University of Johannesburg

Masters Dissertation

A NEAT Inspired GEP Algorithm

Author:

Louis John Hassett

Supervisor:

Prof. Duncan A. Coulter

Co-supervisor:

Daniel Ogwok

A dissertation submitted in fulfillment of the requirements

for the degree of Master in Computer Science

in the

Faculty of Science

Academy of Computer Science and Software Engineering

"It is not the strongest of the species that survives, nor the most intelligent that survives. It is the one that is most adaptable to change. In the struggle for survival, the fittest win out at the expense of their rivals because they succeed in adapting themselves best to their environment."

Charles Darwin

Contents

II	Part Two

List of Figures

List of Tables

Part One

1. Introduction

1.1 Background

The theory of evolution by natural selection, first introduced by Charles Darwin, has profoundly influenced our understanding of the life and adaption in the natural world. Darwin's insight that species evolved over generations through the survival and reproduction of individuals with advantageous traits has not only shaped the biological sciences, but has also inspired computational models that emulate these adaptive processes (Li, Han, et al. 2024). Over millions of years, evolution has given rise to complex biological systems, among which the human brain stands as one of the most intricate. Composed of billions of neurons, the brain processes information through electrochemical signaling across complex interconnected networks, enabling perception, reasoning, and decision-making (Engelbrecht 2007). These biological mechanisms have served as a blueprint for the development of artificial intelligent systems, particular in the field of evolutionary computation and neural networks.

In computer science, evolutionary algorithms simulate the process of natural selection to solve complex optimisation problems. These algorithms operate on populations of candidate solutions, applying genetic operators such as mutation, crossover, and selection

to iteratively improve the problem's solution. In conjunction to the expanding field of evolutionary computing, artificial neural networks (ANNs) which are inspired by the structure and function of biological neurons have become foundational in machine learning. These networks consist of interconnected nodes that process information in layers, enabling machines to learn from data and perform tasks such as classification, prediction, and control (Russell and Norvig 2016). The intersection of evolutionary algorithms and neural networks has given rise to the field of neuroevolution, which seeks to evolve both the structure and parameters of neural networks using evolutionary principles.

Among the many algorithms developed within the field of evolutionary computing and neuroevolution, Gene Expression Programming (GEP) and the NeuroEvolution of Augmenting Topologies (NEAT) stand out due to their unique and complementary approaches to evolving computational structures. Gene Expression Programming (GEP) is an evolutionary algorithm that evolves computer programs or symbolic expressions. It represents solutions as linear chromosomes, which are then expressed as expression trees through an effective genotype-to-phenotype mapping scheme. This approach allows the evolution of tree-like structures in a more robust and flexible manner than traditional genetic programming techniques (Ferreira 2006). NEAT, in contrast, focuses on evolving the topology and weights of neural networks. It introduces several key innovations, including historical markings (innovation numbers) to track structural changes, speciation to preserve diversity within the population, and incremental growth of network complexity to efficiently explore the search space. These features enable NEAT to evolve increasingly sophisticated neural architectures over time (Stanley and Miikkulainen 2002).

This dissertation introduces a novel hybrid algorithm, GEP-NEAT, which seeks to combine the structural expressiveness of GEP with the adaptive topology evolution of NEAT. The motivation for developing GEP-NEAT arises from specific limitations observed in both NEAT and GEP-based neural network approaches. While NEAT has demonstrated success

in evolving neural network topologies, it suffers from computational inefficiencies, particularly due to the overhead introduced by topological sorting during network evaluation which becomes increasingly problematic as networks grow in complexity. On the other hand, GEP-NN, an approach that applies GEP to evolve neural networks, offers promising alternative by representing neural structures as expression trees, but it remains relatively underexplored in the literature and lacks the methodological maturity and empirical validation seen in other neuroevolutionary techniques.

GEP-NEAT is proposed a response to these challenges, aiming to combine the structural flexibility of GEP with the evolutionary dynamics of NEAT. At the heart of GEP-NEAT is a new representation scheme in which innovation numbers are encoded as sub-tree configurations. This approach allows for a more expressive and hierarchical encoding of neural structures, facilitating the reuse of functional subcomponents and promoting the emergence of modular architectures. By integrating GEP's symbolic representation with NEAT's evolutionary dynamics, GEP-NEAT aims to provide a more powerful and flexible tool for evolving neural networks.

1.2 Structure

This dissertation begins with this introductory chapter, which outlines the research motivation, and key research questions. It also highlights the academic contributions of the work, including publications that have emerged from the research process. Following the introduction, a dedicated chapter is presented on the research methodology, which adopts a design science approach. This chapter details the methodological framework used to guide the development, implementation, and evaluation of the proposed algorithm.

The core of the dissertation presents a comprehensive literature review, divided into three chapters. The first of these explores the foundations of evolutionary computing, providing

context for the broader field in which the work is situated. The focuses on neuroevolution, examining how evolutionary algorithms have been applied to the development of neural networks. The third chapter delves into gene expression programming, detailing its mechanisms, advantages, and relevance to the proposed approach.

After establishing the theoretical foundation, the dissertation introduces the GEP-NEAT algorithm in detail. This chapter covers the theoretical underpinnings of the algorithm, its practical implementation, and the experimental setup used to evaluate its performance. The results of these experiments are then presented and analysed, with a focus on assessing the algorithm's effectiveness, efficiency, and potential advantages over existing methods. The final chapter concludes the dissertation by summarising the key findings, discussing their implications, and outlining directions for future research.

1.3   Research Questions

The development of algorithms that evolve neural network architectures remains a dynamic and evolving area of research. While various approaches have been proposed to automate the design of neural networks through evolutionary computation, several open questions persist regarding the efficiency, expressiveness, and adaptability of these methods. This dissertation is driven by a set of research questions that aim to explore and address specific limitations in existing neuroevolutionary techniques, particularly GEP and NEAT.

The first research question investigates the structural limitations of current gene expression programming when applied to neural networks. Traditional neural networks typically include architectural features such as bias nodes and non-linear activation functions, which are essential for enhancing representational capacity. However, many implementations of gene expression programming for neural networks do not incorporate these features. This leads to the first question:

RQ1 *How can gene expression programming be extended to evolve neural networks that closely resemble traditional architectures, including the incorporation of bias nodes and activation functions?*

The second question addresses a known computational bottleneck in topology-based neuroevolutionary algorithms. Specifically, algorithms that evolve network structures often rely on topological sorting to ensure valid signal flow during evaluation. While effective, this process can become increasingly inefficient as networks grow in size and complexity. This raises the question:

RQ2 *Can the computational inefficiencies associated with topological sorting in neural network evaluation be mitigated through alternative representations or evaluation strategies?*

A third area of inquiry concerns the role of innovation numbers in NEAT. Traditionally, innovation numbers are used to track structural changes and align genomes during crossover, however, this usage is largely historical and does not contribute directly to the functional behavior of the algorithm. This lead to the question:

RQ3 *Is it possible to redefine innovation numbers in NEAT to represent meaningful and reusable structural components?*

Building on this idea, the fourth question explores the practical implications of such a redefinition. If innovation numbers can be used to encode modular structures, it is important to understand how this can be leverages to improve algorithmic performance. Thus, the next question is:

RQ4 *Provided that innovation numbers are redefined as reusable structural components, how can this representation be exploited to improve the performance, modularity, or evolutionary dynamics of the algorithm?*

The fifth question considers the broader hypothesis that combining distinct evolutionary strategies may lead to improved outcomes. Specifically, it examines whether integrating symbolic expression-based representations (GEP) with topological representations (NEAT) can result in a more effecting approach to evolving neural networks. This gives rise to the question:

RQ5   *Does the integration of symbolic-based representations (GEP) with topology-evolving strategies (NEAT) result in improved performance, scalability, or expressiveness compared to using either approach in isolation?*

Finally, the sixth question addresses a practical limitation in many symbolic neuroevolutionary systems, that is, the difficulty of evolving neural networks with multiple outputs. Many real-world tasks require networks to produce more than one output simultaneously, yet existing representations often struggle to accommodate this. This leads to the final question:

RQ6   *How can expression trees be adapted to support the evolution of neural networks with multiple outputs, and what are the implications for multi-output learning tasks?*

Together, these research questions form the foundation of this dissertation. They aim to explore the theoretical and practical challenges of evolving neural networks using symbolic and structural representations, and to investigate whether new approaches can overcome the limitations in existing methods.

1.4   Publications Resulting from this Work

A peer-reviewed conference paper derived from this research was published in the proceedings of the **8th International Conference on Information Science and Systems (ICISS 2025)**. As an established forum in its eight iteration, ICISS maintains rigorous academic standards through its double-blind peer review process, where both author and

reviewer identities are concealed to remove bias and ensure impartial evaluation based solely on scholarly merit. The conference brings together leading researchers across ten interdisciplinary tracks spanning artificial intelligence, data science, and information systems.

The accepted paper, which contributes to the Machine Learning and Artificial Intelligence track, presents the algorithm GEP-NEAT with its innovation number novelty, showcasing the ability to solve the XOR and Cart Pole problem effectively. ICISS 2025 facilitated valuable scholarly exchange through keynote presentations by field leaders, technical workshops, and interdisciplinary discussion bridging academic and real-world application. The conference proceedings are to be published into **Communications in Computer and Information Sicence (Electronic ISSN: 1865-0937 & Print ISSN: 1865-0929)** as a proceedings book volume and indexed by EI Compendex, Scopus, INSPEC, SCImago and other databases.

2. Research Methodology

This chapter outlines the methodological framework used to guide the development, implementation, and evaluation of the GEP-NEAT algorithm, a novel hybrid approach that combines Gene Expression Programming (GEP) and NeuroEvolution of Augmenting Topologies (NEAT). To ensure methodological relevance, the research is structured around the Design Science research (DSR) paradigm, which supports the iterative development of innovative artifacts to be used in real-world application. This is complemented by Olivier's insight on DSR, which provides a more refined approach in implementing DSR pragmatically. This chapter is organized in two sections. The first presents the design science methodology and application. The second discusses DSR from Olivier's perspective.

2.1 Design Science Research

Design Science Research (DSR) is a research paradigm centered on the creation of innovative artifacts that contribute meaningfully to the existing body of scientific knowledge within a specific domain. According to Hevner et al. 2004, DSR integrates the principles of relevance, rigor, and iterative design to produce solutions that are both practically useful and theoretically grounded. This methodology is particularly well-suited to algorithmic research, where the objective is to construct novel solutions and evaluate their effectiveness

through cycles of design, implementation, and refinement. In the context of this research, the artifact developed is the GEP-NEAT hybrid algorithm, which aims to address the specific limitations in existing neuroevolutionary approaches by combining the strengths of GEP and NEAT.
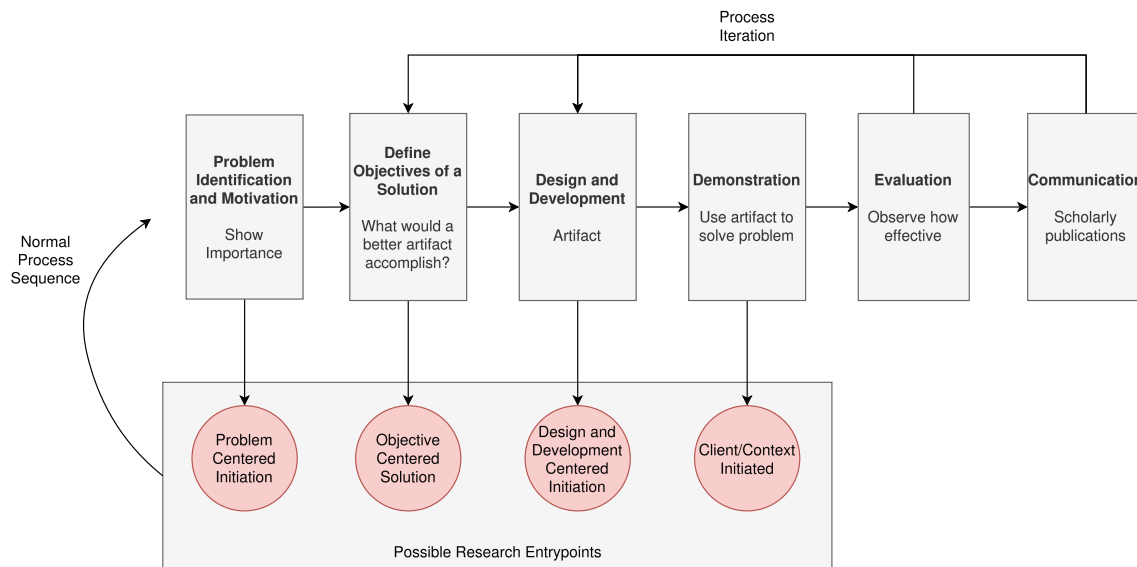
Figure 2.1: Design Science Research Process Diagram (adapted from Hevner et al. 2004)

The use of design science as the chosen research methodology can be examined as 6 process elements as follows with reference to Figure 2.1:

1. **Problem identification and motivation**: The first stage's aim is to formulate the research problem and justify the necessity of a solution. Importantly, this can be broken down into smaller problems in order for the solution to better capture the problem's complexity. Providing the problem and motivation to the reader accomplishes two things, that is, the solution to the problem is motivated to be pursued and secondly, the reader has a much better understanding of what the intention behind the conducted design, development of the prototype and its respective results are.

2. **Objectives of a solution**: The second stage of this methodology is to create a set of objectives based on the problem definition defined above. These objectives can be

either, quantitative, qualitative, or both. Quantitative objectives deal with measurable outcomes that can be expressed numerically whereas qualitative objectives are difficult to quantify and focus on the quality or nature of the solution.

3. **Design and development**: This stage deals with the creation of an artifactual solution along with detailing the artifact's functionality and architecture which will be used to create the actual artifact.

4. **Demonstration**: This stage aims to show the efficacy of the artifact to solve the problem at hand by means of ideologies such as simulation, case studies or experimentation.

5. **Evaluation**: This stage's aim is to measure essentially how well the created artifact supports a solution to the problem which involves the comparison of tried and tested real world results to that of the artifacts. As mentioned in the objective phase, a quantitative and qualitative approach can be taken; the quantitative comparison being based on quantifiable metrics, such as convergence speed, solution quality, etc., and the qualitative comparison being based on solution innovations, adaptability, ease of use, etc.

6. **Communication**: The final stage is to effectively communicate the following:

   - *Problem and it's importance*: This will essentially be the problem statement detailed along with its justification.
   - *Artifact*: An overview of the artifact.
   - *Artifact's utility and novelty*: A background and technical literature that make up the artifact will be provided to the reader.
   - *Rigor of its design*: The way in which the newly formed algorithm/design will be detailed to the reader with explicit explanation in the intricate design choices with mentions to previous results of algorithms the prototype is based on.
   - *Effectiveness*: An analysis will be done on the constructed artifact with comparison to other existing designs using quantitative and qualitative metrics in

order to showcase its use and efficacy to the research field at large.

2.2 Olivier's Insight

While Design Science Research (DSR) provides a high-level framework for the creation and evaluation of innovative artifact, it lacks a fixed set of operational steps or tools. To address this, Olivier 2009, have proposed complementary activities that can be integrated into the DSR process to support knowledge construction, problem exploration, and artifact validation. Each activity contributes to a different phase of the research process, from identifying the problem space to validating the proposed solution.

2.2.1 Literature Review

The research process begins with a comprehensive review of existing literature, which serves as the foundation for identifying gaps in current knowledge and framing the research problem. In line with Olivier's view, the literature view is not a one-time task, but an ongoing process of gathering, filtering, and synthesizing information. It enables the researcher to build a solid theoretical foundation, avoid redundant approaches, and identify opportunities for innovation.

The literature review focuses on three core areas, that is, evolutionary computation, neuroevolution, and gene expression programming. By critically analysing existing work in these domains, the review highlights the limitations of current approaches and motivates the development of a hybrid solution. In selecting sources, priority is given to peer-reviewed journal articles, followed by conference proceedings, textbook, and reputable online resources. While blogs and traditional *'google'* searches are generally treated with caution and used only as a means to support academic findings.

The literature review also serves several strategic purposes, that is, it helps to define the scope of the research problem, identify methodological approaches, avoid unproductive

directions, and uncover new lines of inquiry. In the context of DSR, it provides the initial input for the design cycle by informing the development of the conceptual model and guiding the evaluation criteria for the artifact designed.

2.2.2 Conceptual Modeling

Once the research problem is clearly defined, the next step is to develop a conceptual model that captures the essential components of the proposed solution. In this context, a model serves as a structured representation of the system or process under investigation. It helps to clarify boundaries of the solution space, and provide a blueprint for implementation.

Olivier 2009, emphasises that models can take various forms depending on the research context. They may be descriptive, metaphorical, or formal, and can be developed using principles, scientific notation, or visual languages. This research employs Unified Modeling Language (UML) diagrams to achieve this as it provides a standardised and widely accepted visual language for representing architecture and behaviour (Koc et al. 2021). The diagrams used include:

- **Class and component diagrams**: These diagrams represent the structural composition of the algorithm.
- **Activity diagrams**: These diagrams illustrate the flow of control and decision-making processes.
- **Sequence diagrams**: These diagrams show interactions between components during execution.

UML diagrams are chosen for their clarity and ability to convey complex system interactions in a digestible format, ensuring that the model is comprehensible to both technical and academic audiences. In line with Olivier 2009 perspective, models in computer research can be developed through various means, including formal specification, metaphorical representation, or practical design. In this research, the model is primarily constructed

through design, using system architecture and algorithmic logic to represent the proposed solution. Where appropriate, descriptive metaphors are used to explain abstract concepts, and formal notation is employed to define algorithmic behaviour.

2.2.3   Prototype Development

With the conceptual model in place, the next step is to construct a working prototype that embodies the proposed solution. In DSR, the prototype serves as a tangible instantiation of the model and provides a means of validating the design through various test and benchmarking mechanisms. As noted by Olivier 2009, a prototype is not merely a demonstration tool but a vehicle for inquiry. This allows the researcher to explore the behaviour of the system, identify limitation, and refine the design based on feedback.

Prototypes are also recognized as essential tools for reducing uncertainty and improving design outcomes. Camburn et al. 2017, highlight that prototyping enables real-time feedback, supports iterative development, and facilitates the early identification of design flaws. This allows researchers to test algorithmic behaviour under realistic conditions and to make data-driven decisions about further development. Other research indicates that prototypes provide proof by construction, offering concrete evidence that a theoretical model can be realized in practice. They simply serve as a foundation for further experimentation and analysis, particularly in exploratory research where the goal is to uncover new insights or validate emerging paradigms (Nunamaker Jr, Chen, and Purdin 1990).

2.2.4   Experimental Evaluation

The final activity in the research process is the empirical evaluation of the prototype. Olivier 2009, emphasises that experiments in computing research can serve multiple purposes, that is, they can be used to test hypotheses, explore parameter spaces, or validate theoretical models.

The evaluation of the prototype is approached through both quantitative and qualitative methods, in line with DSR principles of the artifact being rigorously tested to validate the effectiveness in solving the identified problem. Quantitative evaluation in this research involves measuring the algorithm's performance using standard metrics such as accuracy, precision, convergence speed, and computational efficiency (Gregar 2023). Complementing this, qualitative evaluation focuses on the artifacts structural and functional qualities such as modularity, innovation, and scalability. Olivier 2009 notes that qualitative insights are crucial for understanding how well a prototype aligns with its conceptual model and whether it contributes meaningfully to the body of knowledge.

2.3 Conclusion

By integrating literature review, conceptual modeling, prototype development, and experimental into the DSR framework along with Olivier's insight, this research adopts a comprehensive and methodological rigour approach to artifact construction. Each activity contributes to a different phase of the design cycle and supports overarching goal of developing a novel, effective, and theoretically grounded solution to the problem of evolving neural network architectures. The result is a research process that is capable of producing meaningful contributions to both theory and practice.

3. Evolutionary Algorithms

This chapter provides the reader with a comprehensive overview of what evolutionary algorithms entail along with a representative selection of existing work. An introduction to evolutionary algorithms is first given in Section 3.1. Section 3.2 paints a picture of the historical background of evolutionary algorithms. The reader is then provided with fundamentals behind evolutionary algorithms in Section 3.3. Section 3.4 reviews the different evolutionary algorithms that exist and their differences, and finally Section 3.5 provides an overview of current advancements in the field of evolutionary computation.

3.1 Introduction

Evolutionary Algorithms (EAs) belong to a large group of optimisation methods that take inspiration from Darwinian biological evolution (Eiben and Smith 2015). These methods replicate processes like natural selection, mutation, and reproduction. They simulate the adaptive and competitive dynamics observed in nature, where a set of candidate solutions, called the population, is iteratively refined over multiple generations (Li, Han, et al. 2024).

In order to mimic the lifecycle of organisms, selecting a representation of a candidate solution that is amenable to algorithmic manipulation is a prerequisite. With the proper

representation of a candidate solution in place, EAs follow the general framework of mimicking the lifecycle of organisms. To explain simply, this framework works firstly by generating an arbitrary number of handful candidate solutions that make up the initial population. Each candidate solution will have an attempt at solving the problem at hand after which this attempt will be scored and recorded relative to a particular fitness. This scoring scheme is achieved by a *fitness function* and generally has the primary role of determining how strong an individual is at solving the problem at hand. Each candidate solution will then go through a selection process whereby the best solution will be chosen, and the others disregarded, ensuring that only the best-performing candidates pass their desirable traits onto offspring and consequently forthcoming generations. Additionally, a mechanism is established to combine candidate solutions, facilitating the exchange of traits to enhance the diversity and quality of the population over successive iterations. Surviving candidates will produce offspring by applying genetic operators such as mutation (which introduces variability by altering the gene of offspring) and crossover (which combines the traits from two or more parents). After population initialization, this repetitive process of selection, mutation, and reproduction enhances the likelihood that the successive generations migrate towards a global optima while mitigating the chances of falling into a local optima (Mirjalili 2018).

EAs have been applied to a wide range of different fields ranging from engineering, to artificial intelligence. EAs are typically used in optimisation problems where there is a single or multiple objectives due to their ability to explore the solution space in a flexible manner. EAs have application in a vast number of fields where for example EA algorithms were used to optimise solar panel layouts in real-time by aircrafts in the engineering realm. Another example was EAs involvement to predict flood routing in natural channels using gene expression programming (GEP) in the environmental field (Slowik and Kwasnicka 2020).

3.2 Historical Background

EAs go back to the early computational era of the 1950s and 60s where researchers began to draw inspiration from Darwinian principles of natural selection (Petrowski and Ben Hamida 2016). These researchers managed to mathematically model the competitive and adaptive processes found in nature in order to solve and optimise complex and multi-dimensional problems. The idea was simple - just as organisms in nature evolve to adapt according their environment, organisms in the mathematical realm (candidate solutions) can *"evolve"* by competing and improving over time, akin to natural generations. This approach in solving optimisation problems became compelling to be used in problems that were ill-defined or had far too many variables that lead to exhaustive searching and suboptimal solutions.

EAs began with contributions from many important people. British mathematical and geneticist Alex Fraser took early steps to model genetic ideas using computers in the 1950s (Links and BookDust 2002). Hans-Paul Schwefel and Ingo Rechcenberg were inspired by his work and soon developed evolution strategies in Germany during the 1960s primarily for application in engineering (Petrowski and Ben Hamida 2016). Around that time, an American Scientist, John Holland, formulated the concept of genetic algorithms (GAs) which established a formal framework to simulate biological evolution in a mathematical sense. Holland's work in the 1970s revolutionized EAs extending their application in many areas (Petrowski and Ben Hamida 2016). After the publication of the book, *Genetic Algorithms in Search, Optimisation and Machine Learning* by E. Goldberg in 1989, interest in the field became much more widespread (Petrowski and Ben Hamida 2016).

As the field of computers evolved in terms of technological sophistication and breadth of application, so did the capabilities and appeal of evolutionary algorithms. Due to advancements in computer science in the late 20th century, researchers began to explore

more complex and sophisticated EA models. Also PhD student of John Holland, John Koza, a computer science researcher, formulated genetic programming which opened new avenues for using evolution to optimise nonlinear structures such as symbolic regression functions and treelike structures (Koza 1994). Differential Evolution (DE) was introduced in the 1990s which provided a mechanism in solving problems suited for continuous parameter optimisation (Das and Suganthan 2010).

3.3   Core Concepts in Evolutionary Algorithms

EAs are built from several core concepts that mimic Darwinian principles, that is, the process of selection, variation, and inheritance. These core elements form the backbone of the way in which EAs operate, creating a system whereby candidate solutions are evolved over iterative generations to meet or exceed a given objective. Understanding these core concepts are crucial in understanding the mechanics and flexibility of EAs, as each component dictates how the algorithm explores and optimises the search space.
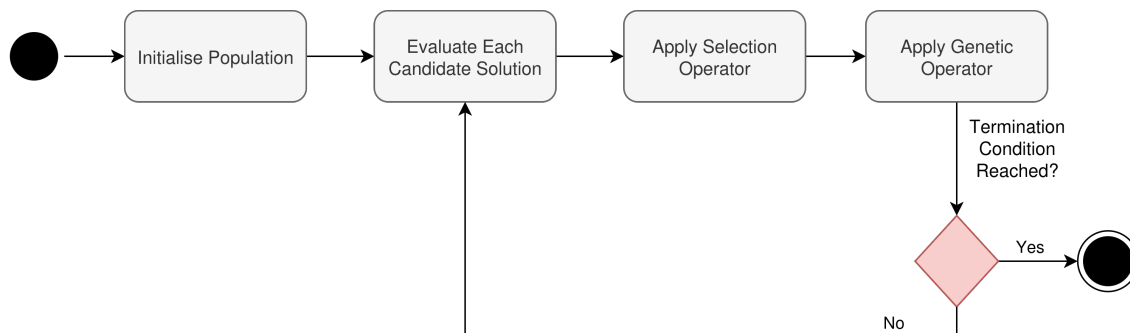

Figure 3.1: Evolutionary Algorithm Framework (adapted from Wirsansky 2024)

3.3.1   Representation

The term chromosome is often used to refer to representations of candidate solutions within the population which are a critical aspect of the evolutionary algorithms, an attempt to model genes and chromosomes seen in nature. The representation dictates how individuals

are encoded, manipulated, and evaluated within the evolutionary process. A representation that is well-defined within the context of the optimisation problem enhances the algorithm's ability to explore the search space efficiently while maintaining diversity and feasibility among solutions.

The genotype-phenotype mapping translates encoded information into a meaningful solution. There are two forms in an evolutionary algorithm, namely, the **genotype** which is the encoded representation of a solution and is typically manipulated directly by genetic operators such as crossover and mutation, and the **phenotype** which is the decoded, functional representation of the solution which is what gets evaluated against the fitness function or objective (Kramer 2017). For example, consider an optimisation problem where a 4-bit binary number that decodes the highest integer value needs to be found. The genotype would be a chromosome representing a 4-bit string (e.g., 1010). The phenotype would be the decoded integer value (e.g., $1010 \rightarrow 10$ in decimal). It is not necessarily for every optimisation problem to have an explicit genotype-phenotype mapping, in particular, when the genotype is the solution itself (Kramer 2017).

There are many ways in which chromosomes can be represented as solutions to the problem at hand. Representations are divided into two categories, namely, direct representations and indirect representations. Direct representations are those in which the genotype-phenotype mapping is one-to-one, allowing genetic operators to be applied directly without transformation. In contrast, indirect representations require an additional decoding step to map genotypes to phenotypes before genetic operators can be applied (Rothlauf 2009). Genetic algorithms have originally been proposed to be binary coded to mimic the genetic material of natural organisms, however, different representations are used based contextually on the optimisation problem at hand (Yu and Gen 2010). Common representations include those seen in Figure 3.2. Binary representations are commonly used for problems with discrete or combinatorial solutions, such as feature selection or Boolean optimisation, where each

bit can encode the presence/absence of a feature or a binary decision. Integer or real-valued representations are suited for numerical optimisation tasks, such as parameter tuning in machine learning or engineering design, where variables require fine-grained precision. Tree-based representations model hierarchical structures, making them ideal for symbolic regression, program synthesis, or parsing tasks, where solutions naturally map to decision trees or recursive expressions.
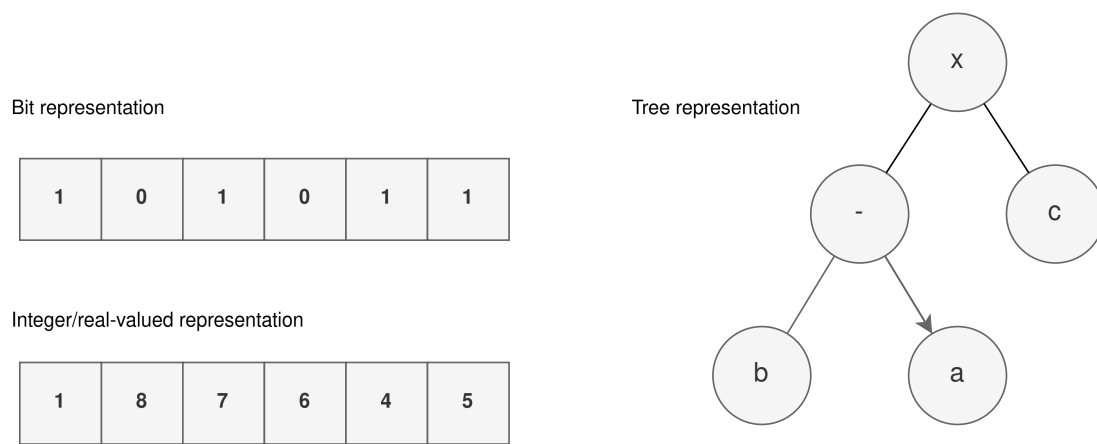

Figure 3.2: Common Gene Representation (adapted from Li, (Shoufe), and Zeng 2024)

3.3.2 Population and Fitness

The starting point in any evolutionary algorithm is the initialization of a population as shown in Figure 3.1 which represents a group of candidate solutions that could potentially solve the problem at hand. Unlike traditional optimisation techniques which mainly focus on a single-solution pathway, EAs work within the entire solution space by means of parallelism (Kramer 2017). To put it simply, each generation evaluates candidate solutions as potential answers, with the algorithm progressively identifying better ones through selection and variation. This parallel search ensures that local optima are avoided, an issue which high-dimensional optimisation typically face. By ensuring that the initial population generated is diversified, there is an increased chance of finding a global or near global optimal solution. There are many methods for initializing a population, with randomized

approaches being the most common, specifically those drawn from a uniform probability distribution, to ensure a broad coverage of the search space (Maaranen, Miettinen, and Penttinen 2007). Other mechanisms for initializing the population include heuristic and intelligent based approaches when existing solutions are known which could potentially ensure a better coverage spread of the search space (Maaranen, Miettinen, and Penttinen 2007). When taking advantage of using existing solutions to explore the search space, repetitive computations are mitigated yielding in an increased efficiency. Likewise, non-random initial populations can directly gear the algorithm to search a particular region of the solution space from the beginning which again, with the aid of known solutions, can allow the algorithm to converge quicker towards an optimal solution. Other mechanisms include deterministic based approaches in which candidate solutions are generated in a more controlled and evenly distributed manner leading to a better coverage of the solution space (Tharwat and Schenck 2021).

In genetic algorithms, diversity refers to the variety of genetic material (e.g., different solution features or values) within a population. The population's diversity closely relates to how well the candidate solutions explore the search space. If individuals are too similar, they lack diversity leading to premature convergence causing stagnation around a suboptimal solution. Conversely, if the individuals are too diverse from one another, convergence can be delayed meaning that more generations are needed in order to hone in on an optimal solution (Kramer 2017). A fine balance is therefore needed between exploration and exploitation. A highly diverse initial population is crucial to ensure thorough exploration of the search space early in the optimisation process. As generations progress, the algorithm selectively converges toward higher-quality solutions, naturally reducing diversity while refining the best candidates. There are well studied mechanisms which can control this balance such as mutation rate adjustment, niche-reserving mechanisms, or even hybrid approaches. Using these mechanisms ensures that both diversity and convergence can be

capitalized.

With reference to Figure 3.1, after the population is initialized, each candidate solution will be vetted by means of a fitness function which is a mathematical measure that scores how well the candidate solution meets the objective (Kramer 2017). This score reflects the quality of the candidate solution in regard to the optimisation goal and is used to guide the selection process in subsequent generations. For example, given an optimisation problem such that x is an integer to maximize the parabolic equation $f(x) = -x^2 + 25$, the fitness function would be taking a candidate solution, and feeding it as input into this equation. Evidently, the maximum fitness to this function would be 5. Fitness functions are defined in different ways depending on the application at hand, for example, in engineering design problems, fitness may represent efficiency, structural stability, or cost-effectiveness, whereas in artificial intelligence, fitness could measure a predictive accuracy. The way in which a fitness function is defined shapes how effectively the EA converges towards an optimal solution, especially in multi-objective problems (MOPs) where the fitness needs to account for multiple goals (Eiben and Smith 2015).

MOPs are mathematical problems that involve optimizing multiple conflicting objectives at the same time. The fitness function in this regard is defined by the algorithm's performance in relation to these conflicting objectives as opposed to one (Kramer 2017). A common approach in formulating the fitness function is by aggregating all the objectives in a singular fashion by means of a weighted sum (Kramer 2017). Another approach in defining the fitness function appropriately is by changing the structure of algorithm by means of co-evolution - instead of evolving a single population, the problem is broken down into smaller problems which in turn have their own population which is evolved (Zhou et al. 2011). In multi-objective optimisation problems which are common in economics and engineering, the fitness function may use methods like Pareto front optimisation which balances competing objectives without forcing single trade-off solutions. Pareto optimality

The fitness function needs to be discussed much earlier than this" → I am trying to follow the thought process of Figure 3.1 by following the flow diagram. I thought this may be easier to introduce for the reader.

describes a solution that cannot be improved in any objective without degrading another. In these kind of scenarios, EAs evolve a population of solutions that collectively represent a spectrum of optimal trade-offs, giving the decision-maker flexibility in choosing the most suitable solution based on contextual priorities (Rachmawati and Srinivasan 2009). EAs are therefore robust and versatile in solving both single and multi-objective problems which can often be riddled with narrowly defined and complex optimisation challenges.

3.3.3 Genetic Operators

In EAs, genetic operators - crossover and mutation - serves as crucial mechanisms by means of variation and adaption to guide the exploration of the solution space while refining existing solutions. Variation in evolution can be divided into two types based on their arity, namely, unary (mutation) and n-ary (recombination) both of which produce offspring (Eiben and Smith 2015). These operators mimic the evolutionary processes seen in nature among organisms, where gene recombination and mutation introduce new traits that contribute to a population's overall fitness and adaptability (Wirsansky 2024).

3.3.3.1 Crossover

Crossover is an evolutionary process based on sexual reproduction whereby genetic material from two parents are combined to produce offspring that inherit characteristics from both (Kramer 2017). Crossover plays a crucial part in diversifying the algorithm's search path by combining beneficial traits and thereby ideally producing offspring with improved fitness. There are different ways in which crossover can occur which provide varying levels of control over the inheritance process. For simplicity, consider linear gene representations for the following crossover mechanisms.

Single-point Crossover selects a single point along the chromosome and exchanges genetic material on either side as shown in Figure 3.3. This simple approach works well when minor structural adjustments results in significant improvement (Li, (Shoufe), and
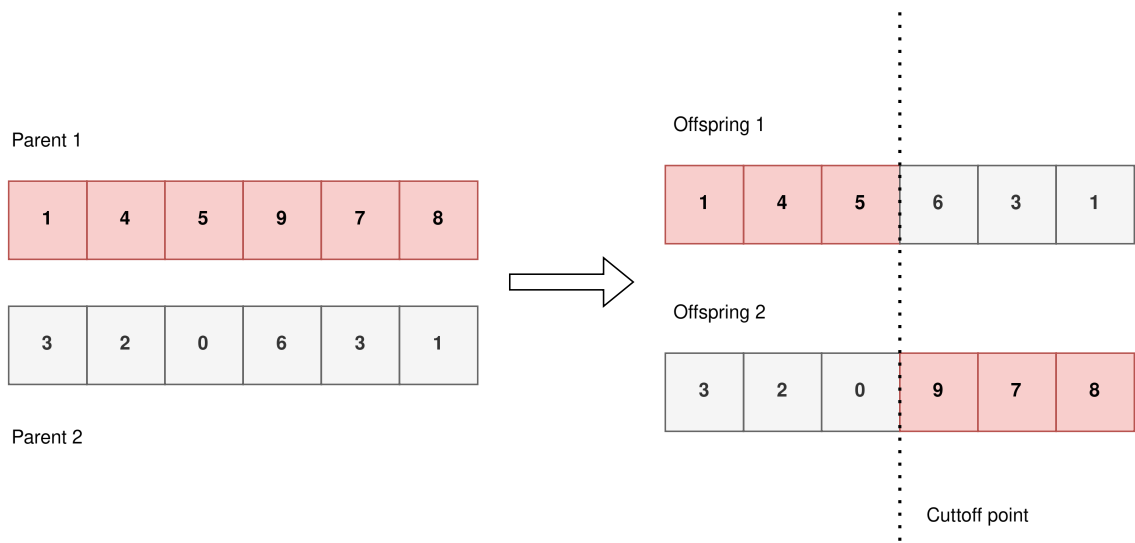
Zeng 2024).

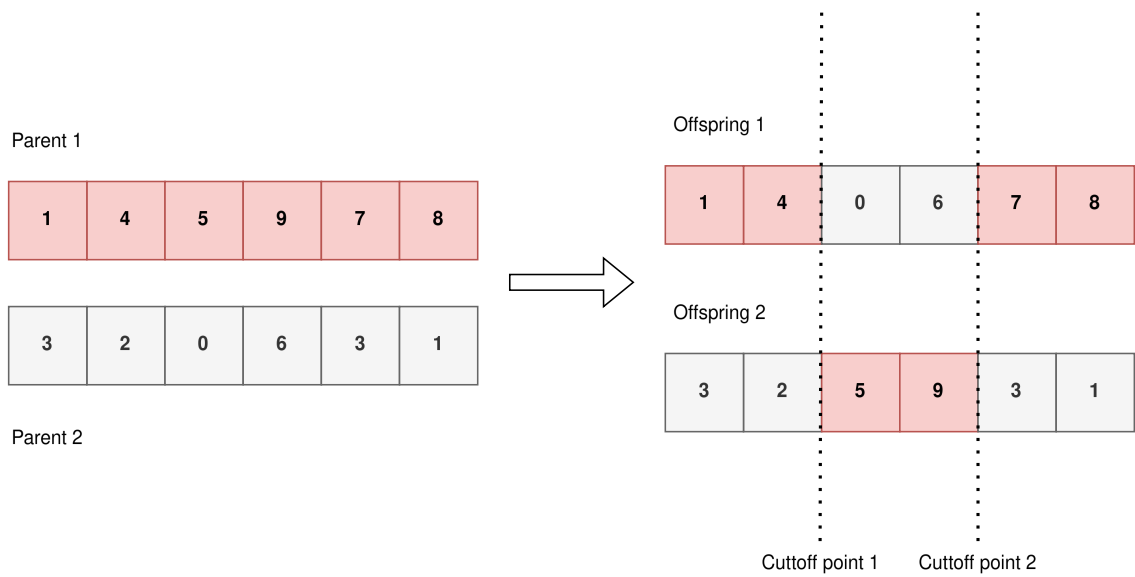Figure 3.3: Single-point crossover

Figure 3.4: Multi-point crossover

Multi-point crossover introduces more than one cutoff point, allowing for more complex combinations of genetic material from each parent thereby creating a more diverse offspring

as shown in Figure 3.4 (Zainuddin, Abd Samad, and Tunggal 2020). Multi-point crossover is typically used in scenarios that are non-linear.

Uniform crossover selects each gene randomly from one of the corresponding genes of the parent as shown in Figure 3.5 (Zainuddin, Abd Samad, and Tunggal 2020). This technique produces highly diversified offspring and is suitable when seeking greater diversity in the population thereby leveraging exploration.
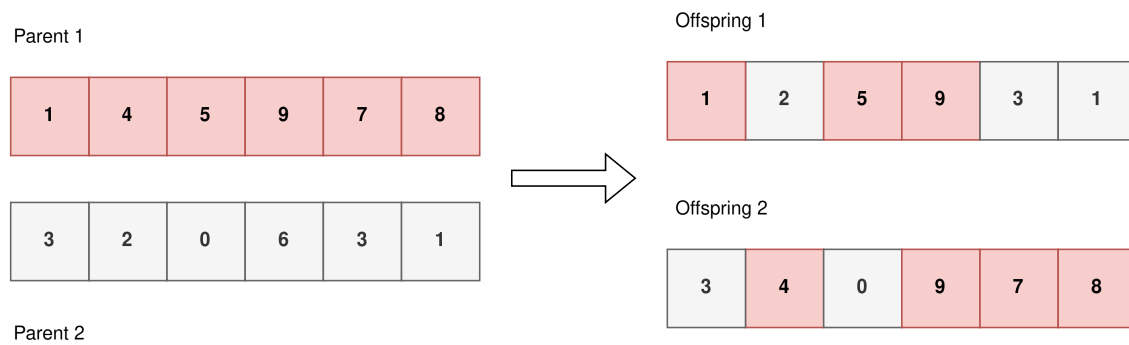
Figure 3.5: Uniform crossover

In scenarios where problem requirements are specific, crossover strategies can be tailored to meet particular constraints. For example, order crossover (OX1) is useful for optimisation problems such as the traveling salesman problem whereby maintaining certain order relationships between genes (cities) is critical (Larranaga et al. 1999). Such customized crossover methods ensure that the offspring retain problem-specific constraints while still inheriting the most desirable traits from parent solutions. Beyond sexual recombination, horizontal gene transfer (HGT), inspired by bacterial gene exchange, can also diversify populations by transferring genetic material directly between solutions without parent-offspring relationships (Rafajłowicz 2018).

3.3.3.2 Mutation

Mutation in contrast to crossover introduces randomness in the algorithm by altering one or more genes in an individual's chromosome in order to create new traits and variation (Mirjalili 2018). Mutation serves as the primary source of diversity within the population and important when solutions begin to become too familiar, a condition known as *genetic drift* (Ahn 2006). When individuals become too similar, this results in stagnation around suboptimal solutions causing convergence towards a local maxima. Mutation can occur in many ways, such as:

Bit-flip mutation (aka bitwise mutation) is applied to chromosomes represented in binary whereby each gene has a probability that its bit is inversed (changing from 0 to 1 or vica-versa) as shown in Figure 3.6 (Li, (Shoufe), and Zeng 2024).
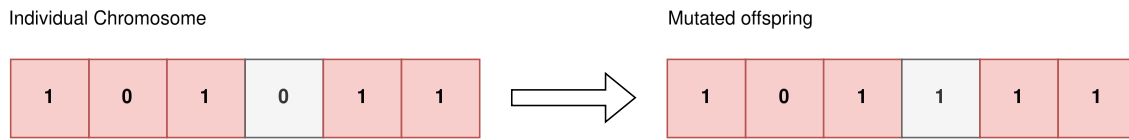
Figure 3.6: Bit-flip mutation

Gaussian mutation is applied to real valued chromosomes whereby a random gene is selected and a Gaussian distribution applied by means of the function below where x_i^i is the offspring, σ is a fixed parameter for all variables, $[b_i, a_i]$ represents the range of random gene, erf^{-1} represents the inverse of erf which is the Gauss error function, defined by $erf(y) = \frac{2}{\sqrt{\pi}} \int_0^y e^{-t^2} dt$ (Bell 2022, Hinterding 1995). Gaussian mutation allows for controlled but continuous alterations.

$$x_i' = \sqrt{2}\sigma(b_i - a_i)erf^{-1}(u'i) \tag{3.1}$$

Inversion mutation selects a subset of genes and inverts the position as shown in Figure

3.7. This mutation scheme is useful in combinatorial optimisation problems where the specific arrangement of elements matters such as the travelling salesman problem (Li, (Shoufe), and Zeng 2024).
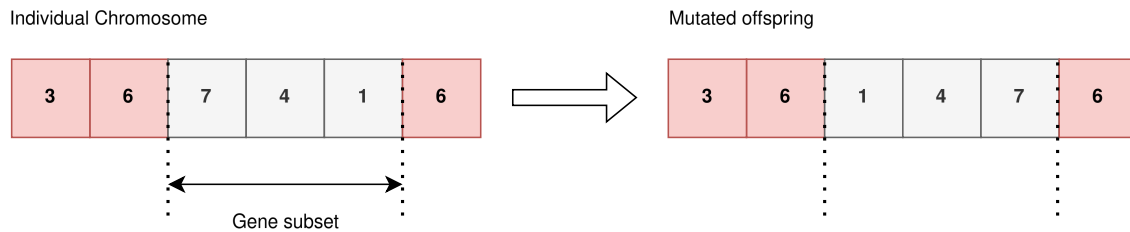

Figure 3.7: Inversion mutation

The parameters that govern genetic operators do not have to be static. There is more complex EAs, these parameters can be adjusted dynamically as the algorithm progresses. Adaptive mutation rates for example can be useful in maintaining diversity during later stages of the search when converging towards an optima is prioritized. Likewise, crossover rates could increase when diversity is low in order to ensure that favourable genetic material continues to recombine as the algorithm refines its search (Meyer-Nieberg and Beyer 2007). In hybrid approaches, self-adaptive genetic operators are used whereby the parameters of mutation and crossover evolve and form part of the chromosome, allowing the EA to essentially learn optimal rates over time based on the fitness at hand (Meyer-Nieberg and Beyer 2007).

Hybrid approaches in genetic operators have become popularized due to its powerful nature. For example, memetic algorithms combine the typical genetic operators described above with local search techniques which allow candidate solutions to further refine the search space by means of exploiting problem-specific knowledge or heuristics (Neri and Cotta 2012). This combination is useful in complex landscapes where small local improvements can lead to an increased fitness score after major variations have been introduced through

crossover and mutation in individuals. Hybrid genetic operators can therefore enhance the EAs ability to explore local optima thoroughly while still retaining a broad search capacity.

There is a fine balance between crossover and mutation in the efficacy of EAs. Mutation ensures that the EA explores new possibilities and avoids premature convergence while crossover allows the search space's best regions to be explored by recombining favourable traits of candidate solutions. These operators form a dynamic balance between exploration and exploitation, equipping EAs with the necessary flexibility to solve complex and non-linear optimisation landscapes.

3.3.4 Selection Mechanisms

Selection occurs in an evolutionary process in two instances - parent selection and survivor selection. Parent selection has its primary purpose in distinguishing individuals based on their quality in order to allow better individuals to become parents of the next generation (Wirsansky 2024). Survivor selection, similar to parent selection, aims to distinguish individuals apart based on quality. Parent selection occurs before variation operators are applied while survivor selection occurs after offspring are produced and evaluated against the fitness function (Eiben and Smith 2015). This selection mechanism mimics the natural selection seen in nature where animals with desirable traits have increased likelihood of producing better offspring - akin to this, candidate solutions with better fitness scores are more likely to produce better quality offspring to solve the objective at hand. An effective selection mechanism will ensure that there is a balance between promoting high-quality solutions whilst maintaining population diversity ideally driving the EA towards a global optimum. Below are common selection mechanisms that are typically used within evolutionary algorithms:

Fitness-proportionate selection which is one of the simplest yet most renowned selection methods. In this selection approach, each candidate solution's probability of being selected

is proportional to its fitness score relative to the total fitness of the population (Shukla, Pandey, and Mehrotra 2015). Mathematically, the probability that each individual is selected is given by the formula below where f_x is the fitness of individual x (Shukla, Pandey, and Mehrotra 2015).

$$p_x = \frac{f_x}{\sum_{i=1}^{n} f_i} \tag{3.2}$$

This selection mechanism is computationally efficient ($O(n)$ complexity) and ensures that fitter individuals are more likely to survive and reproduce while still giving less fit individuals the chance to contribute and preserve genetic diversity thereby preventing stagnation. The most common implementation of this approach is *roulette-wheel selection*, which assigns selection probabilities analogously to slots on a roulette wheel such that higher fitness solutions occupy larger wheel segments and thus have greater likelihood of being selected. Although this process is simple, this mechanism may suffer from genetic drift if highly fit individuals dominate which results in a loss of diversity in subsequent generations. This approach raises considerations about the *takeover time*, that is, the number of generations required for the best candidate solution's genes to dominate the population under selection pressure alone. Short takeover times indicate strong selection pressure that may lead to premature convergence, while longer takeover times encourage diversity within the population at the cost of slower optimisation.

Tournament selection offers a more controlled selection mechanism which is done by firstly by selecting a subset of individuals based on random sampling. Each subset known as tournaments will have individuals compete and the strongest survive. The size of the subset or tournament impacts the selection pressure - larger tournaments increases the chances that the fittest individuals will dominate while smaller tournaments result in a more diversified selection pool of individuals (Shukla, Pandey, and Mehrotra 2015). Tournament selection is computationally efficient ($O(n)$ complexity) and provides a simple way to adjust selection pressure making it a popular choice in EAs (Shukla, Pandey, and Mehrotra

2015).

Rank-based selection addresses the limitations of the fitness-proportionate selection scheme by ranking individuals based on their fitness rather than their raw fitness score (Mitchell 1998). This would mean that individuals with exceptionally low fitness scores would have an increased chance to be selected, lowering the chances of genetic drift. This selection method has complexity $O(nlogn)$ and maintains selection pressure without skewing the algorithm aggressively towards highly fit individuals which will allow a broad exploration of the search space (Mitchell 1998).

Elitism is a technique that is generally used with other selection mechanisms to ensure that the most fit solutions are preserved across generations. In this mechanism, a hall of fame is produced by taking the top n individuals which are then carried to the next generation without modification guaranteeing that the best solutions are always kept (Mitchell 1998). This method essentially prevents the best solutions from being lost due to changes made by genetic operators thereby accelerating convergence. Although elitism speeds up the rate at which the algorithm will converge, it can lower diversity leading into premature convergence on a suboptimal solution (Malik and Wadhwa 2014).

Niche and speciation techniques are employed to encourage diversity in specific traits by creating subpopulations, or niches within the larger population, however, may be computationally more taxing (complexity $O(n^2)$) as opposed to the aforementioned selection techniques (Bäck, Kok, and Rozenberg 2012). Speciation can be categorised into four types based on the geographical result that manifested the speciation (Cilliers 2022):

- **Allopatric speciation** occurs when the population is divided and no interactions occur between the resulting subpopulations. Over many generations, the groups start to diverge until they can be distinguished apart to form their own new species.
- **Peripatric speciation** occurs when a portion of the population migrates away, form-

ing their own subpopulation. Unlike allopatric speciation, individuals from different subpopulations can still interact with each other. Over successive generations, these groups diverge to become their own separate species.

- **Parapetric speciation** is the similar to peripatric speciation with the difference that some interaction can still occur between separated subpopulations. Although the interaction between groups are limited, over time the groups can still diverge to create separate species.

- **Sympathetic speciation** occurs when a group forms within the population still occupying the same geographical space. Over many generations, the group is still likely to diverge to form part of a separate species, however, with the added benefit of interacting with the same pool of individuals during the selection and reproduction stages.

To ensure effective speciation and to thoroughly explore the search space, maintaining a diverse population is essential. The diversity can b categorised at three distinct levels, namely, the gene level, chromosome level, and the population level. At the gene level, variation is evaluated at each genetic locus across the entire population. At the chromosome level, the focus shifts to the diversity within individual chromosomes throughout the population. Finally, at the population level, diversity is considered by examining the distribution of each bit across all chromosomes in the population (Diaz-Gomez and Hougen 2007). To support this diversity and prevent premature convergence, various diversity management techniques are applied. These techniques are designed to encourage exploration and maintain a healthy spread of genetic material throughout the population, and they typically fall into several categories. These can be categorised as follows with an example mechanism (Segura et al. 2016):

- **Selection-based**: These methods regulate the selection pressure during the parent selection phase. Diversity is typically preserved by adjusting parameters such as takeover time and selection intensity. While these techniques can effectively maintain

diversity in the short term they often struggle to sustain it over successive generations due to the homogenising effects of crossover operations (Blickle and Thiele 1996).

- **Population-based**: These approaches modify the structure of the population itself, often by introducing subpopulations or altering mating restrictions. A common example is the use of island models which is discussed below.

- **Crossover/mutation-based**: In this category, diversity is maintained by imposing constraints on genetic operators such as crossover and mutation. For instance, the Cataclysmic mutation scheme (CHC) avoids mating individuals that are too genetically similar and periodically reintroduces variation through reinitialisation mechanisms (Segura et al. 2016).

- **Fitness-based**: These methods modify the way fitness is calculated to encourage the survival of diverse individuals. A well-known example is fitness sharing which is discussed below.

- **Replacement-based**: These strategies focus on how individuals are replaced in the population. A common method is elitism, which as previously discussed, ensures that the best-performing individuals are retained across generations. While generally used to preserve high-quality solutions, elitism can also contribute to diversity by preventing the premature loss of valuable genetic material.

Research has yielded many techniques in order to stimulate niching within evolutionary algorithms:

- **Fitness Sharing** which operates on the principle that fitness is a shared resource among individuals in an attempt to reduce redundancy within a population or search space (Bäck, Kok, and Rozenberg 2012).

- **Clearing** which is a mechanism whereby the top individuals per niche can take primary advantage of the resource of that particular niche (Bäck, Kok, and Rozenberg 2012).

- **The Islands Model** which is a mechanism inspired directly from organic evolution

whereby the population is divided into multiple subpopulations and evolved independently. During evolution, gene information is exchanged between subpopulations in an attempt to encourage trait migration in order to exploit a subpopulations search space (Bäck, Kok, and Rozenberg 2012). A recent development in island model approaches is the CLISDE model, which applies a nested island genetic algorithm to evolve decision trees for classification tasks. By evolving separate populations of decision trees in parallel and periodically exchanging information, the model enhances diversity and improves classification performance (Cullinan, Coulter, and Appel 2023).

3.4 Types of Evolutionary Algorithms

EAs cover a large set of a large set of optimisation approaches that share the fundamental principles of population-based search, genetic variation, and selection, mimicking evolutionary processes. These algorithms have morphed into distinct types each aimed to be used in a particular class of problems. Well known evolutionary algorithms include genetic algorithms, evolution strategies, genetic programming and differential evolution. While all follow the core principles of evolutionary processes, each type is uniquely structured to address specific challenges in optimisation, making EAs a versatile tool for solving for a many complex problems.

3.4.1 Genetic Algorithms

Genetic Algorithms (GAs) are one of the most well known types of EAs, initially formalized by John Holland in the 1970s (Mitchell 1998). In traditional GAs, candidate solutions are represented as chromosomes with binary gene values (bit strings), however, these have now been extended to include real values representations as well. These chromosomes undergo genetic operations like crossover and mutation to produce new offspring for subsequent generations. The basic genetic algorithm follows the basic framework shown

in Algorithm 1.

Algorithm 1 Basic Evolutionary Algorithm that Genetic Algorithms follow (Bäck, Kok, and Rozenberg 2012)

1: Initialise population of individuals

2: Evaluate all individuals by fitness function

3: **repeat**

4:     Select individuals (parents) for reproduction

5:     Vary selected individuals in randomized manner to obtain new individuals (offspring)

6:     Evaluate offspring by fitness function

7:     Select individuals for survival from offspring and possibly parents according to fitness

8: **until** stopping criterion fulfilled

Traditional optimisation algorithms typically struggle with high-dimensional or noisy problem spaces, however, EAs find their solution for a given problem by dynamically adapting their search pathways. Solving for the solution is similarly tautological making EAs favorable when confronted with complex, or poorly-defined problems where the normal traditional algorithms struggle to explore the solution space as intricately (Li, (Shoufe), and Zeng 2024).

GAs have been proven to be effective in both discrete and combinatorial optimisation problems where the solution space is vast and discontinuous (Eiben and Smith 2015). Due to the flexibility by which the chromosomes can be represented, GAs are widely used in engineering design, machine learning, and scheduling problems where they can efficiently explore and exploit the search space at hand which can be of varying complexity (Bäck, Kok, and Rozenberg 2012).

One of the distinguishing features that GAs have to offer are their ability to handle discrete variables and constraints naturally. This is simply done by defining the individual chromosomes as a sequence of discrete numbers as shown in Figure 3.8. This makes GAs highly suitable in applications such as resource allocation, routing, and other combinatorial problems that require search across many permutations (Bäck, Kok, and Rozenberg 2012). The use of crossover and mutation gives GAs the flexibility to explore the search space thoroughly while selection mechanisms help guide the algorithm towards good quality solutions. As mentioned, GAs are characterised as a kind of local search which are also hybridized with other techniques (such as memetic algorithms), to refine solutions and enhance the performance of the search. While genetic algorithms are used as a general purpose approach, they still run a risk of failure. When genetic algorithms are used in optimisation problems that lack detail and are represented poorly, the algorithms' performance is directly affected (Bäck, Kok, and Rozenberg 2012).

Binary chromosome

| 1 | 0 | 1 | 0 | 1 | 1 |

Real-value chromosome

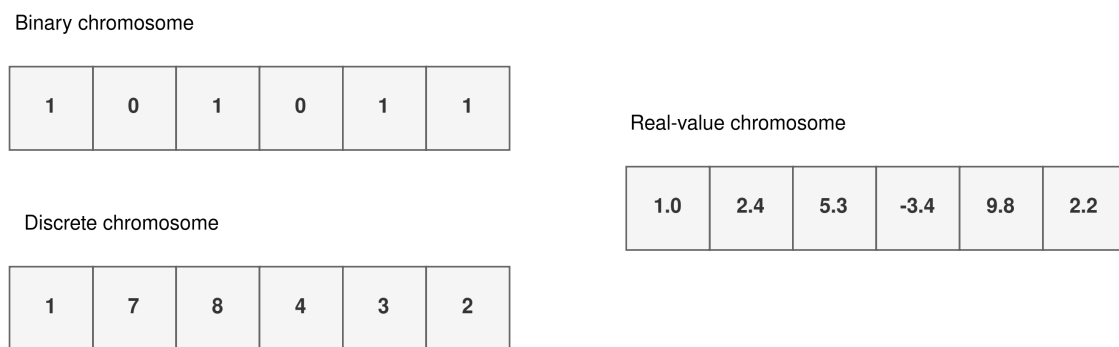| 1.0 | 2.4 | 5.3 | -3.4 | 9.8 | 2.2 |

Discrete chromosome

| 1 | 7 | 8 | 4 | 3 | 2 |

Figure 3.8: Different encoding schemes

In situations where real-valued parameters need to be encoded into binary chromosomes, a common approach is to use a representation similar to the *IEEE 754* floating-point format. (Satman and Akadal 2020). This involves encoding each real umber by separating it into three components, namely, the **sign bit** which indicates whether the number is positive or negative, the **mantissa** which represents the precision bits of the number, and the **exponent**

which scales the value. This method allows real values to be efficiently represented and manipulated within a binary representation of the chromosome.

3.4.2 Evolution Strategies

Evolution Strategies (ES) were developed by Hans-Paul Schwefel and Ingo Rechenberg in the 1960s, as a means for solving continuous optimisation problems in the engineering field (Li, (Shoufe), and Zeng 2024). Unlike traditional GAs which typically use binary or discrete representations, ES uses real-valued vectors to represent solutions making them well-suited for optimizing real-valued parameters. Evolution strategies emphasize mutation as the primary genetic operator, applying random changes to solution vectors to generate offspring while crossover plays a less pivotal role (Bäck, Kok, and Rozenberg 2012). Evolutionary Strategies operates as shown in Algorithm 2.

Algorithm 2 Evolutionary Strategy Algorithm (Bäck, Kok, and Rozenberg 2012)

1: Initialise population of individuals

2: Evaluate all individuals by fitness function

3: **repeat**

4:      Select parents for reproduction uniformly at random

5:      Vary selected individuals in randomized manner to obtain offspring

6:      Evaluate offspring by fitness function

7:      Select individuals from best ranks for survival

8: **until** stopping criterion fulfilled

ES differs from GAs in another fundamental way which is its self-adaptive approach to mutation. This self-adaption is achieved by evolving the strategy parameters alongside the chromosome, allowing ES to tune its search almost autonomously (Bäck, Kok, and Rozenberg 2012). ES has been proven effective in solving engineering design problems where precise adjustments to real-valued parameters are essential. Its ability to control bal-

ance between exploration and exploitation, especially in continuous and high dimensional search spaces makes ES a powerful tool for optimisation problems requiring fine-grain control (Eiben and Smith 2015).

3.4.3 Genetic Programming

Genetic Programming (GP) introduced by John Koza in the early 1990s, represents a unique branch of EAs that evolve tree-like structures as opposed to linear chromosomes of fixed string (Koza 1994). These trees represent executable programs, typically made up of mathematical functions, logical operations, or other computational expressions. GPs are suitable for optimisation problems such as symbolic regression, machine learning, or automated problem-solving. The main idea behind GPs are simple: evolve these program structures iteratively over generations through core evolutionary algorithm operations such as crossover, mutation, reproduction. A key distinction of GP is how fitness is evaluated which is by executing each candidate program on representative test cases with its performance directly determining its fitness value.

GP's primary strengths are its flexibility and interoperability as it allows users to analyse and modify the resulting programs directly. GPs can be applied to generate models from regression problems, develop classification systems, or even design control systems in robotics (O'Neill 2009). For example, in symbolic regression, GPs evolve mathematical expressions that fit observed data, often discovering intricate relationships that traditional regression techniques struggle to formulate (Koza 1994). Additionally, GP has been applied in fields like finance, image analysis, and automated design, where evolving decision rules or functional expressions can produce valuable insights and practical solutions (O'Neill 2009).

GP follows the same framework that was shown in Figure 3.1. A population is firstly initialized which in GP sense is by means of producing randomized trees. In order to

prevent trees that lack too much diversity and are far too complex, the *ramped half-and-half* method is used whereby a fraction of the population is initialized with trees having a maximum depth of 1, another fraction having a depth of 2, and so on (Bäck, Kok, and Rozenberg 2012). Once a population is generated, the algorithm will go through the trivial evolutionary process of selection, mutation, crossover and reproduction. Genetic operators however vary vastly in the way they are applied compared to traditional GAs. For example, when crossover occurs, this is implemented by means of subtree replacement as shown in Figure 3.9 where an entire subtree is exchanged with another to produce offspring.
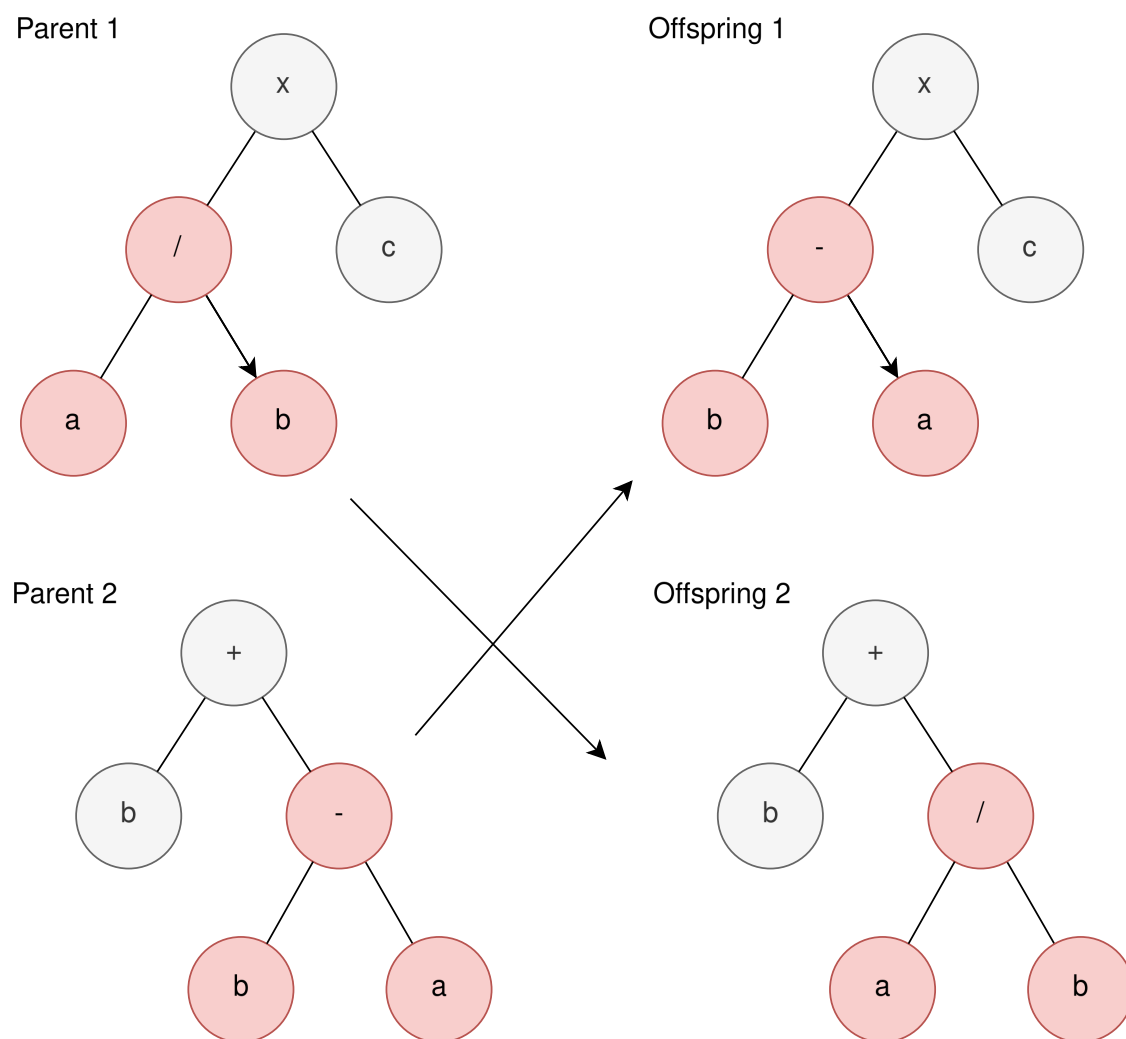

Figure 3.9: Genetic Programming Crossover (adapted from Eiben and Smith 2015)

Subtree mutation, on the other hand, replaces a randomly selected subtree with a newly generated one, introducing new expressions and broadening the search space as shown in Figure 3.10.
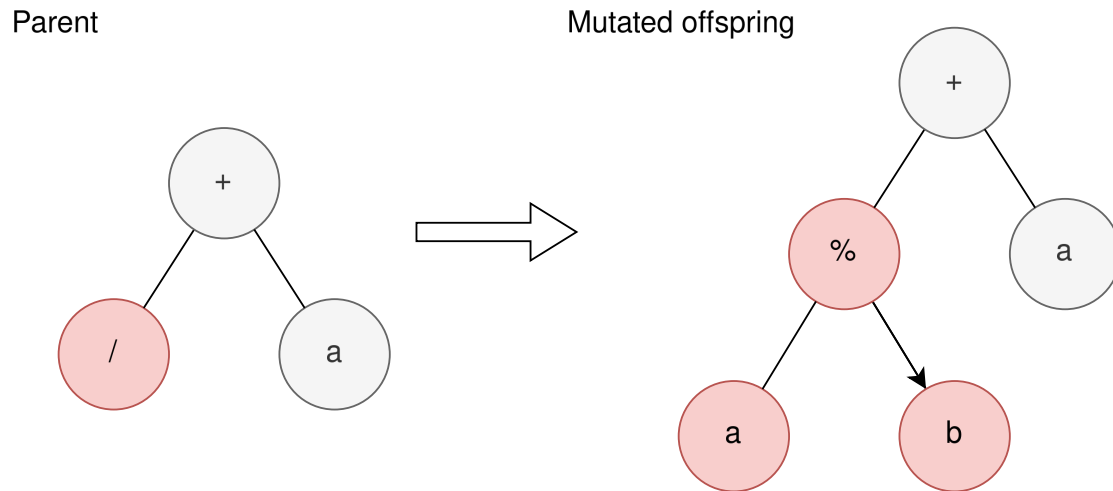

Figure 3.10: Genetic Programming Mutation (adapted from Eiben and Smith 2015)

Crossover and mutation allow GP to navigate a wide variety of functional forms and solution pathways making it well suited for problems that benefit from structural flexibility such as symbolic regression or predictive models (Bäck, Kok, and Rozenberg 2012).

One of the challenges that GP faces is in managing computational resources as evolving complex trees can be computationally intensive (Koza 1994). This means that large populations and lengthy expressions may require significant processing power making GPs slower than better suited evolutionary methods. There are many ways to counteract this, one of which is to limit the tree depth, however, come at the disadvantage of less diverse individuals (Koza 1994). More advance approaches include techniques such lexicographic parsimony pressure where selection is modified to prefer smaller trees only in situations where fitness remains the same (Luke and Panait 2002). Another challenge is the additional effort required in ensuring that the trees resulting from genetic operation are valid programs

(Koza 1994).

3.4.4 Differential Evolution

Differential Evolution (DE) offers a distinctive approach to continuous optimisation problems, particularly suited for high-dimensional search spaces with real-valued parameters (Storn and Price 1995). Unlike GAs which rely heavily on crossover and mutation of binary-coded chromosomes, DEs strength lies in its vector-based mutation scheme. The way in which it works is as follows:

1. For each parent in the population, a trial vector $u_i(t)$ is generated. This trial vector is generated using the following formula:

$$u_i(t) = x_{i_1} + \beta (x_{i_2}(t) - x_{i_3}(t)) \tag{3.3}$$

   where $x_{i_2}(t)$ and $x_{i_3}(t)$ are randomly selected individuals such that $i \neq i_1 \neq i_2 \neq i_3$. i_2 and i_3 are chosen based on uniform sampling, and $\beta \in (1, \infty)$ which is a scaling factor for controlling amplification (Storn and Price 1995)

2. After obtaining the mutated offspring which is the trial vector, crossover occurs which is implemented as follows:

$$x'_{ij}(t) = \begin{cases} u_{ij}(t) \; if \; j \in J \\ x_{ij}(t) \; otherwise \end{cases} \tag{3.4}$$

   where x'_{ij} is gene j of chromosome i, and J is the set of chosen perturbation points, which is essentially the set of trial vectors (Storn and Price 1995).

3. Once the offspring is produced from the crossover operator, the evolutionary process continues over iterative generations until termination conditions are met.

DE's main advantage lies in its simplicity, requiring only a few parameters like the mutation scaling factor and crossover rate which makes it easy to implement (Storn and Price 1995). In addition, DE's efficiency with limited parameters make it feasible for use in high-stakes,

real-time applications like control system design and power system optimisation, where quick convergence on reliable solutions is essential for maintaining system stability and performance (Das and Suganthan 2010).

Despite DEs strengths, there are challenges in balancing exploration and exploitation. For problems with highly complex landscapes, DE may converge slowly if its mutation factor is not carefully tuned (Das and Suganthan 2010). Along with this, DEs reliance on population-based differences for mutation can lead to stagnation in situations where diversity is rather low. To address this issue, adaptive differential evolution (ADE) has been developed where mutation and crossover rates adjust dynamically to encourage exploration in later stages of the generation run (Das and Suganthan 2010). Other variations of this algorithm, like multi-objective differential evolution (MODE) extends DE to handle multiple objectives simultaneously which produces a set of pareto-optimal solutions offering the user a range of trade-off options (Xue, Sanderson, and Graves 2003).

Both GP and DE illustrate its versatility within the field of evolutionary algorithms, each tailored to different use cases. While GP excels at evolving flexible and interpretable models for symbolic and algorithmic problems, DEs efficiency in continuous space make it ideal for high-dimensional, real-valued optimisation tasks. Likewise, in situations where computer programs need to be created to approximate an optimisation task, this is where GP shines. These algorithms showcase the adaptability of evolutionary principles to address a wide range of complex problems.

EAs are particularly valuable for problems where evaluating solution quality is straight-forward, but generating optimal solutions remains computationally difficult. For such problems, EAs provide an effective approach provided that:

1. the candidate solution can be represented in a modifiable format (e.g., trees, graphs, or vectors)

2. the search space permits incremental improvements through variation operators

This contrasts with traditional methods that may require explicit gradients or problem-specific heuristics (Li, (Shoufe), and Zeng 2024).

3.5 Current Research

The field of evolutionary algorithms continues to evolve, driven by advances in computation power, new problem domains, and the growing demand for more robust optimisation tools. Key trends in the evolution of this field include adaptive algorithms, EAs integration with machine learning, and the application of EAs to large-scale systems.

One promising direction is the development of self-adaptive algorithms that can dynamically adjust their evolutionary parameters during the optimisation process. These algorithms eliminate the need for manual tuning, which is often time-consuming and problem-specific. Self-adaptive mechanisms are particularly valuable in solving problems with dynamic environments, applications of which has seen to be used in scheduling problems and vehicle routing during traffic congestion (Dulebenets et al. 2018, Sabar et al. 2019).

Another interesting area of research is EA's involvement in machine learning and artificial intelligence. Evolutionary algorithms are increasingly used to optimise hyperparameters, architectures, and training processes for machine learning models, such as deep neural networks (Young et al. 2015). Beyond neural network, EAs are finding applications in automating machine learning workflows, including feature selection, model selection, and algorithm design (Nikitin et al. 2022).

With the rise of big data, evolutionary algorithms are being adapted to handle large scale datasets. Traditional EAs face computational challenges when applied to large-scale problems, but advancements in distributed computing, cloud-based frameworks, and GPU

acceleration have enabled EAs deployment on massive datasets. EAs are being tailored to solve large-scale optimisation problem in fields such as energy distribution and supply chain, an example of which was its application to schedule the power of distributed systems (Guzek et al. 2014), and another to optimise an integrated system of bioenergy production supply chains (Ayoub et al. 2009).

Try and find even more recent work.
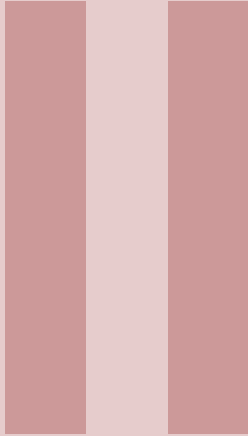
Part Two

Bibliography

Ahn, Chang Wook (2006). *Advances in evolutionary algorithms*. Springer (cited on page 38).

Ayoub, Nasser et al. (2009). "Evolutionary algorithms approach for integrated bioenergy supply chains optimization". In: *Energy Conversion and Management* 50.12, pages 2944–2955 (cited on page 55).

Bäck, Thomas, Joost N Kok, and G Rozenberg (2012). *Handbook of natural computing*. Springer, Heidelberg (cited on pages 42, 44–48, 50, 51).

Bell, Okezue (2022). *Applications of Gaussian Mutation for Self Adaptation in Evolutionary Genetic Algorithms*. arXiv: 2201.00285 [cs.NE]. URL: https://arxiv.org/abs/2201.00285 (cited on page 38).

Blickle, Tobias and Lothar Thiele (1996). "A comparison of selection schemes used in evolutionary algorithms". In: *Evolutionary Computation* 4.4, pages 361–394 (cited on page 44).

Camburn, Bradley et al. (2017). "Design prototyping methods: state of the art in strategies, techniques, and guidelines". In: *Design Science* 3, e13 (cited on page 24).

Cilliers, Michael (2022). "Maintaining population diversity in evolutionary algorithms via epigenetics and speciation". eng. PhD thesis. University of Johannesburg (cited on page 42).

Cullinan, Michéle, Duncan Coulter, and Jacques van Appel (2023). "CLISDE: An Agent-Oriented Cladistic Island Genetic Algorithm". In: *2023 10th International Conference on Soft Computing & Machine Intelligence (ISCMI)*. IEEE, pages 27–33 (cited on page 45).

Das, Swagatam and Ponnuthurai Nagaratnam Suganthan (2010). "Differential evolution: A survey of the state-of-the-art". In: *IEEE transactions on evolutionary computation* 15.1, pages 4–31 (cited on pages 30, 53).

Diaz-Gomez, Pedro A and Dean F Hougen (2007). "Initial population for genetic algorithms: A metric approach." In: *Gem*, pages 43–49 (cited on page 43).

Dulebenets, Maxim A et al. (2018). "A self-adaptive evolutionary algorithm for the berth scheduling problem: Towards efficient parameter control". In: *Algorithms* 11.7, page 100 (cited on page 54).

Eiben, A. E. and James E. Smith (2015). *Introduction to Evolutionary Computing*. 2nd. Springer Publishing Company, Incorporated. ISBN: 3662448734 (cited on pages 27, 34, 35, 40, 46, 49–51).

Engelbrecht, Andries P (2007). *Computational intelligence: an introduction*. John Wiley & Sons (cited on page 11).

Ferreira, Cândida (2006). *Gene expression programming: mathematical modeling by an artificial intelligence*. Volume 21. Springer (cited on page 12).

Gregar, Jan (2023). "Research design (qualitative, quantitative and mixed methods approaches)". In: *Research Design* 8 (cited on page 25).

Guzek, Mateusz et al. (2014). "Multi-objective evolutionary algorithms for energy-aware scheduling on distributed computing systems". In: *Applied Soft Computing* 24, pages 432–446 (cited on page 55).

Hevner, Alan R et al. (2004). "Design science in information systems research". In: *MIS quarterly*, pages 75–105 (cited on pages 19, 20).

Hinterding, Robert (1995). "Gaussian mutation and self-adaption for numeric genetic algorithms". In: *Proceedings of 1995 IEEE International Conference on Evolutionary Computation* 1, pages 384–. URL: https://api.semanticscholar.org/CorpusID:30756203 (cited on page 38).

Koc, Hatice et al. (2021). "UML diagrams in software engineering research: a systematic literature review". In: *Proceedings*. Volume 74. 1. MDPI, page 13 (cited on page 23).

Koza, John R (1994). "Genetic programming as a means for programming computers by natural selection". In: *Statistics and computing* 4, pages 87–112 (cited on pages 30, 49, 51, 52).

Kramer, Oliver (Jan. 2017). *Genetic Algorithm Essentials*. ISBN: 978-3-319-52155-8. DOI: 10.1007/978-3-319-52156-5 (cited on pages 31–35).

Larranaga, Pedro et al. (Jan. 1999). "Genetic Algorithms for the Travelling Salesman Problem: A Review of Representations and Operators". In: *Artificial Intelligence Review* 13, pages 129–170. DOI: 10.1023/A:1006529012972 (cited on page 37).

Li, Changhe, Changhe/Han Li (Shoufe), and Sanyou Zeng (2024). *Intelligent Optimization*. Springer (cited on pages 32, 35, 38, 39, 46, 48, 54).

Li, Changhe, Shoufei Han, et al. (2024). "Basics of Evolutionary Computation Algorithms". In: *Intelligent Optimization: Principles, Algorithms and Applications*. Singapore: Springer Nature Singapore, pages 63–92. ISBN: 978-981-97-3286-9. DOI: 10.1007/978-981-97-3286-9_4. URL: https://doi.org/10.1007/978-981-97-3286-9_4 (cited on pages 11, 27).

Links, Alex Fraser and In MemoriamObituaryRemembranceMemory BookDust (2002). "Alex S. Fraser". In: *IEEE Transactions on Evolutionary Computation* 6.5 (cited on page 29).

Luke, Sean and Liviu Panait (2002). "Lexicographic parsimony pressure". In: *Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation*, pages 829–836 (cited on page 51).

Maaranen, Heikki, Kaisa Miettinen, and Antti Penttinen (2007). "On initial populations of a genetic algorithm for continuous optimization problems". In: *Journal of Global Optimization* 37, pages 405–436 (cited on page 33).

Malik, S and S Wadhwa (2014). "Preventing premature convergence in genetic algorithm using DGCA and elitist technique". In: *International Journal of Advanced Research in Computer Science and Software Engineering* 4.6 (cited on page 42).

Meyer-Nieberg, Silja and Hans-Georg Beyer (2007). "Self-adaptation in evolutionary algorithms". In: *Parameter setting in evolutionary algorithms*. Springer, pages 47–75 (cited on page 39).

Mirjalili, Seyedali (2018). *Evolutionary Algorithms and Neural Networks: Theory and Applications*. 1st. Springer Publishing Company, Incorporated. ISBN: 3319930249 (cited on pages 28, 38).

Mitchell, Melanie (1998). *An introduction to genetic algorithms*. MIT press (cited on pages 42, 45).

Neri, Ferrante and Carlos Cotta (2012). "Memetic algorithms and memetic computing optimization: A literature review". In: *Swarm and Evolutionary Computation* 2, pages 1–14 (cited on page 39).

Nikitin, Nikolay O et al. (2022). "Automated evolutionary approach for the design of composite machine learning pipelines". In: *Future Generation Computer Systems* 127, pages 109–125 (cited on page 54).

Nunamaker Jr, Jay F, Minder Chen, and Titus DM Purdin (1990). "Systems development in information systems research". In: *Journal of management information systems* 7.3, pages 89–106 (cited on page 24).

O'Neill, Michael (2009). *Riccardo Poli, William B. Langdon, Nicholas F. McPhee: A Field Guide to Genetic Programming: Lulu. com, 2008, 250 pp, ISBN 978-1-4092-0073-4* (cited on page 49).

Olivier, Martin S (2009). *Information technology research: A practical guide for computer science and informatics*. Van Schaik (cited on pages 22–25).

Petrowski, Alain and Sana Ben Hamida (Jan. 2016). *Evolutionary Algorithms* (cited on page 29).

Rachmawati, Lily and Dipti Srinivasan (2009). "Multiobjective evolutionary algorithm with controllable focus on the knees of the Pareto front". In: *IEEE Transactions on Evolutionary Computation* 13.4, pages 810–824 (cited on page 35).

Rafajłowicz, Wojciech (2018). "Horizontal Gene Transfer as a Method of Increasing Variability in Genetic Algorithms". In: *Artificial Intelligence and Soft Computing*. Edited by Leszek Rutkowski et al. Cham: Springer International Publishing, pages 505–513. ISBN: 978-3-319-91253-0 (cited on page 37).

Rothlauf, Franz (2009). "Representations for evolutionary algorithms". In: *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers*, pages 3131–3156 (cited on page 31).

Russell, Stuart J and Peter Norvig (2016). *Artificial intelligence: a modern approach*. Pearson (cited on page 12).

Sabar, Nasser R et al. (2019). "A self-adaptive evolutionary algorithm for dynamic vehicle routing problems with traffic congestion". In: *Swarm and evolutionary computation* 44, pages 1018–1027 (cited on page 54).

Satman, Mehmet Hakan and Emre Akadal (2020). "Machine coded compact genetic algorithms for real parameter optimization problems". In: *Alphanumeric Journal* 8.1, pages 43–58 (cited on page 47).

Segura, Carlos et al. (2016). "The importance of proper diversity management in evolutionary algorithms for combinatorial optimization". In: *NEO 2015: Results of the Numerical and Evolutionary Optimization Workshop NEO 2015 held at September 23-25 2015 in Tijuana, Mexico*. Springer, pages 121–148 (cited on pages 43, 44).

Shukla, Anupriya, Hari Mohan Pandey, and Deepti Mehrotra (2015). "Comparative review of selection techniques in genetic algorithm". In: *2015 international conference on futuristic trends on computational analysis and knowledge management (ABLAZE)*. IEEE, pages 515–519 (cited on page 41).

Slowik, Adam and Halina Kwasnicka (2020). "Evolutionary algorithms and their applications to engineering problems". In: *Neural Computing and Applications* 32, pages 12363–12379 (cited on page 28).

Stanley, Kenneth O and Risto Miikkulainen (2002). "Evolving neural networks through augmenting topologies". In: *Evolutionary computation* 10.2, pages 99–127 (cited on page 12).

Storn, Rainer and Kenneth Price (1995). "Differential evolution-a simple and efficient adaptive scheme for global optimization over continuous spaces". In: *International computer science institute* (cited on page 52).

Tharwat, Alaa and Wolfram Schenck (2021). "Population initialization techniques for evolutionary algorithms for single-objective constrained optimization problems: Deterministic vs. stochastic techniques". In: *Swarm and Evolutionary Computation* 67, page 100952. ISSN: 2210-6502. DOI: https://doi.org/10.1016/j.swevo.2021.100952. URL: https://www.sciencedirect.com/science/article/pii/S2210650221001140 (cited on page 33).

Wirsansky, Eyal (2024). *Hands-On Genetic Algorithms with Python*. Sciendo. DOI: 10.0000/9781805121572. URL: https://doi.org/10.0000/9781805121572 (cited on pages 30, 35, 40).

Xue, Feng, Arthur C Sanderson, and Robert J Graves (2003). "Pareto-based multi-objective differential evolution". In: *The 2003 Congress on Evolutionary Computation, 2003. CEC'03*. Volume 2. IEEE, pages 862–869 (cited on page 53).

Young, Steven R et al. (2015). "Optimizing deep learning hyper-parameters through an evolutionary algorithm". In: *Proceedings of the workshop on machine learning in high-performance computing environments*, pages 1–5 (cited on page 54).

Yu, X. and M. Gen (2010). *Introduction to Evolutionary Algorithms*. Decision Engineering. Springer London. ISBN: 9781849961295. URL: https://books.google.co.za/books?id=rHQf_2Dx2ucC (cited on page 31).

Zainuddin, Farah Ayiesya, Md Fahmi Abd Samad, and Durian Tunggal (2020). "A review of crossover methods and problem representation of genetic algorithm in recent engineering applications". In: *International Journal of Advanced Science and Technology* 29.6s, pages 759–769 (cited on page 37).

Zhou, Aimin et al. (2011). "Multiobjective evolutionary algorithms: A survey of the state of the art". In: *Swarm and Evolutionary Computation* 1.1, pages 32–49. ISSN: 2210-6502. DOI: https://doi.org/10.1016/j.swevo.2011.03.001. URL: https://www.sciencedirect.com/science/article/pii/S2210650211000058 (cited on page 34).