

---

## Preface

This book is an adaptation of notes that have been used to teach a class in evolutionary computation at Iowa State University for eight years. A number of people have used the notes over the years, and by publishing them in book form I hope to make the material available to a wider audience.

It is important to state clearly what this book is and what it is not. It is a text for an undergraduate or first-year graduate course in evolutionary computation for computer science, engineering, or other computational science students. The large number of homework problems, projects, and experiments stem from an effort to make the text accessible to undergraduates with some programming skill. This book is directed mainly toward application of evolutionary algorithms. This book is *not* a complete introduction to evolutionary computation, nor does it contain a history of the discipline. It is not a theoretical treatment of evolutionary computation, lacking chapters on the schema theorem and the no free lunch theorem.

The key to this text are the experiments. The experiments are small computational projects intended to illustrate single aspects of evolutionary computation or to compare different methods. Small changes in implementation create substantial changes in the behavior of an evolutionary algorithm. Because of this, the text does not tell students what will happen if a given method is used. Rather, it encourages them to experiment with the method. The experiments are intended to be used to drive student learning. The instructor should encourage students to experiment beyond the stated boundaries of the experiments. I have had excellent luck with students finding publishable new ideas by exceeding the bounds of the experiments suggested in the book.

Source code for experiments, errata for the book, and bonus chapters and sections extending material in the book are available via the Springer website [www.springeronline.com](http://www.springeronline.com) or at

**[www.eldar.http://eldar.mathstat.uoguelph.ca/dashlock/OMEC/](http://www.eldar.mathstat.uoguelph.ca/dashlock/OMEC/)**

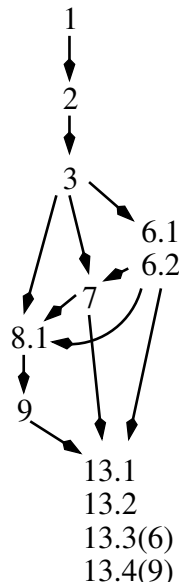
The book is too long for a one-semester course, and I have never managed to teach more than eight chapters in any one-semester offering of the course. The diagrams at the end of this preface give some possible paths through the text with different emphases. The chapter summaries following the diagrams may also be of some help in planning a course that uses this text.

### Some Suggestions for Instructors Using This Text

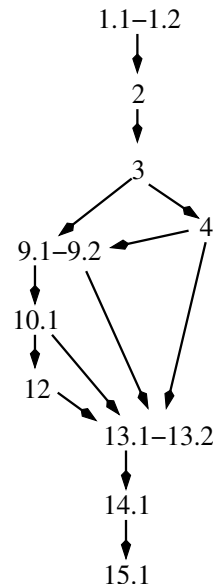
- Make sure you run the code for an experiment before you hand it out to the class. Idiosyncratic details of your local system can cause serious problems. Lean on your most computationally competent students; they can be a treasure.
- Be very clear from the beginning about how you want your students to write up an experiment. Appendix A shows the way I ask students to write up labs for my version of the course.
- I sometimes run contests for Prisoner's Dilemma, Sunburn, the virtual politicians, or other competitive evolutionary tasks. Students evolve competitors and turn them in to compete with each other. Such competitions can be very motivational.
- Assign and grade lots of homework, including the essay questions. These questions are difficult to grade, but they give you, the instructor, excellent feedback about what your students have and have not absorbed. They also force the students that make an honest try to confront their own ignorance.

### Possible Paths Through the Text

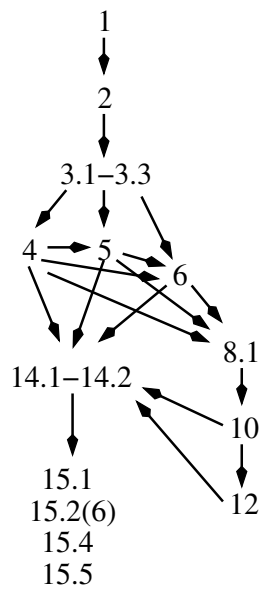
The following diagrams give six possible collections of paths through the text. Chapters listed in parentheses are prerequisite. Thus 13.3(6) means that Section 13.3 uses material from Chapter 6.



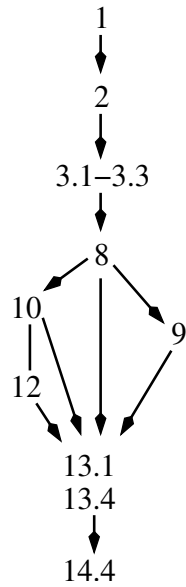
A course on using evolutionary algorithms for optimization.



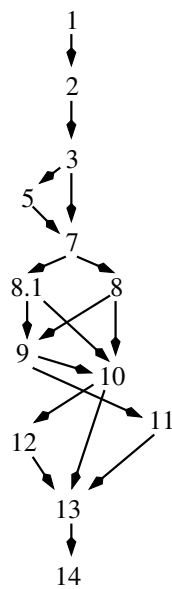
A course on evolutionary algorithms using only simple string data structures.



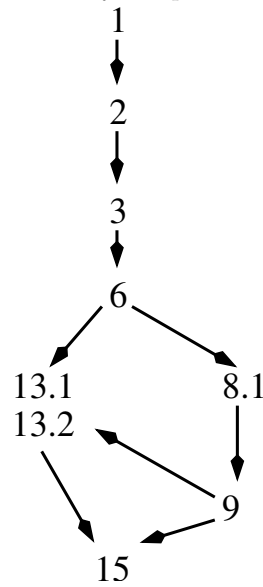
A course on using evolutionary algorithms for modeling.



A course focused on genetic programming.



A broad survey of techniques in evolutionary computation.



A course on evolutionary algorithms potentially useful in bioinformatics.

## A Brief Summary of Chapters

Chapter 1 gives examples of evolutionary algorithms and a brief introduction to simple evolutionary algorithms and simple genetic programming. There is some background in biology in the chapter that may help a computational specialist understand the biological inspiration for evolutionary computation. There is also material included to help the instructor with students deeply skeptical of the scientific foundations of evolution. Chapter 1 can typically be skipped if there is a shortage of time. Most of the technically required background is repeated in greater detail in later chapters.

Chapter 2 introduces simple string data structure evolutionary algorithms. In this context, it surveys the “parts list” of most evolutionary algorithms including mutation and crossover operators, and selection and replacement mechanisms. The chapter also introduces two complex string problems: the Royal Road problem and the self-avoiding walk problem. These set the stage for the discussion of fitness landscapes in Chapter 3. The final section introduces a technical flourish and can easily be skipped.

Chapter 3 introduces real function optimization using a string of real numbers (array) as its representation. The notion of fitness landscape is introduced. The idea of niche specialization is introduced in Section 3.3 and may be included or skipped. Section 4 closely compares two fitness functions for the same problem. Section 5 introduces a simplified version of the circuit board layout problem. The fitness function is constant except where it is discontinuous and so makes a natural target for an evolutionary computation.

Chapter 4 introduces the idea of a model-based fitness function. Both the star fighter design problem (Sunburn) and the virtual politicians use a model of a situation to evaluate fitness. The model of selection and replacement is a novel one: gladiatorial tournament selection. This chapter is not deep, is intended to be fun, and is quite popular with students.

Chapter 5 introduces the programming of very simple artificial neural nets with an evolutionary algorithm in the context of virtual robots. These virtual robots, the symbots, are fairly good models of trophic life forms. This chapter introduces the problem of stochastic fitness evaluation in which there are a large number of fitness cases that need to be sampled. The true “fitness” of a given symbot is elusive and must be approximated. This chapter is a natural for visualization. If you have graphics-competent students, have them build a visualization tool for the symbots.

Chapter 6 introduces the finite state machine representation. The first section is a little dry, but contains important technical background. The second sec-

tion uses finite state machines as game-playing agents for Iterated Prisoner's Dilemma. This section lays out the foundations used in a good deal of published research in a broad variety of fields. The third section continues on to other games. The first section of this chapter is needed for both GP automata in Chapter 10 and chaos automata in Chapter 15.

Chapter 7 introduces the permutation or ordered list representation. The first section introduces a pair of essentially trivial fitness functions for permutation genes. Section 2 covers the famous Traveling Salesman problem. Section 3 covers a bin-packing problem and also uses a hybrid evolutionary/greedy algorithm. The permutations being evolved control a greedy algorithm. Such hybrid representations enhance the power of evolutionary computation and are coming into broad use. Section 4 introduces an applied problem with some unsolved cases, the Costas array problem. Costas arrays are used as sonar masks, and there are some sizes for which no Costas array is known. This last section can easily be skipped.

Chapter 8 introduces genetic programming in its most minimal form. The variable (rather than fixed) sized data structure is the hallmark of genetic programming. The plus-one-recall-store problem, the focus of the chapter, is a type of maximum problem. This chapter tends to be unpopular with students, and some of the problems require quite advanced mathematics to solve. Mathematicians may find the chapter among the most interesting in the book. Only the first section is really needed to go on to the other chapters with genetic programming in them. The chapter introduces the practice of seeding populations.

Chapter 9 introduces regression in two distinct forms. The first section covers the classical notion of parameter-fitting regression, and the second uses evolutionary computation to perform such parameter fits. The third section introduces symbolic regression, i.e., the use of genetic programming both to find a model and fit its parameters. Section 4 introduces automatically defined functions, the "subroutine" of the genetic programming world. Section 5 looks at regression in more dimensions and can be included or not at the instructor's whim. Section 6 discusses a form of metaselection that occurs in genetic programming called bloat. Since the type of crossover used in genetic programming is very disruptive, the population evolves to resist this disruption by having individual population members get very large. This is an important topic. Controlling, preventing, and exploiting bloat are all current research topics.

Chapter 10 introduces a type of virtual robot, grid robots, with the Tartarus task. The robots are asked to rearrange boxes in a small room. This topic is also popular with students and has led to more student publications than

any other chapter in the text. The first section introduces the problem. The second shows how to perform baseline studies with string-based representations. The third section attacks the problem with genetic programming. The fourth introduces a novel representation called the GP automaton. This is the first hybrid representation in the text, fusing finite state machines and genetic programming.

Chapter 11 covers a traditional topic: programming neural nets to simulate digital logic functions. While there are many papers published on this topic it is a little dry. The chapter introduces neural nets in a more complex form than Chapter 5. The chapter looks at direct representation of neural weights and at a way of permitting both the net's connectivity and weights to evolve, and finally attacks the logic function induction problem with genetic programming in its last section.

Chapter 12 introduces a novel linear representation for genetic programming called an ISAc list. ISAc lists are short pieces of simplified machine code. They use a form of goto and so can be used to evolve fully functioning programs with nontrivial flow of control. The chapter introduces ISAc lists in Section 1 and then uses them on the Tartarus problem in Section 2. Section 3 introduces a large number of new grid robot tasks. Section 4 uses ISAc lists as an inspiration to create a more powerful type of string representation for grid robot tasks. This latter section can be skipped.

Chapter 13 introduces a generic improvement to a broad variety of evolutionary algorithms. This improvement consists in storing the evolving population in a geography, represented as a combinatorial graph, that limits selection and crossover. The effect is to slow convergence of the algorithm and enhance exploration of the search space. The first section introduces combinatorial graphs as population structures. The second section uses the techniques on string-based representations. The third uses the graph-based population structure on more complex representations, such as finite state machines and ordered genes. The last section explores genetic programming on graphs. Other than the first section, the sections of this chapter are substantially independent.

Chapter 14 contains four extended examples of a generic technique: storing directions for building a structure rather than the structure itself. This type of representation is called a "cellular" representation for historical reasons. The first section uses a cellular representation to evolve two-dimensional shapes. The second introduces a cellular representation for finite state automata. The third introduces a novel editing representation that permits the evolution of a class of combinatorial graphs. The fourth section uses context free grammars to create a cellular encoding for genetic programming. This technique is quite powerful, since it permits transparent typing of genetic programming systems

as well as the incorporation of domain-specific knowledge. The sections of this chapter may be used independently.

Chapter 15 gives examples of applications of evolutionary computation to bioinformatics. The first three sections are completely independent of one another. Section 1 gives an application of string-type genes to an applied (published) problem in bioinformatics. It both aligns and characterizes an insertion point of a type of genetic parasite in corn. Section 2 uses finite state machines to attempt to learn sets of PCR primers that work well and poorly. The finite state machines are intended as filters for subsequently designed primers. The third section introduces a hybrid evolutionary/greedy representation for a hard search problem, locating error-tolerant DNA tags used to mark genetic constructs. The last two sections give methods of visualizing DNA as a fractal.

### Acknowledgments

I would like to thank my wife, Wendy, who has been a key player in preparing the manuscript and helping me get things done, and who has acted as a sounding board for many of the novel ideas contained in this text. I also owe a great deal to the students who supplied ideas in the book, such as John Walker, who thought up Sunburn and helped develop the symbots; Mark Joenks, the creator of ISAc lists and virtual politicians; Mark Smucker, whose ideas led to graph-based evolutionary algorithms; Warren Kurt vonRoeschlaub, who started the symbots and other projects; and Mike McRoberts, who coded up the first implementation of GP automata. I thank Jim Golden, who was a key participant in the research underlying the fractal visualization of DNA. I am also grateful to the numerous students who turned in edits to the manuscript over the years, including Pete Johnson, Steve Corns, Elizabeth Blankenship, and Jonathan Gandrud. The Bryden, Schnable, and Sheble labs at Iowa State have supplied me with many valuable students over the years who have asked many questions answered in this book. Mark Bryden, Pat Schnable, and Gerald Sheble all provided a valuable driving force toward the completion of this book.

<http://www.springer.com/978-0-387-22196-0>

Evolutionary Computation for Modeling and Optimization

Ashlock, D.

2006, XX, 572 p., Hardcover

ISBN: 978-0-387-22196-0