Application of Genetic Algorithms for the optimization of combinatorial problems

Final Year Project Thesis

Author: Juan Francisco Ariño Sales
Tutor: Federico Barber Sanchís

Course 2020 - 2021

Abstract

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Maecenas sit amet pharetra nibh. Cras dui magna, dignissim suscipit nibh ac, viverra tincidunt dolor. Integer facilisis dictum augue sit amet gravida. Curabitur hendrerit nulla quis felis euismod tincidunt. Nunc ullamcorper porta mi ut scelerisque. Morbi congue, dolor id tristique pharetra, nisi arcu hendrerit est, non porta lorem arcu sed nibh. Donec facilisis ex at massa suscipit iaculis. In pharetra, diam sed mattis ornare, libero quam molestie nisl, ut finibus nibh leo nec lectus. Quisque rutrum turpis quis consectetur finibus. Duis eget purus feugiat, sagittis magna a, lobortis ligula. Maecenas pretium efficitur risus, vel congue est. Quisque eu nisi vestibulum ligula pulvinar imperdiet sed non massa. Proin id tellus vitae elit eleifend molestie. Morbi sit amet elit molestie, hendrerit dolor eu, eleifend diam. Pellentesque ultricies neque lacinia diam porttitor vehicula.

Contents

Chapter 1

Introduction

1.1 Introduction

The Boolean satisfiability problem (SAT) consists of determining an assignment of variables such that a given Boolean expression evaluates to True, or proving no such expression exists; It is a very famous problem in logic and computer science since it was the first problem proven to be NP-Complete, thus finding an efficient general SAT solver is bound to advance the P vs NP problem, which is considered by many to be one of the most important open problems in computer science and is actually one of the unsolved[1] seven millennium problems stated by the Clay Mathematics Institute on May 24, 2000.
This work attempts to study the behavior of genetic algorithms applied to a restricted version of the SAT problem called the 3-SAT problem, which is still NP-complete. It will attempt to do so by analyzing the behavior of genetic algorithms based on different operators and hyper-parameters tested on a series of 3-SAT problems, and then grading them against each other, the best ranking genetic algorithms found will then be compared with some of the most popular open source SAT solvers.

1.1.1 Motivation

A genetic algorithm is a meta-heuristic designed to mimic the behavior of natural selection, it does so by evolving a population of candidate solutions through the use of biologically inspired operators, it is a population based random search algorithm (Cochran et al. 2011) TODO: ADD MORE MOTIVATION

1.1.2 Objectives

1.1.3 Methodology

1.1.4 Document Structure

[1] The Poincaré Conjecture is the only solved millennium problem

Chapter 2

Preliminaries

2.0.1 3-SAT Problem

The boolean satisfiability problem is the problem of determining, for a given boolean expression, if there exists an assignment of values, such that the formula evaluates to True, or proving no such assignment exists and therefore the formula will always evaluate to False. For example, having variables $\{x_1, x_2, x_3, x_4\}$ and the boolean expression:

$$x_1 \wedge (x_1 \vee \overline{x_2}) \wedge (x_2 \vee x_3 \vee \overline{x_4}) \wedge (x_1 \vee \overline{x_2} \vee \overline{x_4}) \wedge (x_1 \vee x_2 \vee \overline{x_3} \vee x_4)$$

It is trivial to find an assignment of variables such that the given expression evaluates to True (eg. $\{True, True, False, False\}$), consequently this expression is said to be *satisfiable*, but for this other boolean expression:

$$\overline{x_1} \wedge (x_1 \vee \overline{x_2}) \wedge (x_2 \vee x_3 \vee \overline{x_4}) \wedge (x_1 \vee \overline{x_2} \vee \overline{x_4}) \wedge (x_1 \vee x_2 \vee \overline{x_3} \vee x_4)$$

There exists no such assignment, the equation will never be solved by tweaking the variable values, therefore it is an *unsatisfiable* expression.
Any algorithm designed to solve SAT instances must distinguish between satisfiable and unsatisfiable problems however, barring some trivial or contradictory expressions, this can only be done by attempting to solve the actual problems, since no one can assert that no solution exists if they haven't searched the whole solution space.
Due to the inherently random nature of Genetic Algorithms one can not be sure the whole solution space has been searched and, in consequence, won't be able to assert with confidence that no solution exists, only that no solution has been found in the given amount of time, this is the main reason why this work will focus only on SAT instances which are known to be solvable.
All the boolean SAT expressions used throughout this work are in conjunctive normal form or CNF, this means that all formulas are a conjunction ($\wedge$) of clauses (or a single clause), these clauses contain the variables and the disjunction ($\vee$) operator, they can also contain the negation ($\neg$) operator; Both of the boolean formulas shown above are in conjunctive normal form.
3-SAT is a restricted version of SAT where each clause has exactly three variables in it, the problem of determining the satisfiability of a 3-SAT formula in conjunctive normal form is also NP-Complete (Karp 2010) (TODO:ADD more text??).

2.0.2 P vs NP

Computational complexity theory is a branch of mathematics and theoretical computer science, which tries to classify different mathematical problems according to the computational resources needed to solve them, usually the measured resources are time and storage space, though other complexity measures can also be used.
The analysis of complexity is done using a mathematical model of computation, this model allows us to analyze the performance of algorithms inside a theoretical computation machine, which means we can compare different algorithms without worrying about the details of their specific implementations.
The most common model of computation used is called a Turing machine, invented in 1936 by mathematician Alan Turing, a Turing machine is a theoretical machine which consists of four parts:

- An infinite *tape* divided into cells
- A *head* that can write and read symbols on the tape and can also move left or right along the tape
- A *state register* that stores the state of the Turing machine
- A finite table of *instructions*, which tell the head what to do based on the state and the symbols on the tape

The machine then operates on the tape, its behavior is defined in the instructions table where instructions are indexed by state and symbol, there are three types of possible instructions:

- Erase or write a symbol on the tape
- Move the head left or right one cell
- Change state.

The machine will begin by reading the first symbol on the tape and then executing instructions until it halts.
A Turing machine is a useful model of computation when trying to study an algorithm's performance on real machines, since anything that can be calculated using a traditional computer can also be computed using a Turing machine.
Different types of Turing machines can be used to define different complexity classes, the most commonly used are:

- *Deterministic Turing machines*, for each combination of state and symbol in the instructions table there exists only one instruction.
- *Non-deterministic Turing machines*, for each combination of state and symbol there can exist more than one instruction in the instructions table, therefore it is non-deterministic since it is not possible to know exactly the next state of the machine based on the current state and the tape symbol.

Though a non-deterministic Turing machine can be fully replicated using a deterministic Turing machine, it is nonetheless a useful abstraction because it allows us to generate computation trees with many branches, and as long as any one of these

branches reaches an accepting state, the machine will halt and accept, it will only reject once all the branches have reached their limit.
This is in contrast with deterministic Turing machines where the computation tree is a single branch with nodes following each other sequentially; A non-deterministic Turing machine can be recreated using a deterministic Turing machine by exploring the whole computation tree generated by the non-deterministic machine using tree traversal algorithms, but the resulting machine is much more convoluted and difficult to reason about. (TODO: Change image for one made by me)

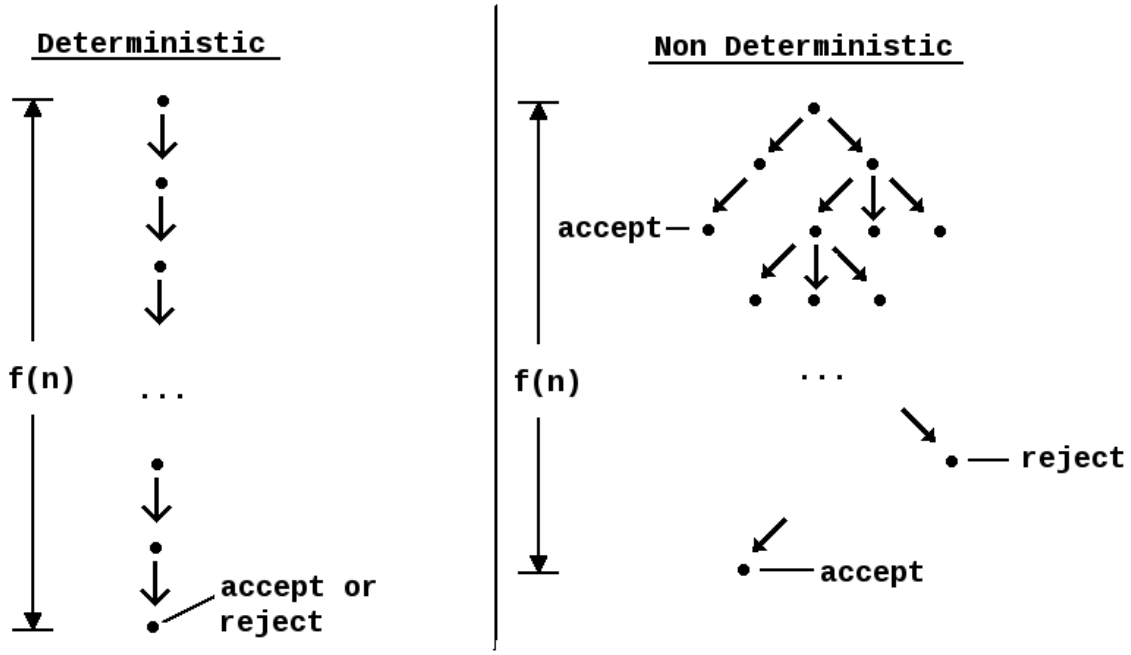

Figure 2.1: Left: Deterministic Turing machine computation tree. Right: Non-deterministic Turing machine computation tree

Different complexity classes are defined by establishing upper bounds to the resources available for the aforementioned Turing machines; For example the complexity class P is composed of all the problems which can be solved by a deterministic Turing machine in polynomial time, P is a class with a time constraint, the amount of space used is irrelevant for this categorization. There exist four fundamental classes based on the resources constrained:

- $DTIME[t(n)]$ is composed of all the problems which can be solved by a deterministic Turing machine in $t(n)$ amount of time
- $NTIME[t(n)]$ is composed of all the problems which can be solved by a non-deterministic Turing machine in $t(n)$ amount of time
- $DSPACE[s(n)]$ is composed of all the problems which can be solved by a deterministic Turing machine in $s(n)$ amount of space
- $NSPACE[s(n)]$ is composed of all the problems which can be solved by a non-deterministic Turing machine in $s(n)$ amount of space

Using these fundamental classes the complexity class P can be defined as:

$$P = DTIME[n^{O(1)}] = \bigcup_{k \geq 1} DTIME[n^k] \tag{2.1}$$

The boolean satisfiability problem was the first problem proven to be NP-Complete (Cook 1971)

2.0.3 Genetic Algorithms

2.0.4 State of the Art

Bibliography

Cochran, James J. et al. (2011). "Random Search Algorithms". In: *Wiley Encyclopedia of Operations Research and Management Science*. DOI: 10.1002/9780470400531.eorms0704.

Karp, Richard M. (2010). *Reducibility among combinatorial problems*. DOI: 10.1007/978-3-540-68279-0_8.

Cook, Stephen A. (1971). *The complexity of theorem-proving procedures*. DOI: 10.1145/800157.805047.