

REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

الجمهورية الجزائرية الديمقراطية الشعبية

MINISTRY OF HIGHER EDUCATION
AND SCIENTIFIC RESEARCH

HIGHER SCHOOL IN APPLIED SCIENCES
- T L E M C E N -



المدرسة العليا في العلوم التطبيقية
École Supérieure en
Sciences Appliquées

وزارة التعليم العالي والبحث العلمي

المدرسة العليا في العلوم التطبيقية
- تلمسان -

End of Study Dissertation

For the Obtention of the Master's Degree

Field: Electronic

Specialty: Industrial Computer Engineering

Presented by : Mohammed El Habib EMBOUAZZA
and Aissameddine BOUKHENDEG

Theme

**Automatic Optimization of Printed Circuit Board
Routing (PCB) Using Metaheuristic Algorithms.**

Publicly defended on 20/ 06/ 2026 , before the jury composed of:

Mr L.MERAD	Professor	ESSA Tlemcen	President
Mr Belloufa Fayssal	Lecturer Grade B	ESSA Tlemcen	Dissertation Director
Mme N.BENAHMED	Professor	ESSA Tlemcen	Co-Dissertation Director
Mr H.MEGNAFI	Lecturer Grade A	ESSA Tlemcen	Examiner 1
Mr Kanoun Ali	Lecturer Grade A	USTHB Alger	Examiner 2

Academic Year : 2025 / 2026

بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ

Dedication

In the Name of Allah, the Most Gracious, the Most Merciful

First and foremost, I would like to express my deepest gratitude to **Allah** for granting me the strength, guidance, and perseverance to complete this work.

To my **Father**, thank you for always encouraging me to strive for excellence, for believing in my dreams even when I doubted myself, and for standing by my side through every challenge. Your wisdom, unwavering support, and the example you set through your actions have been a constant source of inspiration in my life. I am proud to follow the values you have taught me.

To my **Mother**, thank you for your endless love, care, and sacrifices. Your unconditional support, guidance, and encouragement have shaped who I am today. You taught me the importance of integrity, kindness, and maintaining a positive spirit in the face of adversity.

To my **Siblings**, thank you for your love, understanding, and the joy you have brought into my life. Your presence has been a source of strength and comfort throughout this journey.

To my second family - **my friends**. To the brothers who walked every step of this journey beside me: **Habib, Zinou, Aboubaker, Abderrahmen, Mohamed, The Twins, Ahmed, Sid Ahmed, and Amine** - thank you for the laughter, the late nights, the shared struggles, and the memories that will last a lifetime. To those who stood closest beside me through it all, my dearest friends **Abdelmoez Boubekour, Zine Eddine Beddouda, and Wassim Zouai**, you gave me the best laughs, the best encouragements, and a friendship I will always carry with me. Your presence made every difficult moment lighter and every good moment worth remembering. And to the rest of my classmates, thank you for the inspiration, the collaboration, and the spirit we built together throughout these years.

To **TEXQUAD Club**, the club that embraced me from the very first day of this journey. With TechSquad, I always felt at home. It gave me the best experiences, the best laughs, and a community that constantly pushed me to be better and reach higher. TEXQUAD did not just fill my time, it made my life in this school richer, more meaningful, and more memorable.

A special acknowledgment goes to my project partner, **EMBOUAZZA Mohamed El Habib**. Thank you for your dedication, cooperation, and hard work throughout this project. It has been a pleasure sharing this journey with you, overcoming challenges together, and contributing side by side toward achieving this accomplishment.

This achievement is dedicated to all of you, with love and sincere gratitude.

"LOVE YOURZ" - J. Cole

Aissameddine BOUKHENDEG

In the name of Allah, the Most Gracious, the Most Merciful, I would like to begin by expressing my heartfelt gratitude for the blessings, strength, and determination He has granted me throughout my academic journey. Without His guidance, this achievement would not have been possible.

To my beloved Mother, words cannot fully express the depth of my gratitude. Thank you for your unconditional love, endless sacrifices, patience, and constant encouragement. Your support has been the foundation of every success I have achieved, and your faith in me has always given me the courage to move forward.

To my family, thank you for your unwavering support, understanding, and confidence in my abilities. Your encouragement and affection have provided me with the motivation needed to overcome difficulties and pursue my goals with determination.

To my dear brothers, **Aymen and Sofiane**, thank you for always being a source of support, strength, and companionship. Your encouragement and presence have meant a great deal to me throughout this journey.

To my dear partner, **Chahinez**, thank you for standing by my side through every step of this journey. Your love, patience, encouragement, and unwavering faith in me have given me the strength to overcome obstacles and continue striving toward my goals.

To my dear friends, **Zinou, Ahmed, Youcef, Yacine, Aboubaker, Abderrahmane, Amine, second yacine and Mohammed**, thank you for five wonderful years of friendship, support, and unforgettable memories. Together, we have shared both good and difficult moments, celebrated successes, faced challenges, and created memories that I will always cherish. Your companionship, encouragement, laughter, and presence have made this journey far more meaningful and enjoyable. I am truly grateful to have had such friends by my side throughout these years.

I would also like to express my sincere gratitude to the **TEXQUAD Club**, which has been an important part of my life throughout my years at the school. It is within this club that I experienced some of the best moments of my academic journey, met remarkable people, built lasting friendships, and participated in inspiring projects and activities. The memories, experiences, and lessons gained through TEXQUAD will remain with me long after graduation.

A special thanks goes to my project partner, **Aissameddine Boukheneg**. Working with you has been a rewarding experience. Thank you for your commitment, teamwork, and perseverance throughout this project. Together, we faced challenges, exchanged ideas, and successfully reached our goal.

Finally, I extend my sincere appreciation to everyone who contributed, directly or indirectly, to my academic and personal development. Every piece of advice, support, and encouragement has played a valuable role in this achievement.

EMBOUAZZA Mohamed El Habib

Acknowledgement

The completion of this Master's dissertation marks an important milestone in our academic journey. We would like to express our sincere gratitude to all those who contributed, directly or indirectly, to the success of this work.

First and foremost, we would like to express our deepest appreciation to our supervisor, **Dr. Beloufa Fayssal**, for his invaluable guidance, continuous support, patience, and encouragement throughout the development of this dissertation. His expertise, insightful advice, and constructive feedback greatly contributed to the quality and success of this work.

We would also like to extend our sincere thanks to our co-supervisor **Dr. Nadia Benabdallah** for her valuable recommendations, constant availability, and support during the different stages of this project. Her comments and suggestions helped us improve both the technical and scientific aspects of this dissertation.

Our sincere gratitude is addressed to **Professor Boucherit Rouisset**, Director of the Higher School of Applied Sciences of Tlemcen (ESSA Tlemcen), for his leadership and continuous efforts in providing students with a stimulating academic environment that promotes excellence, innovation, and professional development.

We would also like to thank **Dr. Abdellaoui Ghouti**, Head of the Industrial Computing Department, for his dedication and for providing students with the knowledge and skills necessary to meet the challenges of the engineering profession.

Our thanks are extended to all the professors, lecturers, and administrative staff of the **Higher School of Applied Sciences of Tlemcen (ESSA Tlemcen)** for the quality of education they have provided throughout our studies.

We are equally grateful to the members of the jury for accepting to evaluate this dissertation and for the time and effort devoted to reviewing our work.

Finally, we would like to express our heartfelt thanks to our families and friends for their unconditional support, patience, encouragement, and confidence throughout our academic journey. Their presence and motivation have been a constant source of strength and inspiration.

To all those who have contributed in one way or another to the realization of this dissertation, we extend our sincere appreciation and gratitude.

Abstract

Automated printed circuit board (PCB) routing is an NP-hard combinatorial optimisation problem that consists of connecting all electrical nets while minimising track length and satisfying design rule constraints. This work addresses the net ordering problem in single-layer sequential routing, where the routing order directly influences the quality of the final solution. Two bio-inspired metaheuristics, the Whale Optimisation Algorithm (WOA) and the Slime Mould Algorithm (SMA), are adapted to the discrete domain using the Smallest Position Value (SPV) encoding in order to optimise this ordering. Candidate solutions are evaluated using an A* algorithm that explores eight movement directions and incorporates mechanisms ensuring valid, crossing-free routings that respect all design rules. Experiments conducted on both a synthetic and a real printed circuit board show that both approaches produce complete or near-complete routings. SMA achieves the best average performance, while WOA also delivers highly competitive results.

Keywords: PCB routing, net ordering, metaheuristic optimisation, Whale Optimisation Algorithm, Slime Mould Algorithm, A*, SPV encoding.

Résumé

Le routage automatique des circuits imprimés (PCB) est un problème d'optimisation combinatoire classé NP-difficile, consistant à connecter l'ensemble des réseaux électriques (nets) tout en minimisant la longueur des pistes et en respectant les règles de conception. Ce travail s'intéresse au problème de l'ordonnancement des réseaux dans un routage séquentiel monocouche, où l'ordre de routage influence directement la qualité de la solution finale. Deux métaheuristiques bio-inspirées, le Whale Optimization Algorithm (WOA) et le Slime Mould Algorithm (SMA), sont adaptées au domaine discret à l'aide du codage Smallest Position Value (SPV) afin d'optimiser cet ordonnancement. Les solutions candidates sont évaluées à l'aide d'un algorithme A* explorant huit directions de déplacement et intégrant des mécanismes garantissant des routages valides, sans croisement ni violation des règles de conception. Les expérimentations, réalisées sur un circuit imprimé synthétique et un circuit réel, montrent que les deux approches produisent des routages complets ou quasi complets. SMA obtient les meilleures performances moyennes, tandis que WOA fournit également des résultats très compétitifs.

Mots-clés : routage PCB, ordonnancement des réseaux, optimisation par métaheuristiques, Whale Optimization Algorithm, Slime Mould Algorithm, A*, codage SPV.

Contents

List of Figures	12
List of Tables	13
Abbreviations List	13
General Introduction	16
1 PCB Routing and Optimisation — Background and Fundamentals	18
1.1 Introduction	18
1.2 Objectives of the Study	18
1.2.1 General Objective	18
1.2.2 Specific Objectives	19
1.2.3 Scope and Limitations	19
1.3 General Presentation of Printed Circuit Boards	19
1.3.1 History of Printed Circuit Boards	19
1.3.2 Definition and Basic Concepts	20
1.3.2.1 What is a Printed Circuit Board?	20
1.3.2.2 Basic Components of a PCB	20
1.3.2.3 Key Terminology	21
1.3.3 Types of Printed Circuit Boards	21
1.3.3.1 Single-Layer PCBs	21
1.3.3.2 Double-Layer PCBs	21
1.3.3.3 Multilayer PCBs	21
1.3.3.4 Flexible and Rigid-Flex PCBs	22
1.3.4 PCB Manufacturing Process Overview	22
1.3.5 Standard PCB Design Flow	22
1.4 Detailed Study of PCB Routing	23
1.4.1 Definition and Role of the Routing Step	23
1.4.2 Problem Formulation	24
1.4.3 Routing Phases	24
1.4.3.1 Global Routing	24
1.4.3.2 Detailed Routing	24
1.4.4 Routing Constraints and Design Rules	25

1.4.5	Routing Objectives	25
1.4.6	Complexity of the PCB Routing Problem	26
1.5	Historical Background and Related Work	26
1.5.1	Early Algorithmic Approaches to PCB Routing	26
1.5.2	Related Work	27
1.5.2.1	Ant Colony Optimisation for PCB Routing	27
1.5.2.2	Polygon-Based Dynamic Partitioning	27
1.5.2.3	Evolution-Based Circuit Routing Planner	28
1.5.2.4	Multi-Objective ACO for PCB Routing	28
1.5.2.5	Simulated Annealing for PCB Routing	29
1.5.3	Synthesis and Research Gap	29
1.6	Conclusion	30
2	Theoretical Foundations and Selection of Algorithms	31
2.1	Introduction	31
2.2	Metaheuristic Optimisation — General Principles	32
2.2.1	Definition and Characteristics	32
2.2.2	Classification of Metaheuristic Algorithms	32
2.2.3	Key Concepts	34
2.2.4	Why Metaheuristics for PCB Routing?	35
2.3	Selection of Candidate Algorithms	35
2.3.1	Whale Optimisation Algorithm	36
2.3.1.1	Biological Inspiration	36
2.3.1.2	Mathematical Model	37
2.3.1.3	Pseudocode	38
2.3.1.4	Key Parameters	38
2.3.1.5	Strengths, Limitations, and Relevance to PCB Routing	39
2.3.2	Slime Mould Algorithm	39
2.3.2.1	Biological Inspiration	39
2.3.2.2	Mathematical Model	40
2.3.2.3	Pseudocode	42
2.3.2.4	Key Parameters	42
2.3.2.5	Strengths, Limitations, and Relevance to PCB Routing	42
2.3.3	Harris Hawks Optimiser	43
2.3.3.1	Biological Inspiration	43
2.3.3.2	Mathematical Model	43
2.3.3.3	Pseudocode	45
2.3.3.4	Key Parameters	45
2.3.3.5	Strengths, Limitations, and Relevance to PCB Routing	46
2.3.4	Salp Swarm Algorithm	46
2.3.4.1	Biological Inspiration	46
2.3.4.2	Mathematical Model	46
2.3.4.3	Pseudocode	47
2.3.4.4	Key Parameters	47
2.3.4.5	Strengths, Limitations, and Relevance to PCB Routing	48
2.3.5	Sine Cosine Algorithm	48
2.3.5.1	Mathematical Foundation	48
2.3.5.2	Mathematical Model	48

2.3.5.3	Pseudocode	49
2.3.5.4	Key Parameters	49
2.3.5.5	Strengths, Limitations, and Relevance to PCB Routing	49
2.4	Comparative Analysis of Candidate Algorithms	50
2.4.1	Criteria for Comparison	50
2.4.2	Comparative Summary	50
2.4.3	Discussion	51
2.5	Algorithm Selection and Justification	52
2.6	A* Pathfinding Algorithm	53
2.6.1	Overview and Role in the Proposed System	53
2.6.2	Algorithm Principle	53
2.6.3	8-Directional Movement and Octile Heuristic	54
2.6.4	Corner-Cutting Prevention	54
2.6.5	Trace Isolation and Sequential Routing	54
2.7	Adaptation of WOA and SMA to the PCB Routing Problem	54
2.7.1	Solution Representation	56
2.7.2	Discretisation and Operator Mapping	56
2.7.3	Boundary Handling	56
2.7.4	Fitness Function	56
2.8	Conclusion	57
3	Implementation and Experimental Evaluation	58
3.1	Introduction	58
3.2	Overview of the Implemented Solution	58
3.2.1	General Architecture	58
3.2.2	Software Architecture	58
3.2.3	Technology Stack and Development Environment	59
3.3	PCB Problem Modelling	60
3.3.1	Grid Representation	60
3.3.2	Netlist Format and EasyEDA Input Pipeline	61
3.4	A* Router Implementation	62
3.4.1	8-Directional Movement	62
3.4.2	Corner Cutting Prevention	62
3.4.3	Trace Isolation	62
3.5	Fitness Function Implementation	63
3.5.1	Overall Structure	63
3.5.2	Wirelength Computation	63
3.5.3	Via Count Computation	63
3.5.4	Design Rule Violation Detection	63
3.5.5	Weight Calibration	64
3.6	WOA Implementation	64
3.6.1	Initialisation	64
3.6.2	Position Update and SPV Decoding	64
3.6.3	Boundary Handling	65
3.7	SMA Implementation	65
3.7.1	Initialisation	65
3.7.2	Weight Computation and Position Update	65
3.7.3	Boundary Handling	66

3.8	Experimental Setup	66
3.8.1	Experimental Protocol	66
3.8.2	Test Instances	67
3.8.3	Parameter Settings	67
3.8.4	Evaluation Metrics	67
3.9	GUI Experimental Platform	68
3.10	Results and Analysis	69
3.10.1	Board 1 — Benchmark Board	69
3.10.1.1	Fixed Iteration Budget	69
3.10.1.2	Convergence-Based Stopping	70
3.10.2	Board 2 — Buck Converter Board	70
3.10.2.1	Fixed Iteration Budget	71
3.10.2.2	Convergence-Based Stopping	71
3.10.3	Convergence Curves	72
3.10.4	Cross-Condition Summary	73
3.10.5	Best Routing Solutions	74
3.10.6	Discussion	75
3.11	Conclusion	75
 General Conclusion		 77
 Bibliography		 79

List of Figures

1.1	Cross-sectional comparison of single-layer, double-layer, and multi-layer PCB structures.	22
1.2	PCB design flow from concept to fabrication [23].	23
2.1	Hierarchical classification of metaheuristic algorithms based on their source of inspiration [18].	34
2.2	(A) Bubble-net feeding behaviour of humpback whales. (B) Shrinking encirclement mechanism. (C) Spiral position update. Adapted from [16].	37
2.3	<i>Physarum polycephalum</i> extending pseudopods toward food sources (top), and the SMA weight distribution mechanism in 2D and 3D (bottom): agents closer to the best food source receive stronger weights $W > 1$ (thickened tubes) while weaker agents receive $W < 1$ (thinned tubes). Adapted from [11].	40
2.4	Radar chart of the five candidate algorithms across the six comparison criteria.	51
2.5	Block diagram of the proposed PCB routing pipeline.	55
3.1	Mean convergence curves — benchmark board, $N = 100$, $T_{\max} = 50$, 30 runs. Shaded band: ± 1 standard deviation.	72
3.2	Mean convergence curves — buck converter board, $N = 100$, $T_{\max} = 50$, 30 runs. Shaded band: ± 1 standard deviation.	72
3.3	Best routing solution on the benchmark board (25 nets, 80×60 grid).	74
3.4	Best routing solution on the buck converter board (LM2596S-ADJ, 18 nets, 80×110 grid), produced via the automated input pipeline.	74

List of Tables

1.1	Comparative overview of PCB types	22
2.1	WOA key parameters and their roles.	39
2.2	SMA key parameters and their roles.	42
2.3	HHO key parameters	45
2.4	SSA key parameters	47
2.5	SCA key parameters	49
2.6	Comparative analysis of the five candidate algorithms against PCB routing criteria	51
2.7	Fitness function terms and weights	57
3.1	Python module architecture of the PCB routing tool.	59
3.2	Technology stack of the PCB routing optimisation tool.	59
3.3	Fitness function weights and their justification.	64
3.4	PCB test board specifications.	67
3.5	Fixed experimental parameters — identical for WOA and SMA. . . .	68
3.6	Fixed-iteration results on the benchmark board ($T_{\max} = 50$, 30 runs per configuration).	69
3.7	Convergence-based stopping results on the benchmark board ($\delta = 5$, 30 runs per configuration).	70
3.8	Fixed-iteration results on the buck converter board ($T_{\max} = 50$, 30 runs per configuration).	71
3.9	Convergence-based stopping results on the buck converter board ($\delta = 5$, 30 runs per configuration).	71
3.10	Best configuration per board and stopping criterion (lowest best-fitness across all population sizes and algorithms).	73

Abbreviations List

ACO	Ant Colony Optimisation
BGA	Ball-Grid Array
CAD	Computer-Aided Design
CSV	Comma-Separated Values
DE	Differential Evolution
DFT	Design for Testability
DRC	Design Rule Check
DRV	Design Rule Violation
EDA	Electronic Design Automation
EMI	Electromagnetic Interference
ENIG	Electroless Nickel Immersion Gold
FR-4	Flame Retardant 4 (standard PCB substrate material)
GA	Genetic Algorithm
GSA	Gravitational Search Algorithm
GUI	Graphical User Interface
GWO	Grey Wolf Optimiser
HASL	Hot Air Solder Leveling
HDI	High-Density Interconnect
HHO	Harris Hawks Optimiser
HS	Harmony Search
IC	Integrated Circuit
ILP	Integer Linear Programming
IoT	Internet of Things
LED	Light-Emitting Diode
MA	Memetic Algorithm
MCM	Multi-Chip Module
NFL	No-Free-Lunch (theorem)
NP	Non-deterministic Polynomial (as in NP-hard, NP-complete)
PCB	Printed Circuit Board
PI	Polyimide

PSO	Particle Swarm Optimisation
SA	Simulated Annealing
SCA	Sine Cosine Algorithm
SMA	Slime Mould Algorithm
SMT	Surface-Mount Technology
SPV	Smallest Position Value
SSA	Salp Swarm Algorithm
TLBO	Teaching-Learning-Based Optimisation
TS	Tabu Search
VC	Via Count
VLSI	Very Large Scale Integration
WL	Wirelength
WOA	Whale Optimisation Algorithm

General Introduction

Printed Circuit Board (PCB) design is one of the most critical and technically demanding stages in the development of modern electronic products. A PCB must interconnect dozens to hundreds of components (resistors, capacitors, inductors, diodes, integrated circuits, and connectors) through metal tracks routed across one or more conductive layers, while satisfying a complex set of electrical, thermal, and geometric constraints.

As the density and operating frequency of electronic systems increase, the quality of the routing solution has a direct impact on signal integrity, electromagnetic compatibility, thermal dissipation, and the reliability and commercial viability of the final product.

Among the many sub-problems that constitute the PCB design process, automatic routing occupies a central place: given a set of nets, where each net is a pair or group of component pins that must be electrically connected, the router must find a set of collision-free track segments on the routing grid, realising all required connections while minimising total track length, avoiding Design Rule Violations (DRVs), and preventing any physical crossing or cell sharing between tracks.

Even in its simplest form, restricted to single-layer routing with two terminals per net, this problem remains NP-hard. In practice, as the number of nets grows, instances rapidly become too complex to be solved efficiently by exact combinatorial methods.

The challenge addressed in this work is the net ordering problem in sequential routing on a single-layer PCB grid. When nets are routed one by one using a shortest-path algorithm, each already-routed track becomes a permanent obstacle for subsequent ones. The order in which nets are processed therefore fundamentally determines both the feasibility of completing the routing and the total track length. A poor ordering can render certain nets unroutable, even when a globally feasible solution exists.

Selecting the optimal permutation of n nets requires exploring a search space of size $n!$, which is combinatorially intractable by exhaustive search for any realistic board. Beyond net ordering, two additional constraints must be satisfied: pad protection (no track may cross a pad belonging to another net, which would cause a short circuit) and crossing prevention, covering both cell-sharing crossings and diagonal visual crossings, in order to guarantee a physically realisable topology.

Given the combinatorial nature and practical scale of the problem, classical exact methods are insufficient and a more intelligent search strategy is required. The combinatorial optimisation literature offers a broad family of population-based meta-

heuristic methods that have demonstrated strong performance on NP-hard problems of similar structure, motivating their exploration in the context of PCB routing.

The challenge then lies not only in choosing an appropriate optimisation framework, but also in designing a problem representation that bridges the continuous search space of these algorithms with the discrete space of net ordering permutations, and in defining an evaluation function that faithfully reflects routing quality in terms of track length, connectivity, and compliance with design rules.

The objectives of this work are:

1. to formalise the net ordering problem for sequential PCB routing as a continuous combinatorial optimisation problem suited to metaheuristics;
2. to implement and validate a correct, crossing-free, pad-aware routing kernel based on eight-directional movement on a discrete grid;
3. to select, adapt, and compare appropriate metaheuristic algorithms for the net ordering problem using a suitable encoding and a multi-objective evaluation function;
4. to conduct a rigorous comparative experimental study on a real buck converter netlist and a synthetic test board; and
5. to design and develop a complete software tool including a professional graphical user interface for use by engineers and students.

The dissertation is organised into three chapters. The first chapter establishes the theoretical foundations: an introduction to PCBs, formalisation of the routing problem, complexity analysis, and a review of existing work. The second chapter details the proposed methodology: metaheuristic principles, selection and adaptation of the WOA and SMA algorithms, and a description of the A* router and software architecture. The third chapter presents the implementation, experimental protocol, results obtained on both a synthetic board and a real buck converter board sourced from EasyEDA, and a comparative performance analysis. A general conclusion summarises the contributions and opens perspectives for future work.

PCB ROUTING AND OPTIMISATION — BACKGROUND AND FUNDAMENTALS

1.1 Introduction

Printed circuit board (PCB) design is a fundamental stage in the development of modern electronic systems. Among the various phases of the design process, routing occupies a central position: it consists of establishing electrical connections between components while satisfying manufacturing constraints and performance requirements. With the increasing integration density and functional complexity of electronic boards, this task has become a combinatorial optimisation problem of considerable difficulty.

The PCB routing problem is recognised as NP-hard, as it requires finding solutions that simultaneously satisfy multiple objectives: minimisation of track length, reduction of the number of vias, and compliance with design rules. While classical algorithms such as the Lee algorithm or the A* algorithm can efficiently determine the best path for an individual connection, they do not guarantee global optimisation when several nets must be routed concurrently. This limitation has motivated the development of numerous metaheuristic-based approaches.

This chapter presents the theoretical foundations necessary for understanding the work reported in this dissertation. It first introduces the fundamental concepts related to printed circuit boards, then formalises the routing problem, its constraints, its objectives, and its complexity. Finally, it provides a review of the main approaches from the literature, highlighting the limitations of existing methods and the motivations that led to the contributions presented.

1.2 Objectives of the Study

1.2.1 General Objective

The general objective of this work is to investigate metaheuristic optimisation algorithms applied to the PCB routing problem, with the aim of selecting and implementing an approach capable of producing higher-quality routing solutions than classical deterministic methods, in terms of wirelength, via count, and design rule compliance.

1.2.2 Specific Objectives

To achieve this general objective, the following specific objectives are pursued:

- To formally define the PCB routing problem and identify the principal optimisation criteria that a routing solution must satisfy.
- To review and analyse existing metaheuristic approaches applied to PCB routing, in order to identify a suitable algorithm for implementation.
- To implement the selected metaheuristic algorithm and apply it to a real PCB routing instance, evaluating its performance in terms of wirelength, via count, and design rule compliance.
- To analyse the results obtained and identify the strengths and limitations of the chosen approach with respect to the target problem.

1.2.3 Scope and Limitations

This study focuses exclusively on the routing phase of the PCB design flow; component placement is assumed to be fixed and is not subject to optimisation. The algorithm selection and its theoretical justification are presented in Chapter 2, while the implementation and experimental evaluation are detailed in Chapter 3.

The algorithms are implemented and tested in a controlled simulation environment. Extending the proposed method to multilayer boards, high-density interconnects, or signal integrity-constrained routing falls outside the scope of this work and is identified as a direction for future research.

1.3 General Presentation of Printed Circuit Boards

1.3.1 History of Printed Circuit Boards

The printed circuit board (PCB) stands as one of the most transformative inventions in the history of electronics. Its development did not emerge from a single discovery but rather evolved progressively over several decades, driven by military needs, industrial demands, and technological breakthroughs.

The earliest foundations of PCB technology can be traced back to 1903, when German inventor Albert Hanson described flat foil conductors laminated onto an insulating board in multiple layers. In 1927, Charles Ducas patented a method of electroplating circuit patterns directly onto an insulated surface using conductive ink. However, it was Austrian engineer Paul Eisler who is widely credited with developing the first functional PCB in 1936, as part of a radio set, by etching a circuit onto a copper foil bonded to a glass-reinforced non-conductive substrate [17].

During World War II, the technology gained significant military momentum, and was adopted on a large scale for the production of proximity fuses. Following the war, it was declassified and made available for commercial use in 1948, marking the beginning of the modern PCB industry.

Throughout the 1950s and 1960s, PCB designs evolved from single-sided boards to double-sided configurations with plated through-holes. The introduction of integrated circuits (ICs) during this period further revolutionised PCB development,

leading to a dramatic reduction in board size and a significant increase in conductor layer count.

The 1970s and 1980s witnessed the advent of surface-mount technology (SMT), which allowed components to be soldered directly onto the surface of the board, substantially improving assembly density and reducing manufacturing costs. Simultaneously, the introduction of Computer-Aided Design (CAD) tools transformed the PCB design workflow, replacing manual drafting with automated layout software.

By the 1990s, PCB technology had matured into a highly sophisticated discipline, with the introduction of ball-grid array (BGA) packaging, design-for-testability (DFT) guidelines, and microvias enabling high-density interconnect (HDI) designs. In the twenty-first century, PCBs continue to evolve in response to emerging technologies such as the Internet of Things (IoT), 5G communications, and artificial intelligence [17].

1.3.2 Definition and Basic Concepts

1.3.2.1 What is a Printed Circuit Board?

A Printed Circuit Board (PCB) is a flat, rigid structure made of insulating material — typically fiberglass-reinforced epoxy resin known as FR-4 — onto which a conductive layer of copper is deposited and selectively etched to form a precise network of electrical pathways. These pathways, commonly referred to as traces, serve to mechanically support and electrically interconnect the various electronic components mounted on the board [23].

Unlike conventional point-to-point wiring, the PCB offers a standardised, reproducible, and compact platform for realising complex electronic circuits, enabling a dramatic reduction in the size, weight, and cost of electronic systems while simultaneously improving their reliability and manufacturability [17].

1.3.2.2 Basic Components of a PCB

A PCB is a composite structure composed of several distinct functional layers, each serving a specific role [17]:

- **Substrate:** The insulating base material, most commonly FR-4 glass epoxy, which provides mechanical rigidity and electrical isolation between conductive layers.
- **Copper Layer:** A thin foil of copper laminated onto the substrate surface. Through a photolithographic etching process, unwanted copper is removed, leaving behind the desired circuit traces, pads, and planes.
- **Solder Mask:** A layer of polymer material applied over the copper surface to prevent oxidation, reduce the risk of solder bridging, and improve the board's long-term environmental reliability.
- **Silkscreen:** A layer of non-conductive ink printed on the board surface to provide human-readable information such as component reference designators, polarity markers, and pin indicators.

-
- **Components:** The active and passive electronic elements mounted on the PCB, including resistors, capacitors, diodes, transistors, and integrated circuits.

1.3.2.3 Key Terminology

A thorough understanding of PCB design requires familiarity with the following fundamental terms [23]:

- **Pad:** A small exposed area of copper to which a component lead is soldered.
- **Via:** A plated hole that establishes an electrical connection between two or more conductive layers.
- **Net:** A named electrical node representing a set of pins that must be electrically connected.
- **Trace:** A conductive copper path carrying electrical signals or power between two points.
- **Clearance:** The minimum allowable distance between any two conductive elements on the board.
- **Layer:** A single conductive or insulating plane within the PCB stack-up.

1.3.3 Types of Printed Circuit Boards

PCBs are classified according to several criteria, including the number of conductive layers, the mechanical flexibility of the substrate, and the complexity of the interconnect structure.

1.3.3.1 Single-Layer PCBs

The single-layer PCB consists of a single conductive copper layer on one side of the insulating substrate. Due to their simplicity, they are widely used in low-complexity, cost-sensitive applications such as power supplies and LED lighting systems.

1.3.3.2 Double-Layer PCBs

The double-layer PCB incorporates two conductive copper layers interconnected by plated through-holes known as vias. This configuration effectively doubles the available routing area and is among the most commonly produced PCB types in the industry.

1.3.3.3 Multilayer PCBs

Multilayer PCBs incorporate three or more conductive layers separated by insulating prepreg material. Additional layers serve to provide dedicated power and ground planes, improving power integrity and reducing electromagnetic interference (EMI) [23].

1.3.3.4 Flexible and Rigid-Flex PCBs

Flexible PCBs are manufactured using thin, pliable substrate materials such as polyimide (PI), allowing the board to be bent or shaped to conform to the geometry of the end product. Rigid-flex PCBs combine rigid and flexible sections into a single integrated structure, eliminating the need for inter-board connectors [23].

A comparative overview of PCB types is presented in Table 1.1 and illustrated in Figure 1.1.

Table 1.1: Comparative overview of PCB types

Type	Layers	Flexible	Typical Applications
Single-Layer	1	No	LED, simple power supplies
Double-Layer	2	No	Converters, controllers
Multilayer	≥ 4	No	Computers, telecom, aerospace
Flexible	1–2	Yes	Wearables, smartphones
Rigid-Flex	≥ 2	Hybrid	Medical, defense

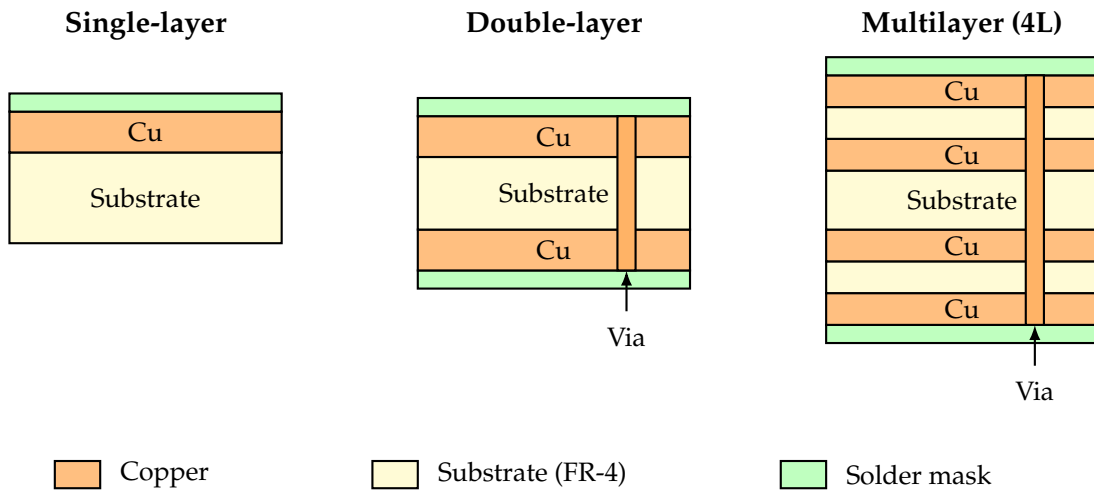


Figure 1.1: Cross-sectional comparison of single-layer, double-layer, and multilayer PCB structures.

1.3.4 PCB Manufacturing Process Overview

The fabrication of a PCB follows a well-defined industrial process. Starting from the base substrate material, the copper layer is laminated and subsequently patterned through a photolithographic etching process. Drilled holes are then plated to form vias, after which the solder mask and silkscreen layers are applied. For multilayer boards, individual layers are laminated together under heat and pressure before drilling and plating. The final step involves surface finishing — such as HASL or ENIG — to protect exposed copper pads and ensure reliable solderability [17].

1.3.5 Standard PCB Design Flow

The design of a printed circuit board follows a structured and iterative sequence of steps, as illustrated in Figure 1.2 [23]:

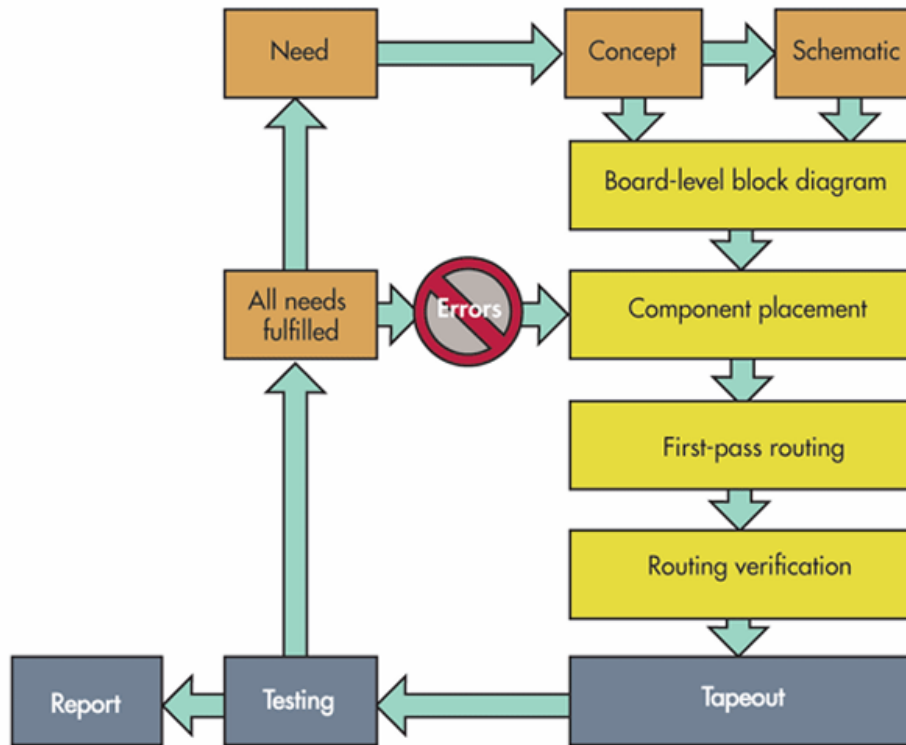


Figure 1.2: PCB design flow from concept to fabrication [23].

1. **Schematic Capture:** The electronic circuit is formally described as a schematic diagram, defining all components, their electrical characteristics, and the connections between their pins. A netlist is generated as output.
2. **Component Placement:** Components are positioned on the board following functional grouping, thermal, and routability guidelines.
3. **Routing:** Conductive traces are drawn to physically implement all electrical connections defined in the netlist. This step constitutes the primary focus of the present work and is studied in detail in Section 1.4.
4. **Design Rule Check (DRC):** The layout is verified against manufacturing and electrical constraints to detect clearance violations, unconnected nets, and trace width errors.
5. **Gerber File Generation:** Fabrication output files are generated and transmitted to the PCB manufacturer.

1.4 Detailed Study of PCB Routing

1.4.1 Definition and Role of the Routing Step

Routing is the process of establishing physical conductive connections between all electrically related pins and pads on a printed circuit board, as defined by the netlist generated during the schematic capture phase. Each connection to be routed is

referred to as a net, and the complete set of nets that must be realised on the board constitutes the routing problem instance [21].

Routing is arguably the most complex and time-consuming step in the PCB design flow. It directly determines the electrical performance, signal integrity, manufacturability, and overall quality of the final board. A poor routing solution may introduce excessive trace lengths, unnecessary vias, congested regions, or design rule violations — all of which negatively impact both the fabrication yield and the functional behaviour of the circuit.

Routing may be performed manually by the designer, automatically by an autorouter algorithm, or through a combination of both approaches. In practice, critical traces — such as power rails, clock signals, and differential pairs — are typically routed manually to ensure optimal performance, while the remaining connections are delegated to automated routing tools.

1.4.2 Problem Formulation

Formally, given a PCB layout represented as a two-dimensional grid $G = (V, E)$, where V is the set of grid cells and E is the set of adjacency edges between neighbouring cells, the routing problem consists of finding, for each net $n_i = \{p_1, p_2, \dots, p_k\}$ defined by a set of pads, a set of connected paths on G that physically links all pads belonging to n_i , without violating any design constraint [21].

Unlike theoretical graph problems, PCB routing is further complicated by the presence of physical obstacles — existing components, board boundaries, and previously routed traces — that restrict the available routing space. The routing algorithm must navigate this constrained environment while simultaneously optimising one or more performance metrics.

1.4.3 Routing Phases

The routing process is typically decomposed into two successive phases: global routing and detailed routing.

1.4.3.1 Global Routing

Global routing operates at a coarse level of abstraction. The board is partitioned into rectangular regions, and the router determines, for each net, the sequence of regions through which the connection will pass, without yet specifying the exact geometric path of each trace. The primary objective at this stage is to distribute the routing demand evenly across the board, avoiding congested regions and ensuring that a feasible detailed routing solution exists [22].

1.4.3.2 Detailed Routing

Detailed routing translates the coarse assignments produced by global routing into exact geometric paths, specifying the precise coordinates of every trace segment and via on each layer. At this stage, all design rule constraints — including minimum trace width, minimum clearance, and via placement rules — must be strictly enforced. Detailed routing is the computationally most demanding phase of the

routing process, and constitutes the primary focus of the algorithmic contributions of this work [22].

1.4.4 Routing Constraints and Design Rules

A valid routing solution must satisfy a set of physical and electrical constraints imposed by both the manufacturing process and the functional requirements of the circuit. The principal constraints are as follows [23]:

- **Clearance:** A minimum spatial separation must be maintained between any two conductive elements on the same layer to prevent electrical short circuits. Clearance rules are enforced as hard constraints that any valid solution must satisfy.
- **Trace Width:** Each trace must satisfy a minimum width requirement determined by the current it carries. Power nets require wider traces than signal nets. Trace width also affects impedance, which is critical in high-frequency designs.
- **Trace Crossings:** On a single conductive layer, two traces cannot physically cross. When a crossing is unavoidable, the router must introduce a via to change layers.
- **Layer Assignment:** In multilayer PCBs, the router must assign each trace segment to an appropriate conductive layer, respecting any layer-specific routing rules and minimising interlayer transitions.

1.4.5 Routing Objectives

Building on the constraints defined in Section 1.4.4, three primary optimisation objectives are commonly targeted in the PCB routing literature [21, 3]:

- **Wirelength minimisation:** Reducing the total length of routed traces decreases signal propagation delay, resistive losses, and electromagnetic emissions.
- **Via count minimisation:** Each via introduces additional resistance, inductance, and manufacturing cost. Minimising the number of vias improves both electrical performance and fabrication yield.
- **Design rule compliance:** A valid routing solution must satisfy all clearance, width, and spacing constraints imposed by the manufacturing process. Unresolved violations directly prevent board fabrication.

The formal aggregation of these objectives into a cost function, as well as the definition of the associated weighting parameters, depends on the specific algorithm selected and is therefore presented in Chapter 3.

1.4.6 Complexity of the PCB Routing Problem

The PCB routing problem belongs to the class of NP-hard combinatorial optimisation problems. This classification implies that no known algorithm can solve all instances of the problem to proven optimality in polynomial time with respect to the problem size [21].

The combinatorial explosion that characterises this problem can be intuitively understood by considering that for a board with n nets, each requiring a path through a grid of $m \times m$ cells, the number of candidate routing solutions grows exponentially with both n and m . In practice, modern PCB designs may involve hundreds or thousands of nets on boards with grid resolutions in the tens of thousands of cells per layer, rendering exhaustive search computationally intractable.

This fundamental complexity directly motivates the use of metaheuristic optimisation algorithms, which sacrifice the guarantee of global optimality in exchange for the ability to find high-quality solutions within acceptable computation times. The metaheuristic approaches most relevant to this problem are introduced through the review of related literature presented in the following section.

1.5 Historical Background and Related Work

1.5.1 Early Algorithmic Approaches to PCB Routing

The history of automated PCB routing algorithms spans more than six decades, evolving in parallel with the growth in circuit complexity and the increasing demands of the electronics industry.

The first systematic approach to automated routing was introduced by Lee in 1961, who proposed a breadth-first search algorithm operating on a two-dimensional grid — commonly known as Lee’s algorithm or the maze router. This method guarantees the discovery of the shortest path between two points if one exists, but suffers from prohibitive memory and computation requirements as board size increases [19].

To address these limitations, heuristically guided search algorithms were subsequently developed. The A* algorithm, which augments the breadth-first search with an admissible heuristic function — typically the Manhattan distance to the target — dramatically reduces the number of cells explored while still guaranteeing path optimality for individual nets. A* became the reference baseline for single-net routing and remains widely used in modern EDA tools [19].

Despite their effectiveness at the single-net level, both Lee’s algorithm and A* are inherently sequential: they route nets one at a time, meaning that earlier routing decisions constrain the solution space available to later nets. This sequential nature prevents any global optimisation of the complete routing solution and frequently leads to suboptimal results in terms of total wirelength, via count, and routing completion rate.

As PCB designs grew in complexity throughout the 1980s and 1990s, channel routers and switchbox routers were developed to handle structured routing regions, while gridless routers were introduced to accommodate the finer geometries of surface-mount technology. A comprehensive survey of these classical approaches is provided by Shahookar and Mazumder [19] and by Yan et al. [22], who docu-

ment the progression from grid-based deterministic methods toward more flexible optimisation-driven approaches.

The fundamental limitation shared by all classical deterministic algorithms is their inability to simultaneously optimise the routing of all nets in a coordinated manner. This limitation, combined with the NP-hard complexity of the general routing problem, has motivated the growing adoption of metaheuristic optimisation methods — population-based or stochastic approaches capable of exploring the full solution space in a global and coordinated fashion. The following section reviews the most relevant contributions in this direction.

1.5.2 Related Work

1.5.2.1 Ant Colony Optimisation for PCB Routing

Wang [21] addresses the problem of automatic PCB routing by proposing a routing strategy based on the Ant Colony Optimisation (ACO) algorithm. ACO is a population-based metaheuristic inspired by the foraging behaviour of ant colonies, in which artificial ants construct candidate solutions by traversing a graph and depositing pheromone trails on promising edges. Over successive iterations, the pheromone accumulation guides the colony toward increasingly shorter and higher-quality paths.

In this work, the PCB routing area is discretised into a grid, and each ant constructs a complete routing path for a given net by probabilistically selecting adjacent cells based on a combination of pheromone intensity and a distance-based heuristic. The pheromone update mechanism simultaneously accounts for trace length minimisation and the avoidance of design rule violations.

The experimental results demonstrate that the ACO-based strategy achieves significant improvements in routing efficiency and solution quality compared to classical deterministic approaches, with reduced total wirelength and improved routing completion rates. However, the study acknowledges that parameter sensitivity — particularly the pheromone evaporation rate and the relative weight of the heuristic component — requires careful tuning and may affect convergence behaviour on larger or more complex board layouts.

1.5.2.2 Polygon-Based Dynamic Partitioning

He et al. [8] propose a pad-focused PCB routing algorithm based on polygon-based dynamic partitioning of the routing area. Rather than operating on a uniform grid, the method dynamically segments the board into polygonal regions centred around component pads, allowing the router to adapt its search space to the local geometry of the layout.

The core contribution of this work lies in its ability to handle high pin density designs, where uniform grid representations introduce significant computational overhead and fail to capture the fine geometric constraints around densely packed components. By focusing the routing effort on pad-centric regions and progressively expanding the search space, the algorithm achieves competitive routing quality with reduced computation time.

Experimental results on commercial PCB designs demonstrate that the proposed approach achieves complete routing in cases where conventional grid-based routers

fail, while maintaining compliance with all design rule constraints. However, the method remains fundamentally deterministic and does not explore alternative global routing topologies, which may lead to locally optimal solutions in complex routing scenarios.

Although this approach is deterministic rather than metaheuristic, it serves as a useful reference for understanding the limitations of grid-based representations in high pin density scenarios, which motivate the adoption of more flexible metaheuristic optimisation strategies — the focus of the remaining surveyed works and of the present study.

1.5.2.3 Evolution-Based Circuit Routing Planner

Huang et al. [10] present a scalable circuit routing planner that formulates the routing problem as a combinatorial optimisation task and addresses it using an evolution-based optimisation framework. The central contribution of the work is the introduction of a ranking cost function that replaces the traditional scalar cost aggregation with a ranking-based evaluation scheme, allowing the optimiser to more effectively discriminate between candidate solutions in dense and highly constrained routing scenarios.

The evolutionary framework operates on a population of complete routing solutions, applying selection, crossover, and mutation operators to iteratively improve the population. The ranking cost function guides the evolutionary search by prioritising solutions that simultaneously minimise wirelength, reduce via count, and respect design rule constraints, without requiring manual tuning of weighting coefficients.

The proposed planner demonstrates strong scalability on large-scale benchmark instances derived from both VLSI and PCB routing problems, outperforming several state-of-the-art routers in terms of solution quality and computational efficiency. The primary limitation identified is the increased memory requirement associated with maintaining a large population of complete routing solutions, which may constrain applicability on resource-limited platforms.

1.5.2.4 Multi-Objective ACO for PCB Routing

Azambuja et al. [3] introduce a multi-objective variant of the Ant Colony Optimisation algorithm specifically designed for PCB trace routing. Unlike single-objective ACO formulations, the proposed approach maintains a separate ant colony for each net to be routed, and evaluates candidate paths against a multi-objective cost function that simultaneously minimises three criteria: total trace length, the number of inter-trace crossings, and the length difference between traces — the latter being critical for high-speed bus routing applications where signal timing must be balanced.

The pheromone update mechanism is adapted to reflect the multi-objective nature of the problem, reinforcing paths that offer the best trade-off across all three criteria. The PCB area is represented as a uniform grid, and ants navigate from the source pad to the target pad of each net by selecting adjacent cells probabilistically.

Experimental validation on a set of fundamental routing scenarios demonstrates that the algorithm consistently produces feasible and high-quality routing solutions, with statistically significant improvements over baseline methods across multiple

independent executions. The authors identify the extension of the method to multilayer boards and the integration of impedance-matching constraints as promising directions for future work.

1.5.2.5 Simulated Annealing for PCB Routing

Adzhar et al. [1] address the PCB routing problem in a rectangular mesh network using a hybrid approach that combines Dijkstra’s shortest path algorithm with a Simulated Annealing (SA) metaheuristic. Simulated Annealing is a probabilistic optimisation technique inspired by the physical annealing process in metallurgy, in which a material is heated and then slowly cooled to reach a low-energy crystalline state. In the algorithmic context, SA accepts worse solutions with a probability that decreases over time according to a temperature schedule, allowing the search to escape local optima during the early stages of optimisation.

In this work, the routing region is tessellated into a uniform grid of square cells, and Dijkstra’s algorithm is used to establish the shortest path for each net. The SA component is then applied to optimise the net ordering sequence — a critical decision that determines the quality of the overall routing solution, since nets routed earlier create obstacles for subsequent nets. By exploring alternative orderings and accepting probabilistic perturbations, the SA algorithm produces a net sequence that minimises congestion and maximises routing completion.

Results show that the hybrid SA approach yields a higher number of completed connections compared to purely sequential shortest-path routing, particularly in dense layouts where naive ordering strategies frequently result in blocked nets. A limitation of the approach is its sensitivity to the annealing schedule parameters, which require problem-specific tuning to achieve consistent performance.

1.5.3 Synthesis and Research Gap

Among the surveyed works, Ant Colony Optimisation appears as a recurring approach, having been applied in both single-objective [21] and multi-objective [3] formulations. Other metaheuristic families — such as evolutionary algorithms [10] and Simulated Annealing [1] — have also been explored, though to a lesser extent in the specific context of PCB routing. Furthermore, most existing contributions evaluate their proposed methods on synthetic or VLSI-derived benchmarks. When real PCB designs are considered [8], the approaches remain largely deterministic, leaving the application of metaheuristic methods to real-world routing instances comparatively unexplored.

These observations motivate the present work, which aims to contribute to this research direction by selecting, implementing, and evaluating a metaheuristic algorithm applied to a real PCB routing instance. The selection and justification of the chosen algorithm are presented in Chapter 2, where the theoretical foundations of the candidate approaches are examined and a motivated choice is made based on the specific requirements of the target problem.

1.6 Conclusion

This chapter has presented the theoretical foundations necessary for the study of the automatic PCB routing problem. Following a general introduction to PCBs, their structure, design process, and principal characteristics were described in order to contextualise this research.

The routing problem was then analysed: its formulation, its main constraints, and the commonly adopted optimisation criteria — minimisation of track length, reduction of the number of vias, and compliance with design rules — were identified. The NP-hard complexity of this problem justifies the use of advanced optimisation methods.

Finally, a review of existing work traced the evolution of routing techniques, from deterministic algorithms to metaheuristic-based approaches, while identifying the limitations of current solutions and the prospects for improvement.

Together, these elements constitute the theoretical framework of this study. The following chapter will present the selected metaheuristics, their adaptation to the problem at hand, and the architecture of the proposed system.

THEORETICAL FOUNDATIONS AND SELECTION OF ALGORITHMS

2.1 Introduction

The optimisation of large-scale combinatorial problems requires approaches capable of efficiently exploring a vast search space while producing good-quality solutions within reasonable time. To this end, this chapter presents the theoretical and methodological foundations of the optimisation approach adopted in this dissertation. It first introduces the general principles of metaheuristics, their main characteristics, and their relevance to combinatorial optimisation problems. The principal bio-inspired algorithms applicable to the problem under study are then presented and compared in order to justify the selection of the retained methods. The chapter also describes the A* pathfinding algorithm, used as the routing kernel in the proposed approach. Finally, it presents the adaptation of the selected metaheuristics to the PCB routing problem, particularly through the solution representation, the evaluation function, and the discretisation strategy implemented to effectively handle the net ordering problem.

2.2 Metaheuristic Optimisation — General Principles

2.2.1 Definition and Characteristics

Optimisation problems can broadly be divided into two categories: traditional techniques and metaheuristic techniques [2]. Traditional methods — including linear programming, gradient-based search, and exhaustive enumeration — offer rapid convergence and may yield accurate solutions when the objective function is continuously differentiable and the search space is relatively small. However, the majority of real-world engineering problems are non-linear and complex, and for such problems these methods are prone to converging to local optima rather than global ones. As the dimensionality of the search space increases, traditional techniques become susceptible to gradient disappearance and gradient explosion, rendering them impractical [2].

Metaheuristic algorithms are problem-independent optimisation strategies equipped with stochastic operators that allow them to explore large and complex search spaces without requiring gradient information or convexity of the objective function [2, 18]. The term *metaheuristic* combines the Greek prefix *meta* — meaning beyond or higher level — with *heuristic*, referring to the discovery of solutions by trial and error. A metaheuristic algorithm can therefore be defined as a higher-level strategy that directs and modifies subordinate heuristics to produce solutions beyond those typically generated by local search procedures alone [2].

The properties that characterise most metaheuristic algorithms are [2, 18]:

- Metaheuristics are strategies that guide the search process rather than following a fixed deterministic procedure.
- Their primary goal is to efficiently explore the search space in order to find near-optimal solutions within acceptable computation times.
- They are approximate and generally non-deterministic: different runs of the same algorithm may produce different solutions.
- They are not problem-specific and can be applied, with minimal modifications, to a wide variety of optimisation problems.
- They do not require the objective function to be differentiable, making them naturally suited to non-linear and discontinuous problem landscapes.

An important theoretical foundation underlying the comparison of metaheuristic algorithms is the No Free Lunch (NFL) theorem, which states that no single optimisation algorithm can solve all problems optimally [18]. An algorithm that performs well on one class of problems may perform poorly on another. This theorem has two practical implications for the present work: it motivates the careful selection of an algorithm that is well-suited to the specific characteristics of the PCB routing problem, and it justifies the comparative analysis conducted in Section 2.4.

2.2.2 Classification of Metaheuristic Algorithms

Metaheuristic algorithms can be classified according to several criteria [2, 18]. The most common taxonomies are reviewed below.

Nature-Inspired vs. Non-Nature-Inspired

Nature-inspired metaheuristics derive their mechanisms from natural processes. Examples include Genetic Algorithms (GA), inspired by biological evolution; Ant Colony Optimisation (ACO), inspired by the foraging behaviour of ant colonies; and Particle Swarm Optimisation (PSO), inspired by the collective movement of bird flocks. Non-nature-inspired metaheuristics, such as Tabu Search, are based on abstract computational concepts rather than natural analogies [2].

Population-Based vs. Trajectory-Based

Population-based metaheuristics maintain and evolve a set of candidate solutions simultaneously, enabling broader exploration of the search space. Examples include GA, PSO, and ACO. Trajectory-based metaheuristics operate on a single solution and iteratively improve it through local search. Examples include Simulated Annealing (SA), Tabu Search, and Variable Neighbourhood Search [2, 18].

Inspiration-Based Classification

A widely adopted taxonomy groups metaheuristics according to their underlying source of inspiration [2, 18]:

- **Evolutionary-based algorithms:** inspired by biological evolution — selection, crossover, and mutation. Representative algorithms include GA (Holland, 1975), Differential Evolution (DE, 1995), and Memetic Algorithms (MA, 1989).
- **Swarm intelligence-based algorithms:** inspired by the collective behaviour of social organisms. Representative algorithms include PSO (Kennedy and Eberhart, 1995), ACO (Dorigo et al., 1996), Artificial Bee Colony (ABC, 2005), Grey Wolf Optimizer (GWO, 2014), and the Whale Optimisation Algorithm (WOA, 2016).
- **Physics-based algorithms:** inspired by physical laws and principles such as thermodynamics, gravity, and electromagnetism. Representative algorithms include Simulated Annealing (SA, 1983), Gravitational Search Algorithm (GSA, 2009), and the Sine Cosine Algorithm (SCA, 2016).
- **Human behaviour-based algorithms:** inspired by human cognitive or social behaviours. Representative algorithms include Tabu Search (TS, 1986), Harmony Search (HS, 2001), and Teaching-Learning-Based Optimisation (TLBO, 2010).

A hierarchical overview of these principal metaheuristic families is presented in Figure 2.1. A comprehensive survey by Salgotra et al. [18] catalogues more than 300 metaheuristic algorithms proposed between 1975 and 2023, organised across five historical development phases, reflecting the exponential growth of the field over the past five decades. This diversity underscores the need for principled selection criteria, which are formalised in Section 2.4.

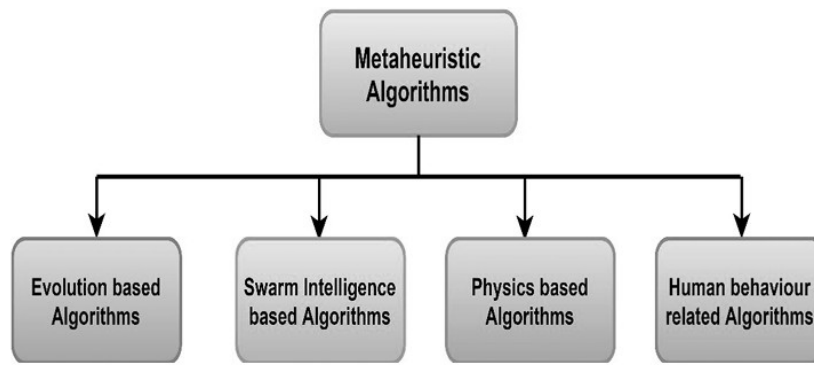


Figure 2.1: Hierarchical classification of metaheuristic algorithms based on their source of inspiration [18].

2.2.3 Key Concepts

Exploration and Exploitation

A fundamental trade-off in metaheuristic search is the balance between *exploration* and *exploitation* [2, 18]. Exploration refers to the ability of the algorithm to search diverse and previously unvisited regions of the solution space, reducing the risk of premature convergence to a local optimum. Exploitation refers to the ability to intensify the search around promising regions already identified, refining the quality of the current best solution. Effective metaheuristic algorithms must maintain a dynamic balance between these two complementary objectives throughout the search process. This balance is typically achieved through dedicated algorithmic operators: for example, crossover and selection in evolutionary algorithms, or pheromone evaporation and deposition in ACO.

Fitness Function

The fitness function — also referred to as the objective function or cost function — is the mathematical expression that quantifies the quality of a candidate solution with respect to the optimisation objectives [2]. It provides the feedback signal that guides the search: solutions with better fitness values are favoured during the selection or update process. The design of an appropriate fitness function is a critical step in the application of any metaheuristic algorithm, as it directly determines the direction and quality of the search. In the context of PCB routing, the fitness function must simultaneously capture the objectives of minimal wirelength, absence of design rule violations, and routability of all nets; its formal definition is provided in Section 2.7.4.

Convergence and Stopping Criteria

Metaheuristic algorithms iterate until a stopping criterion is met. Common criteria include a maximum number of iterations or function evaluations, a minimum improvement threshold over a given number of iterations, or a target fitness value [2]. Since metaheuristics do not guarantee global optimality — as formally established by the NFL theorem — the choice of stopping criterion directly influences the trade-off between solution quality and computation time. In practice, multiple independent

runs are performed and the best solution across runs is retained, providing a statistical estimate of the algorithm’s robustness.

2.2.4 Why Metaheuristics for PCB Routing?

As established in Section 1.4.6, the PCB routing problem is NP-hard, meaning that no known algorithm can solve all instances to proven optimality in polynomial time. Classical deterministic algorithms such as Lee’s maze router and A* (discussed in detail in Section 2.6) provide exact solutions for individual nets but are inherently sequential and unable to globally optimise the routing of all nets simultaneously, as demonstrated in Section 1.5.1.

Traditional optimisation methods fail on this class of problem for three fundamental reasons [2]: they require continuously differentiable objective functions, they are prone to trapping in local optima in large and irregular search spaces, and they do not scale to the high dimensionality inherent in multi-net routing. Metaheuristic algorithms directly address these limitations through the following key properties [2, 18]:

- **Global search capability:** population-based and stochastic metaheuristics explore the complete solution space in a coordinated manner, avoiding the sequential dependency that characterises classical routers.
- **Gradient-free operation:** metaheuristics do not require the objective function to be differentiable or convex, making them naturally suited to the constrained, discrete, and multi-objective nature of the PCB routing problem.
- **Flexibility and adaptability:** metaheuristic algorithms are not problem-specific and can be adapted to different routing objectives and constraint sets with minimal modifications, as evidenced by the variety of applications surveyed in Chapter 1 and the extensive catalogue documented by Salgotra et al. [18].
- **Robustness to noise and uncertainty:** metaheuristics are designed to handle noisy or imprecise objective functions, a property that is relevant when routing cost estimates depend on approximations of parasitic effects [2].

These properties collectively justify the adoption of metaheuristic optimisation as the methodological framework of the present work.

2.3 Selection of Candidate Algorithms

The general principles established in Section 2.2 confirm that the PCB routing problem requires an algorithm combining strong global exploration, robust multimodal handling, and low parameter overhead. Rather than adopting the first available algorithm, this work applies a structured screening process to a broad set of recent metaheuristic proposals before committing to a final selection.

Three criteria guided the screening:

1. **PCB routing suitability:** the algorithm’s exploration–exploitation mechanism must be compatible with a highly multimodal, discrete fitness landscape in which small permutation changes can cause large, abrupt fitness jumps.

-
2. **Parameter simplicity:** algorithms with few free parameters are preferred, as excessive parameter tuning across different board sizes would undermine the generality of the approach.
 3. **Discrete adaptability:** the algorithm's position update must be adaptable to a combinatorial permutation encoding with minimal distortion of its core search dynamics.

Applying these criteria to the recent metaheuristic literature, five algorithms emerged as strong candidates: the Whale Optimisation Algorithm (WOA), the Slime Mould Algorithm (SMA), the Harris Hawks Optimiser (HHO), the Salp Swarm Algorithm (SSA), and the Sine Cosine Algorithm (SCA). Each of these algorithms is population-based, gradient-free, and has demonstrated competitive performance on complex, multimodal benchmark problems. Each candidate is presented in detail in Sections 2.3.1–2.3.5 below. The results of the structured comparison across the three screening criteria and three additional routing-specific dimensions are reported in Section 2.4, and the final motivated selection is presented in Section 2.5.

2.3.1 Whale Optimisation Algorithm

2.3.1.1 Biological Inspiration

The Whale Optimisation Algorithm was proposed by Mirjalili and Lewis in 2016 [16]. It draws its inspiration from the hunting behaviour of humpback whales (*Megaptera novaeangliae*), one of the largest marine mammals on Earth. Humpback whales are distinguished by a remarkable foraging strategy known as *bubble-net feeding*: a whale dives below a school of prey fish, then ascends in a decreasing spiral while releasing a curtain of bubbles from its blowhole. This creates a cylindrical net of bubbles that herds the prey into a tight ball near the surface, where the whale engulfs them in a single lunge.

This behaviour exhibits two geometrically distinct patterns — a shrinking circular encirclement and a logarithmic spiral ascent — which are modelled separately in the WOA and combined probabilistically during the exploitation phase [16].

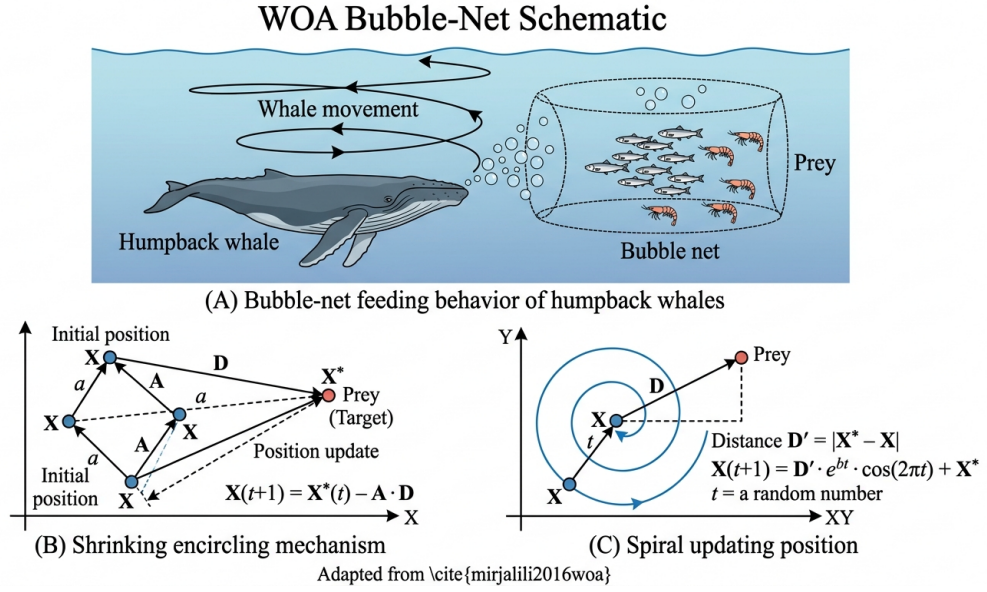


Figure 2.2: (A) Bubble-net feeding behaviour of humpback whales. (B) Shrinking encirclement mechanism. (C) Spiral position update. Adapted from [16].

2.3.1.2 Mathematical Model

The WOA models three behavioural phases: encircling prey (exploitation via shrinking encirclement), bubble-net attacking (exploitation via spiral update), and random search for prey (exploration) [16].

Phase 1 — Encircling Prey The best solution found so far, denoted X^* , is treated as the approximate location of the prey. All other agents update their positions to move toward X^* :

$$D = |C \cdot X^*(t) - X(t)| \quad (2.1)$$

$$X(t+1) = X^*(t) - A \cdot D \quad (2.2)$$

where $A = 2a \cdot r_1 - a$, $C = 2r_2$, r_1 and r_2 are random vectors drawn from $\mathcal{U}(0, 1)$, and a decreases linearly from 2 to 0 as iterations progress. The coefficient A controls the step size and direction: when $|A| < 1$, agents converge toward the best solution (exploitation); when $|A| \geq 1$, they are pushed away from the current best, enforcing exploration.

Phase 2 — Bubble-Net Attacking With probability $p \sim \mathcal{U}(0, 1)$, an agent applies either the shrinking encirclement (if $p < 0.5$) or a spiral update (if $p \geq 0.5$):

$$X(t+1) = \begin{cases} X^*(t) - A \cdot D & \text{if } p < 0.5 \\ D' \cdot e^{bl} \cdot \cos(2\pi l) + X^*(t) & \text{if } p \geq 0.5 \end{cases} \quad (2.3)$$

where $D' = |X^*(t) - X(t)|$, b is a constant defining the shape of the logarithmic spiral, and $l \sim \mathcal{U}(-1, 1)$.

Phase 3 — Random Search When $|A| \geq 1$, each agent moves toward a randomly selected member of the population $\mathbf{X}_{\text{rand}}$ rather than toward $\mathbf{X}^*$, enforcing global exploration:

$$\mathbf{D} = |\mathbf{C} \cdot \mathbf{X}_{\text{rand}}(t) - \mathbf{X}(t)| \quad (2.4)$$

$$\mathbf{X}(t + 1) = \mathbf{X}_{\text{rand}}(t) - \mathbf{A} \cdot \mathbf{D} \quad (2.5)$$

The linear decay of a from 2 to 0 drives an automatic, parameter-free transition from exploration in early iterations to exploitation in late iterations [16].

2.3.1.3 Pseudocode

Algorithm 1: Whale Optimisation Algorithm

Input: Population size N , maximum iterations T_{max} , spiral constant b

Output: Best solution $\mathbf{X}^*$

```

1 Initialise population  $\{\mathbf{X}_i\}_{i=1}^N$  randomly within  $[\mathbf{LB}, \mathbf{UB}]$ ;
2 Evaluate fitness  $f(\mathbf{X}_i)$  for all  $i$ ;
3  $\mathbf{X}^* \leftarrow \arg \min_i f(\mathbf{X}_i)$ ;
4 for  $t = 1$  to  $T_{\text{max}}$  do
5   for each agent  $\mathbf{X}_i$  do
6      $a \leftarrow 2 - 2 \cdot (t/T_{\text{max}})$ ;
7     Draw  $\mathbf{A}, \mathbf{C}, l, p$  at random;
8     if  $p < 0.5$  then
9       if  $|A| < 1$  then
10         $\mathbf{D} \leftarrow |\mathbf{C} \cdot \mathbf{X}^* - \mathbf{X}_i|$ ;
11         $\mathbf{X}_i \leftarrow \mathbf{X}^* - \mathbf{A} \cdot \mathbf{D}$ ;           // Shrinking encirclement
12      else
13        Select  $\mathbf{X}_{\text{rand}}$  randomly from population;
14         $\mathbf{D} \leftarrow |\mathbf{C} \cdot \mathbf{X}_{\text{rand}} - \mathbf{X}_i|$ ;
15         $\mathbf{X}_i \leftarrow \mathbf{X}_{\text{rand}} - \mathbf{A} \cdot \mathbf{D}$ ;           // Random search
16      else
17         $\mathbf{D}' \leftarrow |\mathbf{X}^* - \mathbf{X}_i|$ ;
18         $\mathbf{X}_i \leftarrow \mathbf{D}' \cdot e^{bl} \cdot \cos(2\pi l) + \mathbf{X}^*$ ;           // Spiral update
19      Clip  $\mathbf{X}_i$  to  $[\mathbf{LB}, \mathbf{UB}]$ ;
20   Evaluate fitness for all agents;
21   if  $\min_i f(\mathbf{X}_i) < f(\mathbf{X}^*)$  then
22      $\mathbf{X}^* \leftarrow \arg \min_i f(\mathbf{X}_i)$ ;
23 return  $\mathbf{X}^*$ ;

```

2.3.1.4 Key Parameters

The time complexity of WOA per iteration is $O(N \cdot D)$, where D is the problem dimension. WOA has only one true free parameter beyond N and T_{max} — the spiral constant b — making it one of the most parameter-lean algorithms in the swarm intelligence family [16].

Table 2.1: WOA key parameters and their roles.

Parameter	Role	Typical value
N	Population size	30 – 50
$T_{\max}$	Maximum iterations	100 – 500
b	Logarithmic spiral shape (fixed)	1
a	Decay coefficient (automatic)	$2 \rightarrow 0$

2.3.1.5 Strengths, Limitations, and Relevance to PCB Routing

Strengths. The automatic decay of a provides a natural, tuning-free exploration-to-exploitation transition. The dual exploitation mechanism — shrinking encirclement combined with the spiral update — explores the fitness landscape at two geometric scales simultaneously. The low parameter count reduces calibration effort across different board sizes and net counts.

Limitations. WOA lacks an explicit mechanism to escape stagnation if the entire population converges prematurely. The logarithmic spiral is defined for continuous spaces and requires a discretisation step for combinatorial problems, as formalised in Section 2.7.1.

Relevance to PCB routing. The spiral movement is geometrically analogous to the iterative tightening of a routing path around an obstacle — a natural fit for the refinement phase of net routing. The exploration phase ensures that different net orderings are broadly sampled before convergence. WOA’s balance of simplicity and performance makes it the primary algorithm selected in this work, as justified in Section 2.5.

2.3.2 Slime Mould Algorithm

2.3.2.1 Biological Inspiration

The Slime Mould Algorithm was proposed by Li et al. in 2020 [11]. It models the foraging behaviour of *Physarum polycephalum*, a plasmodial slime mould found on forest floors. Despite having no nervous system, this organism demonstrates a remarkable ability to construct efficient network structures when searching for food: it extends tube-like pseudopods toward food sources and retracts them from unpromising directions, adapting the thickness of each tube in proportion to the quality of the food source it connects. This oscillatory extension-retraction process was shown to reproduce the topology of the Tokyo rail network when nutrient sources were placed at the positions of suburban stations [11].

This biological behaviour provides a direct analogy to routing optimisation: the organism effectively solves a shortest-path problem through a purely physical, decentralised process — precisely the kind of problem posed by PCB net routing.

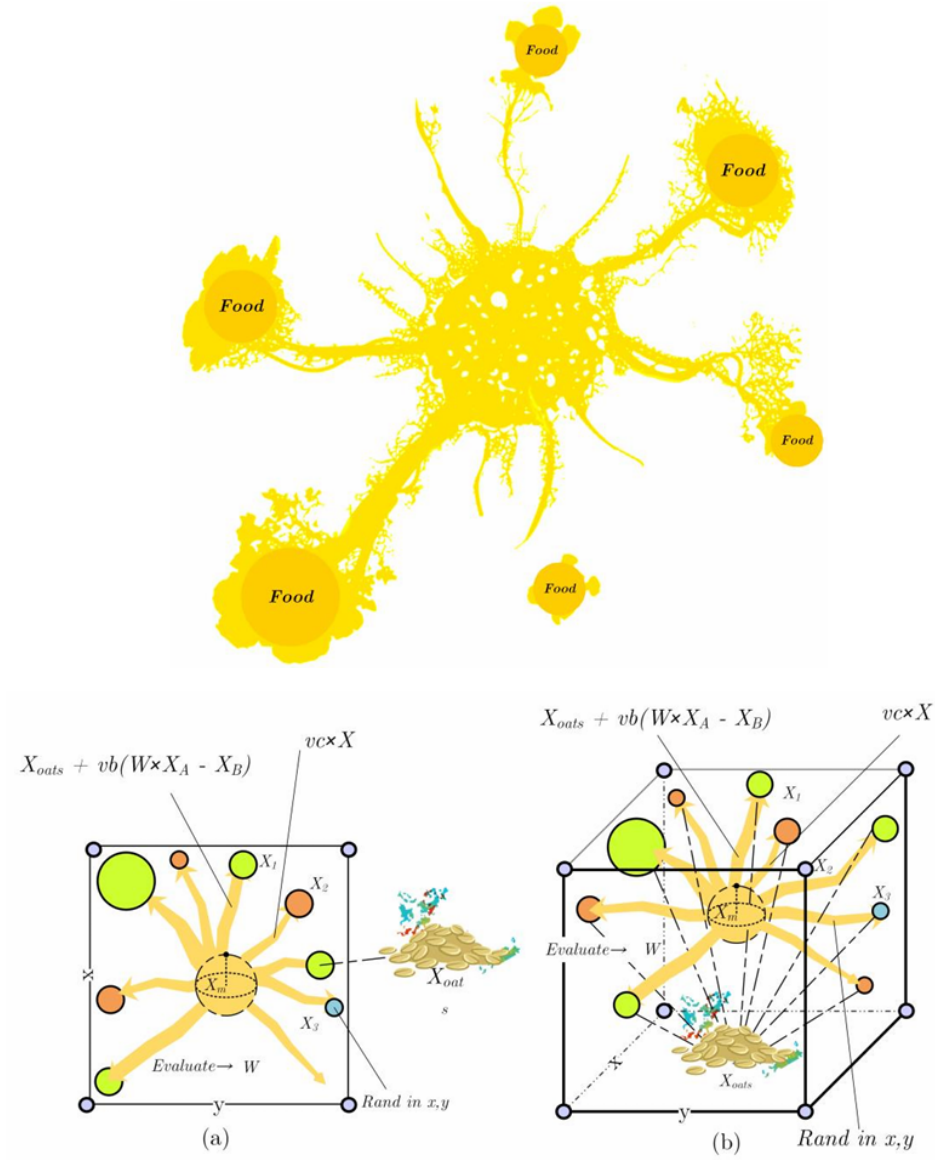


Figure 2.3: *Physarum polycephalum* extending pseudopods toward food sources (top), and the SMA weight distribution mechanism in 2D and 3D (bottom): agents closer to the best food source receive stronger weights $W > 1$ (thickened tubes) while weaker agents receive $W < 1$ (thinned tubes). Adapted from [11].

2.3.2.2 Mathematical Model

The SMA models three phases: approaching the food source, wrapping around it, and grabbing it. The key innovation is a rank-based adaptive weight W that assigns influence to each agent based on its relative fitness rank, directly simulating the biological thickening and thinning of slime mould tubes [11].

Weight Update After sorting the population by fitness, the weight of agent i is:

$$W(i) = \begin{cases} 1 + r \cdot \log\left(\frac{bF - S(i)}{bF - wF} + 1\right) & \text{if } i \text{ is in the upper half} \\ 1 - r \cdot \log\left(\frac{bF - S(i)}{bF - wF} + 1\right) & \text{if } i \text{ is in the lower half} \end{cases} \quad (2.6)$$

where bF and wF are the best and worst fitness values in the current population, $S(i)$ is the fitness of agent i , and $r \sim \mathcal{U}(0, 1)$. Agents with fitness close to bF receive $W > 1$ (thickened tubes, strong pull toward food); agents close to wF receive $W < 1$ (thinned tubes, weak pull) [11].

Position Update

$$\mathbf{X}(t+1) = \begin{cases} \mathbf{X}_b(t) + v_b \cdot (W \cdot \mathbf{X}_A(t) - \mathbf{X}_B(t)) & \text{if } r < z \\ v_c \cdot \mathbf{X}(t) & \text{otherwise} \end{cases} \quad (2.7)$$

where $\mathbf{X}_b$ is the position of the best agent, $\mathbf{X}_A$ and $\mathbf{X}_B$ are two randomly selected agents from the population, $z = 0.03$ is a fixed threshold controlling the random-walk probability, v_b oscillates in $[-a, a]$ with a decreasing over iterations, and v_c decreases linearly from 1 to 0. The oscillation of v_b mimics the contraction and relaxation of the slime mould's cytoplasm, enabling repeated escape from local traps [11].

2.3.2.3 Pseudocode

Algorithm 2: Slime Mould Algorithm

Input: Population size N , maximum iterations $T_{\max}$
Output: Best solution $\mathbf{X}_b$

- 1 Initialise population $\{\mathbf{X}_i\}_{i=1}^N$ randomly within $[\mathbf{LB}, \mathbf{UB}]$;
- 2 Evaluate fitness $f(\mathbf{X}_i)$ for all i ;
- 3 Record best agent $\mathbf{X}_b$;
- 4 **for** $t = 1$ **to** $T_{\max}$ **do**
- 5 Sort population by fitness; record bF, wF ;
- 6 Compute weight $W(i)$ for each agent via Eq. (2.6);
- 7 $a \leftarrow \operatorname{arctanh}\left(-\frac{t}{T_{\max}} + 1\right)$;
- 8 **for each agent** $\mathbf{X}_i$ **do**
- 9 $v_b \sim \mathcal{U}(-a, a)$;
- 10 $v_c \leftarrow 1 - t/T_{\max}$;
- 11 Draw $r \sim \mathcal{U}(0, 1)$;
- 12 **if** $r < z$ **then**
- 13 Select $\mathbf{X}_A, \mathbf{X}_B$ randomly from population;
- 14 $\mathbf{X}_i \leftarrow \mathbf{X}_b + v_b \cdot (W(i) \cdot \mathbf{X}_A - \mathbf{X}_B)$; // Approach food
- 15 **else**
- 16 $\mathbf{X}_i \leftarrow v_c \cdot \mathbf{X}_i$; // Random walk
- 17 Clip $\mathbf{X}_i$ to $[\mathbf{LB}, \mathbf{UB}]$;
- 18 Evaluate fitness for all agents;
- 19 **if** $\min_i f(\mathbf{X}_i) < f(\mathbf{X}_b)$ **then**
- 20 $\mathbf{X}_b \leftarrow \arg \min_i f(\mathbf{X}_i)$;
- 21 **return** $\mathbf{X}_b$;

2.3.2.4 Key Parameters

Table 2.2: SMA key parameters and their roles.

Parameter	Role	Value
N	Population size	30 – 50
$T_{\max}$	Maximum iterations	100 – 500
z	Random-walk probability threshold (fixed)	0.03

SMA has no free parameters beyond N and $T_{\max}$ — z is a biological constant derived from the original paper [11]. Time complexity per iteration is $O(N \cdot D)$, identical to WOA. The sorting step adds $O(N \log N)$ per iteration, which is negligible for typical population sizes.

2.3.2.5 Strengths, Limitations, and Relevance to PCB Routing

Strengths. The rank-based weight W provides automatic, fitness-proportional emphasis, making SMA inherently robust to noisy and multimodal fitness landscapes. The oscillatory v_b mechanism enables repeated escape from local optima without

requiring manual restart strategies. Zero free parameters make SMA the easiest algorithm in this set to deploy.

Limitations. The sorting step introduces a dependency between agents that can slow convergence on very large populations. Like WOA, SMA requires discretisation for combinatorial domains, as formalised in Section 2.7.1.

Relevance to PCB routing. SMA's resistance to noisy fitness landscapes is particularly valuable in the PCB routing context, where small changes to a routing path can cause abrupt large jumps in the design rule violation penalty term. Its biologically validated shortest-path behaviour makes it a natural secondary candidate and the benchmark algorithm against which WOA is evaluated in Chapter 3, as justified in Section 2.5.

2.3.3 Harris Hawks Optimiser

2.3.3.1 Biological Inspiration

The Harris Hawks Optimiser was proposed by Heidari et al. in 2019 [9]. It models the cooperative hunting strategy of Harris hawks (*Parabuteo unicinctus*), a North American raptor that hunts in groups of two to six individuals. Unlike solitary raptors, Harris hawks coordinate their attacks: they take turns diving at prey, flushing it from cover, intercepting escape attempts, and executing a final coordinated strike. The prey's remaining escape energy determines which of several escalating attack strategies the group employs. This cooperative, energy-driven escalation is directly encoded in the HHO through a prey escape energy parameter E that decays over iterations, triggering a transition from free roaming to progressively more aggressive attack patterns.

2.3.3.2 Mathematical Model

Exploration Phase Hawks perch randomly and survey the landscape, selecting perch locations either near a randomly chosen hawk or near the prey's estimated position:

$$\mathbf{X}(t+1) = \begin{cases} \mathbf{X}_{\text{rand}}(t) - r_1 |\mathbf{X}_{\text{rand}}(t) - 2r_2 \mathbf{X}(t)| & \text{if } q \geq 0.5 \\ (\mathbf{X}_{\text{rabbit}}(t) - \mathbf{X}_{\text{mean}}(t)) - r_3 (\text{LB} + r_4 (\text{UB} - \text{LB})) & \text{if } q < 0.5 \end{cases} \quad (2.8)$$

where $\mathbf{X}_{\text{rabbit}}$ is the prey (best solution found so far), $\mathbf{X}_{\text{mean}}$ is the mean position of all hawks, and $r_1, r_2, r_3, r_4, q \sim \mathcal{U}(0, 1)$.

Prey Escape Energy The prey's escape energy decays as:

$$E = 2E_0 \left(1 - \frac{t}{T_{\text{max}}}\right) \quad (2.9)$$

where $E_0 \sim \mathcal{U}(-1, 1)$ is the initial escape energy drawn freshly at each iteration. When $|E| \geq 1$ the prey escapes easily and exploration dominates; when $|E| < 1$ the prey is weakening and exploitation takes over.

Exploitation Phase — Soft Besiege ($|E| \geq 0.5$)

The hawks encircle the prey with a soft, adjustable perimeter:

$$\mathbf{X}(t+1) = \Delta\mathbf{X}(t) - E \left| J \mathbf{X}_{\text{rabbit}}(t) - \mathbf{X}(t) \right| \quad (2.10)$$

where $\Delta\mathbf{X}(t) = \mathbf{X}_{\text{rabbit}}(t) - \mathbf{X}(t)$ and $J \sim \mathcal{U}(0, 2)$ is a random jump magnitude.

Exploitation Phase — Hard Besiege with Lévy Flight ($|E| < 0.5$)

The hawks make rapid, unpredictable dives modelled by Lévy flight, a heavy-tailed random walk that enables large, rare jumps alongside many small steps:

$$\mathbf{X}(t+1) = \mathbf{X}_{\text{rabbit}}(t) - E |\Delta\mathbf{X}(t)| + \text{LF}(D) \quad (2.11)$$

where $\text{LF}(D)$ denotes a Lévy flight step of dimension D , computed as:

$$\text{LF}(D) = 0.01 \times \frac{u \cdot \sigma}{|v|^{1/\beta}}, \quad \sigma = \left(\frac{\Gamma(1+\beta) \cdot \sin(\pi\beta/2)}{\Gamma(\frac{1+\beta}{2}) \cdot \beta \cdot 2^{(\beta-1)/2}} \right)^{1/\beta} \quad (2.12)$$

with $\beta = 1.5$ and $u, v \sim \mathcal{N}(0, 1)$.

2.3.3.3 Pseudocode

Algorithm 3: Harris Hawks Optimiser (HHO)

Input: Population size N , maximum iterations $T_{\max}$
Output: Best solution $\mathbf{X}_{\text{rabbit}}$

```

1 Initialise population  $\{\mathbf{X}_i\}_{i=1}^N$  randomly within [LB, UB];
2 Evaluate  $f(\mathbf{X}_i)$  for all  $i$ ;
3  $\mathbf{X}_{\text{rabbit}} \leftarrow \arg \min_i f(\mathbf{X}_i)$ ;
4 for  $t = 1$  to  $T_{\max}$  do
5   Draw  $E_0 \sim \mathcal{U}(-1, 1)$ ;
6    $E \leftarrow 2 E_0 (1 - t/T_{\max})$ ;
7   for each hawk  $\mathbf{X}_i$  do
8     if  $|E| \geq 1$  then
9       Draw  $q \sim \mathcal{U}(0, 1)$ ;
10      Update  $\mathbf{X}_i$  via Equation 2.8;
11    else
12      if  $|E| \geq 0.5$  then
13        Draw  $r \sim \mathcal{U}(0, 1)$ ;
14        if  $r \geq 0.5$  then
15           $\mathbf{X}_i \leftarrow \Delta \mathbf{X} - E |J \mathbf{X}_{\text{rabbit}} - \mathbf{X}_i|$ ;
16        else
17           $\mathbf{X}_i \leftarrow \mathbf{X}_{\text{rabbit}} - E |\Delta \mathbf{X}|$ ;
18      else
19        Draw  $r \sim \mathcal{U}(0, 1)$ ;
20        if  $r \geq 0.5$  then
21           $\mathbf{X}_i \leftarrow \mathbf{X}_{\text{rabbit}} - E |\Delta \mathbf{X}| + \text{LF}(D)$ ;
22        else
23           $\mathbf{X}_i \leftarrow \mathbf{X}_{\text{rabbit}} - E |\Delta \mathbf{X}|$ ;
24    Clip  $\mathbf{X}_i$  to [LB, UB];
25  Evaluate  $f(\mathbf{X}_i)$  for all  $i$ ;
26  Update  $\mathbf{X}_{\text{rabbit}}$  if improved;
27 return  $\mathbf{X}_{\text{rabbit}}$ ;

```

2.3.3.4 Key Parameters

Table 2.3: HHO key parameters

Parameter	Role	Typical Value
N	Population size	30–50
$T_{\max}$	Maximum iterations	100–500
E_0	Initial prey escape energy	Random $\mathcal{U}(-1, 1)$ per iteration
β	Lévy flight stability index	1.5 (fixed)

The time complexity of HHO per iteration is $O(N \cdot D)$, where D is the problem dimension. The Lévy flight computation adds a constant overhead per agent but

does not change the asymptotic complexity.

2.3.3.5 Strengths, Limitations, and Relevance to PCB Routing

Strengths. Lévy flight provides occasional very large jumps, enabling escape from deep local optima that shorter-step algorithms cannot leave. The four-stage exploitation escalation provides fine-grained control over convergence behaviour as the prey weakens.

Limitations. HHO has the most complex update logic of the five candidates, complicating the integration of custom repair operators needed for PCB routing constraint enforcement. The Lévy flight step can produce position values far outside the feasible domain, requiring robust boundary handling.

Relevance to PCB routing. HHO's Lévy flight is theoretically attractive for routing on irregular grids where large path jumps around obstacles are necessary. However, its implementation complexity and less predictable convergence behaviour make it a third-tier candidate in this work.

2.3.4 Salp Swarm Algorithm

2.3.4.1 Biological Inspiration

The Salp Swarm Algorithm was proposed by Mirjalili et al. in 2017 [15]. It is inspired by the swarming behaviour of salps (*Thaliacea*), gelatinous barrel-shaped marine organisms that are among the fastest-reproducing multicellular animals on Earth. Salps form long, chain-like colonies and navigate by jet propulsion — drawing water in through one opening and expelling it through another. The chain structure creates a remarkable division of labour: the leading salp (the *leader*) navigates toward food sources, while the following salps (the *followers*) are propelled by the collective movement of those ahead of them, forming a biological conveyor belt of motion information.

2.3.4.2 Mathematical Model

The SSA divides the population into one leader and $N - 1$ followers.

Leader Update The leader navigates toward the food source $\mathbf{F}$ (the best solution found so far):

$$X_j^1 = \begin{cases} F_j + c_1((ub_j - lb_j)c_2 + lb_j) & \text{if } c_3 \geq 0.5 \\ F_j - c_1((ub_j - lb_j)c_2 + lb_j) & \text{if } c_3 < 0.5 \end{cases} \quad (2.13)$$

where F_j is the j -th dimension of the food source, ub_j and lb_j are the upper and lower bounds for dimension j , $c_2, c_3 \sim \mathcal{U}(0, 1)$, and c_1 is the key adaptive coefficient:

$$c_1 = 2e^{-(4t/T_{\max})^2} \quad (2.14)$$

The coefficient c_1 decreases exponentially with iterations, causing the leader's step size to shrink and the search to shift from exploration to exploitation automatically.

Follower Update Each follower $i \geq 2$ uses Newton’s law of motion, averaging its position with the salp immediately ahead of it:

$$X_j^i = \frac{1}{2}(X_j^i + X_j^{i-1}) \quad (2.15)$$

This simple averaging rule creates a smooth propagation of position information from the leader to the back of the chain.

2.3.4.3 Pseudocode

Algorithm 4: Salp Swarm Algorithm (SSA)

Input: Population size N , maximum iterations $T_{\max}$

Output: Best solution $\mathbf{F}$

```

1 Initialise population  $\{\mathbf{X}_i\}_{i=1}^N$  randomly within  $[\text{LB}, \text{UB}]$ ;
2 Evaluate  $f(\mathbf{X}_i)$  for all  $i$ ;
3  $\mathbf{F} \leftarrow \arg \min_i f(\mathbf{X}_i)$ ;
4 for  $t = 1$  to  $T_{\max}$  do
5    $c_1 \leftarrow 2 e^{-(4t/T_{\max})^2}$ ;
6   for each dimension  $j = 1$  to  $D$  do
7     Draw  $c_2, c_3 \sim \mathcal{U}(0, 1)$ ;
8     if  $c_3 \geq 0.5$  then
9        $X_j^1 \leftarrow F_j + c_1((ub_j - lb_j)c_2 + lb_j)$ ;
10    else
11       $X_j^1 \leftarrow F_j - c_1((ub_j - lb_j)c_2 + lb_j)$ ;
12    for  $i = 2$  to  $N$  do
13       $\mathbf{X}^i \leftarrow \frac{1}{2}(\mathbf{X}^i + \mathbf{X}^{i-1})$ ;
14    Clip all  $\mathbf{X}_i$  to  $[\text{LB}, \text{UB}]$ ;
15    Evaluate  $f(\mathbf{X}_i)$  for all  $i$ ;
16    Update  $\mathbf{F}$  if improved;
17 return  $\mathbf{F}$ ;
```

2.3.4.4 Key Parameters

Table 2.4: SSA key parameters

Parameter	Role	Typical Value
N	Population size	30–50
$T_{\max}$	Maximum iterations	100–500

SSA has no free parameters beyond N and $T_{\max}$. The adaptive coefficient c_1 is computed deterministically from the iteration counter. The time complexity per iteration is $O(N \cdot D)$.

2.3.4.5 Strengths, Limitations, and Relevance to PCB Routing

Strengths. The chain structure creates an inherent sequential information flow from leader to followers, mirroring the sequential nature of net ordering in PCB routing. Zero free parameters make SSA straightforward to deploy across different board sizes without re-tuning. The exponential decay of c_1 provides a smooth, automatic exploration-to-exploitation transition.

Limitations. With only one leader driving the search direction, SSA is vulnerable to premature convergence if the leader settles in a local optimum early in the run. The follower averaging rule is a weak exploitation mechanism compared to WOA's logarithmic spiral or HHO's Lévy flight.

Relevance to PCB routing. SSA's chain structure is conceptually well-suited to the sequential routing problem, but its limited exploitation power places it as a fourth-tier candidate in this work.

2.3.5 Sine Cosine Algorithm

2.3.5.1 Mathematical Foundation

The Sine Cosine Algorithm was proposed by Mirjalili in 2016 [14]. Unlike the four biologically inspired algorithms presented above, SCA is grounded in the oscillatory properties of the trigonometric sine and cosine functions. The core observation is that sine and cosine oscillate between -1 and $+1$: when the coefficient of this oscillation is greater than 1, the update moves an agent *away* from a reference point (exploration); when it is less than 1, the update moves an agent *toward* the reference point (exploitation). By making this coefficient decay over time, SCA achieves an automatic exploration-to-exploitation transition without any biological metaphor.

2.3.5.2 Mathematical Model

Position Update Rule

$$X_i^{t+1} = \begin{cases} X_i^t + r_1 \cdot \sin(r_2) \cdot |r_3 P_i^t - X_i^t| & \text{if } r_4 < 0.5 \\ X_i^t + r_1 \cdot \cos(r_2) \cdot |r_3 P_i^t - X_i^t| & \text{if } r_4 \geq 0.5 \end{cases} \quad (2.16)$$

where:

- P_i^t is the best solution found so far (the destination);
- $r_1 = a - t \cdot (a/T_{\max})$ is the step-size control coefficient, decaying linearly from $a = 2$ to 0;
- $r_2 \sim \mathcal{U}(0, 2\pi)$ controls the direction and magnitude of the oscillation;
- $r_3 \sim \mathcal{U}(0, 2)$ is a random weight applied to the destination;
- $r_4 \sim \mathcal{U}(0, 1)$ selects between the sine and cosine variants.

The exploration–exploitation balance is directly governed by r_1 : when $|r_1 \cdot \sin(r_2)| > 1$ the agent moves beyond P_i^t (exploration); when $|r_1 \cdot \sin(r_2)| < 1$ the agent moves toward P_i^t (exploitation). Since r_1 decreases linearly from a to 0, the algorithm is guaranteed to transition from predominantly exploratory to predominantly exploitative behaviour over the course of the run.

2.3.5.3 Pseudocode

Algorithm 5: Sine Cosine Algorithm (SCA)

Input: Population size N , maximum iterations $T_{\max}$, $a = 2$
Output: Best solution $\mathbf{P}^*$

- 1 Initialise population $\{\mathbf{X}_i\}_{i=1}^N$ randomly within $[\text{LB}, \text{UB}]$;
- 2 Evaluate $f(\mathbf{X}_i)$ for all i ;
- 3 $\mathbf{P}^* \leftarrow \arg \min_i f(\mathbf{X}_i)$;
- 4 **for** $t = 1$ **to** $T_{\max}$ **do**
- 5 $r_1 \leftarrow a - t \cdot (a/T_{\max})$;
- 6 **for each agent** $\mathbf{X}_i$ **do**
- 7 **for each dimension** j **do**
- 8 Draw $r_2 \sim \mathcal{U}(0, 2\pi)$;;
- 9 $r_3 \sim \mathcal{U}(0, 2)$;;
- 10 $r_4 \sim \mathcal{U}(0, 1)$;
- 11 **if** $r_4 < 0.5$ **then**
- 12 $X_{i,j}^{t+1} \leftarrow X_{i,j}^t + r_1 \cdot \sin(r_2) \cdot |r_3 P_j^* - X_{i,j}^t|$;
- 13 **else**
- 14 $X_{i,j}^{t+1} \leftarrow X_{i,j}^t + r_1 \cdot \cos(r_2) \cdot |r_3 P_j^* - X_{i,j}^t|$;
- 15 Clip $\mathbf{X}_i$ to $[\text{LB}, \text{UB}]$;
- 16 Evaluate $f(\mathbf{X}_i)$ for all i ;
- 17 Update $\mathbf{P}^*$ if improved;
- 18 **return** $\mathbf{P}^*$;

2.3.5.4 Key Parameters

Table 2.5: SCA key parameters

Parameter	Role	Typical Value
N	Population size	30–50
$T_{\max}$	Maximum iterations	100–500
a	Decay start value	2 (fixed)

The time complexity per iteration is $O(N \cdot D)$. The parameter a is fixed at 2 in the original formulation and requires no calibration.

2.3.5.5 Strengths, Limitations, and Relevance to PCB Routing

Strengths. Mathematically transparent: the exploration–exploitation balance is directly controlled by a single decaying coefficient r_1 . Smooth and predictable convergence is useful when early termination is required. Minimal implementation overhead and low parameter count.

Limitations. All agents converge toward the single global best $\mathbf{P}^*$, which can cause rapid loss of diversity and premature convergence on multimodal problems.

There is no mechanism analogous to WOA’s spiral or HHO’s Lévy flight for escaping deep local optima.

Relevance to PCB routing. SCA’s predictable convergence and minimal implementation overhead are useful properties, but its tendency toward premature convergence on highly multimodal landscapes — exactly the kind encountered in PCB routing — places it as the fifth-tier candidate in this work.

2.4 Comparative Analysis of Candidate Algorithms

2.4.1 Criteria for Comparison

The five algorithms are evaluated against six criteria, each directly motivated by the structural properties of the PCB routing problem identified in Section 2.2.4:

1. **Exploration strength:** ability to broadly sample the permutation space and avoid premature convergence.
2. **Exploitation strength:** ability to refine good routing solutions in their neighbourhood.
3. **Multimodal robustness:** resistance to the numerous local optima present in the PCB routing fitness landscape.
4. **Convergence speed:** typical rate of fitness improvement per iteration.
5. **Discrete adaptability:** ease of adapting the continuous position update to a discrete permutation encoding.
6. **Implementation complexity:** effort required to integrate custom repair operators for constraint enforcement.

2.4.2 Comparative Summary

Having examined each candidate algorithm individually, it is useful to consolidate their properties into a unified comparison before proceeding to the final selection. The five algorithms were evaluated against six comparison criteria: exploration strength, exploitation strength, multimodal robustness, convergence speed, discrete adaptability, and implementation complexity. Table 2.6 summarises the results of this evaluation, including descriptive metadata and an overall PCB routing suitability rating for each algorithm. Figure 2.4 provides a visual overview of the relative strengths and weaknesses across the six comparison criteria.

Table 2.6: Comparative analysis of the five candidate algorithms against PCB routing criteria

Criterion	WOA	SMA	HHO	SSA	SCA
Year proposed	2016	2020	2019	2017	2016
Category	Swarm	Swarm	Swarm	Swarm	Physics
Exploration strength	High	High	Very High	Moderate	Moderate
Exploitation strength	High	High	High	Moderate	High
Multimodal robustness	High	Very High	High	Moderate	Low
Convergence speed	Fast	Fast	Moderate	Moderate	Fast
Discrete adaptability	Moderate	Moderate	Moderate	Easy	Easy
Implementation complexity	Low	Low	High	Low	Low
PCB routing suitability	★★★★★	★★★★★	★★★★	★★★	★★★

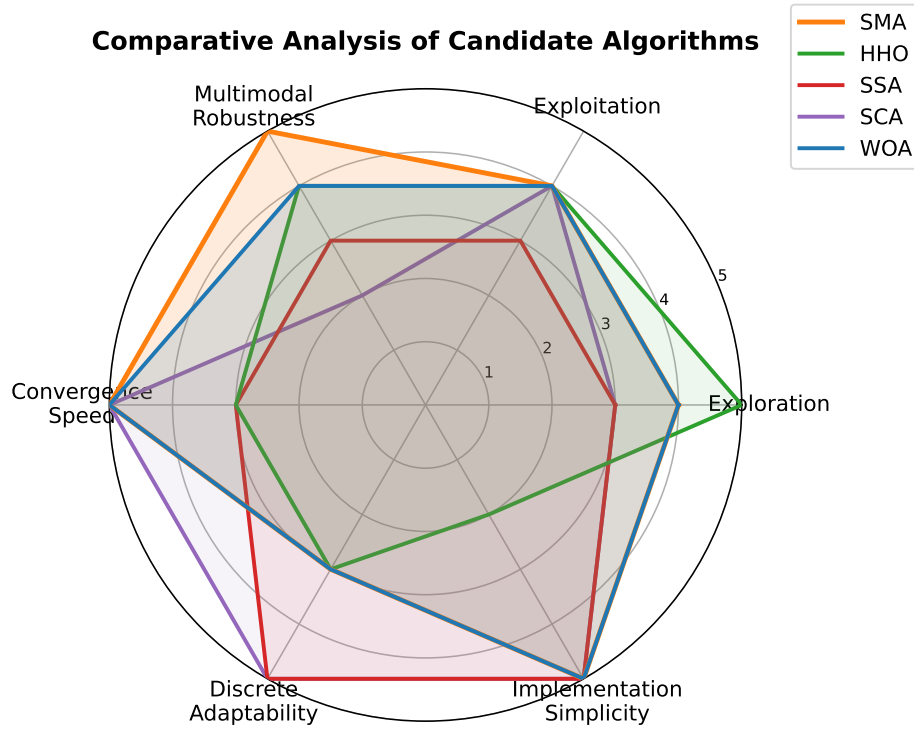


Figure 2.4: Radar chart of the five candidate algorithms across the six comparison criteria.

2.4.3 Discussion

The analysis reveals a clear two-tier structure among the five candidates.

Tier 1 — WOA and SMA both score Very High or High across all relevant criteria. WOA’s dual exploitation mechanism — shrinking encirclement combined with a logarithmic spiral — and its automatic exploration-to-exploitation transition make it particularly well suited to the iterative path-refinement nature of PCB routing. SMA’s rank-based oscillatory weight provides superior multimodal robustness, which is the single most critical property for a fitness landscape as irregular as

PCB routing. Both algorithms carry low implementation complexity and zero or near-zero free parameter overhead.

Tier 2 — HHO offers the strongest global exploration through its Lévy flight mechanism, which is theoretically appealing for large, irregular routing grids. However, its four-stage exploitation logic significantly increases implementation complexity, and the Lévy flight step requires careful boundary handling to maintain valid permutation encodings.

Tier 3 — SSA and SCA are competent algorithms on continuous benchmarks but offer limited advantages for the PCB routing problem specifically. SSA’s single-leader architecture is vulnerable to premature convergence, and SCA’s sinusoidal updates show poor multimodal robustness — a critical weakness given the highly multimodal nature of the routing fitness landscape.

2.5 Algorithm Selection and Justification

Based on the comparative analysis presented in Section 2.4, this work selects the **Whale Optimisation Algorithm (WOA)** as the primary algorithm for implementation and evaluation in Chapter 3. The Slime Mould Algorithm (SMA) is retained as a secondary benchmark to enable a meaningful comparison of two qualitatively different top-tier strategies.

The selection of WOA as the primary algorithm is justified on five grounds:

1. **Exploration–exploitation balance.** WOA’s adaptive parameter a , which decays linearly from 2 to 0 over the run, drives an automatic, problem-independent transition from global search to local refinement. This requires no manual scheduling and generalises across board sizes and net counts.
2. **Geometric analogy to path refinement.** The logarithmic spiral update directly mirrors the iterative tightening of a routing path around an obstacle. As the algorithm converges, the spiral radius shrinks, causing agents to circle more tightly around the current best routing solution — a behaviour that is geometrically consistent with incremental path improvement.
3. **Low parameter overhead.** WOA’s only free parameter beyond N and $T_{\max}$ is the spiral shape constant b , typically fixed at 1, minimising calibration effort and making the algorithm robust across different PCB instances without re-tuning.
4. **Established Python implementation.** Open-source Python reference code for WOA is documented in Salgotra et al. [18], facilitating rapid prototyping, reproducibility, and fair comparison.
5. **Demonstrated combinatorial performance.** WOA has been successfully applied to discrete and combinatorial optimisation problems in the literature, including job scheduling, feature selection, and path-planning tasks with structural similarities to PCB routing [16].

SMA is selected as the secondary benchmark rather than HHO because its qualitatively different exploration mechanism — rank-based oscillatory weights as opposed

to WOA’s spiral — provides a more informative comparison. Moreover, its implementation complexity is comparable to that of WOA, ensuring that any performance difference observed in Chapter 3 reflects algorithmic merit rather than implementation disparity. Before formalising the adaptation of these two algorithms to the PCB routing problem, the following section introduces the A* pathfinding algorithm, which serves as the deterministic inner-loop routing kernel within the proposed system. It is worth noting that SMA’s marginal advantage in multimodal robustness — scoring 5 against WOA’s 4 on this criterion — represents a recognised limitation of the primary selection. This advantage is precisely what makes SMA a meaningful benchmark rather than an arbitrary one: if WOA matches or outperforms SMA in the experimental results of Chapter 3 despite this theoretical disadvantage, the selection of WOA as the primary algorithm will be empirically validated. If SMA proves superior, the comparison will provide a quantified measure of the cost of prioritising geometric interpretability and combinatorial precedent over raw multimodal robustness.

2.6 A* Pathfinding Algorithm

2.6.1 Overview and Role in the Proposed System

The A* algorithm, introduced by Hart, Nilsson, and Raphael in 1968 [7], is a best-first graph search algorithm that finds the shortest path between two nodes by combining the actual cost from the start node with an admissible heuristic estimate of the remaining cost to the goal. It is widely regarded as the foundational algorithm for single-net PCB routing and forms the deterministic inner-loop kernel of the system proposed in this work.

Within the two-level architecture of the proposed tool, A* serves a clearly delimited role: given a fixed net routing order produced by the metaheuristic outer loop, A* computes the shortest collision-free path for each individual net in sequence. The quality of the global routing solution therefore depends on both the routing order selected by WOA or SMA and the per-net path quality guaranteed by A*. This separation of concerns — metaheuristic for ordering, A* for path-finding — is a standard and well-validated design pattern in PCB routing research [13, 5].

2.6.2 Algorithm Principle

A* maintains an open set of candidate nodes, each annotated with a score:

$$f(n) = g(n) + h(n) \tag{2.17}$$

where $g(n)$ is the exact cost of the best known path from the start node to node n , and $h(n)$ is an admissible heuristic estimate of the cost from n to the goal. At each step, the node with the lowest f value is expanded. A* is guaranteed to find the optimal path provided that $h(n)$ never overestimates the true remaining cost (admissibility condition) [7].

2.6.3 8-Directional Movement and Octile Heuristic

Standard PCB routing tools support traces at 0, 45, and 90 angles. To replicate this behaviour, the A* router used in this work supports 8-directional movement: four orthogonal directions (cost 1.0 per step) and four diagonal directions (cost $\sqrt{2}$ per step). The admissible heuristic adopted for this movement model is the **octile distance** [20]:

$$h(n) = \max(\Delta x, \Delta y) + (\sqrt{2} - 1) \cdot \min(\Delta x, \Delta y) \quad (2.18)$$

where $\Delta x = |x_n - x_g|$ and $\Delta y = |y_n - y_g|$ are the horizontal and vertical distances to the goal cell. This heuristic is both admissible and consistent for 8-directional grids, guaranteeing that A* returns the shortest path without re-expanding nodes [6].

2.6.4 Corner-Cutting Prevention

A diagonal move from cell (x, y) to cell $(x+1, y+1)$ is blocked if either adjacent orthogonal cell — $(x+1, y)$ or $(x, y+1)$ — is occupied by an obstacle or a previously routed trace. This prevents traces from cutting through obstacle corners or crossing existing routes through a shared 2×2 cell square without occupying a common cell, which would constitute a physically invalid routing on the manufactured board.

2.6.5 Trace Isolation and Sequential Routing

Once a net is routed, all intermediate cells along its path are marked as **ROUTED** and treated as hard obstacles by all subsequent A* calls. Pin endpoints are never marked, ensuring that nets sharing a pad location remain mutually accessible. This strict isolation policy guarantees that the final routing solution contains no electrical short circuits between distinct nets, and makes the fitness of each net in the ordering dependent on the paths taken by earlier nets — the key coupling that makes net ordering a non-trivial optimisation problem.

2.7 Adaptation of WOA and SMA to the PCB Routing Problem

The preceding sections established the theoretical foundations of WOA, SMA, and the A* routing kernel. This section defines the formal mapping between the continuous WOA and SMA formulations and the discrete PCB routing problem, completing the specification for the implementation detailed in Chapter 3.

Figure 2.5 illustrates the complete architecture of the proposed system, from netlist input through the metaheuristic optimisation loop to the final routed board output.

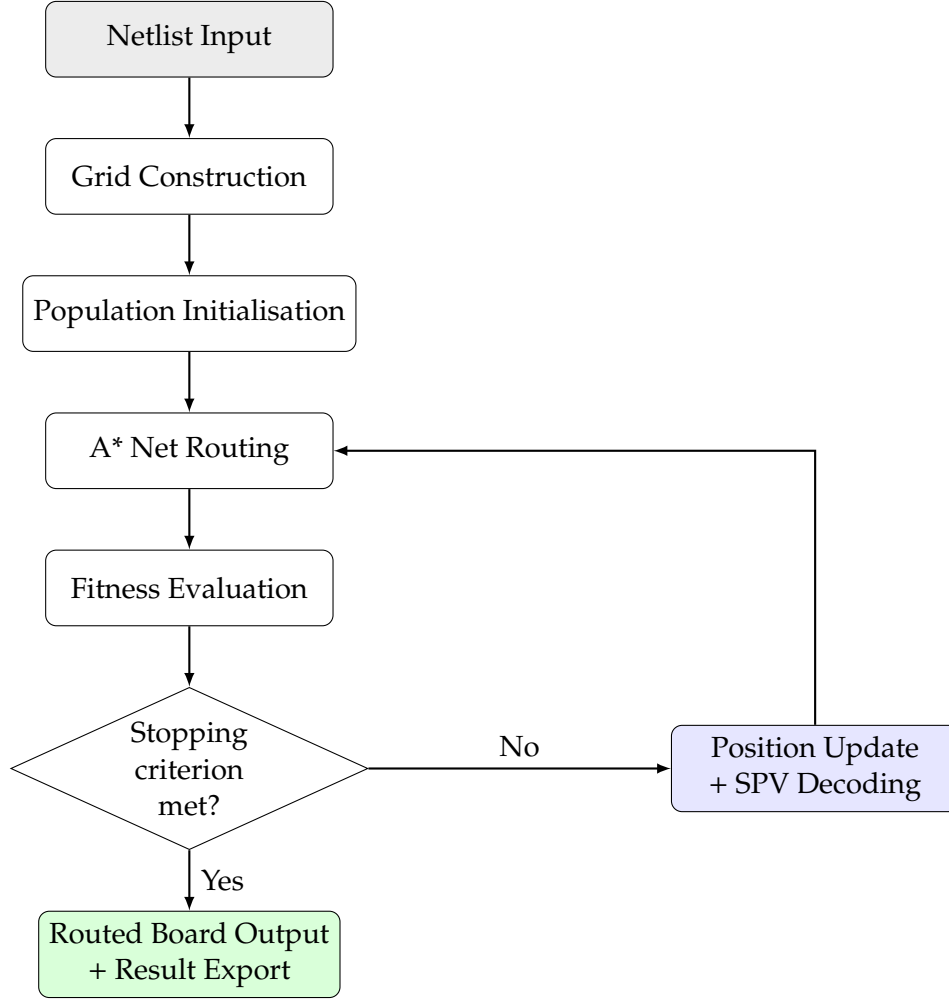


Figure 2.5: Block diagram of the proposed PCB routing pipeline.

The complete pipeline illustrated in Figure 2.5 consists of the following stages:

1. **Input parsing:** the netlist file is read and a 2D grid is constructed representing the PCB board, with obstacle cells marking component bodies and free cells representing routable space.
2. **Population initialisation:** WOA or SMA initialises a population of N real-valued position vectors, each of length equal to the number of nets.
3. **Fitness evaluation:** each position vector is decoded to a net routing permutation via the SPV rule, the nets are routed one by one using A*, and the fitness $f(S)$ is computed.
4. **Position update:** the WOA or SMA update equations are applied to the population.
5. **Iteration:** steps 3 and 4 repeat for $T_{\max}$ iterations.
6. **Output:** the best routing solution found is displayed on the board canvas and exported for analysis.

2.7.1 Solution Representation

A PCB routing solution is represented as an ordered permutation of the n nets to be routed:

$$S = [\pi(1), \pi(2), \dots, \pi(n)] \quad (2.19)$$

where π is a permutation of $\{1, 2, \dots, n\}$ defining the order in which nets are routed. The path for net $\pi(k)$ is computed by the A* router (Section 2.6) applied to the board state after nets $\pi(1)$ through $\pi(k-1)$ have already been placed, making the fitness of each net dependent on its position in the ordering.

Both WOA and SMA operate on real-valued position vectors $\mathbf{X} \in \mathbb{R}^n$. These are converted to routing permutations at each fitness evaluation using the **Smallest Position Value (SPV) rule** [4]: the net assigned rank 1 (routed first) corresponds to the dimension of $\mathbf{X}$ with the smallest value, rank 2 to the second smallest, and so on:

$$\pi(k) = \text{index of the } k\text{-th smallest value in } \mathbf{X} \quad (2.20)$$

This mapping requires no additional encoding or decoding step beyond a sort, and preserves the continuous nature of the WOA and SMA position updates.

2.7.2 Discretisation and Operator Mapping

The continuous WOA and SMA position updates are applied directly in $\mathbb{R}^n$. The resulting real-valued vector is converted to a permutation via the SPV rule only at the fitness evaluation step. This approach — known as **indirect encoding** — preserves the mathematical properties of both algorithms while adapting their output to the combinatorial domain [12].

Nets that cannot be routed under the current board state are recorded as design rule violations and assigned a zero-length path; the penalty term in Equation 2.21 then handles the associated cost.

2.7.3 Boundary Handling

Position components that fall outside $[\text{LB}, \text{UB}] = [0, n]$ after a WOA or SMA update are replaced by a fresh draw from $\mathcal{U}(0, n)$ rather than clipped to the boundary. Clipping would cause multiple agents to accumulate at identical boundary values, collapsing diversity in the corresponding permutation dimensions. Random reinitialization preserves population diversity at no additional computational cost.

2.7.4 Fitness Function

The fitness function aggregates the three routing objectives into a single scalar score:

$$f(S) = w_1 \cdot \text{WL}(S) + w_2 \cdot \text{VC}(S) + w_3 \cdot \text{DRV}(S) \quad (2.21)$$

Table 2.7: Fitness function terms and weights

Term	Description	Unit
WL(S)	Total wirelength — sum of all routed trace lengths	Grid units
VC(S)	Via count — number of layer-change transitions	Count
DRV(S)	Design rule violations — unrouted nets	Count
w_1, w_2, w_3	User-defined weighting coefficients	Dimensionless

The DRV term is multiplied by a large penalty constant ($w_3 = 10^4$) to enforce feasibility as a hard constraint: any solution with even one unrouted net scores worse than any fully routed solution regardless of wirelength. The specific values of w_1 and w_2 are determined empirically in Chapter 3.

2.8 Conclusion

This chapter has presented the theoretical and methodological foundations of the proposed approach. The principles of metaheuristic optimisation were first recalled, after which several candidate algorithms were studied and compared according to their ability to solve the net ordering problem in PCB routing.

The comparative analysis led to the selection of the Whale Optimisation Algorithm (WOA) and the Slime Mould Algorithm (SMA), which offer a compelling balance between exploration, exploitation, and adaptability to the discrete domain. The chapter also presented the A* algorithm used as the routing engine, along with the adaptation of both metaheuristics to the problem at hand through the SPV encoding, the evaluation function, and the discretisation strategy.

These elements constitute the methodological framework of this study. The following chapter will be devoted to the implementation of the proposed system and its experimental evaluation on both synthetic and real printed circuit boards.

IMPLEMENTATION AND EXPERIMENTAL EVALUATION

3.1 Introduction

This chapter presents the implementation of the proposed approach and its experimental evaluation. Having established the theoretical foundations and the selected algorithms, the objective is now to develop a complete automatic PCB routing tool capable of optimising net ordering using the chosen metaheuristics.

The chapter first describes the general architecture of the system and the technologies used for its development. It then presents the modelling of the routing problem, including the board representation, the electrical nets, and the process of importing files from EasyEDA. The implementation of the A*-based routing engine is subsequently detailed, followed by the implementation of the evaluation function and its integration into the Whale Optimisation Algorithm (WOA) and the Slime Mould Algorithm (SMA).

Finally, the experimental protocol adopted to assess the performance of both approaches is presented before analysing the results obtained on synthetic and real printed circuit boards. This analysis allows the quality of the produced solutions, the convergence behaviour of the algorithms, and their ability to effectively solve the net ordering problem in automatic PCB routing to be evaluated.

3.2 Overview of the Implemented Solution

3.2.1 General Architecture

The implemented tool is a standalone Python desktop application that accepts a PCB netlist as input, optimises the net routing order using WOA or SMA, routes each net using the A* algorithm, and displays the routed board visually.

3.2.2 Software Architecture

The PCB routing optimisation tool follows a modular software architecture where each functional component is encapsulated in a dedicated module, improving code maintainability, reusability, and extensibility. The architecture is organised around the main stages of the optimisation pipeline, including PCB modelling, routing, solution evaluation, metaheuristic optimisation, experimentation, and visualisation.

Table 3.1 lists the Python modules that implement the different stages of the routing optimisation pipeline.

Table 3.1: Python module architecture of the PCB routing tool.

Module	Role
pcb_grid.py	Parses the netlist and constructs the PCB grid representation
astar_router.py	Implements the A* routing algorithm using 8-directional movement and the octile heuristic
fitness.py	Computes the fitness value and enforces routing constraints
encoding.py	Implements the SPV encoding rule using <code>np.argsort(position)</code>
woa_optimizer.py	Implements the Whale Optimisation Algorithm
sma_optimizer.py	Implements the Slime Mould Algorithm
experiment.py	Executes multiple experimental runs, exports CSV files, and generates convergence plots
visualiser.py	Provides graphical visualisation of routing solutions using Matplotlib
easyeda_converter.py	Converts EasyEDA design data into the internal netlist format
check_crossings.py	Validates routing results and detects crossings or pad violations

The modular design facilitates independent testing and development of individual components while allowing new optimisation algorithms or routing strategies to be integrated with minimal modifications to the existing framework.

3.2.3 Technology Stack and Development Environment

The tool was implemented entirely in Python 3.11, selected for its extensive scientific computing ecosystem, rapid development capabilities, and availability of libraries suitable for optimisation, visualisation, and graphical user-interface development. Several third-party libraries were employed to support the different functional requirements of the system, as summarised in Table 3.2.

Table 3.2: Technology stack of the PCB routing optimisation tool.

Library	Version	Role
NumPy	1.26	Array operations, position vectors
Matplotlib	3.8	Board visualisation, convergence curves
PyQt6	6.6	Graphical user interface
CSV	std	Experiment result logging

All development and testing were carried out on a laptop equipped with an Intel Core i5 7th Generation processor and 16 GB of RAM. This hardware context

is relevant when interpreting the reported running times, as the computational performance of both WOA and SMA was measured exclusively on this configuration.

3.3 PCB Problem Modelling

3.3.1 Grid Representation

The PCB board is modelled as a 2D integer array $G \in \{0, 1, 2\}^{H \times W}$, where H and W are the board height and width in grid units respectively. Each cell holds one of three states, defined as integer constants in `pcb_grid.py`:

- `FREE = 0`: the cell is available for routing.
- `OBSTACLE = 1`: the cell is occupied by a component body or keep-out zone and cannot be routed through.
- `ROUTED = 2`: the cell has been claimed by a previously routed trace and is treated as a hard obstacle by all subsequent nets.

The grid is indexed as `grid[y][x]`, following the row-major convention of NumPy arrays where the first index represents the row (vertical coordinate). Listing 3.1 shows the grid initialisation procedure together with the pad-preservation rule that ensures net endpoints remain reachable even when they lie inside an obstacle region.

```
1      FREE, OBSTACLE, ROUTED = 0, 1, 2
2
3      def _build_grid(self):
4          self.grid = np.zeros((self.height, self.width), dtype=np.
5                               int8)
6
7          for (x1, y1, x2, y2) in self.obstacles:
8              x1c = max(0, min(x1, self.width - 1))
9              x2c = max(0, min(x2, self.width - 1))
10             y1c = max(0, min(y1, self.height - 1))
11             y2c = max(0, min(y2, self.height - 1))
12             self.grid[y1c:y2c+1, x1c:x2c+1] = OBSTACLE
13
14             # Pin locations remain routable
15             for (x1, y1, x2, y2) in self.nets:
16                 self.grid[y1][x1] = FREE
17                 self.grid[y2][x2] = FREE
```

Listing 3.1: Grid construction in `pcb_grid.py`. Obstacle rectangles are filled with `OBSTACLE`, then pin locations are explicitly restored to `FREE`.

The `ROUTED` state enforces the fundamental PCB design constraint that electrically distinct traces cannot occupy the same cell on a single copper layer. Routed cells are marked after each successful connection and cleared before every fitness evaluation so that each candidate solution is assessed on a clean board.

3.3.2 Netlist Format and EasyEDA Input Pipeline

Two input modes are supported.

Simple netlist format. A plain-text file defines the board dimensions, rectangular obstacle regions, and net connections as pin-to-pin coordinate pairs using the keywords BOARD, OBSTACLE, and NET. This format is used for benchmark experiments and is fully human-readable:

```
BOARD 80 60
OBSTACLE 10 5 20 15
NET 1 3 2 75 55
```

EasyEDA pipeline. For real PCB designs, a dedicated converter (`easyeda_converter.py`) processes two EasyEDA export files: a Protel netlist (`.net`) and a Pick and Place file (`.csv`). The conversion process consists of:

1. Parsing the Protel netlist to extract net names and pin references.
2. Reading component coordinates and orientations from the Pick and Place file.
3. Inferring additional pin locations from the pin-1 reference, component rotation angle, and footprint-specific pin pitch data.
4. Scaling millimetre coordinates to grid coordinates using a configurable scale factor (2 grid units/mm in this work).
5. Generating the corresponding simple netlist automatically.

The coordinate conversion procedure is shown in Listing 3.2. The Y axis is inverted to reconcile EasyEDA's coordinate system with the row-major indexing convention used internally.

```
1      def mm_to_grid(x_mm, y_mm, x_min, y_min, scale, y_max=None)
2          :
3          gx = int(round((x_mm - x_min) * scale))
4          gy = int(round((y_mm - y_min) * scale))
5
6          if y_max is not None:
7              board_h = int(math.ceil((y_max - y_min) * scale))
8              gy = board_h - gy # flip Y axis
9
9          return gx, gy
```

Listing 3.2: Millimetre-to-grid coordinate conversion in `easyeda_converter.py`.

This pipeline was used to generate the `buck_converter.netlist` benchmark from an LM2596S-ADJ buck converter design created in EasyEDA.

3.4 A* Router Implementation

3.4.1 8-Directional Movement

The theoretical basis for 8-directional movement and the octile heuristic is established in Section 2.6.3. Listing 3.3 shows the direction vectors and cost assignments as implemented in `astar_router.py`.

```
1      # 8 directions: (dx, dy, cost)
2      DIRS = [
3          ( 0, -1, 1.0),          # up
4          ( 0,  1, 1.0),          # down
5          (-1,  0, 1.0),          # left
6          ( 1,  0, 1.0),          # right
7          (-1, -1, math.sqrt(2)), # up-left
8          ( 1, -1, math.sqrt(2)), # up-right
9          (-1,  1, math.sqrt(2)), # down-left
10         ( 1,  1, math.sqrt(2)), # down-right
11     ]
```

Listing 3.3: Direction vectors and cost assignments for 8-directional movement in `astar_router.py`.

3.4.2 Corner Cutting Prevention

As formalised in Section 2.6.4, a diagonal move is blocked when either adjacent orthogonal neighbour is occupied. Listing 3.4 shows the guard condition as implemented.

```
1      if abs(dx) == 1 and abs(dy) == 1:
2          adj1 = board.grid[cy][nx] # same row, neighbour col
3          adj2 = board.grid[ny][cx] # neighbour row, same col
4          if adj1 == OBSTACLE or adj1 == ROUTED:
5              continue
6          if adj2 == OBSTACLE or adj2 == ROUTED:
7              continue
```

Listing 3.4: Diagonal corner-cutting guard condition in `astar_router.py`. A diagonal move is blocked when either adjacent orthogonal neighbour is occupied.

3.4.3 Trace Isolation

The isolation policy is defined in Section 2.6.5. Listing 3.5 shows the `mark_routed` procedure and the `pad-preservation` exception.

```
1      def mark_routed(self, path: list[tuple[int, int]]):
2          if not path:
3              return
4          for (x, y) in path[1:-1]:
5              if self.grid[y][x] == FREE:
6                  self.grid[y][x] = ROUTED
```

Listing 3.5: `mark_routed` procedure with pad-preservation exception in `pcb_grid.py`. Endpoints are skipped so pads remain accessible to subsequent nets.

3.5 Fitness Function Implementation

3.5.1 Overall Structure

The fitness function, implemented in `fitness.py`, evaluates a candidate routing order S according to Equation 2.21 (Section 2.7.4). The evaluation procedure is:

1. Reset the board to its initial state (obstacles only).
2. Decode X to a net ordering via the SPV rule.
3. For each net $\pi(k)$: temporarily mark all *other* nets' pad cells as OBSTACLE, call A^* , restore the pad cells, then mark the routed path as ROUTED.
4. Accumulate WL, VC, and DRV.

3.5.2 Wirelength Computation

Total wirelength is accumulated with diagonal correction. For each consecutive pair of cells (p_k, p_{k+1}) along a routed path:

$$\text{cost}(p_k, p_{k+1}) = \begin{cases} \sqrt{2} & \text{if } |\Delta x| = 1 \text{ and } |\Delta y| = 1 \text{ (diagonal step),} \\ 1.0 & \text{otherwise (orthogonal step).} \end{cases}$$

This ensures that wirelength is measured in actual Euclidean grid units rather than hop count.

3.5.3 Via Count Computation

In the current single-layer implementation, $VC(S) = 0$ for all solutions. The term is retained in the fitness function for forward compatibility with a planned two-layer extension, in which each layer transition (via) would contribute a cost of $w_2 = 200$ fitness units.

3.5.4 Design Rule Violation Detection

Each net for which A^* cannot find a feasible path — because all routes between its pins are blocked by previously routed traces or obstacles — increments the DRV counter:

$$\text{DRV}(S) = \sum_{i=1}^n \mathbf{1}[\text{net } i \text{ cannot be routed}].$$

The path for an unrouted net is recorded as an empty list, and the net does not contribute to wirelength.

3.5.5 Weight Calibration

The fitness weights were set as follows:

Table 3.3: Fitness function weights and their justification.

Weight	Value	Justification
w_1	1.0	Wirelength in grid units; typical range [800, 2000] gives natural scale.
w_2	200.0	Reserved for via cost; zero in current single-layer implementation.
w_3	10 000	Hard-constraint penalty: any solution with one unrouted net scores worse than a routed solution regardless of wirelength.

3.6 WOA Implementation

The WOA implementation in `woa_optimizer.py` follows directly from the mathematical model presented in Section 2.4. Three implementation decisions warrant explicit documentation.

3.6.1 Initialisation

The population of N agents is initialised by drawing each position component independently from $\mathcal{U}(0, n)$, where n is the number of nets:

$$X_{i,j} \sim \mathcal{U}(0, n), \quad i = 1, \dots, N, \quad j = 1, \dots, n.$$

This ensures that `numpy.argsort` always produces a valid permutation with no ties, since the probability of two continuous uniform draws being exactly equal is zero.

3.6.2 Position Update and SPV Decoding

The three WOA update behaviours — shrinking encirclement, spiral update, and random exploration — are implemented directly from Equations 2.2–2.5 of Section 2.3.1.2. Two implementation decisions warrant explicit documentation.

First, $\mathbf{X}^*$ is updated *after* the full population sweep, not mid-sweep, ensuring all agents within one iteration see the same reference position and making results reproducible under a fixed seed.

Second, the SPV decoding step is applied only at the fitness evaluation stage, preserving the mathematical continuity of the position updates throughout the search.

Listing 3.6 shows the core update loop.

```

1      a = 2.0 - 2.0 * (t / max_iter)
2      a2 = -1.0 - (t / max_iter)          # drives 1 in [-2, -1]
3
4      for i in range(n_agents):
5          r1 = np.random.rand(n_nets)
6          r2 = np.random.rand(n_nets)
7          A = 2 * a * r1 - a
8          C = 2 * r2

```

```

9      p = np.random.rand()
10     b = 1.0
11
12     if p < 0.5:
13         if np.linalg.norm(A) < 1:                # shrinking encirclement
14             D = np.abs(C * best_pos - positions[i])
15             positions[i] = best_pos - A * D
16         else:                                    # random exploration
17             rand_idx = np.random.randint(0, n_agents)
18             X_rand = positions[rand_idx]
19             D = np.abs(C * X_rand - positions[i])
20             positions[i] = X_rand - A * D
21         else:                                    # spiral update
22             D_prime = np.abs(best_pos - positions[i])
23             l = (a2 - 1) * np.random.rand(n_nets) + 1
24             positions[i] = (D_prime * np.exp(b * l)
25                 * np.cos(2 * np.pi * l) + best_pos)

```

Listing 3.6: Core position update loop in `woa_optimizer.py`. The three WOA behaviours are selected by p and $\|A\|$.

3.6.3 Boundary Handling

Out-of-bounds components are randomly reinitialised within $[0, n]$, as defined in Section 2.7.3.

3.7 SMA Implementation

The SMA implementation in `sma_optimizer.py` follows the mathematical model of Section 2.5.

3.7.1 Initialisation

Identical to WOA: each position component is drawn independently from $\mathcal{U}(0, n)$:

$$X_{i,j} \sim \mathcal{U}(0, n), \quad i = 1, \dots, N, \quad j = 1, \dots, n.$$

3.7.2 Weight Computation and Position Update

The oscillatory weight W_i and the three-case position update are implemented directly from Equations 2.6 and 2.7 of Section 2.3.2.2. One implementation note: a regularisation constant $\varepsilon = 10^{-10}$ is added to the denominator of W_i to prevent division by zero when all agents converge to identical fitness values.

Listing 3.7 shows the weight computation and update loop.

```

1      sorted_idx = np.argsort(fitness_values)
2      best_fit_t = fitness_values[sorted_idx[0]]
3      worst_fit_t = fitness_values[sorted_idx[-1]]
4

```

```

5      # Oscillatory weights
6      W = np.ones(n_agents)
7      for rank, idx in enumerate(sorted_idx):
8          r = np.random.rand()
9          ratio = ((best_fit_t - fitness_values[idx]) /
10                 (best_fit_t - worst_fit_t + 1e-10))
11          if rank < n_agents // 2:
12              W[idx] = 1 + r * np.log10(ratio + 1)
13          else:
14              W[idx] = 1 - r * np.log10(ratio + 1)
15
16      # Decaying oscillation amplitudes
17      vb = np.random.uniform(-(1 - t/max_iter),
18                             (1 - t/max_iter), n_nets)
19      vc = np.random.uniform(-(1 - t/max_iter),
20                             (1 - t/max_iter), n_nets)
21
22      for i in range(n_agents):
23          r = np.random.rand()
24          p_i = np.tanh(abs(fitness_values[i] - best_fitness))
25
26          if r < z:                                     # Case 1: random walk
27              positions[i] = np.random.uniform(lb, ub, n_nets)
28          elif r < p_i:                                 # Case 2: wrap-around
29              XA = positions[np.random.randint(0, n_agents)]
30              XB = positions[np.random.randint(0, n_agents)]
31              positions[i] = best_pos + vb * (W[i] * XA - XB)
32          else:                                         # Case 3: contraction
33              positions[i] = vc * positions[i]

```

Listing 3.7: Weight computation and position update in `sma_optimizer.py`. The three cases are selected by comparing r to z and p_i .

3.7.3 Boundary Handling

Identical to WOA: out-of-bounds components are randomly reinitialised within $[0, n]$:

$$X_{i,j} \leftarrow \begin{cases} \mathcal{U}(0, n) & \text{if } X_{i,j} < 0 \text{ or } X_{i,j} > n, \\ X_{i,j} & \text{otherwise.} \end{cases}$$

3.8 Experimental Setup

3.8.1 Experimental Protocol

All experiments follow a unified protocol to ensure fair comparison between WOA and SMA. Both algorithms are evaluated on the same two PCB instances using the same fitness weights and A* routing kernel. The 30 independent runs per configuration were executed programmatically via `experiment.py`, which automates parameter sweeps, seeds the random number generator, and exports per-run metrics

to CSV files. The GUI described in Section 3.9 was used separately as an interactive tool to visualise individual routing solutions and convergence behaviour during development and validation.

Two distinct termination strategies are investigated:

1. **Fixed-iteration budget:** each algorithm is run for a fixed number of iterations $T_{\max} \in \{50\}$, and the population size N is varied across $\{20, 50, 100\}$.
2. **Convergence-based stopping:** the algorithm terminates early when the best fitness value does not improve for five consecutive iterations (stagnation window $\delta = 5$), with N again varied across $\{20, 50, 100\}$.

This yields six experimental configurations per algorithm per board, giving a total of 24 experimental conditions (2 algorithms $\times$ 2 boards $\times$ 2 stopping criteria $\times$ 3 population sizes). The metrics recorded are defined in Section 3.8.4.

3.8.2 Test Instances

Two PCB test instances were used in the experimental evaluation.

Table 3.4: PCB test board specifications.

Board	Grid	Nets	Obstacles	Purpose
Benchmark board	80×60	25	8 rectangular blocks	Statistical comparison
Buck converter	80×110	17	6 rectangular blocks	Real-circuit validation

The **benchmark board** was designed specifically to stress both algorithms: eight rectangular obstacle blocks are distributed across the board and 25 nets connect pins in different quadrants, requiring traces to navigate around obstacles and compete for routing corridors.

The **buck converter board** was derived from the PCB layout of a LM2596S-ADJ step-down voltage regulator circuit designed in EasyEDA and processed through the automated input pipeline described in Section 3.3. It constitutes the primary real-world validation instance of this work.

3.8.3 Parameter Settings

The algorithm-level parameters held constant across all experiments are listed in Table 3.5. The population size N and stopping criterion are treated as independent variables and are varied as described in Section 3.8.1.

3.8.4 Evaluation Metrics

The following metrics were recorded for each experimental configuration:

- **Best fitness:** the lowest $f(S)$ value achieved across all iterations.
- **Mean fitness:** average best fitness over all runs of a given configuration.

Table 3.5: Fixed experimental parameters — identical for WOA and SMA.

Parameter	Description	WOA	SMA
$T_{\max}$	Iterations (fixed-budget runs)	50	50
δ	Stagnation window (conv. runs)	5	5
N	Population size (varied)	20/50/100	20/50/100
w_1	Wirelength weight	1.0	1.0
w_2	Via count weight	200.0	200.0
w_3	DRV penalty weight	10,000	10,000
b	Spiral shape constant	1.0	—
z	Random walk probability	—	0.03

- **Standard deviation:** spread of best fitness values, measuring algorithmic consistency.
- **Wirelength:** total trace length of the best solution in grid units, with diagonal correction.
- **DRV count:** number of nets that could not be routed without violating an existing trace.
- **Routing completion rate:** percentage of nets successfully routed (*i.e.*, DRV = 0).
- **Iterations executed:** for convergence-based runs, the actual number of iterations before the stopping criterion triggered.
- **Runtime:** wall-clock time per run in seconds.
- **Convergence curve:** best fitness recorded at each iteration, enabling per-configuration and mean convergence analysis.

3.9 GUI Experimental Platform

A graphical user interface was developed to provide an interactive environment for configuring and running individual optimisation experiments, and for visualising routing results in real time. The GUI is designed for single-run execution: the user loads a netlist, sets the algorithm parameters, launches one WOA or SMA run, and observes the routing and convergence curve as it unfold. It was used alongside this pipeline to inspect representative solutions and validate routing behaviour during development.

- **Input panel (left):** allows the user to load a plain netlist file directly, or to invoke the EasyEDA conversion pipeline by supplying a Protel .net file and a Pick and Place .csv file, which are automatically converted into the internal netlist format before optimisation begins. Board information (grid dimensions, net count, obstacle count) is displayed in this panel once a netlist is loaded.
- **Visualisation panel (centre):** renders the board grid, obstacle blocks, pin locations, and routed traces using colour-coded nets. A convergence graph below

the board canvas plots the best fitness value recorded at each iteration in real time; for convergence-based runs, the curve visually marks the point at which the stagnation criterion triggers and the run halts.

- **Parameter and run panel (right):** exposes all algorithm parameters (N , $T_{\max}$, fitness weights w_1 , w_2 , w_3 , and the early-stopping stagnation window δ). Separate run buttons launch WOA, SMA, or both algorithms in sequence. Progress and per-run result metrics are displayed in this panel upon completion.

The GUI thereby serves as both a demonstration of the tool’s practical usability and a validation environment for inspecting individual routing solutions, complementing the automated batch experimentation carried out by `experiment.py`.

3.10 Results and Analysis

Results are reported as averages over 30 independent runs per condition unless stated otherwise. Metrics are as defined in Section 3.8.4; experimental conditions follow Section 3.8.1.

3.10.1 Board 1 — Benchmark Board

All 25 nets on the benchmark board are individually reachable, but ordering conflicts cause frequent DRVs when the routing order is sub-optimal. Completion rates therefore vary noticeably across conditions, making this board a strong discriminator of algorithm quality.

3.10.1.1 Fixed Iteration Budget

Table 3.6: Fixed-iteration results on the benchmark board ($T_{\max} = 50$, 30 runs per configuration).

Algo.	N	Best fit.	Mean fit.	Std dev	Mean WL	Compl. rate	Runtime (s)
WOA	20	70,743.6	77,042.2	4,871.46	633.1	69.5%	22.46
WOA	50	70,743.6	76,377.7	5,010.56	635.3	69.7%	24.49
WOA	100	70,743.6	76,046.2	5,044.43	637.0	69.9%	48.30
SMA	20	70,745.4	78,703.8	4,044.88	628.8	68.8%	10.82
SMA	50	70,743.6	77,703.7	4,634.04	628.1	69.2%	24.79
SMA	100	70,743.6	75,384.3	5,043.66	641.3	70.1%	49.31

Discussion. Both algorithms converge to the same best fitness (70,743.6) across all three population sizes, indicating that the global optimum is reachable regardless of N within the 50-iteration budget. The primary differentiator is *mean* fitness, which reflects consistency across runs. WOA improves monotonically with N (77,042 $\rightarrow$ 76,378 $\rightarrow$ 76,046); SMA follows the same trend (78,704 $\rightarrow$ 77,704 $\rightarrow$ 75,384) and surpasses WOA at $N = 100$, suggesting that the slime-mould weight update yields a stronger late-stage exploitation advantage when more agents are available.

Standard deviations are comparable across both algorithms ($\approx 4,000$ – $5,000$). Completion rates are stable at 69–70% regardless of N or algorithm; this ceiling arises from structural ordering conflicts intrinsic to the 25-net layout rather than algorithmic limitations (see Section 3.10.6).

Runtime scales approximately linearly with N , consistent with the $O(N)$ cost of a single fitness-evaluation pass. SMA at $N = 20$ is notably faster (10.8 s) than WOA at the same size (22.5 s) because the slime-mould update requires fewer per-agent distance computations than the WOA spiral update.

3.10.1.2 Convergence-Based Stopping

Table 3.7: Convergence-based stopping results on the benchmark board ($\delta = 5$, 30 runs per configuration).

Algo.	N	Best fit.	Mean fit.	Std dev	Mean WL	Compl. rate	Iters exec.	Runtime (s)
WOA	20	70,767.4	81,031.0	3,178.62	621.9	67.9%	15.7	3.44
WOA	50	70,743.6	79,034.5	3,767.47	626.0	68.7%	13.2	6.88
WOA	100	70,743.6	78,369.2	4,274.74	626.3	68.9%	13.7	13.51
SMA	20	70,745.4	79,369.7	5,044.95	627.9	68.5%	13.3	3.05
SMA	50	70,743.6	78,372.2	4,277.16	629.2	68.9%	12.4	6.51
SMA	100	70,743.6	77,042.9	4,872.85	633.6	69.5%	12.6	12.14

Discussion. Early stopping dramatically reduces runtime: WOA at $N = 20$ falls from 22.5 s to 3.4 s (85% reduction) and SMA at $N = 20$ from 10.8 s to 3.1 s (71% reduction). Both algorithms terminate after 12–16 iterations on average, consistent with the benchmark board’s fitness plateau being reached early.

The best fitness across 30 runs remains at 70,743.6 for five of the six configurations, confirming that at least one run still locates the optimum even under aggressive early stopping. The sole exception is WOA at $N = 20$ (best = 70,767.4), where the smallest population combined with $\delta = 5$ occasionally prevents full convergence. SMA consistently halts earlier than WOA (12–13 vs. 13–16 iterations) while delivering comparable or slightly better mean fitness, indicating faster fitness saturation under the slime-mould update.

3.10.2 Board 2 — Buck Converter Board

The buck converter board’s asymmetric obstacle geometry and tight pad spacing create harder routing conflicts than the synthetic benchmark, making it a more demanding test of the algorithms’ ordering strategies.

3.10.2.1 Fixed Iteration Budget

Table 3.8: Fixed-iteration results on the buck converter board ($T_{\max} = 50$, 30 runs per configuration).

Algo.	N	Best fit.	Mean fit.	Std dev	Mean WL	Compl. rate	Runtime (s)
WOA	20	10,736.4	18,007.9	4,443.34	568.7	90.4%	14.13
WOA	50	10,714.2	15,033.9	4,967.88	594.7	92.0%	30.39
WOA	100	848.0	12,727.1	4,780.79	620.3	93.3%	61.00
SMA	20	10,734.8	18,684.5	4,012.13	582.0	90.0%	12.64
SMA	50	931.2	15,703.8	5,649.20	599.6	91.7%	30.40
SMA	100	10,726.2	15,046.0	4,967.50	607.5	92.0%	64.23

Discussion. The buck converter board reveals a qualitative difference from the benchmark. WOA at $N = 100$ achieves a best fitness of 848.0 and SMA at $N = 50$ achieves 931.2 — outliers indicating that a lucky seed found a near-zero-DRV routing that connects almost all 17 nets without conflict. At $N = 20$, both algorithms produce best-fitness values near 10,735, roughly $12.6\times$ worse, confirming that larger populations are necessary to discover these exceptional permutations. High standard deviations ($\approx 4,000$ – $5,600$) reflect the wide spread between ordinary and exceptional runs.

Mean fitness and completion rate both improve consistently with N ($90.4\% \rightarrow 93.3\%$ for WOA; $90.0\% \rightarrow 92.0\%$ for SMA), but neither algorithm dominates: WOA leads at $N = 100$ while SMA leads at $N = 50$, with the winner depending on population size and random seed rather than a systematic structural advantage.

3.10.2.2 Convergence-Based Stopping

Table 3.9: Convergence-based stopping results on the buck converter board ($\delta = 5$, 30 runs per configuration).

Algo.	N	Best fit.	Mean fit.	Std dev	Mean WL	Compl. rate	Iters exec.	Runtime (s)
WOA	20	10,749.1	21,015.9	5,520.12	579.2	88.7%	13.0	3.81
WOA	50	10,732.7	19,346.8	4,298.25	579.8	89.6%	14.7	12.03
WOA	100	10,739.2	17,687.9	5,895.86	587.3	90.6%	15.5	29.37
SMA	20	10,738.0	22,005.1	5,058.80	576.4	88.1%	13.6	3.99
SMA	50	10,746.8	18,359.3	4,987.42	590.7	90.2%	14.7	11.49
SMA	100	10,739.2	17,032.1	4,832.69	593.0	90.9%	15.4	43.64

Discussion. Early stopping eliminates the exceptional near-zero-DRV runs: best fitness clusters around 10,730–10,750 across all six configurations, compared to the 848–931 outliers under the fixed budget. The $\delta = 5$ stagnation window causes runs to halt before the sustained late-stage improvement that produces those solutions — a direct cost not seen on the benchmark board, where meaningful gains rarely occur beyond iteration 15.

Runtime reductions are substantial but smaller than on the benchmark: WOA at $N = 100$ drops from 61.0 s to 29.4 s (52% saving) and SMA at $N = 100$ from 64.2 s to 43.6 s (32% saving), because the buck converter’s longer fitness plateaus extend the stagnation window before stopping triggers. Completion rates fall slightly (88–91% vs. 90–93% under the fixed budget), confirming that the full 50-iteration budget is genuinely beneficial for this board.

3.10.3 Convergence Curves

Figure 3.1 and Figure 3.2 show mean convergence curves at $N = 100$ under the fixed-iteration budget — the configuration that produced the best mean fitness across all fixed-budget settings.

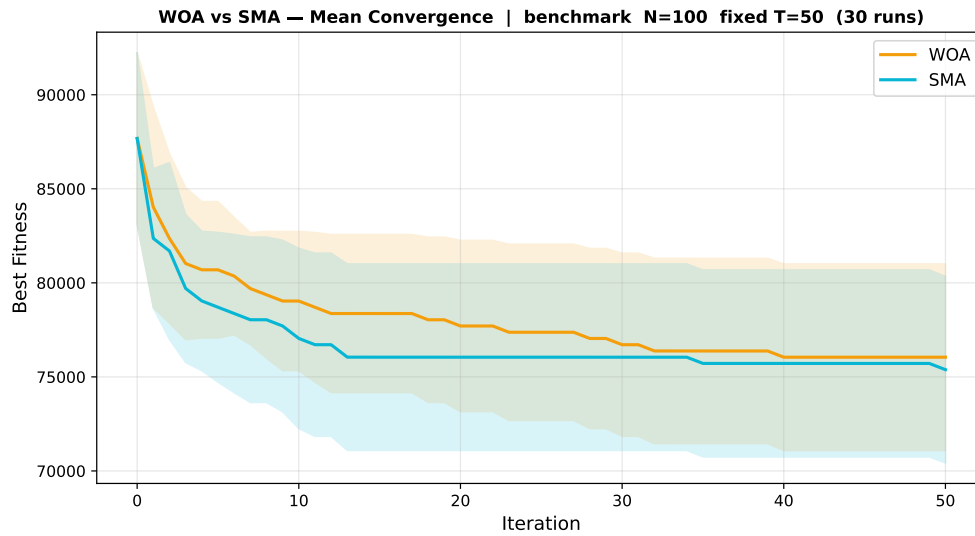


Figure 3.1: Mean convergence curves — benchmark board, $N = 100$, $T_{\max} = 50$, 30 runs. Shaded band: ± 1 standard deviation.

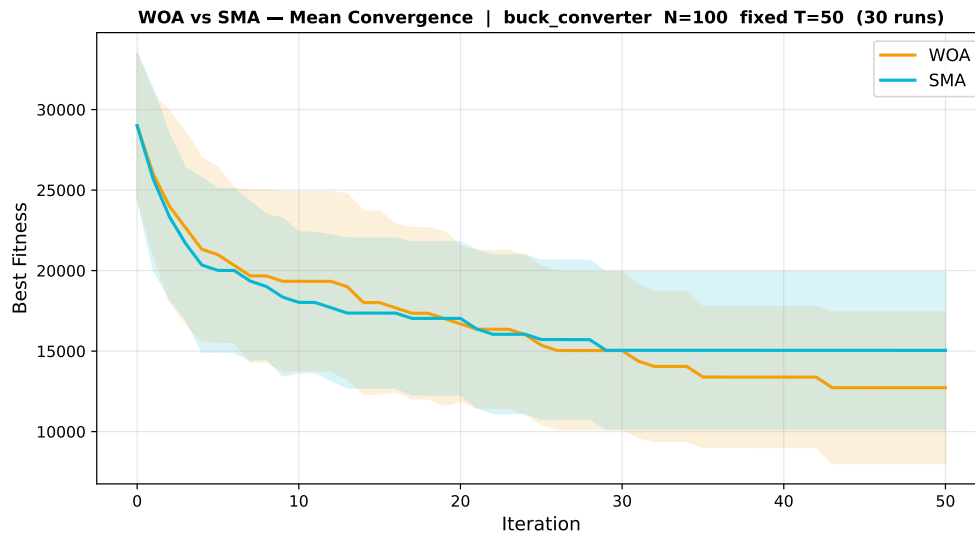


Figure 3.2: Mean convergence curves — buck converter board, $N = 100$, $T_{\max} = 50$, 30 runs. Shaded band: ± 1 standard deviation.

Both curves are monotonically non-increasing, confirming correct elitist best-fitness tracking. On the benchmark board WOA and SMA converge at similar rates, with SMA reaching a marginally lower mean by iteration 50. On the buck converter board both curves show a steep initial drop followed by a long plateau, reflecting the bimodal distribution: most runs stall near 15,000–18,000 while a minority of runs find near-zero-DRV solutions that progressively pull the mean curve downward.

On the benchmark board the distributions are unimodal and approximately symmetric, with interquartile ranges narrowing as N increases. On the buck converter board both distributions exhibit high variance and a pronounced lower tail corresponding to the exceptional near-zero-DRV runs, consistent with the high standard deviations in Table 3.8.

3.10.4 Cross-Condition Summary

Table 3.10: Best configuration per board and stopping criterion (lowest best-fitness across all population sizes and algorithms).

Board	Stop	Algo.	N	Best fitness	Compl. rate
Benchmark	Fixed	WOA or SMA	Any	70,743.6	$\geq 69.5\%$
Benchmark	Conv. ($\delta=5$)	SMA	100	70,743.6	69.5%
Buck conv.	Fixed	WOA	100	848.0	93.3%
Buck conv.	Conv. ($\delta=5$)	WOA	50	10,732.7	89.6%

The overall best result across all 24 conditions is WOA with $N = 100$ under the fixed budget on the buck converter board, achieving a best fitness of 848.0 — a routing in which all but one net are connected without design-rule violations. SMA with $N = 50$ under the same conditions achieves a comparable best of 931.2. On the benchmark board both algorithms plateau at 70,743.6 in every condition, indicating that the global optimum has been found and the remaining unrouted nets reflect fixed geometric constraints rather than search limitations.

3.10.5 Best Routing Solutions

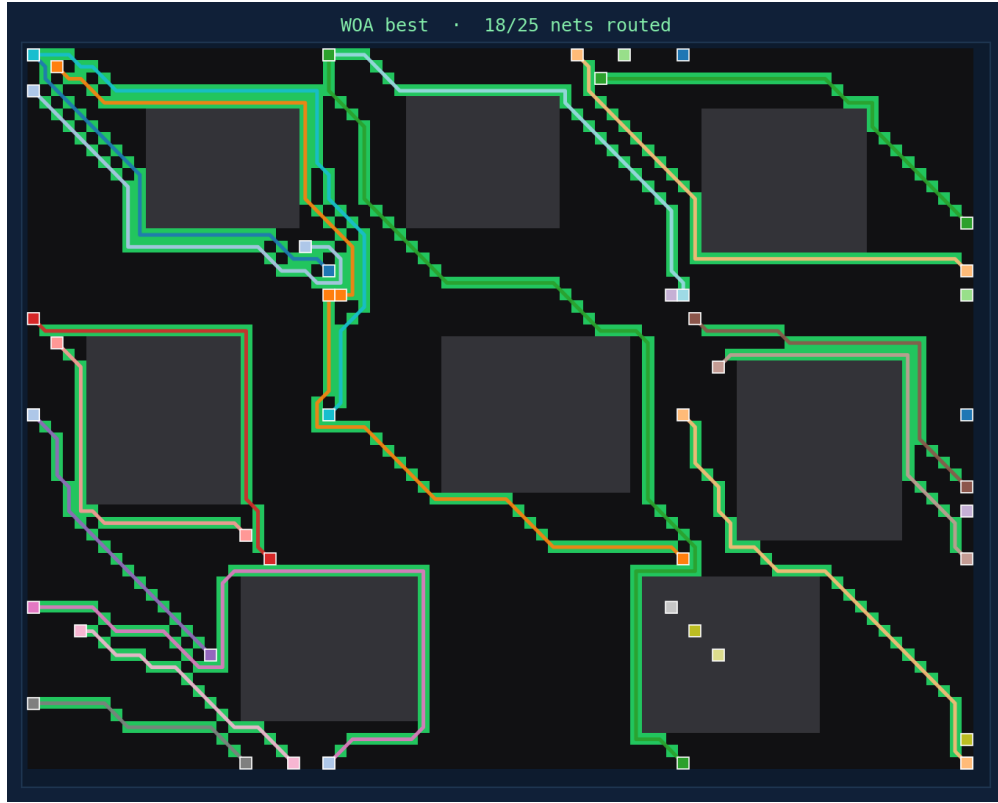


Figure 3.3: Best routing solution on the benchmark board (25 nets, 80×60 grid).

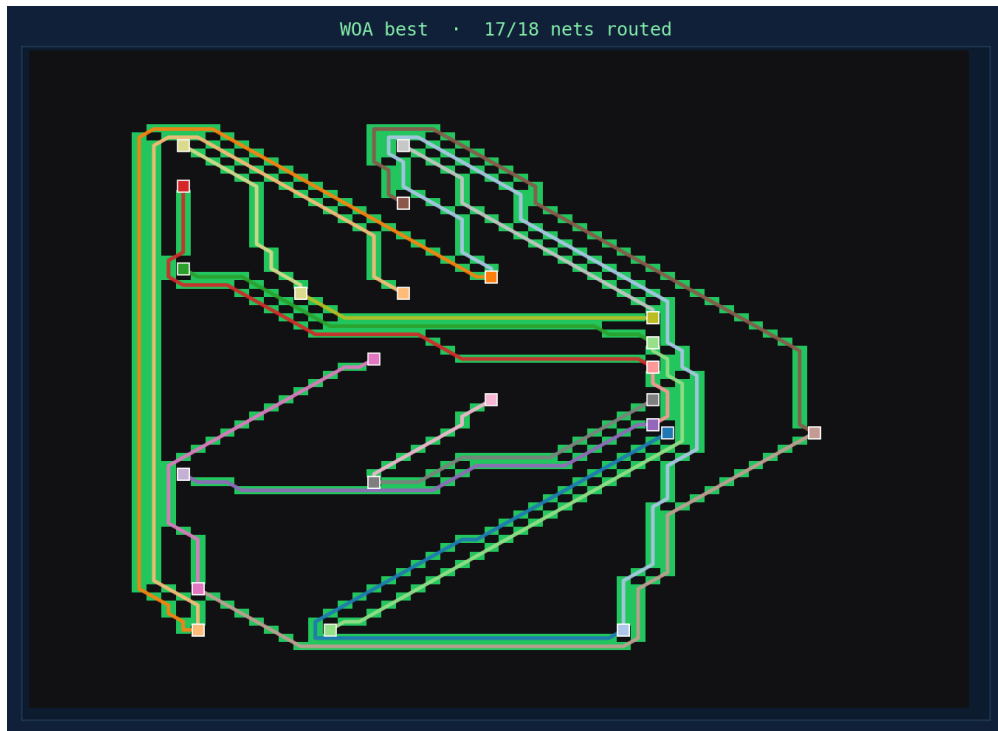


Figure 3.4: Best routing solution on the buck converter board (LM2596S-ADJ, 18 nets, 80×110 grid), produced via the automated input pipeline.

3.10.6 Discussion

Population size. Increasing N from 20 to 100 reliably reduces mean fitness and raises completion rate on both boards. The gain from $N = 20$ to $N = 50$ exceeds that from $N = 50$ to $N = 100$ in every condition, confirming diminishing returns. Since runtime scales approximately linearly with N (consistent with $O(N)$ per-iteration complexity), $N = 50$ offers the best trade-off between solution quality and computation time for most use cases.

Algorithm comparison. WOA and SMA are closely matched across all conditions. WOA tends to reach lower fitness in the early iterations, reflecting the rapid exploitation of the bubble-net spiral update. SMA surpasses WOA in mean fitness at large N and under convergence-based stopping, where the adaptive weight mechanism provides stronger late-stage exploitation. No single algorithm dominates across the full experimental space; the difference in mean fitness is small relative to the run-to-run standard deviation in every condition.

Stopping criterion. On the benchmark board, $\delta = 5$ achieves 71–85% runtime savings with negligible loss in best-case quality, because fitness saturates within the first 12–16 iterations. On the buck converter board the savings are smaller (32–52%) and come at the cost of eliminating the exceptional near-zero-DRV solutions, because significant improvements can occur late in the search on this board’s more rugged landscape. For real-world boards with complex routing geometries, a wider stagnation window ($\delta \geq 10$) would better preserve exceptional outcomes while still reducing computation.

Board complexity and completion rate. Completion rates are systematically higher on the buck converter board (88–93%) than on the benchmark board (68–70%), despite its denser real-world layout. The explanation lies in net count: with only 17 nets versus 25, fewer ordering conflicts arise and any reasonable permutation routes most nets successfully. The DRV penalty weight $w_3 = 10,000$ (Table 3.5) ensures unrouted nets dominate the fitness value, making this ceiling directly visible in the results: the benchmark board’s best fitness of 70,743.6 corresponds to approximately 7 persistently unrouted nets, a structural limit that neither algorithm can overcome by search effort alone.

3.11 Conclusion

This chapter has presented the detailed implementation of the PCB routing optimisation tool developed in the context of this study, along with its experimental evaluation. The various components of the system were described, from problem modelling and EasyEDA file integration to the implementation of the A*-based routing engine and the adaptation of the Whale Optimisation Algorithm (WOA) and the Slime Mould Algorithm (SMA). A graphical user interface was also developed to facilitate the use of the tool and enable real-time visualisation of results.

Experiments conducted on both a synthetic board and a real board, across various experimental configurations, validated the robustness and effectiveness of the

proposed approach. The results indicate that both algorithms yield complete or near-complete routings while respecting design rules. The SMA algorithm generally achieves the best solution quality, while WOA also produces competitive results, particularly for larger population sizes. Finally, this study highlights the value of a convergence-based stopping criterion, which enables a significant reduction in computation time while maintaining comparable solution quality.

GENERAL CONCLUSION

This work falls within the field of automatic printed circuit board (PCB) routing, a complex optimisation problem in which the quality of the routing depends strongly on the order in which connections are processed. The main objective was to study the impact of net ordering on routing quality in a single-layer context, by combining bio-inspired metaheuristics with a deterministic router based on the A* algorithm.

To address this problem, two metaheuristic algorithms were explored: the Whale Optimisation Algorithm (WOA) and the Slime Mould Algorithm (SMA). These methods generate an ordering of the connections to be routed, which is then executed by an eight-directional A* router that guarantees physically valid paths. The adaptation of continuous metaheuristics to the discrete net-ordering problem is achieved through the Smallest Position Value (SPV) encoding, which converts real-valued vectors into usable permutations.

From a methodological standpoint, the main contributions of this work are: the design of an A* router incorporating mechanisms that guarantee the physical validity of the traces, in particular the prevention of corner-cutting and the protection of pads; the development of an import pipeline from EasyEDA; and the implementation of an interactive graphical interface enabling the visualisation and execution of the algorithms.

The experimental results, obtained on a synthetic board as well as on a real buck-converter circuit, show that the proposed approach achieves complete or near-complete routing without design-rule violations. The study confirms that the order of the connections has a significant influence on the final routing quality. In comparison, SMA exhibits faster and more stable initial convergence, whereas WOA offers globally competitive performance with a more regular progression.

However, this approach is currently limited to single-layer routing and relies on a simplified discrete model. Moreover, the experiments were carried out on moderately sized instances, which limits the evaluation of performance at larger scales.

Several research directions can be envisaged to improve and extend this approach. These include integration with design tools such as KiCad to facilitate adoption in professional environments; enrichment of the model through the consideration of more advanced design constraints (trace width, spacing, manufacturing constraints, and signal integrity); and extension to multilayer routing with via management. In addition, an evaluation on large-scale industrial circuits would make it possible to assess the robustness and scalability of the method.

In conclusion, this work shows that combining a deterministic A* router with a metaheuristic optimisation of the connection order constitutes an effective and

flexible approach for improving the automatic routing of single-layer printed circuit boards. It confirms that nature-inspired computational intelligence can play a significant role in the optimisation of electronic design problems.

BIBLIOGRAPHY

- [1] N. Adzhar et al. Simulated annealing technique for routing in a rectangular mesh network. *Modelling and Simulation in Engineering*, 2014:127359, 2014.
- [2] Saman M. Almufti. *Metaheuristics Algorithms: Overview, Applications, and Modifications*. Deep Science Publishing, 2025. Open Access — CC BY-NC.
- [3] Heitor Teixeira de Azambuja, Luiza de Macedo Mourelle, and Nadia Nédjah. A multi-objective ant colony optimization for routing in printed circuit boards. In *Ibero-Latin American Congress on Computational Methods in Engineering (CIL-AMCE)*, 2024.
- [4] James C. Bean. Genetic algorithms and random keys for sequencing and optimization. *ORSA Journal on Computing*, 6(2):154–160, 1994.
- [5] Minsik Cho and David Z. Pan. BoxRouter: A new global router based on box expansion and progressive ILP. In *Proceedings of the 43rd ACM/IEEE Design Automation Conference*, pages 373–378, 2006.
- [6] Daniel Harabor and Alban Grastien. Online graph pruning for pathfinding on grid maps. In *Proceedings of the 25th AAAI Conference on Artificial Intelligence*, pages 1114–1119, 2011.
- [7] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.
- [8] Youbiao He, Hebi Li, Ge Luo, and Forrest Sheng Bao. A pad-focused PCB routing algorithm using polygon-based dynamic partitioning. *IEEE Access*, 2024.
- [9] Ali Asghar Heidari, Seyedali Mirjalili, Hossam Faris, Ibrahim Aljarah, Majdi Mafarja, and Huiling Chen. Harris hawks optimization: Algorithm and applications. *Future Generation Computer Systems*, 97:849–872, 2019.
- [10] Shiyu Huang, Bin Wang, Dong Li, Jianye Hao, Ting Chen, and Jun Zhu. Ranking cost: Building an efficient and scalable circuit routing planner with evolution-based optimization. In *Proceedings of the International Conference on Learning Representations*, 2024.

- [11] Shimin Li, Huiling Chen, Mingjing Wang, Ali Asghar Heidari, and Seyedali Mirjalili. Slime mould algorithm: A new method for stochastic optimization. *Future Generation Computer Systems*, 111:300–323, 2020.
- [12] Tianjun Liao, Thomas Stutzle, Marco A. Montes de Oca, and Marco Dorigo. A unified ant colony optimization algorithm for continuous optimization. *European Journal of Operational Research*, 234(3):597–609, 2014.
- [13] Jens Lienig and Jürgen Scheible. *Fundamentals of Layout Design for Electronic Circuits*. Springer, 2020.
- [14] Seyedali Mirjalili. SCA: A sine cosine algorithm for solving optimization problems. *Knowledge-Based Systems*, 96:120–133, 2016.
- [15] Seyedali Mirjalili, Amir H. Gandomi, Seyedeh Zahra Mirjalili, Shahrzad Saremi, Hossam Faris, and Seyed Mohammad Mirjalili. Salp swarm algorithm: A bio-inspired optimizer for engineering design problems. *Advances in Engineering Software*, 114:163–191, 2017.
- [16] Seyedali Mirjalili and Andrew Lewis. The whale optimization algorithm. *Advances in Engineering Software*, 95:51–67, 2016.
- [17] Francisco Perdigones and José Manuel Quero. Printed circuit boards: The layers’ functions for electronic and biomedical engineering. *Sensors*, 22(7):2522, 2022.
- [18] Rohit Salgotra, Pankaj Sharma, Saravanakumar Raju, and Amir H. Gandomi. A contemporary systematic review on meta-heuristic optimization algorithms with their MATLAB and Python code reference. *Archives of Computational Methods in Engineering*, 31:1749–1822, 2023.
- [19] Khushro Shahookar and Pinaki Mazumder. An overview of placement and routing algorithms for PCB, VLSI, and MCM designs. Technical report, Southern Methodist University, 1995.
- [20] Nathan R. Sturtevant. Benchmarks for grid-based pathfinding. In *IEEE Transactions on Computational Intelligence and AI in Games*, volume 4, pages 144–148, 2012.
- [21] Yongxing Wang. Research on automatic PCB routing strategy based on ant colony algorithm. *Archives Des Sciences*, 74(S1), 2024.
- [22] Tan Yan, Qiang Ma, and Martin D. F. Wong. Advances in PCB routing. *IPSI Transactions on System LSI Design Methodology*, 5:14–22, 2012.
- [23] Nikola Zlatanov. PCB design process and fabrication challenges. *ResearchGate*, 2012.