Daniel Ashlock

Evolutionary Computation for Modeling and Optimization

Springer

Evolutionary Computation for Modeling and Optimization

Daniel Ashlock

Evolutionary Computation for Modeling and Optimization

With 163 Figures

Springer

Daniel Ashlock
Department of Mathematics and Statistics
University of Guelph
Guelph, Ontario NIG 2W1
CANADA
dashlock@uguelph.ca

Mathematics Subject Classification (2000): 6801, 68T20, 68T40

Library of Congress Control Number: 2005923845

ISBN-10: 0-387-22196-4
ISBN-13: 978-0387-22196-0

Printed on acid-free paper.

Printed in the United States of America.

9 8 7 6 5 4 3 2 1

springeronline.com

Daniel Ashlock

Evolutionary Computation for Modeling and Optimization

October 21, 2005

Springer
Berlin Heidelberg New York
Hong Kong London
Milan Paris Tokyo

To my wife, Wendy

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 or at

www.eldar.http://eldar.mathstat.uoguelph.ca/dashlock/OMEC/

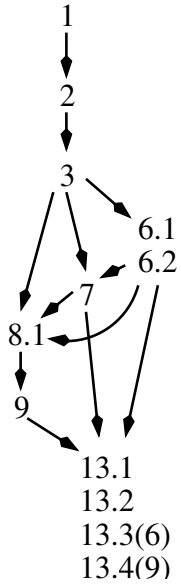
The book is too long for a one-semester course, and I have never managed to teach more than eight chapters in any one-semester offering of the course. The diagrams at the end of this preface give some possible paths through the text with different emphases. The chapter summaries following the diagrams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

Some Suggestions for Instructors Using This Text

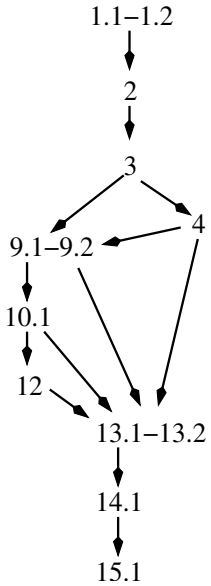
- Make sure you run the code for an experiment before you hand it out to the class. Idiosyncratic details of your local system can cause serious problems. Lean on your most computationally competent students; they can be a treasure.
- Be very clear from the beginning about how you want your students 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

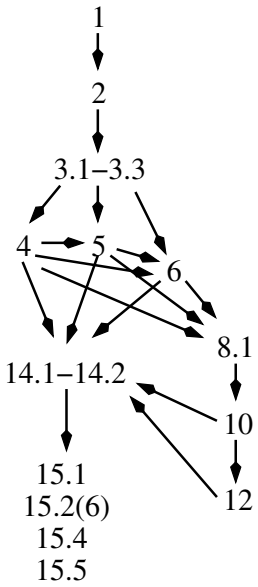
The following diagrams give six possible collections of paths through the text. Chapters listed in parentheses are prerequisite. Thus 13.3(6) means that Section 13.3 uses material from Chapter 6.



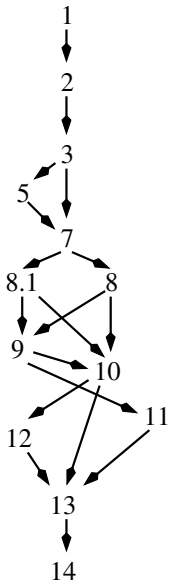
A course on using evolutionary algorithms for optimization.



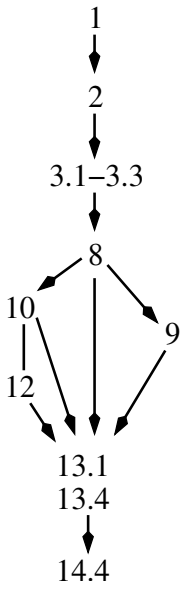
A course on evolutionary algorithms using only simple string data structures.



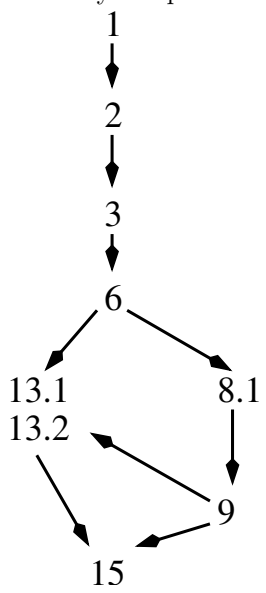
A course on using evolutionary algorithms for modeling.



A broad survey of techniques in evolutionary computation.



A course focused on genetic programming.



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.

Contents

1

An Overview of Evolutionary Computation

Evolutionary computation is an ambitious name for a simple idea: use the theory of evolution as an algorithm. Any program that uses the basic loop shown in Figure 1.1 could be termed evolutionary computation. In this text we will explore some of the many ways to fill in the details to the simple structure in Figure 1.1. Evolutionary algorithms operate on populations. We will choose data structures to represent the population, quality measures, and different ways to vary the data structures. We will need to decide how to tell when to stop. For any given problem there are many ways to implement an evolutionary computation system to attack the problem.

Generate a population of structures
Repeat
    Test the structures for quality
    Select structures to reproduce
    Produce new variations of selected structures
    Replace old structures with new ones
Until Satisfied

Fig. 1.1. The fundamental structure of an evolutionary computation.

The field of evolutionary computation has many founders and many names. A concise summary of the origins of evolutionary computation can be found in [8]. You may wonder how the notion of evolutionary computation could be discovered a large number of times without later discoverers noticing those

before them. The reasons for this are complex and serve as a good starting point.

The simplest reason evolutionary computation was discovered multiple times is that techniques that cannot be applied yet are not remembered. During the Italian Renaissance, Leonardo da Vinci produced drawings for machines, such as the helicopter, that did not exist as working models for centuries. If he were not a genius and well remembered for other works, his version of the helicopter might well have faded into history. The idea of taking the techniques used by nature to produce diverse complex systems and use them as algorithms is a natural one. Fields like neural computation (computation with artificial neural nets) and fuzzy logic also draw inspiration from biology. The problem is that before the routine availability of large powerful computers these biologically derived ideas could not be implemented. Without big iron, even extremely simplified simulated biology is too slow for most applications.

Limited work with various levels of application and interest began in the 1950s. Sustained and widespread research in evolutionary computation began in the 1970s. By the late 1980s, computer power and human ingenuity combined to create an explosion of research. Vast numbers of articles can be found by searching the World Wide Web with any of the keys "Artificial Life," "Evolutionary Algorithms," "Genetic Algorithms"[29], "Evolutionary Programming"[23, 24], "Evolution Strategies"[13], or "Genetic Programming"[38, 39, 9]. To get a manageable-sized stack, you must limit these search keys to specific application or problem domains.

The second reason that evolutionary computation was discovered a large number of times is its interdisciplinary character. The field is an application of biological theory to computer science used to solve problems in dozens of fields. This means that different groups of people who *never* read one another's publications had the idea independently of using evolution as an algorithm. Early articles appear in journals as diverse as the *IBM Journal of Research and Development*, *the Journal of Theoretical Biology*, and *Physica D.* It is a very broad-minded scholar who reads journals that many floors apart in the typical university library. The advent of the World Wide Web has lowered, but not erased, the barriers that enabled the original multiple discoveries of evolutionary computation. Even now, the same problem is often attacked by different schools of evolutionary computation with years passing before the different groups notice one another.

The third source of the confused origins of evolutionary computation is the problem of naming. Most of the terminology used in evolutionary computation is borrowed from biology by computational scientists with essentially no formal training in biology. As a result, the names are pretty arbitrary and also annoying to biologists. People who understand one meaning of a term are resistant to alternative meanings. This leads to a situation in which a single word, e.g., "crossover," describes a biological process and a handful of different computational operations. These operations are quite different from

one another and linked to the biology only by a thin thread of analogy: a perfect situation for confusion over who discovered what and when they did so. If you are interested in the history of evolutionary computation, you should read *Evolutionary Computation, the Fossil Record* [22]. In this book, David Fogel has compiled early papers in the area together with an introduction to evolutionary computation. The book supplies a good deal of historically useful context in addition to collecting the early papers.

As you work through this text, you will have ideas of your own about how to modify experiments, new directions to take, etc. Beware of being overenthusiastic: someone may have already had your clever idea; check around before trying to publish, patent, or market it. However, evolutionary computation is far from being a mature field, and relative newcomers can still make substantial contributions. Don't assume that your idea is obvious and must have already been tried. Being there first can be a pleasant experience.

1.1 Examples of Evolutionary Computation

Having warned you about the extended and chaotic beginnings of evolutionary computation, we will now look at some examples of applications of the discipline. These examples require no specific technical knowledge and are intended only to give the flavor of evolutionary computation to the novice.

1.1.1 Predators Running Backward

In Chapter 5, we will be looking at a variation of an experiment [12] reported in the first edition of the journal *Adaptive Behavior*. In this experiment, the authors use evolutionary computation to evolve a simple virtual robot to find a signal source. The robot's brain is an artificial neural net. The robot's sensors are fed into two of the neurons, and the robot's wheels are driven by two of the neurons. Evolution is used to select the exact weight values that specify the behavior of the neural net. Imagine the robot as a wheeled cart seeking a light source. Robots that find (stay near) the light source are granted reproductive rights. Reproduction is imperfect, enabling a search for robot brains that work better than the current best ones.

A student of mine, Warren Kurt vonRoeschlaub, attempted to replicate this experiment as a final project for an artificial intelligence class. Where the published experiment used a robot with six nonlinear artificial neurons, vonRoeschlaub created a robot with eight linear neurons. Without going into the technicalities, there is good reason to think that linear neurons are a good deal less powerful as a computational tool. In spite of this, the robots were able to find the simulated light source.

VonRoeschlaub, who had a fairly bad case of "hacker," modified the light source so that it could drift instead of holding still. The robots evolved to stay close to the light source. Not satisfied with this generalization, he then

went on to give the light source its own neural net and the ability to move. At this point, the light source became "prey," and the simulation became a simulation of predator and prey. In order to generalize the simulation this way, he had to make a number of decisions.

He gave the predator and prey robots the same types of sensors and wheels. The sensors of prey robots could sense predators only in a 60-degree cone ahead of the robot; likewise, predator robots could sense prey only in a 60-degree cone ahead of them. Both species had two sensors pointing in parallel directions on one side of their body. Neither could sense its own type (and robots of the same type could pass through one another). If a predator and prey intersected, then the prey was presumed to have been eaten by the predator. At this point, several things happened simultaneously. The prey robot was deleted. The oldest surviving prey animal fissioned into two copies, one of which had its neural net modified. The predator also fissioned, and one of the two copies was also modified, and the oldest predator was deleted. This system exhibits many of the features of natural evolution. Reproduction requires that a creature "do a good job," either by eating or avoiding being eaten for the longest possible time. Children are similar but not always identical to their parents.

VonRoeschlaub ran this system a number of times, starting over each time with new randomly initialized neural nets for his predator and prey robots. In all the populations, the creatures wandered about almost at random initially. In most of the populations, a behavior arose in which a predator would leap forward when the signal strengths of its sensors became equal. Unless multiple prey are both in front of and near the predator, this amounts to the sensible strategy of leaping at prey in front of you.

Once this predator behavior had arisen, the prey in many cases evolved an interesting response. When their predator sensors returned similar strengths, they too leaped forward. This had the effect of generating a near miss in many cases. Given that the predators can see only in front of them and have effectively no memory, a near miss is the cleanest possible form of getaway. The response of some of the predators to this strategy was a little startling. The average velocity of predators in three of the populations became negative. At first, this was assumed to be a bug in the code. Subsequent examination of the movement tracks of the predator and prey robots showed that what was happening was in fact a clever adaptation. The predators, initially of the "leap at the prey" variety, evolved to first lower their leaping velocity and later to actually run backward away from prey.

Since the prey were leaping forward to generate a near miss, leaping more slowly or even running backward actually meant getting *more* prey. We named the backward predators the "reverse Ferrari lions," and their appearance illustrates the point that evolutionary computation can have surprising results. There are a number of other points worth considering about this experiment. Of about 40 initial populations, only three gave rise, during the time the experiment was run, to backward-running predators. Almost all generated the

forward leaping predator, and many produced the near-miss-generating prey. It may be that all three of these behaviors would have been discovered in every simulation, if only it had been run long enough. It may also be that there were effective predator and prey behaviors that evolved that do not include these three detected behaviors. These alternative evolutionary paths for the robot ecology could easily have gone unnoticed by an experimenter with limited time and primitive analysis tools. It is important to remember that a run of simulated evolution is itself a sample from a space of possible evolutionary runs. Typically, random numbers are used to generate the starting population and also in the process of imperfect reproduction. Different outcomes of the process of evolution have different probabilities of appearing. The problem of how to tell that all possible outcomes have been generated is unsolved. This is a feature of the technique: sometimes a flaw (if all solutions must be listed) and sometimes a virtue (if alternative solutions have value).

The experiment that led to the backward-running predators is one offshoot of the original paper that evolved neural nets to control virtual robots. Chapter 5 of this text is another. The experiments that led to Chapter 5 were motivated by the fact that on the robotic light-seeking task, eight linear neurons outperformed six nonlinear neurons. The linear neurons were removed in pairs until minimum training time was found. Minimum training time occurred at *no* neurons. Readers interested in this subject will find a rich collection of possible experiments in [14].

This example, a predator–prey system, is absolutely classical biology. The advantage of adding evolutionary computation to the enterprise is twofold. First, it permits the researcher to sample the space of possible strategies for the predators and prey, rather than designing or enumerating them. Second, the simulation as structured incorporates the vagaries of individual predators and prey animals. This makes the simulation an *agent-based* one.

There are many different ways to derive or code biological models. Agent-based models follow individual animals (agents) through their interactions with the simulated environment. Another sort of model is a statistical model. These are usually descriptive models, allowing a researcher to understand what is typical or atypical behavior. Yet another sort of model is the equation-based model. Predator–prey models are usually of this type. They use differential equations to describe the impact of prey on the growth rate of the predator population as well as the impact of predators on the growth rate of the prey [44, 46]. Equation-based models permit the theoretical derivation of properties of the system modeled, e.g., that one should observe cyclic behavior of predator and prey population sizes. Each of these types of model is good for solving different types of problems.

1.1.2 Wood-Burning Stoves

Figure 1.2 shows the plan of a type of wood-burning stove designed in part using evolutionary computation. In parts of Nicaragua, stoves such as these or

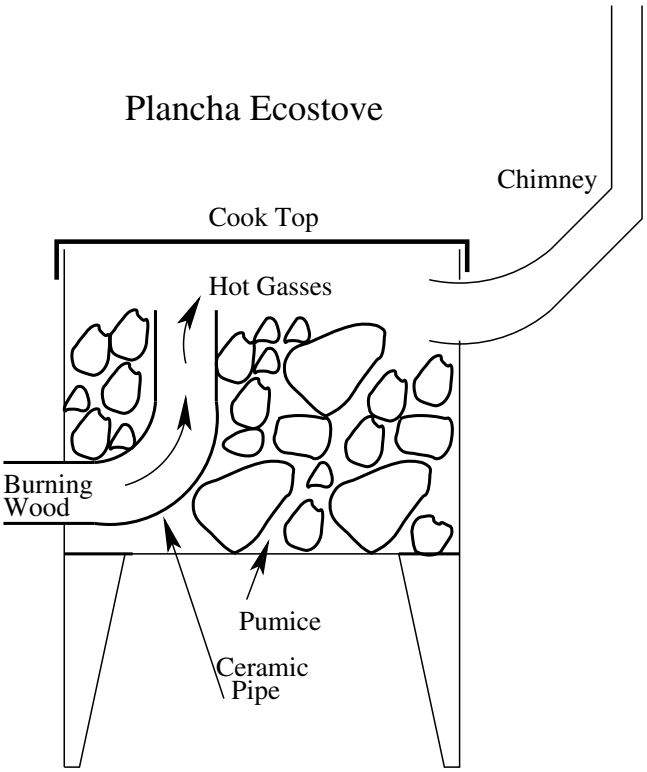


Fig. 1.2. Side cutaway view of a wood-burning Plancha EcoStove.

open hearths are found in most households. People spend as much as a quarter of their income on fuel wood for cooking. Indoor fires are often not vented, and thus they put smoke into the home causing respiratory problems and infant blindness. A not uncommon cause of injury is the "skirt fire." Improving the design of these stoves has the potential to dramatically improve the quality of people's lives. The stove design in Figure 1.2 has many advantages over an open hearth and other stove designs in current use. It uses a good deal less wood to cook the same amount of food, has a chimney to vent the fire, and can be made with inexpensive local materials by local craftsmen. The role of evolutionary computation in the project was to make a greater part of the cooktop useful for cooking by finding a way to spread the heat out more evenly.

The original design for the stove used a cooktop that was a square metal box, open on one side, that fit over the rest of the stove. Hot air would flow from the ceramic pipe where the wood was burned directly to the chimney. Over the ceramic pipe the cooktop was hot enough to boil water. Along the path from the hot spot to the chimney was a usefully hot cooking surface.