Co-evolving Memetic Algorithms: A learning approach to robust scalable optimisation

J. E. Smith
Faculty of Computing, Engineering and Mathematical Sciences
University of the West of England
Bristol, U.K.
james.smith@uwe.ac.uk

Abstract- This paper presents and examines the behaviour of a system whereby the rules governing local search within a Memetic Algorithm are co-evolved alongside the problem representation. We describe the rationale for such a system, and then describe the implementation of a simple version in which the evolving rules are encoded as (condition:action) patterns applied to the problem representation. We investigate the behaviour of the algorithm on a suite of test problems, and show considerable performance improvements over a simple Genetic Algorithm, a Memetic Algorithm using a fixed neighbourhood function, and a similar Memetic Algorithm which uses random rules, i.e. with the learning mechanism disabled. Analysis of these results enables us to draw some conclusions about the way that even the simplified system is able to discover and exploit certain forms of structure and regularities if these exist within the problem space. We show that this "meta-learning" of problem features provides a means of creating highly scalable algorithms for some types of problems. We further demonstrate that in the absence of this kind of exploitable patterns, the use of continually evolving neighbourhood functions for the local search operators adds robustness to the Memetic Algorithm in a manner similar to Variable Neighbourhood Search. Finally we draw some initial conclusions about the way in which this meta-learning takes place, via examination of the use of different pivot rules and pairing strategies between the population of solution and the population of rules.

1 Introduction

The performance benefits which can be achieved by hybridising Evolutionary Algorithms (EAs) with Local Search operators (LSOs), so-called *Memetic Algorithms* (MAs), have now been well documented across a wide range of problem domains (see [1] for a comprehensive bibliography). Commonly in these algorithms, a Local Search improvement step is performed on each of the products of the generating (recombination and mutation) operators, prior to selection for the next population. There are of course many variants on this theme, for example one or more of the generating operators may be absent,or the order in which the operators are applied may vary.

There are three principal components which affect the workings of a Local Search. The first is the choice of pivot rule, which can be *Steepest Ascent* or *Greedy Ascent*. In the former the entire neighbourhood $n(i)$ of the current solution i is searched and the best neighbour used, whereas the latter stops exploring the neighbourhood as soon as an improvement is found. The second component is the depth of the local search. This can vary from a minimum of only one improving step being applied, through a fixed number of iterations, to the extremum where the local search is iteratively applied until a local optimum is reached. Considerable attention has been paid to studying the effect of changing this parameter within MAs e.g. [2], and along with the choice of pivot rule it can be shown to have an effect on the performance of the Local Search algorithm, both in terms of time taken, and in the quality of solution found.

The third, and primary factor that affects the behaviour of the LS is the choice of neighbourhood generating function. This can be thought of as defining a set of points $n(i)$ that can be reached by the application of some move operator to the point i. The provision of a scalar fitness value, f, defined over the search space means that we can consider the graphs defined by different move operators as "fitness landscapes" [3]. A number of statistical measures can used to characterise fitness landscapes, and have been proposed as potential measures of problem difficulty by various authors. Merz and Freisleben [4] discuss a number of these in the particular context of MAs, and show that the choice of move operator can have a dramatic effect on the efficiency and effectiveness of the Local Search, and hence of the resultant MA.

In some cases, domain specific information may be used to guide the choice of neighbourhood structure within the Local Search algorithms. However, it has recently been shown that the optimal choice of operators can be not only instance specific within a class of problems [4, pp254–258], but also dependent on the state of the evolutionary search [5]. This result is not surprising when we consider that points which are locally optimal with respect to one neighbourhood structure may not be with respect to another (unless of course they are globally optimal). Thus if a set of points has converged to the state where all are locally optimal with respect to the current neighbourhood operator, then changing the neighbourhood operator may provide a means of progression, in addition to recombination and mutation. This observation forms the heart of the *Variable Neighbourhood Search* algorithm [6].

In previous papers [7, 8] we reported initial results from a COevolving Memetic Algorithm (COMA) system within which the definitions of Local Search operators applied within the MA may be changed during the course of optimisation. The systems described therein used a simple self-adaptive mechanism to govern the evolution of rules, but were still able to exhibit significant performance improve-

ments over a simple GA, a MA with a simple bit-flipping hill-climber, and a version of the system which used random rules i.e. with the learning turned off in the rule population. Building on those initial results we now turn our attention to examining two distinctly different co-evolutionary processes which lead to either faster or more reliable optimisation. We also consider the effects of varying two of the major factors affecting behaviour, namely the choice of pivot rules, and the nature of the co-evolutionary coupling.

The rest of this paper proceeds as follows: In Section 2 we discuss some previous work in this area, describe our proposed approach, and the simplified model that we will use. In Section 3 we draw some parallels between this work and related work in different fields, in order to place this work within the context of more general studies into adaptation, development and learning. In Section 4 we present the results and analysis of a set of experiments designed to investigate whether, and if so how, the use of adaptive rules is able to benefit the optimisation process, in the presence of regularities within the problem space which can be exploited by a meta-learning process, paying particular attention to the issue of scalability. Following this, in Section 5 we present results and analysis from a set of test problems that are deliberately chosen not to have regularities that can be exploited by our simplified rule definition. In Section 6 we discuss the implications of these results, before drawing conclusions and suggesting future work in Section 7.

2 A Rule-Based Model for the Adaptation of Move Operators

The aim of this work is to provide a means whereby the definition of the local search operator (LSO) used within a MA can be varied over time, and then to examine whether evolutionary processes can be used to control that variation, so that a beneficial adaptation takes place. In order to accomplish this aim, we require the provision of five major components, namely:

- A means of representing a LSO in a form that can be processed by an evolutionary algorithm
- A set of initialisation and variation operators, so as recombination and mutation that can be applied to the LSO representation.
- A means of assigning fitness to the LSO population members
- A choice of population structures and sizes, along with selection and replacement methods for managing the LSO population
- A set of experiments, problems and measurements designed to permit evaluation and analysis of the behaviour of the system.

The representation chosen for the LSO is a tuple $<$*Pivot_rule, Depth, Pairing, Move, Fitness*$>$. The first two elements in the tuple have been described above, and can be easily mapped onto an integer or cardinal representation as desired, and manipulated by standard genetic operators.

The element *Pairing* efectively co-ordinates the evolution of the two populations. When an candidate solution is to be evaluated, a member of the LSO population is chosen to operate on it, hopefully yielding improvements. The fitness of the candidate solution is thus affected by the choice of LSO to operate on it, and the fitness assigned to the LSO is in turn afected by the candidate solution to which it is applied.

Values for *Pairing* are taken from the set $\{$*linked, fitness_based, random*$\}$. Although the long-term goal is to examine a "mixed-economy" of paring strategies, for the purposes of this paper we restrict ourselves to the situation where the whole population uses the same value. The different values have the following effects:

- For a *linked* pairing strategy, the LSOs can be considered to be extra genetic material which is inherited and varied along with the problem representation. Thus if the k^{th} candidate solution is created from parents i and j, then a LSO is created by the actions of recombination and mutation on members i and j of the current LSO population. This new LSO is used to evaluate the new candidate solution and becomes the k^{th} member of the next LSO population. Note that this assumes the two population are the same size. The fitness is assigned to the new LSO is immaterial since selection to act as parents happens via association with good members of the solution population.

- For a *fitness-based* pairing strategy, when a candidate solution requires evaluation, a LSO is created and put into the next LSO population as above. However the two LSOs which acts as parents for recombination are now chosen using a standard selection mechanism acting on the current LSO population. A number of methods can be used to define the fitness of an LSO.

- For a *random* pairing strategy, the same process occurs, except that all LSOs are given the same fitness.

The representation chosen for the move operators was as *condition:action* pairs, which specify a pattern to be looked for in the problem representation, and a different pattern it should be changed to. Although this representation at first appears very simple, it has the potential to represent highly complex moves via the use of symbols to denote not only single/multiple wildcard characters (in a manner similar to that used for regular expressions in Unix) but also the specifications of repetitions and iterations. Further, permitting the use of different length patterns in the *condition* and *action* parts of the rule gives scope for *cut* and *splice* operators working on variable length solutions.

In themselves, the degrees of freedom afforded by the five components listed above provide basis for a major body of research. While the framework that we have described above is intended to permit a full exploration of these issues, we shall initially restrict ourselves to considering a simple system, and examining its behaviour on a well understood set of test problems. For these initial investigations we therefore restrict ourselves to considering only a

single application of rules (i.e. $depth = 1$) where the *condition* and *action* patterns are of equal length. The rules are composed of values taken from the set of permissible allele values of the problem representation, augmented by a "don't care" symbol # which is allowed to appear in the *condition* (but not the *action*, although this could be interpreted as "leave alone") part of the rule. The neighbourhood of a point i then consists of all those points where the substring denoted by *condition* appears in the representation of i and is replaced by the *action*. Note that the neighbourhood of i therefore potentially includes i itself, for example by means of as rule with identical *condition* and *action* parts.

To give an example, if we have a solution represented by the binary string 1100111000 and a rule 1#0:111, then this matches the first, second, sixth and seventh positions, and the neighbourhood is the set {1110111000, 11111111000, 1100111100,1100111110}. Note that in this work we do not considered the string as toroidal (although this will be considered in later work), and that a random permutation is used to specify the order in which the neighbours are evaluated, so as not to introduce positional bias into the local search when greedy ascent is used.

In practice, each rule was implemented as two 16 bit strings (the restriction to 16 bits here is purely for ease of visualisation). This representation for the rules means that "standard" genetic operators (uniform/1 point crossover, point mutation) can be used to vary this part of the LSO chromosome. It is augmented by a value *rule_length* which detailed the number of positions in the pattern string to consider. In [7] we examined the effects of different fixed rule sizes, and a version which had the ability to adapt via the action of mutation operators on this value. The results obtained demonstrated that the version with adaptive rule lenths was able to quickly adapt to using the appropriate optimal size rule for the different problems.

3 Related Work

The COMA system can be related to a number of different branches of research, all of which offer different perspectives and means of analysing its behaviour. These range from Multimemetic Algorithms and the Self-Adaptation of search strategies, through co-evolutionary, learning and developmental systems, to the evolutionary search for generalised rules as per Learning Classifier Systems. Space precludes a full discussion of each of these, so we will briefly outline some of these perspectives.

Krasnogor and Smith [5] describe what they call a "MultiMemetic Algorithm", in which a gene is added to the end of each chromosome indicating which of a fixed set of static LSOs ("memes") should be applied to the individual solution. Variation is provided during the mutation process, by randomly resetting this value with a low probability. They report that their systems are able to adapt to use the best meme available for different instances of TSP. Krasnogor and Gustafson have extended this and proposed a grammar for "Self-Generating MAs" which specifies, for instance, where in the evolutionary cycle local search takes place [9]. Noting that each meme potentially defines a different neighbourhood function for the local search part of the MA, we can also see an obvious analogy to the Variable Neighbourhood Search algorithm [6], where a heuristic is used to control the order of application of a set of local searchers (using different, fixed, neighbourhood structures) to a single improving solution. The difference here lies in the population based nature of COMA, so that not only do we have multiple candidate solutions, but also multiple adaptive neighbourhood functions in the memes.

If the populations are of the same size, and are considered to be linked, then this instantiation of the COMA framework can be considered as a type of Self Adaptation. The use of the intrinsic evolutionary processes to adapt step sizes governing the mutation of real-valued variables has long been used in Evolution Strategies [10], and Evolutionary Programming [11]. Similar approaches have been used to self-adapt mutation probabilities [12, 13] and recombination operators [14] in genetic algorithms as well as complex generating operators which combined both mutation and recombination [15]. This body of work contains many useful results concerning the conditions necessary for strategy adaptation, which could be used to guide implementations of COMA.

If the two populations are not linked, then we have a co-operative coevolutionary system, where the fitness of the members of the LSO population is assigned as some function of the relative improvement they cause in the "solution" population. Paredis has examined the co-evolution of solutions and their representations [16], and Potter and DeJong have also used co-operative co-evolution of partial solutions in situations where an obvious problem decomposition was available [17], both with good reported results. Bull [18] conducted a series of more general studies on co-operative co-evolution using Kauffmann's static NKC model [19]. In [20] he examined the evolution of linkage flags in co-evolving "symbiotic" systems and showed that the strategies which emerge depend heavily on the extent to which the two populations affect each others fitness landscape, with linkage preferred in highly interdependent situations. He also examined the effect of different pairing strategies, [21] with mixed results, although the NKC systems he investigated used fixed interaction patterns, whereas in the systems used here are more dynamic in nature. There has also been a large body of research into competitive-co-evolution, (an overview can be seen in [22]) whereby the fitnesses assigned to the two populations are directly related to how well individuals perform "against" the other population, what has been termed "predator-prey" interactions.

In both the co-operative and competitive co-evolutionary models, the different populations only affect each other's perceived fitness, unlike the COMA framework where the LSO population can directly affect the genotypes within the solution population. A major source of debate and research within the community has focused around the perception that this phase of improvement by LSO can be viewed as a kind of lifetime learning. This has lead naturally to speculation and research into whether the modified phenotype which is the outcome of the improvement process should

be written back into the genotype (Lamarkian Learning) or not (Baldwinian Learning). Note that although the description of local search, and the discussion above assumes Lamarkian learning, this is not a prerequisite of the framework. However, even if a Baldwinian approach was used, the principal difference between the COMA approach and the co-evolutionary systems above lies in the fact that there is a selection phase within the LSO, that is to say that if all of the neighbours of a point defined by the LSO rule are of inferior fitness, then the point is retained unchanged within the population.

If we were to discard this criterion and simply apply the rule (possibly iteratively) we could view the system as a type of "developmental learning" akin to the studies in Genetic Code e.g. [23] and the "Developmental Genetic Programming" of Keller and Banzhaf [24, 25]

Finally, and perhaps most importantly, we should consider that if a rule has an improving effect on different parts of a solution chromosome over as number of generations, then the evolution of rules can be seen as learning generalisations about patterns within the problem representation, and hence the solution space. This point of view is akin to that of Learning Classifier Systems. For the case of unlinked fitness-based selection of LSOs, insight from this field can be used to guide the credit assignment process.

4 Scalability via the exploitation of regularities in the search space

4.1 Experimental set-up

In all of the following experiments we used a population size of 500 for both solutions and rules. A generational strategy was used with tournament selection into the mating pool of solutions. The mating pool of rules was selected according to the pairing strategy. One point crossover was applied to both mating pools with probability 0.7 and bit-flipping mutation was applied to the resultant offspring with probability 0.01 for the solutions and 0.0625 (the inverse of the maximum rule length) to the rules. With probability 0.0625 a value of one was randomly added or subtracted to the rule length, subject to staying within the range 1 to 16. These GA parameters were taken as "standard" from the literature as we interested in creating robust problem solvers rather than spending considerable time tuning our EA for each problem instance.

For each problem instance 50 runs were made of each algorithm, stopping when the global optimum was found, or one million evaluations were made, whichever was sooner. Note that this last criterion takes into account the search effort incurred in the local search process.

In addition to a GA, and a Simple Memetic Algorithm (SMA) employing a bit-flipping hill-climber, we examined variants of COMA using all possible combinations of greedy or steepest ascent with linked, random or fitness-based pairing strategies. In the latter case rules were selected to be used via binary tournaments, with the fitness of each rule defined as the fitness improvement which it caused in the candidate solution with which it was last evaluated. The implications of this credit assignment strategy will be discussed later.

4.2 Test Suite

As noted above, previous results have suggested that the COMA algorithm is able identify and utilise regularities in the problem space, either in the form of "building blocks" in classic GA test functions [7] or secondary structural motifs in proteins [8]. In order to investigate this phenomenon further we constructed multiple length variants of two test functions whose properties are well known.

The first of these, which we will refer to as "4Trap", comprised multiple concatenated copies of Deb's 4-bit fully deceptive function given in [26]. The fitness of each sub-problem i is given by its unitation $u(i)$ (i.e. the number of bits set to "one"):

$$f(i) = \begin{cases} 0.6 - 0.2u(i) & : \quad u(i) < 4 \\ 1 & : \quad u(i) = 4 \end{cases} \tag{1}$$

We examined functions with lengths in the range 40 to 200 increasing in steps of 20. As discussed in [7] the nature of this function is such that the use of an inappropriate neighbourhood function tends to lead to a local optimum: for instance simple hill-climbers are notoriously poor at these functions.

The second test function is Watson's HIFF function, which is a highly epistatic problem designed to examine the virtues of recombination. At the bottom level, fitness is awarded to matching pairs of adjacent bits in a solution s, i.e.

$$f_1 s = \sum_{i=0}^{l/2-1} 1 - XOR(s_{2i}, s_{2i+1}) \tag{2}$$

and this process is applied recursively, so that a problem of size $l = 2^k$ has k levels. In each ascending level the number of blocks is reduced by a factor of two, and the fitness awarded for each matching pair is increased by a constant factor, in our case 2. Thus for example the four-bit sub-solution 1100 scores for two matching blocks at the first level, but at the second level we have 10 so no credit is earned and a null value is carried forward to the next level. As can be seen there are a number of sub-optima, and two global optima corresponding to the all-ones and all-zeroes strings. We used problem sizes ranging from 8 to 512 bits, corresponding to a range of 3 to 9 levels.

4.3 Results

Figure 1 shows the average number of evaluations to find the global optimum for the different algorithms on the 4Trap problem as a function of length. The annotations display the number of successful runs where this is less than 50. Note that the results for the SMA are not shown as these are extremely poor. As can be seen in many cases the speed-up is near-linear, although the rate of increase and the change in success rate differs. We can distinguish the following observations:

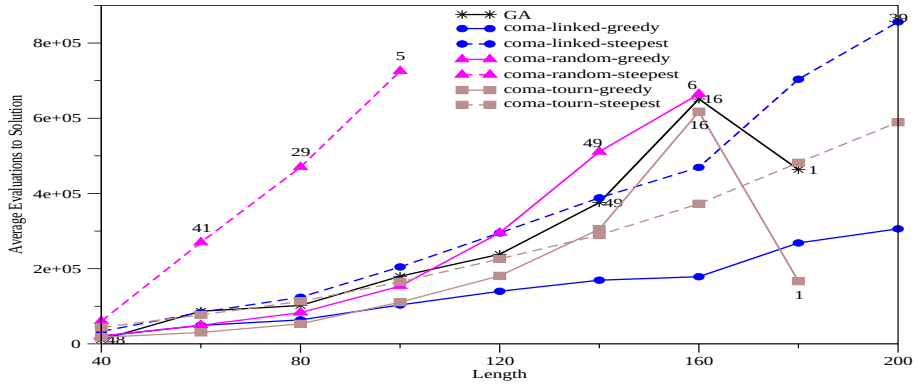

Figure 1: Mean time to solution as a function of length for 4 Trap problem. Annotations display number of successful runs where this is less than 50.

- In general the greedy version of the algorithm find the optimum in less time that the steepest ascent versions. This indicates that on average more than one element in each neighbourhood is fitter than the starting position.
- The versions of COMA with random pairing perform worse than the GA.
- The two versions of COMA with linked rules (which we will subsequently refer to as CLG and CLS), and the combination of tournament pairing with steepest ascent (CTS) all exhibit good scalable performance, that of the CLG algorithm being particularly noticeable.
- The fitness-based tournament pairing works well with steepest ascent but much less so with greedy ascent. Bearing in mind the first observation, this suggest that the assignment of fitness to rules is extremely susceptible to noise.

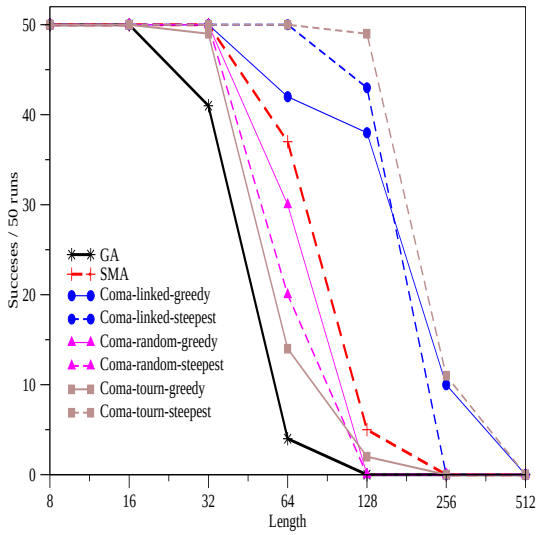

Figure 2: Success rate as a function of length for HIFF problem.

Figures 2 and 3 show the success rate and mean time to solution respectively for the HIFF problem as a function of length. We now see that all of the MAs show an improvement over the GA, and again the same three variants of COMA show the best performance, with only the CLG and CTS variants solving the 256 bit problem. Note that the length of the blocks that must be identified and matched at the highest levels far exceeds the maximum length of the rules. In general the greedy ascent versions find the optimum faster than the equivalent steepest ascent versions but not as reliably. Given that the success rate of most of the algorithms is less than 100% for lengths above 32, we performed an analysis of variance (ANOVA) on the best fitness at the end of each run, which confirmed that the performance is statistically significantly different with 95% confidence. Post-hoc analysis via the LSD measure confirms that the CLG, CLS and CTS variants have a higher mean best fitness than all other algorithms but do not significantly differ .

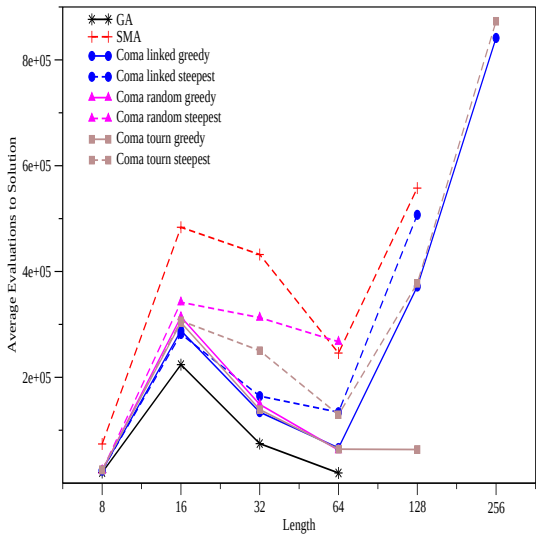

Figure 3: Mean number of evaluations to solution as a function of length for HIFF problem.

5 Escaping local optima by changing neighbourhood definitions

The performance improvements exhibited above clearly arise from a situation in which the adjacent epistatic interactions within the problem give rise to patterns in the search space which can be exploited by COMA. In order to examine the behaviour when this is not the case we used two additional 64 bit variants of the 4Trap function.

In the first of these,(DistTrap) the subproblems were interleaved i.e. sub-problem i was composed of the genes $i, i+16, i+32, i+48$. This separation ensures that even the longest rules allowed in these experiments would be unable to alter more than one element in any of the sub-functions in a single application. On this function the only algorithms which ever located the global optimum were the CLG, CLS and CST, all three of which always located the global optimum. The mean times to solution were 90237, 206266 and 168898 evaluations respectively, and these differences are significant with 99.9% confidence.

The poor results of the GA can be attributed to the fact that the representation of the problem means that with probability $61/64$ the crossover point will fall within a given subproblem, disrupting transmission and mixing of any optimal sub-solutions found. Given that the deceptive problems are specifically designed to be mutation-difficult, this also helps to explain the poor performance of the SMA.

A third variant of this problem (Shifted-Trap) was designed to be more "difficult" than the first for the COMA algorithm, by making patterns which were optimal in one sub-problem, sub-optimal in all others. This was achieved by noting that each sub-problem as defined above is a function of unitation, and therefore can be arbitrarily translated by defining a 4-bit string and using the Hamming distance from this string in place of the unitation. Since we have 16 sub-problems, we simply used the binary coding of the sub-problem's index as basis for its fitness calculation.

On this problem the SMA found the optimum 45 times out of the 50 runs, and the random-pairing steepest ascent version of COMA (CRS) 39 times. The GA and all other variants of COMA always located the global optimum in the time allowed. Note that the performance of the GA on this problem is identical to that on a 64 bit version of 4Trap as would be expected. Figure 4 shows the mean time to solution for each algorithm. As can be seen, although it might be expected that attempting to reuse a pattern on different sub-problems would hinder the progress of the COMA algorithms, in fact the mean solution time is not significantly different to that of the GA for all but SMA and the CRS variant, and there is a noticeable reduction in the variability of time to solution.

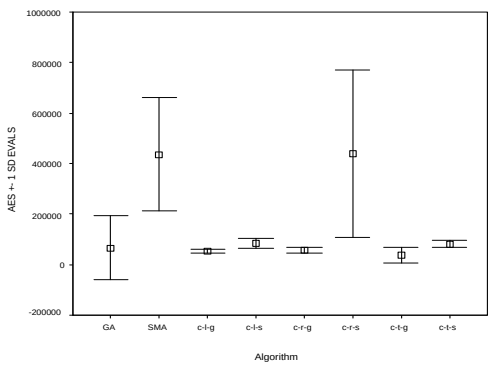

Figure 4: Mean number of evaluations to solution for Shifted-Trap problem. Error bars show +/- 1 standard deviation

6 Discussion and Analysis

6.1 Evolution of Rule-Base

In [7] we presented an analysis of the behaviour of the evolving rule-sets on the three versions of the Trap function which showed that distinctly different behaviour could be observed according to whether the simple structure of the rules used here was able to identify and represent the optimal solution to a sub-problem. Lack of space precludes reproducing those results in full, but we can summarise them as follows.

For the 4Trap function, the system rapidly evolves medium length ($3-4$), general (specificity $<$ 50%) rules whose action is to set all the bits to 1 (mean unitation approaches 100%). Note that in the absence of selective pressure (i.e. the pivot rules meant that the solutions were left unchanged), all three of these values would be expected to remain at their initial values, so these changes result from beneficial adaptation. Closer inspection of the evolving rule-base confirms that the optimal subproblem string is being learned and applied.

For the Shifted-Trap function, where the optimal sub-blocks are all different, the rule length decreases more slowly from its initial mean value of 8. The specificity also remains higher, and the unitation remains at 50%, indicating that different rules are being maintained. This is borne out by closer examination of the rule sets.

The behaviour on Dist-Trap is similar to that on 4Trap, albeit over a longer time-scale. Rather than learning specific rules about sub-problems, which cannot possibly be happening (since no rule is able to affect more than one locus of any subproblem), the system is apparently learning the general rule of setting all bits to 1. The rules are generally shorter than for 4Trap, which means that the number of potential neighbours is higher for any given rule. Equally, the use of wildcard characters, coupled with the fact that there may be matches in the two parts of the rules, means that length of the rules used defines a maximum radius in Hamming space for the neighbourhood, rather than a fixed distance from the original solution. Both of these observations, when taken in tandem with the longer times to solution, suggest that when the system is unable to find a single rule that matches the problems' structure, a more diverse search using a more complex neighbourhood is used, which slowly adapts itself to the state of the current population of solutions.

The same analysis was performed for the experiments

presented here, which showed that for the successful variants of COMA examined, the same behaviour was exhibited. In the case of 4Trap this creates a highly scalable algorithm by identifying the string corresponding to the optimal solution for each sub-problem, and then applying to each sub-problem in the string in successive generations. A linear regression showed that a straight line of the form $Mean\_Solution\_Time = -11931 + 484.71 \cdot length$ forms an extremely good fit to the observed results with a correlation co-efficient of 0.97 for the CLG algorithm.

In contrast to this, analysis suggests that for the HIFF problem the improved scalability arises from allowing the system to make a decision between the "ones" blocks and zeroes blocks and then apply these throughout the string. The choice between 1s and 0s appeared to occur with equal probability, that is to say that both peaks were identified in the 50 runs.

6.2 Pairing and Pivot Strategies

The results presented above show that the choice of pivot and pairing strategies is crucially important.

Unsurprisingly, the greedy variants almost always run faster than the steepest ascent equivalents when they do solve the problems. Noting that the random pairing strategy does not perform especially well, and taking into account the the poor results for the CTG strategy (especially compared to the good ones for CTS), suggests an explanation in terms of a need for accurate selection in the LSO population.

In the linked variants, this selective pressure towards the evolution of good rules is created implicitly via a continued association with fit solutions. In the case of random pairing, there is no selective pressure in the LS population, so the rule base will remain diverse until genetic drift causes an eventual convergence. For the Shifted-Trap function this is not a problem, since it is desirable to maintain different *condition* parts of the rules for different sub-problems, and CRG shows good performance. However for the other problems it prevents identification and use of rules with high unitation, and a corresponding decrease in performance is observed.

It would appear that for the CTG algorithm the extra noise introduced by using a greedy rather than a steepest ascent is sufficient to "fool" the simple credit assignment mechanism used in these experiments. Thus a good rule might only get a low fitness if the first place in which it matches only leads to a small improvement, whereas larger improvement might be seen if it was applied elsewhere in the solution. However a "good rule" such as $\#\#\#\# \rightarrow 1111$ for 4Trap could get a low fitness under steepest ascent with the scheme used here, depending on what individual it is paired with. It is possible that a more sophisticated method such as Paredis Life Time Fitness Evaluation (LTFE) (in which a running average of the last twenty pairings is used) may well provide a more stable and robust credit assignment mechanism whilst retaining the speed-benefits of greedy ascent.

7 Conclusions

We have examined the behaviour of a simple implementation of the COMA framework on two different classes of problem and observed that performance improvement can arise from different mechanisms. When the representation of the rules is able to capture regular repeated features within the problem space, we see highly scalable behaviour - for example the linear speed-up on the 4trap function. This arises from the rapid evolution of the system to a rule-set which captures and represents knowledge about how to solve the problem. We have observed that in order for this to occur it is necessary to maintain sufficient *accurate* selection pressure within the population of Local Searchers.

In contrast to this, when there is not sufficient selective pressure for evolution, for example when a "good" pattern only applies to one position in the solution, or when the rule representation cannot possibly capture the regularities present in the space, a "fall-back" position is observed. In this case we again see improved reliability, but at the expense of speed of solution. Essentially what happens is that even if the solution population converges to a local optimum for its crossover and mutation operators, the continued generation of new rules, which define new neighbourhood structures, means that eventually a landscape is discovered in which the solutions are not locally optimal and improvements can occur. This is akin to Variable Neighbourhood Search, but with the advantage that it is not necessary to specify, or be bound by, a fixed set of neighbourhood functions.

Clearly there remains much work to be done analysing the possibilities of this framework, and it would be fatuous to claim that we have discovered a fabulous all-purpose problem solver. However the two modes of operation noted above, coupled with the promise of an algorithm which is able to explicitly represent the information that it has learned and is using to solve the problem at hand, would seem to offer much potential.

Bibliography

[1] P. Moscato, "Memetic algorithms' home page," http://www.densis.fee.unicamp.br /~moscato/memetic_home.html, 2002.

[2] W. E. Hart, "Adaptive global optimization with local search," Ph.D. dissertation, University of California, San Diego, 1994.

[3] T. Jones, "Evolutionary algorithms, fitness landscapes and search," Ph.D. dissertation, The University of New Mexico, Albuquerque, NM, 1995.

[4] P. Merz and B. Freisleben, "Fitness landscapes and memetic algorithm design," in *New Ideas in Optimization*, D. Corne, M. Dorigo, and F. Glover, Eds. McGraw Hill, 1999, pp. 245–260.

[5] N. Krasnogor and J. Smith, "Emergence of profitiable search strategies based on a simple inheritance mecha-

nism," in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, Spector et. al., Eds. Morgan Kaufmann, 2001, pp. 432–439.

[6] P. Hansen and N. Mladenovìc, "An introduction to variable neighborhood search," in *Meta-Heuristics: Advances and trends in local search paradigms for optimization. Proceedings of MIC 97 Conference*, S. Voßet. al. Eds. Dordrecht, The Netherlands: Kluwer Academic Publishers, 1998. [Online]. Available: http://www.crt.umontreal.ca/ pierreh/puben.html or http://www.wkap.nl/book.htm/0-7923-8369-9

[7] J. Smith, "Co-evolution of memetic algorithms : Initial investigations," in *Proceedings of the 7th Conference on Parallel Problem Solving from Nature*, J. M. Guervos et. al., Eds., LNCS no.2439. Springer, Berlin, 2002, pp. 537–548.

[8] ——, "The co-evolution of memetic algorithms for protein structure prediction," in *Advances in Nature-Inspired Computation: The PPSN VII Workshops*, Corne et. al., Eds. Reading, UK: PEDAL, University of Reading, 2002, pp. 14–15.

[9] N. Krasnogor and S. Gustafson, "Toward truly "memetic" memetic algorithms: discussion and proofs of concept," in *Advances in Nature-Inspired Computation: The PPSN VII Workshops*, Corne et. al., Eds. Reading, UK: PEDAL , University of Reading, 2002, pp. 9–10.

[10] H.-P. Schwefel, *Numerical Optimisation of Computer Models*. New York: John Wiley and Sons, 1981.

[11] D. B. Fogel, "Evolving artificial intelligence," Ph.D. dissertation, University if California, 1992.

[12] T. Bäck, "Self adaptation in genetic algorithms," in *Toward a Practice of Autonomous Systems: Proceedings of the 1st European Conference on Artificial Life*, F. Varela and P. Bourgine, Eds. The MIT Press, Cambridge, MA, 1992, pp. 263–271.

[13] J. Smith and T. Fogarty, "Self adaptation of mutation rates in a steady state genetic algorithm," in *Proceedings of the 1996 IEEE Conference on Evolutionary Computation*. IEEE Press, Piscataway, NJ, 1996, pp. 318–323.

[14] J. Schaffer and A. Morishima, "An adaptive crossover distribution mechanism for genetic algorithms," in *Proceedings of the 2nd International Conference on Genetic Algorithms and Their Applications*, J. Grefenstette, Ed. Lawrence Erlbaum Associates, 1987, pp. 36–40.

[15] J. Smith and T. Fogarty, "Adaptively parameterised evolutionary systems: Self adaptive recombination and mutation in a genetic algorithm," in *Proceedings of the 4th Conference on Parallel Problem Solving from Nature*, H.-M. Voigt et. al. Eds., LNCS 1141. Springer, Berlin, 1996, pp. 441–450.

[16] J. Paredis, "The symbiotic evolution of solutions and their representations," in *Proceedings of the 6th International Conference on Genetic Algorithms*, L. Eshelman, Ed. Morgan Kaufmann, San Francisco, 1995, pp. 359–365.

[17] M. A. Potter and K. DeJong, "A cooperative coevolutionary approach to function optimisation," in *Proceedings of the 3rd Conference on Parallel Problem Solving from Nature*, Y. Davidor, H.-P. Schwefel, and R. Männer, Eds., LNCS 866. Springer, Berlin, 1994, pp. 248–257.

[18] L. Bull, "Artificial symbiology," Ph.D. dissertation, University of the West of England, 1995.

[19] S. Kauffman, *Origins of Order: Self-Organization and Selection in Evolution*. Oxford University Press, New York, NY, 1993.

[20] L. Bull and T. C. Fogarty, "Horizontal gene transfer in endosymbiosis," in *Proceedings of the 5th International Workshop on Artificial Life : Synthesis and Simulation of Living Systems (ALIFE-96)*, C. G. Langton and K. Shimohara, Eds. Cambridge: MIT Press, May 16–18 1997, pp. 77–84.

[21] L. Bull, "Evolutionary computing in multi agent environments: Partners," in *Proceedings of the 7th International Conference on Genetic Algorithms*, T. Bäck, Ed. Morgan Kaufmann, San Francisco, 1997, pp. 370–377.

[22] J. Paredis, "Coevolutionary algorithms," in *Handbook of Evolutionary Computation*, T. Bäck, D. Fogel, and Z. Michalewicz, Eds. Institute of Physics Publishing, Bristol, and Oxford University Press, New York, 1998.

[23] H. Kargupta and S. Ghosh, "Towards machine learning through genetic code-like transformations," Computer Science and Electrical Engineering Department, University of Maryland Baltimore County, Tech. Rep. TR-CS-01-10, 2001.

[24] R. E. Keller and W. Banzhaf, "Genetic programming using genotype-phenotype mapping from linear genomes into linear phenotypes," in *Proceedings of the 1st Annual Conference on Genetic Programming*, Koza et. al., Eds. MIT Press, 1996, pp. 116–122.

[25] ——, "The evolution of genetic code in genetic programming," in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-1999)*, Banzhaf et. al. Eds. Morgan Kaufmann, 1999, pp. 1077–1082.

[26] T. Bäck, D. Fogel, and Z. Michalewicz, Eds., *Handbook of Evolutionary Computation*. Institute of Physics Publishing, Bristol, and Oxford University Press, New York, 1997.