A Simulated Annealing Algorithm For Cartographic Map Generalization With Multiple Operators

J. Mark Ware[1], Christopher B. Jones[2] & Nathan Thomas[1]
[1]School of Computing, University of Glamorgan, Pontypridd CF37 1DL, UK
[2]Department of Computer Science, Cardiff University, Cardiff CF24 3XF, UK
e-mail: jmware@glam.ac.uk

1 Introduction

Displaying map data at scales smaller that its source scale can result in spatial conflict in which objects become either too small to be seen or too close to each other to be distinguishable. Map generalization, the process by which such conflict is removed, involves the application of a range of generalization operators in combination. For example, objects too small to be seen clearly can be deleted or enlarged; large objects taking up too much map space can be reduced in size; and objects lying too close to each other can be displaced or amalgamated.

This paper, building upon work presented previously ([1], [2]), describes an algorithm that makes use of the displacement, deletion, reduction and enlargement of multiple map objects in order to resolve graphic conflict (Figure 1). It adopts a trial position approach (similar to that used previously in point feature label placement e.g. [3]), in which each of n discrete polygonal objects is assigned k candidate trial positions into which it can possibly move; these positions represent the original, displaced, deleted, reduced and enlarged states of the object. This results in a possible k^n distinct map configurations; the assumption is that some of these configurations will contain less conflict than the original. Finding the best (or, at least, an acceptable) configuration by means of an exhaustive search is, however, not practical for realistic values of n and k. Instead the algorithm makes use of the simulated annealing search technique [4], which has been shown to be successful in solving large optimization problems quickly e.g. [5]. For our purposes, simulated annealing reduces the number of map configurations needing to be generated and evaluated.

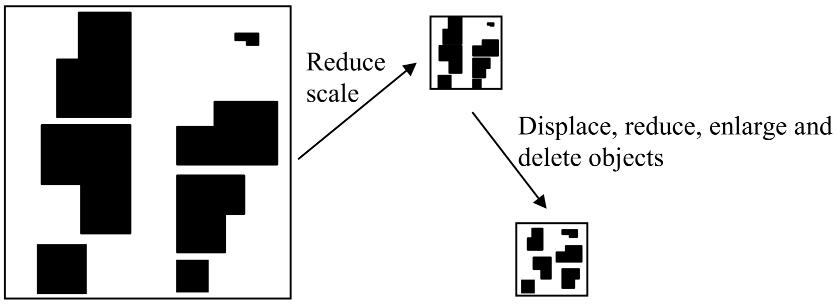


Figure 1. Reducing scale causes conflict, which can be resolved by a combination of object displacement, reduction, enlargement and deletion.

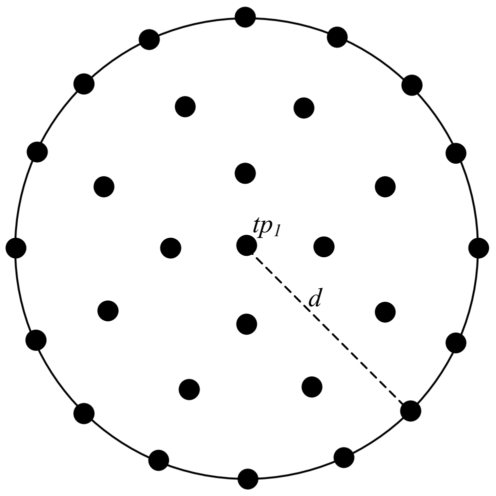


Figure 2. Displacement vector template for generating trial positions. tp_1 = trial position 1. In this example, there are 28 displaced state trial positions.

2 Conflict Reduction by Object Displacement

In previous work the authors consider a map display D made up of fixed linear objects F and n modifiable detached polygonal objects M. Each modifiable object $m_i \in M$ has k possible states, providing a total of k^n possible configurations of D.

Object States

At any given time a particular object m_i exists in one of its k states (we will also refer to these states as trial positions). An object's initial map position is designated as being trial position 1. Additional trial positions arise as a result of object displacement only. (Additional generalization operators are considered in section 5). It is assumed that during the course of generalization, each object m_i can be displaced up to some maximum distance d from its original position (i.e. there is a continuous space that extends from m_i by a distance d into which it is permissible for m_i to move). The displacement trial positions associated with m_i represent a discrete approximation to this continuous space. Each object m_i has k displacement trial positions, which are distributed evenly about the object (Figure 2).

Evaluation of Map Display

For a particular configuration D_j, each object m_i has an associated cost. The cost is determined by the extent to which m_i is in conflict. Two categories of spatial conflict are considered:

- Type-1. Conflict between a pair of polygonal objects (i.e. m_i and m_j lie too close to each other to be distinguishable). This conflict occurs when the minimum separating distance (in viewing coordinates) between m_i and m_j is less than some predefined threshold d_{min1}. An occurrence of this type of conflict carries a cost c_1;
- Type-2. Conflict between a polygonal object and a linear object (i.e. m_i and $f_j \in F$ lie too close to each other to be distinguishable). This conflict occurs when the minimum separating distance (in viewing coordinates) between m_i and f_j is less than some predefined threshold d_{min2}. An occurrence of this type of conflict carries a cost c_2.

The total cost $\mathbf{C}(D_j)$ associated with a realization D_j is found by summing the costs associated with each object $m_i \in M$. Our goal is to find a minimum cost configuration D_{min} such that

$$\mathbf{C}(D_{min}) = \mathbf{MIN}(\mathbf{C}(D_j), j=1,2\ldots k^n).$$

The set of all configurations is referred to as the search space. If the search space is small enough then D_{min} can be found by generating and evaluating each configuration D_j ($j=1,2\ldots k^n$) in turn. However, this is not practical for realistic values of n and k. For example, a relatively simple display consisting of 10 modifiable object, each with 8 trial positions, gives rise to approximately 1,000,000,000 configurations. Results reported later in this paper suggest it would take approximately 10 days to process this many configurations (generating and evaluating a single configuration takes approximately 0.00005s).

A Simulated Annealing Solution

A well-established approach to solving large optimization problems of the kind described is to adopt an iterative improvement algorithm. The concept of iterative improvement can be illustrated by considering the search space (i.e. in our case, all map configurations) to be laid out on the surface of a landscape. The elevation at any point on the landscape represents the cost for the particular configuration associated with that point. An iterative improvement algorithm will move around the landscape in an attempt to find the lowest troughs, which correspond to low cost configurations [7]. Two major classes of iterative improvement algorithms are Gradient Descent and Simulated Annealing.

```
function GradientDescent
        input: $D_{initial}$
        $D_{current} \leftarrow D_{initial}$
        do
            $D_{new} \leftarrow$ LowestCostSuccessor($D_{current}$)
            if C($D_{new}$) $\geq$ C($D_{current}$) then Return($D_{current}$)
            $D_{current} \leftarrow D_{new}$
        end
        Return($D_{current}$)
```

Algorithm 1. Gradient Descent.

```
function SimulatedAnnealing
        input: $D_{initial}$, Schedule, Stop_Conditions
        $D_{current} \leftarrow D_{initial}$
        T $\leftarrow$ initialT(Schedule)
        while NotMet(StopConditions)
            $D_{new} \leftarrow$ RandomSuccessor($D_{current}$)
            $\Delta E \leftarrow$ C($D_{current}$)-C($D_{new}$)
            if $\Delta E > 0$ then $D_{current} \leftarrow D_{new}$
            else
                $P = e^{-\Delta E / T}$
                R=Random(0,1)
                if (R<P) then $D_{current} \leftarrow D_{new}$
            end
            T $\leftarrow$ UpdateT(Schedule)
        end
        Return($D_{current}$)
```

Algorithm 2. Simulated Annealing.

Algorithm 1 describes a simple gradient descent implementation. The algorithm accepts an initial map configuration $D_{initial}$ (i.e. each object in trial position 1), which is immediately designated as being the current solution $D_{current}$. Next the lowest cost successor D_{new} to $D_{current}$ is found. A particular successor to $D_{current}$ is found by moving a single object m_i to an alternative trial position; the lowest cost successor can be found by generating and evaluating all possible successors (of which there are kn-1). If D_{new} represents an improvement on $D_{current}$, then D_{new} becomes $D_{current}$ and the next lowest cost successor is generated. This process is repeated until a D_{new} is generated that offers no improvement; at this stage the algorithm terminates, with $D_{current}$ being returned as the solution. 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 any single displacement generates 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 available (e.g. random-restart, backtracking and multiple-moves). However, the exponential nature of most realistic search spaces make such remedies impractical.

Searches based on simulated annealing (Algorithm 2) attempt to overcome the problem of getting caught in local minima. They achieve this by sometimes allowing non-improving configurations to be accepted. As with gradient descent, simulated annealing always accepts D_{new} if it offers a better solution than $D_{current}$. However, in cases where D_{new} provides no improvement, simulated annealing will accept the new configuration with some probability P (P<1). Like gradient descent, the algorithm begins by accepting an initial map configuration $D_{initial}$ (i.e. each object in trial position 1); this is immediately designated as being the current solution $D_{current}$. Next a random successor D_{new} is generated by moving a randomly chosen object m_i to a randomly chosen trial positions k_j. If the displacement results in a display configuration with a lower cost ($\mathbf{C}(D_{new}) < \mathbf{C}(D_{current})$), then the object remains in the chosen trial position ($D_{current} \leftarrow D_{new}$). If, however, the new display has a higher or equal cost (i.e. $\mathbf{C}(D_{new}) \geq \mathbf{C}(D_{current})$), then the object is either returned to its previous position or remains in its new position, depending on probability P. The process of attempting a random object displacement continues until stop conditions are met (e.g. a solution which meets a target cost is found or a pre-defined maximum number of iterations have taken place or a pre-defined maximum amount of time has elapsed).

At each iteration the probability P is dependant on two variables, ΔE (the change in conflict, measured by the difference in cost between the new and current states) and T (the current temperature), and is defined as:

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

T is assigned a relatively high initial value; its value is decreased in stages throughout the running of the algorithm. At high temperatures poor displacements (large negative ΔE) will often be accepted. At low temperatures poor displacements will tend to be rejected (although displacements resulting in small negative ΔE might still sometimes be accepted). The acceptance of some poor displacements is permitted so as to allow escape from locally optimal solutions. 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 value of T and the rate by which it decreases is governed by what is called the annealing schedule. Generally, the higher the initial temperature and the slower the rate of change, the better the result (in cost reduction terms); however, the processing overheads associated with the algorithm will increase as the rate of change in T becomes more gradual.

Finding a minimum cost configuration D_{min} by simulated annealing is statistically guaranteed, provided that the temperature reductions are small enough, and that for each temperature the number of configurations tested is large enough [3]. However, most practical applications settle for near optimal solutions, and make corresponding compromises in the annealing schedule; a suitable schedule is usually decided upon after some preliminary experimentation.

Cost Function
The viability of any iterative improvement algorithm depends heavily on it having an efficient cost function, the purpose of which is to determine for any given element of the search space (i.e. any map realization) a value that represents the relative quality of that element. The cost function used here, $\boldsymbol{C}$, is called repeatedly and works by calculating and summing the costs associated with objects $m_i \in M$. When invoked initially, $\boldsymbol{C}$ must calculate the cost associated with every object $m_i \in M$. A record of these costs is maintained for future reference, meaning that, in any further call, $\boldsymbol{C}$ has to consider only objects with costs affected by the most recent displacement. Calculating the cost associated with a polygonal object m_i involves identifying all other polygonal objects lying within a distance d_{min1} and all linear objects lying within a distance d_{min2}. Identifying these conflicting objects quickly requires the use of a spatial index of some kind. The current implementation makes use a triangle-based data structure called the SDS, together with an associated search procedure (extensive details of both are given in [1],[8] and [9]).

Initial Results
Initial experiments, reported in [1], demonstrate that both gradient descent and simulated annealing approaches are successful in reducing conflict while limiting the number of realizations examined. When compared against each other, the simulated annealing approach is clearly superior with regard to the degree of conflict reduction achieved. It is for this reason that the simulated annealing algorithm was chosen as the focus for further research, as discussed in the remainder of this paper.

The experiments reported in [1] make use of IGN-France BDTopo data (1:25,000) consisting of 321 polygonal objects contained within 16 regions. The minimum separating distance tolerances used assume a visual perception threshold of 0.15mm and a map scale reduction to 1:50,000. The tolerance values (d_{min1} and d_{min2}), displacement value (d), initial conflict values and cost values used are shown in Table 1. In the experiments, Type-1 conflict is deemed less serious than Type-2 conflict; the cost values c_1 and c_2 are set so as to reflect this fact. The experiments made use of a C implementation of the simulated annealing algorithm (compiled with –O3 option) running under UNIX on a Sun Enterprise 2 model 2200 (2x200MHz Ultrasparc). The annealing schedule parameters were set as follows (see [1] for details): Initial temperature V=3.0; Temperature reduction value X=0.1; Cooling rate values W=100 and Y=30; and Maximum number of temperature stages Z=50 (in practice the maximum number of temperature stages is never reached). As can be seen from the results given in Table 1 and illustrated in Figure 3, the simulated annealing approach reduces the amount of spatial conflict by up to 90%. It achieves this while at the same time limiting the number of realizations having to be generated and evaluated (approximately 300,000 out of a possible 29^{321}).

Tolerance values and initial conflict	d_{min1}	d_{min2}	d	c_1	c_2	**Type-1**	**Type-2**
	7.5	7.5	7.5	1	10	236	36

Results	**average Type-1**	**s.d. Type-1**	**average Type-2**	**s.d. Type-2**	**average number of tests**	**s.d. number of tests**	**average execution time**	**s.d. execution time**
Original	26.6	2.9	0.0	0.0	342302.2	18613.1	39.67	2.17
800MHz	25.4	2.7	0.0	0.0	328282.5	17439.3	13.46	0.67

Table 1. Initial and updated results (s.d. = standard deviation).

Updated Results

For the sake of valid comparison with results reported later in this paper, the same C implementation (again compiled with –O3 option) has now been run under LINUX on an 800MHz Pentium III machine (128Mb RAM). The new results are shown in Table 1. The only significant change is in execution time, which has been reduced from 39.67s to 13.46s; this reduction is a direct result of using a faster machine. The number of realizations tested and final conflict remain more or less the same, as is expected; the small differences are due to the random nature of simulated annealing.

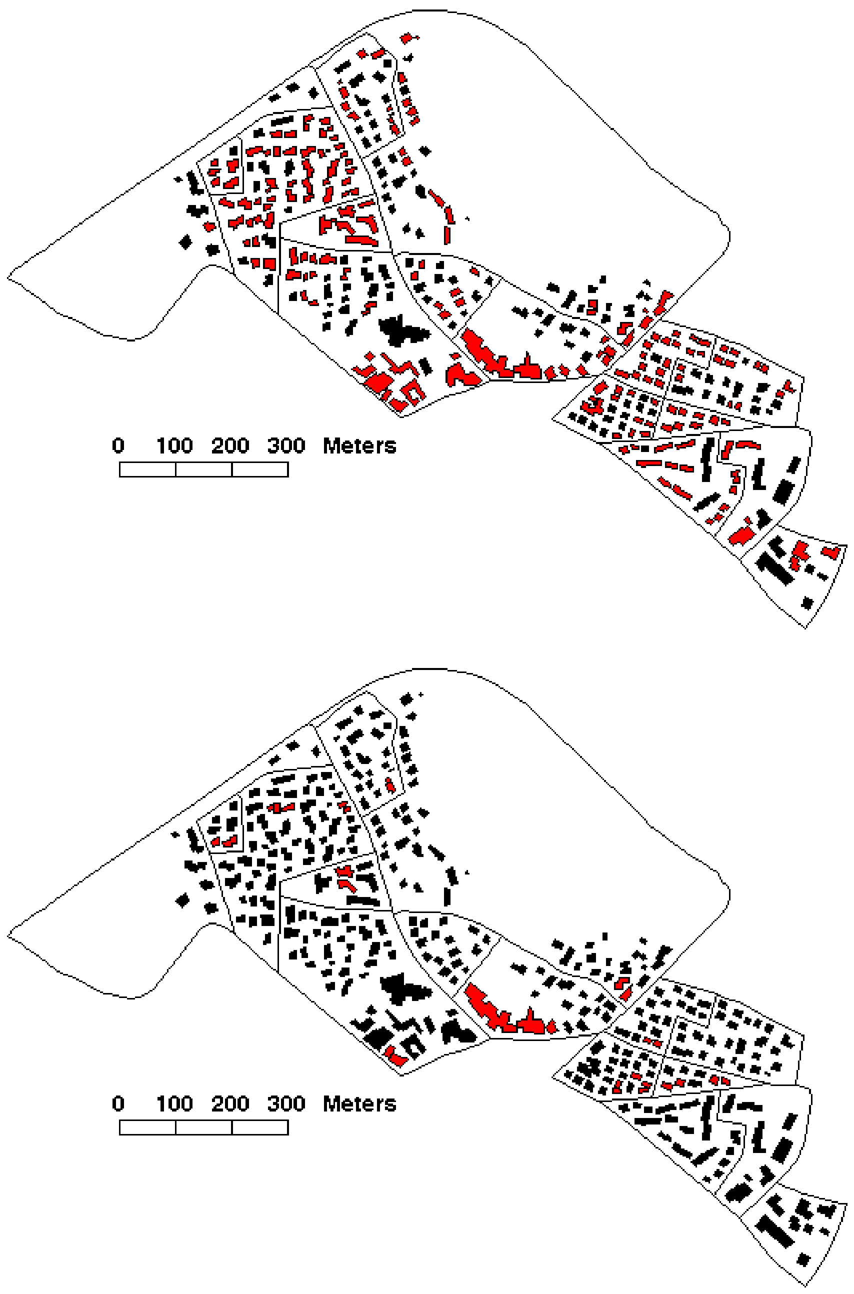


Figure 3. BDTopo data before (top) and after (bottom) application of simulated annealing algorithm Objects shown in red are involved in spatial conflict of some kind.

3 Execution Time Improvement

A shortcoming of the simulated annealing algorithm as it stands is that execution times remain too slow for it to be considered for use in applications requiring on-the-fly generalization (e.g. web mapping, in-car navigation and location based services). This section describes two techniques for reducing execution times. Both techniques achieve improvement by reducing the number of realizations having to be generated and tested.

Data Partitioning

Firstly, we suggest that execution can be speeded up by dividing the map into autonomous regions (i.e. such that there is no possibility ever of objects in a particular region coming into conflict with objects in any other region). The reasoning here is that by dividing the data it is possible to produce a separate annealing schedule for each region, each schedule being set to meet the specific demands of its associated region. One problem to be solved here is finding a technique for dividing up the map. In some instances it may be possible to make use of naturally occurring regions, such as those formed by a road network or administrative boundaries. Other situations will require analysis of the distribution of objects in order to find groupings of objects that sustain no influence from objects external to their group.

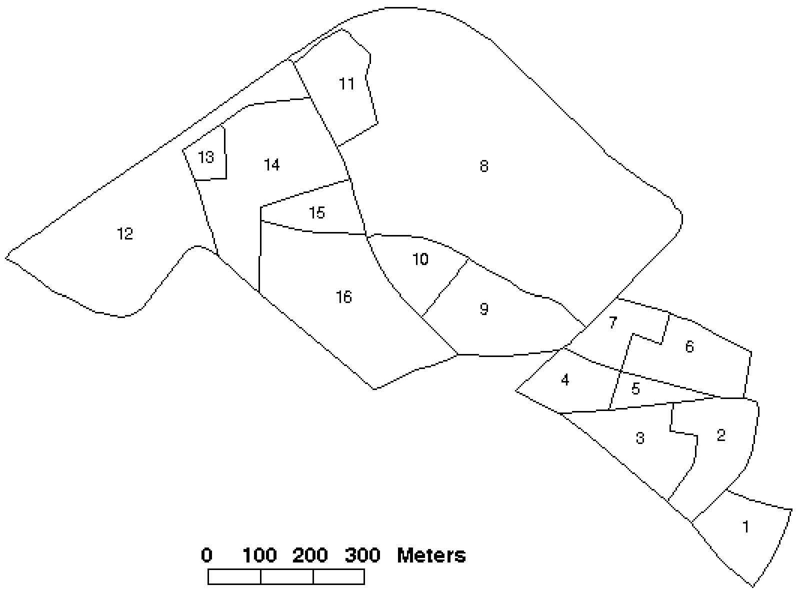


Figure 4. Division of BDTopo data into 16 autonomous regions.

The current work makes use of a road network to divide data. Using this approach, the BDTopo is divided into 16 regions (Figure 4). The simulated annealing algorithm can now be applied to regions individually. Through experimentation, it is possible to produce an appropriate annealing schedule for each region (Table 2). Note, that the initial temperature *V*, the temperature reduction value *X* and the maximum number of temperature stages *Z*, are the same in each case, and are the same as used previously. We found that the number of realizations tested could be controlled adequately by changing just the cooling rate values *W* and *Y*. The results obtained using these schedules are given in Table 2. It can be seen that, on average, the total number of realizations tested has been reduced to 78706.9. The average execution time sees a corresponding fall to 3.2s, a reduction of just over 75% . The conflict remaining in the data has not changed significantly.

region	**initial**		**W**	**Y**	**results (averages)**			
	Type-1	**Type-2**			**Type-1**	**Type-2**	**number of tests**	**execution time**
1	2	1	10	3	0.0	0.0	83.6	0.003
2	4	0	30	10	0.0	0.0	22.4	0.001
3	16	1	30	10	2.0	0.0	5869.5	0.247
4	6	7	30	10	0.0	0.0	6703.5	0.268
5	8	1	30	10	2.0	0.0	3201.9	0.128
6	12	2	30	10	0.0	0.0	5017.2	0.221
7	24	0	30	10	2.0	0.0	6923.9	0.284
8	18	4	10	3	0.0	0.0	2201.2	0.088
9	10	7	30	10	2.0	0.0	4490.3	0.186
10	6	0	30	10	0.0	0.0	3516.1	0.148
11	12	2	30	10	1.6	0.0	5666.7	0.224
12	0	1	10	3	0.0	0.0	14.2	0.001
13	6	1	30	10	2.0	0.0	2435.6	0.097
14	70	1	12	4	6.9	0.0	7037.3	0.303
15	6	2	30	10	1.8	0.0	2425.8	0.102
16	36	6	12	4	7.2	0.0	23097.7	0.935
total	236	36			27.5	0.0	78706.9	3.235

Table 2. Data partitioning results. . Initial temperature, temperature reduction value and maximum number of temperature stages are the same for each region. The values used are V=3.0, X=0.1 and Z=50.

Note again that in this approach, each region has its own annealing schedule, each schedule tailored to meet the requirements of its associated region. At present, the schedules are arrived at manually via experimentation. A difficulty arises when the procedure is applied to a different data set; the problem is that new annealing schedules have to be produced manually for each of the regions present in the data. The procedure cannot therefore be regarded as fully automated. In response to this, future research will consider methods for automatic generation of annealing parameters. One possible way forward is to use Artificial Neural Networks to generate suitable schedules based on data characteristics, such as number of objects, object density and amount of conflict. Initial inspection of results obtained to date suggest some correlation between these characteristics and the annealing parameters chosen.

Two-Stage Approach

Some authors suggest that optimization can be made more efficient by adopting a two-stage approach [6]. In two-stage simulated annealing a faster heuristic algorithm is used to replace the simulated annealing actions occurring at higher temperatures. This is followed by a conventional simulated annealing approach initiated at lower temperatures in an attempt to improve on the initial solution. The approach adopted in our implementation differs slightly in that the faster heuristic algorithm is replaced by simulated annealing in conjunction with an annealing schedule that involves an initial high temperature followed by rapid cooling. The second stage again makes use of simulated annealing, this time with a much lower initial temperature followed by much slower cooling. The annealing parameters used are given in Table 3. Experimental results (Table 4) show that execution times are reduced by approximately 75% when adopting the two-stage approach.

stage	**V**	**X**	**W**	**Y**	**Z**
1	3.3	0.4	10	10	50
2	0.2	0.1	30	10	50

Table 3. Parameters used in two-stage annealing.

Overall Improvement

The two modifications described have been combined in a single implementation (i.e. the two-stage procedure is applied to each of the 16 regions in turn). Experimental results, shown in Table 4, reveal an overall improvement of approximately 88%, with average execution times falling from 13.46s to 1.55s.

results	**average Type-1**	**s.d. Type-1**	**average Type-2**	**s.d. Type-2**	**average number of tests**	**s.d. number of tests**	**average execution time**	**s.d. execution time**
original	26.6	2.9	0.0	0.0	342302.2	18613.1	39.67	2.17
800MHz	25.4	2.7	0.0	0.0	328282.5	17439.3	13.46	0.67
partitioned	27.5	▨	0.0	▨	78706.9	▨	3.235	▨
two-stage	27.4	3.0	0.0	0.0	74154.5	3659.4	3.12	0.1
combined	26.5	2.8	0.0	0.0	37283.8	1864.7	1.55	0.08

Table 4. Two-stage approach results and combined results (original, 800MHz and partitioned results also shown).

4 Displacement Cost

Objects are displaced during the course of annealing, the overall aim being to reduce spatial conflict. However, many of the displacements prove unnecessary (i.e. they do not lead ultimately to a reduction in spatial conflict) and occur only as a consequence of the algorithms occasional acceptance of neutral and negative displacement. The result is a final display containing objects displaced from their original location without benefit. In order to minimize displacement of this type, an object displacement cost is introduced. If an object exists in a displaced state then a cost is incurred:

- Displacing an object carries a cost δc_3, where δ represents the magnitude of displacement.

The costs associated with spatial conflict and object modification are combined to give the overall cost associated with an object. For example, consider an object m_i that currently exists in a displaced state (cost = δc_3) and lies in conflict with two other polygonal objects (cost = $2c_1$) and one linear object (cost = c_2). Its associated cost would equal ($2c_1 + c_2 + \delta c_3$). Assigning appropriate values to c_1, c_2, and c_3 (i.e. $0<c_3<c_2,c_3$) creates an incentive for displaced objects to return, during the course of annealing, as near to their original location as is possible without a resulting increase

in conflict. Displays produced with and without consideration to displacement cost are shown in Figure 5.

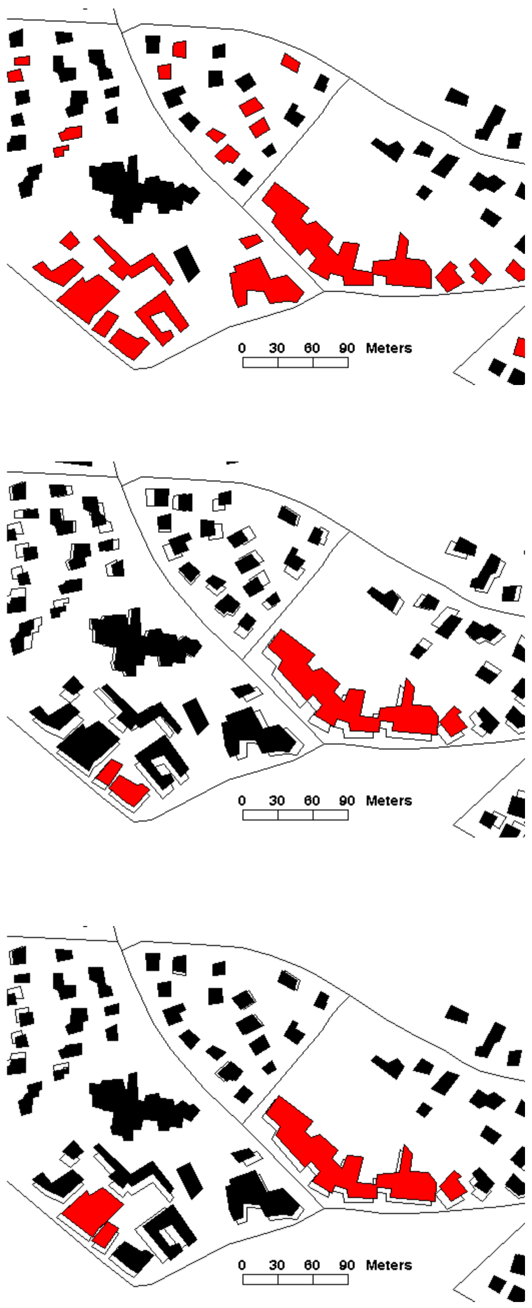


Figure 5. Example of the effect of including displacement cost. Top - original data. Middle - result obtained without consideration to displacement cost (original object locations shown in background). Bottom - result obtained when displacement cost is taken into account; unnecessary displacement has been reduced.

5 Additional Operators

A further shortcoming of the initial algorithm is that it does not guarantee the removal of all spatial conflict. For example, the best result obtained during experiments was a final Type-1 conflict cost of 22. It is clear that displacement on its own is not sufficient to resolve all conflict and additional generalization operators are required. We consider three additional operators:

- reduction;
- enlargement;
- deletion.

Object States

Again we consider a map display D made up of fixed linear objects F and n modifiable detached polygonal objects M. Each modifiable object $m_i \in M$ has k possible states providing us with k^n possible configurations of D. At any given time a particular object m_i exists in one of its k possible states; its (k-1) modified states arise as a result of either displacing the object, reducing the size of the object, increasing the size of the object or deleting the object. In the current implementation, the possible object states are:

- *Unmodified State*. Each object m_i has a single unmodified state;
- *Deleted State*. An object m_i has a single deleted state trial position; this trial position represents the situation where the object has been removed from the display.
- *Displaced States*. Each object m_i has q displaced state trial positions, which are distributed evenly about the object.
- *Reduced States*. Each object m_i has $q+1$ reduced state trial positions; these trial positions result from applying a scaling factor s_r ($0<s_r<1$) to the unmodified state and displaced states of m_i.
- *Enlarged States*. Each object m_i has $q+1$ enlarged state trial position; these trial positions result from applying a scaling factor s_e ($s_e \geq 1$) to the unmodified state and displaced states of m_i.

The object reduction value s_r is fixed for all objects (e.g. 0.75). The object enlargement value s_e varies for each object and is dependant on object display area. For an object with display area less than a minimum area tolerance a_{min}, s_e is set so as to increase object display area to a_{min}. Objects with display area greater than or equal to a_{min} have s_e set to 1 (i.e. its application has no effect).

Evaluation of Map Display

For a particular configuration D_j, an object m_i has an associated cost. This cost is a measure of both the spatial conflict in which the object is involved and the extent to which the object is modified.

Three categories of spatial conflict are considered:

- Type-1. Conflict between a pair of polygonal objects (i.e. m_i and m_j lie too close to each other to be distinguishable). This conflict occurs when the minimum separating distance (in viewing coordinates) between m_i and m_j is less than some predefined threshold d_{min1}. An occurrence of this type of conflict carries a cost c_1;
- Type-2. Conflict between a polygonal object and a linear object (i.e. m_i and $f_j \in F$ lie too close to each other to be distinguishable). This conflict occurs when the minimum separating

distance (in viewing coordinates) between m_i and f_j is less than some predefined threshold d_{min2}. An occurrence of this type of conflict carries a cost c_2.

- Type-3. Conflict involving a single object (i.e. m_i is too small for it to be seen clearly). This type of conflict occurs when area of m_i (in viewing coordinates) is less than a_{min}. An occurrence of this type of conflict carries a cost c_3;

If an object exists in a modified state then a cost is incurred:

- Displacing an object carries a cost δc_4, where δ represents the magnitude of displacement;
- Reducing an object carries a cost c_5, where c_5 is proportional to the scale of reduction;
- Enlarging an object carries a cost c_6, where c_6 is proportional to the scale of enlargement;
- Deleting an object carries a cost c_7.

The costs associated with spatial conflict and object modification are combined to give the overall cost associated with an object. For example, consider an object m_i that has been reduced in size and lies in conflict with two other polygonal objects and one linear object. Its associated cost would equal ($2c_1 + c_2 + c_5$). As before, the total cost $\mathbf{C}(D_j)$ associated with a realization D_j is found by summing the costs associated with each object m_i; our goal is to find a minimum cost configuration D_{min}.

Implementation and Testing
In implementation terms, introducing reduction, enlargement and deletion capabilities is straightforward; these additional modified states of an object are treated as additional trial positions for that object. Object reduction and object enlargement are achieved by applying a suitable scaling factor (s_r and s_e) to the object, while deletion is accommodated by means of a simple Boolean flag (i.e. an object is either deleted or not deleted). The cost function $\boldsymbol{C}$ is updated to take account of the additional costs.

Cost Setting
It is important to make sure that conflict costs (c_1, c_2 and c_3) and object modification costs (c_4, c_5, c_6 and c_7) are set appropriately; it is these costs that govern the likelihood of any given object/trial position pairing being accepted should they be chosen during the annealing process. As such, the cost values must be set so as to accommodate any orders of precedence that might exist between the various operators and conflict types. For example, consider an application that simply requires the removal of all spatial conflict (i.e. reducing spatial conflict is the stated primary goal, and minimizing object modification is an implied secondary goal). This might be achieved by assigning a relatively high value to each of the conflict costs (e.g. c_1=100, c_2=100 and c_3=100) and a relatively low value to each of the object modification costs (e.g. c_4= 5, c_5=5, c_6=5 and c_7=5). It could be the case, however, that the modification operators have an order of precedence in which object displacement is preferred to object reduction, which in turn is preferred to object deletion; the relevant cost value are changed accordingly (c_4= 5, c_5=10 and c_7=15).

Displacement and Deletion
Deletion was the first of the additional operators to be tested. The main thing to note here is that care must be taken when setting the deletion cost value c_7. If it is set too high then, in some situations, not enough deletion takes place, and, as a consequence, spatial conflict is not always removed (i.e. the cost of deleting is greater than the cost associated with the spatial conflict). On the other hand, if c_7 is set too low, then too many deletions tend to occur (i.e. objects are prematurely

deleted in situations where displacement could have succeeded). Finding values that always result in necessary deletions only is not always possible. In our experiments, we have used c_7 values that always result in removal of all spatial conflict (sometimes leading to over-use of deletion). An example of each of the deletion cost setting scenarios is given in Figure 6.

Reduction and Enlargement
Object reduction and enlargement operators are now introduced. Again note that care must be taken when setting object modification costs and regard must be given to any operator precedence that may exist. Figure 7 shows a display produced with object displacement preferred to object reduction, and object reduction preferred to object deletion

Cost Weighting
When generalizing a map it is important to consider the relative importance of objects; important objects should be less prone to modification than unimportant objects. Consider a tourist map in which an object m_m, representing a museum, lies in conflict with an object m_h, representing an ordinary house. In this context, m_m can be regarded as being more important than m_h , and hence should be less susceptible to generalization. In this situation there are a number of alternative conflict resolution strategies that could be employed. For example, conflict resolution could initially involve displacement and reduction of m_h only; if this failed, m_h could be deleted. An alternative strategy might again involve the initial displacement and reduction of m_h only. If this failed then the next step would be displace and reduce m_h and m_m in combination; continued failure at this stage would result in the deletion of m_h. Relative object importance is incorporated into the simulated annealing procedure by assigning a cost weighting w_i to each object m_i. Whenever an object m_i and trial position k_j pairing are chosen during the annealing process, the cost associated with k_j is multiplied by w_i to give a modified cost. A particular cost weighting value w_i will be based on one or more of the attributes of m_i. In our experiments to date, and in the absence of any other measure of importance, an object's importance, and hence its cost weighting value, is assumed to be proportional to object area (with large objects deemed more important than small objects). Figure 8 (top) shows output produced with no account taken of object importance. The large object to the left has been deleted in order to resolve conflict. If we consider large objects to be more important than small, then it would make more sense to have deleted the smaller object to the right. This can be achieved by making use of appropriate cost weightings, as shown in Figure 8 (bottom).

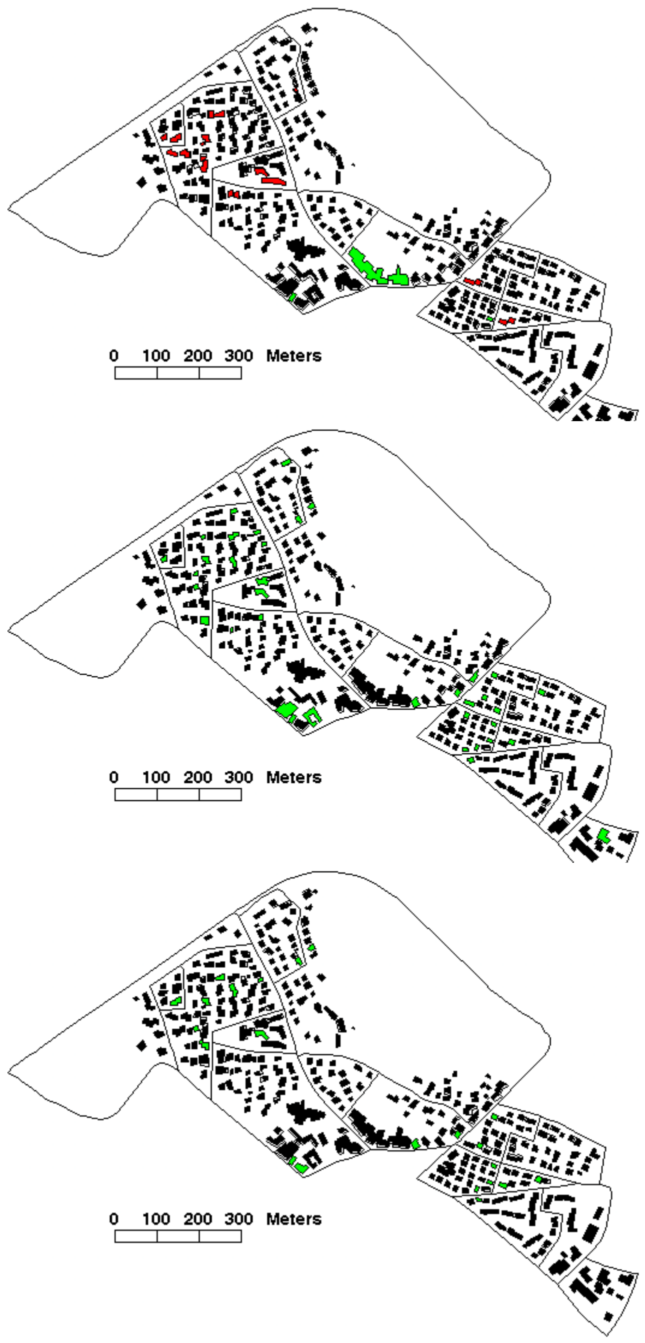


Figure 6. Application of deletion operator. Deleted objects shown in green. Top - deletion cost set too high, not all spatial conflict removed. Middle - deletion cost set too low, over-deletion. Bottom - better deletion cost, some unnecessary deletion.

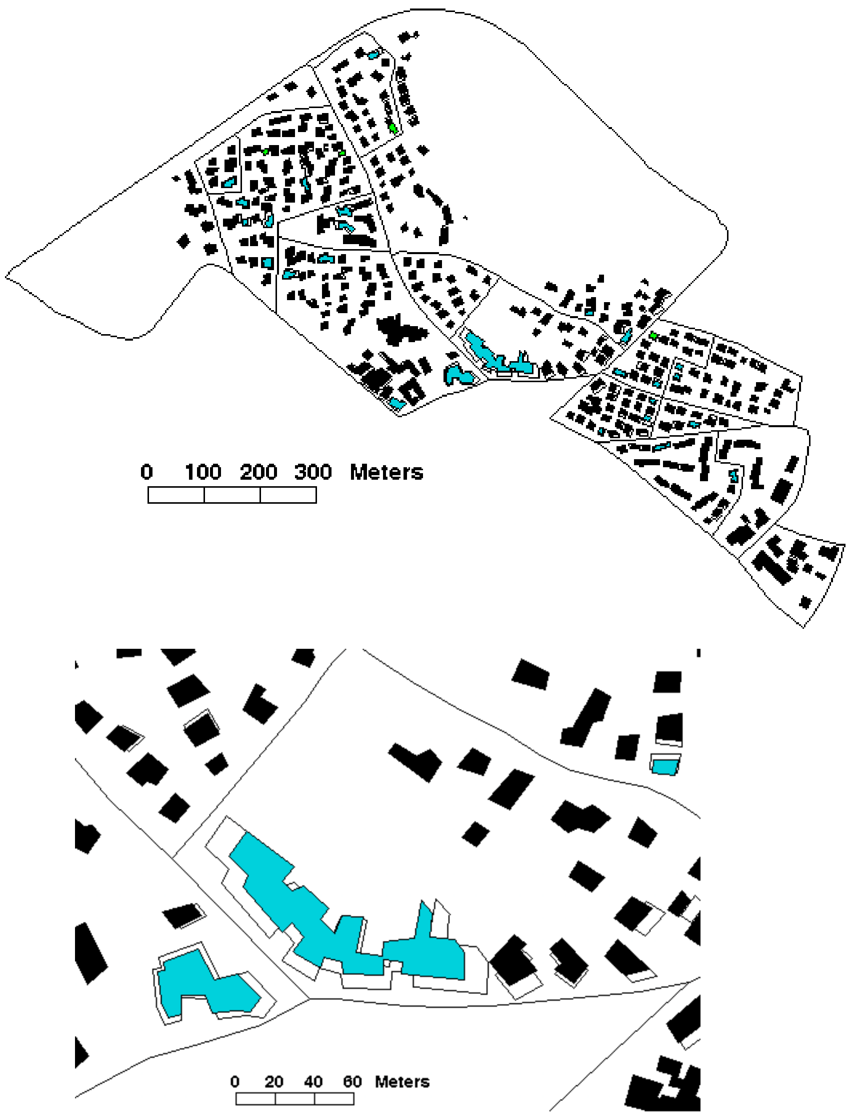
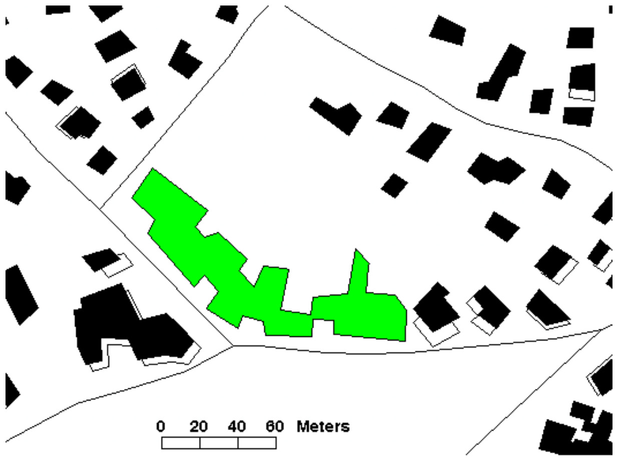


Figure 7. Full and zoomed-in version of display obtained using object displacement, reduction, enlargement and deletion. Reduced objects shown in blue (no object enlargement required).



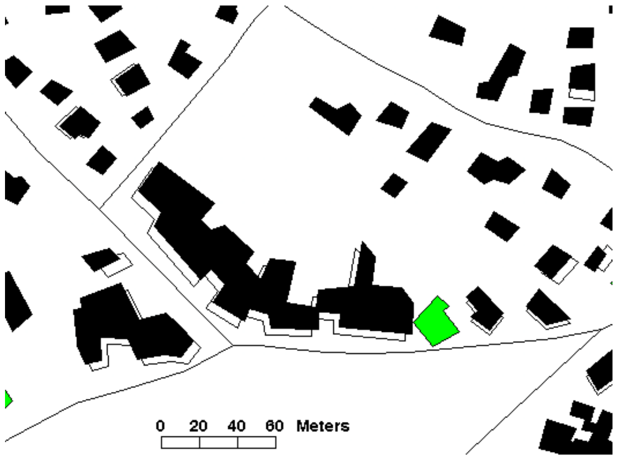


Figure 8. Results obtained with (bottom) and without (top) cost weighting.

Group Application

In the current implementation, the modification operators are applied to a single object only at any given time. Other strategies could be accommodated using the simulated annealing approach. For example, there is no reason why deletion could not be applied to groups of objects on-mass, where groupings are determined by some attribute such as object class or object area. The on-mass deletion could be treated as just another trial position for the objects in question (i.e. the objects could be reintroduced as a group at some later stage of the annealing process), or could be applied as a more permanent culling of objects at pertinent stages of the generalization process (e.g. at the start following an initial assessment of the problem or at certain stages during the generalization process when it becomes apparent that the other operators will not succeed in resolving conflict). Similarly an associated collection of objects, perhaps representing a row of buildings, could be displaced (or reduced or enlarged) as a group, so as to maintain, for example, feature alignment.

Acknowledgement

Nathan Thomas is funded by EPSRC CASE Studentship 00802722, which is carried out in collaboration with the Ordnance Survey. The authors express thanks to the Institut Géographique National for permission to use their data in parts of the work presented.

References

[1] Ware, J.M. and Jones, C.B., 1998, “Conflict Reduction in Map Generalization Using Iterative Improvement”, Geoinformatica, 2(4): 383-407.
[2] Ware, J.M., Jones, C.B. and Lonergan, M.E., 2000, "Map generalization by Iterative Improvement", Presented at First International Conference on Geographic Information Science (GIScience 2000).
[3] Zoraster, S., 1997, “Practical results using simulated annealing for point feature label placement”, Cartography and Geographical Information Systems, 24(4): 228-238.
[4] Kirkpatrick, S., Gelath, C.D. and Vecchi, M.P., 1983, “Optimization by simulated annealing”, Science, 220:671-680.
[5] Emden-Weinert, T. and Proksch, M., 1999, “Best practice simulated annealing for the airline crew scheduling problem”, Journal of Heuristics, 5(4): 403-418.
[6] Varanelli, J. and Cohoon, J.P., 1993, “Two-Stage Simulated Annealing”, ACM Physical Design Workshop, Lake Arrowhead, CA, April 1993.
[7] Russell, S. and Norvig, P., 1995, Artificial Intelligence : A Modern Approach (Prentice-Hall), 1995.
[8] Jones, C.B. and Ware, J.M., 1998, “Proximity relations with triangulated spatial models”, The Computer Journal, Volume 41, Number 2, pages 71-83.
[9] Jones, C.B., Ware, J.M. and Eynon. C.D., 1999, “Triangulated Spatial Models and Neighbourhood Search: An Experimental Comparison with Quadtrees”, The Visual Computer, Volume 15, Number 5, pages 235-248.