Evolutionary Computation Teaching at Birmingham

Riccardo Poli
School of Computer Science
The University of Birmingham
Birmingham, B15 2TT, UK
R.Poli@cs.bham.ac.uk

Abstract- This paper first illustrates the motivations of the author for teaching evolutionary computation and supervising students interested in this field. Then it describes the evolutionary computation course taught by the author at the School of Computer Science of The University of Birmingham for the last four years. Finally, the paper describes the lessons learnt from doing this.

1 Introduction

This paper is organised as follows. In Section 2, I discuss the motivations for teaching evolutionary computation and supervising students interested in this field. In Section 3, I describe SEM3A4, the evolutionary computation course I have been teaching in the School of Computer Science of The University of Birmingham for the last four years. Then (Section 4), I discuss the lessons learnt from doing this and (Section 5) I draw some conclusions.

2 Motivations for Teaching Evolutionary Computation

There are at least two motivations for teaching Evolutionary Computation (EC) courses. I discuss them in the following two subsections.

2.1 Engineering Motivations

Quite often I hear EC people say that in order to make this field successful, we really need to get industry involved. I have often asked myself why this should be important and what it means to be successful for a field.

I suppose that in places where the level of government funding is low, getting industrial funding allows people to get research fellows and PhD students, to go to conferences, to buy new beautiful toys (e.g. flashy computers), to improve their CVs, etc. However, industrialists nearly always are not very interested in long term research and they will only fund projects which are aimed at developing some practical applications which exploit the technology as it is rather then develop it further. Only occasionally long term fundamental research is funded by industry. So, industrial money nearly always means to do exploitative engineering oriented research rather than science.

Some people believe that in order to get industry interested, we (the EC scientific community) need to show them “killer” applications, i.e. applications which clearly show the superiority of an EC technique over all other AI search technologies. Unfortunately, these efforts seem to be doomed to fail. The no-free-lunch theorems tell us that without assuming prior knowledge about the problems at hand, all search techniques provide on average the same level of performance. So, the only way to develop a “killer” application is to perform a big knowledge engineering effort, i.e. to extract a lot of knowledge about a domain and to embed it into an algorithm. Nearly always it is this knowledge that allows one technique to beat other techniques, not the particular search algorithm used. So, one can develop a GA that solves a specific problem very well, if he or she is prepared to invest several months of work into this. However, the research necessary to do this is really (knowledge) engineering. Only occasionally this will lead to an important scientific advancement. So, the people trying to develop “killer” applications are steering the field even more towards engineering.

In any case, for the moment industry does not seem to be particularly impressed by what the EC community is doing: they have seen many technologies come and go. So, they are very prudent in investing money and human resources in EC-based projects, since for many projects there are other, more established, technologies they can use.

Technology transfer is very difficult even if it is properly planned. For example, technology transfer is one of the main activities of the EvoNet (the Network of Excellence in Evolutionary Computation) [1] and it is the reason why the European Union is funding it, but the results seem to come very slowly.

So, assuming that involving industry is important, how can we convince them of the potential of EC? Unless there is a breakthrough in the field *and* then an enormously successful application attracts the media, I don’t think we can. Probably the chances of these events are quite close to those of winning the national lottery.

The only reliable way I can see to spread EC technology to industry is to train people before they go to industry, i.e. to offer university courses and PhD programmes in evolutionary computation. Of course, this strategy will require quite some time to become fully successful. Undergraduate and MSc students taking an EC course today will be in industry perhaps in a year or two. This would not be too bad. However, before these people reach higher-up positions where decisions on whether to adopt a technology are made, a few more years are necessary. The same happens to PhD students doing research in evolutionary computation. It usually takes three or more years before they are in industry (unless they remain in academia), and then some more time before they can make strategic decisions. So, the benefits of this strategy in terms of industrial involvement in EC might become visible only in the long run.

2.2 Scientific Motivations

Whether becoming more engineering oriented is good or bad for EC is probably a matter of opinion. My view is that the engineering side of the coin is important, but that the scientific effort of understanding why evolutionary search works and whether there is anything really new and deep in it is much more important.

One reason for this is that the field is not entirely mature. There are so many things we don't know on how and why EC algorithms work, that it seems very risky to assume that we can apply what we have got now to any problem with the certainty of being successful. We know from the no free lunch theorems that there is nothing special in evolutionary algorithms in general. So, really the only way to be able to honestly claim that our EC algorithms are better is to identify the classes of problems for which each of these algorithms is best attuned. However, so far we have only done this in a relatively disorganised way. Given a new problem an engineer can only go and look at the literature on EC hoping to find algorithms which were reported to solve vaguely similar problems. The literature on the topic is huge, for example there are several thousands papers published on GAs [2] and more than a thousand on genetic programming [3], so quite often something will come up from this search. However, nearly always the algorithms found will not be provably better than other algorithms. At present, the maximum one can hope is that such algorithms have been compared with other more standard methods scoring well, thus suggesting that adapting them to the new problem *might* produce something that performs reasonably.

Personally, I believe that evolutionary computation will be successful when we will know exactly what it can do for us, and when all the research efforts we now are making in the field will produce new, completely different, directions for the future. So, paradoxically EC will be successful when it is ready to become obsolete as a field for scientific (not engineering) research!

Assuming that extending our understanding of EC and finding new directions is important, how can we achieve this? One possibility is that someone currently working in the field produces in the next few years a theoretical breakthrough that all explains, all predicts, all suggests. That would settle the matter. However, again, probably the chances of this happening are quite close to those of winning the national lottery.

So, again, the only reliable way I can see to extend the study of EC as a science is to train future EC researchers, i.e. to offer university courses and PhD programmes in evolutionary computation. Again, this strategy will require years to become fully successful.

2.3 How Should We Teach Evolutionary Computation?

Whatever the recipe to make EC successful is, it seems that a necessary ingredient is to train people in EC. However, in order for this to be a route to success there are two conditions to be met.

Firstly, it is necessary to teach as many people as possible. This means that many universities, perhaps the majority, should offer courses and PhD programs in EC. However, this is not the case. Four years ago John Koza [4] compiled a list of university courses on genetic algorithms. At the time, around 30 people replied sending a description of their courses. Even assuming that not everyone replied to Koza's call for syllabi, 30 courses are not many compared with the number of universities worldwide (only in the UK there are nearly 100 universities). To see whether the situation had changed, towards the end of 1998, Xin Yao and myself emailed a similar invitation to several mailing lists and newsgroups interested in EC. We received only around 40 course descriptions [5]. Again, perhaps not everybody found the time to send information, but we have no reasons to believe that there are more than a few tens of university courses on EC worldwide. The situation seems slightly better in terms of PhD programmes in EC. For example, according to [6], for PhDs in genetic programming alone, in August/September 1998 the situation was: about 30 PhD theses completed and 30 students still working towards their PhD. However, the ratio between PhD students in EC and undergraduates taking EC courses is probably something like one over twenty. So, it seems that EC is mostly promoted by university undergraduate/MSc courses. However, it is clear that there are not enough of them. How can we counteract this situation? Well, we (CS teachers interested in EC) need to do every effort to start new courses, even if sometimes this adds to our teaching load.

Secondly, offering many courses in EC and therefore training many people is not enough: it is necessary to teach our students well. Only in this way they will become excited by the potentialities of EC technology and will be aware of its limitations. These are extremely important. It is crucial not to overstate the case for evolutionary algorithms. If we say that EC can do something that in fact cannot do, we will prepare the stage for a massive recession of the field. This is because today's students will work tomorrow in industry. If they try an evolutionary algorithm to solve a particular problem and it does not work (either because they are not competent enough, or because we overstated what EC can do), they will start thinking that EC is no good, and so will the people who interact with them. So, bad or over-enthusiastic teaching can easily create a bad name for EC. That would not only reduce the ability of EC to penetrate industry, but quite likely it would reduce government funding thus breaking the back to the more scientific side of EC research. This is a very realistic risk: recession happened to AI in the 80's after the disappointment derived from overstating the power of expert systems, it happened in the 70's to neural networks after Minsky and Papert's book on perceptrons, etc. This is why we believe that

> "There is an urgent need to look at various teaching issues in evolutionary computation again, e.g., how we can help students to best learn the subject effectively, what kind of topics we should cover in a one-semester course, how much practical lab work should be involved, what a suitable textbook should be, etc."[5]

3 Evolutionary Computation Teaching at Birmingham

At present only one course in EC is offered at Birmingham, which is described in the following sections. However, the Evolutionary and Emergent Behaviour Intelligence and Computation (EEBIC) group at Birmingham [7] is expanding rapidly at present. One new professor of EC (Dr. Xin Yao) will join the School in April 1999 and a new lecturer in EC will be recruited during 1999. So, the number and the depth of EC courses at Birmingham are very likely to grow significantly in the near future.

3.1 Background

The School of Computer Science of The University of Birmingham has offered for many years various computer science and software engineering degrees. Since 1993 a half BSc degree in artificial intelligence was introduced, which can be combined with computer science, math, psychology or arts [8]. The AI degree, like the BSc degrees in most British universities, is achieved at the end of a three year course.

I joined the School in September 1994, and in the academic year 1995/96 I started teaching (among other things) a course, "SEM3A4 Evolutionary Computation", on evolutionary algorithms. I have taught SEM3A4 every year since then.

SEM3A4 is an optional course. In 1995 the course was open only to third year AI students. These were the first final-year students we had in AI. Nearly all of them (16 students) registered for it. However, since many non-AI students were interested in the course from 1996, SEM3A4 was offered, as an option, to every final year undergraduate and to advanced-CS MSc students.[1] Since the beginning the course was hugely popular among the students. In 1996 about 85 students registered, while in 1997 and 1998 65 students took the course,[2] which makes of SEM3A4 the most popular AI course in the School.

The course is a 10 credit course, i.e. a course lasting for one semester and taking one sixth of the time available to the students. One semester in British universities is considerably shorter than in many other countries. It only includes 12 weeks of teaching. One of these (week 7) is a reading week where the students deepen their understanding of the material presented so far, and where no new material is introduced. Another (week 12) is a revision week where again no new material is introduced, and the students revise the material covered in the course in preparation for their exams. So, in total only 10 weeks are really available for new material.

[1]The School offers an advanced MSc in Computer Science for people with a good degree or equivalent in Computer Science or a closely related topic, which is partly taught and partly research-based [8]. We also offer an MSc in Computer Science which is a postgraduate conversion course intended for graduates in disciplines other than Computer Science and assumes no formal knowledge of Computer Science.

[2]The drop since 1995 seems mainly due to the School expanding significantly the number of options available.

3.2 Aims and Objectives of SEM3A4

The aims and objective of my course were drawn and refined over time considering the ingredients necessary to prepare good future EC researchers and engineers, as discussed in Section 2. The main aims of the course are:

> "To give an introduction to the main techniques and applications of evolutionary computation.
>
> To give students some practical experience on when evolutionary techniques are useful, how to use them in practice and how to implement them with different programming languages." [9]

The objectives of the course are:

> "On completion of this course, the student should be able to:
>
> > Understand the relations between the most important evolutionary algorithms presented in the course, new algorithms they will find in the literature now or in the future, and other search and optimisation techniques.
> >
> > Discover where to look for to find out more details on specific evolutionary algorithms.
> >
> > Implement evolutionary algorithms.
> >
> > Determine the right parameter settings to make different evolutionary algorithms work well.
> >
> > Design new evolutionary operators, representations and fitness functions for specific practical and scientific applications."[9]

Obviously achieving all this to a great depth is a relatively short course is very difficult. So, some of the objectives listed above are achieved to a greater extent than others.

3.3 Delivery and Assessment

The course material is covered in lectures and exercise classes (tutorials). I normally do around 13 lectures and 7 tutorials: 20 hours in total. So, the material in the course can only be introduced at a relatively basic level. Particular care has to be taken in presenting notions requiring a background in math, since the AI degree does not require having specialised in math before coming to university.

No programming exercises are explicitly included in the course. This was originally requested by the School's Lab Director when the course started, since it was feared that having several tens of students running evolutionary algorithms concurrently would have put too much strain on our computing equipment. At present our labs could easily cope with this, so it is possible that programming exercises will be introduced in future editions of the course.

During the course, I mainly use OHP transparencies, but I also occasionally show runs on a X terminal connected to a video projector. Before each lecture and tutorial a handout containing (reduced) copies of the slides I use during the lecture is distributed to the students (free of charge). An old version of the slides is also available on-line [10]. In total around 200 transparencies are distributed. Also, exercises, code listings and old exam papers are distributed throughout the course.

The course is currently assessed by a two hour written exam including around 10 questions requiring essay like answers.

3.4 Syllabus

The syllabus of the course (available online from [9]) is reported in Appendix A. The main emphasis of the course is on Genetic Algorithms (GAs) and Genetic Programming (GP), with a couple of lectures on classifier systems (CFSs) and one lecture on other EC techniques (when time permits). Hard choices have to be made in short courses. One has to choose between covering all EC but in a shallow way and covering only a subset of it but in some depth. Given the motivations discussed in Section 2 and the research interests of the author, the latter choice seemed preferable.

In the first lecture the course is presented and an annotated reading list is provided to the students (see Appendix B). The course starts with the basic principles of natural evolution.

Then the course introduces the most basic GA one could imagine and the students are immediately shown an example with a tiny population. The objective of the example is to illustrate how the GA works, how the individuals in the population change under crossover and selection and how the average fitness of the population is affected by selection and crossover.

The course then delves into the details of GAs: encoding schemes, different selection procedures, fitness functions and their transformations, crossover and mutation operators. These topics are followed by some example applications on unconstrained optimisation problems in different domains. Constrained optimisation using GAs with the penalty method is also briefly introduced.

Nearly two lectures are spent on combinatorial optimisation with GAs. Given the limited time available only material related to the travelling salesperson problem is covered (partially matched crossover, order crossover, cycle crossover, edge recombination crossover, mutation operators).

The schema theory for GAs comes next. Schemata are introduced in terms of sets, similarity templates and hyperplanes. Then the schema theorem for GAs is derived. Since I cannot assume more than a basic knowledge of math, I am very careful in introducing the theorem. I try to give the students an intuitive idea of why the schema theorem may be important using slides like the one shown in Figure 1.

After the schema theory, about three lectures are devoted to genetic programming. In addition to describing the way GP works, details on how to implement the crossover operator, the interpreter, macros, etc. are given using Pop-11 and C code fragments.

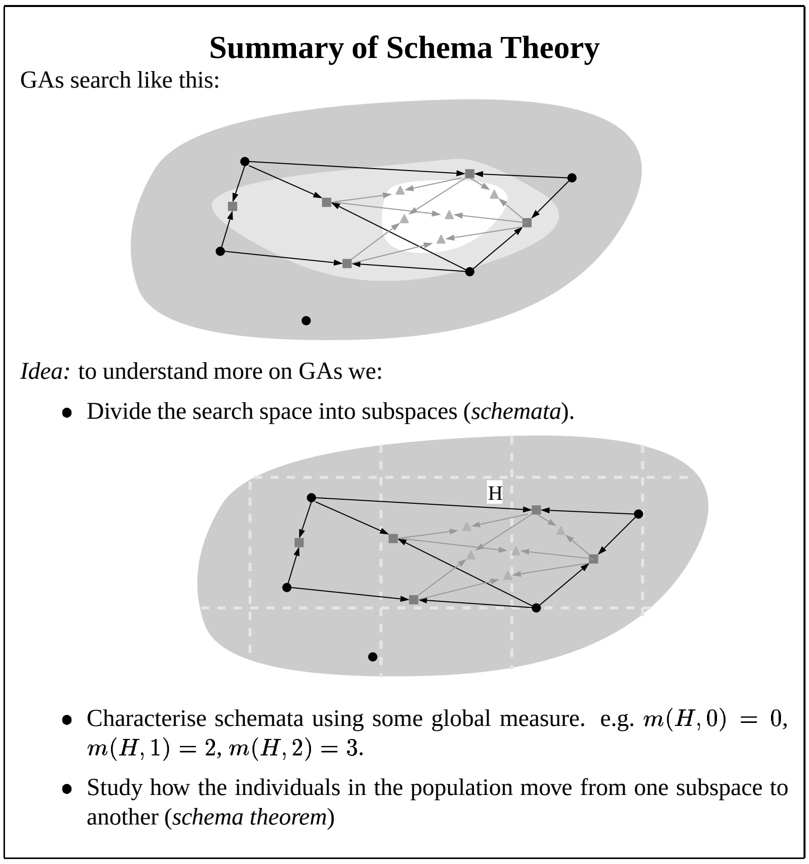
Finally the course presents classifier systems (CFSs). The non-learning version of CFSs is introduced as a simplified kind of rule-based system with input/output interfaces, so as to relate CFSs to other AI concepts as much as possible. Then the learning version of CFSs in introduced with examples to clarify how the bucket brigade algorithm works.

During the course lectures are interleaved with tutorials where the main topics in the lectures are first revised and then applied in simple exercises that the students have to complete (alone or in small groups) in the class. The tutorials cover the following topics: simple generational GA, representation of real numbers, Gray code in GAs, fitness proportionate selection and stochastic universal sampling, rank selection, mapping objective functions into fitness functions, PMX crossover, ERX crossover, schemata, effect of fitness proportionate selection on schemata, GP program representations, program space, crossover and mutation in GP, program evaluation in GP, past exams.

4 Lessons Learnt

During the last four years the students seem to have really enjoyed the course. The questionnaire distributed this year towards the end of the course indicates that the students have found the course: very interesting and very relevant but slightly difficult, with the right amount of handouts which are very good, with slightly too few references to the literature, with about right speed of presentation, with slightly too little stress on key points, self-contained but without too much repetition of previous material, with slightly not enough time devoted to questions, and with an acceptable level of discipline. Tutorials were mentioned by nearly all students as a strong point in the course. Some students, with little background in math, found the more mathematical parts of the course difficult. On the contrary, the more mathematically minded students wanted more math. Some non-AI students were slightly unhappy with the first lecture on CFSs, since I introduced (very quickly) rule-based systems (which they did not know) and referred to them in several occasions. Instead the AI students found that approach quite nice. Also, every year there are one or two (masochist?) students who, having put a lot of effort in the course, are unhappy because the exam paper seems too easy (to them) and other, less diligent, students are able to get reasonable marks without so much effort.

Finding the right balance between references to AI and self-containedness, and between mathematics and intuition is the trickiest thing in this course, together with setting up the



Summary of Schema Theory

GAs search like this:

Idea: to understand more on GAs we:

- Divide the search space into subspaces (*schemata*).
- Characterise schemata using some global measure. e.g. $m(H,0) = 0$, $m(H,1) = 2$, $m(H,2) = 3$.
- Study how the individuals in the population move from one subspace to another (*schema theorem*)

Figure 1: Slide used to give students an intuitive idea of why schemata and the schema theorem may be important to understand how and why GAs work. Circles, squares and triangles represent individuals in different generations. Arrows represent crossover events.

right kind of exam paper. This is a no-win situation. Whatever decisions one makes, some students will be unhappy.

Every year I have revised the course and changed slightly my delivery to find better compromises between these irreconcilable requirements. I think after four iterations I am getting closer to a (local?) optimum. My next iterations will be to introduce some practical programming exercises, to increase the number of tutorials and to cover some GP theory (which was not possible this year).[3]

5 Conclusions

In this paper I have described my motivations for teaching evolutionary computation and then presented the course on evolutionary computation course I teach in the School of Computer Science of The University of Birmingham. The course has been very successful so far, but I have kept improving it on the basis of the feedback the students gave me.

Writing this paper has been difficult, because out of the about 80 papers I have co-authored this is the first time I had to write about something which is not really research. I don't think I am the only one in this position. EC teaching is something we are not used to write about, to talk about, and I fear, to think about too much. However, as I discussed in Section 2 not teaching EC well and widely might have a very significant negative impact on our field. So, let us start and keep talking about it!

Acknowledgements

The author wishes to thank the members of the Evolutionary and Emergent Behaviour Intelligence and Computation (EEBIC) group at Birmingham for useful comments and discussion.

A SEM3A4 Syllabus

- Natural and Artificial Evolution
  - How nature does it:
    Theories of evolution, how natural selection works, cell, DNA, reproduction, growth
  - Evolutionary Computation:
    Motivations for simulating the process of evolu-

[3]Some other changes will be necessary, if, as I hope, we will introduce new EC courses in the AI degree next year.

tion, nature-to-computer mapping, generic evolutionary algorithm

- Genetic Algorithms
  - Introduction:
    Simple generational GA, an example
  - Solution Encoding in GAs:
    Gray code in GAs, dynamic parameter encoding
  - Selection in GAs:
    Fitness proportionate selection, rank selection, tournament selection, elitist selection
  - Crossover in GAs
  - Mutation in GAs
- Optimisation with Genetic Algorithms
  - Mapping Objective Functions into Fitness Functions
  - GA for Unconstrained Optimisation:
    Examples
  - GAs for Constrained Optimisation
  - GAs for Combinatorial Optimisation:
    Solving TSP with standard GAs, position dependent representations, design of genetic operators, permutation-respecting crossover
- Schemata in Genetic Algorithms
  - Introduction
  - Geometric Interpretation of Schemata:
    Fitness of schemata/hyperplanes, some observations on schemata
  - Schema Theorem:
    Effect of fitness proportionate selection on schemata, effect of one-point crossover on schemata, effect of mutation on schemata, schema theorem
- Genetic Programming
  - Introduction
  - Function Set and Terminal Set in GP
  - Initial Population in GP
  - Crossover in GP
  - Mutation in GP
  - Running Programs in GP:
    Interpreting programs, problems with "non-functional" instructions, macros in GP
  - Fitness functions in GP
  - Symbolic Regression
- Automatically Defined Functions
  - Divide-and-conquer Strategy
  - Automatic Function Definition
  - Changes required by ADFs
  - Example: implementing the even-4 parity function:
    Solution without automatically defined functions, solution with automatically defined functions
- Schemata in Genetic Programming
  - Schema definitions
  - One-point crossover and point-mutation
  - Schema theorem for GP
- Classifier Systems
  - Introduction
  - Components of a CFS:
    Message list, classifier list, input and output interfaces
  - The CFS Main Cycle
  - What CFSs can do:
    Default hierarchies
  - Learning Classifier Systems:
    Good and bad classifiers, the need for competition, quality of classifiers, adaption by credit assignment, adaption by rule discovery
  - Main Cycle of Learning Classifier Systems
- Other evolutionary algorithms
  - Evolutionary Strategies
  - Evolutionary Programming
- Exercises

B Reading List

1. David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, Massachusetts, 1989. Still probably the best introduction to genetic algorithms and classifier systems.

2. Melanie Mitchell, *An introduction to genetic algorithms*, A Bradford Book, MIT Press, Cambridge, MA, 1996. A very good introduction to GAs, with GA theory and a bit on genetic programming. Cheap.

3. John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992. The "bible" of genetic programming.

4. Darrell Whitley. A genetic algorithm tutorial. Technical Report CS-93-103, Department of Computer Science, Colorado State University, August 1993. A very good introduction to GAs and GA theory.

5. John H. Holland, *Adaptation in Natural and Artificial Systems*, second edition, A Bradford Book, MIT Press, Cambridge, MA, 1992. The most influential book on GAs, but requires some background in math.

6. John H. Holland, *Hidden order: How adaptation builds complexity*, Addison-Wesley, Reading, MA, 1995. An inspiring introduction to complex adaptive systems, including GAs, GA theory, classifier systems, artificial life, etc.

7. Thomas Bäck and Hans-Paul Schwefel. An overview of evolutionary algorithms for parameter optimization. *Evolutionary Computation*, 1(1):1–23, 1993. A good review article on various evolutionary algorithms.

8. John R. Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press, Cambridge, Massachusetts, 1994. Mainly devoted to GP with automatically defined functions and architecture altering operations. Advanced. Contains many applications.

9. Kenneth E. Kinnear, Jr., editor. *Advances in Genetic Programming*. MIT Press, 1994. A collection of articles by prominent researchers in GP. Several applications and advanced techniques are described.

10. Peter J. Angeline and Kenneth E. Kinnear, Jr., editors. *Advances in Genetic Programming 2*. MIT Press, 1996. A collection of articles by prominent researchers in GP. Several applications and advanced techniques are described. A recent picture of the state of the art in GP.

11. Lawrence Davis, editor. *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, New York, 1991. An introduction to GAs with a strong emphasis on hybrid techniques. Several case studies.

12. Zbigniew Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, Berlin, second edition, 1994.

13. David B. Fogel, *Evolutionary Computation: Towards a New Philosophy of Machine Intelligence*, IEEE Press, New York, 1995.

14. Wolfgang Banzhaf, Peter Nordin, Robert E. Keller and Frank D. Francone, *Genetic Programming – An Introduction; On the Automatic Evolution of Computer Programs and its Applications*, Morgan Kaufmann, 1998. The first university textbook on GP. Very good.

15. School of Computer Science Technical Reports on Evolutionary Computation. These are available in the School's library and also at `http://www.cs.bham.ac.uk/system/tech-reports/tr.html` (run a search with keywork "genetic").

Bibliography

[1] EvoWeb, "WWW home page of The European Network of Excellence in Evolutionary Computing", `http://www.dcs.napier.ac.uk/evonet/Coordinator/evonet.html`, Last updated 1 February 1999.

[2] David E. Goldberg, Kerry Zakrzewski, Brad Sutton, Ross Gadient, Cecilia Chang, Pilar Gallego, Brad Miller, and Erick Cantu-Paz, "Genetic algorithms: A bibliography", Tech. Rep. 97011, Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign, December 1997.

[3] W. B. Langdon and J. R. Koza (Maintainers), "Genetic Programming Bibliography", Available on line at `http://liinwww.ira.uka.de/bibliography/Ai/genetic.programming.html`, 1999.

[4] J. R. Koza, Ed., *University Courses on Genetic Algorithms 1995*. Stanford Bookstore, 1995.

[5] R. Poli and Xin Yao, "Survey on evolutionary computation courses", `http://www.cs.bham.ac.uk/~rmp/eebic/ec_teaching/`, February 1999.

[6] J. R. Koza, "WWW home page of Genetic Programming Conferences Inc.", `http://www.genetic-programming.org/`, Last updated 13 December 1998.

[7] R. Poli (Maintainer), "Evolutionary and Emergent Behaviour Intelligence and Computation (EEBIC) group, School of Computer Science, The University of Birmingham", `http://www.cs.bham.ac.uk/~rmp/eebic/`, Last modified February 1999.

[8] T. H. Axford (Maintainer), "Degree Programmes for 1998/99, School of Computer Science, The University of Birmingham", `http://www.cs.bham.ac.uk/degreeregs/`, Last updated 26th May 1998.

[9] R. Poli, "SEM3A4 Evolutionary Computation, School of Computer Science, The University of Birmingham", `http://www.cs.bham.ac.uk/~rmp/courses/sem3a4.html`, Last updated 28 July 1998.

[10] R. Poli, "Introduction to Evolutionary Computation, School of Computer Science, The University of Birmingham", `http://www.cs.bham.ac.uk/~rmp/slide_book/`, 11 October 1996.