Reducing Epistasis in Combinatorial Problems by Expansive Coding

David Beasley, David R. Bull and Ralph R. Martin
pages 400–407

The Proceeding of the Fifth International Conference on Genetic Algorithms
University of Illinois at Urbana–Champaign
July 17–21 1993
Edited by Stephanie Forrest
© Morgan Kaufmann Publishers, Inc.
San Mateo, CA, USA

Reducing Epistasis in Combinatorial Problems by Expansive Coding

David Beasley
Dept. of Computing Mathematics,
University of Wales
College of Cardiff
Cardiff, CF2 4YN, UK
david.beasley@cm.cf.ac.uk

David R. Bull
Dept. of Electrical and
Electronic Engineering,
University of Bristol
Bristol, BS8 1TR, UK
dave.bull@bristol.ac.uk

Ralph R. Martin
Dept. of Computing Mathematics,
University of Wales
College of Cardiff
Cardiff, CF2 4YN, UK
ralph.martin@cm.cf.ac.uk

Abstract

This paper describes a new technique for tackling highly epistatic combinatorial optimization problems. Rather than having a simple representation, simple operators, a simple fitness function, but a highly epistatic search space, this technique is intended to spread the problem's complexity more evenly. Using our new technique, known as *expansive coding*, the representation, operators and fitness function become more complicated, but the search space becomes less epistatic, and therefore easier for a GA to tackle. In effect, the combinatorial task is changed to a function optimization one. We demonstrate how this technique can be applied in the field of arithmetic algorithm design/electronic circuit simplification. In the design of a multiplier for quaternion numbers, consistently good results are obtained.

1 INTRODUCTION

In any genetic algorithm (GA) the choice of problem representation is crucial to a successful outcome. The representation must be designed so that gene-interaction (*epistasis*) is kept as low as possible. It is also advantageous to arrange the coding so that *building blocks* may form, so aiding the convergence towards the global optimum.

Finding a suitable representation can be especially difficult for combinatorial optimization problems. These can be thought of as being of two types. The first type involves finding a configuration of entities which maximizes or minimizes some function, whilst satisfying particular constraints (for example, scheduling). The second type involves finding *any* configuration which satisfies the constraints. In the latter situation, typically only a few points in the search space have any usefulness (i.e. fitness)—all the others are of no interest to us (i.e. have zero fitness). An example is the combination to open a safe: either it is right, in which case the door will open, or it is wrong, in which case it will not open. All wrong combinations are equally ineffective at opening the door, whether they have all digits wrong, or all-but-one right.

Such all-or-nothing problems are very difficult to solve, and cannot generally be tackled directly by a GA. Several ideas have been suggested to overcome this difficulty. What is required is to allocate a fitness value to the invalid solutions, in such a way that they will lead the GA towards valid ones. One method is to allocate a fitness in relation to the *degree* by which constraints have been violated (Richardson *et al*, 1989). Alternatively, meaningful *subgoals* can be identified, and fitness allocated according to how many subgoals have been achieved (Cramer, 1985; De Jong & Spears, 1989). Preliminary work by us in the area of algorithm simplification shows that choosing a representation for which these techniques will work can be very difficult.

The difficulties of representation can be illustrated by the simple example of two-dimensional bin-packing. The task is to find a way to pack a number of 2-D objects into a 2-D "bin" of a particular size. (A practical example of this problem is cutting shapes out of standard-sized sheet-metal blanks.) A direct representation method would have each object to be packed represented by a gene which encodes its absolute 2-D position. This representation is, however, unsuitable for several reasons. Most chromosomes would represent arrangements which include overlapping objects, and thus yield invalid solutions. There would also be no way to order the genes to promote building blocks, for two reasons. The first and more trivial reason is that the neighbors of an object, after packing, can vary from chromosome to chromosome. The second reason is that the positions of many (perhaps all) other objects influence what is a good position of one object—not just its neighbors. Changing the position of a single object may require moving every other object in the set. So we are unlikely to find a set of absolute positions for a small group of objects that will increase the fitness of chromosomes it is passed on to. That is, we cannot expect building blocks to form.

For these reasons, such a direct representation scheme would never be used. Instead, better ones have been devised. In the case of bin-packing, scheduling and trav-

elling salesperson tasks, *order-based* representations have been found to be successful (Goldberg, 1985; Davis, 1985; Glover, 1987; Syswerda, 1991). These employ chromosomes where the genes have fixed *values*, but variable *positions* within the chromosome. However, this technique is not applicable to all combinatorial optimization problems—for example, some are more related to *selection* than *ordering*.

We have developed *expansive coding*, a new approach for solving combinatorial problems using GAs. This involves splitting the task into a number of separate sub-problems. By adding extra dimensions to the problem, much of the gene-interaction can be eliminated. The problem space becomes larger when this is done, although the difficulty for the GA is reduced.

We have considered the problem area of arithmetic and logic expression simplification (for the purposes of achieving reduced execution times in software or reduced circuit complexity in hardware). Once the problem representation has been *expanded*, fairly simple local constraints can be applied to each of the sub-problems. With appropriately designed operators, invalid chromosomes can be prevented from ever appearing in the population. Fitness is measured by the extent to which the solutions to the sub-problems (the *sub-solutions*) may be "merged" to form a single, unified, solution. In effect the problem is changed from a combinatorial one to a function optimization one.

In this paper we describe the technique of expansive coding, and show how it can be applied to the design of a multiplier for quaternion numbers. We also make use of approximate fitness function evaluation (Goldberg, 1989) and a 2-dimensional chromosome representation (Cohoon & Paris, 1986) in our solution of this particular problem.

2 EXPANSIVE CODING

The central idea of expansive coding is to split a large, highly epistatic problem into a number of sub-problems, so that even though high epistasis may remain *within* each sub-problem, epistasis *between* sub-problems is lower. Because the regions of high epistasis are localized, they are easier to deal with. Sub-problem validity can be ensured by appropriate coding and operators. This leaves the GA with the greatly simplified task of arranging relatively weakly-interacting sub-solutions to find the overall solution of highest fitness.

Here we outline the basic principles of expansive coding, illustrated by the bin-packing problem described above. (This is not to suggest that this technique would be the best one to use for such a problem.) This problem can be approached by solving a related problem—packing a number of objects in to the *minimum number* of 2-D bins of a particular size. If we find a solution to the new problem which requires only *one* bin, then we have solved the original problem.

First we *split* the task into sub-problems. We put each object in a separate, standard-size bin, so each sub-solution represents an object positioned somewhere in its own bin. The trivial interpretation of any chromosome is therefore that as many bins as objects will be required to pack all the objects. This is obviously a far from optimal solution to the problem of finding the *minimum* number of bins, but the important point is that *all* chromosomes represent valid solutions.

After splitting the task, a *merging* algorithm must be devised to do the following. It must take pairs of sub-solutions, and check to see if both sets of objects from the two bins can be placed, (without changing their positions), into a single bin, without any overlap. If they do not overlap, the two sub-solutions may be "merged", thus reducing the number of bins required by one. After running the merge operation on a chromosome we obtain a count of the number of bins required. The more merges performed, the fewer bins will be required, and the higher the fitness.

The steps involved in designing the coding and the GA can be described more generally as:

- **Splitting.** The problem is split into sub-problems, so that any combination of valid sub-solutions gives a valid overall solution. Sub-problem representations are concatenated together to form a chromosome.
- **Local constraints.** These must be placed on each sub-problem, to ensure that the sub-solutions represented are always valid. They can be enforced either by using a careful coding scheme which makes it impossible to represent invalid sub-solutions, or by using, instead of conventional crossover and mutation, problem-specific operators which always maintain the validity of a sub-problem.
- **Local fitness.** In some tasks any valid sub-solution may be just as fit as any other. In other tasks, however, it may be possible to assign a partial fitness value to each sub-solution in isolation. For example, in a 3-D bin packing problem, we may prefer to place heavier objects at the bottom of bins.
- **Merging algorithm.** The major fitness calculation comes from attempting to merge the sub-solutions into a global solution. Methods for doing this will be problem-specific. The more merging that is successfully carried out, the fewer distinct sub-solutions will remain, and the higher the fitness. The total fitness is computed from some combination (e.g. the weighted sum) of the merge fitness and the local fitnesses.

Even in the absence of local fitness values, sub-solutions which are easy to merge with other sub-solutions will tend to increase in the population. Also, sensible *ordering* of the sub-problems can help promote the creation of building blocks. In these ways, the population converges towards a set of coherent sub-solutions.

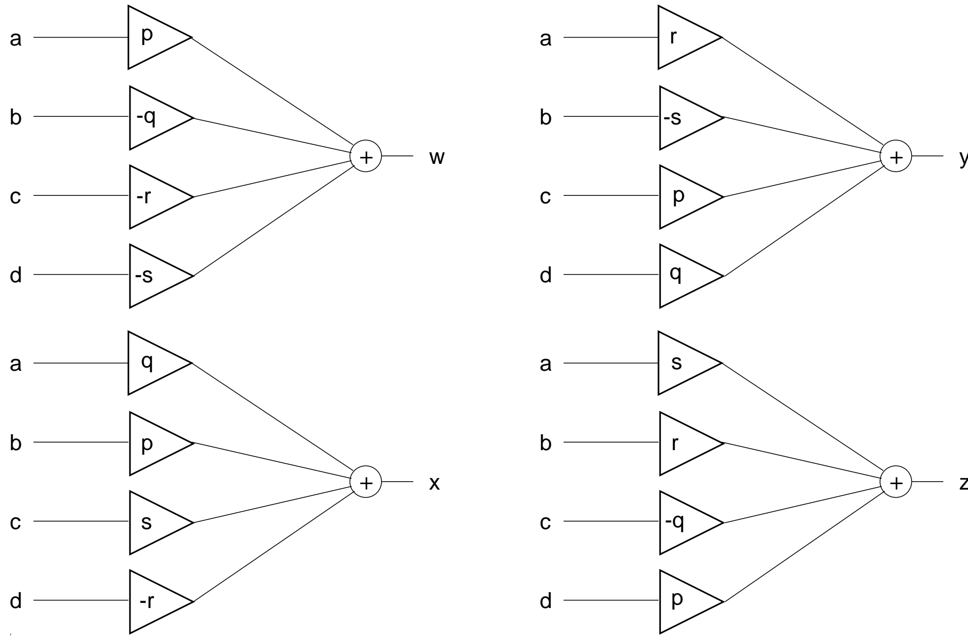


Figure 1: Simple Quaternion Multiplication

2.1 REPRODUCTION OPERATORS

Operators must be designed to maintain the validity of sub-solutions. While such problem-specific operators are being designed, thought may also be given to including sensible heuristics.

To prevent **crossover** from disrupting the validity of any sub-solution, it will often be necessary to restrict crossing sites to lie between sub-problems. This means that the effective number of symbols in the chromosome will be equal to the number of sub-problems. While the number of symbols is reduced in this way, the cardinality of each symbol is increased. An alternative method is to allow crossover at any site, and then repair the sub-solution, if necessary, to make it valid again.

Similarly, **mutation** operators will normally work on one symbol (i.e. a whole sub-problem) at a time, rather than on one bit at a time.

3 THE QUATERNION MULTIPLIER PROBLEM

The quaternion multiplier problem was chosen as a simple, yet challenging task in the field of arithmetic algorithm optimization. Quaternion numbers have various applications in three-dimensional geometry (Funda, Taylor & Paul, 1990; Shoemake, 1985), including performing object rotation. In two-dimensions, rotation can be accomplished by representing each of the points of an object as a complex number. If all these are then multiplied by another complex number, the result is a rotated set of points. In three-dimensions, a similar effect can be achieved by multiplying appropriate quaternions.

The trivial algorithm for multiplying two quaternion numbers requires sixteen real-number multiplications. If an algorithm can be devised which uses fewer multiplications, implementations can be improved.

3.1 QUATERNION NUMBERS

Quaternions (Brand, 1947; Hamilton, 1899; Martin, 1983) have *four* components, and may be written as $q \equiv a + \mathrm{i}b + \mathrm{j}c + \mathrm{k}d$, where a, b, c and d are real numbers, and i, j and k are the three *quaternion operators*. Each of these is analogous to the complex number operator, i. If two quaternions, $q_1 \equiv (a + \mathrm{i}b + \mathrm{j}c + \mathrm{k}d)$ and $q_2 \equiv (p + \mathrm{i}q + \mathrm{j}r + \mathrm{k}s)$ are multiplied to give $(w + \mathrm{i}x + \mathrm{j}y + \mathrm{k}z)$, then the components to be computed are:

$$w = (ap - bq - cr - ds), \quad (1)$$
$$x = (aq + bp + cs - dr), \quad (2)$$
$$y = (ar - bs + cp + dq), \quad (3)$$
$$z = (as + br - cq + dp). \quad (4)$$

Note that, in general, $q_1 q_2 \neq q_2 q_1$.

3.2 FORMULATING THE PROBLEM

We may view the task as one of mapping four input variables, (a, b, c, d), to four output variables, (w, x, y, z) via sixteen multipliers, with (p, q, r, s) as parameters. For example, from Eqn.(1), the mapping $a \rightarrow w$, is achieved by multiplying by the value p. This can be represented as a linear signal flow graph, as shown in Figure 1 (where triangles represent multiplication, and circles represent addition).

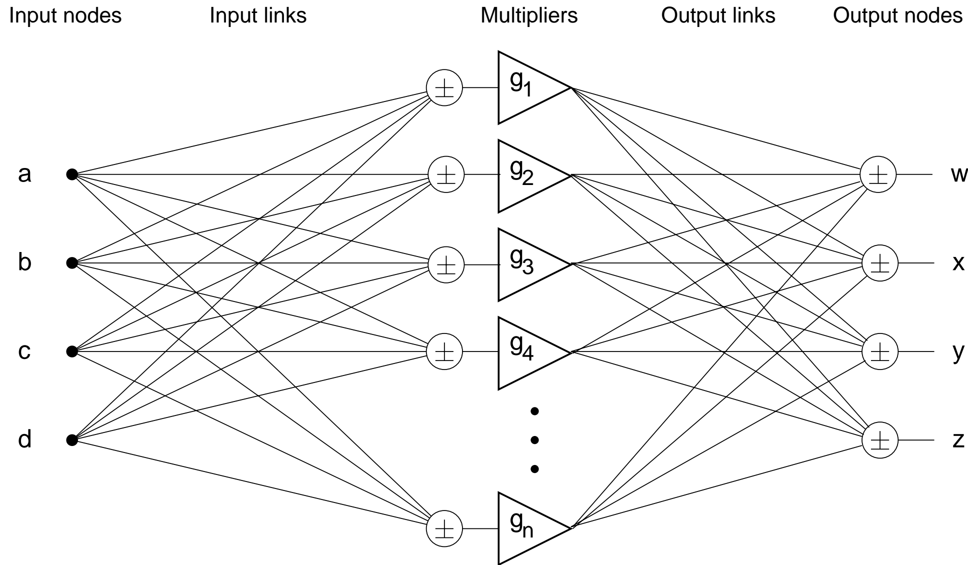


Figure 2: Generic Circuit

In Figure 1, each multiplier is used exactly once. However, because of redundancy, it is possible to *reuse* multipliers, so that inputs and/or outputs can *share* multipliers, reducing the total number needed. A generic circuit arrangement for this sharing scheme is shown in Figure 2. Here there are n multipliers, which multiply by factors $g_1, g_2, g_3 \ldots\ g_n$. Each of these factors, or *gains*, g_i, can be specified by four coefficients, $h_p(g_i), h_q(g_i), h_r(g_i)$ and $h_s(g_i)$, such that $g_i = h_p(g_i)p + h_q(g_i)q + h_r(g_i)r + h_s(g_i)s$. The input of each multiplier is connected via a set of *input links* to each of the input nodes. Each input link represents a *potential* connection between an input node, and an adder/subtracter unit at the input to a multiplier. Each input link therefore represents a connection with a gain in $\{-1, 0, 1\}$. Similarly, the output of each multiplier is connected to each output node via an *output link*, with a gain in $\{-1, 0, 1\}$.

With this arrangement, it is possible for several inputs to share a common multiplier, and also for the output of one multiplier to be shared among several outputs. By a suitable choice of input and output link gains, and multiplier gains, it is possible to represent a broad class of input/output transfer functions, a subset of which will correctly perform quaternion multiplication. The lower the value of n, the higher the fitness of the circuit. The problem to solve is, what is the minimum value of n, and what gain values are required to achieve this?

3.3 APPLYING A GENETIC ALGORITHM

Clearly this task is difficult for a GA. If we were to use a direct coding scheme, in which values for input link gains, output link gains, and multiplier gains were all simply coded into the chromosome, there would be a very high degree of epistasis. Changing any multiplier gain value can, potentially, alter *all* of the input/output transfer functions. Changing just one link gain value can make a valid solution invalid, and to repair this, many other link gains and multiplier gains would need to be changed, with these changes, in turn, requiring further changes. So interdependent are the gains, it is impossible to improve a reasonably good chromosome by making small changes to it. No building blocks could ever form, since the fitness of any sub-group of genes is highly dependent on the values of most of the other genes.

To overcome these problems our expansive coding technique is used, as detailed below.

3.3.1 Splitting

The problem is split into sixteen sub-problems, one for each input/output transfer function. Each sub-problem has S multipliers of its own with which to fulfil the correct transfer function. As far as the representation is concerned, these multipliers are *not* shared with any other sub-problems. For the $a \rightarrow w$ transfer function, shown in Figure 3, the multipliers have gains $g_{aw1}, g_{aw2}, g_{aw3} \ldots\ g_{awS}$, and the mapping is therefore:

$$w_a = a \sum_{i=1}^{S} g_{awi} \tag{5}$$

The total output is given by $w = w_a + w_b + w_c + w_d$. The other 15 mappings are treated in the same way, which means the chromosome must hold information for $16 \times S$ multipliers.

Each multiplier in Figure 3 can be represented by four gain coefficients, h_p, h_q, h_r and h_s (Figure 5(c)). For simplicity, we assume that each of these will be in $\{-1, 0, 1\}$. Therefore, two bits are needed for each coefficient, or eight bits per multiplier (Figure 5(d)). Input and output link gains do not need to be represented explicitly. An input or output

link gain of zero can is equivalent to a multiplier gain of zero. Similarly, link gains of ± 1 can be absorbed by changing the overall sign of the multiplier gain. The required link gain values can be deduced during the merging phase (Section 3.3.4).

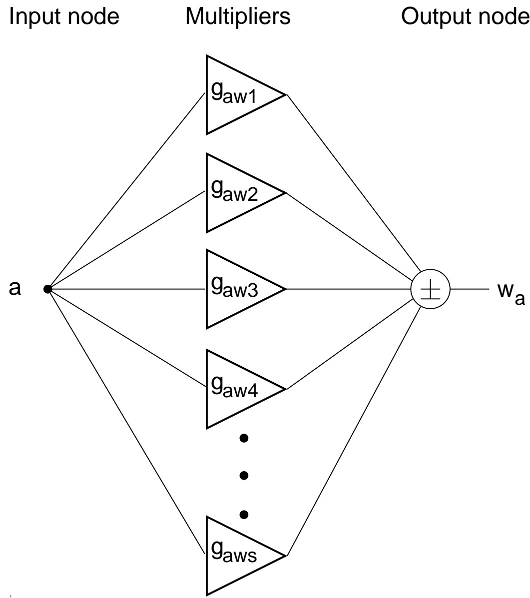


Figure 3: Circuit For One Sub-Problem

3.3.2 Local Constraints

Having split the problem into sixteen sub-problems, we now consider what local constraints should be applied to each of these. For a sub-problem mapping input $u \in \{a, b, c, d\}$ to output $v \in \{w, x, y, z\}$, the transfer function is

$$\begin{aligned} v_u \; = \; & p \sum_{i=1}^{S} h_p(g_{uvi}) + q \sum_{i=1}^{S} h_q(g_{uvi}) + \\ & r \sum_{i=1}^{S} h_r(g_{uvi}) + s \sum_{i=1}^{S} h_s(g_{uvi}) \end{aligned} \quad (6)$$

or, putting $H(u, v, p) \equiv \sum_{i=1}^{S} h_p(g_{uvi})$, etc. this becomes

$$\begin{aligned} v_u \; = \; & pH(u, v, p) + qH(u, v, q) + \\ & rH(u, v, r) + sH(u, v, s) \end{aligned} \quad (7)$$

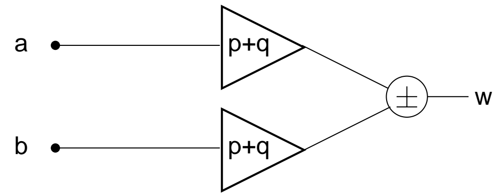
Eqns. (1) to (4) give the total gains required in each of the sixteen cases. For example, for the $a \rightarrow w$ mapping, we require $H(a, w, p) = 1, \; H(a, w, q) = 0, \; H(a, w, r) = 0, \; H(a, w, s) = 0$. This means that within each sub-problem, the sums of the gain coefficients, h_p, h_q, h_r and h_s, must be maintained at specific values. To achieve this, chromosomes in the initial population are set up with valid sums, and the operators used are designed to maintain this validity (Section 3.3.6).

3.3.3 Local Fitness

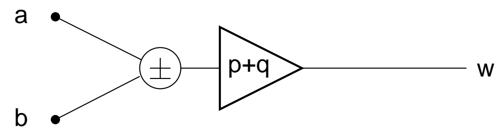
In each sub-problem, any set of gain values conforming to Eqns. (1) to (4) will be valid and, in isolation, each is equally good. Thus, local fitness values are not used.

3.3.4 Merging Algorithm

The merging algorithm must bring together multipliers which have equal gains and compatible input/output connections. For example, suppose two multipliers each have a gain of $(p + q)$, (that is, $h_p = 1$, $h_q = 1$, $h_r = 0$ and $h_s = 0$). If one has an input connection to a, the other an input connection to b, and both have output connections to w, then they may be merged. This would give a single multiplier with gain $(p + q)$, its output connected to w and its input the sum (or difference) of a and b. This is shown in Figure 4.



(a) Before Merging



(b) After Merging

Figure 4: Illustration Of Merging

In general, there will be several different ways of merging a set of multipliers, so to find the optimum merging pattern, an exhaustive search must be done. This is a slow process, but fortunately we can use an *approximate* fitness evaluation method (Goldberg, 1989, pp 138, 206). A *greedy algorithm* finds an optimal merging pattern in most cases, so our GA uses this to determine an approximate fitness for each chromosome during a run. Only when the GA has converged, and a solution found is the exhaustive search algorithm used to determine the exact fitness.

After merging, the number of distinct multipliers with non-zero gain is taken as the fitness value. The GA must minimize this value.

3.3.5 Chromosome Organization

Careful organization of the chromosome will allow building blocks to form. A set of sub-solutions is well adapted if many of their multipliers can be merged. If they are also close together on the chromosome, they can form a building block. Merging can only take place between multipliers which share common input or output connections. So, a chromosome organization is needed where sub-problems are close together if they share common inputs, *or* if they share common outputs. This cannot be achieved with a conventional 1-dimensional chromosome, but is easily arranged on a 2-dimensional chromosome.

The most natural organization is therefore a 4×4 array of the sub-problems, (Figure 5(a)), where each sub-problem is represented by S multipliers (Figure 5(b)). Each *row* of the array contains sub-problems relating to the same output node. Conversely, each *column* contains sub-problems relating to the same input node.

Since merging can only take place between multipliers in the same row or column, it is possible for building blocks to form as coherent rows or columns evolve.

3.3.6 Operators

To avoid creating invalid chromosomes, crossover points are only allowed at sub-problem boundaries. Three different crossover operators are used. *Whole row* and *whole column* crossover are 2-D projections of normal 2-point crossover. Two cut points are chosen, and complete rows (or columns) are swapped over between the parents to produce two offspring. In *square patch* crossover, the chromosome is treated as a torus, and two row cut points and two column cut points are chosen, defining a square patch. This is then swapped from one parent to the other, giving two offspring.

Two mutation operators are used. Both first select a sub-problem to work on. A multiplier is chosen, and one or more of its gain coefficients is incremented or decremented. A record is kept of the alteration made. To regain the validity of the sub-solution, another multiplier is chosen, and compensating decrements or increments are made to its gain coefficients. As pointed out in Section 3.3.2, the effect is to maintain the sums of the gain coefficients, h_p, h_q, h_r and h_s, at specific values. Occasionally it proves to be impossible to make the required change to a selected multiplier, (because the change would take a gain coefficient outside the allowed range -1 to $+1$), in which case another multiplier is chosen, and that changed instead.

The two mutation operators differ in how the alteration is made to the first multiplier chosen. The *swap component* mutation operator increments or decrements one of the multiplier's gain coefficients by 1. The *zeroize gain* operator sets the multiplier gain to zero. The latter operator will tend to reduce the number of multipliers in use in the chromosome, and therefore create a pressure to simplify the circuit.

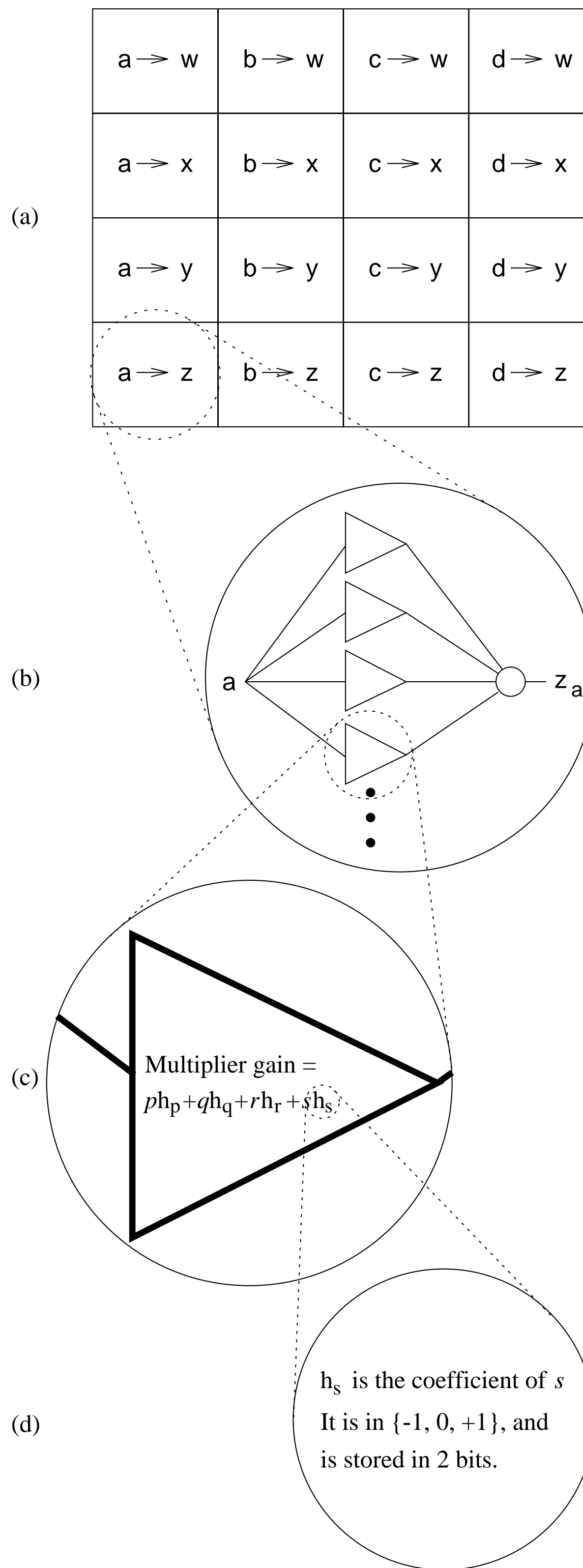


Figure 5: Chromosome Organization:
(a) 4×4 sub-problem array; (b) a single sub-problem; (c) a single multiplier; (d) a single gain coefficient.

4 RESULTS

In preliminary work, we investigated an analogous, but much simpler, minimization task—that of a multiplier for two complex numbers. We tried direct representation methods of the type described earlier. Despite incorporating techniques which others had found useful in combinatorial tasks (Cramer, 1985; De Jong & Spears, 1989; Richardson *et al*, 1989), the optimum three multiplier solution was not found once during 500 runs. After adopting the expansive coding scheme, however, optimum solutions were found in 85% of runs, with typically 2030 function evaluations. When addressing the harder quaternion multiplier task, we therefore used only the expansive scheme, as the direct method clearly would not work.

The expansive representation scheme was tested using a generational replacement GA, based on Goldberg's SGA (Goldberg, 1989), implemented in Pop-11. Two methods of fitness scaling were tried: *linear*, with truncation to zero for individuals whose expected reproductive count would have been below zero (similar to *sigma truncation* (Goldberg, 1989, p124)); and *ranked*, with linearly-allocated expected offspring values. In both cases, the number of reproductive opportunities for the most fit individual in each generation was 2.0. We used a crossover probability of 0.8, and mutation probability of 0.064 per sub-problem. (This is not as high a mutation rate as it might seem, since there are only 16 symbols in the chromosome. There is a 35% probability of zero mutations, and an average of 1.0 mutations per chromosome, for $S = 8$.) To produce a pair of offspring, one of the crossover operators was chosen and applied, then one of the mutation operators was chosen, and applied to each child. The probabilities of use of each operator were held fixed during each run. A run was terminated when the population average fitness, measured over a moving window of a fixed number of generations, stopped increasing.

For crossover, we used an equal probability for each of the three operators in most runs, although a few runs with only square patch crossover seemed to perform equally well. For mutation, the probabilities were weighted 10:3 in favour of swap component mutation. This is because we expect the swap component operator to cause exploration of the search space, and the zeroize gain operator to cause exploitation, and we want to encourage exploration.

When we began our trials we were aware of a solution to the problem which required only 12 multipliers. We initially chose S to have a value of 6, as a compromise between producing too large a chromosome, and restricting the type of solutions it was possible for the GA to find. Within 100 runs, a solution with only 10 multipliers was found. (We subsequently discovered that a 10-multiplier solution is already known (de Groote, 1975).) The 10-multiplier solution requires 6 multipliers connected to three out of the four inputs. By coincidence, this is the upper limit we had chosen for the number of multipliers used per sub-problem. To be sure that our GA could work even without lucky initial guesses, further runs had $S = 8$. This gives a chromosome of 1024 bits, and a search space of approximately 10^{150} valid chromosomes. The results for different population sizes, averaged over 100 runs, are summarized in Table 1.

The effectiveness of the expansive coding technique is clearly demonstrated. Every run came up with a solution better than the obvious 16-multiplier arrangement. With larger populations, very good 10-multiplier solutions were found regularly. The best population size out of those tried appears to be 200, and linear truncated fitness scaling performs better than ranking.

5 CONCLUSIONS

At first sight, expansive coding seems counter-intuitive, since it makes the search space much larger. The problem is, however, made simpler, since the interaction (epistasis) between the elements which the GA has to manipulate (the sub-problems) is reduced.

The effect of expansive coding is to convert a combinatorial optimization problem into a function optimization one. In the case of the quaternion multiplier, the original problem could be expressed as: *"All chromosomes represent circuits with 10 multipliers, find one which also represents the correct gains for a quaternion multiplier."* After expansion, however, the problem becomes: *"All chromosomes represent a quaternion multiplier, find one which has the minimum number of multipliers."*

With appropriate representation and operators, the inherent complexity of a problem may be shifted, so that although the fitness decoding function becomes more complicated, the GA finds the task easier. In theory, this allows any problem to be made trivially easy to solve, from the point of view of the GA (Vose & Liepins, 1991). GAs are good for problems of intermediate epistasis (Davidor, 1990). On

Table 1: Percentage Of Runs Finding A 10-Multiplier Solution

Scaling	Pop. size	Window size	Evals./run	Worst solution	% Success
ranking	100	30	14000	15 mults.	1
ranking	200	60	55800	13 mults.	24
ranking	400	100	155000	13 mults.	32
linear	100	30	16100	15 mults.	2
linear	200	60	56600	13 mults.	33
linear	400	100	147700	13 mults.	58

highly epistatic problems, therefore, a suitable representation and operator set must be found which sufficiently reduces the epistasis. The expansive coding technique is one such approach. Complexity, in terms of epistasis in the original problem, is traded for complexity in terms of an increased chromosome size, a more complicated fitness function, and the need for problem-specific operators. We have therefore split one large dose of complexity into three smaller doses—a divide-and-conquer approach.

We are still investigating issues such as heuristics, parent selection methods and optimum operator probabilities. One avenue for further research is dynamic operator probabilities (Davis, 1991).

Our practical application of this method shows that it can be highly effective. The area of algorithm and circuit design does not require absolutely optimal solutions at great speed; any technique which can find designs better than existing ones is useful. We are encouraged by these results, and intend to apply this technique to more interesting and practical problems of algorithm design and circuit optimization.

We also feel that this technique is sufficiently general to be useful beyond this area.

References

Brand, L. (1947). *Vector and Tensor Analysis*. Wiley.

Cohoon, J. P. and Paris, W. D. (1986). Genetic placement. In *Proc. IEEE International Conference on Computer Aided Design*.

Cramer, N. L. (1985). A representation for the adaptive generation of simple sequential programs. In J. J. Grefenstette, editor, *Proceedings of the First International Conference on Genetic Algorithms*, 183–187. Lawrence Erlbaum Associates.

Davidor, Y. (1990). Epistasis variance: Suitability of a representation to genetic algorithms. *Complex Systems*, 4:369–383.

Davis, L. (1985). Applying adaptive algorithms to epistatic domains. In *9th Int Joint Conf on AI*, pages 162–164.

Davis, L. (1991). *Handbook of Genetic Algorithms*. Van Nostrand Reinhold.

de Groote, H. F. (1975). On the complexity of quaternion multiplication. *Information Processing Letters*, 3(6):177–179.

De Jong, K. A. and Spears, W. M. (1989). Using genetic algorithms to solve NP-complete problems. In J. D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, 124–132. Morgan Kaufmann.

Funda, J., Taylor, R. H., and Paul, R. P. (1990). On homogeneous transforms, quaternions, and computational efficiency. *IEEE Trans. on Robotics and Automation*, 6(3):382–388.

Glover, D. E. (1987). Solving a complex keyboard configuration problem through generalized adaptive search. In L. Davis, editor, *Genetic Algorithms and Simulated Annealing*, 12–31. Pitman.

Goldberg, D. E. (1985). Alleles, loci, and the TSP. In J. J. Grefenstette, editor, *Proceedings of the First International Conference on Genetic Algorithms*, 154–159. Lawrence Erlbaum Associates.

Goldberg, D. E. (1989). *Genetic Algorithms in search, optimization and machine learning*. Addison-Wesley.

Hamilton, W. R. (1899). *Elements of Quaternions*. Cambridge University Press.

Martin, R. R. (1983). Rotation by quaternions. *Mathematical Spectrum*, 17(2):42–48.

Richardson, J. T., Palmer, M. R., Liepins, G. E. and Hillard, M. R. (1989). Some guidelines for genetic algorithms with penalty functions. In J. D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, 191–197. Morgan Kaufmann.

Shoemake, K. (1985). Animating rotation with quaternion curves. *Computer Graphics*, 19(3):245–254.

Syswerda, G. (1991). Schedule optimization using genetic algorithms. In L. Davis, editor, *Handbook of Genetic Algorithms*, 332–349. Van Nostrand Reinhold.

Vose, M. and Liepins, G. E. (1991). Schema disruption. In R. K. Belew and L. B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, 237–242. Morgan Kaufmann.