Extracted from:

Genetic Algorithms in Elixir

Solve Problems Using Evolution

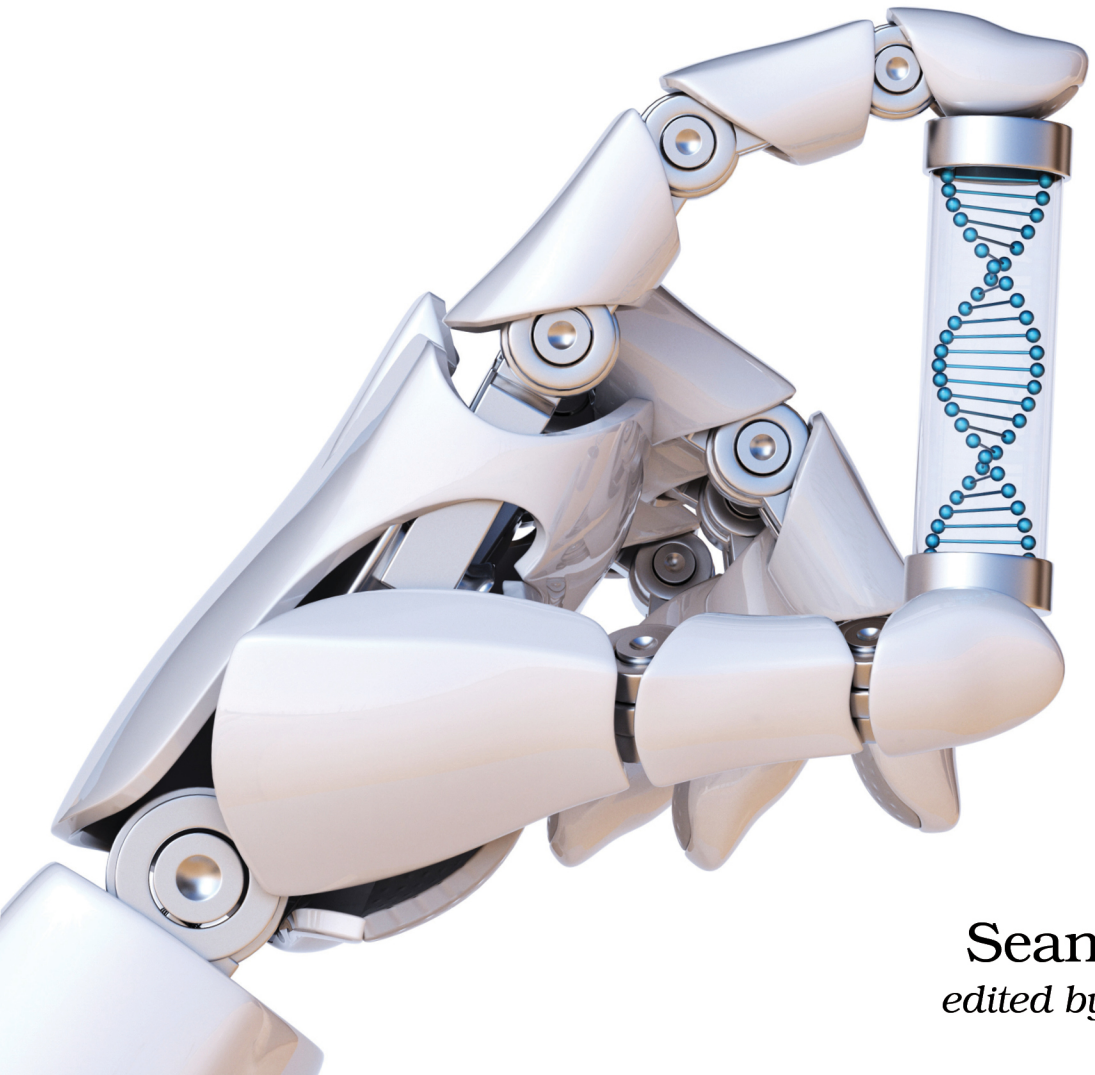
The Pragmatic Bookshelf

Raleigh, North Carolina

The Pragmatic Programmers

Genetic Algorithms in Elixir

Solve Problems Using Evolution



Sean Moriarity

edited by Tammy Coron

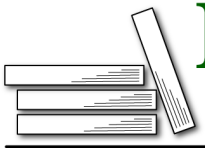
Genetic Algorithms in Elixir

Solve Problems Using Evolution

Sean Moriarity

The Pragmatic Bookshelf

Raleigh, North Carolina



Our Pragmatic books, screencasts, and audio books can help you and your team create better software and have more fun. Visit us at *https://pragprog.com*.

The team that produced this book includes:

Publisher: Andy Hunt
VP of Operations: Janet Furlow
Executive Editor: Dave Rankin
Development Editor: Tammy Coron
Copy Editor: L. Sakhi MacMillan
Indexing: Potomac Indexing, LLC
Layout: Gilson Graphics

For sales, volume licensing, and support, please contact *support@pragprog.com*.

For international rights, please contact *rights@pragprog.com*.

ISBN-13: 978-1-68050-794-2
Encoded using the finest acid-free high-entropy binary digits.
Book version: P1.0—January 2021

Preface

Genetic algorithms are a powerful and often overlooked tool for solving difficult problems. Some of the most beautiful solutions to practical problems are inspired by or modeled after solutions found in mother nature. Genetic algorithms are no exception. Inspired by the original optimization algorithm—evolution—genetic algorithms can be used to solve a variety of problems in a variety of fields. As you'll see in this book, genetic algorithms have applications in finance, logistics, artificial intelligence, and more.

Unfortunately, despite being one of the first "artificial intelligence" algorithms, there's a surprising lack of resources available for programmers to explore the ins and outs of using evolution to solve problems. Even still, there are no books designed specifically with Elixir programmers in mind.

My goal in writing this book is to introduce Elixir programmers to a field of programming they might have never been exposed to or were too intimidated to try. Technology evolves rapidly, and programmers need to constantly seek out and learn about new fields and new technologies. While Elixir may not be the ideal language for solving computationally expensive problems, a programmer shouldn't be forced to learn a new language just to learn about genetic algorithms.

In this book, you'll learn everything you need to know to start working with genetic algorithms. As you work through the book, you'll build a framework for problems using genetic algorithms. By the end, you'll have a full-featured, customizable framework complete with statistics, genealogy tracking, and more, and you'll have learned everything you need to solve practical problems with genetic algorithms. You'll do all of this using Elixir. Along the way, you'll learn some Elixir-specific tips and tricks to idiomatically encode problems and solutions, speed up your code, and verify the correctness of the algorithms you implement.

I hope this book forces you to think outside the box and inspires you to further explore the beauty of genetic algorithms.

Who This Book Is For

This book is for programmers with some experience or familiarity working with Elixir, who are looking to expand their knowledge into the field of genetic algorithms. While not traditionally thought of as a language suited for computationally expensive problems, Elixir's simple syntax and functional style make for the creation of idiomatic solutions to optimization problems with genetic algorithms. These solutions gently introduce the user to genetic algorithms and optimization problems without the overhead of learning a completely new programming language.

If you have no experience with Elixir, you might find this book difficult to follow at times. Before getting started, I recommend checking out Elixir School[1] or the Elixir Guides[2] to get some familiarity with the language.

What's in This Book

In Chapter 1, Writing Your First Genetic Algorithm, on page ?, you'll learn the basics of the genetic algorithm by solving an introductory optimization problem. You'll learn about the core concepts of a genetic algorithm by writing an Elixir script. By the end of the chapter, you'll get to see a genetic algorithm in action, and you'll begin to understand the kinds of problems best suited for using genetic algorithms.

In Chapter 2, Breaking Down Genetic Algorithms, on page ?, you'll dive deeper into the core concepts you learned about in the first chapter and you'll use some of Elixir's code constructs to turn the script you wrote in Chapter 1, Writing Your First Genetic Algorithm, on page ?, into a reusable framework for solving optimization problems. You'll learn more about each step in the evolutionary process of a genetic algorithm and, by the end of the chapter, have a barebones framework for using genetic algorithms.

In Chapter 3, Encoding Problems and Solutions, on page ?, you'll learn about how to use Elixir to represent optimization problems and solutions to optimization problems. You'll learn about how genetic algorithms represent solutions and how you can use a variety of strategies to represent real-world solutions using code. Finally, you'll create a program that learns how to spell in order to see how you can use Elixir behaviours to represent any optimization problem imaginable.

1. https://elixirschool.com/en/
2. https://elixir-lang.org/getting-started/introduction.html

In Chapter 4, Evaluating Solutions and Populations, on page ?, you'll explore how genetic algorithms learn to find better and better solutions by evaluating a set of solutions. You'll learn more about the concept of fitness. You'll also learn about how to write different fitness functions and termination criteria for different types of problems, including shipping optimization, portfolio optimization, and website optimization.

In Chapter 5, Selecting the Best, on page ?, you'll learn about the first operator in a genetic algorithm—selection. You'll learn about why selection is important, how selection rate affects your algorithms, and how to write different types of selection strategies. You'll learn about how different selection strategies apply best to different types of problems, and you'll learn how to customize them within your genetic algorithms.

In Chapter 6, Generating New Solutions, on page ?, you'll learn about how genetic algorithms create new solutions from existing ones using crossover. You will learn about different types of crossover strategies and how to implement them in Elixir. You'll learn how to solve the N-queens problem to see how crossover strategies can affect the solutions produced by your genetic algorithm.

In Chapter 7, Preventing Premature Convergence, on page ?, you'll learn about a common problem in genetic algorithms—premature convergence—and how to solve it using mutation. You'll create a basic password cracker to demonstrate premature convergence. You'll learn how to implement several different types of mutation strategies, and you'll learn which ones apply best to different problems.

In Chapter 9, Tracking Genetic Algorithms, on page ?, you'll learn about the different metrics and statistics you can track while running your genetic algorithms. You'll learn how to implement an evolutionary simulation using genetic algorithms, and you'll build statistics and genealogy tracking mechanisms around that problem.

In Chapter 10, Visualizing the Results, on page ?, you'll use the statistics collected in Chapter 9, Tracking Genetic Algorithms, on page ?, to create visualizations using different plotting tools. Next, you'll create a genetic algorithm that learns how to play Tetris, and you'll learn how to use different tools to watch your algorithm in action.

In Chapter 11, Optimizing Your Algorithms, on page ?, you'll work through a general optimization process to learn how to get the most performance out of your code. You'll learn how to use Elixir tools to benchmark and profile your algorithms. You'll learn how to write faster Elixir and faster algorithms.

You'll learn how to parallelize your algorithms and how to implement NIFs that run faster than pure Elixir code.

In Chapter 12, Writing Tests and Code Quality, on page ?, you'll learn how to use Elixir features and packages to test and type check your code. You'll learn a bit about writing tests that work well with randomness. You'll then learn how to write typespecs and how to verify your typespecs are correct.

In Chapter 13, Moving Forward, on page ?, you'll be introduced to a variety of practical applications of genetic algorithms. From artificial intelligence to finance to advertising, you'll learn how genetic algorithms are applied in practice, and you'll learn about how you can use them in almost any field.

How to Use This Book

Each chapter in this book builds on the last by making additions to a genetic algorithm framework in some meaningful way. You should read this book in successive order and follow along with the code examples as they are presented to you. If, for some reason, you want to skip around, you can download the code from each chapter on the book's web page.

How Does Elixir Fit In?

Before you start reading this book, you're likely wondering two things:

- Why would I do this in Elixir?
- How does Elixir fit in the bigger picture of genetic algorithm design?

Elixir is certainly not a popular choice for genetic algorithm design; however, that doesn't mean it's not a good choice.

First, the significant increases in available computing power over the last decade have meant the need for incredibly efficient code has diminished. That's not to say you shouldn't pay attention to efficiency and writing efficient code; however, the need to optimize code for low-power hardware has significantly decreased.

Second, as you'll see in Chapter 11, Optimizing Your Algorithms, on page ?, parallelism in Elixir is a straightforward task. The BEAM is especially optimized for running numerous processes at once, so writing and running parallel code is easy. Genetic algorithms are by nature very parallel. A portion of research[3] into genetic algorithms takes advantage of the parallelism offered by Erlang to experiment with parallel genetic algorithms.

3. http://personal.denison.edu/~lalla/MCURCSM2011/6.pdf

Finally, Elixir's syntax and design patterns lend themselves nicely to writing idiomatic genetic algorithms. As you'll see throughout this book, Elixir offers a number of useful features for creating a general framework for genetic algorithm design. This is not only excellent for learning but also for rapid prototyping of new ideas.

You might not choose to implement a production-level genetic algorithm in Elixir, but using Elixir to prototype and experiment can save you significant amounts of time and effort.

Now, it's time to get started writing your first genetic algorithm.