An Introduction to Evolutionary Computing in Design Search and Optimisation

A. Keane
Lecture notes produced by A. Rogers

Department of Mechanical Engineering
University of Southampton
Southampton SO17 1BJ, UK
E-mail: *andy.keane@soton.ac.uk*

Abstract. Evolutionary computing (EC) techniques are beginning to find a place in engineering design. In this tutorial we give a brief overview of EC and discuss the techniques and issues which are important in design search and optimisation.

Keywords
Evolutionary computing, design search, optimisation, problem solving environment

1 Introduction

The purpose of this introduction is to present, to a wide audience, some of the ideas which are important in the application of evolutionary computing (EC) algorithms to engineering design. This leads on to an area known as *design search and optimisation* where evolutionary computing techniques are beginning to find real applications.

Perhaps the first question to ask is: why do we do this at all? The general answer is that life is becoming more computationally complex and in the field of engineering we are presented with an increasing number of computationally complex problems to solve. The computational complexity of optimising these systems challenges the capabilities of existing techniques and we seek some other heuristics. When we look around at the real world, we see many examples of complex systems, ourselves included, which have arisen through evolution. This naturally leads to some desire to emulate evolution in our attempts at problem solving.

As an example of a computationally complex problem, we can consider designing aircraft wings – an area from which several of the examples in this tutorial will be drawn. Alternatively, we may be trying to schedule some process in a factory or simply trying to fit curves to data. In some senses, all these things are about design, and by design, we mean the idea that we want to create something. The whole point to evolutionary computing is the creation of designs, particularly in the domain of complex problems and environments.

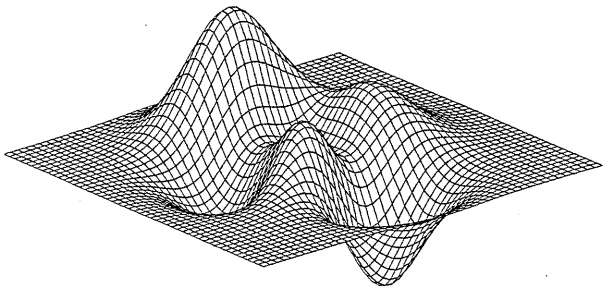

Figure 1. Example of an optimisation landscape

It is useful to briefly consider what design is, how it is done and where evolutionary computing fits in. We are all surrounded every day by extremely complicated technology which we generally take for granted. As an example, consider the cars we drive. Should we have a crash, the airbags will explode and the car will crumple safely around us because someone has thought about and modeled all the processes and components involved in that crash. To do this nowadays, automotive engineers use high-performance computer networks which are highly developed into sophisticated environments.

Designing complex things such as cars requires us to put lots of things together. It requires us to know something about what it is we are trying to design, to have some way of interfacing it to the user and perhaps some type of search and optimisation routine with resource management. The whole thing may be called a *problem solving environment* (PSE) and it is into this environment that evolutionary computing techniques fit.

In summary, designers in the broadest sense – simply people trying to create something – increasingly see the need for a PSE. Ideally, these PSEs contain some sort of search and optimisation and it is here that evolutionary computing techniques find an application. Thus, evolutionary computing is not used in isolation but as a part of a design tool which has to be of some utility.

What follows are some introductory remarks on several topics which will be expanded on later in the course of this tutorial.

Search and Optimisation. Two terms which are often used in evolutionary computing are search and optimisation. We can illustrate what these mean by considering a simple landscape – see figure 1. Optimisation is about trying to walk up a hill and hopefully, through effectively searching the landscape, walking up the highest hill.

At its simplest this is all we are trying to do. We have some landscape which we are trying to walk through. Usually we cannot see the landscape, so the whole business of evolutionary computing is trying to do this blindfold. This introduces questions

such as: how do we know we are at the top of the hill and how do we know it is the right hill?

Resources. Search and optimisation always requires some other piece of code to be plugged into it – we apply evolutionary computing to something. It is usually a piece of analysis code and we are trying to capture the intentions of the designer in this code. Given the computational complexity of the problems we are considering, computing power dominates this process. Given overwhelming computing power, most problems can be solved. With finite computing resources we have to be more careful.

Current Techniques. What is the current state of the art in engineering design? Most PSEs now include some sort of hill-climbing optimisation and parallel computation is common. Evolutionary search is known about but it is not generally in everyday use. The use of meta-computing techniques and resource scheduling are beginning to be considered as approaches to the problems of limited computing power.

Representation. Representation is one of the key issues in this work. How do we represent what it is that we are trying to design? Inevitably people represent their designs with numbers, so the method of encoding between the two is important. However, an engineer also has a selection of models which can be used which range in sophistication and cost. It is not simply a question of representation and method but representation, method, and model which seem to be the most important. Designers have aspirations, constraints, and varying and multiple objectives. They often don't know what they want or cannot express what it is exactly they want. But in general, they want robust designs.

Classical search methods have been around for at least fifty years and they are well tried. Crucially, people don't believe they are likely to lead to any more innovation. On the other hand, evolutionary computing methods have some advantages but they have a problem – computational expense. If we come up with an evolutionary computing paradigm which is really very powerful but is too computationally expensive, it will not be used.

2 A Brief Overview of Evolutionary Computing

The history of evolutionary computing goes back to the 1960s with the introduction of ideas and techniques such as genetic algorithms, evolutionary strategies and evolutionary programming [3]. Whilst all differ slightly in their actual implementations, all these evolutionary computing techniques use the same metaphor of mapping problem solving onto a simple model of evolution.

EVOLUTION		PROBLEM SOLVING
Individual	$\longleftrightarrow$	Candidate Solution
Fitness	$\longleftrightarrow$	Quality
Environment	$\longleftrightarrow$	Problem

We have a population which we want to evolve from time step t to time step $t+1$ by selection, recombination and mutation. There is some contention about which of the two, mutation or recombination, is the more powerful, with there being a whole history in the German academic community of using only mutation in evolutionary strategies. To a certain extent though, the ingredients can be mixed and matched. One is trying to introduce diversity in the designs through mutation and recombination, and then exploit this exploration through selection of better solutions: balancing the growth of ideas alongside the selection of ideas. One interesting aspect is that this balance occurs all the way through but the nature of the algorithms we use implies that there tends to be a large diversity at the beginning and very little towards the end.

Hopefully through the correct choice of operators, we achieve a balance between exploration and exploitation and our solutions steadily get better.

Although a great deal of the evolutionary computing literature concerns comparisons of one algorithm to another, some general points about the advantages of evolutionary computing techniques can be made:

- Widely applicable.
- Low development and application cost.
- Easily incorporated into other methods.
- Solutions are interpretable.
- Can be run interactively and allows incorporation of user-proposed solutions.
- Provide many alternative solutions.

Whilst they have been shown to be useful in many areas, evolutionary computing techniques have several disadvantages when compared to some other techniques:

- No guarantee for optimal solution within finite time.
- Weak theoretical basis.
- May need parameter tuning for good performance.
- Often computationally expensive and thus slow.

Apart from the computational expense, perhaps that which most directly effects the acceptance of evolutionary computing is the weakness of their theoretical basis. With no theory, we are left with a set of rules of thumb for choosing and tuning the algorithms.

3 Evolutionary Computing in Design Search and Optimisation

When actually implementing an evolutionary computing algorithm we are presented with a wide range of choices. Which algorithm should we use? Which operators should we use? How do we set the parameters? There are a number of books available which help with these questions [1–4,7]; however, when using these algorithms in engineering design we have a number of other additional issues which must be considered.

3.1 Representation

Deciding on a good representation is fundamental to the performance of evolutionary computing techniques. The algorithms work on numbers, usually binary, but we are not trying to design a string of numbers. We may be trying to design a wing, a communications network or a schedule, and in some way we have to link the two together. This point turns out to be absolutely critical to the success of the application.

It is self-evidently true that we can trade-off the ability of any representation to be compact, against its ability to represent all possible designs. This is a really fundamental decision. Do we want to describe any possible design or do we want to describe something in a small region around where we currently stand?

If we are using an evolutionary algorithm to design the wing for a jet aircraft, do we want to think about bi-planes? Probably not. Do we want to consider a wing of alternative materials? Maybe. Someone has to make these decisions and it cannot be done without some domain expertise. Experience shows you need a domain expert and an evolutionary computation expert together, as it is this critical stage which often determines the outcome of the project.

If we again consider our wing design problem suggested above, the classical aerodynamics of the wing are based on concepts of wing span, sweep, chord and camber. The interesting thing about these words is that these are ways which aerodynamicists have worked out to describe wings. They are a particular representation of a wing which is relevant when considering the flow of air over it.

An alternative would be to use the x,y and z coordinates for the surfaces. This presents the problem that whilst we can describe a wing, we can also describe virtually anything else as well. Moving down the trade-off in compactness, we are increasing the complexity of the object which we can describe and are thus increasing the domain in which the potential designs can exist. We are making the problem potentially harder and if we do not need the complexity at this level, we run the risk of making the problem almost insoluble.

The opposite danger comes from sticking too rigidly to an existing representation. Through existing design techniques the representation may have become so specific that it traps the possible solutions into a local optimum and does not have the generality to describe wing shapes which may indeed be better.

Choices across this sort of domain make a real difference and it is clear that they cannot be made without some domain knowledge.

3.2 Constraints and Multiple Objectives

Often the objectives of a design are not completely defined. An engineer will often have multiple objectives and in some sort of hierarchy. Unless this appraisal of potential designs is somehow formulated and included in the analysis code, the expectations of the engineer will differ from the designs which our algorithm is actually producing.

Constraints are another issue which is yet to be fully addressed. How do we handle a list of certain criteria which invalidate a particular design? There are a number of techniques, some of which are discussed by Eiben in this volume (page 13), and we must decide which one is most relevant.

3.3 Mutation

Mutation is critical to a lot of these results. It is often said to be there just to ensure that every part of the search space may be reached. However, successful search algorithms are often run just using mutation and selection without crossover. Indeed evolutionary strategies are predominantly used in this fashion.

The idea behind mutation is that we are making local steps in our landscape. Our choice of mutation operator and representation should mean that we are making small changes to our design and not leaping to a radically different solution.

3.4 Recombination

Recombination or crossover allows parents to pass on some of their characteristics to their children. The motivation being that we can use good parents to develop even better children. There are many ways of doing this but all involve the combination of parts of one parent with the complementary parts from another.

There is a trade-off between recombination and mutation which reflects some of the history of the trade-off between the various communities. Recombination tends to be seen as more of an exploitation operator and mutation as more of a exploration operator. Some people would tend to suggest mixing these in, so we do more exploration at the beginning and more exploitation towards the end. These are more examples of parameter and algorithm tuning which are dictated by rules of thumb rather than a solid theoretical basis.

3.5 Niching

When we initially discussed optimisation, we considered trying to find the single global optimum in the problem. In engineering applications this may not be the case and the question has to be asked whether we want the best solution or do we want to know about a number of areas in the problem space where there are good candidate solutions? Depending on our aspirations, we may want to impose some form of niching techniques to allow the population to divide into smaller sub-populations, each focused around a different part of the problem space.

3.6 Repeatability and Elitism

Evolutionary computing techniques are stochastic and thus no two runs will necessarily produce the same results. Whilst in some applications this is acceptable, it can be quite discouraging to produce a particular good solution once and never to be able to find it again.

This leads to many issues concerning the repeatability of evolutionary computing algorithms. Do we just want to get one very good solution or do we want a good solution every time? If we were doing process scheduling we might settle for the former but an automotive engineer designing a car would probably prefer the latter.

Another concern which stems from the stochastic nature of the algorithm, is that a good solution once found within the population may be lost later in the evolution. Whilst it always makes sense to keep a record of the best solutions, elitism strategies ensure that these solutions stay in the population.

3.7 When to Stop?

Having set up the representation and run the evolutionary computing algorithm, the final decision is when to stop it. Without any prior knowledge, it is impossible to tell whether the best solution has been reached. The decision most often comes down to one of time and computing resources.

4 An Example of EC Design

As an example of the techniques and problems discussed we will consider an actual case of an evolutionary computing algorithm being used to solve an engineering problem. Figure 2 shows a photograph from a NASA mission in 1987 as part of a proof of concept programme to show that astronauts could build structures in space.

One of the problems with these structures is that they tend to have very severe vibration problems. They are light regular alloy structures in an environment where there

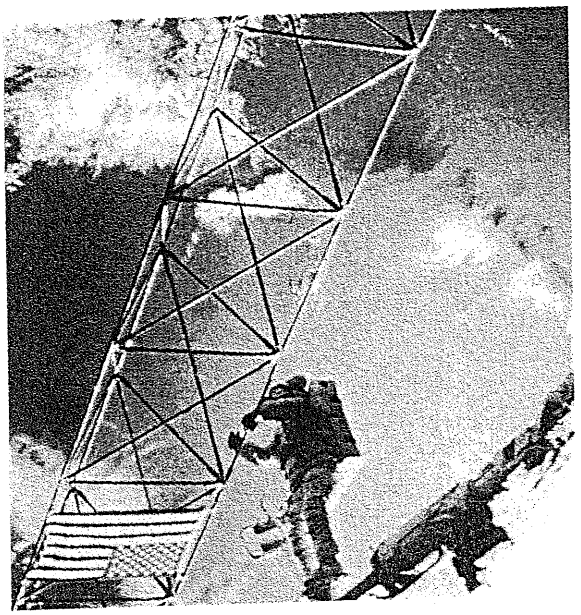

Figure 2. NASA photograph showing an astronaut constructing a boom in space

is no air to provide damping. We started to look at the vibrational characteristics of structures like these when subject to some vibrational noise at one end [6,5] . The objective was to use evolutionary computing techniques to design the geometry of the beam such that the vibrational noise does not travel through it. This is a straightforward engineering job which is actually critical to both ESA and NASA who plan to launch future missions with booms of this type fifty metres in length.

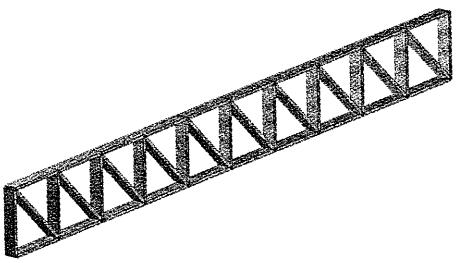

Figure 3. Diagram of simple 2D boom design

Initially a flat two-dimensional beam was considered. As in the cases discussed previously, we are combining computational expensive structural analysis routines with an optimisation technique. Figure 3 shows the initial simple two-dimensional boom design which we are trying to improve upon.

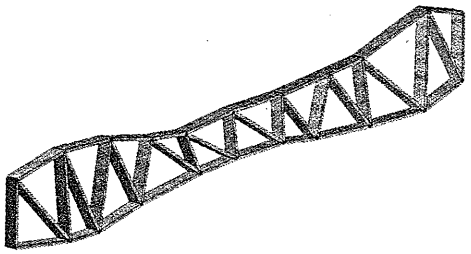

Figure 4. Final GA optimised 2D boom design

A genetic algorithm was used for the optimisation with population size of 300. It was run for fifteen generations which ideally would have been longer but for the high computational cost of evaluating each design. As it was, the complete run took three weeks of computation using eleven parallel workstations. Figure 4 shows the best design found by the GA at the end of the run.

Having obtained this design, it was actually constructed and tested for vibrational performance – a check that the improvements predicted by the analysis code could be realised in actuality. Figure 5 shows a comparison of the initial and final beam vibrational characteristics. The shaded area in the center, from 100–250 Hz, represents the range of frequencies over which the analysis code was run and is thus the range over which the response has been optimised. The interesting thing about this graph is that the vertical scale is in decades of performance and we have something like three decades of improvement in the vibration performance. It is not often that you can improve things by several thousands of percent.

Figure 6 shows the results of applying the same technique to the three-dimensional boom shown in the photograph. Obviously, whilst it has improved vibration characteristics, it is somewhat harder to manufacture. This illustrates the point that unless included in the analysis code, one cannot expect all the engineering criteria to be satisfied.

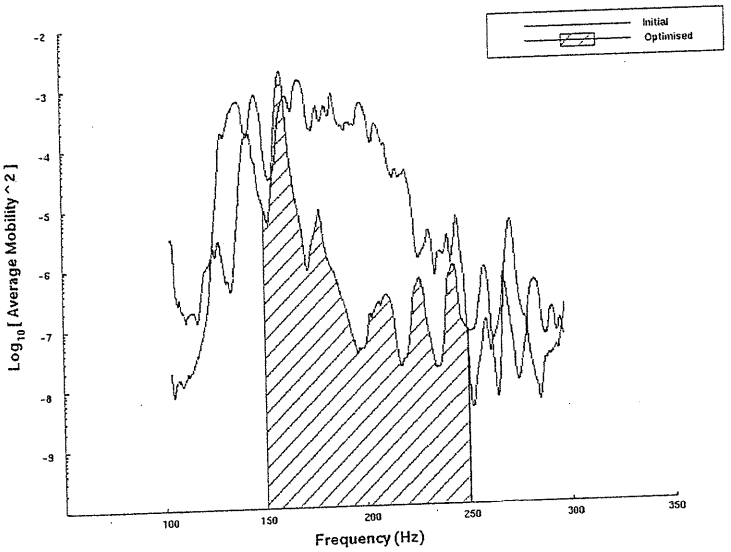

Figure 5. Comparison of vibrational characteristics for the initial and optimised 2D boom design

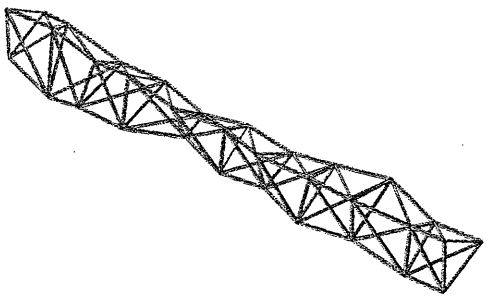

Figure 6. Final GA optimised 3D boom design

5 Conclusions

Evolutionary computing algorithms have been shown to be of great value in design search and optimisation. The concept of a problem solving environment clearly increases the utility of these algorithms but the problems of computational expense

must be addressed. This will not come through improvements in hardware as our rising aspirations continually outstrip the improvement in processor performance.

A short-term goal to address this issue, is to look at a layered approach to models in order to enhance speed and better represent what the user wants. A more medium term goal would be the use of meta-computing techniques to fully manage the available corporate computing resources. Long-term goals – possibly twenty years away – are fully agent-based environments which would offer seamless integration of optimisation techniques, computing resources, and access to corporate databases.

However this work progresses, over the next ten years evolutionary search using expensive modelling techniques will doubtless become common-place.

References

1. T. Bäck. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford University Press, Oxford, 1996.
2. L. Davis, M. Vose, and K. De Jong. *Evolutionary Algorithms*. Volumes in Mathematics and Its Applications, Vol 111. Springer-Verlag, Berlin Heidelberg New York, 1996.
3. D. Fogel. *Evolutionary Computation: The Fossil Record*. IEEE Press, New York, 1998.
4. D. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison–Wesley, Reading, MA, 1988.
5. A. J. Keane. Experiences with optimizers in structural design. In I. C. Parmee, editor, *Proceedings of the Conference on Adaptive Computing in Engineering Design and Control 94*, pages 14–27. PEDC, Plymouth, 1994.
6. A. J. Keane and S. M. Brown. The design of a satellite boom with enhanced vibration performance using genetic algorithm techniques. In I. C. Parmee, editor, *Proceedings of the Conference on Adaptive Computing in Engineering Design and Control 96*, pages 107–113. PEDC, Plymouth, 1996.
7. M. Mitchell. *An Introduction to Genetic Algorithms*. MIT Press, Boston, MA, 1996.