Applications of genetic algorithms

Peter Ross and Dave Corne
Department of AI, University of Edinburgh
80 South Bridge, Edinburgh EH1 1HN

{peter,dave}@aisb.ed.ac.uk

Abstract

Genetic algorithms have been applied to a very wide range of practical problems, often with valuable results. This paper surveys just a few examples, to illustrate the diversity of approaches and to point to some of the considerations that have proved important in making applications successful. Because GAs provide a fairly comprehensible way to address a wide range of difficult engineering and optimisation problems producing good if not optimal results, it seems that the technology is finding its way into real-world use much more easily than, say, expert systems did.

University of Edinburgh, Department of Artificial Intelligence, Genetic Algorithms Research Group, Paper No. 94-007

Submitted for a special issue of the AISB Quarterly on Evolutionary Computation, edited by Terry Fogarty

Applications of genetic algorithms

Peter Ross and Dave Corne
Department of AI, University of Edinburgh
80 South Bridge, Edinburgh EH1 1HN

{peter,dave}@aisb.ed.ac.uk

Abstract

Genetic algorithms have been applied to a very wide range of practical problems, often with valuable results. This paper surveys just a few examples, to illustrate the diversity of approaches and to point to some of the considerations that have proved important in making applications successful. Because GAs provide a fairly comprehensible way to address a wide range of difficult engineering and optimisation problems producing good if not optimal results, it seems that the technology is finding its way into real-world use much more easily than, say, expert systems did.

1 What is an application?

Over the past few years genetic algorithms (GAs) have been a hot topic in computer science and AI, and the technology has found its way into a surprising range of commercial applications. The word 'applications' can mean different things. When academics talk of an application of a certain software idea they typically refer to a prototype showing how that idea might be applied to a specific practical area. Usually some publication describes the prototype and illustrates its use with the aid of one or more examples chosen more for their comprehensibility than for their realism. When business people talk of an application they typically refer to some software that they have come to depend upon. Unfortunately such applications are often not publicised, for various reasons: they do not wish to lose a commercial advantage, or they have no time to spare for such marginal activities as non-profitable writing of papers, or because they are shy of revealing their state of technical awareness, or simply because dependable technology can seem too routine to be worth going to the trouble of describing.

The upshot is that, just as with expert systems, there are many academic publications describing applications of GAs but few describing proven examples in everyday use. Nevertheless we have heard privately of many such working systems. An example is in a strip steel mill's annealing plant, helping a crane operator to decide where to stack rolls of steel before annealing, how to assemble batches on annealing beds and how to regroup annealed batches in order to assemble complete orders ready for dispatch. In other cases, much activity with GAs in a particular application area is rather under-represented by just a few publications. This seems to be the case in timetabling, where the GA is used to satisfy hard constraints and optimise over various objectives in exam and/or course timetabling problems. The variety of such anecdotal evidence convinces us that GA technology is turning out to be a remarkable success story for the research funding agencies that backed the initial research.

In this paper we describe a few illustrative examples that we hope will convey some sense of the variety of GA applications, but it is impossible to provide a comprehensive survey, especially in the space available. We will also try to summarise just why GAs seem to be so attractive for so many purposes. The best-known introduction to GAs is still [9]; a different introduction and descriptions of a number of applications can be found in [13]. A report by Jarmo Alander of the University of Vaasa, Finland, gives a multiply-indexed bibliography of 2521 GA publications between 1957 and 1993 inclusive and is a useful place to start should you wish to track down other applications. The report is available over the net as garbo.uwasa.fi:pc/research/2500GArefs.ps.gz. Much more (papers, sources, etc) can be obtained through the Evolutionary Computation repository (ENCORE) at `ftp.cs.cuhk.hk:pub/EC` in Asia, `ftp.germany.eu.net:pub/research/softcomp/EC` in Europe, `alife.santafe.edu:pub/USER-AREA/EC` in USA, or `ftp.dcs.warwick.ac.uk:pub/mirrors/EC` in the UK.

2 Terminology

A *chromosome* is an array or sequence of *genes*; the position of a specific gene is called its *locus*. The values that a gene might possibly have are called its *alleles*. Early GA work focussed strongly on GAs in which the alleles were just 0 or 1; now there are many examples in which the allele set is a finite set of integers, or an interval of real numbers, or even a set of elaborate data structures. In some GAs the set of alleles varies between loci.

A GA's job is to discover ever-fitter chromosomes until a known optimal fitness is reached or until the GA can do no better. Everything thus depends on the representation chosen and on how the *fitness function* expresses the quality of a chromosome as the representation of a good solution. In the following sections we shall encounter a range of such representational ideas.

3 Optimisation of networks

One of the earliest GA applications is to the control of a gas pipeline in steady-state and transient conditions, described in [9]. Gas pressure naturally drops down the pipeline, but rises through compressors on the way; the job is to maintain adequate pressure at delivery points while minimising energy used in compression and satisfying various other constraints. There is also a need to report probable leaks based on pressure readings, without too many false alarms. Because this example is very well known we shall not describe it further here. However, there are a number of broadly related applications. Davis and Coombs describe an application to the design of corporate communication networks that link various widely-separated sites [14, 17]. From a customer's requirements Bolt Beranek and Newman Communications Corporation would produce a final topology for the communications network, conforming to requirements such as that any site should be reachable by more than one route so that no site could be isolated by a single link failure. Figure 1 shows a very crude example in a highly stylised USA. When developing this topology the design team would start by assuming that all links were of very high speed and reliability, but such links are very expensive. Thus, after settling on a topology, an important step was to discover how much the link speeds and therefore costs could be reduced while still meeting customer requirements. This is a much harder task than it appears – for example, *increasing* a link speed might degrade the performance of the whole network because statistical routers might then prefer to send traffic down the faster links and so create bottlenecks. The design team might typically spend several weeks using their expertise to tweak the set of link speeds and types, each time evaluating a network by simulating it on a Symbolics Lisp machine. Davis and Coombs found that a GA could be used for the same task, typically finding a solution that was a few percent better (representing a significant cost saving) than the human design team's best effort and in a much shorter time. In their GA a chromosome was a list of link speeds, one for each link in the network. They had to develop some application-specific genetic operators; this GA was one of the earliest which included an operator to repair a chromosome that violated certain hard constraints. The evaluation of a chromosome was essentially by statistical simulation of the network which it represented, and included such aspects as checking the performance of the network assuming the failure of a single link, doing this for every link in turn. Such a simulation is extremely time-consuming if done exhaustively, so their GA did a partial evaluation instead, re-evaluating the penalty for just a few link failures at each generation and otherwise using values obtained in simulations conducted for earlier generations.

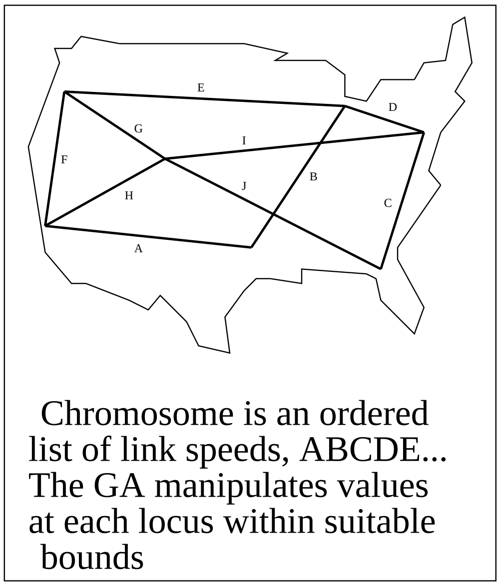

Figure 1: A simple comms net topology

Although the representation used is simple, this application illustrates an important point, namely that few real-world applications conform exactly to the idealised model so often presented in introductory accounts that has binary genes, an easily-computed and deterministic fitness function and simple domain-independent operators. A much more recent but related application is to pressure regulation in a water distribution network, described in [5]. Leakage losses from underground pipes obviously increase dramatically with pressure, so it is desirable to keep the pressure low in all parts of a network while still ensuring that minimum pressures are still provided to customers. A simple method of control is to open or close valves in various links of the network, thus adjusting the topology of the flow. The GA's job in this case is thus to find a suitable topology which minimises pressure losses while meeting customer requirements. Each chromosome represents a graph of a possible topology in a fairly direct way, as two sets: a spanning tree and a co-tree of additional links. A generational strategy is used, and parents produce two children using a specialised recombination operator that guarantees to produce only feasible networks. Fitness is determined by a full hydraulic analysis of the represented topology. A basic comparison with enumerative methods suggests that using a GA is worthwhile for this problem, but since most randomly generated topologies will commit the sin of leaving some customers without access to water it may be that there are smarter versions of conventional techniques that will compare well with the GA or outperform it.

In some cases though, binary-valued genes are a natural and good choice, but such an application almost never conforms otherwise to the common 'introductory' model. An example concerns an expert system called CAP, used by the Bonneville Power Authority to decide on the causes of complicated alarm conditions in their microwave communications systems. Potter et al [20] describe an upgrade, CAP-II, which diagnoses multiple faults using a GA to find the best combination of faults that would explain the symptoms. This contrasts with the much better-known technique of using an ATMS, as exemplified by de Kleer and Williams' General Diagnostic Engine. In the GA within CAP-II each chromosome is a bit string indicating which faults might be present, and a form of probabilistic causal net is used to assess the quality of that set as an explanation. A problem-specific operator called 'engineered conditioning' is used to improve the best chromosome in each new generation. This is a variant of hill-climbing; the local search is limited to chromosomes attainable by changing a single 1 to 0 or vice-versa, or a single swap of a 1 and a 0. Experiments show that this use of a GA considerably improves upon the reliability of the earlier expert system, which also used a causal net to produce a sorted list of possible explanations.

4 Optimisation of schedules

[4, 16] describe the practical use of GAs for exam timetabling. Here the task is to place each of a set of exams into one of a fixed number of timetable slots, so that no student has to be in two places at once and so that no student has exams in consecutive slots nor too many in any one day, certain exams must be excluded from various slots and so on. The representation is direct: there is one gene per event, and the allele set is just the set of timetable slots. Thus most random chromosomes will violate many constraints (we have tried generating hundreds of thousands at random – none were useful!). Fitness is determined by a penalty function approach, which has proved to be adequately insensitive to the actual penalty awarded for each kind of constraint violation. The GA is thus pretty conventional in form, except that we have found it very beneficial to use a form of mutation which implements a gentle pressure towards improving the timetable. Using mutation which applies strong pressure, by making the single change which most improves quality, turns out to be highly counterproductive (with hindsight we can see why). Although several other approaches to GA-based timetabling have been described in the literature we mention our own here because the Department of AI uses it in earnest. Extensive testing has shown that it produces vastly better timetables than humans used to, much faster – minutes rather than weeks. For example, for the 1993-94 data set the GA could find a solution in which just a single student had the inconvenience of consecutive exams, and it found such a solution (not always the same one) in about 50% of its runs. Moreover, separate theorem-proving methods have shown that this is the best that can be done with the given data. We have applied similar methods to generate lecture timetables too; in this case a chromosome specifies start time and location for each lecture and there are extra penalties for violating such constraints as room capacity, inter-lecture travel constraints and so on. More conventional timetabling algorithms, such as those based on graph-colouring, are faster if the constraints are merely of the "these events must not clash" kind; the GA is preferable when there are more complicated constraints too.

In contrast to such a direct representation, Paechter et al [15] has solved hard timetabling problems at Napier University with an indirect chromosome representation; here, instead of an allele indicating an exam or lecture's 'absolute' timeslot, alleles are instead interpreted relative to a dynamically updated list of possible times or places that the given event can inhabit without violating certain constraints. Such a method slows down the fitness function considerably, but also cuts down the search space a lot. We have used such an indirect representation in a GA that solves job- and open-shop scheduling problems (eg [10]). In such problems there are various resource bottlenecks, usually termed machines because such problems were originally phrased in terms of management of throughput in an old-fashioned engineering factory. Each job consists of tasks, where a task means a visit to a certain machine for a certain amount of time – in job-shop problems the order of such tasks for any given job is fully specified, in open-shop problems it isn't. The chromosome is a sequence of simple instructions to a schedule builder. The schedule builder maintains a circular list of those jobs having some tasks that have not yet been slotted into the growing schedule, plus of course information about precisely which tasks have not yet been placed. Having a circular list means that the allele set can be of fixed size, and that all possible chromosomes will represent legal schedules. The schedule builder still has choices to make, about which task to consider next or how to decide where to place it in the schedule. If the builder were to use a fixed heuristic for the purpose it could always be possible to tailor-make a problem for which that heuristic did very poorly. We therefore let the GA evolve the choice of heuristic. A chromosome consists of pair of genes: one gene indexes into the circular list to find a job with outstanding tasks, and the first of that irrelevantly-ordered list of tasks is taken. The second gene indexes into a fixed array of heuristics, and the chosen one is used to place the task into the schedule. This approach works remarkably well; having tried it on a large number of publicly available benchmarks, the GA can in many cases produce equal or better results than other known scheduling algorithms! Indeed, in a few cases this GA has been able to attain the previously-unattained theoretical lower bound for the overall duration of the schedule. The GA performs similarly well against published results when a range of other measures of schedule quality are to be optimised instead; it is usually not the fastest algorithm, but the speed seems acceptable and the quality of results is very good indeed.

A number of other GA applications also use such *order-based* representations, or variants on the idea, in which the values of earlier genes in the sequence affect the meaning of later ones – see for example [6] and a related chapter in [13]. This interdependency of gene values, generally termed *epistasis*, was thought until recently to be worth struggling to avoid not least because it invalidated the schema theorem account of how a GA worked. However, an order-based representation often works (but not always) and a range of general operators have been developed for use with such representations. A key aspect of this kind of technique is the natural opportunity it offers to hybridise a GA with a heuristic (or greedy) method. Broadly, a heuristic rule or algorithm builds a good schedule in a way sensitive to factors such as the order in which tasks are placed; the GA searches the space of variations of such factors. Thus, the combined and complementary power of the hybridised approach often outweighs any supposed disadvantages of the underlying representation.

Constraint satisfaction problems, such as many of those occurring in timetabling and scheduling, are normally transformed into constrained optimisation problems for a GA application. A GA relies on the fitness function's conveying information about how far a possible solution is from any state of desired perfection, for which purposes the yes/no nature of constraint satisfaction can be unduly limiting. The most widely-used method of doing this is the kind of *penalty function* approach outlined above for timetabling. [1] reports some empirically-discovered guidelines for the design of such a function. It seems not to matter much if a high proportion of possible chromosomes represent infeasible solutions to the satisfaction problem, provided that the fitness function conveys a strong indication of the distance from feasibility. But of course GAs are as bad as everything else at needle-in-a-haystack problems.

5 Some engineering applications

The penalty function approach can be applied to some important engineering problems. For example, [2] use a GA for a version of the channel routing problem in the area of VLSI design. Stated simply, a rectangular board has a row of inputs along its bottom edge and a row of outputs along its top edge. Various connection nets need to be established, say N of them in all: in each such net, a given set of inputs and a given set of the outputs are all connected. The routine way to do this is to have vertical links from the inputs and outputs, all on one side or layer of the board, all connecting to a stretch of horizontal conductor on the other side or on a second layer. Figure 2 shows a very simple example. The conductor may not occupy the full width

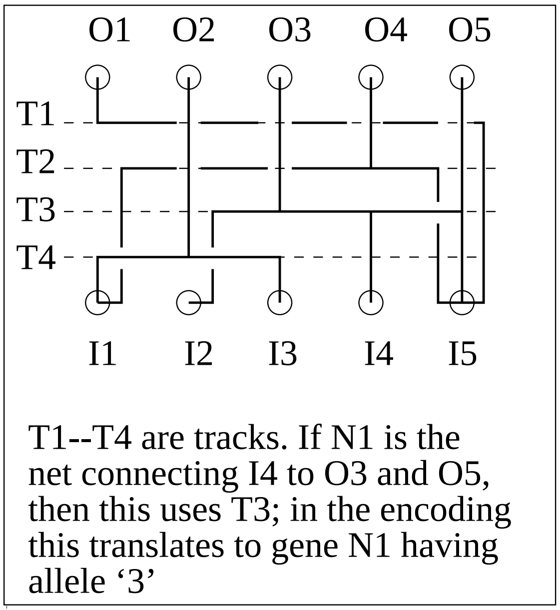

Figure 2: A simple channel routing example

of the board, so there may not need to be as many horizontal tracks as there are nets; clearly the length and location of the conductor on a track is fully determined by the leftmost and rightmost of the inputs and outputs involved in that net. The representation in the GA is direct: gene K specifies which of up to N horizontal track the K^{th} net should use, so that most chromosomes will be infeasible. A special-purpose mutation operator is used but crossover is one of the standard ones, 2-point. On benchmark problems from the literature on the subject, the GA performed well, finding the smallest number of tracks required in each case. A rather different application is to the design of low vibration rotors for helicopters. Boeing Helicopters have an elaborate simulation program that, given such information as the mass distribution and elasticity of the blade, external airloading and so on, computes the vibratory response of the rotor blades during a single revolution. The function effectively implemented by this simulator is very complicated and also noisy, so that gradient-based methods of searching for the optimum parameters are not really practicable; and there are huge numbers of local minima. The parameters at the optimum need to specified to a considerable degree of numeric accuracy. If they were represented to the desired precision right from the start, the search space would be prohibitively large even for a GA. Thus a staged approach is used. Initially the parameters are represented with low precision, so that the GA can find the right general area of the space. When this GA converges, the low-precision solution it has found is taken as the new origin of parameter co-ordinates for a slightly higher-precision GA-based search. The process is repeated until the desired precision is reached. Thus, in a sense the GA is being used to conduct local searches in a form of coarse-grained hill-climbing.

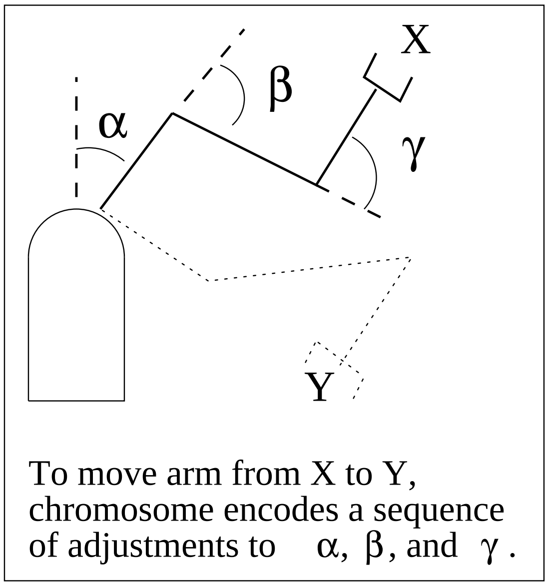

Figure 3: Robot effector co-ordinates

The general idea is further discussed in [3] although it had been used earlier by Davidor [21] in a GA for optimising the trajectory of a robot arm. Robot arm motion is, at a low level, expressed in terms of a sequence of sets of parallel adjustments to effector positions – see figure 3.

For example, the first set might specify increasing the shoulder angle by 5.5 degrees while simultaneously decreasing the forearm angle by 3.9 degrees and increasing the wrist angle by 8.3 degrees. Clearly, if the sequence consists of many such sets of modestly-sized changes the overall motion will be very slow, but if there are just a few sets of larger-scale adjustments the end-effector is likely to move along a very complicated curve – but straight-line motion of the end-effector is often what is preferred, for example if the robot is carrying a heavy weight. In this GA application, a chromosome is a variable-length sequence of sets of effector adjustments. Because chromosome lengths vary even within a population, specialised recombination and mutation operators are needed. For example, once crossover points have been chosen in one parent, corresponding crossover points are chosen in the other parent so as to minimise any discontinuities of end-effector position in the children. As before, the space of possible trajectories is colossal, so the strategy of iteratively increasing the precision is used. Experimental results reported in [21] suggest that it works well.

6 Rule discovery

Grefenstette's SAMUEL system [11] tries to discover rules by which a slower but more manoeuvrable aircraft can evade a faster but less agile missile until the missile runs out of fuel. Figure 4 shows the start of a chase. The missile becomes somewhat more agile and slower as its fuel load drops. The aircraft can sense the range, speed, heading and bearing of the missile although the sensors themselves are noisy. Rules are of a fairly uniform sort, such as to turn right by 90 degrees if the missile's range, heading and bearing lies within certain intervals; a rule can thus be assumed to have a fixed number of conditions, some of which might be vacuous. A chromosome in this GA is essentially an ordered, variable-length set of such rules. There are specialised operators for recombination, mutation and if necessary repair to ensure that a rule set always contains some applicable rule. Fitness evaluation is by simulation; a rule set earns a reward according to how long the aircraft typically manages to evade the missile, and the reward is then shared out among the rules involved. A recent re-implementation of this work ([19]) suggests that the trickle-down scheme known as the 'bucket brigade algorithm' used in classifier systems is ineffective in this system; instead, the 'profit-sharing plan' in which all rules that fired share the reward equally seems to work well. The computational requirements are considerable – on a typical workstation it can take hours to evaluate the initial population!

There are successful commercial applications of the technology. For example in the Dutch airline KLM, aircrew each have a seniority rating. Whenever a vacancy appears qualified aircrew who are free to do so can bid for it, and the most senior bidder wins; this process is determined by a collective labour agreement under Dutch law. However, each job carries a commitment period with it. For example, a second officer on a 747/400 might bid for a vacancy as second officer on a 747/300 – but the winner must then stay in that post for its specific commitment period of six years. It was therefore very useful for KLM's forward planning to be able to anticipate pilots' bidding behaviour in this process; as one person moves up their previous post falls vacant, and such a chain of vacancies with awkward commitment periods could then lead to a troublesome shortage of suitable staff for certain kinds of post. Syllogic, a Dutch software firm, worked with KLM to produce a set of rules that would predict pilots' final bidding behaviour. OR and expert systems approaches to the problem were rejected, and after considering GAs and neural nets they opted for a GA approach. Available historic information was used in the fitness function, producing a system that was around 80% right on the historic data. The system was officially tested against actual pilot biding behaviour for around six months from August 1992, and over this period was about 90% right. This was highly satisfactory from the point of view of KLM's planning. However, to the best of our knowledge little of the specific details of this system have been published.

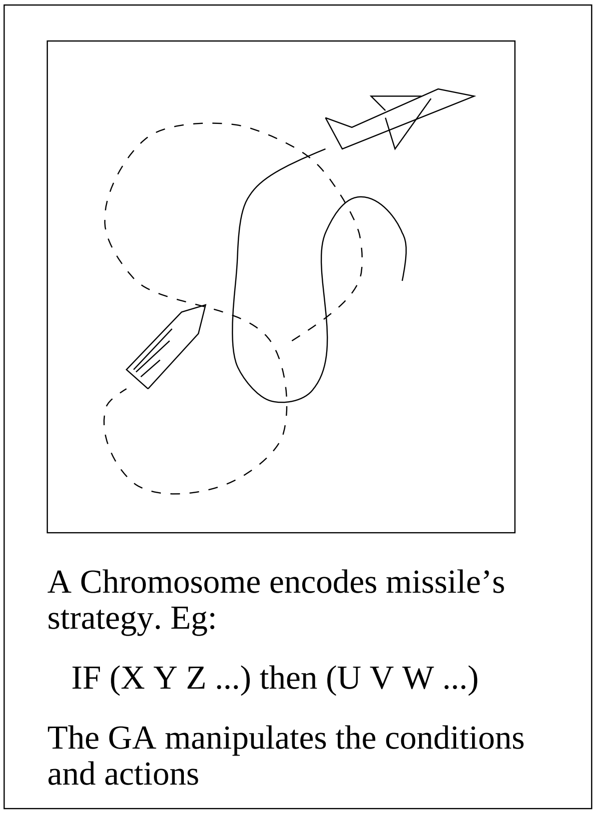

Figure 4: Missile chasing plane

7 Some reflections

There are many detailed academic publications on other applications such as vehicle routing, pallet loading, fitting functions to noisy data, database query construction, design of laminates and so on; there is even a separate conference proceedings and mailing list for those interested in using GAs to produce neural nets and other GA/NN hybrids. However, the interested reader is referred initially to the various Proceedings of the International Conferences on GAs; this biennial series happens in odd-numbered years, and there have been five so far – the sixth will be in Pittsburgh in 1995. It is interesting to observe that the much-described and analysed classical model of a GA due to Holland and later Goldberg, namely a generational model with fixed-size bit-string encoding and simple problem-independent recombination and mutation, has not been so widely used in practice. Instead, developers have made sensibly pragmatic decisions about the design of their GA. For evident reasons, useful applications of GAs tend to surface just in those cases where a promising representation has been found for the problem. If the problem is hard, and if a reasonable chromosome encoding is naturally apparent (or arises from hard thinking), then it is worth giving GAs a try. Success then often depends on just how reasonable the representation is. The question of what 'reasonable' means comes down to a combination of important factors. The chromosome encoding should ideally cover all good candidate solutions, and eliminate all or most useless or invalid candidates. It should ideally have considerable continuity, in the sense that the quality of a chromosome is a decent clue as to the quality of other encodings close to it in genotypic space. It is also often important that distinct aspects of the problem should be 'distinct' in the encoding (for example, different parameters represented by different genes), and so on . . . . Often, all these factors and more will be considered, at least implicitly.

It is easy to understand why parameter optimisation problems, for which reasonable direct chromosome encodings are simply and readily designed, are the basis of many attempted GA applications. Where this direct approach fails to work, the reason is usually one of two: either the constraints make infeasible or useless chromosomes too common and hence the direct encoding is unreasonable in one or more ways, or it simply cannot reasonably be seen as a parameter optimisation problem.

The first difficulty is addressed by the kind of indirect encoding techniques discussed above, which find use in many kinds of scheduling application. Here, the unhelpful landscape of the direct encoding is replaced by a more fruitful space, but usually there is much redundancy introduced; the latter seems empirically to be far less a problem than the former. The second difficulty is evidently addressed differently according to what the problem actually is, but a very common and important class of application types worth singling out is that of modeling, or system identification problems. Here, there is a need to model the way components of a system interact to produce a set of known results given known inputs. The space of possible interactions is much like (if not exactly like) the space of possible mathematical formulae. This 'encoding need' is well met by genetic programming (GP) [12], in which a chromosome is an expression tree, rather than a string, with function and terminal symbols chosen from a given initial set. GP has indeed been used to address many kinds of modeling and system-identification problems. For instance, [8] describes a commercial tool named OMEGA that employs GP (although it is called a GA in the paper) to generate models that fit given data. It was originally tested on real, large loan-application data sets from certain UK banks, and it outperformed the methods then in actual use; other tests suggest it can outperform neural nets and rule induction methods for similar tasks. OMEGA is a spin-off of an EC-sponsored ESPRIT III project. In other such modeling or 'model-fitting' applications, the model may already be tied down enough (eg: to the need to find values for a given set of parameters) to obviate the need for a GP approach, but remains too hard for an analytical solution. An example is in adapting a mathematical model to fit samples taken from gold and silver assays made over many years from a copper mine in Europe. To find out which bits of the ore body are worth mining further, lots of extensive drilling needs to be done; the problem is to find a good plan for where to drill which maximises the predictive value of the data obtained, while minimising the considerable expense of wasted drilling operations. Everett has found an appropriately configured GA to be quite successful at this task [7].

In GA work, academics sometimes have an unreasonable preoccupation with the search for optimal performance, tweaking parameters or algorithms in order to squeeze a little more out. It seems to us that this is often misguided. The strength of GAs is that they are usually straightforward to code and that they often work very well even if they don't find a global optimum. GAs do not provide a magic key that will unlock the P=NP problem; hard problems remain hard for GAs too, even though there is growing evidence that only a relatively few members of the much-studied classes of NP-hard problems are genuinely hard, whether for theorem provers and the like or for GAs – most are relatively easily solved or are easily found to be unsolvable (though [18] even show by example that basing fitness on the estimated difficulty can significantly improve the GA's performance). Parameter and algorithm tweaking is also a part of many searches, but the effort involved is often concealed from the reader. If a GA is to be an acceptable way of tackling some problem then it ought to be possible to find a suitable GA configuration using no more than simple and short hill-climbing in the space of possible algorithms and parameters. In many cases, fortunately, this seems to be true so that GAs appear to be finding their way into real-world use much more easily than, say, expert systems did. There is also an encouraging trend towards comparing the performance of GAs with the performance of other algorithms in a properly scientific way, so that GAs can stand on their own merits rather than on the attractiveness of an evolutionary metaphor.

Acknowledgements

Dave Corne is supported by an EPSRC research project, no. GR/J44513: "Genetic algorithms for generic timetabling problems".

References

[1] A.E.Smith and D.M.Tate. Genetic optimization using a penalty function. In S.Forrest, editor, *Proceedings of the Fifth International Conference on GAs*, pages 499–503. Morgan Kaufmann, 1993.

[2] A.T.Rahmani and N.Ono. A genetic algorithm for channel routing problem. In S.Forrest, editor, *Proceedings of ICGA-93*, pages 494–498, San Mateo, Ca., 1993. Morgan Kaufmann.

[3] C.Lin and P.Hajela. Genetic search strategies in large scale optimisation. In *AIAA Structures, Structural Dynamics and Materials Conference*, 1993.

[4] Dave Corne, Hsiao-Lan Fang, and Chris Mellish. Solving the module exam scheduling problem with genetic algorithms. In Paul W.H. Chung, Gillian Lovegrove, and Moonis Ali, editors, *Proceedings of the Sixth International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, pages 370–37. Gordon and Breach Science Publishers, 1993.

[5] D.A.Savic and G.A.Walters. An evolution program for pressure regulation in water ditribution networks. Technical Report 94/15, University of Exeter Centre for Systems and Control Engineering, 1994.

[6] D.Whitley, T.Starkweather, and D.Fuquay. Scheduling problems and the travelling salesman problem: the genetic edge recombination operator. In J.D.Schaffer, editor, *Proceedings of the Third International Conference on GAs*, pages 133–140. Morgan Kaufmann, 1989.

[7] J. E. Everett. Model building, model testing, and model fitting. In Peter Stampoultzis, editor, *The Application Handbook of Genetic Algorithms*. CRC Press, 1994.

[8] E.W.Haasdijk, R.F.Walker, D.Barrow, and M.C.Gerrets. Genetic algorithms in business. In J.Stender, E.Hilleldrand, and J.Kingdom, editors, *GAs in Optimisation, Simulation and Modelling*. IOS Press, Amsterdam, 1994.

[9] David Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, 1987.

[10] H-L.Fang, P.M.Ross, and D.Corne. A promising hybrid ga/heuristic approach for open-shop scheduling problems. In A.G.Cogn, editor, *Proceedings of ECAI-94*, pages 590–594. John Wiley, 1994.

[11] J.J.Grefenstette. Strategy acquisition with genetic algorithms. In L.Davis, editor, *Handbook of Genetic Algorithms*, pages 186–201. Van Nostrand Reinhold, 1991.

[12] J. Koza. *Genetic Programming: on the programming of computers by natural selection*. MIT Press, Cambridge, MA, 1992. Contains clear description of a basic Genetic Algorithm as well as a detailed description of Genetic Programming.

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

[14] L.Davis and S.Coombs. Genetic algorithms and communication link speed design: theoretical considerations. In J.Grefenstette, editor, *Proceedings of the Second International Conference on Genetic Algorithms*, pages 252–256, Hillsdale, New Jersey, 1987. Lawrence Erlbaum Associates.

[15] B. Paechter, H. Luchian, A. Cumming, and M. Petruic. Two solutions to the general timetable poblem using evolutionary methods. In *Proceedings of the IEEE Conference on Evolutionary Computation*, 1994.

[16] Peter Ross, Dave Corne, and Hsiao-Lan Fang. Improved evolutionary timetabling through delta evaluation and directed mutation. In Y. Davidor and H-P. Schwefel, editors, *Parallel Problem Solving from Nature III*. Springer-Verlag, 1994. to appear.

[17] S.Coombs and L.Davis. Genetic algorithms and communication link speed design: constraints and operators. In J.Grefenstette, editor, *Proceedings of the Second International Conference on Genetic Algorithms*, pages 257–260, Hillsdale, New Jersey, 1987. Lawrence Erlbaum Associates.

[18] S.H.Clearwater and T.Hogg. Exploiting problem structure in genetic algorithms. In *Proceedings of AAAI-94*, pages 1310–1315, San Mateo, 1994. Morgan Kaufmann.

[19] S.Reeks. Genetic algorithms for strategy acquisition. B.Sc. Final Year Project Report (unpublished), 1994.

[20] W.D.Potter, J.A.Miller, B.E.Tonn, R.V.Gandham, and C.N.Lapena. Improving the reliability of heuristic multiple fault diagnosis via the ec-based genetic algorithm. *Applied Intelligence*, 2:5–24, 1992.

[21] Y.Davidor. *Genetic Algorithms and Robotics: A Heuristic Strategy for Optimization*. World Scientific, Singapore/New Jersey/London, 1990.