International Journal of Advance and Applied Research
www.ijaar.co.in
ISSN – 2347-7075 Impact Factor – 8.141
Peer Reviewed Bi-Monthly
Vol. 7 No. 5 January – February - 2026

AI in Mathematical Algorithms

S. H. Kurhade
Department of Mathematics, Sharadchandra Pawar Mahavidyalaya, Lonand
DOI - 10.5281/zenodo.18899129

Abstract:

While learning mathematics has traditionally been a significant challenge for many, the advancement of artificial intelligence offers a powerful solution. By diagnosing individual learning obstacles and providing personalized support, AI technologies can optimize student performance and help overcome the difficulties often associated with math courses. This paper presents a systematic study of various AI algorithms used in mathematics. We have discussed here two mathematical algorithms namely gradient descent algorithm and genetic algorithm with examples. By integrating core concepts like linear algebra, calculus, and statistics, AI algorithms can process data, identify patterns, and optimize solutions. These mathematical foundations serve as the backbone for everything from image recognition to complex modeling, enabling AI to solve problems faster than humans and uncover insights that might otherwise be missed. Furthermore, these algorithms assist in generating new mathematical conjectures, providing personalized learning experiences, and even automating the generation of proofs.

Keywords: AI, Artificial Intelligence, Mathematical Algorithms, Genetic Algorithm, Gradient Descent Algorithm

Introduction:

"What is AI?" This is a question that may arise in everyone's mind. Basically most researchers never bother themselves with this question. While the number of papers specifically dedicated to defining AI is relatively small, several notable studies exist. Comprehensive overviews of these definitions can be found in the work of Wang (2019) [1] and the extensive research of Hernández-Orallo (2012, 2014a, 2014b, 2014c, 2017)[2][3][4][5][6]. The first intuitive (informal) definition of AI was provided by Alan Turing and is known as the Turing Test (Turing, 1950)[7]. While that definition is remarkably simple, it presents a significant problem: the Turing Test measures "trained intellect" (i.e. intelligence + education). Ideally, we require a definition of "untrained intellect" (i.e. pure intelligence independent of education). The first such definition of pure intelligence was proposed by Pei Wang in 1995[8]. It reads as follows: Intelligence is the capacity of an information-processing system to adapt to its environment while operating with insufficient knowledge and resources. This definition is very important because it is the first definition that separates intelligence from education. Later, in 2000, a simplified (refined) version of Pei Wang's definition appeared. That version was published in Dobrev (2000)[9]. Today, it is the first result listed by Google on the topic of AI Definition. The first result returned by Google in response to a query for Definition of Artificial Intelligence is the paper of Dobrev (2005a)[10], which is an improved version of Dobrev (2000). Here is the simplified version of Pei Wang's definition: AI will be such a program which in an arbitrary world will cope not worse than a human. The definition from the AI HLEG provides the necessary freedom of scope. Describing AI as

"systems that display intelligent behavior by analyzing their environment and taking actions – with some degree of autonomy – to achieve specific goals", this encompasses all the applications we currently qualify as AI and at the same time provides scope for future changes to that qualification.[13]

In recent time the best definition of Artificial Intelligence (AI) is the ability of a computer or machine to perform tasks that typically require human intelligence, like learning, problem-solving, recognizing patterns, understanding language, and making decisions, essentially mimicking human cognitive functions to operate autonomously or assist humans. It's the "brain" behind things like voice assistants (Siri, Alexa) or recommendation systems (Netflix), using data to learn and act without constant human programming.

The term 'algorithm' is derived from the name of the ninth-century Persian mathematician Mohammed ibn Musa al-Kharizmi and refers to a specific instruction for solving a problem or performing a calculation. The intersection of AI and Mathematics is a two-way street: while math provides the foundational "engine" for AI, AI algorithms are increasingly being used to solve, prove, and discover new mathematical concepts. People use algorithms all the time, without even knowing it. In many cases, people work, travel, cook, and do many other things according to algorithms. For example, we may speak about algorithms for counting, algorithms for going from Pune to Mumbai or to some other place, algorithms for chip production or for buying some goods, products or food. Famous Gödel incompleteness theorem [11] for formal systems is true only when conventional algorithms are used for proofs and this explains how people solve problems related to incomplete systems with undecidable properties. In modern mathematics, different types of algorithms model computers and their software.[12]

Different Algorithms used in AI:

Listing all algorithms in AI is nearly impossible because new ones are developed every week. However, AI algorithms are generally categorized by how they learn or the specific problem they solve. Here is a comprehensive breakdown of the essential algorithms used in modern AI.

Machine Learning (Core Algorithms):

These are the foundational "workhorses" that learn from data to make predictions or find patterns. The following are some examples of these types of algorithms. Linear Regression: Predicts continuous values (e.g., house prices). Logistic Regression: Predicts categories (e.g., spam vs. not spam). Decision Trees: Uses a flowchart-like structure to reach a conclusion. Random Forest: An "ensemble" of many decision trees to improve accuracy. Support Vector Machines (SVM): Finds the best boundary (hyperplane) to separate data classes. Naive Bayes: A probabilistic algorithm often used for text classification. K-Nearest Neighbors (KNN): Classifies data based on how close it is to other examples. Gradient Boosting (XGBoost, LightGBM): Sequentially builds models to correct the errors of previous ones. K-Means Clustering: Groups data into "K" number of clusters based on similarities. Principal Component Analysis (PCA): Reduces the complexity (dimensions) of data while keeping key info. Hierarchical Clustering: Builds a tree of clusters to show relationships.

Apriori Algorithm: Used for association rule learning (e.g., "people who buy bread also buy butter").

Deep Learning (Neural Networks):

Deep learning uses layers of "neurons" to process complex data like images, audio, and text. Examples are Multilayer Perceptron (MLP): The basic "vanilla" neural network. Convolutional Neural Networks (CNN): The gold standard for Computer Vision (facial recognition, medical scans). Recurrent Neural Networks (RNN): Used for sequential data like time series or speech. Long Short-Term Memory (LSTM): An advanced RNN that can remember information for long periods. Transformers: The architecture behind ChatGPT and modern NLP. They use "attention" to understand context. Generative Adversarial Networks (GANs): Two networks competing to create realistic data (images, deepfakes). Diffusion Models: The math behind modern image generators like Midjourney and DALL-E.

Search and Optimization Algorithms:

These are often used in robotics, gaming (like Chess or Go), and pathfinding. Examples are A (A-Star):* Finds the shortest path between two points. Minimax: Used in two-player games to minimize the possible loss for a maximum gain.
Genetic Algorithms: Mimics biological evolution (mutation and selection) to find the "fittest" solution. Simulated Annealing: A probabilistic technique to find the global optimum in a large search space. Alpha-Beta Pruning: Optimizes the Minimax algorithm by ignoring branches that won't be chosen.

Reinforcement Learning (RL):

These algorithms learn through trial and error, receiving "rewards" for good actions. Examples are Q-Learning: A model-free algorithm that learns the value of an action in a particular state. Deep Q-Networks (DQN): Combines Q-Learning with Deep Learning (used by DeepMind to play Atari games). Proximal Policy Optimization (PPO): A popular algorithm for training robots and game-playing agents.

Domain-Specific Algorithms:

These algorithms are used in specific domains or filed like: Computer Vision: Canny Edge Detection, YOLO (You Only Look Once) for real-time object detection, and SIFT. NLP: Word2Vec (word embeddings), BERT, and PageRank (used by Google Search).

AI Algorithms for Solving Math:

In recent years, researchers have developed AI that can perform high-level mathematical reasoning, moving beyond simple arithmetic, this is done with the help of different algorithms. Some of them listed below.

Symbolic Computation & Solvers:

Traditional calculators use "numerical" methods (approximations), but AI-driven systems like WolframAlpha or SymPy use symbolic logic to manipulate variables. Capabilities: Factoring huge polynomials, solving differential equations, and performing exact integration.

Automated Theorem Proving (ATP):

AI algorithms like Lean, Coq, and Isabelle: are used to verify and even generate formal mathematical proofs. Neural Theorem Proving: New models use Reinforcement Learning (RL) to "search" through a tree of logical steps to find a proof.
Reinforcement Learning: Algorithms like GRPO (Group Relative Policy Optimization) are specifically designed to help AI models "reason" through multi-step math problems by rewarding logical consistency rather than just the final answer.

Pattern Discovery in Higher Math:

DeepMind's AI has famously been used to find "conjectures" (unproven mathematical ideas) in knot theory and representation theory. The AI looks at massive datasets of mathematical objects and identifies relationships that human mathematicians hadn't noticed.

Explaining some Algorithms with examples:

We will explain here two algorithms namely Gradient Descent and Genetic Algorithm. The Gradient Descent algorithm is fundamentally inspired by the physical and intuitive experience of navigating a landscape to find its lowest point. The Gradient Descent algorithm was first developed by the French mathematician Augustin-Louis Cauchy in 1847. Whereas The Genetic Algorithm (GA) is fundamentally inspired by Charles Darwin's theory of natural evolution, specifically the principle of "Survival of the Fittest." Developed by John Holland in the 1960s and 70s, the algorithm mimics the biological processes that allow species to adapt to their environments over many generations.

Gradient Descent:

The "Engine" of AI:

Gradient Descent is the foundational algorithm used to train neural networks. If you imagine a "Cost Function" as a hilly landscape, the algorithm's goal is to find the lowest valley (the point of minimum error). How it works mathematically:

The algorithm updates the model's parameters (weights) using the following formula:

$x_{n+1} = x_n - \alpha * \nabla f(x_n)$, x_n : Your current position (current weights). α : The Learning Rate. This determines how big of a "step" the AI takes. Too large, and it might overstep the valley; too small, and it will take forever to learn. $\nabla f(x_n)$: The Gradient. This is the vector of partial derivatives that points in the direction of steepest ascent. By subtracting it (using the minus sign), we move in the direction of steepest descent.

let's look at how to code a basic Gradient Descent algorithm.

Example 1: Find the minimum of a function: $f(x) = x^2, x \geq 0$ using gradient descent algorithm.

Solution: Manual answer is $x = 0$.

Answer by using Gradient Descent algorithm: The Python Implementation

Python code:

```
def gradient_descent(start_x, learning_rate, iterations):
    x = start_x

    for i in range(iterations):
        # 1. Calculate the derivative (gradient) of f(x) = x^2
        # The derivative is 2x
        gradient = 2 * x

        # 2. Update x by moving against the gradient
        x = x - (learning_rate * gradient)

        print(f"Iteration {i+1}: x = {x:.4f}, f(x) = {x**2:.4f}")

    return x

# Run the algorithm
# Start at x = 10, Step size = 0.1, do it 20 times
final_x = gradient_descent (start_x=10, learning_rate=0.1, iterations=20)
```

Output :

Iteration 1: x = 8.0000, f(x) = 64.0000	Iteration 2: x = 6.4000, f(x) = 40.9600
…… ……	Iteration 20: x = 0.1153, f(x) = 0.0133

Genetic Algorithm:

A Genetic Algorithm (GA) is a powerful search/optimization method in computer science, inspired by natural evolution, that finds near-optimal solutions to complex problems by mimicking natural selection, heredity, crossover, and mutation to evolve a population of candidate solutions over generations. It represents solutions as "chromosomes," evaluates their "fitness," and iteratively selects, combines (crossover), and slightly alters (mutation) the best ones to create better offspring, eventually converging on high-quality answers for tasks like optimization and machine learning. We explain this algorithm by using below example.

Example 2. Solving the 8 Queens Problem using a Genetic Algorithm (GA) is a fascinating way to find a solution without checking every possible combination (which totals 4,426,165,368). The goal is to place 8 queens on an 8×8 chessboard so that no two queens threaten each other (none in the same row, column, or diagonal).

Solution:- Genetic Algorithm involves following steps

Step 1: Representing individuals (i.e. Chromosomes)

Instead of a 2D grid, we represent the board as an array of 8 numbers. Each index represents a column, and the value at that index represents the row position of the queen.

Example Chromosome: [3, 2, 7, 5, 2, 4, 1, 1]

This represents a queen at (Col 1, Row 3), (Col 2, Row 2), and so on.

Step 2: Generate initial population of size 4 (we can select any size)

Generate random arrangement of 8 queens on chess board as below

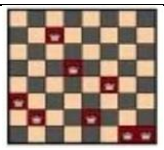
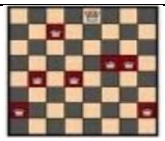
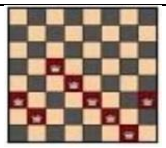
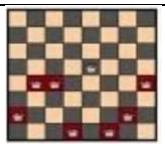
Chromosome 1 = [3,2,7,5,2,4,1,1] , Chromosome 2 = [2,4,7,4,8,5,5,2]

Chromosome 3 = [3,2,5,4,3,2,1,3] , Chromosome 4 = [2,4,4,1,5,1,2,4]

Step 3 : Apply fitness function

Here, Fitness = No. of non attacking pairs

Table: 1

Chromosome	**Queen positions**	**Fitness (f_i)**		**Total fitness $f = \sum f_i$**	**Fitness % = $\frac{f}{\sum f}$**
Chromosome 1		Queen 1 = 6 Queen 2 = 5 Queen 3 = 4 Queen 4 = 3	Queen 5 = 3 Queen 6 = 2 Queen 7 = 0 Queen 8 = 0	23	29%
Chromosome 2		Queen 1 = 6 Queen 2 = 5 Queen 3 = 4 Queen 4 = 4	Queen 5 = 3 Queen 6 = 1 Queen 7 = 1 Queen 8 = 0	24	31%
Chromosome 3		Queen 1 = 3 Queen 2 = 4 Queen 3 = 1 Queen 4 = 1	Queen 5 = 0 Queen 6 = 1 Queen 7 = 1 Queen 8 = 0	11	14%
Chromosome 4		Queen 1 = 5 Queen 2 = 4 Queen 3 = 3 Queen 4 = 3	Queen 5 = 3 Queen 6 = 1 Queen 7 = 1 Queen 8 = 0	20	26%

Step 4: Selection

We select two chromosomes for crossover using Roulette wheel selection method, in this method we arrange all the chromosomes on wheel according to their fitness percentage, this wheel has two arrows facing to each other as shown in figure 1. When spinning a standard roulette wheel, each result has the same chance of occurring. In roulette wheel selection, however, each result is weighted such that some are more likely to occur than others. Hence the chromosomes having larger fitness factor have

more likely to occur than others. Now let us spin the roulette wheel in clockwise direction, suppose wheel stopped in the position so that arrows are pointing towards the chromosomes 2 and chromosome 4 shown in figure1. So, we select the chromosome 2 and chromosome 4 for crossover.

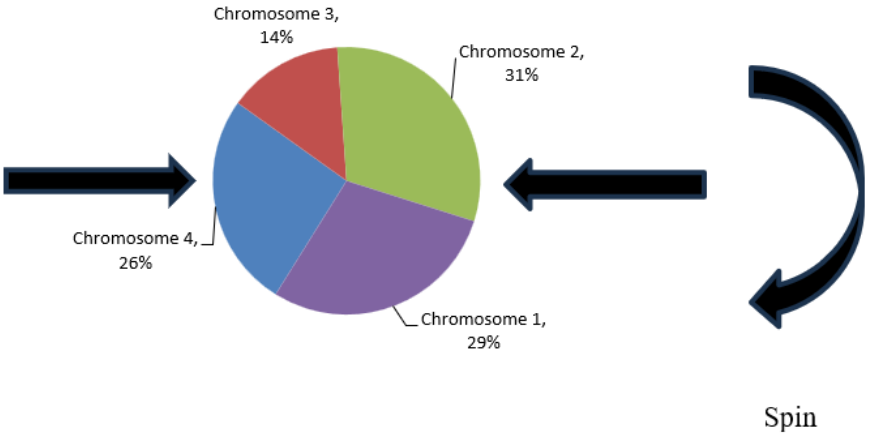


Figure 1

Step 5 : Crossover

In this step we do single point crossover to get next generation as below, here we interchange the part of chromosomes 2 and 4 after the dotted line so as to get new pair of chromosomes 5 and 6 as shown in figure 2.

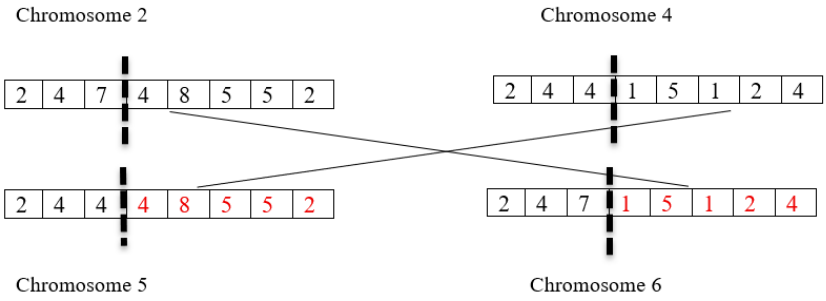


Figure 2

Step 6 : Mutation

In mutation, we change any single gene of both Chromosome so as to get randomness in the selection as shown below in figure 3

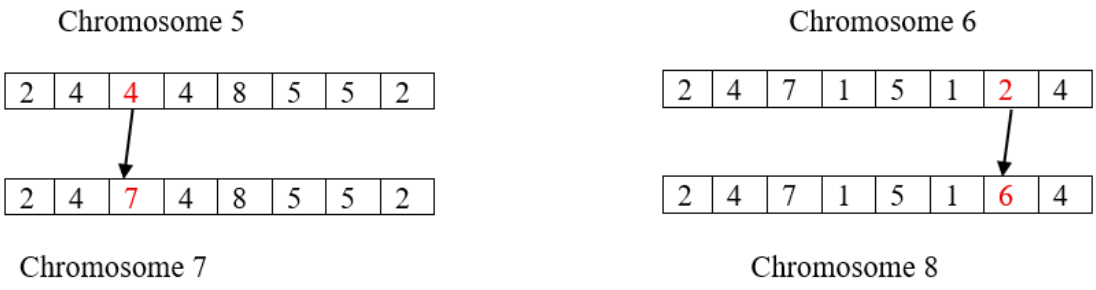


Figure 3

Step 7 : Repeat Steps 1 to 6 until we get best solution.

In this example Best solution = Highest fitness score (28 in this case)

Python code

```
import random
def get_fitness(chromosome):
    """Calculates the number of non-attacking pairs (Max: 28)."""
    clashes = 0
    n = len(chromosome)
    for i in range(n):
        for j in range(i + 1, n):
            # Same row clash
            if chromosome[i] == chromosome[j]:
                clashes += 1
            # Diagonal clash
            if abs(i - j) == abs(chromosome[i] - chromosome[j]):
                clashes += 1
    return 28 - clashes

def reproduce(p1, p2):
    """Single-point crossover."""
    n = len(p1)
    c = random.randint(0, n - 1)
    return p1[:c] + p2[c:]

def mutate(chromosome):
    """Randomly changes one queen's position."""
    n = len(chromosome)
    c = random.randint(0, n - 1)
    m = random.randint(1, n)
    chromosome[c] = m
    return chromosome

def genetic_queen(population, fitness_fn):
    mutation_probability = 0.1
    new_population = []

    # Selection based on fitness probability
    probabilities = [fitness_fn(ind) for ind in population]
    total = sum(probabilities)
    norm_probs = [p/total for p in probabilities]

    for _ in range(len(population)):
        # Pick two parents
        p1, p2 = random.choices(population, weights=norm_probs, k=2)
        child = reproduce(p1, p2)

        # Chance to mutate
        if random.random() < mutation_probability:
            child = mutate(child)

        new_population.append(child)
        if fitness_fn(child) == 28: break

    return new_population

# --- Run the Algorithm ---
population_size = 100
population = [[random.randint(1, 8) for _ in range(8)] for _ in range(population_size)]
generation = 0

while True:
    generation += 1
    best_individual = max(population, key=get_fitness)
    current_fitness = get_fitness(best_individual)

    if generation % 100 == 0:
        print(f"Generation {generation}: Best Fitness = {current_fitness}")

    if current_fitness == 28:
        print(f"\nSolved in Generation {generation}!")
        print(f"Solution Chromosome: {best_individual}")
        break
    population = genetic_queen(population, get_fitness)
```

Output:

Solved in Generation 219!

Solution Chromosome: [6, 2, 7, 1, 3, 5, 8, 4]

Future Scope:

As a very few researchers have concentrated on giving examples in paper of Mathematical algorithms in AI. So, there is scope in future to discuss with example the algorithms like

- Particle Swarm Optimization
- Bat search algorithm
- Cuckoo search algorithm

Conclusion:

This paper examines various types of Artificial Intelligence algorithms and their practical applications, covering Machine Learning (core algorithms), Deep Learning (neural networks), Search and Optimization algorithms, Reinforcement Learning (RL), and Domain-Specific algorithms. Our main focus in this paper is on discussing various types of mathematical algorithms used in Artificial intelligence in doing so we have taken two algorithms gradient descent algorithm and genetic algorithm to study. We have solved one example of each algorithm and it is found that these algorithms are really very useful to solve problems in mathematics. It saves the time needed for calculations and it improves the accuracy while solving mathematical problems. From the example solved using genetic algorithm it can be observe that 8 queen's placing problem was really very difficult and time consuming, but with the help of genetic algorithm we can solve this problem within a minute. So, mathematical algorithms are really helping us to solve a complex and critical problems arises in any area of science and technology and even in our day to day life also.

References:

1. Wang, P. (2019) On Defining Artificial Intelligence. Journal of Artificial General Intelligence 10(2) 1-37, 2019.
2. Dowe, D.L., & Hernández-Orallo, J. IQ tests are not for machines, yet. Intelligence (2012), doi:10.1016/j.intell.2011.12.001
3. Dowe, DL.; Hernández Orallo, J. (2014a). How universal can an intelligence test be?. Adaptive Behavior. 22(1):51-69. doi:10.1177/1059712313500502.
4. Hernández-Orallo, J.; Dowe, DL.; Hernández Lloreda, MV. (2014b). Universal psychometrics: measuring cognitive abilities in the machine kingdom. Cognitive Systems Research. 27:50-74, ISSN:1389-0417. doi:10.1016/j.cogsys.2013.06.001.
5. Javier Insa-Cabrera, José Hernández-Orallo (2014c). Definition and properties to assess multi-agent environments as social intelligence tests. arXiv:1408.6350 [cs.MA].
6. Hernández-Orallo, J. (2017) Evaluation in artificial intelligence: from task-oriented to ability-oriented measurement. Artificial Intelligence Review 48, 397–447, ISSN:0269-2821.
7. Turing, A. (1950) Computing machinery and intelligence. Mind, LIX:433–460
8. Wang, P. (1995) Non-Axiomatic Reasoning System: Exploring the Essence of Intelligence. Ph.D. Dissertation, Indiana University
9. Dobrev, D. (2000). AI - What is this. PC Magazine - Bulgaria, 11/2000, pp.12-13 (on https://dobrev.com/AI/definition.html in English).
10. Dobrev, D. (2005a). A Definition of Artificial Intelligence. In: Mathematica

Balkanica, New Series, Vol. 19, 2005, Fasc. 1-2, pp.67-74.

11. Gödel, K., 1995, Collected Works III. Unpublished Essays and Lectures, S. Feferman et al. (eds.), Oxford: Oxford University Press
12. Dreyfus, Hubert L. What Computers Can't Do: A Critique of Artificial Reason. New York: Harper and Row, 1972.
13. High-Level Expert Group on Artificial Intelligence. (2019). A definition of AI: Main capabilities and scientific disciplines. European Commission. Available at: https://ec.europa.eu/newsroom/dae/document.cfm?doc_id=56341