0.067)
5	0.4	(0.8)	0.0	(0.0)	17706.2	(1869.2)	4.491	(0.522)
BDTopo	26.6	(2.9)	0.0	(0.0)	342302.2	(18613.1)	39.665	(2.173)

Figure 13. Roads and buildings extracted from IGN-France BDTopo data (1:25,000). Buildings involved in conflict of some sort are shown in gray. (a) Original data. (b) After conflict reduction using *SGD*. (c) After conflict reduction using *SA*.

conflict. The poor performance of *SGD* is a result of its inability to move conflict-free objects (of which there are 12 originally), which means that less opportunities exist to free up space for objects that are in conflict to move into.

Both algorithms succeed in limiting the number of map realizations needed to be generated and evaluated. For example, in the case of the BDTopo data, out of a total of 29^{321} possible realizations, *SGD* generated and evaluated 1,045,856 and *SA* generated and evaluated, on average, 342,302. The corresponding average execution times are 103.649 s and 39.665 s. It is anticipated that some improvement in execution times may be achieved by filtering candidate trial positions to omit those that will always result in conflict due, for example, to the presence of an immovable symbol. The use of trial position filtering may also be relevant to the generation of solutions that retain original patterns of map objects.

6. Concluding remarks

This paper has sought to apply the concept of iterative improvement to the problem of automated map generalization. It has provided two new algorithms, *SGD* and *SA*, for

Figure 13. (*Continued*)

carrying out map conflict reduction via displacement operations. Experimental results have shown both approaches to be successful in reducing conflict. In short, we have shown iterative improvement techniques to be of potential benefit in helping us reach the goal of a fully automated generalization system. Of the two algorithms, *SA* (simulated annealing) has proven to be the most successful. This is explained by its ability to escape local minima. The results lead us to believe that continued development of the *SA* approach is a promising way forward in the search for improved solutions.

In order to automate more fully the generalization task, there is the need to integrate other generalization functions within conflict resolution procedures. As indicated earlier in the paper, the success of such integration will require the development of more advanced evaluation functions that take account of a wider range of constraints, including those of form and structure.

Acknowledgments

The authors express thanks to the Institut Géographique National for permission to use their data in parts of the work presented here, and also to Christopher D. Eynon for his help in the preparation of this paper. JMW was supported by the NERC grant GR3/10569.

Figure 13. (*Continued*)

References

1. K.E. Brassel and R. Weibel. ''A review and conceptual framework of automated map generalization,'' *International Journal of Geographical Information Systems*, Vol. 2(3):229–244, 1988.
2. G.Ll. Bundy, C.B. Jones, and E. Furse. ''Holistic generalisation of large scale cartographic data,'' *GIS and Generalization Methodology and Practice* (edited by J.C. Muller, J.-P. Lagrange and R. Weibel, Taylor and Francis):106–119, 1995.
3. G.Ll. Bundy, C.B. Jones, and E. Furse. ''A topological structure for the generalisation of large scale cartographic data,'' *Innovations in GIS 2* (edited by P. Fisher, Taylor and Francis):19–31, 1995.
4. B.P. Buttenfield and R.B. McMaster (editors). Map Generalization: Making Rules for Knowledge Representation (Longman), 1991.
5. P.L. Chew. ''Constrained Delaunay triangulations,'' *Algorithmica*, Vol. 4:97–108, 1989.
6. J. Christensen, J. Marks, and S. Shieber. ''An empirical study of algorithms for point-feature label placement,'' *ACM Transactions on Graphics*, Vol. 14(3):203–232, 1995.
7. C.B. Jones. ''Cartographic name placement with Prolog,'' *IEEE Computer Graphics and Applications*, Vol. 9(5):36–47, 1989.
8. C.B. Jones. ''Conflict resolution in cartographic name placement,'' *Computer Aided Design*, Vol. 22(3):173–183, 1990.
9. C.B. Jones, G.Ll. Bundy, and J.M. Ware. ''Map generalisation with a triangulated data structure,'' *Cartography and GIS, Special Issue: Automated Map Generalization*, Vol. 22(4):317–331, 1995.
10. C.B. Jones and J.M. Ware. ''Proximity relations with triangulated spatial models,'' *The Computer Journal*, Vol. 41(2):71–83, 1998.

11. S. Kirkpatrick, C.D. Gelath, and M.P. Vecchi. ''Optimization by simulated annealing,'' *Science*, Vol. 220:671–680, 1983.
12. J.-C. Muller, J.-P. Lagrange, and R. Weibel (editors). GIS and Generalization Methodology and Practice (Taylor and Francis), 1995.
13. S. Openshaw and C. Openshaw. Artificial intelligence in geography (John Wiley), 1997.
14. E. Rich and K. Knight. Artificial Intelligence (2nd Edition, McGraw-Hill), 1991.
15. A.H. Robinson, J.L. Morrison, A.J. Muehrcke, S.C. Guptill, and A.J. Kimerling. Elements of Cartography (John Wiley), 1995.
16. A. Ruas and C. Plazanet. ''Strategies for automated generalization,'' *Proceedings of 7th International Symposium on Spatial Data Handling*, Vol. 1:6.1–6.18, 1996.
17. S. Russell and P. Norvig. Artificial Intelligence: A Modern Approach (Prentice-Hall), 1995.
18. K.S. Shea and R.B. McMaster. ''Cartographic generalization in a digital environment: When and how to generalize,'' *Proceedings of 9th International Symposium on Computer-Assisted Cartography (Auto-Carto 9)*, 56–67, 1989.
19. J.M. Ware, C.B. Jones, and G.Ll. Bundy. ''A triangulated spatial model for cartographic generalisation of areal objects,'' *Springer-Verlag Lecture Notes in Computer Science*, *988*:173–193, 1995.
20. J.M. Ware and C.B. Jones. ''A spatial model for detecting (and resolving) conflict caused by scale reduction,'' *Advances in GIS Research 2* (edited by M.J. Kraak and M. Molenaar, Taylor and Francis), 547–558, 1996.
21. R. Weibel (guest editor). ''Special Contents Issue: Automated Map Generalization,'' Cartography and Geographical Information Systems, 22(4), 1995.
22. P.H. Winston. Artificial Intelligence (Addison-Wesley), 1992.
23. S. Zoraster. ''Practical results using simulated annealing for point feature label placement,'' *Cartography and Geographical Information Systems*, Vol. 24(4):228–238, 1997.

Mark Ware is a Research Fellow in the School of Computing at the University of Glamorgan. He received a B.Sc. (1989) in mathematics and computing from the Polytechnic of Wales, and a Ph.D. (1994) from the University of Glamorgan for research concerning multiscale data storage schemes for spatial information systems. His current research interests include automated cartography (map generalization), multiscale databases, spatial data uncertainty, environmental change detection and digital terrain modeling.

Chris Jones is Professor of Geographical Information Systems at the University of Glamorgan. He has worked previously at the British Geological Survey, BP Exploration and the University of Cambridge. He graduated in geology from Bristol University, and received a Ph.D. from the University of Newcastle upon Tyne, for research on periodicities in fossil growth rhythms. Current research interests include the use of geographical information in hypermedia for public information access; multiscale spatial databases; data integration; environmental change detection; computer reconstruction and visualization of fossils; and automated cartographic design with regard to map generalization and automated text placement. He is author of the text Geographical Information Systems and Computer Cartography, published by Longman. He directed cartographic research and development resulting in the Maplex product for automated text placement.