Self-Adaptation by Co-evolution

Andrew Tuson and Peter Ross
Department of Artificial Intelligence, University of Edinburgh
80 South Bridge, Edinburgh EH1 1HN, U.K.
{andrewt,peter}@aisb.ed.ac.uk

Abstract

Traditional genetic algorithms use operator settings such as the crossover rate or number of crossover points that are fixed throughout a given run. The choice of settings can have a major effect on performance, but finding good settings can be hard. One option is to encode the operator settings onto each member of the GA population, and allow them to evolve too. This paper describes an empirical investigation into co-evolving operator settings in genetic algorithms. The results indicate that the problem representation and the choice of operators that are applied to the encoded operator settings is important for useful adaptation to take place.

1 Introduction

The choice of operator settings (e.g. the mutation rate) has a significant impact upon GA performance. However, finding a good set of choices is considered difficult as there is often considerable interaction between them. Also, the ideal operator settings may be time-dependent [2]. It may be advantageous, therefore, to employ a method that adjusts the operator probabilities automatically. One approach — co-evolution — will be investigated in this paper.

2 What is Co-evolution?

This paper makes a distinction between an *operator probability* — the probability of an operator being fired, and any parameters associated with an operator (*operator parameters*), such as the bitwise mutation rate. The term *operator setting* will be taken to mean both of the terms above.

Operator adaptation methods based on the co-evolutionary metaphor encode some or all operator settings as an extra part of each member of the population and allow the settings themselves to evolve. The rationale behind this that solutions which have encoded operator settings that tend to produce fitter children will survive longer and so the useful operator settings will spread through the population.

Co-evolution is implemented as follows. For any given problem there is a predetermined crossover operator and a mutation operator; for example, for the flowshop problem modified PMX crossover and shift mutation are used, because investigation shows they perform well. A chromosome contains the usual problem-specific solution encoding, and also a vector of real numbers. In the simplest case, where only operator probabilities are to be co-evolved, the vector encodes the operator probabilities, which are used as follows. A chromosome is chosen by rank-based selection. Its encoded $p_o p$ is used to decide whether the operator is to be applied. If necessary, a second parent is chosen and the operator applied. Either way, a single child is produced.

If operator parameters are to be co-evolved, the extra vector would contain these. For example, in parameterised uniform crossover a probability p_f determines the chance that a gene is copied from the first parent rather than the second. In genewise mutation, a probability p_m gives the probability of randomly changing any given gene.

How do these vectors get changed in the child? They undergo crossover and mutation themselves. We explored two variants: using either *strongly disruptive* or *weakly disruptive* operators. The former uses Radcliffe's random respectful recombination if there are two parents; each probability is randomly chosen from the range delimited by the parents' values. It uses uniform random mutation if there is one parent. The latter variant uses parameterised uniform crossover with parameter 0.1 if there are two parents, so the child's vector is likely to be close to its first parent's. It uses a biased mutation if there is only one parent: the new value is chosen from a Gaussian distribution of SD 0.05 whose mean is the parent's value.

This general idea raises various questions:

- **Should we evolve operator parameters or probabilities?** In this study we consider both, applying these ideas to five well-known problems. We try a steady-state GA and a generational GA with elitism in each case. Population size is kept fixed at 100.

- **What operators should be used on the encoded settings?** In particular should they be *strongly*, or *weakly* disruptive?

- **What level of co-evolution?** An easy approach fixes the co-evolution crossover probability, but there is a risk that the chosen value will be poor. A higher level approach encodes this probability onto the string too (termed *meta-learning*).

3 The Investigation

The first of the five test problems is a hard scheduling/operations research problem. The other members have been selected on account of their theoretical interest and familiarity. All but the flowshop problem use uniform crossover (parameter 0.5) and genewise mutation with probability $1/l$ (l=length) for the solution part of the chromosomes.

- **Flowshop Sequencing**. A Taillard [7] benchmark problem: 20 jobs and 20 machines with RNG seed 47934044.

- **Max Ones** of length 100.

- **Order-3 Deceptive** [4] of length 30.

- **Royal Road** (R1) [3] of length 64.

- **Long Path** [6] of length 29.

A large number of experiments were performed in this study; see [8] for any results and plots mentioned but not given here. Each tabulated result is based on fifty runs. Two measures of performance were used: the quality of solution obtained after 10,000 evaluations, and the number of evaluations after which improvements in solution quality were no longer obtained (or 10,000 evaluations, whichever is smaller).

For comparison, results for a conventional GA using a fixed crossover probability (p_c) are shown in Table 1. For each problem, an extensive search was made to find the best crossover probability for that problem, using values from 0.0 to 1.0 in steps of 0.05. This also revealed how sensitive GA performance

	Generational GA		Steady-State GA	
Problem	Fitness	Evaluations	Fitness	Evaluations
Flowshop	2441 (71) [0.95]	7714 (1573) [0.15]	2387 (71) [0.05]	5064 (1616) [0.55]
Max Ones	99.96 (0.20) [0.8]	7714 (1025) [0.8]	100.00 (0.00) [0.0-0.90]	2172 (702) [0.80]
Deceptive	289.68 (2.41) [0.05]	4484 (1883) [0.90]	289.28 (3.22) [0.1]	2095 (1552) [0.95]
Royal Road	35.52 (6.02) [0.35]	5626 (2254) [0.95]	40.64 (7.65) [0.95]	2876 (2260) [0.25]
Long Path	48687 (3441) [0.55]	3976 (1717) [0.95]	48024 (2907) [0.05]	5129 (4088) [0.2]

Table 1: Best Results Obtained Using a GA With Fixed Operator Probabilities

is to the crossover probability, giving some indication of how hard the GA is to tune, and allowing later comparison of tuning difficulty when co-evolution is used. In Table 1, the standard deviation is given in parentheses, and the best-found crossover probability used is given in *square* brackets. The choice of crossover probability appeared to depend upon the problem to be solved, the population model (but the steady-state GA gave consistently better performance), and the performance criterion being used.

3.1 Co-evolving Probabilities

For both strongly and weakly disruptive co-evolution operators, an extensive search was carried out: a genetic algorithm was run 50 times for each of 21 different co-evolution crossover probabilities from 0.0 to 1.0 with steps of 0.05. Experiments were also performed using meta-learning.

The best average performances attained for each problem/population model are given in Tables 2 and 3 with standard deviations given in parentheses. The underlined table entries indicate a significant difference (by t-test) in performance when compared to the conventional GA.

For both operator sets, performance of the GA was found, in all cases, to be insensitive to the choice of co-evolution crossover probability. The effect of using meta-learning was found to be insignificant.

Comparison with a conventional GA shows a drop in performance when co-evolution is used, in many cases. The cases in which performance remained comparable were those that had suitable crossover probabilities of around 0.5; or were insensitive to the crossover probability anyway. The operator set used made little impact.

Why the drop in performance? A more detailed study showed that with strongly disruptive operators, the encoded p_c tended to stay around 0.5; adaptation was having little effect. With weakly disruptive operators, adaptation was found to occur only in the flowshop sequencing and max ones problems and only with a steady-state model. In the other cases adaptation was not seen to occur. Therefore the choice of co-evolutionary operators is important for adaptation.

However, performance was worse even when adaptation was observed. Could this be due to the time taken to evolve a suitable crossover probability? By the time that the GA has found a 'good' crossover probability, the population can be close to the optimum or to convergence anyway, allowing little scope for a positive impact on performance. This requires further investigation. Observing the effect upon GA performance of initialising the crossover probability closer to a known suitable value may provide useful information.

	Generational GA		Steady-State GA	
Problem	Fitness	Evaluations	Fitness	Evaluations
Flowshop	2444 (67)	7758 (1688)	2394 (72)	5063 (1628)
Max Ones	99.90 (0.30)	8374 (839)	100.00 (0.00)	2478 (418)
Deceptive	288.28 (2.80)	5692 (2159)	288.60 (2.31)	2518 (2124)
Royal Road	35.36 (6.01)	6794 (1805)	32.80 (9.23)	2759 (1961)
Long Path	49164 (107)	5150 (1797)	46649 (6831)	4622 (4165)

Table 2: Results Obtained Using Co-evolution With Strongly Disruptive Operators

	Generational GA		Steady-State GA	
Problem	Fitness	Evaluations	Fitness	Evaluations
$n/m/P/C_{max}$	2444 (73)	7946 (1795)	2397 (71)	5572 (2000)
Max Ones	99.82 (0.43)	8070 (911)	100.00 (0.00)	2475 (842)
Deceptive	288.56 (2.91)	5814 (2275)	289.00 (3.18)	2436 (1592)
Royal Road	35.20 (5.77)	6898 (1907)	31.68 (8.758)	2663 (2202)
Long Path	48687 (3441)	4768 (1870)	44509 (9753)	4933 (4067)

Table 3: Results Obtained Using Co-evolution With Weakly Disruptive Operators

3.2 Co-evolving Parameters

What happens if the crossover and mutation parameters are encoded, rather than their probabilities of being used? The modified PMX used in the flowshop problem has no such parameter, so this problem is omitted here. On the basis of the earlier results, it was decided to use weakly disruptive operators, without meta-learning. The operator probabilities were fixed throughout the GA run — only the operator parameters evolved.

For the max ones problem adaptation does take place, to around the widely-recommended $1l$; it also happens in the long path problem. However, adaptation only happened in the generational model. Further investigation is needed to ascertain why. No adaptation was observed for either the deceptive or royal road problems. For the deceptive problem, it could be the case that the optimal mutation parameter lies in the middle of the encoded range. An experiment where the mutation parameter is initialised towards one end of the range should resolve this question. For the royal road problem, adaptation may be made difficult by the stepwise fitness function, which could make information on mutation parameter performance intermittent.

The performance of the GA was found to be insensitive to the choice of co-evolution crossover probability. When the quality of results (Table 5) is examined for the cases for which adaptation took place, GA performance was found to be degraded. As suggested earlier, this may be a result of the time it takes for the GA to adapt — good choices of the operator parameters are more important earlier in the GA run than later.

The speed to solution for the deceptive problem was improved, possibly due to a mutation parameter close to the theoretically optimal value. For the royal road problem, an exchange of decreased solution quality, in favour of increased speed of search was observed, an effect of the increased mutation parameter.

Problem	Generational GA p(Xover)	Fitness	Evaluations	Steady-State GA p(Xover)	Fitness	Evaluations
Max Ones	0.80	99.96 (0.20)	7714 (1025)	0.80	100.00 (0.00)	2172 (702)
Deceptive	0.05	289.68 (2.41)	5960 (1965)	0.20	289.12 (3.08)	3506 (2523)
Royal Road	0.35	35.52 (6.02)	7804 (1390)	0.95	40.64 (7.65)	3786 (1741)
Long Path	0.55	48686 (3441)	5370 (2138)	0.20	47312 (4765)	5129 (4088)

Table 4: The Fixed Operator Probabilities Used To Compare Co-evolution of Operator Parameters And Their Performance

Problem	Generational GA Fitness	Evaluations	Steady-State GA Fitness	Evaluations
Max Ones	99.52 (0.64)	8438 (857)	100.00 (0.00)	2791 (1011)
Deceptive	289.12 (2.83)	5306 (1918)	289.32 (2.61)	2555 (2124)
Royal Road	31.36 (6.16)	6086 (2505)	29.76 (8.32)	2428 (1937)
Long Path	46906 (7342)	5892 (2461)	46375 (5523)	5671 (3768)

Table 5: Results Obtained When Co-evolving Operator Parameters

4 Conclusions

First, it was confirmed that the choice of operator settings has a marked effect upon GA performance. The choice of co-evolution operators was found to be important: weakly disruptive operators are required for adaptation. Performance was seen to decline even when adaptation took place, possibly due to the time taken to adapt or because the gains obtainable by varying were small. No trends in the externally set co-evolution crossover probability were found, but as adaptation does not occur reliably, this is not surprising.

Should adaptation of operator probabilities be tackled by a different approach? COBRA (COst Based operator Rate Adaptation) [1], periodically re-ranks operator probabilities according to a defined measure of operator performance. Good results have been obtained by COBRA for timetabling problems and preliminary results for the problems studied here are promising.

Encoding of operator parameters was found to be more successful: adaptation was observed more often, however performance was still found to be degraded.

Similar work [5] adapted the standard deviation of a Gaussian mutation (as in evolution strategy approaches). That paper concluded that, in many cases, adaptation by co-evolution lead to improved GA performance. This suggests, along with the results here, that binary-encoded problems have less potential for this approach than real-coded problems.

A much more detailed account of the work in this paper can be found in [8].

5 Acknowledgements

We would like to express our gratitude to the Engineering and Physical Sciences Research Council (EPSRC) for their support of Andrew Tuson via a research studentship 95306458.

References

[1] D. Corne, P. Ross, and H.-L. Fang. GA Research Note 7: Fast Practical Evolutionary Timetabling. Technical report, University of Edinburgh Department of Artificial Intelligence, 1994.

[2] L. Davis, editor. *Handbook of Genetic Algorithms*. New York: Van Nostrand Reinhold, 1991.

[3] S. Forrest and M. Mitchell. Relative Building Block Fitness and the Building Block Hypothesis. In L. Darrell Whitely, editor, *Foundations of Genetic Algorithms 2*. San Mateo: Morgan Kaufmann, 1993.

[4] D.E. Goldberg, B. Korb, and K. Deb. Messy genetic algorithms: Motivation, analysis, and first results. *Complex Systems*, 3:493–530, 1989.

[5] R. Hinterding. Representation and Self-adaption in Genetic Algorithms. In *Proceedings of the First Korea-Australia Joint Workshop on Evolutionary Computation*, Sept 1995.

[6] J. Horn, D. E. Goldberg, and K. Deb. Long Path Problems. In Y. Davidor, H.-P. Schwefel, and R. Männer, editors, *Parallel Problem Solving from Nature, PPSN III*, pages 149–159. Springer Verlag, 1994.

[7] E. Taillard. Benchmarks for basic scheduling problems. *European Journal of operations research*, 64:278–285, 1993.

[8] A. L. Tuson. Adapting Operator Probabilities In Genetic Algorithms. Master's thesis, Department of Artificial Intelligence, Univeristy of Edinburgh, 1995.