Restricted Evaluation Genetic Algorithms with Tabu Search for Optimising Boolean Functions as Multi-level AND-EXOR Networks.

Julian F. Miller and Peter Thomson
Department of Computer Studies, Napier University,
219 Colinton Road, Edinburgh, EH14 1DJ, UK
email: j.miller@napier.ac.uk

Abstract:

In GAs applied to engineering problems (in our case, the optimisation of logic circuits) the fitness function is usually complex and the fitness evaluation is time consuming. The run time is therefore a major consideration when designing a GA for optimisation, thus a look-up table for fitness evaluation is desirable. As a consequence, it is appropriate to limit the number of different chromosome fitness evaluations that any particular run of the GA will be allowed to examine. In this situation the user is uninterested in the number of generations required. It is necessary in this approach to guarantee the users that they will be able to find a good and reliable problem solution within the limited number of evaluations, and hence time available. We refer to this type of GA as a restricted evaluation GA. In this paper we suggest a number of hybrid algorithms which combine a GA with a neighbourhood search (TABU) technique to provide this performance and reliability. The effectiveness of each of these methods is compared and contrasted, and underlying principles are suggested as to why these techniques might prove to be useful in these types of problem.

1. Introduction.

The complexity of logic circuits can often be markedly reduced by applying some kind of optimisation algorithm. The type of gates used can have a large effect on compactness of the final circuit. Traditionally logic circuits have been built out of Boolean components such as the NOT, AND and OR gates. The rules for manipulating representations with these operations obey Boolean Algebra. However any logic function may also be built using NOT, AND and exclusive-OR gates (EXOR), the algebra associated with this is known as Reed-Muller algebra. The simplification of logic functions in this domain is much more difficult as the algebraic rules allow many more representations of the same logic function, however this often allows a more compact representations.

Until recently EXOR gates were not favoured for logic implementation as they were expensive in silicon terms as compared with OR gates. However many modern circuit devices such as field-programmable logic arrays (FPGAs) contain large arrays of uncommitted logic and there is no cost penalty associated with the use of EXOR gates. Optimising circuit designs for these devices is still very important as the larger arrays are still fairly expensive and anyway small designs are easier to test and can be placed and routed more easily. Implementing Boolean functions as AND-EXOR networks has been shown to produce, on average [19], more compact circuits. In addition it has recently been shown [5] that multi-level AND-EXOR networks are more testable than their Boolean counterparts. From this it would appear that a logic designer would, at the very least, wish to have the option of expressing his design in this form, and this in turn implies the need for reliable synthesis tools for finding low-cost AND-EXOR representations.

The major problem with finding AND-EXOR representations for Boolean functions, is the large number of possible expansions which may be generated. It requires a powerful search technique to be able to find a good solution amongst the many possibilities. Recently,
the authors of this paper, and others, have developed, with some success, software using Genetic Algorithms with small populations to address this problem [7][14][15][21].

It has been our experience that our low population GAs converged very rapidly to a good solution. This observation led us to wonder whether the GAs was perhaps spending too much time in a not very effective near-convergence state where the genetic diversity was very low and where mutation was the only effective operator. This concern prompted us to introduce a neighbourhood search technique (TABU search) which could be switched in to replace the GA when this point of near convergence was reached. Following on from this, we experimented with various ways in which to introduce the neighbourhood search in association with the existing GA. The three approaches which we investigated were: (i) Appended local search, where the TABU search is switched in after the GA has converged, (ii) Embedded local search, where the TABU search is switched in after a variable number of GA generations and (iii) Dynamic local search, where the TABU search is switched in after the genetic diversity of a GA generation has fallen below some pre-determined level. We present comparative results for each of these techniques operating upon standard industrial benchmarks for combinational logic circuits.

The method of comparison is based upon the idea of fixing, in advance, the number of different chromosome fitness evaluations that the combined algorithms will be required to process. This is important because it not only limits the total execution time for the algorithm - a significant consideration in engineering synthesis - but it also compares each algorithm’s performance for a fixed percentage of the search space.

The remainder of this paper will give a presentation of the problem, a description of the modified TABU search used to explore neighbourhoods, a description of the three hybrid algorithms and, finally, a set of experimental results which demonstrate the relative effectiveness of each method.

2. The Problem Specification.

Every Boolean function describes a logical proposition as a combination of the set of AND, inclusive-OR and NOT operators acting upon a set of input variables. The most common form of expression is the so called sum-of-products (SOP) form in which groups of input variables are logically ANDed as product terms, and these are then finally combined into a sum by ORing each product together. In electronic design, it is possible after having derived the desired combinational logic expression which describes a required functionality, to physically implement this as an electronic circuit. It is then desirable to manipulate the design so that if possible the minimum number of physical components is actually used. An alternative approach to the problem of finding a simple representation of the circuit is to convert the inclusive-OR SOP form to the exclusive-OR (or EXOR) representation, and to attempt simplification (by cancellation of terms) using the different algebraic rules which are attendant upon this method. The only drawback with this method is that the number of possible expansions that may be generated is much larger than the corresponding Boolean set, and, for a long time, this made this approach unattractive to designers. As a result of this many early approaches employed specialised subsets of general AND-EXOR expressions with fewer members such as the fixed-polarity form, mixed-polarity etc. [13][19]. However, more recently, modern search heuristics such as GAs combined with decision based approaches have been used to find compact representations taken from the general sets of AND-EXOR [6][21].
To illustrate the difficulty of search space size in the AND-EXOR method, consider
the small example of the Boolean function:

\[ \overline{x_2 \cdot \overline{x_1} \cdot x_0} + \overline{x_3 \cdot x_1 \cdot x_0} \]

alternatively, using AND-EXOR form, this may be written as:

\[ x_3 \cdot x_1 \cdot x_0 \oplus \overline{x_3 \cdot x_1 \cdot x_0} \]

The inclusive-OR operator (+) can be replaced by the exclusive-OR operator (⊕)
because the two products are mutually exclusive in logical terms. Using the identity
\[ \overline{x_n} = x_n \oplus 1 \]
the number of possible AND-EXOR expansions is now, for this example, \(2^n\), and
\(2^{n \times m}\) in general, where \(n\) is the number of input variables and \(m\) is the number of product
terms. Even for a trivial function \((n < 10)\) the search space is potentially very large.

In order to alleviate some of the effects of this particular drawback, and to take
advantage of the benefits of multi-level design, we have modelled the problem not as a two-
level representation (as in SOP), but as an algebraic EXOR ternary decision diagram
(AETDD). A decision diagram (DD) [1][3][4] is a directed acyclic graph based description of
a logic function. Associated with each vertex is an input variable of the logic function under
consideration, where the complexity of the DD is strongly dependent upon the order in which
these node variables are presented. If the node variables are chosen to be fixed throughout
the same level in the decision tree and once used in a particular level cannot be used again, then
the decision diagram is said to be \textit{ordered}. This gives rise to the so called variable ordering
problem and there are many heuristics which have been devised (largely in the Boolean
domain) to determine an ordering which simplifies the decision tree
[2][3][8][9][10][12][17][22]. Modelling the optimisation problem as a decision tree reduces
the number of genes required for a chromosome to be merely the same as the number of input
variables. This leads to a greatly simplified data representation for a GA than attempting to
model each individual variable expansion, as in the two-level case, but with the expense of a
more complex fitness function.

This method we have devised carries out \textit{algebraic} simplification of a four node-two
variable exclusive-OR ternary decision diagram (ETDD) to produce an economical network in
which the nodes may be any of the four fundamental types: Ternary, Shannon, Positive Davio
and Negative Davio [20].

The AETDD differs from a ETDD in that \textit{algebraic} rules are applied to a two-variable
ETDD to produce a simplified sub-tree:
The ETDD outputs for the four node sub-tree in Fig. 1 are given by,

\[
\begin{align*}
    f_0 &= f_{00} \times x_{n-1} \oplus f_{01} \times x_{n-1} \oplus f_{02} \\
    f_1 &= f_{10} \times x_{n-1} \oplus f_{11} \times x_{n-1} \oplus f_{12} \\
    f_2 &= f_{20} \times x_{n-1} \oplus f_{21} \times x_{n-1} \oplus f_{22} \\
\end{align*}
\]

where \( f_{ij} = f_{ij}(x_0, \ldots, x_{n-2}) \) and

\[
f = f_0 \times x_n \oplus f_1 \times x_n \oplus f_2
\]

It is convenient to define the ternary vector \( f \) with respect to variables \( x_{n-1} \) and \( x_n \) as

\[
f = (f_{00}, f_{01}, f_{02}, f_{10}, f_{11}, f_{12}, f_{20}, f_{21}, f_{22})
\]

If some of the \( f_{ij} \) are equal to one another (equality conditions) (5) can be expressed in a simpler form:

\[
\begin{align*}
    f_0 &= f_{10} & f &= (0, f_{01}, f_{02}, 0, f_{11}, f_{12}, f_{20} \oplus f_{00}, f_{21}, f_{22}) \\
    f_0 &= f_{11} & f &= (0, f_{01}, f_{02}, f_{10}, 0, f_{12} \oplus f_{00}, f_{20} \oplus f_{00}, f_{21}, f_{22}) \\
    f_0 &= f_{20} & f &= (0, f_{01}, f_{02}, f_{10} \oplus f_{00}, f_{11}, f_{12}, 0, f_{21}, f_{22}) \\
    f_0 &= f_{10} & f &= (f_{00}, 0, f_{02}, 0, f_{11}, f_{12} \oplus f_{01}, f_{20}, f_{21} \oplus f_{01}, f_{22}) \\
    f_0 &= f_{11} & f &= (f_{00}, 0, f_{02}, f_{10}, 0, f_{12}, f_{20}, f_{21} \oplus f_{01}, f_{22}) \\
    f_0 &= f_{21} & f &= (f_{00}, 0, f_{02}, f_{10}, f_{11} \oplus f_{01}, f_{12}, f_{20}, 0, f_{22}) \\
    f_0 &= f_{12} & f &= (f_{00}, f_{01}, 0, f_{10}, f_{11}, 0, f_{20}, f_{21}, f_{22} \oplus f_{02}) \\
    f_0 &= f_{22} & f &= (f_{00}, f_{01}, 0, f_{10}, f_{11}, f_{12} \oplus f_{02}, f_{20}, f_{21}, 0) \\
    f_0 &= f_{20} & f &= (f_{00} \oplus f_{10}, f_{01}, f_{02}, 0, f_{11}, f_{12}, 0, f_{21}, f_{22})
\end{align*}
\]
\[ f_{11} = f_{22} \quad f = (f_{00}, f_{01} \oplus f_{11}, f_{02}, f_{10}, 0, f_{12}, f_{20}, 0, f_{22}) \]  

\[ f_{12} = f_{22} \quad f = (f_{00}, f_{01}, f_{02} \oplus f_{12}, f_{10}, f_{11}, 0, f_{20}, f_{21}, 0) \]  

There are many more equations resulting from equality of \( f_{ij} \) inputs however we list only those which result in obvious simplification. Thus the AETDD is a ETDD which has been simplified using the algebraic rules defined above. Note that equality conditions may be satisfied as a result of applying a simplification rule e.g. if \( f_{00} = f_{20} \) and \( f_{01} = f_{10} \oplus f_{00} \) in this case (9) may be applied but only after (8) has been applied. This implies a ‘knock-on’ effect which may not happen often but which may lead to considerable further simplification. It is important when implementing an algorithm which employs the above rules to carefully take account of this knock-on effect. If the algorithm tested for an equality condition and then immediately applied the simplification, the new ternary vector may have been altered so that no more equality conditions exist. This type of algorithm produces a strongly non-commutative effect and we have previously devised a GA to tackle this second ordering problem [14]. When a number of equality conditions occur in the same ternary vector a unique simplified vector results. The number of possible combinations of equality conditions is quite large. To take into account a number of equality conditions we first replaced all expressions in the ternary vector by symbols. All appropriate simplification rules were then applied

\[ Z = P_0 \cdot x_1 + \overline{x}_2 \cdot P_1 \]
\[ P_0 = x_3 + \overline{x}_2 \]
\[ P_1 = x_3 + \overline{x}_0 \]

\( \text{Fig. 2. Example circuit specification simplified using AETDD method and variable ordering 1 3 0 2.} \)
An example of a complete ternary decision diagram which has produced a simple multi-level representation of an input function is seen in Fig. 2. The logic function specification (.pla file) is seen in the upper left corner. First the input field is re-arranged (column-wise) according to the variable ordering 1 3 0 2. Then the products are sorted in ternary order. At this stage the new products are loaded onto the ternary decision diagram. If any non-trivial four node sub-trees are present (see the sub-tree enclosed by a rectangle) then the symbolic simplification rules (6)-(16) are applied. The final simplified function is shown in multi-level form at the bottom right section of the figure.

3. TABU Search and the GA.

TABU Search (TS) is a local search technique devised by Glover [11]. It requires the definition of a neighbourhood. The neighbourhood of an ordered set of numbers \( r = \{ r_i \} \) is defined as being the set of all permutations of \( \{ r_i \} \) where each permutation differs from \( r \) by a single pairwise swap. The TS described here uses a more restricted neighbourhood function requiring that the pairwise swaps are all possible pairwise combinations of the first five variables. This is chosen because this set of variables represents the topmost part of the DD, and the ordering of these variables is most influential in reducing the overall tree complexity. This gives a neighbourhood size of 10, and leads to a local search which explores a coarse neighbourhood. In the very low number of evaluations limit that we impose, it is not feasible to explore the neighbourhood more thoroughly.

TS generates from a random variable ordering (generator) a set of neighbours, the best of which \( c_b \) is chosen as a new generator provided that it is not already on the TABU list \( T \), whilst the previous generator is appended to the TABU list. If \( c_b \) is already present on list \( T \), then the second best is chosen as the generator and so on. Other measure have to be taken if all the neighbours of a particular generator are already present on \( T \). In our investigations, this did not happen. We have found that the performance of TS depends markedly upon the quality of the initial generator, as will be seen in the results section.

The GA which is used in concert with the TS uses an ordinal list representation of the DD variable ordering. A variable ordering \( \{ v_i \} \) is obtained from an ordinal chromosome \( \{ c_i \} \) where \( i=1 \) to \( n \), in the following way: \( v_i \) is the \( c_i \)th item chosen from \( V = \{ v_i \} \) where \( k=1 \) to \( i-1 \) and \( V = \{ 0, \ldots, n-1 \} \), the set of input variables. This representation avoids the generation of the invalid solutions which would inevitably result if a direct representation of the variable ordering was used.

The GA uses the traditional single-point crossover and random mutation operators. Parent selection is accomplished by a modified form of size 2 tournament selection, whereby the best of a pair of randomly chosen chromosomes is accepted as a parent with a probability determined by what we refer to as a tournament discriminator (td). We found that employing straightforward tournament selection led to rapid convergence, and so we introduced the tournament discriminator as a way of slowing this process. If the tournament discriminator is chosen to be 0.5, then there is no preferential selection, while if we chose this value to be 1.0, then we restore the standard tournament method.


The essential motivation of this work is the investigation of hybrid algorithms which employ some form of local search in order to extend the effectiveness of the GA. We found that in the
AND-EXOR optimisation problem, the GA spent a significant percentage of the overall execution time exploring very small regions of the search space (at or near convergence). In other words, the GA was very effective at exploring the search space initially, but soon became confined to what may be considered a good neighbourhood. We believed, therefore, that at this point, it might be useful to introduce a local search technique.

We investigated three ways of combining a GA with TABU search. The first was an Appended Local Search (ALS) where we simply carried out a TABU search using the best variable ordering found by the GA. We fixed the total number of different variable orderings to be looked at in advance and varied the proportions carried out in the GA and TS phases. The drawback of this approach is that the best way to divide the number of allowed evaluations between the GA and TS components had to be determined experimentally. We give results on this in the next section. The effectiveness of this approach depends on whether the GA terminates with a variable ordering in the relatively close neighbourhood of an optimum.

A second scheme we tried was Embedded Local Search (ELS) where we allowed the GA to run through a fixed number of generations and then carried out one iteration of a TS. This scheme more closely fits the memetic algorithm [18]. The best variable ordering was then re-encoded as a chromosome and inserted back into the original GA population. Then the GA was run again for a few more generations. This process was iterated until the required number of evaluations was achieved. We varied the number of generations before the local search iteration to try to ascertain whether there was an ideal ratio of GA evaluations to TS evaluations. One difficulty associated with Embedded Local Search was that it was impossible to ensure that all the allotted TS evaluations were carried out during the alternating phases of GA followed by TS. This is because elitism was employed in the GA so that, if no change had occurred in the best variable ordering between TS phases, then the next TS phase was redundant since it would have carried out an identical search to the previous one. Thus TS was only carried out if the best variable ordering had changed in the GA phase. This meant that we had to sometimes carry out some appended local search to ensure that the allotted number of TS variable orderings were evaluated.

The final scheme investigated was what we have termed Dynamic Local Search (DLS). In this method we initiated an Appended Local Search when the genetic diversity of a GA population fell below a certain pre-determined value. The genetic diversity was defined to be the average number of genetic differences per gene in the population. It was calculated on the decoded variable orderings and not the chromosomes since small differences in ordinal chromosomes can imply large differences in the decoded variable orderings. A genetic diversity of 1.0 would mean that for that population of chromosomes no two decoded variable orderings would have the same variable ordering in the same position, while a genetic diversity of 0.5 would mean that half of all variables in the decoded variable orderings would not have the same variable present in the same position. The advantage of DLS was that apart from a simple cut-off figure no other input was required of the user (apart from the usual GA parameter settings). We performed experiments to determine an effective genetic diversity cut-off figure. We expected the DLS to perform better than ALS as early GA convergence would result in a more prolonged TABU search, whereas in ALS, TABU Search would only be carried out after a certain number of GA evaluations regardless of the genetic diversity. It occurred to us after we had devised DLS that this might provide a mechanism for removing the need for mutation because in the low population scenario we feel that the only role for mutation is as a randomised form of local search. Thus, since it has been our experience that some form of local search/GA hybrid out-performs the pure GA, the implication is that mutation has little effect. Removing mutation in the GA-DLS scenario does not cause
problems with terminating the GA because DLS provides a termination condition which triggers automatically. The other techniques do not possess a similar mechanism to prevent the GA being locked indefinitely into a converged state, and, hence, mutation cannot be realistically removed. As a result of this we carried out some experiments to test this hypothesis.

The following section presents the results of the various experiments using these techniques.

5. Results.

In all experiments the GA parameters were as follows: population size = 10, breeding rate = 60%, mutation rate = 5% and elitism was used unless stated otherwise. The combinational circuit benchmarks used were radd and test5 both of these have 8 input variables giving a search space size of 40,320. These benchmarks were chosen because for these functions the fitness landscape is sufficiently complicated and the fitness is relatively quick to determine. In figures 3-5 we present graphs showing how the average gate count, best gate count and genetic diversity for each generation varies with number of generations when averaged over 10 independent runs of the GA. The total number of different chromosomes that the GA may visit was fixed in advance at 100 and as a result the generation at which the GA terminates is allowed to vary. We found this number of generations to vary markedly. Consequently we calculated the average number of generations and only used data up to this point. This means that towards the end of each plot one is not averaging over 10 GA runs but over a smaller number because some runs of the GA terminate early. As a result the latter parts of each plot will involve GA runs which have not evaluated 100 different chromosomes. The three plots in each figure correspond to different values of the tournament discriminator (td). The behaviour of the plot with td = 0.8 in figure 4 is almost ideal in that the average best gate count rapidly declines to the lowest value and does not level off. This shows that on average the GA was always improving the result and not spending a lot of time in a local optimum. The plot corresponding to td = 0.6 also declines rapidly however it stops too soon. The problem with the plot with td =1.0 is that it levels off at a sub-optimum gate count. The behaviour of the GA relates well to the dependence of average genetic diversity on generation (fig 5). In this figure we see that td=0.8 must correspond to a more ideal average genetic diversity, in that it is high enough to allow the GA to explore the search space effectively and yet being quite discriminating about the fruitful areas to explore. Too high a diversity (td = 0.6) causes too much global exploration and insufficient local exploration, on the other hand too low a diversity (td = 1.0) results in too little global exploration and too much local exploration.
Fig. 3. Average average gate count (radd)

Fig. 4. Average best gate count (radd)

Fig. 5. Average genetic diversity (radd)
In figures 6 and 7 we show the performance of the GA with ALS (TABU) for benchmarks radd and test5 as compared with a random search with ALS (td = 0.8). It is interesting to note the two-minima ‘W’ shape of the two figures and that the plot of the random search with ALS also was broadly similar (in the case of test5). It is clearly beneficial to carry out some local search on the best solution found by the GA. It should be noted that a combination of a small random search followed by local search can perform better than the GA without ALS (fig 6 with Tabu component equal to 40, and fig 7 with Tabu component equal to 100). Also note that the ratios of evaluations in GA to evaluations in Tabu which give an optimum are different for the two benchmarks.

![Graph showing performance of GA with ALS and random search with ALS for benchmarks radd and test5.](image)

**Fig. 6. Average gate counts for ALS (radd)**

![Graph showing another performance comparison for benchmarks radd and test5.](image)

**Fig. 7. Average gate counts for ALS (test5)**
The average best gate count for different ratios of Tabu evaluations to GA evaluations in Embedded Local Search (ELS) for benchmark test5 are presented in table 1. The number of different evaluations was set at 160. Note that the GA+ELS performed better than ALS (where the best average was 409.7) and that the results for the different ratios are very similar.

<table>
<thead>
<tr>
<th>Ratio</th>
<th>Average Best gate count (20 runs) for test5</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>405.6</td>
</tr>
<tr>
<td>3</td>
<td>407.0</td>
</tr>
<tr>
<td>4</td>
<td>407.3</td>
</tr>
<tr>
<td>5</td>
<td>406.2</td>
</tr>
</tbody>
</table>

Table 1. Performance of GA+ELS for different evaluation ratios.

The similarity of the performance figures for the different ratios may be connected with the fact that a Tabu search of one iteration is carried out on the best candidate of the GA population, thus if over a few generations the best doesn’t improve the local searches will look at the same neighbours of the best candidate (the look-up table prevents these duplicates from actually being evaluated). A possibly better way of carrying out ELS would be to carry out the next iteration (and so on) of the Tabu search in the event that the generator provided by the GA was the same as its previous value.

The average best gate count results for the GA with Dynamic Local Search with different genetic diversity cut-offs (diversity at which GA is suspended and Local Search is initiated) are presented in table 2.

<table>
<thead>
<tr>
<th>Genetic diversity cut-off</th>
<th>Average best gate count (20 runs) for test5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.025</td>
<td>415.8</td>
</tr>
<tr>
<td>0.05</td>
<td>417.0</td>
</tr>
<tr>
<td>0.1</td>
<td>411.7</td>
</tr>
<tr>
<td>0.15</td>
<td>411.4</td>
</tr>
<tr>
<td>0.2</td>
<td>415.2</td>
</tr>
<tr>
<td>0.25</td>
<td>414.0</td>
</tr>
<tr>
<td>0.30</td>
<td>418.0</td>
</tr>
<tr>
<td>0.35</td>
<td>415.8</td>
</tr>
</tbody>
</table>

Table 2. Performance of GA+DLS for different genetic diversity cut-offs
The performance figures for GA+DLS are not very good although cut-offs of 0.1 and 0.15 are a bit better. One of the problems with DLS triggered by genetic diversity is that it does not take into account the fact that in the course of the GA the diversity may go very low on several occasions as the GA converges onto different local optima. These large fluctuations do not show up on figure 5 as the plot shows the averages over a number of runs. A more effective scheme might be a combination of DLS and ELS so that when a GA started to converge and the genetic diversity fell below some cut-off a local search could be initiated which fed back into the GA and so on. Further investigation is required to determine whether it is more effective to carry out local search routinely after a number of generations than to intelligently switch-in the local search after some indicator of GA malaise such as low genetic diversity.

The relative merits of using elitism or not were also investigated and the results are presented in figs 8 and 9 (td = 0.8). From the figures it seen that there is a marked improvement in the GA performance if elitism is used even though this leads to a more rapid decline to a lower level of genetic diversity.

Fig. 8. Average best gate count with/without elitism (radd)

Fig. 9. Average genetic diversity with/without elitism (radd)

As discussed in the last paragraph of section 4, we had wondered whether mutation could be removed in the GA+DLS as we felt that a local search might be more effective when the genetic diversity was low. The assumption being that crossover was the effective exploratory
mechanism in the GA. However our experimental findings confounded our expectations. We found that removing mutation worsened GA performance considerably and that different breeding rates (including zero) had little effect on GA performance. We concluded that for very small populations (~10) crossover is not a very effective operator as there is rapid convergence and similar parent chromosomes are unlikely to produce very different offspring. Mutation was important to provide the shift to a new more fertile area of the search space. We emphasise that the situation might be quite different for larger populations. We investigated how average GA performance was affected by choice of population size over the range 4 to 30. We found again somewhat to our surprise that there was no discernible change in the performance with different population sizes. One would expect breeding to be more useful with larger populations as it would be likely to increase genetic diversity. Whether this in turn would lead to an improved performance remains a topic for further work.

6. Conclusions

In this paper we have presented the results of our investigations on the usefulness of combining GAs with local search (Tabu) for the optimisation of multi-level AND-EXOR representations of Boolean functions. This problem has an expensive fitness function which means that for the GA to be efficient a look-up table for chromosome evaluation is required. We feel that in this kind of situation it is necessary to assess optimisation performance on the basis a fixed number of different chromosome evaluations so that different algorithms are compared on the basis of equal CPU time required.

There are a number of ways in which local search can be combined with a GA and we have explored three different types: Appended Local Search - where a local search is carried out on the best problem solution discovered by the GA, Embedded Local Search - where the local search is carried out during the progress of the GA and the two algorithms communicate with one another, and finally, Dynamic Local Search where the changeover from GA to Local search is determined during the execution of the GA, in our case this was triggered by a genetic diversity cut-off. Strictly speaking our Dynamic Local Search was a dynamic Appended Local Search.

We found that the Embedded Local Search proved to be the most effective. However we suspect that it could be improved still further by switching on the embedded search during the GA using the genetic diversity as a trigger i.e. Dynamic Embedded Local Search. Further one could deepen the Local Search while the best candidate from the GA was not improving. This would also remove the need for some appended local search to use up the predetermined amount of local search. The better performance of ELS fits in with a picture of GA evolution in which there is punctuated convergence. This is probably typical of restricted evaluation GAs in which the number of different chromosome evaluations is severely limited. We hope that some our findings might be useful in other problems with expensive fitness functions and where the number of allowed different chromosome evaluations is severely limited.

References.


