Honors Discrete Mathematics: Course Project Topics 10-20

Professor Gerandy Brito Spring 2023

Sarthak Mohanty

10 Genetic Algorithms

Genetic Algorithms (GAs) are a type of optimization algorithm inspired by the process of natural selection and genetics. They are used to find an optimal solution to a problem by mimicking the process of evolution.

A general framework for a GA starts with a set of solutions called the population. Each solution is represented by a chromosome, which is a binary string of bits. The chromosomes are then evolved through several generations, with each generation having a better solution than the previous one. The evolution process is done using a combination of selection, crossover, and mutation operations.

Selection is the process of choosing the best solutions from the current population to form the next generation. This is done using a fitness function that evaluates each solution and assigns it a score. The solutions with the highest scores are chosen as parents to create offspring for the next generation.

Crossover is the process of combining two solutions to create a new solution. It is done by randomly selecting a crossover point in the binary string of each parent and then swapping the bits between the two parents to create the offspring. This process creates new solutions that have a combination of the best traits from both parents.

Mutation is the process of randomly changing bits in the binary string to introduce new solutions into the population. This helps maintain genetic diversity and allows the GA to explore new solutions.

Main ideas for this project can go in many directions. You could explore the learning aspect through deep neural networks. You could also analyze the implementation of GAs and how they use techniques such as particle swarm or simulated annealing. A couple of possible areas for extension or generalizations are as follows:

- Study of hybrid genetic algorithms that combine GAs with other optimization algorithms
- Study of parallel and distributed GAs, including island models and particle swarm optimization
- Application of GAs to real-world problems, including optimization of engineering designs and financial portfolios

11 Cryptography in the Modern Age

Cryptography plays a vital role in ensuring secure communication in the modern age. In this project, we will be focusing on modern encryption algorithms, specifically Advanced Encryption Standard (AES). AES is a symmetric-key encryption algorithm that is widely used in various applications such as internet security protocols, electronic funds transfers, and secure communications.

The implementation of an encryption system using AES will be one of the main deliverables in this project. In addition to the implementation, we will delve into the mathematical foundations of AES and other encryption algorithms. This will include an in-depth analysis of the underlying mathematical concepts such as number theory, Galois fields, and cryptography-specific concepts like trapdoor functions.

A trapdoor function is a mathematical function that is easy to compute in one direction but difficult to compute in the opposite direction without knowing some additional information, called the trapdoor.

As an example, the trapdoor function in RSA encryption is the modular exponentiation function, $c = m^e \mod N$, where m is the plaintext, e is the public exponent, N is the modulus and c is the ciphertext. The trapdoor information is the private key, which consists of the factors of N. In the opposite direction, decryption is performed by computing $m = c^d \mod N$, where d is the private exponent derived from the private key.

Another example is the trapdoor function in Elliptic Curve Cryptography (ECC). In ECC, the trapdoor function is the scalar multiplication operation $Q = kP$, where P is a point on the elliptic curve, k is the scalar and Q is the result of the scalar multiplication. The trapdoor information is the private key, which is the scalar k. Decryption is performed by computing $Q = -kP$.

The trapdoor function in AES encryption is based on the substitution-permutation network (SPN) structure. The SPN structure consists of repeated rounds of substitution and permutation operations, where the substitution operation is performed by the use of S-boxes, and the permutation operation is performed by the use of linear transformations. The S-boxes are non-linear functions that scramble the input data, and their inverse operations are difficult to compute without the knowledge of the key used in AES encryption.

Specifically, let x be the plaintext and y be the ciphertext. The encryption operation in AES can be expressed as $y = E(x, K)$, where K is the secret key. The S-boxes used in AES are non-linear functions $S(x)$, and their implementation in AES can be expressed as:

$$y = S(x) \oplus K,$$

where $\oplus$ represents the bitwise exclusive-or operation. The inverse operation, decryption, can be expressed as:

$$x = S^{-1}(y) \oplus K$$

The inverse operation $S^{-1}(y)$ is difficult to compute without knowledge of the key K. This is the essence of the trapdoor function in AES encryption. The implementation of the S-boxes and their properties, as well as the use of the key schedule, make AES a secure encryption algorithm.

I recommend using this comic as a starting point.

12 Recursive Algorithms and Fractals

A *fractal* is a self-similar geometric shape that can be divided into smaller copies of itself. One way to generate fractals is through the use of recursive algorithms.

One famous example of a fractal is the *Koch snowflake*, which is created by starting with an equilateral triangle, and then dividing each line segment into three equal parts, and replacing the middle part with two line segments that form a smaller equilateral triangle. This process is repeated an infinite number of times, creating a fractal shape with an infinitely complex boundary. The Koch snowflake can be described mathematically as a fractal curve with the fractal dimension, D, which is a measure of its roughness, given by the equation $D = \log_3 4$.

Another example of a fractal is the *Mandelbrot set*, which is a set of complex numbers that exhibit complex, self-similar patterns when plotted on a graph. The algorithm for generating the Mandelbrot set is also recursive, and involves iterating the function $f_c(z) = z^2 + c$ for each value of c in the complex plane. The boundary of the Mandelbrot set is a fractal shape with a fractal dimension given by $D = 2$.

The mathematics involved in fractals and recursive algorithms includes complex numbers, iterative algorithms, and infinite sequences. The complex numbers are represented in the form $z = a + bi$, where a and b are real numbers and i is the imaginary unit, $i^2 = -1$. The iterative algorithm for the Mandelbrot set involves checking the behavior of the sequence defined by the recursive formula $z_{n+1} = f_c(z_n) = z_n^2 + c$ starting from $z_0 = 0$. The sequence is considered to be a member of the Mandelbrot set if it remains bounded in the complex plane, i.e. if $|z_n| < \infty$ for all n.

Your main task for this project should be where you demonstrate some of the recursive techniques that you have learned to create interesting fractals. It should also include some coding implementation of your techniques.

A couple of possible areas for extension or generalization are as follows:

- Study different types of fractals and the algorithms used to generate them, including the *Sierpinski triangle*, *dragon curve*, and *Julia sets*.
- Explore the properties of fractals, including their dimensionality and self-similarity, and how these properties relate to the underlying mathematical concepts, such as complex dynamics and iterated function systems.
- Study applications of fractals and recursive algorithms in areas such as computer graphics, data compression, and chaos theory, where the self-similarity and infinite detail of fractals can be utilized to achieve efficient representations of data and to model complex, dynamic systems.

13 Master Theorem

The Master Theorem is a result in the theory of algorithms that provides a framework for analyzing the running time of divide-and-conquer algorithms. Given a divide-and-conquer algorithm that solves a problem of size n by dividing the problem into subproblems of size n/b, then solving each subproblem in $O(n^d)$ time, and then combining the solutions of the subproblems in $O(n^c)$ time, the running time of the algorithm is given by the following:

$$T(n) = \begin{cases} O(n^d) & \text{if } d > c \\ O(n^c \log n) & \text{if } d = c \\ O(n^c) & \text{if } d < c \end{cases}$$

The Master Theorem is a powerful tool that simplifies the analysis of divide-and-conquer algorithms, but it has some limitations, and there are many variants of the theorem that are needed to handle specific cases.

In this project, we will prove the standard version of the Master Theorem and its most common variants, including the case when the running time of the divide step is $O(n^{\log_b a})$, and the case when the subproblems are not of equal size. There are also situations where the Master Theorem simply cannot hold, and one must use other techniques to solve recurrences.

A couple of possible areas for extension or generalization are as follows:

- Consider the case when the running time of the algorithm depends on additional parameters, such as the size of the input or the solution, or the number of subproblems. Analyze algorithms that involve additional parameters and generalize the Master Theorem to handle these cases.
- Consider the case when the subproblems are not solved in a divide-and-conquer manner, but are solved using other techniques, such as dynamic programming or greedy algorithms. Analyze algorithms that involve multiple problem-solving techniques and extend the Master Theorem to handle these cases.

While this topic may be titled "Master Theorem", it is also closely related to divide and conquer algorithms, which are widely used in computer algorithms for solving problems in computer graphics, cryptography, and computer networks. Students should find plenty to explore for this project.

14 Lambda Calculus

Lambda calculus is a mathematical formalism that is used to describe and study computation and algorithms. It was first introduced by Alonzo Church in the 1930s, and has since become an important area of research in theoretical computer science, programming languages, and logic.

In lambda calculus, functions are treated as first-class citizens and can be passed as arguments to other functions, assigned to variables, and manipulated in other ways. The basic building block of lambda calculus is the *lambda expression*, which is used to define anonymous functions. A lambda expression has the form

$$\lambda x.\ M$$

where x is the argument of the function and M is the expression representing the function's body.

The evaluation of a lambda expression is performed by substituting the argument x in M with a specific value and reducing the expression until it can no longer be simplified. This process is called *beta reduction*. The rules of beta reduction are well-defined and allow for the calculation of the value of a lambda expression.

Here are a few examples of lambda expressions:

- $\lambda x.\ x$ is the identity function, which simply returns its argument.
- $\lambda x.\ x + 1$ is a function that adds 1 to its argument.
- $\lambda x.\ \lambda y.\ x + y$ is a function that takes two arguments and returns their sum.

Lambda calculus has many interesting properties and features, including the ability to represent any computable function, the capability to encode simple data structures and algorithms, and the property of Turing completeness, which means that it is capable of simulating any algorithm that can be implemented on a Turing machine.

The aim of this project is to study the theory of lambda calculus, including its syntax, semantics, and reduction rules, as well as its applications in programming languages and computer science. In addition, students will implement a simple interpreter for lambda calculus in order to gain a deeper understanding of the subject.

Possible areas for extension or generalizations are as follows:

- Consider the theory of combinatory logic, which is a close relative of lambda calculus. Combinatory logic is characterized by the absence of variables, and as such, all functions are anonymous. The goal of this extension would be to study the theory of combinatory logic and to compare it with lambda calculus.
- Study the theory of type systems and type inference in the context of lambda calculus. This would include an introduction to Hindley-Milner type systems, the study of type inference algorithms, and the relationship between type systems and the ability to reason about the behavior of programs.
- Consider the use of lambda calculus in functional programming languages, such as Haskell, Scheme, and ML. Study the theory and practice of functional programming, and implement simple programs in one or more functional programming languages.

15 Godel's Incompleteness Theorem

Godel's Incompleteness Theorem is one of the most famous results in the foundations of mathematics. It was proved by Kurt Gödel in the early 20th century and states that, within any sufficiently powerful mathematical system, there will always exist statements that are true, but unprovable within the system.

To understand the theorem, we first need to understand the idea of Godel numbering. Given a mathematical system, we can assign a unique integer (referred to as a "Godel number") to each mathematical statement, symbol, or proof. This allows us to encode mathematical statements as integers, and perform operations on them using arithmetic. For example, the statement "$2+2=4$" can be encoded as the integer 123456. The statement "For all x, the statement '$x+x=2x$' is true" can also be encoded as an integer, say 654321.

Once we have assigned Godel numbers to our mathematical statements, we can use these numbers to reason about the statements themselves. For example, we can define the statement "A implies B" as the statement "$\neg A \vee B$", where $\neg$ represents negation and $\vee$ represents disjunction. This means that if A is the statement "$x+x=2x$" and B is the statement "$x=x$", then the statement "A implies B" is equivalent to the statement "$\neg(x+x=2x) \vee (x=x)$". We can even encode this statement as its own Godel number! This encoding is how Godel proved his Incompleteness Theorems.

Godel's Incompleteness Theorem is significant because it shows that within any sufficiently powerful mathematical system, there will always be statements that are true but cannot be proven within the system itself. This highlights the limits of mathematical systems and the need for a more comprehensive understanding of mathematical truth. The theorem is a crucial contribution to the foundations of mathematics and continues to be a topic of interest and research in the field today. In this project, you'll have the opportunity to further understand the theorem by encoding mathematical statements as Godel numbers and studying the first and second incompleteness theorems in detail.

I **highly** recommend using this article as a starting point.

16 Combinatorial Games

A combinatorial game is a two-player game with perfect information, where the players make moves in turn, with the aim of being the last player to make a move. In this course, we will focus on two particular games: Sprouts, and Dots and Boxes.

Sprouts: Sprouts is a game played on a sheet of paper, with a pen. The game starts with a dot on the paper, and players take turns adding a new dot and connecting it to one of the existing dots by a line. The line cannot cross over any other lines. The game ends when a player is unable to add a new dot. The player who adds the last dot is the winner.

Dots and Boxes: Dots and Boxes is played on a square grid of dots. The players take turns connecting two adjacent dots by a line. If a player completes a box, they write their symbol inside the box and take another turn. The game ends when all boxes have been completed. The player with the most boxes is the winner.

To study these games, we will use the theory of impartial combinatorial games, which assigns a mathematical value to each game, known as the nim-value or Grundy number. This allows us to determine who has a winning strategy and who does not, and also to combine games to form more complex games.

Sprague-Grundy Theorem. Let $G_1, G_2, \ldots, G_n$ be combinatorial games. The game $G = \bigoplus_{i=1}^{n} G_i$ is defined to be the game where players can make moves in any of the games $G_1, G_2, \ldots, G_n$ in any order they choose, but once a player is unable to move in a game, they cannot make any further moves in that game. Then the nim-value of the game G is equal to the nim-value of the game $G_1 \oplus G_2 \oplus \cdots \oplus G_n$.

A couple of possible areas for extension or generalizations are as follows:

- Consider the game of NIM, where the players take turns removing stones from a heap. Investigate the relationship between the size of the heap and the nim-value of the game.
- Generalize the Sprague-Grundy theorem to m-player games, where $m \geq 2$ players take turns making moves. Investigate how to calculate the nim-value of such games, and the conditions under which a player has a winning strategy.
- Consider games where the moves are not deterministic. Investigate how to extend the Sprague-Grundy theorem to these games, and the conditions under which a player has a winning strategy.

17 Create Your Own Game

In this project, you will design, create, and analyze a math-based game. The game must be complex enough to provide a challenge to players, and must involve a significant mathematical component, such as strategy and problem-solving.

There are many different types of math-based games to choose from, but yours should be completely unique. When selecting a game to create, consider the following questions:

- What mathematical concepts are involved in the game?
- What is the objective of the game?
- How is the game played? What are the rules and how do they impact strategy?
- What are some examples of similar games and what makes your game unique?

Once you have selected a game to create, you will need to carefully design and implement it. This will involve creating rules, designing a board or playing field, creating pieces or tokens, and possibly even coding a computer version of the game.

After you have created the game, you will need to analyze it. This will involve studying the various strategies and tactics that players can use, as well as analyzing the mathematical aspects of the game. You may also want to play the game yourself or have others play it to see how it works in practice.

The final step of the project will be to write a report on your game. This should include the following elements:

- An introduction to the game and its mathematical foundations
- A description of the rules and how the game is played
- An analysis of the game's mathematical properties, such as strategies, tactics, and any relevant theorems or equations
- An explanation of any coding or programming that was involved in creating the game
- A discussion of any challenges or limitations encountered during the creation process
- Conclusion and future directions for the game.

Good luck and have fun creating your math-based game!

18 Monte Carlo Methods

Monte Carlo methods are a family of computational algorithms that rely on repeated random sampling to compute numerical results. They are widely used in fields such as physics, finance, and engineering, among others, where mathematical models are too complex to solve analytically.

The basic idea behind Monte Carlo methods is to generate random samples from a given distribution, and then use these samples to estimate the desired quantity. For example, in physics, Monte Carlo methods can be used to compute the properties of complex systems, such as molecules or materials, by generating random configurations and computing the properties of interest for each one.

In this project, we will focus on a specific type of Monte Carlo method: Markov Chain Monte Carlo (MCMC). MCMC methods are a class of algorithms that generate random samples from a target distribution by constructing a Markov Chain that has the target distribution as its stationary distribution. The samples generated by the MCMC method are then used to estimate the desired quantity.

The main theorem in this area is the Metropolis-Hastings algorithm, which provides a general framework for constructing MCMC methods. The Metropolis-Hastings algorithm works as follows:

- Choose an arbitrary starting state.
- At each step, generate a proposal state from the current state using a proposal distribution.
- Compute the acceptance probability of the proposal state, based on the target distribution and the proposal distribution.
- With probability equal to the acceptance probability, accept the proposal state as the next state. Otherwise, reject the proposal state and stay in the current state.
- Repeat the above steps to generate a sequence of states, which form a Markov Chain.

The Metropolis-Hastings algorithm can be seen as a generalization of the Metropolis algorithm, which is a special case of the Metropolis-Hastings algorithm where the proposal distribution is symmetric.

A couple of possible areas for extension or generalizations are as follows:

- Consider different types of proposal distributions, such as Gaussian distributions or random walks, and compare their performance in terms of convergence to the target distribution and computational efficiency.
- Consider the use of parallel and distributed computing techniques to speed up the computation. For example, consider using GPU programming or cloud computing to run large-scale Monte Carlo simulations.
- Consider the use of adaptive MCMC methods, which dynamically update the proposal distribution based on the history of the Markov Chain, in order to improve the convergence and efficiency of the simulation.

19 Recommendation Systems (Linear Algebra Required)

A *recommendation system* is a system that predicts the "rating" or "preference" a user would give to an item. It is commonly used in a variety of applications such as movie recommendation, product recommendation, etc. They are used everywhere nowadays, from Google advertisements to TikTok videos.

In this project, we will consider a specific type of recommendation system, *collaborative filtering*. Collaborative filtering is based on the idea that users who have similar preferences in the past are likely to have similar preferences in the future.

Given a user-item matrix R with entries r_{ij} denoting the rating user i gives to item j, the goal of collaborative filtering is to estimate the missing entries in the matrix. One popular method to accomplish this is *matrix factorization*. The idea is to factorize the matrix R into two matrices U and V, where U is a user-feature matrix and V is an item-feature matrix. That is,

$$R \approx UV^T$$

where $U_{i,:}$ is a feature vector for user i and $V_{j,:}$ is a feature vector for item j.

The optimization problem of matrix factorization is to minimize the difference between the estimated ratings and the actual ratings:

$$\min_{U,V} \sum_{(i,j)\in\Omega} (r_{ij} - u_i^T v_j)^2$$

where Ω is the set of observed entries in the matrix R.

A couple of possible areas for extension or generalizations are as follows:

- Consider incorporating additional information such as user demographics or item features into the recommendation system. For example, we can add a bias term for each user and item, and a feature term for each item, resulting in the following equation:

$$r_{ij} = u_i^T v_j + b_i + c_j + \sum_{k=1}^{d} x_{j,k} w_k$$

  where b_i and c_j are biases for user i and item j, and $x_{j,k}$ is a feature of item j and w_k is a feature weight.

- Consider the case where the ratings are not explicit, but instead are binary (e.g., a user either clicked or did not click on an item). One popular method in this case is *matrix completion with implicit feedback*. The idea is to assume that the observed binary feedback can be considered as an implicit rating and use matrix factorization to estimate the missing ratings.

The coding component of this project should be relatively clear: develop a recommendation algorithm, and demonstrate its functionality in some way.

20 Unsolved Problems in Mathematics

A *mathematical problem* is a statement or question that can be answered through mathematical methods and reasoning. Many of these problems have been solved over time, however, some remain unsolved.

In this course, we will focus on some of the most famous and important unsolved problems in mathematics. These problems come from a variety of areas such as number theory, geometry, analysis, and combinatorics.

One of the oldest unsolved problems is the *Pell's equation*, which is a Diophantine equation of the form

$$x^2 - dy^2 = 1$$

where $d \in \mathbb{N}$ and d is not a perfect square. The aim of this project is to find an algorithm that finds all solutions to Pell's equation, or prove that no such algorithm exists.

Another famous problem is the *Riemann Hypothesis*, which states that the non-trivial zeros of the Riemann zeta function $\zeta(s)$ are all located on the critical line of $1/2$. Proving or disproving this hypothesis remains one of the most important open problems in mathematics, and it has significant implications in number theory and prime number distribution.

Another intriguing problem is the *Collatz Conjecture*, which is a conjecture in the field of discrete mathematics. Given any positive integer n, we define the following sequence:

$$n \to \frac{n}{2} \quad \text{if } n \text{ is even,}$$

$$n \to 3n + 1 \quad \text{if } n \text{ is odd.}$$

The Collatz Conjecture states that this sequence always reaches 1, regardless of the starting value of n.

Your main task will be to take a close look at some of these problems, read research papers, and try to udnrestand some of the recent developments surrounding the problem. **In my opinion, this is the most difficult project.**

A couple of possible areas for extension or generalizations are as follows:

- Consider other famous unsolved problems such as the *Goldbach Conjecture* (every even integer greater than 2 can be expressed as the sum of two prime numbers), the *Hodge Conjecture* (compact Kähler manifolds are projective), or the *Birch and Swinnerton-Dyer Conjecture* (the rank of an elliptic curve is equal to the order of vanishing at infinity of its L-function). Research these problems, and try to create visualizations.
- Consider a problem from another area of mathematics, such as *Navier-Stokes Equation* (describing the motion of fluids), *Yang-Mills Equation* (describing the behavior of particles in quantum field theory), or the *Mass Gap Conjecture* (massless particles cannot appear in theories of massive particles). Research these problems and attempt to find some significant results.