People's Democratic Republic of Algeria
Ministry of Higher Education and Scientific Research
Mohamed Khider University of Biskra
FACULTY OF EXACT SCIENCES
DEPARTMENT OF MATHEMATICS

Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of

"**Master in Mathematics**"

Option: Partial Differential Equations and Numerical Analysis

Submitted and Defended By

Sayah Mohamed El Amir

Title:

Metaheuristic Algorithms.

Examination Committee Members:

Dr.	GUIDAD Daradji	MKUB	President
Dr.	Ghedjemis Fatiha	MKUB	Supervisor
Dr.	OUAAR Fatima	MKUB	Examiner

02/06/2025

Dedicace

"And the conclusion of their call will be, Praise be to Allah, Lord of the Worlds".
Praise be to Allah at the beginning and at the end.
I would like to express my deepest gratitude and appreciation to my beloved family for their unconditional love, support, and encouragement. My mother and father are gems in my life. My brother is someone I can lean on during difficult times, and I am grateful to all my family. I would also like to mention my classmates — they are people I will always remember. And to one special friend, SB, who was truly a gift from God

Thank you all.

Acknowledgements

First and foremost, I thank Allah, who granted me the ability to write and think, the strength to believe, the patience to pursue my dream until the end, and the joy to raise my hands to the sky and say: *"Alhamdulillah."*

I would like to express my deepest gratitude first and foremost to my teacher, Mrs. **Ghedjemis Fatiha**, She has been an exceptional mentor and guide, not only throughout this thesis but also in various other academic projects. I also express my sincere gratitude and appreciation to the members of the defense committee, Mr. **GUIDAD Daradji** and Mr. **OUAAR Fatima**, for honoring me with their presence, evaluating this work, and providing their valuable academic contributions.

I would also like to extend my deepest thanks and appreciation to all the professors of the Mathematics Department for their efforts in teaching and guiding us. Without their dedication, we would not have reached where we are today.

Contents

List of Figures

Notation

- $f(\mathbf{x})$: Objective function to be optimized
- $\mathbf{x} = (x_1, x_2, \ldots, x_n)$: Vector of decision variables
- F: Feasible region (subset of S "Search space" satisfying constraints)
- $g_j(\mathbf{x}) \leq 0$: Inequality constraints
- $h_i(\mathbf{x}) = 0$: Equality constraints
- x^*: Global optimal solution
- x_L: Local optimal solution
- $\nabla f(\mathbf{x})$: Gradient of the objective function
- $\nabla^2 f(\mathbf{x})$: Hessian matrix of the objective function
- S: The original image.
- AS: An adversarial example ($AS = S + A$).
- A: The perturbation.
- TM: The target black-box model.
- $\phi(AS)$: The fitness function of AS.
- $P(AS)$: The attack performance component of the fitness.
- $Z(A)$: The perturbation size evaluation component.
- y_0: The true label of S.
- y_1, y_2: Labels for S with the first and second highest confidence from TM.
- $p(y|AS)$: The confidence of AS being labeled as y by TM.
- P_c, P_m: Crossover and mutation probabilities.

Introduction

Finding optimal solutions is a fundamental attempt across various scientific, engineering, and economic disciplines. *Optimization problems*, which include obtaining the best possible solution from an ensemble of available alternatives under stated constraints, are ubiquitous. While classical optimization techniques have demonstrated effective for well-behaved, usual convex and differentiable problems, many real-world scenarios introduce formidable challenges. These involve high dimensionality, non-convex and multi-modal landscapes, non-differentiable objective functions, complex constraints, and the presence of noise or uncertainty. Such complexities usually render traditional methods insufficient or inapplicable at all.

To answer to these challenges, *metaheuristic algorithms* have appeared as a powerful class of general-purpose optimization strategies. Inspired by natural facts such as physical annealing processes, swarm intelligence, or biological evolution. Metaheuristics offer high-level frameworks for mapping heuristic algorithms able to explore vast and rugged search spaces to obtain near-optimal solutions. They consistently work in a gradient-free manner, what makes them suitable for black-box optimization problems, and usual incorporate stochastic elements to avoid local optima and attain a better balance between exploration and exploitation of the search space.

Within the wide array of metaheuristics, *Genetic Algorithms (GAs)* stand out as a important and widely used approach. GAs have been inspired by the principles of Darwinian natural selection and Mendelian genetics, including a population of candidate solutions over successive generations through processes like selection, crossover, and mutation. They have successful applications in numerous domains, due to their robustness and versatility.

This thesis delves into metaheuristic algorithms, with a particular concentration on Genetic Algorithms. We start by instituting the context of optimization problems, underlying their importance and the inherent difficulties that need advanced solution approaches (Chapter 1). Subsequently, we offer a detailed exposition of the Genetic Algorithm, enveloping its mathematical formulation, core components, evolutionary operators, and theoretical underpinnings (Chapter 2).

The primary donation of this work lies in proving the practical application of GAs to a current challenging problem in the field of machine learning security: the generation of adversarial attacks against Deep Neural Networks (DNNs). Specially, we investigate how GAs can be used to craft subtle perturbations to input data (e.g., images) that make a target DNN

to misclassify it, specially in a black-box setting where internal model details are unknown. We will look into the POBA-GA (Perturbation Optimized Black-box Attack via Genetic Algorithm) framework as a case study to demonstrate the adaptation of GA principles for this specific task (Chapter 3). This application not only highlights the problem-solving abilities of GAs but also illuminates the vulnerabilities of modern AI systems.

Chapter 1

Optimization problem

1.1 What are optimization problems and their importance

One of the majority essential drives of the human chase is more then just to obtained solutions to problems, problems that humanity makes itself, but to obtain the *best* solution. This deep-seated wish for improvement,excellence, and efficiency lies at the heart of optimization. Optimization problems encircle us, showing themselves in practically every field of economics, engineering, science, and even going as far as to life itself. They are the mathematical form of this necessity to "inhance" be it to maximize gains,to improve performance, to cut costs, or to discover the most optimal result given some restrictions[10, 11].

1.1.1 Defining Optimization Problems

At its main idea, an **optimization problem** is involved with choosing the best element from some ensemble of available alternatives, based on a specific conditions. Mathematically, it can be defined by:

Definition 1.1.1 (Optimization Problem)**.** *An optimization problem seeks to obtain the values of* ***decision variables*** *that attained the* ***best*** *possible value of an* ***objective function****, among an ensemble of* ***constraints****[11].*

Let's deconstruct these key components:

- **Decision Variables:** These are the variables that we can manipulate or adjust to effect the outcome. They signify the possibilities we can make. Let's that the set of decision variables is a vector $\mathbf{x} = (x_1, x_2, \ldots, x_n) \in \mathbb{R}^n$. For example, in architecting a bridge, decision variables might be the materials used, the thickness of the steel beams, or the number of support pillars. In asset allocation, they might be the proportions of several assets in the portfolio.

- **Objective Function:** This is a function, habitually denoted as $f(\mathbf{x})$, that quantifies the quality or the performance of a solution $\mathbf{x}$. It's the function we need to maximize or minimize. The objective function interprets our goal into a mathematical expression. Such as:
  - *Cost function:* We could desire to minimize the cost of transportation, manufacturing, or production.
  - *Profit function:* We could wish to maximize the profit from investments, operations, or sales.
  - *Performance metric:* We could wish to maximize the strength of a structure,the accuracy of a prediction model, or the efficiency of an algorithm.
  - *Error function:* In machine learning, we need to minimize an error function that measures the difference between actual and predicted values .

  The objective function offers a way to compare several solutions and know which one is "better" based on our defined criteria.

- **Constraints:** These are conditions or restrictions that should be attained by the decision variables. Constraints is the feasible region of solutions. They can correspond to regulatory requirements, resource availability, physical limitations, or desired performance levels. Constraints can be expressed as equalities or inequalities. Usual kinds of constraints include:
  - *Equality constraints:* For instance, the all budget must be exactly spent, or the all flow into a network node must equal the all flow out. Mathematically, they are often correspond to $h_i(\mathbf{x}) = 0$ for $i = 1, 2, \ldots, m$.
  - *Inequality constraints:* Such as, the weight of a structure should not more than a certain limit, the production capacity should be among a certain range, or the concentration of a chemical should be more then a minimum threshold. Mathematically, they are usually corresponds to $g_j(\mathbf{x}) \leq 0$ for $j = 1, 2, \ldots, p$.

  The ensemble of all solutions $\mathbf{x}$ that fulfill all the constraints is names the **feasible region** or **feasible set**, denoted as F. We are only curious about finding the optimal solution among this feasible region.

- **Goal (Minimization or Maximization):** Optimization problems can be categorized into two main groups based on the goal:
  - *Minimization problems:* The aim is to obtain a solution $\mathbf{x}^* \in F$ such that $f(\mathbf{x}^*) \leq f(\mathbf{x})$ for all $\mathbf{x} \in F$.
  - *Maximization problems:* The aim is to obtain a solution $\mathbf{x}^* \in F$ such that $f(\mathbf{x}^*) \geq f(\mathbf{x})$ for all $\mathbf{x} \in F$.

It's crucial to note that any maximization problem can be transferred into a minimization problem (and vice-versa) by using the negative of the objective function. For example, maximizing $f(\mathbf{x})$ is the same as minimizing $-f(\mathbf{x})$.

Therefore, a general optimization problem might be expressed as:

$$\begin{aligned} &\underset{\mathbf{x}}{\text{minimize (or maximize)}} && f(\mathbf{x}) \\ &\text{subject to} && g_j(\mathbf{x}) \leq 0, \quad j = 1, 2, \ldots, p \\ &&& h_i(\mathbf{x}) = 0, \quad i = 1, 2, \ldots, m \\ &&& \mathbf{x} \in X \subseteq \mathbb{R}^n \quad \text{(or other domain } S\text{)} \end{aligned}$$

where X corresponds to the basic domain of the variables (e.g., $\mathbb{R}^n$, $\mathbb{Z}^n$, $\{0,1\}^n$) and the explicit constraints indicates the feasible region F within X.

1.1.2 The Importance of Optimization Problems

The importance of optimization problems arises from their widespread presence and their capability to give innovation, efficiency, and amelioration across various domains. Here are some key arguments why optimization is so important:

1. **Efficiency and Resource Management:** In a world of growing demands and finite resources, optimization offers the tools to use resources most effectively. By reducing costs, maximizing output, and minimizing waste, optimization assists organizations and individuals attained more with less. This is crucial in domains like logistics, resource allocation, manufacturing, and energy production. For example, in supply chain management, optimization algorithms are utilized to minimize delivery times, inventory levels, and transportation costs, this leads to significant benefits and enhanced efficiency.

2. **Improved Decision-Making:** Optimization offers a quantitative and structured approach to decision-making. Instead of basing only on intuition or trial-and-error, optimization permits to analyze different options, appraised their potential outcomes, and choose the best course of action provides on well-defined objectives and constraints. This is specially possible in complex situations with several interacting factors, like strategic planning, operational management, and financial portfolio management.

3. **Engineering and Design Innovation:** Optimization is important to engineering design. Engineers frequently try hard to design products and systems that are more reliable, efficient, safer, and cost-saver. Optimization techniques are used to design bridges that can withstand maximum charges with minimal material, aircraft wings that minimize drag, circuits that use the least power, and chemical processes that minimize waste while maximizing yield . Optimization lead to create innovative and high-performance solutions that would be difficult or impossible to attained through manual processes alone.

4. **Scientific Discovery and Modeling:** In scientific research, optimization represents a crucial role in parameter estimation ,data analysis , and model development. Researchers utilize optimization techniques to find optimal parameters for simulations, suit numerical models to experimental data, and detect patterns and relationships in complex datasets. For instance, in machine learning, optimization algorithms are used in the training models that can estimate outcomes, make intelligent decisions, and classify data. In physics and chemistry, optimization is utilized to obtain the minimum energy configurations of molecules and materials[1].

5. **Advancements in Technology and Artificial Intelligence:** The rapid developments in technology, specially in artificial intelligence and machine learning, are hard in relation with optimization. Training sophisticated AI models, like deep neural networks, includes solving massive optimization problems to modify millions or even billions of parameters. Optimization algorithms are also important for control systems, autonomous systems, and robotics permitting them to generate optimal decisions in dynamic and uncertain environments.

6. **Solving Real-World Problems:** Optimization is more than a theoretical concept; it is a dominant tool for addressing real-world problems among several sectors. From scheduling airline operations to optimizing traffic flow in cities, from managing water resources to designing personalized medical treatments , optimization offers practical solutions to enhance the quality of life and address societal challenges.

In conclusion, optimization problems go beyond academic exercises; they are fundamental to develop and productivity in countless areas. knowing the basics of optimization and progressing productive methods to solve these problems is very important for scientists, engineers, economists, and decision-makers. As we go deeper into the difficalties of the modern world, the ability to formulate and solve optimization problems will only become more essential for achieving sustainable progress and driving innovation .

1.2 Challenges in optimization

While the idea of optimization obtaining the best solution $\mathbf{x}^*$ with $f(\mathbf{x}^*) \leq f(\mathbf{x})$ (for minimization) for all feasible $\mathbf{x} \in F$ is clear, the actual procedure meets considerable obstacles, specially when the search space S or the feasible region F is not simple. These challenges frequently provide simple or classical optimization approaches deficient. Mathematically, these challenges shown as:

- **High Dimensionality:** The search space S is presented frequently as a subset of $\mathbb{R}^n$ where the dimension n is big (e.g., hundreds, thousands, or even millions in machine learning). The volume of the search space develops exponentially with n, constituting exhaustive search computationally impractical. For example, examining a function on

a coarse grid with just 10 points per dimension demands 10^n evaluations. This number quickly becomes astronomical: $10^3 = 1000$ for $n = 3$, but 10^{10} for $n = 10$, and 10^{100} (a googol) for $n = 100$. Many algorithms afflicted by this "problem of dimensionality," where their execution lowers vastly or the demanded number of function evaluations explodes as n grows (see Figure 1.1).

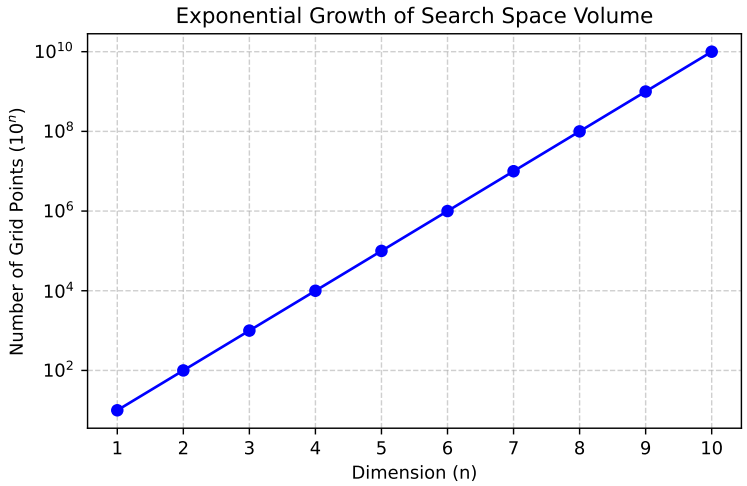


Figure 1.1: Illustration of the curse of dimensionality: The number of grid points need for exploration increases exponentially (k^n) with the dimension n, quickly becoming too complex for practical computation.

- **Non-Convexity and Multi-Modality:** The objective function $f(\mathbf{x})$ can be non-convex. Generally, f is convex if for any $\mathbf{x}_1, \mathbf{x}_2 \in F$ (supposing F is convex) and $\lambda \in [0, 1]$, $f(\lambda\mathbf{x}_1 + (1-\lambda)\mathbf{x}_2) \leq \lambda f(\mathbf{x}_1) + (1-\lambda)f(\mathbf{x}_2)$. If this condition is not satisfied, f is non-convex. Such functions can have many local minima, points that are optimal among a local neighborhood but not necessarily in the all the domain. A point $\mathbf{x}_L \in F$ presents a local minimum if there is a neighborhood $N(\mathbf{x}_L)$ sush that $f(\mathbf{x}_L) \leq f(\mathbf{x})$ for all $\mathbf{x} \in F \cap N(\mathbf{x}_L)$. A global minimum $\mathbf{x}^*$ satisfies $f(\mathbf{x}^*) \leq f(\mathbf{x})$ for all $\mathbf{x} \in F$. Multi-modality means there are multiple distinct local minima $\mathbf{x}_{L1}, \mathbf{x}_{L2}, \ldots$. Simple iterative methods like gradient descent, beginning from different points, can easily get trapped in suboptimal local minima, failing to obtain the true global optimum $\mathbf{x}^*$ (see Figure 1.2)[10, 11].

- **Non-Differentiability:** Several classical optimization methods depend hard on the gradient $\nabla f(\mathbf{x})$ or even the Hessian matrix $\nabla^2 f(\mathbf{x})$ to guide the search way. However, the objective function $f(\mathbf{x})$ could not be differentiable everywhere in the search space S. This intends $\nabla f(\mathbf{x})$ does not exist at certain points, often happening at "kinks" or discontinuities in the function (e.g., functions concluding piecewise definitions, absolute values like $f(x) = |x|$, or max functions). Gradient-based methods unsuccess or demand specialized sub-gradient techniques at points of non-differentiability. Furthermore, in several real-world "black-box" optimization scenarios (e.g., optimizing parameters of a complex simulation), the analytical form of $f(\mathbf{x})$ is unknown, causing symbolic differentiation

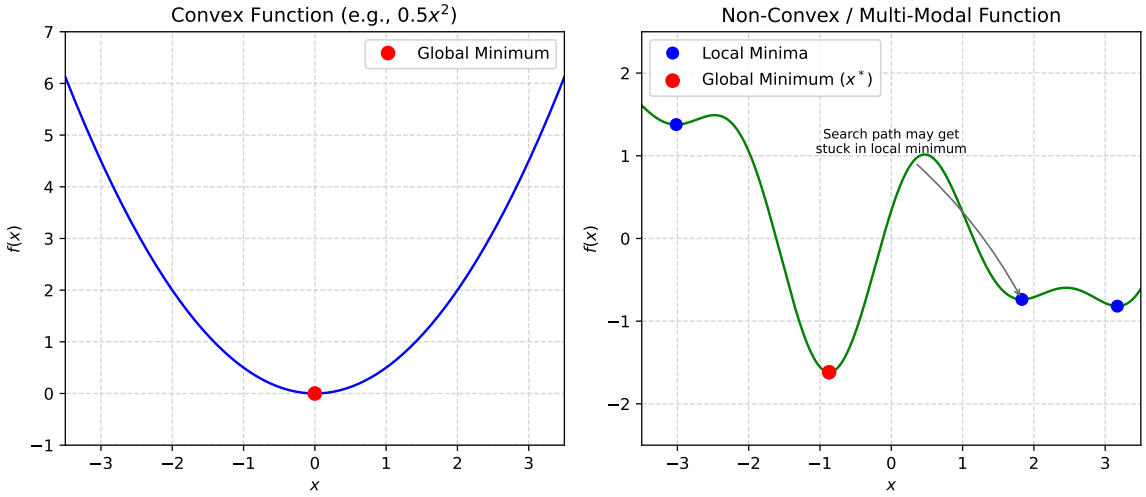


Figure 1.2: Comparison of a convex function (left) with a single global minimum, and a non-convex, multi-modal function (right) with multiple local minima (x_{L1}, x_{L2}) and one global minimum (x^*). Gradient-based methods started within the basin of a local minimum may converge there instead of finding x^*.

unachievable. Numerical approximation of gradients can be numerically expensive and sensitive to noise or step size (see Figure 1.3)[10].

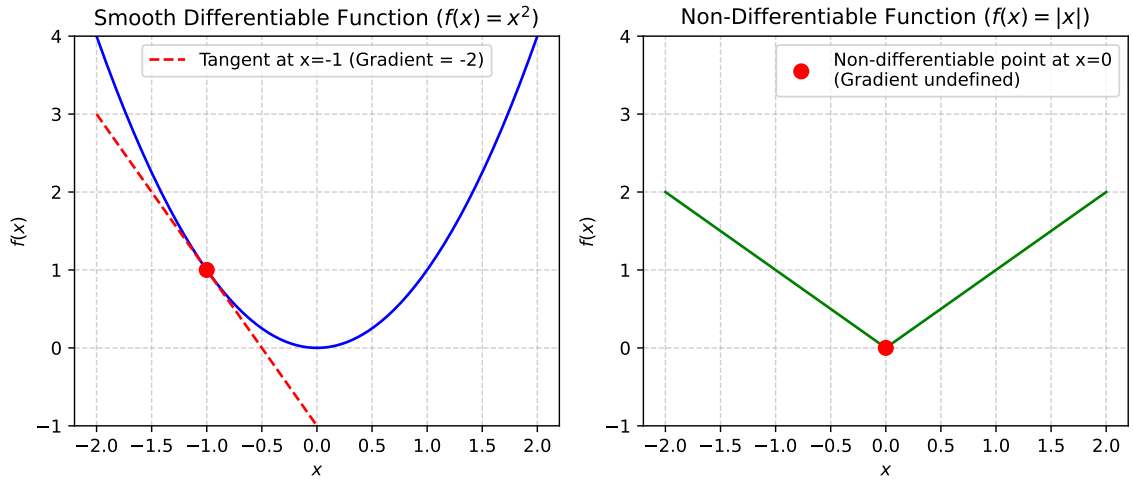


Figure 1.3: Comparison of a smooth, differentiable function (left) where the gradient is defined everywhere, and a non-differentiable function (right, $f(x) = |x|$) with a 'kink' at $x = 0$ where the gradient is undefined. Gradient-based methods cannot be directly applied at such points.

- **Computational Cost:** Valuing the objective function $f(\mathbf{x})$ and/or the constraints $g_j(\mathbf{x}), h_i(\mathbf{x})$ for one candidate solution $\mathbf{x}$ may be numerically expensive. This evaluation time, T_{eval}, could vary from microseconds (simple formulas) to seconds, minutes, or even hours (e.g., complex physics simulations such as Finite Element Analysis (FEA), training a deep neural network, or Computational Fluid Dynamics (CFD)). If an optimization algorithm demands N_{eval} evaluations to converge, the whole runtime $T_{total} \approx N_{eval} \times T_{eval}$ may easily be prohibitive, restricting the feasibility of methods that demand a huge number of evaluations.

- **Complex Constraints:** The feasible region $F = \{\mathbf{x} \in X \subseteq \mathbb{R}^n \mid g_j(\mathbf{x}) \leq 0, j = 1..p; h_i(\mathbf{x}) = 0, i = 1..m\}$ can have a complex geometry. Constraints $g_j(\mathbf{x})$ or $h_i(\mathbf{x})$ might be highly non-linear, making the boundary of the feasible region irregular. The feasible set F itself might be non-convex (even if $f(\mathbf{x})$ is convex), or even disjoint (composed of multiple separate regions). Navigating such complex feasible regions is challenging for many algorithms. Staying within the feasible region (constraint handling) becomes difficult, and moving between disconnected feasible regions might be impossible for simple local search methods (see Figure 1.4).

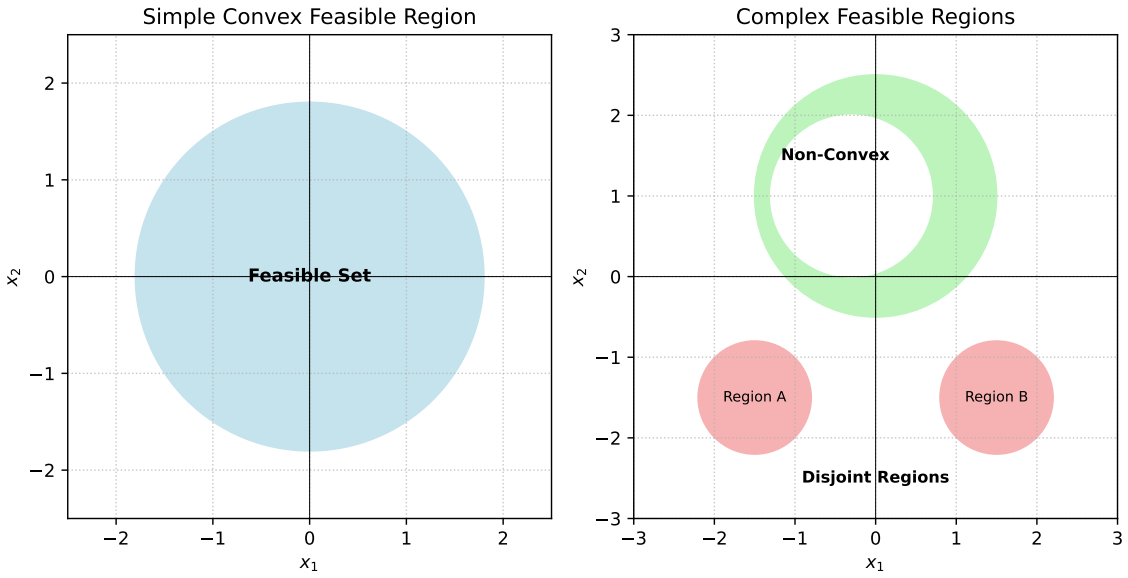


Figure 1.4: Comparison of a simple convex feasible region (left) and complex feasible regions (right): non-convex (top right) and disjoint (bottom right). Finding the optimum within or navigating such complex regions poses significant challenges.

- **Noise and Uncertainty:** In several realistic applications, function evaluations are dependent on noise or stochasticity. Instead of remarking the true value $f(\mathbf{x})$, we could remark $\hat{f}(\mathbf{x}) = f(\mathbf{x}) + \epsilon$, where ϵ is a random variable illustrating environmental fluctuations, measurement error, or simulation stochasticity. Comparing two solutions $\mathbf{x}_1$ and $\mathbf{x}_2$ according to noisy evaluations $\hat{f}(\mathbf{x}_1)$ and $\hat{f}(\mathbf{x}_2)$ turn into statistically challenging, particularly if the noise magnitude σ_ϵ is significant depend on the true difference $|f(\mathbf{x}_1) - f(\mathbf{x}_2)|$. Optimization might also require to be done under uncertainty, such as f based on explicitly on random parameters ω, i.e., $f(\mathbf{x}, \omega)$, demanding robust optimization (average performance or optimizing worst-case) or stochastic programming techniques (see Figure 1.5).

These mathematical properties and practical challenges underscore the require for optimization techniques that may effectively deal with such difficulties.

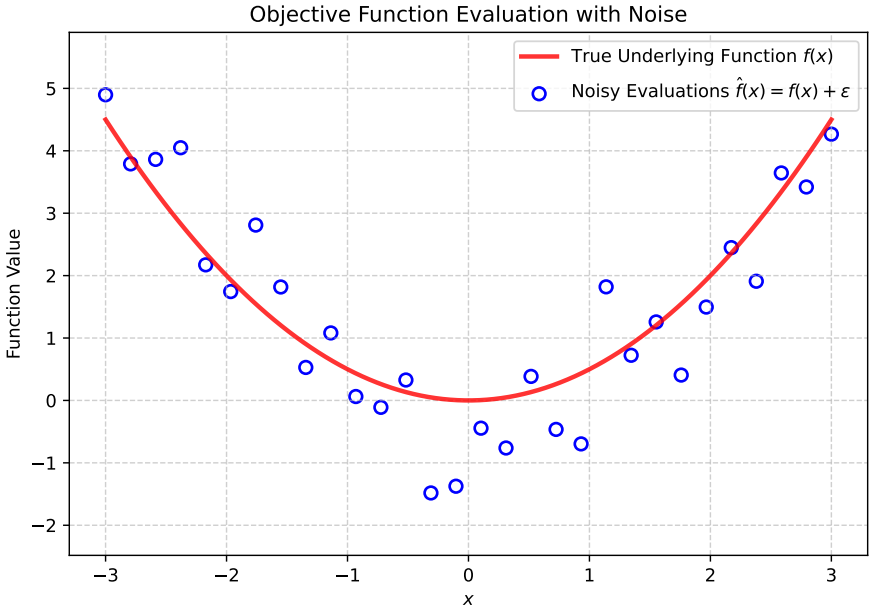


Figure 1.5: Illustration of a noisy objective function. The solid line represents the true underlying function $f(x)$, while the points represent noisy evaluations $\hat{f}(x) = f(x) + \epsilon$. The noise obscures the exact location and value of the minimum, complicating the optimization process.

1.2.1 Why metaheuristic algorithms are crucial and how the traditional optimization methods are restricted

Traditional optimization methods, often derived from calculus (like gradient descent, Newton's method) and mathematical programming (like linear or convex programming), exhibit significant limitations, particularly when facing the challenges outlined in Section 1.2. Metaheuristics gain importance precisely because they offer robust strategies to circumvent these limitations.

Limitations of Traditional Methods (Mathematically Framed):

- **Gradient Dependency:** Methods such that Gradient Descent ($\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \nabla f(\mathbf{x}_k)$) or Newton's Method ($\mathbf{x}_{k+1} = \mathbf{x}_k - [\nabla^2 f(\mathbf{x}_k)]^{-1} \nabla f(\mathbf{x}_k)$) explicitly need the calculation of the gradient $\nabla f(\mathbf{x})$ and might the Hessian $\nabla^2 f(\mathbf{x})$. They fundamentally don't succeed if these derivatives do not exist (non-differentiable f, see Figure 1.3) or are unattainable (black-box f).

- **Local Optima Entrapment:** Iterative approaches usually generate sequences $\{\mathbf{x}_k\}$ that converge to a stationary point $\mathbf{x}_{stat}$ where $\nabla f(\mathbf{x}_{stat}) = 0$ (or a point satiating similar first-order necessary conditions). If $f(\mathbf{x})$ is non-convex (multi-modal, see Figure 1.2), this $\mathbf{x}_{stat}$ is usually just a local minimum $\mathbf{x}_L$ or even a saddle point, where it's not certain being the global minimum $\mathbf{x}^*$. The algorithm becomes "trapped" in the basin of attraction of the first stationary point it nears, powerfully according to the starting point $\mathbf{x}_0$.

- **Assumptions on Problem Structure:** Many strong traditional methods need powerful assumptions. Linear Programming supposes $f(\mathbf{x}) = \mathbf{c}^T\mathbf{x}$ and linear conditions $A\mathbf{x} \leq \mathbf{b}, A_{eq}\mathbf{x} = \mathbf{b}_{eq}$. Convex programming needs $f(\mathbf{x})$ to represent a convex function and the

feasible ensemble F to represent a convex set (like the left panel in Figure 1.4). Several real-world problems do not fulfill these limited structural assumptions.

- **Difficulty with Non-Continuous Spaces:** Classical methods are first created for continuous search spaces $S \subseteq \mathbb{R}^n$. Conforming them effectively to discrete domains binary domains ($S \subseteq \{0,1\}^n$), combinatorial spaces (e.g., spaces of permutations or graph structures), or ($S \subseteq \mathbb{Z}^n$) or is usually non-trivial or not realizable, as the concept of a continuous gradient way does not implement.

- **Scalability Issues:** Some approaches have numerical difficulty per iteration that scales badly with the dimension n. For example, shaping and inverting the Hessian matrix in Newton's method holds approximately $O(n^3)$ operations, which befits prohibitive for the high-dimensional problems talk about earlier (Figure 1.1).

Importance of Metaheuristic Algorithms (Mathematical Perspective): Metaheuristics address these limitations directly:

- **Gradient-Free Nature:** They commonly operate depend only on evaluating the objective function value $f(\mathbf{x})$ for candidate solutions $\mathbf{x}$. They do not need calculating $\nabla f(\mathbf{x})$ or $\nabla^2 f(\mathbf{x})$, driving them directly applicable to non-differentiable (Figure 1.3) and black-box problems in which the gradient information is unreliable or unavailable.

- **Global Search Capability:** They include stochastic components and exploration-exploitation mechanisms particularly designed to control local optima trick (Figure 1.2). For example, algorithms such as Simulated Annealing may admit worsening moves ($f(\mathbf{x}_{new}) > f(\mathbf{x}_{current})$) with a sure probability, permitting them to escape local basins. Population-based methods such as Particle Swarm Optimization or Genetic Algorithms maintain a diverse ensemble of solutions discovering many regions of the search space at the same time.

- **Flexibility and Broad Applicability:** They get far fewer thoughts about the mathematical structure of the objective function $f(\mathbf{x})$ or the feasible ensemble F. The same primary metaheuristic framework (e.g., Genetic Algorithm) may generally be implemented (with appropriate adjustments in variation operators and solution representation) to discrete ($\mathbb{Z}^n, \{0,1\}^n$), continuous ($\mathbb{R}^n$), or complex combinatorial problems, and may pick up non-convex or disjoint feasible regions (Figure 1.4).

- **Robustness to Noise:** Since they generally depend on rankings or comparisons of solutions rather than detailed function values or gradient approximates, some metaheuristics may reveal greater robustness to noise in function evaluations (Figure 1.5) compared to methods that rely on accurate computational differentiation.

Briefly, metaheuristics offers a pragmatic and strong ensemble of tools when the intrinsic mathematical properties of the optimization problem (non-differentiability, non-convexity, complex

constraints, noise, black-box nature, high dimensionality) allow the efficient or effective implementation of traditional, generally gradient-based or structure-dependent, optimization techniques. They deal the mathematical guarantee of obtaining the *exact* global optimum (which traditional approaches generally only offer under strict conditions such as convexity) for the real ability to obtain high-quality, near-optimal solutions for a wide range of complex, real-world problems in acceptable calculated time.

1.3 Metaheuristic algorithms

1.3.1 Introduction to metaheuristic algorithms and their significance

Constructing upon the restrictions of traditional methods emphasized in Section 1.2.1, we institute **metaheuristic algorithms**. The term "meta" means "beyond" or "higher level," and "heuristic" connects to the process of obtaining solutions, generally by trial-and-error or guided search, without promises of optimality.

Definition 1.3.1 (Metaheuristic Algorithm)**.** *A metaheuristic is a high-level problem-individualistic algorithmic structure that offers an ensemble of strategies or guidelines to inhence heuristic optimization algorithms. Metaheuristics are often designed to obtain near-optimal solutions to difficult optimization problems where classic methods are deficient, particularly for problems with non-convexity, large search spaces, combinatorial structures, or non-differentiability [2].*

Metaheuristics do not assure obtaining the global optimal solution for all problem examples. However, their goal is to obtain excellent solutions along a reasonable amount of calculating time, being them highly valuable for feasible applications. Their importance trunk directly from their capability to address the challenges sketched in Section 1.2:

- **Handling Complexity:** They are built to explore complex search landscapes, involving those with several local optima (multi-modality), high dimensionality, and discontinuities.
- **Gradient-Free Operation:** Most metaheuristics depend sonly on objective function evaluations, making them adaptable for non-differentiable functions, black-box optimization, and problems where derivatives are impossible or expensive to compute, .
- **Versatility:** The general structure of several metaheuristics may be adapted to many types of optimization problems, involving continuous, discrete, combinatorial, and mixed-variable problems, generally by changing the solution presentation and search operators.
- **Global Search Focus:** As opposed to simple local search approaches, metaheuristics involve mechanisms to balance exploitation (intensification) around promising regions with exploration (diversification) of the search space, growing the likelihood of obtaining globally ruthless solutions.

The main idea beyond several metaheuristics includes mimicking successful strategies remarked in nature or other processes. Usual inspirations involve:

- **Biological Evolution:** Algorithms like Genetic Algorithms (GA), Genetic Programming (GP), and Differential Evolution (DE) simulate processes like natural selection, crossover, and mutation.
- **Swarm Intelligence:** Algorithms like Particle Swarm Optimization (PSO) and Ant Colony Optimization (ACO) mimic the collective behavior of social insects or animal groups (e.g., bird flocking, ant foraging).
- **Physical Processes:** Simulated Annealing (SA) models the process of annealing in metallurgy, where controlled cooling allows a material to reach a low-energy state.
- **Human Problem Solving:** Tabu Search (TS) includes memory structures to avoid revisiting previously explored solutions and to guide the search away from local optima.

A key predictable combining these diverse approaches is the strategic employ of **stochasticity** (randomness) and **heuristic rules** to conduct the search process. Stochastic elements assist in exploring new regions of the search space and avoiding local optima, while heuristic rules use knowledge reached during the search to focus on hopeful areas. The interchange between **exploration** (searching broadly to recognize strongly good regions) and **exploitation** (searching intensively among promising regions to obtain the best solutions there) is essential to the performance of metaheuristics.

Remark 1.3.1 (Exploration vs. Exploitation)**.** *The balance between exploration and exploitation is crucial. Too much exploration yields to a inefficient, slow search that can never converge properly. A lots of exploitation yields to an early convergence, where the algorithm rapidly gets trapped in the first local optimum it obtains, failing to find better solutions away in the search space. Efficient metaheuristics dynamically control this trade-off all over the optimization operation[4, 2].*

In conclusion, metaheuristic algorithms are strong and adaptable class of optimization techniques. Their capability to address complex, real-world problems in which traditional methods strive makes essential tools in domains vary from operations research and engineering design to bioinformatics and machine learning. The following chapters will explore one specific and widely utilized metaheuristic: the Genetic Algorithm.

Chapter 2

The Genetic Algorithm

The Genetic Algorithm (GA) components a prominent class of stochastic optimization algorithms inclusion to the broader field of Evolutionary Computation [4]. It has been inspired by principles of Darwinian natural selection and Mendelian genetics , GAs offer a framework for searching complex, usual high-dimensional spaces to obtain approximate solutions to problems of optimization [8, 7]. They work iteratively on a population of potential solutions, using mechanisms analogous to biological evolution to progressively improve the population approaching regions of higher solution quality.

2.1 Mathematical Formulation

Assuming an optimization problem focused on finding a solution x^* within a search space $\mathcal{S}$ that optimizes (e.g., minimizes) an objective function $g : \mathcal{S} \to \mathbb{R}$. A GA approaches this by including a population of potential solutions.

2.1.1 Representation (Encoding)

Each possible solution $x \in \mathcal{S}$ is encoded as a chromosome (or individual). This encoding guides the solution representation (phenotype) to a genetic representation (genotype), usually a vector or string structure. Assuming an individual be represented by $x = (g_1, g_2, \ldots, g_L)$, where g_i presents the i-th gene. The nature of the genes relies on the problem domain and encoding scheme:

- **Binary Encoding:** $g_i \in \{0, 1\}$. The chromosome is a binary string $x \in \{0, 1\}^L$.
- **Real-Valued Encoding:** $g_i \in [\text{lower}_i, \text{upper}_i] \subset \mathbb{R}$. The chromosome is a vector $x \in \mathbb{R}^L$.
- **Permutation Encoding:** x is a permutation of $\{1, 2, \ldots, L\}$, commonly used for ordering problems.

The choice of representation significantly reflects on the effectiveness and design of the genetic operators [7].

2.1.2 Population

The algorithm conserves a population $P(t)$ of N individuals at generation t:

$$P(t) = \{x_1(t), x_2(t), \ldots, x_N(t)\}$$

where $x_j(t) \in \mathcal{S}$ are the j-th individual at generation t. The initial population $P(0)$ is commonly generated randomly, sampling uniformly from the search space $\mathcal{S}$ where feasible, or using problem-specific heuristics.

2.1.3 Fitness Function

The quality of each individual $x_j(t)$ is evaluated using a fitness function $f : \mathcal{S} \to \mathbb{R}^+$. The fitness function is obtained from the objective function $g(x)$. For a minimization problem $g(x)$, a typical transformation is $f(x) = 1/(1 + g(x))$ if $g(x) \geq 0$, or $f(x) = C_{\max} - g(x)$ for some large constant $C_{\max}$, guarantee higher fitness corresponds to better solutions (lower objective values). Maximization problems could use $f(x) = g(x)$ directly if $g(x) > 0$. The fitness $f(x_j(t))$ quantifies the reproductive possible of individual $x_j(t)$.

2.2 Evolutionary Operators

The transition from population $P(t)$ to $P(t + 1)$ is extracted by selection, crossover, and mutation.

2.2.1 Selection

Selection is identifying individuals from $P(t)$ to serve as parents for the next generation. The probability of selecting an individual $x_j(t)$, is $p_s(x_j(t))$, it is consistently proportional to its relative fitness among the population. A typical method is Fitness Proportional Selection (or Roulette Wheel Selection), where:

$$p_s(x_j(t)) = \frac{f(x_j(t))}{\sum_{k=1}^{N} f(x_k(t))}$$

Other mechanisms like Tournament Selection or Rank Selection are also frequently used to control selection pressure and mitigate issues such as premature convergence [7]. Selection constructs an intermediate population or mating pool $M(t)$.

2.2.2 Crossover (Recombination)

Crossover works on pairs of parent individuals chosen from the mating pool $M(t)$ to generate offspring. It is used with a specific crossover probability p_c. Suppose that $x_a, x_b \in M(t)$ are two selected parents. The crossover operator $c : \mathcal{S} \times \mathcal{S} \to \mathcal{S} \times \mathcal{S}$ (or $c : \mathcal{S} \times \mathcal{S} \to \mathcal{S}$) creates one

or more offspring x'_a, x'_b. For example, in one-point crossover for binary strings $x_a = (a_1 \dots a_L)$ and $x_b = (b_1 \dots b_L)$, a crossover point $k \in \{1, \dots, L-1\}$ is selected randomly. The offspring are:

$$x'_a = (a_1 \dots a_k, b_{k+1} \dots b_L)$$

$$x'_b = (b_1 \dots b_k, a_{k+1} \dots a_L)$$

Several crossover operators exist, adapted to different representations (e.g., blend crossover (BLX-α) for real values, partially mapped crossover (PMX) for permutations) [4]. Crossover makes easy the exploration of combinations of parental features (schemata [8]).

2.2.3 Mutation

Mutation represents random alterations to individual genes among an offspring chromosome, achieving with a common low mutation probability p_m per gene or per chromosome. The mutation operator $m : \mathcal{S} \to \mathcal{S}$ changes an individual x' to create x''. For binary strings, bit-flip mutation modifies a selected g_i from 0 to 1 or vice-versa. For real-valued encoding, mutation might include adding a random value from a particular distribution (e.g., Gaussian) to a gene g_i. Mutation aims at maintain genetic diversity and frustrates irreversible loss of genetic material, allowing escape from local optima [7].

2.3 Algorithmic Structure

The canonical Genetic Algorithm advances iteratively:

1. **Initialization ($t = 0$):** Create initial population $P(0) = \{x_1(0), \dots, x_N(0)\}$. Calculate fitness $f(x_j(0))$ for all $j = 1, \dots, N$.
2. **Iteration Loop ($t = 0, 1, 2, \dots, T_{\max}$):**
   (a) **Selection:** Generate a mating pool $M(t)$ by choosing individuals from $P(t)$ based on fitness $f(x_j(t))$ employing a chosen selection scheme (e.g., N selections with replacement).
   (b) **Crossover:** Pair individuals from $M(t)$ and apply the crossover operator c with probability p_c to create an offspring population $O_c(t)$. Individuals not undertaking crossover are generally copied directly.
   (c) **Mutation:** Use the mutation operator m to individuals in $O_c(t)$ with probability p_m to create the final offspring population $O_m(t)$.
   (d) **Evaluation:** Calculate fitness $f(x')$ for all offspring $x' \in O_m(t)$.
   (e) **Replacement:** Build the next generation population $P(t+1)$ by choosing individuals from $P(t)$ and $O_m(t)$. Typical strategies involve generational replacement (replace entire $P(t)$ with $O_m(t)$ if $|O_m(t)| = N$) or including elitism (keeping the best individual(s) from $P(t)$ into $P(t+1)$).

(f) Increment generation counter: $t \leftarrow t + 1$.

3. **Termination:** Stop when a stopping criterion is satisfied (e.g., maximum number of generations $T_{\max}$, convergence of fitness values, time limit).

4. **Output:** Give back the individual with the highest fitness come across during the whole process as the best-found solution x_{best}.

2.4 Theoretical Aspects

The theoretical underpinnings of GAs were explored by Holland [8] via the concept of schemata (hyperplanes representing subsets of chromosomes with resemblances at certain positions). The Schema Theorem offers insight into how GAs implicitly favor the proliferation of short, low-order, high-fitness schemata over generations, although its direct predictive power for complex GA dynamics is debated [7]. GAs are recognized as robust stochastic search methods able of navigating complex fitness landscapes efficiently [4].

2.5 Illustrative Applications

Genetic Algorithms have been successfully applied to a wide array of optimization problems across different domains. Their versatility stems from their problem-agnostic nature at a high level, demanding only a suitable representation, fitness function, and genetic operators. Below, we explore two common application scenarios.

2.5.1 Optimization of Test Functions

A common practice in judging the performance of optimization algorithms, involving GAs, is to apply them to a suite of standard benchmark or test functions. These functions have been familiar by characteristics, such as the location and value of their global optimum, the presence and nature of local optima, dimensionality, and separability.

- The **Sphere function** is a simple, unimodal, convex function, often used as an initial test case. For $x = (x_1, \ldots, x_D)$, it is typically defined as $g(x) = \sum_{i=1}^{D} x_i^2$. Its global minimum is at $x_i = 0$ for all i, with $g(x^*) = 0$.

- The **Rastrigin function** is a more complex, multimodal function characterized by a large number of local optima. For $x = (x_1, \ldots, x_D)$, it is defined as $g(x) = 10D + \sum_{i=1}^{D}[x_i^2 - 10\cos(2\pi x_i)]$. Its global minimum is also at $x_i = 0$ for all i, with $g(x^*) = 0$. This function tests an algorithm's ability to escape local optima and find the global optimum in a rugged landscape.

For these types of continuous optimization problems, real-valued encoding is typically used. The performance of the GA is generally visualized by plotting the landscape of the objective function (e.g., utilizing contour plots for 2D problems) and overlaying the positions of individuals in the population. Figures 2.1 and 2.2 illustrate this for the Sphere and Rastrigin functions, respectively, revealing the state of the population at the beginning of the search and after 100 generations.

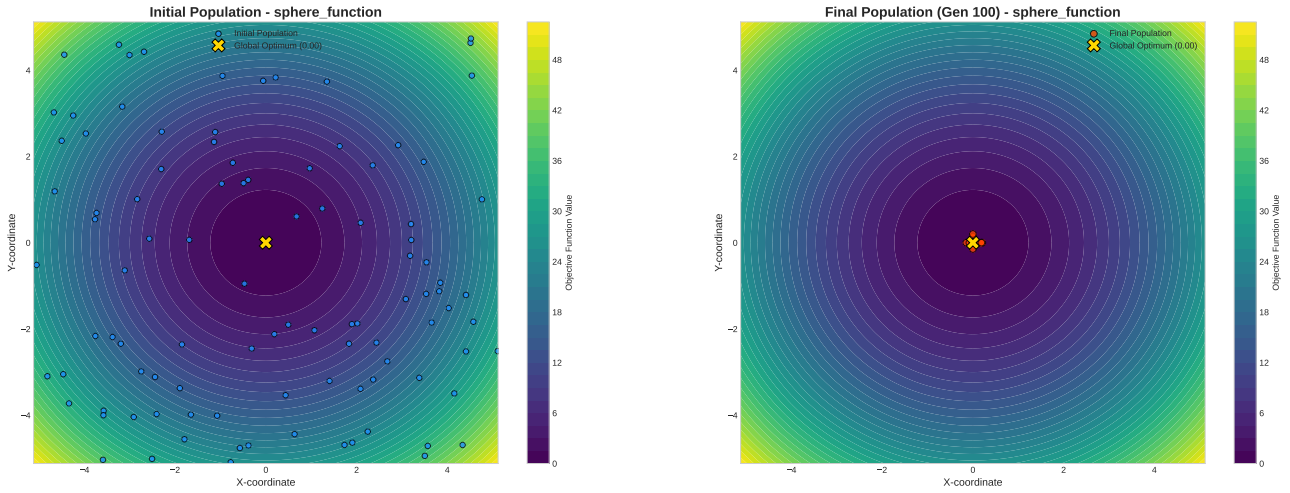


Figure 2.1: Visualization of GA performance on the 2D Sphere function between the initial population and the final population at Gen 100. The global optimum is at (0,0) with a value of 0.00.

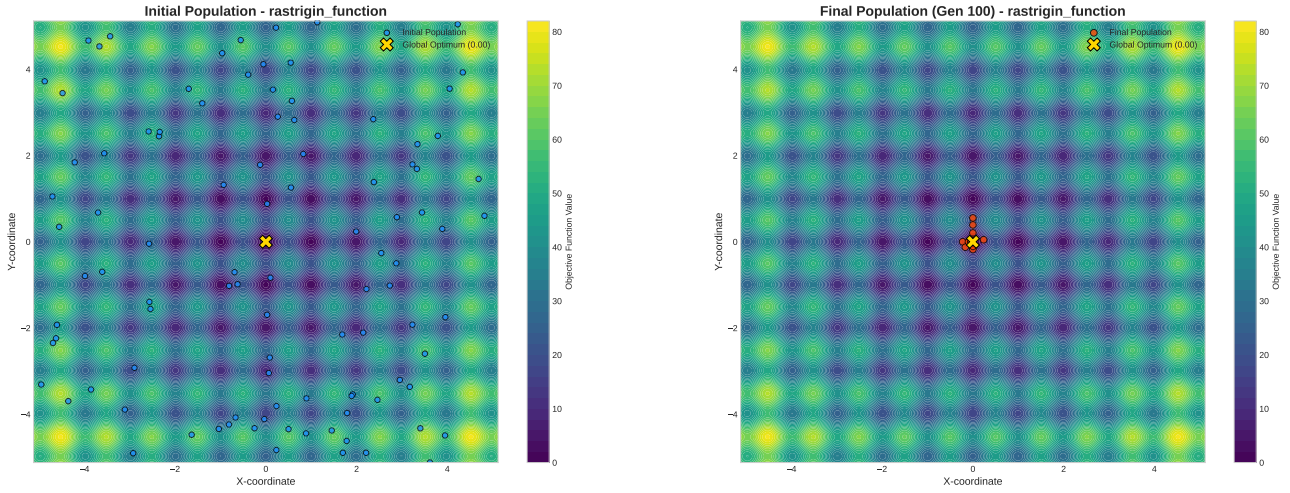


Figure 2.2: Visualization of GA performance on the 2D Rastrigin function between the initial population and the final population at Gen 100. The global optimum is at (0,0) with a value of 0.00.

The plots prove key aspects of the GA's search process:

- The **Initial Population plots** (the two plots on the left) exhibit individuals (typically depicted as blue circles with black outlines in the offered image style) scattered widely within the search space. This affects the random initialization of the population. The global optimum is commonly marked (e.g., a yellow 'X' marker).

- The **Final Population plots** (the two plots on the right) exhibit the state of the population after a certain number of generations (here, 100). The individuals (often orange circles with black outlines) are expected to have converged towards the area of the global optimum. For the unimodal Sphere function (Figure 2.1), this convergence is commonly direct and pronounced. For the multimodal Rastrigin function (Figure 2.2), the plot illustrate the GA's ability to navigate a complex landscape with many local optima to find the global optimum. The tight clustering of the final population around the global optimum (0.00) signals successful optimization.

Such visualizations are invalid for understanding the search dynamics of the GA, involving its exploration and exploitation characteristics, convergence speed, and overall efficiency on different fitness landscapes.

2.5.2 Combinatorial Optimization: The Traveling Salesperson Problem (TSP)

The Traveling Salesperson Problem (TSP) is a classic NP-hard problem in combinatorial optimization. The target is to find the shortest possible route that visits each city in a given list of cities exactly once and then returns to the city of origin. Because of its complexity, exact solutions are often infeasible for large numbers of cities, making heuristic methods like GAs attractive.

Applying a GA to the TSP includes specific considerations:

- **Representation:** Permutation encoding is the natural choice. A chromosome is a permutation of the cities, e.g., $(c_1, c_2, \ldots, c_N)$, representing the order in which the cities are visited.

- **Fitness Function:** The objective is to minimize the total tour length. The fitness function is typically inversely proportional to this length, e.g., $f(\text{tour}) = 1/\text{length}(\text{tour})$.

GAs have demonstrated considerable success in finding high-quality solutions to TSP instances, generally outperforming simpler heuristics and offering good approximations to the optimal solution,specially for moderately sized problems. They excel in investigating the vast search space of possible tours, balancing the exploitation of good existing sub-tours with the exploration of new tour structures.

Chapter 3

Applying Genetic Algorithms to Adversarial Attacks on Neural Networks

The vulnerability of deep neural networks (DNNs) to adversarial attacks has become a significant concern in machine learning. Adversarial attacks include crafting subtle, often imperceptible, perturbations to input data that cause a DNN to misclassify it with high confidence [3]. This chapter investigates the application of Genetic Algorithms (GAs) to generate such adversarial examples, especially in a black-box setting where the attacker has no knowledge of the target model's architecture or parameters, only access to its input-output behavior (e.g., predicted labels and confidence scores).

Black-box attacks may be seted as optimization problems: the aim is to find a minimal perturbation that, when joint to an original input, successfully fool the target model. Genetic Algorithms, as powerful stochastic optimization techniques, are well-suited for this operation due to their capability to navigate complex search spaces without demanding gradient information. This chapter illustates the POBA-GA (Perturbation Optimized Black-box Attack via Genetic Algorithm) method proposed by Chen et al. [3] as a concrete example of applying GAs in this domain.

3.1 Problem Definition for Adversarial Attacks

The main problem is to generate an adversarial example from an original input that deceives a target DNN.

Definition 3.1.1 (DNN-based Image Label (from [3]))**.** *Set a Deep Neural Network (DNN) trained for image classification, for an input image S, the DNN outputs a label y_1, presented as $TM(\Theta, S) = y_1$, where Θ presents the model parameters, and y_1 is the output label in the company of the highest confidence.*

Definition 3.1.2 (Adversarial Attack (from [3]))**.** *Given a DNN and an original image S, for which the model outputs the true label y_0 (i.e., $TM(\Theta, S) = y_0$), an adversarial attack method AM generates an adversarial image AS such that $TM(\Theta, AS) = y_1$, where $y_1 \neq y_0$. The images*

S and AS should be almost indistinguishable to the human eye, meaning the added perturbation $A = AS - S$ is small.

The goal of a black-box adversarial attack is to find such a perturbation A (and thus AS) that minimizes its detectability while guarantee misclassification. This fundamentally forms an optimization problem where we attempt to:

- Maximize the probability of misclassification (or the confidence of the wrong class).
- Minimize the size of the perturbation A.

POBA-GA [3] utilizes a genetic algorithm to improve perturbations that achieve these dual objectives.

3.2 The POBA-GA Framework

POBA-GA translates the black-box adversarial attack into finding an optimal perturbation through an evolutionary process. The framework, as showed in Figure 3.1 (inspired by Figure 2 in [3]), consists of various key stages:

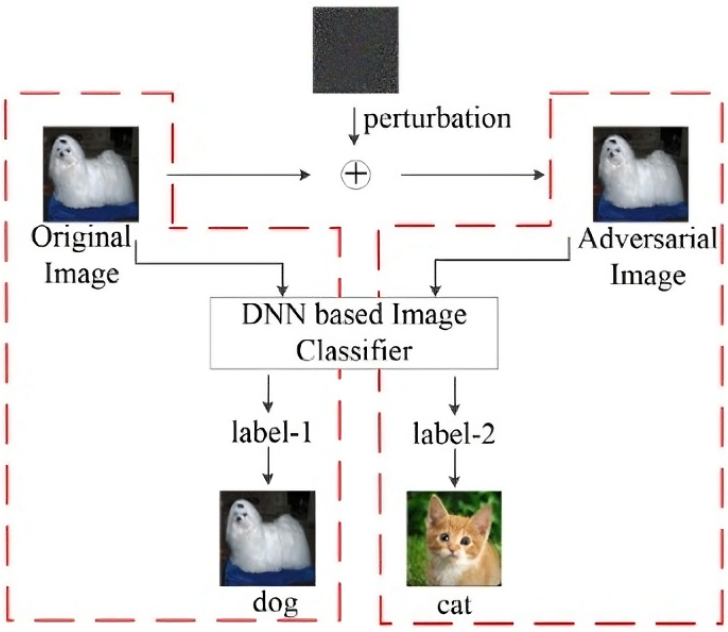


Figure 3.1: Conceptual block diagram of the POBA-GA framework. It includes initializing a population of perturbations, calculating their fitness based on attack success and perturbation size, and iteratively refining them through selection, crossover, and mutation until a satisfactory adversarial example is met. (Adapted from Chen et al. [3], Figure 2).

1. **Initialization:** A population of various initial perturbations is generated. These perturbations, when included to the original image S, form the initial population of adversarial example candidates AS_i^0.

2. **Fitness Evaluation:** Each candidate adversarial example AS_i in the current population is fed to the target model TM. The output (predicted label and confidences) is used to calculate a fitness score $\phi(AS_i)$. This fitness score quantifies both the attack ability (how well it deceives the model) and the perturbation control (the small-scale of the perturbation).

3. **Termination Check:** The algorithm checks if a stop condition is met. This might be a maximum number of generations, obtaining an adversarial example that meets predefined criteria (e.g., target misclassification with sufficient confidence and low perturbation), or a budget on the number of queries to the target model. If met, the best adversarial example obtained is output.

4. **Evolutionary Operations:** If stop conditions are not met, the algorithm proceeds to produce a new generation of perturbations:

   - **Selection:** Parent perturbations are chosen from the current population based on their fitness scores.
   - **Crossover:** Selected parent perturbations are combined to create offspring perturbations, inheriting characteristics from both parents.
   - **Mutation:** Offspring perturbations undergo small random changes to present new genetic material and maintain diversity.

   The newly generated perturbations is the next population, and the cycle repeats from the Fitness Evaluation stage.

Key symbols used in POBA-GA (from Table 2 in [3]) involve:

3.3 Genetic Algorithm Components in POBA-GA

POBA-GA tailors the standard GA components for the specific task of generating adversarial perturbations.

3.3.1 Initialization

The initial population of perturbations $A^{t=0}$ is important for the diversity of the search. As described by Chen et al. [3], for a stated original image S, initial perturbations δ are created using random Gaussian noise:

$$A_i^{t=0} = \delta_i \quad \text{where } \delta_i \sim \mathcal{N}(\mu, \sigma_i^2)$$

The initial adversarial corresponding examples are $AS_i^{t=0} = S + A_i^{t=0}$. To improve diversity, different initial perturbations are created based on varying parameters like the variance σ_i^2, the number of noise points, and the noise point size.

3.3.2 Fitness Function

The fitness function $\phi(AS)$ is mapped to balance attack capability and perturbation size. An excellent adversarial example have strongly deceive the model while being minimally modified from the original. POBA-GA utilizes a two-part fitness strategy [3].

Suppose y_0 be the true label of S. When AS is fed to TM, let y_1 be the label with the highest confidence and y_2 be the label with the second-highest confidence. The confidence of AS being classified as label y is $p(y|AS)$.

The attack performance $P(AS)$ is defined as:

$$P(AS) = \begin{cases} p(y_1|AS) - p(y_0|AS) & \text{if } y_1 \neq y_0 \text{ (successful attack)} \\ p(y_2|AS) - p(y_0|AS) & \text{if } y_1 = y_0 \text{ (failed attack)} \end{cases} \tag{3.1}$$

If the attack is successful ($y_1 \neq y_0$), $P(AS)$ measures the confidence margin of the misclassification over the true label. If the attack fails ($y_1 = y_0$), $P(AS)$ measures the margin of the second-best class over the true label; a smaller (more negative) value is better here as it pushes the model further from the true class.

The perturbation size is denoted by $Z(A)$. Initially, the algorithm concentrates solely on achieving a successful attack. Thus, the parameter α (which weights the perturbation penalty) is effectively 0. Once an attack succeeds at iteration t_{succ}, the fitness function incorporates the perturbation penalty. The updated fitness function is (from Eq. (3) in [3]):

$$\phi(AS) = \begin{cases} (p(y_1|AS) - p(y_0|AS)) - \alpha \dfrac{Z(A)}{Z_{\max}(A^{t_{succ}})} & \text{if } y_1 \neq y_0 \\ p(y_2|AS) - p(y_0|AS) & \text{if } y_1 = y_0 \end{cases} \tag{3.2}$$

Here, $Z_{\max}(A^{t_{succ}})$ is the maximum perturbation size within the successfully attacked examples at iteration t_{succ} (when the first successful attack occurred for any individual in the population, or a similar normalizing factor for perturbation). This term normalizes the perturbation penalty. The parameter α controls the trade-off: a larger α prioritizes smaller perturbations.

3.3.3 Evolutionary Operations

Selection

POBA-GA employs Roulette Wheel Selection [3]. For each individual AS^i in the population with fitness $\phi(AS^i)$, its selection probability $f_p(AS^i)$ is:

$$f_p(AS^i) = \frac{\phi(AS^i)}{\sum_{j=1}^{N} \phi(AS^j)} \tag{3.3}$$

where N is the population size. Individuals with higher fitness scores have a greater chance of being selected as parents for the next generation. The cumulative probability $fr(AS^i)$ is:

$$fr(AS^i) = \sum_{j=1}^{i} f_p(AS^j) \tag{3.4}$$

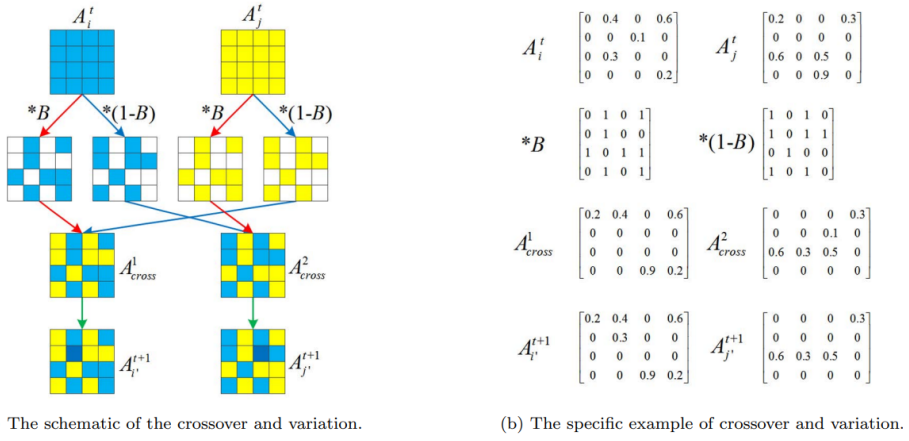
Crossover

POBA-GA uses a uniform crossover tailored for perturbations, which are essentially matrices of pixel changes [3]. Given two parent perturbations A^1 and A^2, and a binary mask matrix B of the same dimensions (where elements of B are randomly 0 or 1), two offspring perturbations A^{c1} and A^{c2} are generated as:

$$A^{c1} = (A^1 \odot B) + (A^2 \odot (\mathbf{1} - B)) \tag{3.5}$$

$$A^{c2} = (A^1 \odot (\mathbf{1} - B)) + (A^2 \odot B) \tag{3.6}$$

where $\odot$ denotes element-wise multiplication and $\mathbf{1}$ is a matrix of ones. This operation is performed with a crossover probability P_c. Chen et al. [3] set $P_c = 1$ to ensure exploration of combined features. Figure 3.2 (inspired by Figure 3 in [3]) illustrates this.



(a) The schematic of the crossover and variation.

(b) The specific example of crossover and variation.

Figure 3.2: Schematic of crossover and mutation operations on perturbations in POBA-GA. Crossover combines parts of two parent perturbations. Mutation introduces small random changes to an offspring perturbation. (Adapted from Chen et al. [3], Figure 3).

Mutation

Multi-point mutation is applied to the offspring perturbations A^c with a small mutation probability P_m [3]. For an offspring perturbation A_q^c, the mutated perturbation A_q^m is obtained by:

$$A_q^m = \begin{cases} A_q^c \odot C & \text{if rand}(0,1) < P_m \\ A_q^c & \text{otherwise} \end{cases} \tag{3.7}$$

where C is a mutation matrix, typically an identity matrix with a few randomly chosen elements modified (e.g., multiplied by values between 0 and 2) to alter corresponding perturbation pixels. P_m is kept low (e.g., 0.001-0.003) to avoid disrupting good schemata too drastically.

3.3.4 Generation Update and Termination

POBA-GA uses a father-son mixed selection for generation updates, which is a form of elitism [3]. The N perturbations with the highest fitness values from the combined set of current parent perturbations (A^t) and the newly generated (mutated) offspring perturbations (A^m) form the population for the next generation (A^{t+1}). This ensures that high-quality solutions are not lost.

The algorithm terminates when a predefined maximum number of generations is reached, or when an adversarial example's fitness function value exceeds a certain threshold γ, indicating a sufficiently successful and subtle attack.

3.4 Perturbation Evaluation Metric in POBA-GA

Traditional metrics for perturbation size (L_0, L_2, L_∞) have limitations in capturing human perception of changes. POBA-GA introduces a novel perturbation evaluation metric $Z(A)$ (detailed in Eq. (9) of [3]) designed to be more aligned with visual assessment and to better distinguish small perturbations. The metric $Z(A)$ is given by:

$$Z(A) = \sum_{r=1}^{m_r} \sum_{c=1}^{m_c} \left(\frac{1}{1 + e^{-(A_{(rc)} \cdot pm_1 + pm_2)}} - \frac{1}{1 + e^{pm_2}} \right) \tag{3.8}$$

where $A_{(rc)}$ is the perturbation value at pixel (r, c), and pm_1, pm_2 are parameters that shape the sigmoid-like mapping of pixel perturbation values to their contribution to the overall $Z(A)$ score. When $pm_1 = 15, pm_2 = 3$ (as used for machine evaluation in [3]), this metric effectively amplifies the perceived difference between small perturbations, guiding the GA to optimize towards smaller, less perceptible changes once an attack is successful. This $Z(A)$ is the quantity used in the fitness function (Equation 3.2).

3.5 Demonstration of Successful Attacks

This section presents illustrative examples of successful adversarial attacks generated using a Genetic Algorithm-based approach, such as POBA-GA, against target neural network models. The effectiveness of these attacks is typically demonstrated by visually comparing the original input with the generated adversarial example, showcasing the subtle nature of the perturbation, and reporting the model's misclassification.

Figures 3.3 and 3.4 showcase such successful attacks on representative images. For each example, the original image, the computed perturbation, and the resulting adversarial image are displayed. The labels indicate the model's prediction for the original and the adversarial

input, highlighting the misclassification achieved. The perceptibility of the attack can be gauged by visually inspecting the adversarial example and the magnitude of the perturbation.

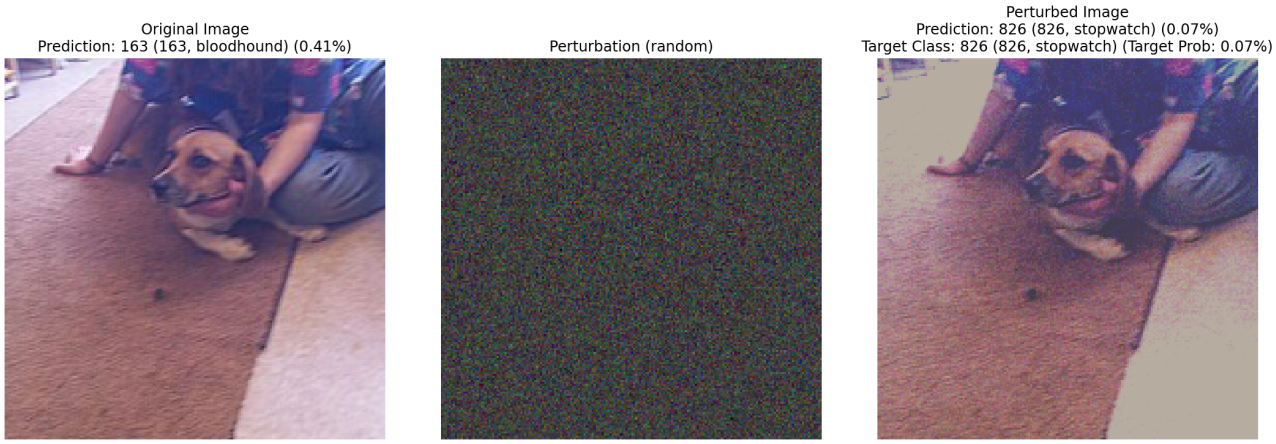


Figure 3.3: Original Image (Label: Bloodhound, Conf: 0.41%) Pertuebed Image (Label: Stopwatch, Conf:0.07%

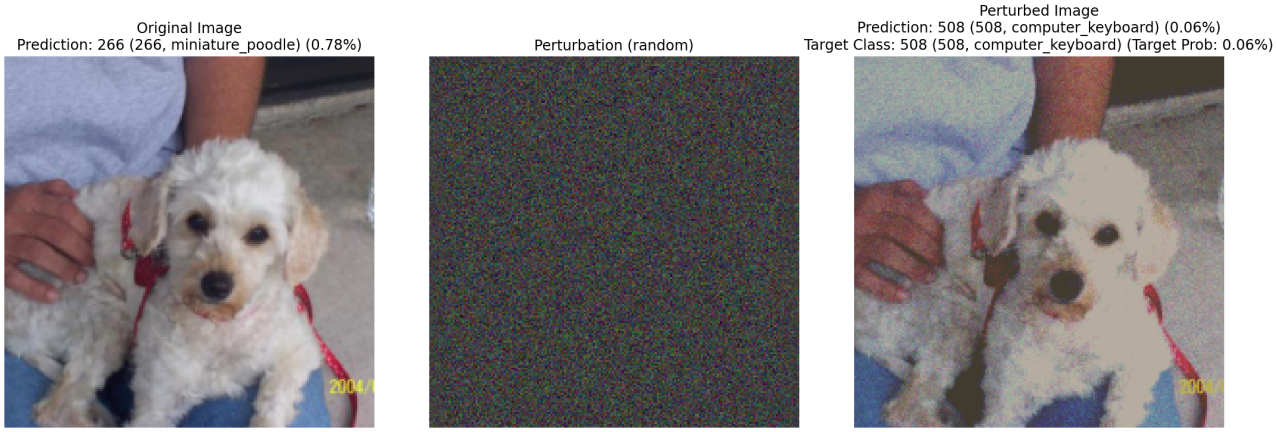


Figure 3.4: Original Image (Label: Miniature Poodle, Conf: 0.78%) Pertuebed Image (Label: Computer Keyboard, Conf:0.06%

In Figure 3.3, the original image, confidently classified as [Bloodhound], is subtly modified. The resulting adversarial image, visually very similar to the original, is misclassified by the target model as [Stopwatch] with high confidence.

Similarly, Figure 3.4 demonstrates an attack on an image of a [Miniature Poodle]. The GA successfully found a perturbation leading to a misclassification as [Computer Keyboard], while the changes remain largely imperceptible to the human eye.

Summary of POBA-GA Performance on Various Benchmarks (Non-Targeted Attacks)

Dataset / Target Model	Perturbation (Per-pixel L_2)	Attack Success Rate (%)	Query Count (Total (Initial Success))
MNIST	3.0e-03	100%	423 (94)
CIFAR-10	6.8e-05	100%	381 (78)
ImageNet64 / VGG19	1.5e-05	96%	3786 (536)
ImageNet64 / ResNet50	1.4e-05	98%	3614 (492)
ImageNet64 / Inception-V3	1.7e-05	95%	3573 (471)

Note: Results extracted from Chen et al. [3], Table 5. "Perturbation" refers to the average per-pixel L_2 norm of the adversarial noise. "Query Count" indicates the average number of model queries needed for a successful attack, with the number for initial success in parentheses. These are for non-targeted attacks.

These demonstrations underscore the capability of GA-based methods to systematically explore the input space and identify vulnerabilities in DNNs. The ability to generate successful attacks with low perceptibility in a black-box scenario highlights the practical threat these techniques pose and motivates further research into robust defense mechanisms.

The POBA-GA method proves a successful application of Genetic Algorithm principles to the challenging problem of black-box adversarial attacks against deep neural networks. By carefully designing the genetic perturbations (representation) , fitness function (balancing attack success and perturbation subtlety), and evolutionary operators (tailored for image perturbations), POBA-GA can efficiently search for and optimize adversarial examples. The employ of a new perturbation metric $Z(A)$ further refines the optimization process towards generating less perceptible attacks. This approach highlights the versatility and power of GAs in tackling complex, real-world optimization tasks in the domain of machine learning security.

Conclusion

This thesis started on an exploration of metaheuristic algorithms, with focusing on Genetic Algorithms (GAs), and ended in a practical demonstration of their application to the complex problem of generating black-box adversarial attacks against Deep Neural Networks (DNNs).

On this work, we first illustrated the fundamental concepts of optimization problems, highlighting their prevalence and the outstanding challenges raised by real-world complexities.

Chapter 2 offered an in-depth exploration of Genetic Algorithms. We underscoring their bio-inspired mechanisms, involving solution representation, population-based search, fitness evaluation, and the main evolutionary operators of selection, crossover, and mutation.

The main contribution of this work, introduced in Chapter 3, involved applying these GA principles to the field of machine learning security. We concentrated on the POBA-GA approach, proving how GAs can be effectively adapted to generate adversarial examples.

Future work could be enlarge from this foundation in different directions.

- Investigate other metaheuristic algorithms or hybrid frameworks for adversarial attack generation could lead even more efficient attack strategies.
- Explore GAs as a component in designing defense mechanisms, for example, by including robust model designs or adversarial training data.

In conclusion, metaheuristic algorithms, and Genetic Algorithms in particular, provide a potent and adaptable model for tackling complex optimization problems. Their implementation to the generation of adversarial attacks not only operates as a compelling case study of their abilities but also contributes to the bigger understanding of AI security.

Bibliography

[1] Bishop, C. M. (2006). *Pattern recognition and machine learning*. Springer.

[2] Boussaïd, I., Lepagnot, J., & Siarry, P. (2013). A survey on optimization metaheuristics. *Information Sciences*, *237*, 82–117.

[3] Chen, J., Su, M., Shen, S., Xiong, H., & Zheng, H. (2019). POBA-GA: Perturbation optimized black-box adversarial attacks via genetic algorithm. *Computers & Security*, *85*, 89–106.

[4] Eiben, A. E., & Smith, J. E. (2003). *Introduction to evolutionary computing*. Springer.

[5] Fister Jr, I., Yang, X.-S., Fister, I., Brest, J., & Fister, D. (2013). A brief review of nature-inspired algorithms for optimization. *Elektrotehniški Vestnik*, *80*(3). Preprint: arXiv:1307.4186.

[6] Gendreau, M., & Potvin, J.-Y. (2005). Metaheuristics in combinatorial optimization. *Annals of Operations Research*, *140*(1), 189–213.

[7] Goldberg, D. E. (1989). *Genetic algorithms in search, optimization and machine learning*. Addison-Wesley.

[8] Holland, J. H. (1975). *Adaptation in natural and artificial systems*. University of Michigan Press. (Reprinted MIT Press, 1992).

[9] Katoch, S., Chauhan, S. S., & Kumar, V. (2021). A review on genetic algorithm: Past, present, and future. *Multimedia Tools and Applications*, *80*, 8091–8126.

[10] Kochenderfer, M. J., & Wheeler, T. A. (2019). *Algorithms for optimization*. MIT Press.

[11] Nocedal, J., & Wright, S. J. (2006). *Numerical optimization* (2nd ed.). Springer.

[12] Yang, X. S. (2010). *Nature-inspired metaheuristic algorithms*. Luniver Press.

الملخص

تتناول هذه الأطروحة الخوارزميات الجينية (GAs) لحل مشاكل التحسين المعقدة، مع التركيز على هجمات الصندوق الأسود العدائية ضد الشبكات العصبية العميقة .(DNNs) تهدف هذه الهجمات إلى إحداث تصنيف خاطئ عبر اضطرابات مدخلات خفية. تعد GAs مناسبة لطبيعتها الخالية من التدرج وقدرتها على التعامل مع مساحات البحث المعقدة. تقدم الأطروحة إطار عمل ،POBA-GA الذي يخصص مكونات ،GAs بما في ذلك دالة لياقة متوازنة (نجاح الهجوم وتقليل الاضطراب) ومقياس Z(A) جديد لإدراك الضوضاء. يبرز نجاح POBA-GA فعالية ،GAs ونقاط ضعف نماذج التعلم الآلي، ويحفز البحث في آليات الدفاع القوية.

الكلمات المفتاحية:خوارزميات الميتاهيوريستيك, الخوارزميات الجينية, التحسين, الهجمات العدائية.
الشبكات العصبية العميقة

Abstract

This thesis explores Genetic Algorithms (GAs) for complex optimization, focusing on black-box adversarial attacks against Deep Neural Networks (DNNs). These attacks generate subtle input perturbations to induce misclassification. GAs are well-suited due to their gradient-free nature and ability to navigate complex search spaces. The POBA-GA framework is introduced, customizing GA components like a specialized fitness function (balancing attack success and minimal perturbation) and a novel perceptibility metric, Z(A). POBA-GA's success highlights GAs' versatility, machine learning vulnerabilities, and the need for robust defense mechanisms.

Keywords:Metaheuristic Algorithms,Genetic Algorithms,Optimization.
Adversarial Attacks,Deep Neural Networks

Résumé

Cette thèse utilise les Algorithmes Génétiques (AG) pour des attaques adversariales en boîte noire contre les Réseaux de Neurones Profonds (RNP), générant des perturbations subtiles pour induire une mauvaise classification. Le cadre POBA-GA, adapté par une fonction de fitness équilibrée et la métrique Z(A), prouve la polyvalence des AG et révèle les vulnérabilités des RNP, soulignant le besoin de défenses robustes.

Mots clés:Algorithmes Métaheuristiques, Algorithmes Génétiques, Optimisation
Attaques Adversariales, Réseaux de Neurones Profonds.