Reducing Graphic Conflict In Scale Reduced Maps Using A Genetic Algorithm

Dr. Ian D. Wilson
School of Technology, University of Glamorgan, Pontypridd CF37 1DL, UK

Dr. J. Mark Ware
School of Computing, University of Glamorgan, Pontypridd CF37 1DL, UK
jmware@glam.ac.uk

Prof. J. Andrew Ware
School of Computing, University of Glamorgan, Pontypridd CF37 1DL, UK

1 Problem Definition

Effective map generalisation involves careful examination of the interactions between all map symbols. 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 (for example, in the alignment of buildings, parallelism between neighbouring rivers and roads, and the clustering of woods and lakes). 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, the application of an individual operator may have an effect on a map symbol that was not previously in conflict, resulting in propagation of conflict within the map space. A partial solution to this problem is for objects to be moved after scaling so that they remain distinct, visual entities. The research presented here, which is related to previous work [1] [2], describes a procedure that makes use of the displacement of multiple map objects in order to resolve graphic conflict.

The solution presented here is a combinatorial problem the size of which depends upon the number of objects represented and the position of each object within an (x, y) co-ordinate space. Given that it is undesirable to displace an object too far from its original position, it is necessary to constrain its movement to within a short distance, which reduces the co-ordinate space considered for each object. Here, each of n discrete polygonal objects is assigned a continuous space of v co-ordinates (x, y pairs) into which they can possibly move. This results in v^n possible distinct map configurations; the assumption being that some of these configurations will contain less conflict than the original. Finding an acceptable configuration by means of an exhaustive search is, however, not practical for realistic values of n and v giving rise to the need for a heuristic search approach. The next section provides an introduction to the heuristic procedure, the Genetic Algorithm, selected by the authors.

2 Genetic Algorithms

Genetic Algorithms are adaptive search methods that can be used to solve optimisation problems. They are based on the genetic process of evolution within biological organisms. Which is to say that, over many generations, populations have evolved according to the principles of natural selection. By adopting this process, a GA is able to ‘evolve’ solutions to real world problems [3].

Solutions are evolved utilising a genome (or structure of the problem, where a single instance of which represents a solution to the problem) and a genetic algorithm (the procedure utilised to control how evolution takes place). The GA makes use of genome operators (associated with the genome) and selection/replacement strategies (associated with the GA) to generate new individuals. The GA uses an objective function to determine how fit each of these individual genomes is for survival. So, given a choice of GA, three things that are required to solve a problem are given below:

- The structure of the problem must be defined and a representation for the genome determined;
- Given the genome, define suitable genetic operators;
- Using the genome, define an objective function that measures the relative quality of a solution.

In summary, when using a GA to solve an optimisation problem, a series of variables are combined to form a single solution to the problem within a single genome. The GA creates a population of solutions based on the genome. The GA then operates on this population to evolve an optimum, or near optimum, solution to the problem utilising the objective function.

2.1 The Genome

This section outlines the decision making process that determines how an individual solution (the genome) should be modelled and physically represented. When defining a representation appropriate to the problem at hand, a data structure that is minimal but also completely expressive should be selected. For example, if a real value and a number of integers can represent a solution to a problem, then the genome's data structure should be defined using these characteristics. The representation should not include any information other than what is required to express a solution to the problem. Although it may appear beneficial to include extra genetic material beyond that which is required to fully express a solution, this tends to increase the size of the search space and hinder the performance of the algorithm. In addition to defining the structure of the genome's content, ranges of acceptable values for each constituent part of the genome are also provided in a separate data structure. For example, each of a set of integers may be constrained to fall within an individual, pre-specified, range. Finally, each genome will have a 'fitness' score associated with it that determines its prospects for selection. This representation is an independent component of the general GA procedure, allowing for separate decision making processes to be made. For example, a different GA procedure might be adopted without any need to change the structure of the genome. This is possible because the operators necessary to evolve new solutions are associated with the genome and not the GA itself.

2.2 The Genome Operators

Given a general GA procedure and genome it is also necessary to determine how operators specific to the genome should behave. Three operators can be applied to the genome, these being initialisation, mutation and crossover. These operators allow a population to be given a particular bias and allow for mutations or crossovers specific to the problem representation.

The initialisation operator determines how each genome is initialised. Here, the genome is 'filled' with the genetic material from which all new solutions will evolve. Next, the mutation operator defines the procedure for mutating the genome. Mutation, when applied to a child, randomly alters a gene with a small probability. It provides a small amount of random search that facilitates convergence at the global optimum. Finally, the crossover operator defines the

procedure for generating a child from two parent genomes. The crossover operator produces new individuals as 'offspring', which share some features taken from each 'parent'.

These operators are independent functions in themselves, specific to the structure of the genome, which may be altered in isolation to the other components described in these sections. For example, the crossover operator might be changed from a single point to a two-point implementation without any need to adjust the other components.

2.3 Objective Functions and Fitness Scaling

Genetic algorithms are often more attractive than gradient search methods because they do not require complicated differential equations or a smooth search space. The genetic algorithm needs only a single measure of how good an individual is compared with the other individuals. The objective function provides this, needing only a genome, or solution, and genome specific instructions for assigning and returning a measure of the solution's quality. The objective score is the raw value returned by the objective function. The fitness score is the possibly transformed objective score used by the genetic algorithm to determine the fitness of individuals for mating. Typically, the fitness score is obtained by a linear scaling of the raw objective scores. Given this, the objective function can be altered in isolation from the GA procedure and genome operators, and, once a representation for the problem has been decided upon, without any need to change the structure of the genome.

2.4 The Genetic Algorithm

Here, we present an overview of a general GA procedure, explaining how each phase fits into the evolutionary process. The GA procedure (illustrated in Figure 1) determines when the population is initialised, which individuals should survive, which should reproduce, and which should die. At each generation certain, highly fit, individuals are allowed to reproduce (through selection) by 'breeding' with other individuals within the population. Offspring may then undergo mutation, which is to say that a small part of their genetic material is altered. Offspring are then inserted into the population, in general replacing the worst members of the existing population although other strategies exist (*e.g.* random). Typically, evolution stops after a given number of generations, but fitness of best solution, population convergence, or any other problem specific criterion can be used.

2.5 Steady-state GA

Of the variations of GA available, the work presented in this paper utilised the Steady State GA (SSGA), similar to that outlined by [4]. The SSGA uses overlapping populations with a pre-specified amount of overlap (expressed here as a percentage), these being the initial and next generation populations. The SSGA first creates a population of individuals by cloning the initial genome. Then, at each generation during evolution, the SSGA creates a temporary population of individuals, adds these to the previous population and then removes the worst individuals in order that the current population is returned to its original size. This strategy means that the newly generated offspring may or may not remain within the new population, dependant upon how they measure up against the existing members of the population. The following sections examine each component of our SSGA implementation.

3 Problem decomposition and modelling

In this section, considerations relating to the map generalisation model are described, and an overview of the model's underlying representation and physical implementation is provided, along with a detailed discussion about the objective function and its mathematical formulation.

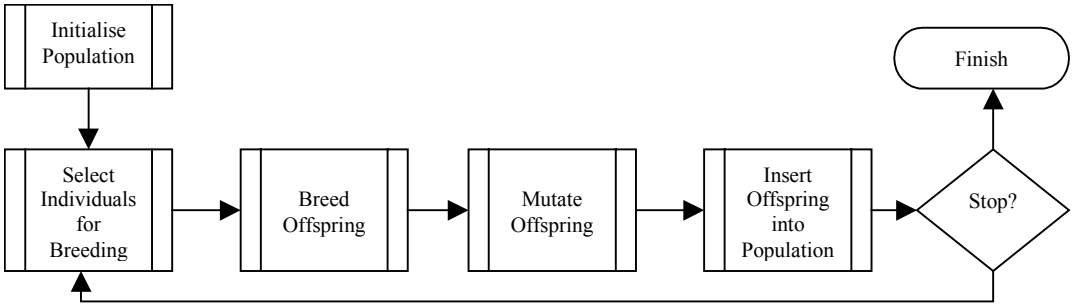

Fig. 1. Genetic Algorithm Evolution Procedure.

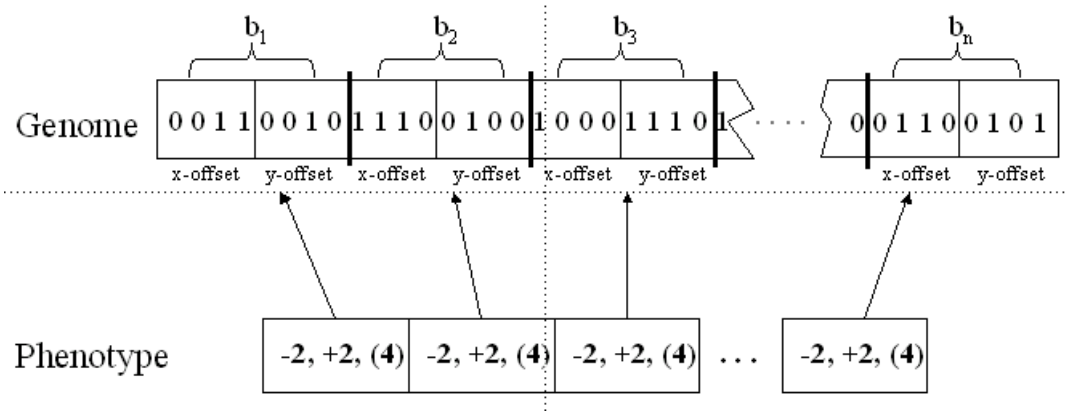

Fig. 2. Genome and Phenotype. Each offset has been allocated 4 bits and can range between -2 and +2.

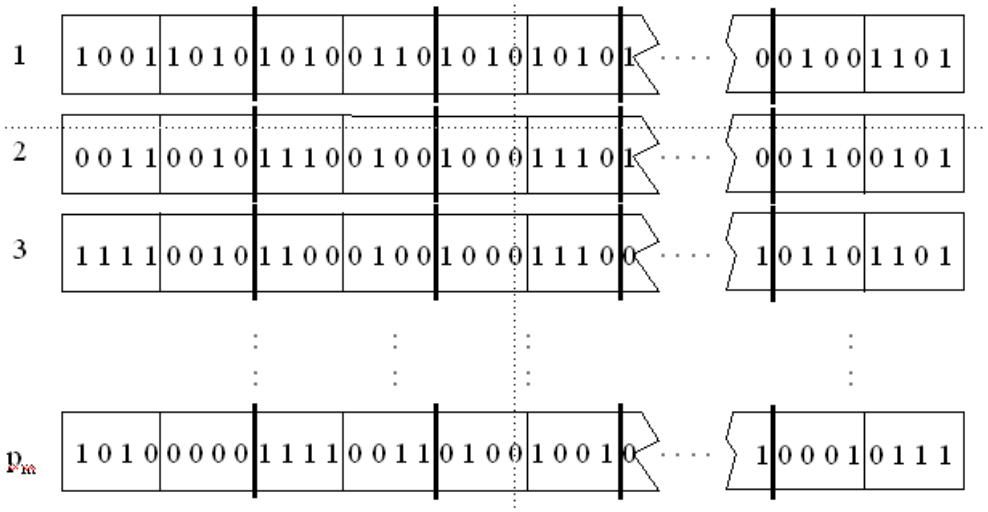

Fig. 3. A population of genomes.

3.1 Underlying structure

Here we examine the structure of a map display and introduce the concept of a Displacement Vector Template (DVT), within which an object can be moved. A map display is made up of fixed, linear objects and modifiable, detached polygonal objects. Each linear or polygonal object, n, is represented by a collection of vertices, $\gamma \in \{\gamma_{n1} \ldots \gamma_{ni}\}$ (stored as x, y co-ordinates) and edges (arcs between vertices). In order that conflict is removed polygonal objects are displaced to a new position where conflict is minimised. The nominally available search space is the whole x, y co-ordinate space but this is unrealistic, and unnecessary as we will show, so a continuous neighbourhood, of radius d, of accessible co-ordinates is associated with each polygonal object.

The final position an object can occupy can not extend beyond the area delineated by the DVT. Therefore, each modifiable object has a continuous space of v (x, y pairs) possible states, providing a total of v^n possible configurations for a given map display. For each solution, an object exists in one of its displaced positions at any given time. An object's initial map position within its DVT is stored separately from its associated displacement value, with the two combining to give its current location in geometric space.

3.2 Map display evaluation

The success of any discrete optimisation problem rests upon its objective function, the purpose of which is to provide a measure for any given solution that represents its relative quality. The objective function used here works by calculating and summing the penalties associated with the collection of objects within our state representation. The objective score associated with a given configuration is an abstraction of the penalties associated with the relationships between each conflicting polygonal and linear object and, to a lesser extent, the distance each object has been displaced. A spatial index together with a search procedure [5] is used to quickly identify conflicting objects. The extent to which an object is in conflict determines its individual associated penalty. In full, we consider two categories of spatial conflict and an additional measure of quality within our objective function, namely:

- Conflict between a pair of polygonal objects, where their proximity renders them indistinguishable from each other). This conflict occurs when the minimum separating distance (in viewing co-ordinates) between two objects is less than some predefined threshold.
- Conflict between a polygonal object and a linear object, where their proximity renders them indistinguishable or they overlap. This conflict occurs when the minimum separating distance (in viewing co-ordinates) between a polygonal and linear object is less than some predefined threshold.
- Distance each object is displaced (in viewing co-ordinates) from its starting position. This measure helps minimise the amount of disruption within the generalised map display relative to its source.

3.2.1 Underlying model and definitions.

The object function used to evaluate solutions to the map generalisation problem requires a number of definitions that model the problem's underlying structure, specifically:

- O: $\{o_1,\ldots,o_n\}$ is the set of all polygonal objects;
- L: $\{l_1,\ldots, l_r\}$ is the set of all linear objects;
- n is the number of polygonal objects;

- r is the number of linear objects;
- do_{min} is the minimum distance threshold between polygonal objects;
- dl_{min} is the minimum distance threshold between linear and polygonal objects;
- $DO_{ij} = 1$ if o_I is within do_{min} of o_j else 0;
- $DL_{ij} = 1$ if o_I is within dl_{min} of o_j else 0;
- dx_I = the normalised (-0.75 to +0.75) distance an object has been displaced in the X axis;
- dy_I = the normalised (-0.75 to +0.75) distance an object has been displaced in the Y axis.

3.2.2 The object relationship fitness function

The objective function used to evaluate solutions to the map generalisation problem examines the weighted relationship between linear and polygonal objects. The general expression of the objective function is:

$$f = ((f_1 * w_1) + (f_2 * w_2) + (f_3 * w_3)) \qquad (1)$$

Where f_i and w_i represent, respectively, the number of conflicting objects and the weight of that particular measure, with a low value of f indicating a good solution.

The first term of the objective function, f_1, counts the number of polygonal objects that conflict with each other. Minimising the number of polygonal objects in conflict with each other produces a more attractive map display.

$$f_1 = \sum_{i=1}^{n} \sum_{j=1}^{n} DO_{ij} \qquad (2)$$

The second term of the objective function, f_2, counts the number of polygonal objects that conflict with each linear object. Again, minimising the number of polygonal objects in conflict with each linear object produces a more attractive display.

$$f_2 = \sum_{i=1}^{r} \sum_{j=1}^{n} DL_{ij} \qquad (3)$$

Finally, the third term of the objective function, f_3, sums the normalised, absolute, distance each object has been displaced from its starting position. Normalising this displacement value minimises its impact upon generalisation relative to the two conflict resolution operators described above.

$$f_3 = \sum_{i=1}^{n} \left(dx_i^2 + dy_i^2 \right)^{1/2} \qquad (4)$$

4 Implementation

In this section, we present our implementation of the GA algorithm for solving the map-generalisation problem. Special consideration is given to the sub-ordinate heuristics used to direct the procedure through the search space.

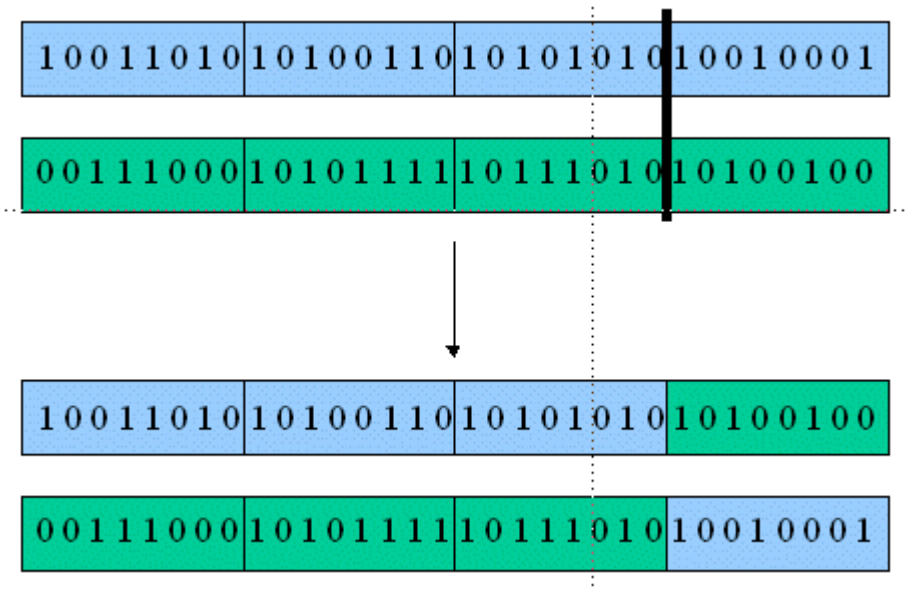

Fig. 4. Single point crossover.

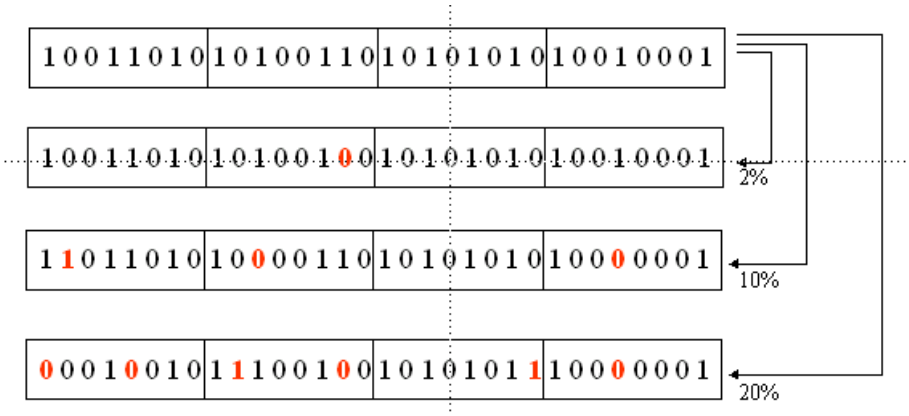

Fig. 5. Random mutation.

4.1 Configuration, search space and cost function

Given a map display comprised of *n* modifiable polygonal and *r* fixed linear objects, a configuration *s* corresponds to an individual arrangement of these objects. The search space S is therefore composed of all such configurations. According to equation (1), for each solution $s \in S$, $f(s)$ corresponds to a combination of the total number of polygonal and linear objects in conflict, and a measure of how much objects have been moved from their starting position.

4.2 State representation

Our state representation, the genome, is physically stored as an array of real numbers, with each pair of real numbers corresponding to the x, y displacement value associated with that object. Another data structure, called a genome phenotype, is used to both physically constrain the range of each real number within the genome and determine how many binary bits will be used to represent this value. This allows a physical implementation for an individual solution within

the state space of all possible states to be stored in real co-ordinates that map to a binary string during evolution. Genome and phenotype are illustrated in Figure 2.

4.3 Population size and maximum generations

The complexity of each sub-problem is, in part, a function of the different number of objects within each map segment and the range of initial conflict after scaling. Therefore, for any given map segment (m), the population size (p_m) and maximum number of generations (g_m) are set using experimentally determined heuristics, namely:

$$p_m = 4C(s_m) \quad (5)$$

$$g_m = 15s_m \quad (6)$$

These heuristic values were determined experimentally and allow solutions to be generated that have optimal, or near optimal, measures of conflict. A sample population is illustrated in Figure 3.

4.4 Neighbourhood examination

Given that each object can only be displaced into its immediate area, only the edges of objects that could feasibly come into conflict with the edge of given object need be checked. Therefore, a list of edges that can theoretically come into conflict with a given a edge are stored in a list associated with that object. This data structure is parsed to determine if, indeed, any of these edges have come into contact with a given edge when generating the objective score for a particular configuration. Adopting this strategy significantly decreased the number of edges checked for conflict during evaluation resulting in much reduced computation times.

4.5 Crossover, replacement and mutation

The probability of crossover (reproduction) determines how often crossover will occur at each generation. The single point crossover strategy (illustrated in Figure 4) was adopted for all experiments. If the probability of crossover is set to zero (asexual reproduction), then each offspring would be at this stage an exact copy of its parents (although the mutation operator may result in offspring that are different to their parents).

Conversely, if the crossover probability is set to 100%, then all offspring are made by crossing part of each parent with the other. Each time crossover occurs, an offspring is created using material from each of its parents. The results for all experiments presented in this paper were generated using a crossover percentage of 50%, which is to say that at each generation 50% of the new population were generated by splicing two parts of each genomes' parents together to make another genome.

The probability of mutation determines how much of an each genome's genetic material is altered, or mutated. If mutation is performed, part of chromosome is changed. For example, if the mutation probability was 100%, then each element within the genome would be changed, but if it was 0%, nothing would be changed. Mutation is introduced to facilitate movement away from local optimality to a place closer to global optimality. However, mutation should not occur too often as this would be detrimental to the search exercise. Consequently, the results presented here were generated using a 2% mutation probability, which was determined experimentally, utilising a single bit flip mutation operator (illustrated in Figure 5).

5 Initial Results

The algorithm has been implemented in VC++ 6.0, utilsing Wall's Galib library [6]. The implementation has been tested using building polygon data extracted from OS MasterMap data and road centre lines taken from OS Oscar data. An example of initial results is illustrated in Figure 6. It can be seen that the the algorithm has resolved a large proporation of the graphic conflict that resulted from road symbolisation.

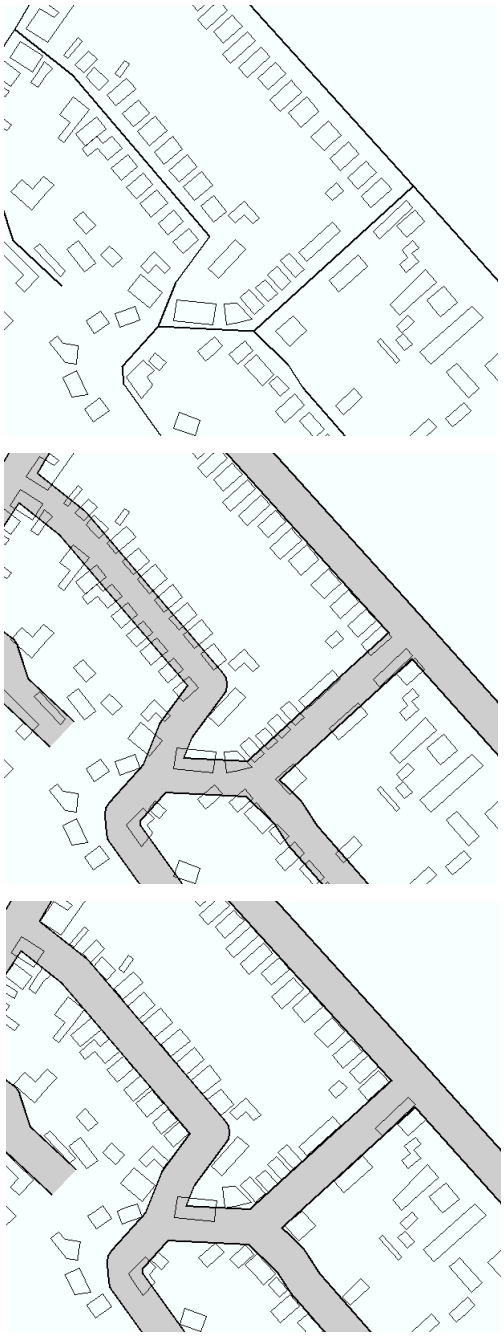

Fig. 6. Sample output, object displacement only.

6 Scaling Operator

In some instances conflict has not been resolved; this is not so much to do with limitations of the algorithm, but rather a limitation of the displacement operator in general. It is clear that additional operators are required. A relatively straighforward improvement can be made by introducing a feature scaling operator that acts to reduce the size of features in situations where displacement alone does not succeed. This can be achieved by modifying the genome representation such that each object is assigned additional bits that correspond to a scaling factor (Figure 7); the cost function is also updated to take size reduction into account. An example of results obtained is illustrated in Figure 8.

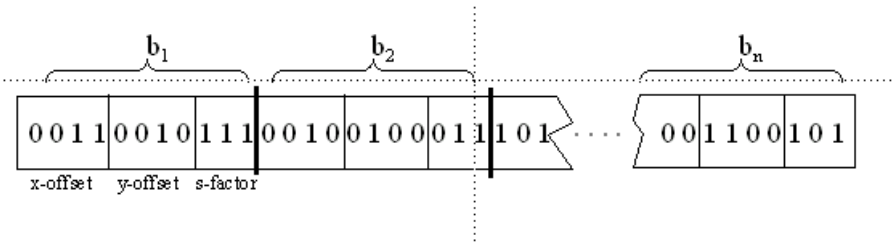

Fig. 7. Scaling factor added to genome.

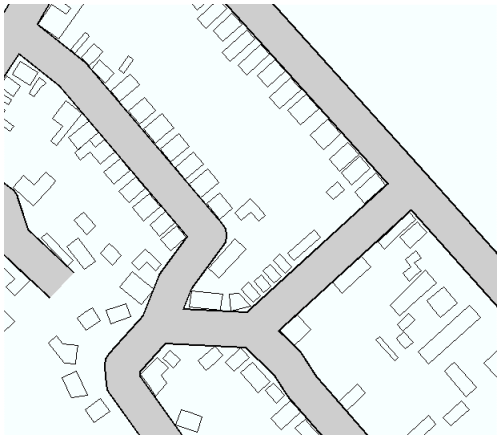

Fig. 8. Sample output, object displacement and scaling.

7 Conclusion

Given these promising, initial, experiences with the application of a GA, it is the authors' intention to expand upon the work presented. The displacement method presented here works well where there is plenty of free map space into which objects may move. In situations where displacement is either impractical because of space constraints or too expensive in terms of overall disruption to the map display, further additional operators (e.g. deletion, amalgamation, and simplification) will be used in combination. Future work will concentrate on introducing these operators. The cost function will be expanded to take account of each new operator, with appropriate penalties consistent with the map disruption included.

8 Acknowledgement

The authors express thanks to the Ordnance Survey for providing the data used in this work.

9 References

[1] Ware, J.M. and Jones, C.B., 1998, “Conflict Reduction in Map Generalisation Using Iterative Improvement”, Geoinformatica, 2:4, 383-407.
[2] Ware, J.M., Jones, C.B. and Thomas, N., 2001, “Map Generalization, Object Displacement and Simulated Annealing: Two Techniques for Execution Time Improvement”, Proceedings of GIS Research UK 2001 Conference (GISRUK'01), 36-38.
[3] Goldberg, D. (1989). Genetic Algorithms in Search, Optimisation and Machine Learning. Addisen-Wesley Publishing Company, Inc.
[4] DeJong, K. and Sarma, J. (1993) Generation Gaps Revisited. Foundations of Genetic Algorithms -2-, D. Whitley, ed. Morgan-Kaufmann.
[5] Jones, C.B., J.M. Ware and C.D. Eynon. (1999). Triangulated Spatial Models and Neighbourhood Search: An Experimental Comparison with Quadtrees. The Visual Computer 15(5), 235-248.
[6] Wall, M. Galib - A C++ Library of Genetic Algorithm Components. http://lancet.mit.edu/ga/.