HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

GRADUATION THESIS

Evolutionary Computation and Large Language Model for Automatic Heuristics Design

PHẠM VŨ TUẤN ĐẠT
dat.pvt210158@sis.hust.edu.vn

Major: Computer Science

Supervisor:	Associate Professor Huỳnh Thị Thanh Bình	______________
		Signature
Department:	Computer Science	
School:	School of Information and Communications Technology	

HANOI, 01/2025

ACKNOWLEDGMENT

For me, finishing this project signifies not just a milestone but the start of an exciting new journey. While conducting this thesis, I have often felt lucky. This work would not have been possible without the invaluable guidance and mentorship of Associate Professor Huỳnh Thị Thanh Bình. I am also grateful to my mentor, Long Đoàn, for our weekly discussions and his thoughtful feedback. I would like to extend my thanks to the members of the MSO Lab, who offered their time to enrich this work in countless ways. Last but not least, I want to express my heartfelt appreciation to my beloved family and best friends, who have always been my source of emotional support and encouragement during the most challenging times. **Thank you all from the bottom of my heart!**

ABSTRACT

Automatic Heuristic Design (AHD) is an active research area due to its utility in solving complex search and NP-hard combinatorial optimization problems in the real world. Recent advances in Large Language Models (LLMs) introduce new possibilities by coupling LLMs with Evolutionary Computation (EC) to automatically generate heuristics, known as LLM-based Evolutionary Program Search (LLM-EPS). While previous LLM-EPS studies obtained great performance on various tasks, there is still a gap in understanding the properties of heuristic search spaces and achieving a balance between exploration and exploitation, which is a critical factor in large heuristic search spaces. This research addresses this gap by proposing two diversity measurement metrics and performing an analysis of previous LLM-EPS approaches, including FunSearch, EoH, and ReEvo. Results on black-box AHD problems reveal that while EoH demonstrates higher diversity than FunSearch and ReEvo, its objective score is unstable. Conversely, ReEvo's reflection mechanism yields good objective scores but fails to optimize diversity effectively. In light of these findings, HSEvo was introduced as an adaptive LLM-EPS framework that strikes a balance between diversity and convergence through the use of a harmony search algorithm. Experimentation demonstrated that HSEvo achieved high diversity indices and good objective scores while remaining cost-effective. These results underscore the importance of balancing exploration and exploitation and understanding heuristic search spaces in designing frameworks in LLM-EPS.

Student

Phạm Vũ Tuấn Đạt

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

Abbreviation	Definition
ACO	Ant Colony Optimization
AHD	Automatic Heuristic Design
AI	Artificial Intelligence
BPO	Bin Packing Online
CDI	Cumulative Diversity Index
COPs	Combinatorial Optimization Problems
EC	Evolutionary Computation
EP	Evolutionary Programming
ES	Evolution Strategies
GA	Genetic Algorithm
GLS	Guided Local Search
GP	Genetic Programming
HHs	Hyper-Heuristics
HS	Harmony Search
LLM-EPS	LLM-based Evolutionary Program Search
LLMs	Large Language Models
MST	Minimum Spanning Tree
NCO	Neural Combinatorial Optimization
OP	Orienteering Problem
SDI	Shannon Diversity Index
SWDI	Shannon-Wiener Diversity Index
TSP	Traveling Salesman Problem

INTRODUCTION

NP-hard Combinatorial Optimization Problems (COPs) frequently arise in real-world scenarios. Examples include groups of problems such as routing and logistics, scheduling and planning, network design, and others. To address these problems, heuristic methods are among the most commonly employed approaches due to their superior efficiency in terms of time and resources needed to find solutions that approximate the optimal. Over the past few decades, significant efforts have been dedicated to designing effective search methods, resulting in techniques such as simulated annealing, tabu search, and iterated local search, among many others. These manually crafted methods have been successfully applied in numerous real-world applications.

However, due to the diverse nature of practical problems, each application comes with its constraints and objectives, often requiring customization or the selection of specific search methods tailored to the situation at hand. Manually creating, tuning, and configuring search methods for particular problems is not only labor-intensive but also demands deep expertise. This represents a bottleneck in many application domains. Consequently, AHD has emerged as a promising solution, aiming to automate the selection, tuning, or construction of efficient search methods for specific classes of problems.

The goal of AHD is to address the difficulties involved in the creation of manual heuristics by utilizing computational methods to automatically generate or enhance heuristics. Among the various approaches within AHD, both Hyper-Heuristics (HHs) [1], [2] and Neural Combinatorial Optimization (NCO) [3]–[5] have gained substantial attention for their potential to address the limitations of traditional heuristic development.

HHs operate within predefined heuristic spaces curated by human experts, selecting or generating heuristics from these established sets. While HHs have demonstrated success, their performance is inherently limited by the scope and quality of the predefined heuristic space, often restricting the discovery of novel or more effective heuristics.

On the other hand, NCO employs neural networks to learn patterns and predict solutions for COPs. By leveraging data-driven models, NCO seeks to generalize across problem instances, offering a more dynamic approach to heuristic development. Despite its promise, NCO faces significant challenges. Effective generalization across diverse problem instances requires robust inductive biases [6], which are difficult to design and implement. Additionally, the interpretability and explainability of the learned models remain pressing concerns [5], making it challenging to validate or trust the solutions produced. These limitations highlight the need for NCO frameworks to evolve further, incorporating mechanisms that enhance their adaptability and transparency.

The constraints associated with HHS and NCO underscore the need for more flexible, adaptive, and innovative methods in AHD. Addressing these limitations can unlock the full potential of heuristic design, enabling the development of systems that not only adapt to complex and evolving problem landscapes but also expand the horizons of heuristic discovery beyond current boundaries. Such advancements will play a crucial role in shaping the future of optimization and decision-making in both academic research and practical applications.

Recently, the rise of LLMs has opened up new possibilities for AHD. It is believed that LLMs [7], [8] could be a powerful tool for generating new ideas and heuristics. However, standalone LLMs with prompt engineering can be insufficient for producing novel and useful ideas beyond existing knowledge [9]. Some attempts have been made to couple LLMs with EC to automatically generate heuristics, known as LLM-EPS [10]–[12]. Initial works such as FunSearch [13] and subsequent developments like Evolution of Heuristic (know as EoH) [14] and Reflective Evolution (know as ReEvo) [15] have demonstrated significant improvements over previous approaches, generating quality heuristics that often surpass current methods. Even so, ReEvo yields state-of-the-art and competitive results compared with evolutionary algorithms, neural-enhanced meta-heuristics, and neural solvers.

A key difference between LLM-EPS and classic AHD lies in the search spaces of heuristics. Classic AHD typically operates within well-defined mathematical spaces such as R^n, whereas LLM-EPS involves searching within the space of functions, where each function represents a heuristic as a program. This functional search space introduces unique challenges and opportunities, necessitating a deeper understanding of the properties and characteristics of heuristics within this domain. Despite the advancements made by LLM-EPS frameworks, there remains a significant gap in foundational theories and principles that underpin AHD within these expansive and complex search spaces.

Scope of research

This research focuses on the intersection of EC and LLMs, exploring their potential synergy in AHD. History has demonstrated that evolutionary algorithms excel in optimization tasks ranging from combinatorial problems and image enhancement to industrial scheduling thanks to their population-based search mechanisms that balance exploration and exploitation. Simultaneously, LLMs such as ChatGPT, Claude, and Gemini have achieved remarkable advancements in natural language processing and code generation, showcasing their ability to produce coherent and contextually relevant outputs.

The primary objective of this research is to explore how these two fields can collaborate to design heuristics for solving COPs. While current research has made progress in metrics such as solution quality and runtime efficiency, maintaining and leveraging diversity among candidate solutions remains underexplored. This oversight may limit the range of potential solutions, ultimately hindering opportunities for innovation.

Thus, the scope of this study aims at three objectives. First, "tools" should be developed to measure the diversity level promoted by various methods within LLM-EPS frameworks. This quantification will enable straightforward comparison and selection of suitable diversity-enhancing methods, while also serving as a foundation for assessing the balance between exploration and exploitation. Second, the study seeks to analyze how varying levels of diversity impact the search quality of different LLM-EPS frameworks. Finally, this work aims to design and evaluate a new LLM-EPS framework that balances exploration and exploitation, incorporating insights drawn from the second objective through the tools developed in the first objective. This framework will be tested on AHD problems, with results compared against existing LLM-EPS methods. This comparison will help evaluate the proposed framework and provide evidence of its potential contributions based on the findings observed.

Contributions

This thesis makes three key contributions to the field of AHD:

- Two new metrics are the Shannon-Wiener Diversity Index (SWDI) and the Cumulative Diversity Index (CDI) proposed to evaluate and track the diversity of heuristic populations in LLM-EPS frameworks. SWDI evaluates diversity based on the distribution of heuristic clusters, encouraging a balance between exploration and exploitation. CDI assesses the overall spread and uniformity of heuristics in the search space by analyzing their distribution through a minimum spanning tree. These metrics address challenges in quantifying diversity in function-based search spaces. Both metrics offer practical insights into population dynamics, revealing how diverse heuristics contribute to escaping local optima and improving optimization outcomes.

- A comprehensive analysis of heuristic search spaces has been conducted. This investigation explores the relationship between the proposed diversity metrics and the performance of various LLM-EPS frameworks across different optimization problems. The analysis highlights that while higher diversity generally supports exploration, excessive diversity without proper optimization can hinder performance. Conversely, frameworks that overly prioritize exploitation risk premature convergence. These findings establish the significance of balanced diversity in heuristic design.
- Introduce a novel framework called HSEvo. HSEvo integrates the Harmony Search (HS) algorithm to optimize diversity and convergence within LLM-EPS frameworks. It refines traditional evolutionary phases such as initialization, crossover, and mutation by incorporating advanced techniques like flash reflection and elitist mutation. Flash reflection uses LLM-driven analyses to guide heuristic improvements, while elitist mutation focuses on refining top-performing heuristics. The HS component tunes heuristic parameters to enhance population quality while maintaining diversity. Experimental results demonstrate that HSEvo outperforms existing frameworks by achieving high diversity indices and superior objective scores, providing a cost-effective and robust solution for AHD.

Thesis outline

The remainder of this thesis is organized as follows.

Chapter 1 offers a thorough review of relevant literature, focusing on the foundational concepts of EC and LLMs. It also explores the emerging field of LLM-EPS and emphasizes the significance of diversity in evolutionary algorithms. The chapter concludes by identifying gaps in current research and laying the groundwork for the proposed work.

Chapter 2 introduces two novel diversity metrics SWDI and CDI to measure and analyze the diversity of heuristic populations in LLM-EPS frameworks. Additionally, it explores the correlation between these diversity metrics and the objective scores of various optimization problems, providing insights into the role of diversity in heuristic search spaces.

Chapter 3 explains the proposed HSEvo framework, which integrates the HS algorithm with EC to balance exploration and exploitation in heuristic search spaces. It provides detailed descriptions of HSEvo, including population initialization, flash reflection, crossover, elitist mutation, and harmony search.

Chapter 4 presents the experimental setup, benchmark datasets, and results of applying HSEvo to various AHD problems, including Bin Packing Online, Traveling Salesman Problem, and Orienteering Problem. After that, compares the performance of HSEvo with previous LLM-EPS frameworks and includes an ablation study to evaluate the effectiveness of the proposed components.

Finally, **Conclusions** summarizes the key findings of the thesis, highlighting the importance of diversity in LLM-EPS frameworks and the effectiveness of the proposed HSEvo framework. It also suggests potential directions for future research in the field of AHD.

CHAPTER 1. LITERATURE REVIEW

Artificial Intelligence (AI) is evolving at an extraordinary pace, inspiring innovative methods that blend disparate computational fields to tackle some of the most demanding challenges. In this rapidly shifting landscape, two particular paradigms stand out for their strengths yet have traditionally advanced on separate tracks: EC and LLMs. EC, inspired by natural evolution, explores populations of candidate solutions to discover high-quality outcomes, whereas LLMs are adept at generating coherent, context-aware outputs ranging from textual prose to executable code. As AI applications become more ambitious, a growing area of research, known as LLM-EPS, seeks to harness the best of both worlds: leveraging the adaptive search prowess of EC and the creative and generative capabilities of LLMs. This emerging fusion offers a novel route for solving challenges in AHD.

Initial research indicates that aligning the iterative adjustments of EC with the adaptable generation abilities of LLMs can lead to significant advancements, especially in the creation of heuristic programs, code fragments, or even written solutions. Nevertheless, an essential question persists: How can diversity be ensured throughout the evolutionary process? This is particularly important when candidate solutions are no longer merely simple numeric vectors but are instead diverse forms of code or text. Preserving diversity is paramount to avoid converging too quickly on suboptimal solutions and to maintain an expansive search of the solution landscape. The subsequent sections provide an overview of EC principles, discuss the capabilities and constraints of modern LLMs, and survey recent strides in LLM-EPS research, with special attention to how future investigations might harness diversity to unlock even more powerful solutions.

1.1 Related works

Although the convergence of EC and LLMs for optimization and AI is still a relatively new domain, it builds upon two longstanding bodies of research that have more often evolved in isolation. EC traditionally tackles numerical or combinatorial challenges, while LLMs excel at text-oriented tasks like language translation, content generation, and summarization. Current work in LLM-EPS illustrates how these fields can be merged: LLMs become generators of candidate solutions whether code snippets, pseudo-code, or textual explanations, and evolutionary frameworks guide these outputs toward continuous improvement.

Recent studies show this collaboration. In code generation, studies [7], [16], [17] showcase evolutionary strategies that refine LLM suggested programs for enhanced quality and resilience. In text generation [18], [19] illustrates how evolutionary refinement can produce more precise and diverse linguistic outputs. Together, these works point to tangible gains, including improved error resolution, larger solution spaces, and quicker convergence.

Nevertheless, most existing research [13]–[15] focuses on performance indicators without explicitly examining how population diversity can be steered or maintained when solutions take the form of human-readable code or text. This missing piece is critical for keeping evolutionary searches from prematurely collapsing on local optima and for fostering creative, wide-ranging solution sets. Future efforts, therefore, might delve deeper into incorporating diversity metrics to ensure that LLM-driven evolutionary processes not only excel in performance but also yield robust and flexible results across numerous domains.

1.2 Evolutionary computation

EC [20] is one computational intelligence model used to mimic the biological evolution phenomenon. Currently, EC includes four algorithms: Genetic Algorithm (GA), Evolutionary Programming (EP), Evolution Strategies (ES), and Genetic Programming (GP). GA was proposed by the American scholar Holland in the 1950s in a study of self-adapting control. To study the finite-state machine of AI, the American scholar Fogel proposed EP in the 1960s. At nearly the same time, the German scholars Rechenberg and Schwefel proposed ES to solve numerical optimization. In the 1990s, based on the GA, the American scholar Koza proposed GP to study the automatic design of computer programs.

Although the four algorithms were proposed by different scholars for different purposes, their computing processes are similar and can be described as follows.

1. One group of initial feasible solutions are created;
2. The properties of the initial solutions are evaluated;
3. The initial solutions are selected according to their evaluation results;
4. The selected solutions are conducted by the evolutionary operations and the next generation of feasible solutions can be obtained;
5. If the feasible solutions obtained during the above step can meet the requirements, then the computation will stop. Otherwise, the feasible solutions obtained during the above step are taken as the initial solutions, and the computation process returns to step 2.

Generally, as one global optimization method, EC has the following characteristics: (i) The search process begins from one group and not from one point; (ii) only the objective function is used in the search process; and (iii) the random method is used in the search process. Therefore, this method has the following advantages: (i) Highly versatile and can be used for different problems; (ii) it can solve problems that are highly nonlinear and nonconvex; and (iii) the model's plasticity is very high and can be deserialized very easily.

The three evolutionary computation algorithms related to the research on this LLM-EPS include GA, EP, and GP. In the next section of the chapter, these algorithms are briefly described.

1.2.1 Genetic algorithm

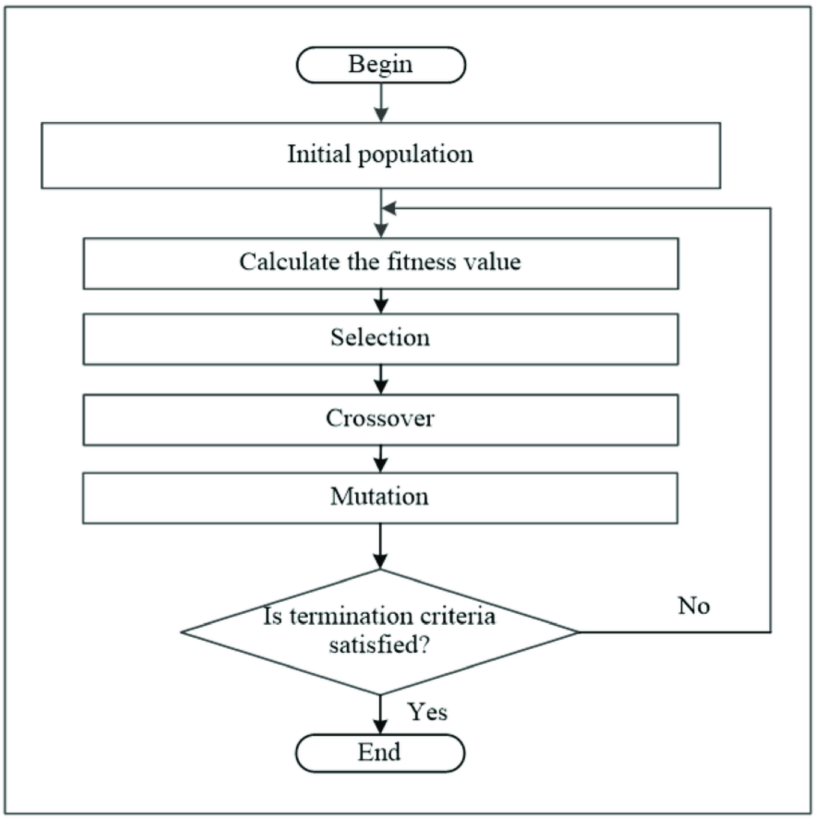

Begin
Initial population
Calculate the fitness value
Selection
Crossover
Mutation
Is termination criteria satisfied?
No
Yes
End

Figure 1.1: Flow chart of GA.

As the most widely used EC method, GA has a solid biologic foundation. Its basic principles are from Darwin's evolution theory and Mendel's genetic theory [21]. The basic GA flow chart is shown in Figure 1.1. The main operations of GA are as follows.

1. **Individual encoding.** According to Mendel's genetic theory, the individual must be represented by its genotype. Therefore, the binary encoding method is generally applied, which is one string of numbers 0 and 1. However, for a complicated engineering problem, the real encoding method is always used, which is one string of a real number.

2. **Crossover operation.** This operation is completed to mimic the genetic recombination during biological mating. Generally, random pairing is applied. For different encoding methods, the crossover operation is different. For binary encoding, the point crossover is utilized. For real encoding, the discrete crossover and arithmetic crossover are always used. Moreover, this operation is conducted with a large probability.

3. **Mutation operation.** This operation is completed to mimic the gene mutation during the genetic process. Generally, for binary encoding, the simple point mutation is utilized. For real encoding, the uniform mutation and non-uniform mutation are always used. Moreover, this operation is conducted with a little probability.

4. **Selection operation.** This operation is completed to mimic the environment selection of Darwin's evolution theory. Therefore, it is also called the reproduction operation. Generally, roulette wheel selection is widely used. However, stochastic tournament selection is a better operation for complicated engineering problems.

5. **Termination criterion.** Generally, the termination criterion specifies that the difference between the maximum fitness value and average fitness value of one group is less than one error ϵ. To avoid infinite iteration, a maximum number of evolutionary generations is also specified. Moreover, there are some parameters to be determined: the number of individuals, crossover probability, mutation probability, ϵ and maximum number of evolutionary generations. Generally, these parameters are determined based on experience and a test.

1.2.2 Evolutionary programming

The main operations of EP are as follow [22], [23].

1. **Mutation operation.** This operation is the sole optimization operation; this is the specialty of the EP. Generally, Gauss random mutation is utilized.

2. **Testing feasibility of the individual.** The fundamental nature of mutation involves a type of random alteration to the original individual, which means it cannot ensure that the newly mutated individual remains within the search space; in other words, a non-feasible individual could be generated. The existence of a non-feasible individual can not only make the result incorrect but also make the efficiency low. To overcome this problem, a simple and easy method is proposed [23]. If the new mutated individual is non-feasible, it will be replaced by a randomly created feasible individual. However, this technique

may have a very low efficiency for certain problems.

3. **Selection operation.** The selection operation is a stochastic tournament model, where individuals are probabilistically chosen based on their fitness, ensuring a balance between exploration and exploitation.

4. **Termination condition.** This is the same as that of the GA. Moreover, there are some parameters to be determined: the number of individuals, ϵ and maximum number of evolutionary generations. Thus, the number of parameters for EP is less than that for the GA. Generally, these parameters can also be determined based on experience and a test.

1.2.3 Genetic programming

Because GP is proposed based on the basic GA principles, the procedures of two algorithms are nearly the same [24]. The main operations of GP are as follows.

1. **Individual encoding.** The tree encoding method is applied. In other words, the layering tree structure expression is applied, which is shown in top of Figure 1.3.

2. **Crossover operation.** Generally, random pairing and point crossover are utilized. This operation is also conducted with a large probability.

3. **Mutation operation.** Generally, simple point mutation is utilized. This operation is conducted with a little probability.

4. **Selection operation.** Generally, roulette wheel selection is used.

5. **Termination criterion.** This is the same as that of GA. Moreover, the parameters are the same as those of GA.

1.3 Large language models

LLMs are sophisticated AI systems that specialize in understanding, generating, and predicting human-like text. By analyzing extensive datasets, these models have made remarkable strides in various natural language processing tasks. Their capabilities include text generation, translation, and summarization, enhancing communication and information retrieval across different applications.

The evolution of language models has progressed from statistical methods to sophisticated neural network-based approaches:

- **Statistical language models:** Early models, such as n-grams, estimated the probability of a word based on its preceding words but struggled with data sparsity and limited context understanding.

- **Neural network models:** The introduction of neural networks allowed for continuous word representations, alleviating some limitations of statistical language models. Recurrent neural networks and long short-term memory networks were employed to capture sequential data dependencies.
- **Transformer architecture:** Introduced in [25], transformers utilize self-attention mechanisms to process input data in parallel, enabling the capture of long-range dependencies more effectively. This architecture underpins many modern LLMs.

LLMs are built upon several key concepts. Fristly, LLMs assign probabilities to sequences of words, modeling the likelihood of a word following a given context. This is formalized as:

$$P(w_1, w_2, \ldots, w_n) = \prod_{i=1}^{n} P(w_i \mid w_1, w_2, \ldots, w_{i-1}) \tag{1.1}$$

where $P(w_i \mid w_1, w_2, \ldots, w_{i-1})$ represents the conditional probability of word w_i given its preceding words.

Secondly, LLMs employ deep neural networks with multiple layers and parameters to learn complex language patterns. The training process involves adjusting these parameters to minimize a loss function, typically the cross-entropy loss:

$$\mathcal{L} = -\sum_{i=1}^{N} \log P(w_i \mid \text{context}) \tag{1.2}$$

where N is the number of words in the training corpus.

Finally the heart of modern LLMs lies the Transformer architecture (Figure 1.2). Transformers use self-attention mechanisms to weigh the importance of different words in a sequence, enabling the model to capture contextual relationships effectively. The self-attention mechanism is computed as:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) V \tag{1.3}$$

where Q (queries), K (keys), and V (values) are matrices derived from the input embeddings, and d_k is the dimensionality of the keys.

Training LLMs involves several key steps. First, data collection is essential, as it requires aggregating extensive text data from diverse sources to capture a wide range of language patterns. Once the data is collected, model training begins, utilizing high-performance computing resources to optimize the model's parameters through techniques such as gradient descent. Additionally, scaling laws have been empirically observed, which suggest that increasing the model size, training data, and computational resources leads to improved performance. One notable scaling law is represented by the equation

$$C = C_0 \times N \times D \tag{1.4}$$

where C is the training cost in FLOPs, N is the number of model parameters, D is the size of the training dataset in tokens, and C_0 is a constant.

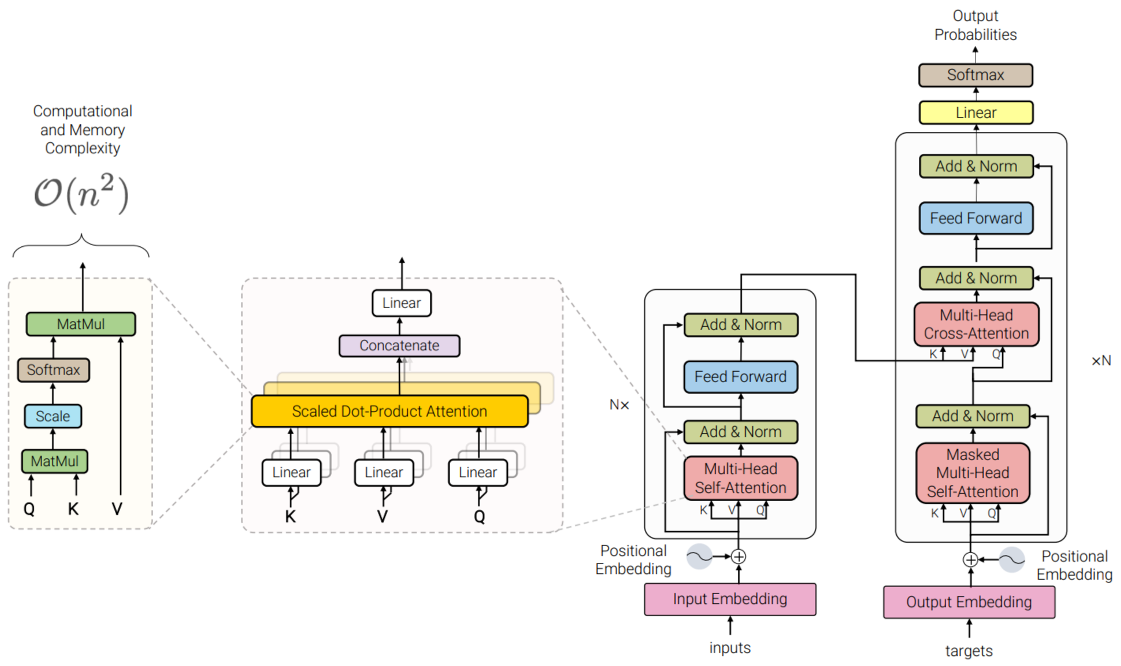
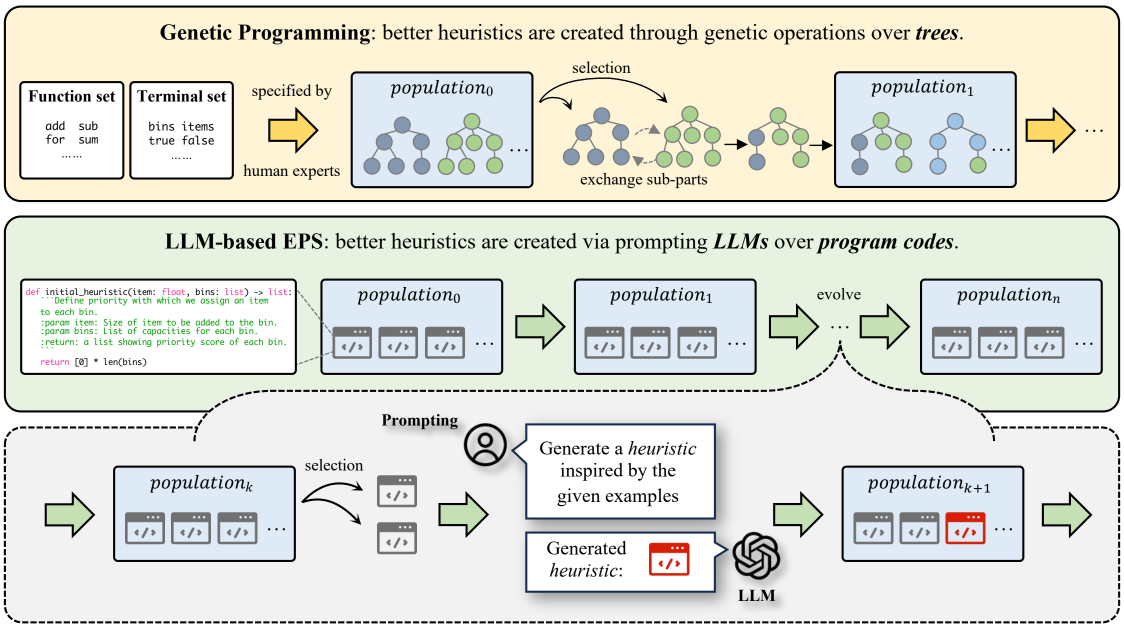


Figure 1.2: Architecture of large neural models [25], [26].

Despite their capabilities, LLMs face with several limitations. LLMs may generate factually incorrect or contextually misleading outputs colloquially known as "hallucinations." In addition, their large, black-box structures can complicate interpretability, making it challenging to understand precisely why a particular token or code snippet is produced. Moreover, controlled generation, where specific constraints or style guides must be applied, can be non-trivial to implement. Regardless, their unprecedented ability to generate complex textual or code-based artifacts has spurred interest in integrating LLMs into evolutionary workflows, particularly for tasks that benefit from a human-like creative spark.



~~~
def initial_heuristic(item: float, bins: list) -> list:
    ```Define priority with which we assign an item
    to each bin.
    :param item: Size of item to be added to the bin.
    :param bins: List of capacities for each bin.
    :return: a list showing priority score of each bin.
    ```
    return [0] * len(bins)
~~~

Figure 1.3: An illustration of the LLM-based EPS paradigm, with respect to the GP-based paradigm (top section), for AHD [27].

1.4 LLM-based evolutionary program search

LLM-EPS combines the adaptive search capabilities of evolutionary algorithms with the generative power of LLMs, unlocking new possibilities for AHD. Unlike traditional optimization methods that produce numeric solutions, LLM-EPS generates interpretable programs or code snippets, allowing for seamless interaction and validation by human expert an attractive feature for real-world applications. Figure 1.3 illustrates the fundamental differences between GP and LLM-EPS.

1.4.1 FunSearch

FunSearch, short for "search in the function space," represents a groundbreaking approach in leveraging LLMs for program synthesis and optimization. Unlike traditional methods, which often face limitations due to hallucinations or verification challenges. FunSearch innovatively combines an LLM with a systematic evaluator. This evaluator rigorously scores candidate programs, ensuring correctness and quality, and plays a pivotal role in evolving low-performing solutions into high-quality, optimized programs.

At the core of FunSearch's architecture, as shown in Figure 1.4, is its evolutionary methodology, which starts with a user-defined "skeleton" program. This skeleton encapsulates known functional structures and isolates the critical logic for optimization, thereby reducing the search space and improving accuracy. The system employs an island-based model to maintain a diverse pool of candidate programs, as shown in Figure 1.5. By avoiding local optima and encouraging diversity

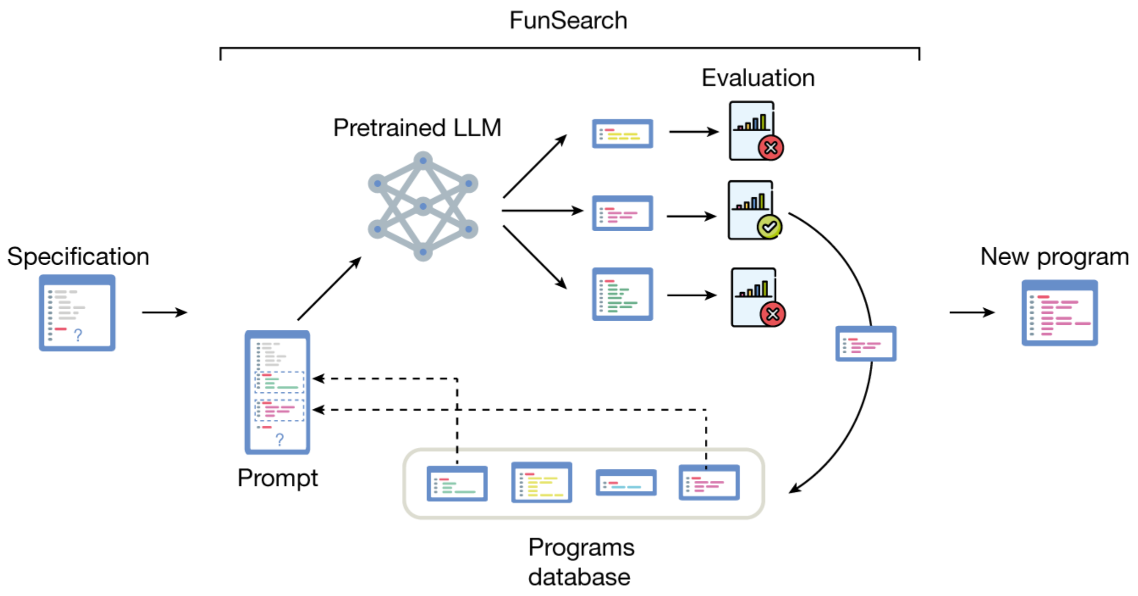


Figure 1.4: The FunSearch process [13]. The LLM is shown a selection of the best programs it has generated so far (retrieved from the programs database), and asked to generate an even better one. The programs proposed by the LLM are automatically executed, and evaluated. The best programs are added to the database, for selection in subsequent cycles. The user can at any point retrieve the highest-scoring programs discovered so far.

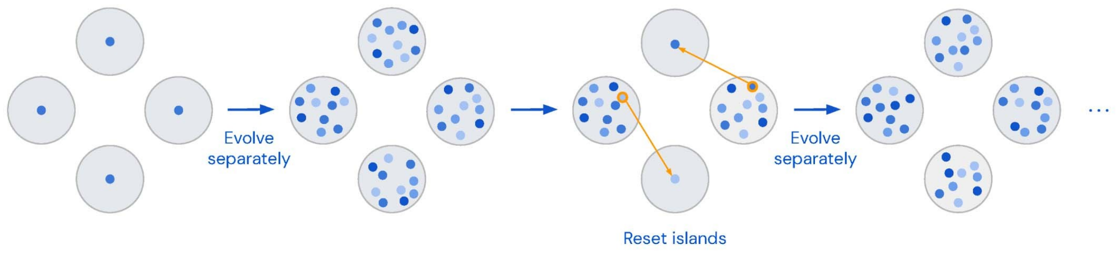


Figure 1.5: Program clusters within islands of FunSearch [13].

through genetic-like operations, FunSearch achieves robust exploration across a wide range of program variations.

The distributed architecture of FunSearch enhances scalability and efficiency. The system comprises three main components: a programs database, samplers, and evaluators. The programs database stores correct and promising programs, while samplers use LLMs to generate new variations based on prompts constructed from high-scoring programs. Evaluators assess these programs by executing them against a predefined "evaluate" function, which measures the quality of outputs on specific inputs. These components operate asynchronously, leveraging parallelism to handle computationally intensive tasks efficiently.

FunSearch's iterative process integrates several innovative techniques. Best-shot prompting ensures that high-quality programs guide the evolution process, while clustering within islands helps preserve program diversity. The inclusion of Boltzmann selection mechanisms and fitness-based sampling further refines the evolutionary trajectory, favoring both performance and interpretability. The system's distributed nature supports scalability, enabling it to tackle large, complex problems across a variety of domains, from mathematical discovery to algorithm design.

This architecture not only delivers impressive results but also highlights FunSearch's adaptability. By systematically blending the creativity of LLMs with rigorous evaluation and evolutionary improvements, FunSearch sets a new benchmark for program synthesis, pushing the boundaries of what LLMs can achieve in structured problem-solving.

1.4.2 Evolution of heuristics

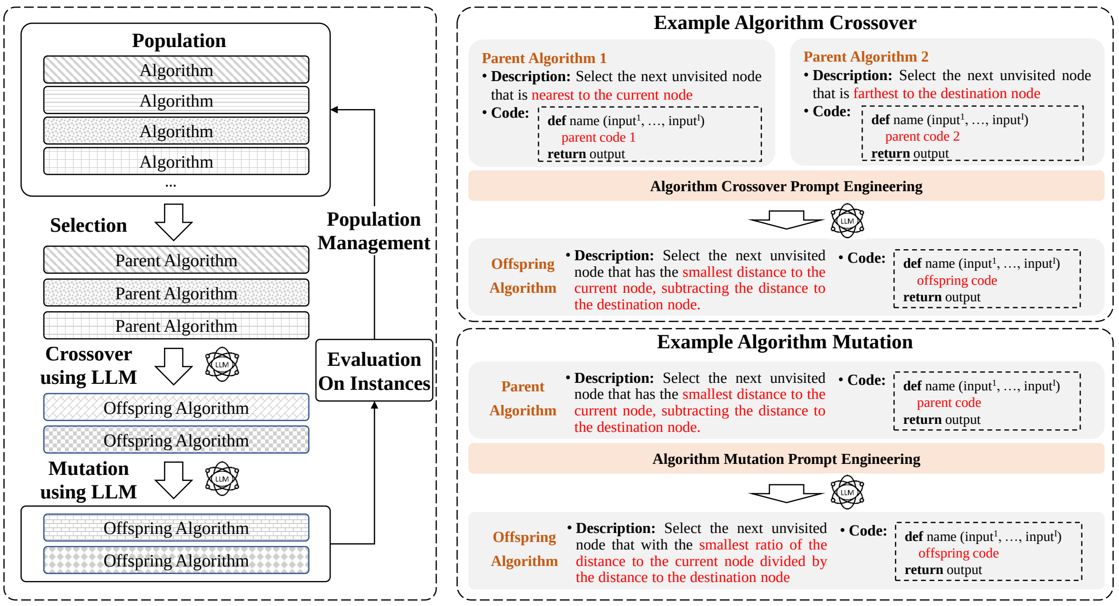

Population
Algorithm
Algorithm
Algorithm
Algorithm
Selection
Parent Algorithm
Parent Algorithm
Parent Algorithm
Crossover using LLM
Offspring Algorithm
Offspring Algorithm
Mutation using LLM
Offspring Algorithm
Offspring Algorithm
Population Management
Evaluation On Instances
LLM
Example Algorithm Crossover
Parent Algorithm 1
• Description: Select the next unvisited node that is nearest to the current node
• Code: def name (input¹, …, inputᴵ) parent code 1 return output
Parent Algorithm 2
• Description: Select the next unvisited node that is farthest to the destination node
• Code: def name (input¹, …, inputᴵ) parent code 2 return output
Algorithm Crossover Prompt Engineering
Offspring Algorithm
• Description: Select the next unvisited node that has the smallest distance to the current node, subtracting the distance to the destination node.
• Code: def name (input¹, …, inputᴵ) offspring code return output
Example Algorithm Mutation
Parent Algorithm
• Description: Select the next unvisited node that has the smallest distance to the current node, subtracting the distance to the destination node.
• Code: def name (input¹, …, inputᴵ) parent code return output
Algorithm Mutation Prompt Engineering
Offspring Algorithm
• Description: Select the next unvisited node that with the smallest ratio of the distance to the current node divided by the distance to the destination node
• Code: def name (input¹, …, inputᴵ) offspring code return output

Figure 1.6: An illustration of the AEL framework [28], which serves as a foundational prior for EoH.

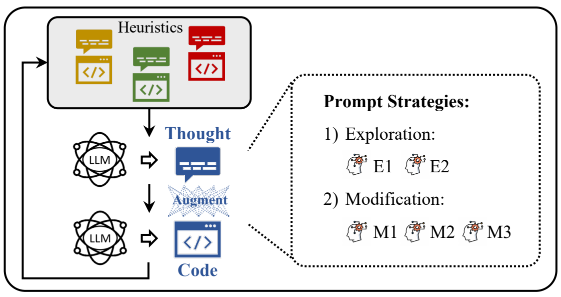


Figure 1.7: EoH evolves both thoughts and codes using LLMs [14].

Another noteworthy approach is Evolution of Heuristics (EoH), aimed explicitly at AHD while minimizing manual labor. EoH represents heuristics both as natural language "thoughts" and as executable code, mirroring the way human experts conceptualize and then implement strategies. By employing a population-based algorithm, EoH evolves these heuristics over successive generations with selection, crossover, and mutation operations guided by LLMs.

EoH employs a population-based evolutionary model where heuristics evolve over generations. Selection, crossover, and mutation operations enhanced by LLMs generate new heuristics iteratively (Figure 1.6). Five specific prompt strategies guide this evolution (Figure 1.7). Divide to, Exploration: includes generating novel heuristics (E1) or exploring variations based on common ideas (E2); and Modification: focuses on refining heuristics for improved performance (M1), adjusting parameters (M2), or simplifying structures by eliminating redundancies (M3).

This method requires fewer queries to LLMs compared to previous techniques such as FunSearch. Additionally, it eliminates the need for custom-crafted features or additional training. The upshot is a more streamlined method that outperforms manually created heuristics across diverse optimization benchmarks. As EoH reduces the overhead of domain-specific engineering, it underscores the breadth of potential applications for LLM-EPS.

1.4.3 Reflective evolution

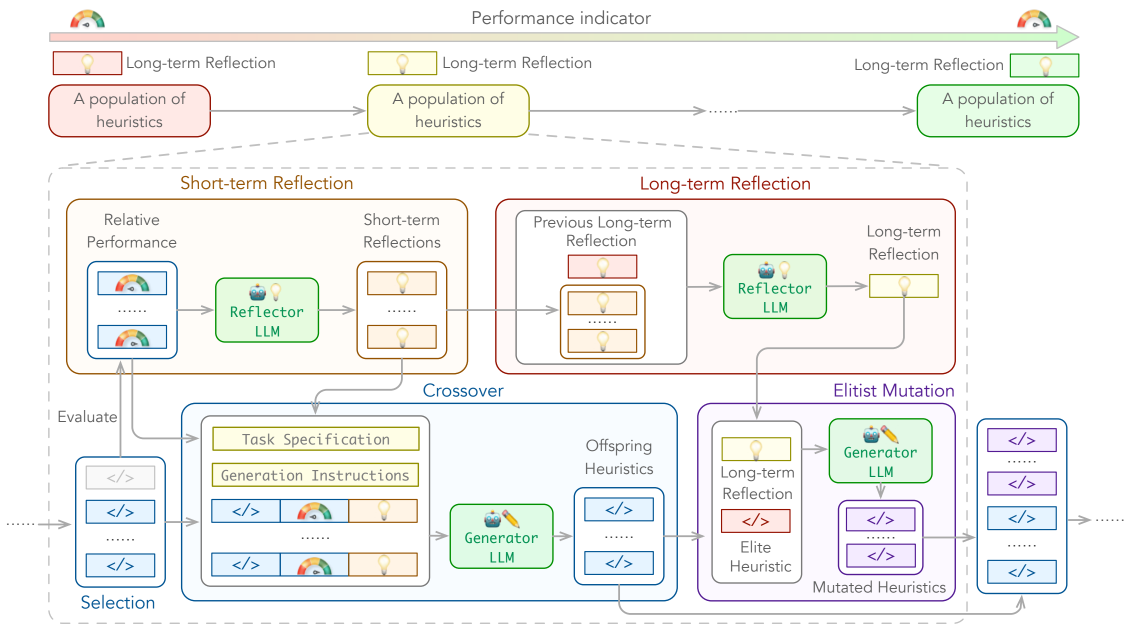


Figure 1.8: ReEvo pipeline [15].

In pursuit of state-of-the-art results, Reflective Evolution (ReEvo) integrates LLMs with a GP foundation to tackle NP-hard combinatorial optimization tasks. ReEvo employs two types of LLMs: a Generator LLM, which produces initial code snippets or heuristics, and a Reflector LLM, which provides both short-term and long-term reflections to guide iterative improvements.

The genetic cycle in ReEvo, as shown in Figure 1.8 consists of the following steps: (i) Population Initialization: Prompts generate initial heuristic candidates. (ii) Selection: Parent heuristics are chosen based on their performance. (iii) Short-term Reflection: Comparative analysis of parent performance informs offspring generation. (iv) Crossover: Generates offspring by combining parent features. (v) Long-term Reflection: Consolidates insights from iterations to improve heuristic generation. (vi) Elitist Mutation: Refines the best heuristic further using accumulated reflections.

By synthesizing immediate feedback (short-term reflection) and broader insights (long-term reflection), ReEvo delivers sample-efficient heuristics, surpassing frameworks like EoH and FunSearch. Its verbal "gradient," derived from reflective insights, leads to a gentler fitness landscape and boosts optimization stability, making it a potent approach for real-world combinatorial problems.

1.5 Diversity in evolutionary computation

Diversity is a cornerstone of successful evolutionary algorithms, especially in scenarios with multiple objectives [29]. A population rich in diverse solutions maintains broad coverage of the search space, mitigating the dangers of stagnation in local optima and supporting adaptability to non-static or convoluted fitness landscapes. Various methods clustering, niching, or entropy-based measures have been proposed to safeguard diversity, with Shannon entropy being a popular metric for assessing how "spread out" a set of solutions is [30], [31].

However, embedding these diversity concepts into LLM-EPS raises fresh challenges. Populations are not numeric vectors but heuristic code snippets or descriptive text, which calls for alternative ways to quantify "distance" or "difference." What strategies can be employed to evaluate the semantic gap between two heuristic code snippets or two text proposals while effectively capturing their functional differences? These inquiries highlight a pressing need for innovative strategies in diversity management that are compatible with LLM-based searches.

Crucially, diversity acts not only as a buffer against premature convergence, but also as a catalyst for creative or unconventional strategies, a vital ingredient in advanced heuristic design. By developing refined diversity metrics and preserving variability throughout the evolutionary process, researchers can unlock a broader range of potential solutions, leading to improved performance and more generalizable outcomes.

This chapter has charted the emerging partnership between EC and LLMs, showcasing how their combined strengths can transform AHD. In LLM-EPS, LLMs contribute a wellspring of fresh heuristic ideas, and evolutionary methods systematically sharpen and validate these concepts. Yet, to extract maximum value from this synergy, careful attention must be paid to diversity the balancing force that fosters both robust exploration and refined exploitation.

From traditional GA to newer frameworks like FunSearch, EoH, and ReEvo, the literature consistently highlights the importance of extensive exploration to avoid becoming trapped in suboptimal solutions. Diversity is the crucial component that integrates a genuinely adaptive search process, facilitating breakthroughs in problem-solving. In the following chapter, advanced methods for incorporating diversity into LLM-EPS will be examined.

CHAPTER 2. DIVERSITY MEASUREMENT METRICS

This chapter introduces a method for encoding the LLM-EPS population and proposes two metrics for measuring diversity: SWDI and CDI. Following this, a diversity analysis will be conducted on previous LLM-EPS frameworks, including FunSearch, EoH, and ReEvo.

2.1 Population encoding

One particular problem in measuring diversity in LLM-EPS is how to encode the population. While each individual in the traditional evolutionary algorithm is encoded as a vector, in LLM-EPS they are usually presented as a string of code snippets/programs. This poses a challenge in applying previous diversity metrics to the population of LLM-EPS frameworks. To address this challenge, an encoding method is proposed that involves three steps: (i) removing comments and docstrings using abstract-syntax tree, (ii) standardizing code snippets into a common coding style (e.g., PEP8[1]), (iii) converting code snippets to vector representations using a code embedding model.

2.2 Shannon–Wiener diversity index

Inspired by ecological studies, [32] provides a quantitative measure of species diversity within a community. In the context of search algorithms, this index aims to quantify the diversity of the population at any given time step based on clusters of individuals. To compute the SWDI for a specific set of individuals within a community, or archive, it is first necessary to determine the probability distribution of individuals across the clusters. This is represented as:

$$p_i = \frac{|C_i|}{M} \tag{2.1}$$

where C_i is a cluster of individuals and M represents the total number of individuals across all clusters $\{C_1, \dots, C_N\}$. The SWDI, $H(X)$, is then calculated using the Shannon entropy:

$$H(X) = -\sum_{i=1}^{N} p_i \log(p_i) \tag{2.2}$$

This index serves as a crucial metric in maintaining the balance between exploration and exploitation within heuristic search spaces. A higher index score suggests a more uniform distribution of individuals across the search space, thus pro-

[1]https://peps.python.org/pep-0008

moting exploration. Conversely, a lower index score indicates concentration around specific regions, which may enhance exploitation but also increase the risk of premature convergence.

To obtain the SWDI score, at each time step t, all individuals $R_i = \{r_1, \ldots, r_n\}$ generated during that time step are added to an archive. The population encoding method described in Section 2.1 is utilized to derive vector representations $\mathcal{V} = \{v_1, \ldots, v_n\}$. Following this, cosine similarity is used to compute the similarity between two embedding vectors v_i and v_j:

$$\text{similarity}(v_i, v_j) = \frac{v_i \cdot v_j}{\|v_i\|\|v_j\|} \tag{2.3}$$

To identify clusters in the archive, each embedding vector v_i is analyzed. Next, v_i will be assigned to a cluster C_i if the similarity between v_i and all members of C_i exceeds a specified threshold α. In mathematical terms, v_i is assigned to C_i if:

$$\text{similarity}(v_i, v_k) \geq \alpha \quad \forall k \in \{1, \ldots, |C_i|\} \tag{2.4}$$

If no cluster C_i in $\{ C_1, \ldots, C_N \}$ meets the condition, a new cluster C_{N+1} will be created and v_i will be assigned to C_{N+1}. Finally, the SWDI computed score is determined by calculating Equations (2.1) and (2.2) across the identified clusters.

2.3 Cumulative diversity index

While SWDI focuses on diversity of different groups of individuals, the CDI plays a crucial role in understanding the spread and distribution of the whole population within the search space [33], [34]. In the context of heuristic search, the CDI measures how well a system's energy, or diversity, is distributed from a centralized state to a more dispersed configuration.

Many entropy interpretations have been suggested over the years. The best known are disorder, mixing, chaos, spreading, freedom and information [35]. The first description of entropy was proposed by Boltzmann to describe systems that evolve from ordered to disordered states. Spreading was used by Guggenheim to indicate the diffusion of a energy system from a smaller to a larger volume. Lewis stated that, in a spontaneous expansion gas in an isolated system, information regarding particles locations decreases while, missing information or uncertainty increases.

In graph theory, each edge can be assigned a weight representing, for example, the energy required to traverse that edge. A graph with a complex structure and varying edge weights exhibits higher entropy, indicating a more disordered system.

Conversely, a simpler structure with uniformly distributed weights corresponds to lower entropy.

An Minimum Spanning Tree (MST) of a connected, edge-weighted graph is a subset of the edges that connects all vertices without any cycles and with the minimum possible total edge weight. Constructing an MST can be seen as a process of transitioning from a high-entropy state (the original graph with all its edges and weights) to a lower-entropy state. This transition reflects the idea of energy distribution from a "better located" state to a "more distributed" one, as the MST represents the most efficient way to connect all vertices, minimizing the total "energy" or weight.

Consider a connected, undirected graph $G = (V, E)$ with a weight function $w : E \to \mathbb{R}^+$. The entropy H of the graph can be associated with the distribution of edge weights. One way to define this entropy is:

$$H = -\sum_{e \in E} p(e) \log p(e) \tag{2.5}$$

$$p(e) = \frac{w(e)}{\sum_{e' \in E} w(e')} \tag{2.6}$$

where $p(e)$ is probability-like measure of edge e based on its weight.

The MST, $T = (V, E_T)$, is a subgraph where $E_T \subseteq E$ and the total weight $\sum_{e \in E_T} w(e)$ is minimized. By selecting the subset of edges that connect all vertices with the minimal total weight, the MST reduces the system's complexity and, consequently, its entropy.

To calculate the CDI, let's consider all individuals within an archive, represented by their respective embeddings. By constructing a MST that connects all individuals within the archive A, where each connection between individuals is calculated using Euclidean distance. This MST provides a structure to assess the diversity of the population. Let d_i represent the distance of an edge within the MST, where $i \in \{1, 2, \ldots, \#A - 1\}$. The probability distribution of these distances is given by:

$$p_i = \frac{d_i}{\sum_{j=1}^{\#A-1} d_j} \tag{2.7}$$

The cumulative diversity is then calculated using the Shannon entropy:

$$H(X) = -\sum_{i=1}^{\#A-1} p_i \log(p_i) \tag{2.8}$$

This method enables the collection of the complete diversity present in the search space, offering valuable information about the range of solutions. Higher CDI values indicate a more distributed and diverse population, which is essential for maintaining a robust search process.

An interesting point in Shannon Diversity Index (SDI) theory is that normalizing the SDI with the natural log of richness is equivalent to the evenness value $\frac{H'}{\ln(S)}$ where H' represents the richness of SDI and S is the total number of possible categories [36]. The significance of evenness is to demonstrate how the proportions of p_i are distributed.

Now, consider normalizing with the two current diversity indices. Firstly, in the case of SWID, if there are N clusters and the count of individuals in each cluster from C_1 to C_N is uniform, then the evenness will consistently equal 1, regardless of the value of N. This is not the preferred outcome since the number of clusters should be considered as a factor of diversity. Likewise, normalizing the CDI will also overlook the influence of the number of candidate heuristics, which correspond to the nodes in the MST. This leads to the proposal not to normalize the SWDI and CDI metrics to achieve a $[0, 1]$ bound.

2.4 Exploring the correlation between objective score and diversity measurement metrics

To investigate the relationship between the two diversity measurement metrics and the objective score, three experimental runs were performed using ReEvo on three distinct AHD problems, Bin Packing Online (BPO), Traveling Salesman Problem (TSP) with Guided Local Search (GLS) [37], and Orienteering Problem (OP) with Ant Colony Optimization (ACO) solver [38]. Details of the experiments can be found in Section 4.1. The diversity evaluation process was conducted with the code embedding model used is `CodeT5+`[2] [39], and similarity threshold $\alpha = 0.95$ where $\alpha \in [0; 1]$. The maximum budget for each run is 450K tokens. The objective scores and the two diversity metrics from these runs on BPO, TSP, and OP are presented in Figure 2.1, Figure 2.2, and Figure 2.3, respectively.

From figure 2.1, there are a few observations that can be drawn. First, the two diversity measurement metrics have a noticeable correlation with the objective scores

[2]https://huggingface.co/Salesforce/codet5p-110m-embedding

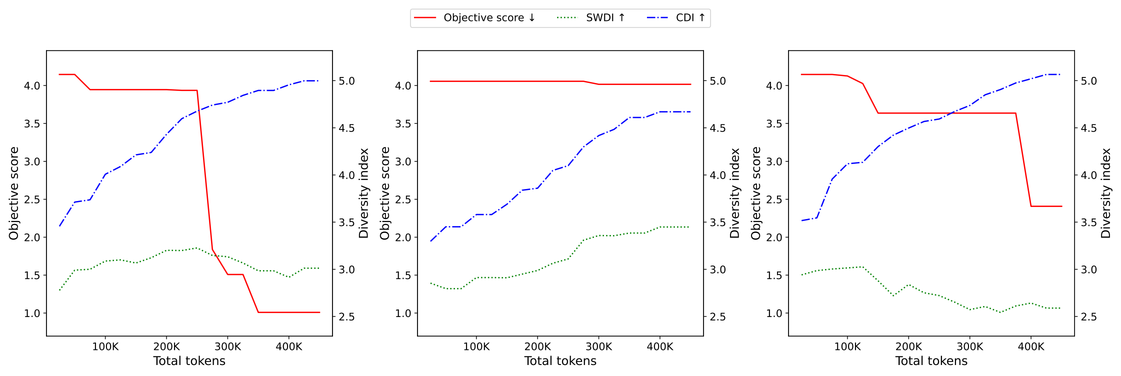


Figure 2.1: Diversity indices and objective scores of ReEvo framework on BPO problem through different runs.

of the problem. When the objective score is converged into a local optima, the framework either tries to focus on the exploration by increasing the diversity of the population, through the increase of SWDI in the first and second runs or tries to focus on the exploitation of the current population, thus decrease the SWDI in the third run. Focusing on exploration can lead to considerable gains in the objective score, whereas concentrating on exploitation may cause the population to remain stuck in local optima for an extended period. However, if the whole population is not diverse enough, as can be seen with the low CDI of the second run, the population might not be able to escape the local optima. These observations about the correlation between the objective score and diversity measurement metrics are similarly reflected in Figures 2.2 and 2.3.

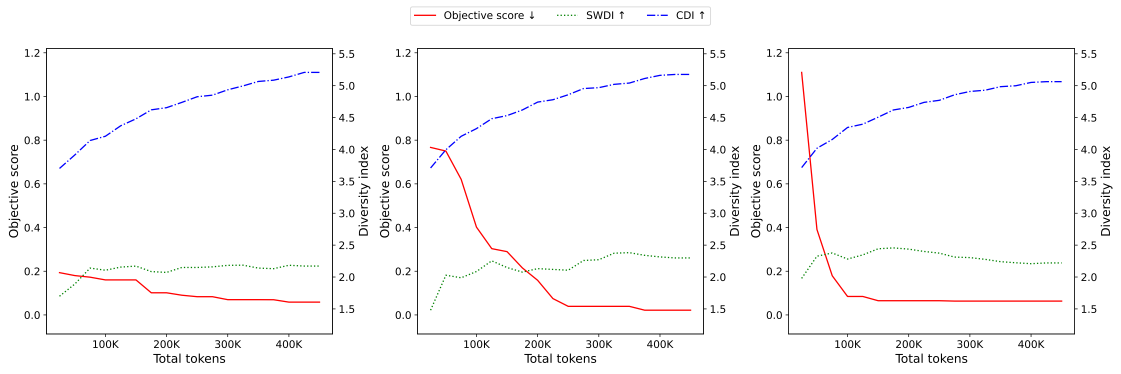


Figure 2.2: Diversity indices and objective scores of ReEvo framework on TSP through different runs.

2.5 Diversity analysis on previous LLM-EPS framework

To analyze the overall diversity of previous LLM-EPS frameworks, experiments were conducted on three distinct LLM-EPS frameworks: FunSearch [13], ReEvo [15], and EoH [14], focusing on three separate AHD problems (BPO, TSP, OP). The details of each experiment are presented in Section 4.1. Note that, as high-

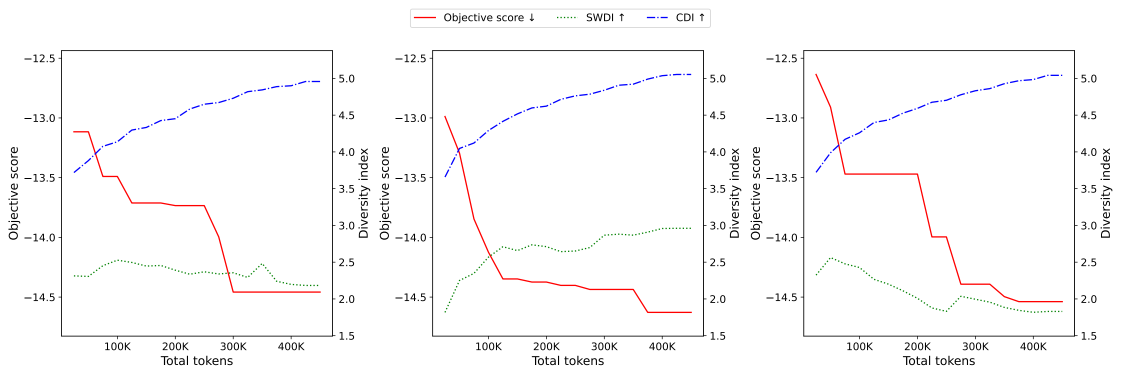


Figure 2.3: Diversity indices and objective scores of ReEvo framework on OP through different runs.

lighted in the previous section, SWDI focuses on understanding the diversity of the population during a single run. As such, it may not help quantify the diversity across multiple experiment runs. Fig. 2.4 presents the experiment results.

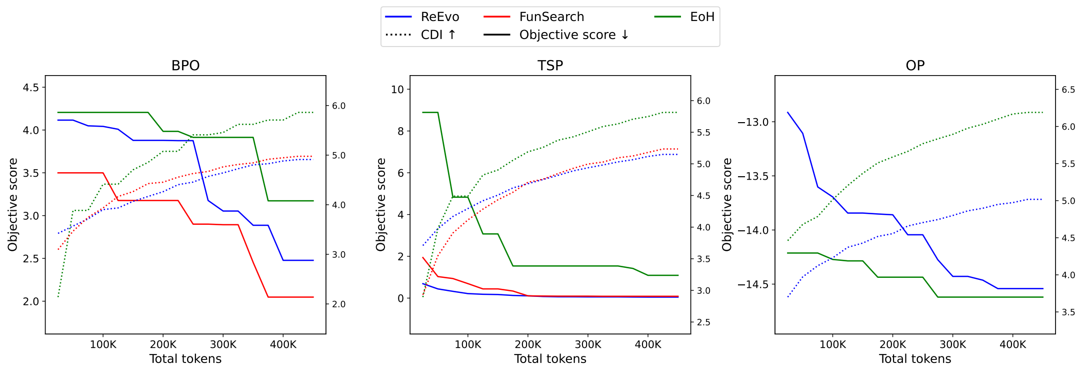


Figure 2.4: CDI and objective scores of previous LLM-EPS on different AHD problems.

In BPO and TSP, EoH obtained the highest CDI but got the worst objective score. This implies that EoH does not focus enough on exploitation to optimize the population better. In contrast, while ReEvo and FunSearch obtain lower CDI than EoH on BPO and TSP, they achieve better objective performance on all three problems. The experiments conducted on the BPO and TSP problems illustrate the inherent trade-off between diversity and objective performance in LLM-EPS frameworks. However, for the OP problem, it is evident that a high CDI is necessary to achieve a better objective score, which aligns with the findings discussed in Section 2.4.

In conclusion, this chapter has introduced a comprehensive approach to measuring diversity within LLM-EPS frameworks, focusing on SWDI and CDI. The proposed population encoding method addresses the unique challenge of representing code snippets as vectors, enabling the application of these diversity metrics. Through extensive experiments on various AHD problems, the correlation between diversity metrics and objective scores has been explored, revealing the delicate balance between exploration and exploitation in heuristic search spaces. The analysis of previous LLM-EPS frameworks, including FunSearch, ReEvo, and EoH, underscores the trade-offs between diversity and performance, highlighting the importance of maintaining an optimal level of diversity to avoid premature convergence and enhance solution quality.

CHAPTER 3. AUTOMATIC HEURISTIC DESIGN WITH HSEVO

This chapter introduces a novel LLM-EPS framework called **H**armony **S**earch **Evo**lution (HSEvo). HSEvo aims to enhance the diversity of the population while improving optimization performance and relieving the trade-off between these two aspects through an individual tuning process based on harmony search. Additionally, the framework aims to minimize costs associated with LLMs by incorporating an efficient flash reflection component. Figure 3.1 illustrates the pipeline of the HSEvo framework.

3.1 Automatic heuristic design with HSEvo

Individual encoding. In HSEvo, each individual within the evolutionary process is represented as a string of executable code (Figure 3.2a). This concept is a direct inheritance from earlier LLM-EPS such as EoH and ReEvo. This code snippet, generated by a LLMs, embodies a specific heuristic function tailored to the target optimization problem. This encoding method is deliberately flexible, eschewing any predefined format or structure, which allows for the generation of diverse and complex heuristics. Unlike traditional methods that might use fixed-length vectors or predefined data structures. HSEvo's code-based approach enables the expression of a wide range of algorithmic strategies, including conditional logic, loops, and function calls, making it highly adaptable to various problem domains.

The choice of code as the encoding mechanism also facilitates direct evaluation of the generated heuristics. Each code snippet is directly executable, either within a sandboxed environment or through a standard Python subprocess, simplifying the evaluation process and eliminating the need for intermediate translation steps. This direct executability is particularly advantageous for AHD problems, where the goal is to automatically discover effective heuristics.

Initialization. HSEvo initializes a heuristic population by prompting the generator LLM with task specifications that describe the problem and detail the signature of the heuristic function to be searched. Additionally, to leverage existing knowledge, the framework can be initialized with a seed heuristic function and/or external domain knowledge. To promote diversity, prompts are created with various role instructions (e.g., "You are an expert in the domain of optimization heuristics...", "You are Albert Einstein, developer of relativity theory..."), details instructions in the next Section 3.2.1. This approach aims to enrich the heuristic generation process by incorporating diverse perspectives and expertise.

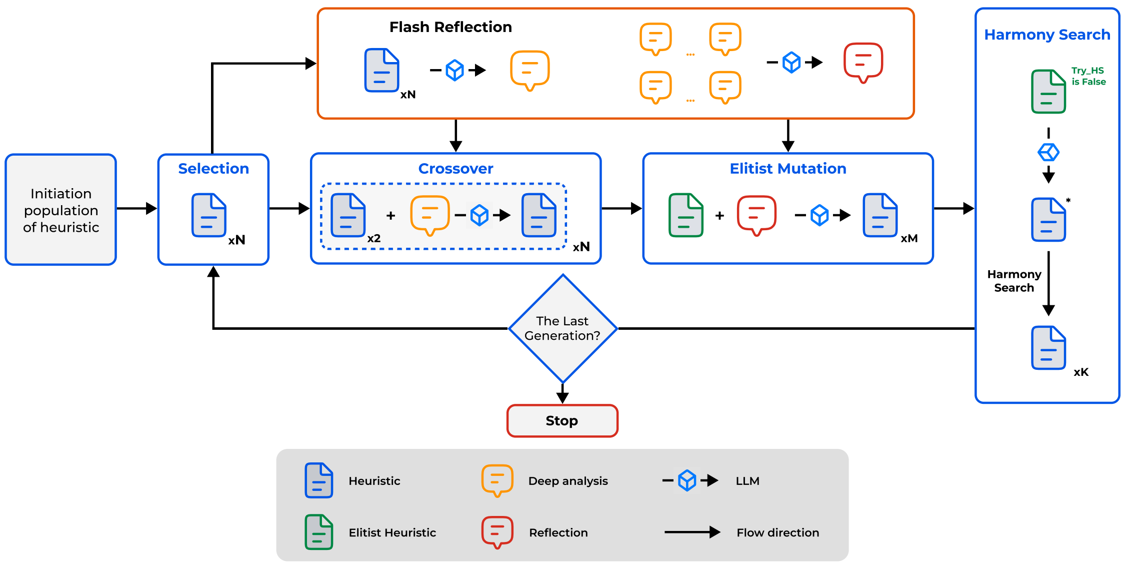


Figure 3.1: Overview of the HSEvo framework.

Selection. In HSEvo, the selection of parent pairs for crossover is performed using a random selection strategy. This approach involves choosing two individuals from current population with equal probability, without any bias towards their fitness or performance. The primary goal of this random selection is to maintain a balance between exploration and exploitation during optimization process. By not favoring the best-performing individuals, HSEvo ensures that search space is not prematurely narrowed, allowing for the discovery of potentially novel and effective heuristics that might be overlooked by more deterministic selection methods.

Furthermore, random selection mechanism is designed to counteract premature convergence, a phenomenon where the population becomes homogeneous too quickly, hindering further exploration. This is particularly important in the context of LLM-EPS, where the initial population might be biased towards certain types of solutions. The random selection helps to maintain diversity within the population, preventing the search from getting stuck in local optima and promoting a more thorough exploration of the heuristic space. This is supported by observations of the SWDI trajectory in Section 2.4, which suggests that random selection can help to maintain a broader search width, thus avoiding premature convergence.

Flash reflection. Reflections can provide LLMs with reinforcement learning reward signals in a verbal format for code generation tasks, as discussed by [40]. Later, [15] also proposed integrating Reflections into LLM-EPS in ReEvo as an approach analogous to providing "verbal gradient" information within heuristic spaces. HSEvo argue that reflecting on each pair of heuristic parents individually is not generalized and wastes resources. To address these issues flash reflection proposed to alternative Reflection method for LLM-EPS. The flash reflection process is structured into two distinct steps.

- First, at each time step t, the individuals selected during the selection phase are grouped, and any duplicate individuals are removed. This ensures that the analysis is performed on a unique set of heuristics. The LLM then undertakes a deep analysis of the performance ranking of these individuals. This take inputs as mixed ranking pairs (i.e., one good performance parent and one worse performance), good ranking pairs (i.e., both good performance), and worse ranking pairs (i.e., both worse performance), then return a comprehensive description on how to improve as a text string.
- The second step, flash reflection involves comparing the current analysis at time step t with the previous analysis from time step $t-1$. This comparison is performed by the LLM, which then generates guide information based on the differences and similarities between the two analyses. This guide information is designed to be used in subsequent stages of the evolutionary process (e.g. crossover and mutation) to further refine the heuristics. By incorporating this temporal aspect, flash reflection allows LLM to learn from its past experiences and adapt its code generation strategy over time. This iterative process of analysis and comparison enables LLM to continuously improve its understanding of the heuristic space and generate increasingly effective solutions.

In essence, flash reflection in HSEvo acts as a form of meta-learning, where the LLM not only generates heuristics but also learns how to generate better heuristics over time. By providing a more generalized and resource-efficient approach to reflection, HSEvo aims to overcome the limitations of previous methods and achieve more effective and efficient heuristic discovery. The textual feedback generated by flash reflection provides a rich source of information that guides the LLM towards generating more promising solutions, ultimately leading to improved performance in the target optimization problem.

Crossover. In this stage, new offspring algorithms are generated by combining elements of the parent algorithms. The goal is to blend successful attributes from two parents via guide result guide information part of flash reflections. Through this step, HSEvo hopes to produce offspring that inherit the strengths of both, which can potentially lead to better-performing heuristics. The prompt includes task specifications, a pair of parent heuristics, guide information part of flash reflection, and generation instructions.

Elitist mutation. HSEvo employs an elitist mutation strategy, focusing its mutation efforts on the "elite" individual, which represents the best-performing heuristic found so far. This approach leverages the generator LLM to mutate this elite individual, incorporating insights derived from the LLM-analyzed flash reflections. The mutation process is not random; instead, it is guided by the accumulated knowledge and feedback from the flash reflection, aiming to enhance the elite heuristic's performance. Each mutation prompt includes detailed task specifications, the code of the elite heuristic, the deep analysis from the flash reflections, and specific generation instructions, providing the LLM with the necessary context to perform a targeted and informed mutation.

This elitist mutation strategy is designed to ensure continuous improvement in performance while preserving the quality of the top solutions. By focusing on the best-performing heuristic, HSEvo aims to refine and enhance its capabilities, rather than randomly exploring the search space. The incorporation of insights from the flash reflections ensures that the mutation process is not arbitrary but rather directed towards making meaningful improvements based on the accumulated knowledge of the evolutionary process. This approach allows HSEvo to effectively exploit the best solutions found so far, while still allowing for exploration through other mechanisms like crossover and the initial population generation.

Harmony Search. From the analysis in Section 2.4 and 2.5. The hypothesis is that if the population becomes too diverse, individual heuristics within it are more likely to be unoptimized, which can negatively impact the overall optimization process. To mitigate this, HSEvo employs HS to fine-tune the parameters (e.g., thresholds, weights, etc.) of the best-performing individuals in the population. The process is as follows:

- First, once an individual is selected, the LLM is tasked with extracting parameters from the best individual (i.e., code snippets, programs) of the population and defining a range for each parameter (Figure 3.2).
- Following the parameter extraction, HSEvo employs the HS algorithm, a metaheuristic optimization technique inspired by the improvisation process of musicians. The HS algorithm iteratively adjusts the extracted parameters within their defined ranges, aiming to find a combination that maximizes the heuristic's performance. This process involves creating a "harmony memory," which stores a set of parameter combinations, and iteratively generating new harmonies by adjusting existing ones or creating new ones randomly. The performance of each harmony is evaluated, and the harmony memory is updated with better-performing combinations. This iterative process continues for a

predefined number of iterations, allowing the HS algorithm to explore parameter space and identify optimal or near-optimal parameter settings.

- After the parameter optimization is complete, the optimized individual is marked to prevent it from being optimized again in future time steps. This ensures that the HS algorithm is not repeatedly applied to the same individuals, allowing the framework to focus on optimizing other promising heuristics. The optimized individual is then added back to the population, contributing to the overall diversity and performance of the population. This integration of HS into the evolutionary process allows HSEvo to fine-tune the best-performing heuristics, enhancing their performance and mitigating the negative effects of excessive diversity.

To sum up, the HS component in HSEvo is a critical mechanism for enhancing the performance of the best-performing heuristics by fine-tuning their parameters. This process involves using the LLMs to extract parameters and define their ranges, followed by the application of the HS algorithm to optimize these parameters. The optimized individuals are then added back to the population, contributing to the overall performance and diversity of the population. This strategic integration of HS allows HSEvo to effectively balance exploration and exploitation, leading to the discovery of more robust and high-performing heuristics.

An overview of the flow of HSEvo is also described through pseudo-algorithm 1, where `function_evals` < `max_fe` serves as the stopping condition. Here, `function_evals` is a counter variable tracking the number of evaluated individuals, and `max_fe` is the maximum number of individuals HSEvo will evaluate.

Algorithm 1: HSEvo Framework

```
Data: Configuration files, problem-specific parameters
Result: Best code found and its path
Initialization: Load configuration, initialize LLM prompts, and generate
  initial population;
while $function\_evals < max\_fe$ do
    Selection: Select parents randomly from the population;
    if selection fails then
        Terminate the loop;
    end
    Reflection: Perform flash and comprehensive reflection using LLM;
    Update long-term memory;
    Crossover: Generate and evaluate new individuals by crossing parents
      using LLM;
    Mutation: Generate and evaluate new individuals by mutating the best
      individual using LLM;
    Harmony Search: for $i \leftarrow 1$ to $3$ do
        Select an individual for harmony search;
        Extract parameters and code for harmony search using LLM;
        if extraction fails then
            Continue;
        end
        Initialize harmony memory and create a new population;
        while harmony search is ongoing do
            Update harmony memory and population;
        end
        if harmony search succeeds then
            Add the best individual to the population;
            Break;
        end
    end
    Update: Update the best individual (elitist);
    Increment iteration count and log results;
    Check and update reflection lists based on elitist changes;
end
Return the best code found and its path;
```

3.2 HSEvo prompt examples

This section synthesizes the prompts used in the HSEvo framework. The HSEvo prompt system includes four stages: initialization, population, flash reflection, crossover, and elitist mutation.

3.2.1 Task description prompts

Prompt 1: System prompt for generator LLM.

{role_init} helping to design heuristics that can effectively solve optimization problems.
Your response outputs Python code and nothing else. Format your code as a Python code string: "`python ... "`.

Prompt 2: Task description.

{role_init} Your task is to write a {function_name} function for {problem_description}
{function_description}

In this work, the variable `role_init` is utilized to assign various roles to the LLMs, allowing the creation of distinct personas [41] that can provide different perspectives and promote population diversity. The roles are chosen in a round-robin sequence from the list:

- *You are an expert in the domain of optimization heuristics,*
- *You are Albert Einstein, relativity theory developer,*
- *You are Isaac Newton, the father of physics,*
- *You are Marie Curie, pioneer in radioactivity,*
- *You are Nikola Tesla, master of electricity,*
- *You are Galileo Galilei, champion of heliocentrism,*
- *You are Stephen Hawking, black hole theorist,*
- *You are Richard Feynman, quantum mechanics genius,*
- *You are Rosalind Franklin, DNA structure revealer,*
- *You are Ada Lovelace, computer programming pioneer.*

Variable `function_name`, `problem_description` and `function_description` correspond to the heuristic function name, the specific problem description, and the function description, respectively, which will be detailed in the section 3.2.7.

3.2.2 Population initialization prompt

Prompt 3: User prompt for population initialization.

{task_description}
{seed_function}
Refer to the format of a trivial design above. Be very creative and give '{function_name}_v2'. Output code only and enclose your code with Python code block: ```` ```python ... ``` ````.

Here, `task_description` refer to Prompt 2, `seed_ function` is a trivial heuristic function for the corresponding problem.

3.2.3 Flash reflection prompts

Prompt 4: System prompt for reflector LLM.

You are an expert in the domain of optimization heuristics. Your task is to provide useful advice based on analysis to design better heuristics.

Prompt 5: User prompt for flash reflection pharse 1.

List heuristics
Below is a list of design heuristics ranked from best to worst.
{list_ranked_heuristics}
Guide
- Keep in mind, list of design heuristics ranked from best to worst. Meaning the first function in the list is the best and the last function in the list is the worst.
- The response in Markdown style and nothing else has the following structure:
‘**Analysis:**
Experience:’
In there:
+ Meticulously analyze comments, docstrings and source code of several pairs (Better code - Worse code) in List heuristics to fill values for **Analysis:**.
Example: “Comparing (best) vs (worst), I see ...; (second best) vs (second worst) ...; Comparing (1st) vs (2nd), I see ...; (3rd) vs (4th) ...; Comparing (worst) vs (second worst), I see ...; Overall:...”
+ Self-reflect to extract useful experience for design better heuristics and fill to **Experience:** (< 60 words).
I’m going to tip $999K for a better heuristics! Let’s think step by step.

`list_ranked_heuristics` is a list of heuristic functions obtained by grouping parent pairs from selection, removing duplicates, and ranking based on the objective scores.

Prompt 6: User prompt for flash reflection pharse 2.

Your task is to redefine ‘Current self-reflection’ paying attention to avoid all things in ‘Ineffective self-reflection’ in order to come up with ideas to design better heuristics.
Current self-reflection
{current_reflection}
{good_reflection}
Ineffective self-reflection
{bad_reflection}
Response (<100 words) should have 4 bullet points: Keywords, Advice, Avoid, Explanation.
I’m going to tip $999K for a better heuristics! Let’s think step by step.

`current_reflection` is the output of the flash reflection step 1 from the current generation. `good_reflection` is the output of the flash reflection step 2 from previous generations where a new heuristic was discovered. Conversely, `bad_reflection` is the output of the flash reflection step 2 from previous generations where no new heuristic was discovered.

3.2.4 Crossover prompt

Prompt 7: User prompt for crossover.

{task_description}
Better code
{function_signature_better}
{code_better}
Worse code
{function_signature_worse}
{code_worse}
Analyze & experience
- {flash_refection}
Your task is to write an improved function ‘{func_name}_v2‘ by COMBINING elements of two above heuristics base Analyze & experience.
Output the code within a Python code block: “‘python ... “‘, has comment and docstring (<50 words) to description key idea of heuristics design.
I’m going to tip $999K for a better heuristics! Let’s think step by step.

`function_signature_better`, `code_better` and `function_ signature_worse`, `code_worse` are the function signatures and code of the better and worse parent individuals, respectively. `flash_refection` is the output of the flash reflection step 2 from the current generation.

3.2.5 Elitist mutation prompt

Prompt 8: System prompt for elitist mutation.

{task_description}
Current heuristics:
{function_signature_elitist}
{elitist_code}
Now, think outside the box write a mutated function '{func_name}_v2' better than current version. You can using some hints if need:
{flash_reflection}
Output code only and enclose your code with Python code block: ```python ... ```.
I'm going to tip $999K for a better solution!

`function_signature_elitist`, `elitist_code` are the function signatures and code of elitist individuals of current generation.

3.2.6 Harmony search prompts

Prompt 9: System prompt for Harmony Search.

You are an expert in code review. Your task extract all threshold, weight or hardcode variable of the function make it become default parameters.

Prompt 10: User prompt for Harmony Search.

{elitist_code}
Now extract all threshold, weight or hardcode variable of the function make it become default parameters and give me a 'parameter_ranges' dictionary representation. Key of dict is name of variable. Value of key is a tuple in Python MUST include 2 float elements, first element is begin value, second element is end value corresponding with parameter.
- Output code only and enclose your code with Python code block: ```python ... ```.
- Output 'parameter_ranges' dictionary only and enclose your code with other Python code block: ```python ... ```.

`elitist_code` is the snippet/string code of the elitist individual in the current population that has not yet undergone harmony search.

Table 3.1: Problem descriptions used in prompts

Problem	Problem description
BPO	Solving online Bin Packing Problem (BPP). BPP requires packing a set of items of various sizes into the smallest number of fixed-sized bins. Online BPP requires packing an item as soon as it is received.
TSP	Solving Traveling Salesman Problem (TSP) via guided local search. TSP requires finding the shortest path that visits all given nodes and returns to the starting node.
OP	Solving a black-box graph combinatorial optimization problem via stochastic solution sampling following "heuristics".

Table 3.2: Function descriptions used in prompts

Problem	Problem description
BPO	The priority function takes as input an item and an array of bins_remain_cap (containing the remaining capacity of each bin) and returns a priority score for each bin. The bin with the highest priority score will be selected for the item.
TSP	The 'update_edge_distance' function takes as input a matrix of edge distances, a locally optimized tour, and a matrix indicating the number of times each edge has been used. It returns an updated matrix of edge distances that incorporates the effects of the local optimization and edge usage. The returned matrix has the same shape as the input 'edge_distance' matrix, with the distances adjusted based on the provided tour and usage data.
OP	The 'heuristics' function takes as input a vector of node attributes (shape: n), a matrix of edge attributes (shape: n by n), and a constraint imposed on the sum of edge attributes. A special node is indexed by 0. 'heuristics' returns prior indicators of how promising it is to include each edge in a solution. The return is of the same shape as the input matrix of edge attributes.

3.2.7 Problem-specific prompts

Problem-specific prompts are given below:

- Table 3.1 presents `problem_description` for all problems.
- Table 3.2 lists the `function_description` for all problem settings.
- Figure 3.3 shows the `function_signature`, which also includes the `function_name` of each problem.
- Figure 3.6 shows the `seed_function` used for each problem.

3.3 Generated heuristics

The best heuristics generated by HSEvo for all problem settings are presented in Figure 3.9, 3.10, 3.11, 3.12, 3.13, and 3.14.

To summarize, this chapter presented HSEvo, an innovative LLM-EPS framework that transforms heuristic optimization by integrating population diversity and performance through advanced techniques such as flash reflection and harmony search. HSEvo utilizes code-based individual encoding to enhance its adaptability in addressing a wide range of intricate optimization challenges. Key components, such as the random selection mechanism, flash reflection, and elitist mutation, ensure thorough exploration of the heuristic space while preventing premature convergence. By incorporating the HS algorithm for fine-tuning parameters, HSEvo further optimizes its best-performing heuristics, achieving a synergy between exploration and exploitation. Along with this, detailed descriptions of prompts, pseudocode, and the best heuristics on various benchmarks using HSEvo are provided.

```python
import numpy as np

def heuristics_v2(prize: np.ndarray,
distance: np.ndarray, maxlen: float) -> np.ndarray:
    reward_distance_ratio = prize / distance
    cost_penalty = np.exp(-distance)

    heuristics = (prize * prize[:, np.newaxis])
    heuristics[distance > maxlen] = 0

    return heuristics
```

(a) Origin Code

```python
import numpy as np

def heuristics_v2(prize: np.ndarray,
distance: np.ndarray, maxlen: float,
reward_threshold: float = 0,
distance_threshold: float = 0,
cost_penalty_weight: float = 1) -> np.ndarray:
    reward_distance_ratio = prize / distance
    cost_penalty = np.exp(-distance)

    heuristics = (prize * prize[:, np.newaxis]) / (distance * distance) * cost_penalty
    heuristics[(distance > maxlen) | (reward_distance_ratio < reward_threshold) | (distance < distance_threshold)] = 0

    return heuristics

parameter_ranges = {
    'reward_threshold': (0, 1),
    'distance_threshold': (0, 100),
    'cost_penalty_weight': (0, 2)
}
```

(b) Modified Code using LLMs

Figure 3.2: An example of how the harmony search component works in HSEvo.

```python
def priority_v{version}(item: float, bins_remain_cap: np.ndarray) -> np.ndarray:
```

Figure 3.3: Function signatures used in HSEvo for BPO.

```python
def update_edge_distance_v{version}(edge_distance: np.ndarray, local_opt_tour: np.ndarray, edge_n_used: np.ndarray) -> np.ndarray:
```

Figure 3.4: Function signatures used in HSEvo for TSP.

```python
def heuristics_v{version}(node_attr: np.ndarray, edge_attr: np.
    ndarray, node_constraint: float)->np.ndarray:
```

Figure 3.5: Function signatures used in HSEvo for OP.

```python
def priority_v1(item: float, bins_remain_cap: np.ndarray) -> np.
    ndarray:
    """Returns priority with which I want to add item to each bin
        .

    Args:
        item: Size of item to be added to the bin.
        bins_remain_cap: Array of capacities for each bin.

    Return:
        Array of same size as bins_remain_cap with priority score
            of each bin.
    """
    ratios = item / bins_remain_cap
    log_ratios = np.log(ratios)
    priorities = -log_ratios
    return priorities
```

Figure 3.6: Seed heuristics used in HSEvo for BPO.

```python
def update_edge_distance(edge_distance: np.ndarray,
    local_opt_tour: np.ndarray, edge_n_used: np.ndarray) -> np.
    ndarray:
     """
     Args:
         edge_distance (np.ndarray): Original edge distance matrix
            .
         local_opt_tour (np.ndarray): Local optimal solution path.
         edge_n_used (np.ndarray): Matrix representing the number
            of times each edge is used.
     Return:
         updated_edge_distance: updated score of each edge
            distance matrix.
     """

     num_nodes = edge_distance.shape[0]
     updated_edge_distance = np.copy(edge_distance)

     for i in range(num_nodes - 1):
         current_node = local_opt_tour[i]
         next_node = local_opt_tour[i + 1]
         updated_edge_distance[current_node, next_node] *= (1 +
            edge_n_used[current_node, next_node])

     updated_edge_distance[local_opt_tour[-1], local_opt_tour[0]]
         *= (1 + edge_n_used[local_opt_tour[-1], local_opt_tour
         [0]])
     return updated_edge_distance
```

Figure 3.7: Seed heuristics used in HSEvo for TSP.

```python
def heuristics_v1(node_attr: np.ndarray, edge_attr: np.ndarray,
    edge_constraint: float)->np.ndarray:
     return np.ones_like(edge_attr)
```

Figure 3.8: Seed heuristics used in HSEvo for OP.

```python
import numpy as np
import random
import math
import scipy
import torch


def priority_v2(item: float, bins_remain_cap: np.ndarray, overflow_threshold: float = 1.0987600915713542, mild_penalty: float = 0.5567025232550017, adaptability_lower: float = 0.7264590977149653, adaptability_higher: float = 1.9441643982922379) -> np.ndarray:
    """Enhanced dynamic scoring function for optimal bin selection in online BPP with a more holistic approach."""

    # Avoid division by zero by adjusting remaining capacities
    adjusted_bins_remain_cap = np.maximum(bins_remain_cap, np.finfo(float).eps)

    # Calculate effective capacities
    effective_cap = np.clip(bins_remain_cap - item, 0, None)
    valid_bins = effective_cap >= 0

    # Calculate occupancy ratios with controlled overflow representation
    occupancy_ratio = item / adjusted_bins_remain_cap
    occupancy_scores = np.where(valid_bins, occupancy_ratio, 0)

    # Enhanced overflow penalty: stronger influence for near-overflows
    overflow_penalty = np.where(occupancy_ratio > overflow_threshold, mild_penalty * (occupancy_ratio - overflow_threshold + 1), 1.0)

    # Logarithmic penalty for remaining capacity to encourage SPACE utilization
    log_penalty = np.where(bins_remain_cap > 0, np.log1p(adjusted_bins_remain_cap / (adjusted_bins_remain_cap - item)), 0)


    # ...
```

Figure 3.9: The best heuristic for BPO found by HSEvo.

```
    #....


    # Adaptability based on remaining mean capacity
    remaining_mean = np.mean(bins_remain_cap[bins_remain_cap >
        0])
    adaptability_factor = np.where(bins_remain_cap <
        remaining_mean, adaptability_lower, adaptability_higher)

    # Comprehensive scoring integrating all metrics for a robust
        approach
    scores = np.where(valid_bins, occupancy_scores *
        overflow_penalty * log_penalty * adaptability_factor, -np.
        inf)

    # Normalize scores for prioritization
    max_score = np.max(scores)
    prioritized_scores = (scores - np.min(scores)) / (max_score -
         np.min(scores)) if max_score > np.min(scores) else scores

    # Invert scores for selecting the highest priority bin
    inverted_priorities = 1 - prioritized_scores

    return inverted_priorities
```

Figure 3.10: The best heuristic for BPO found by HSEvo (continue).

```python
import numpy as np


def update_edge_distance_v2(edge_distance: np.ndarray, local_opt_tour: np.ndarray, edge_n_used: np.ndarray,
                            penalty_factor: float = 0.6713404008357979, bonus_factor: float = 1.343302294236627, decay_factor: float = 0.3821795974433295,
                            scaling_factor: float = 1.116349420562543, min_distance: float = 7.965736386169868e-05,
                            penalty_threshold: float = 29.850922399224466, boost_threshold: float = 70.53785604399908) -> np.ndarray:
    """
    Update edge distances using adaptive penalties and bonuses, considering edge usage dynamics
    while ensuring nuanced performance in line with real-time data patterns.
    """
    num_nodes = edge_distance.shape[0]
    updated_edge_distance = np.copy(edge_distance)

    # Calculate average usage for dynamic adjustments
    avg_usage = np.mean(edge_n_used)

    for i in range(num_nodes):
        current_node = local_opt_tour[i]
        next_node = local_opt_tour[(i + 1) % num_nodes]
        usage_count = edge_n_used[current_node, next_node]

        # Adaptive penalty for overused edges
        if usage_count > penalty_threshold:
            # Penalty increases exponentially with usage
            penalty = penalty_factor * (usage_count - penalty_threshold) ** 2
            updated_edge_distance[current_node, next_node] += penalty
            updated_edge_distance[next_node, current_node] += penalty  # Ensure symmetry


        # ...
```

Figure 3.11: The best heuristic for TSP found by HSEvo.

```python
        # ...


        elif usage_count < boost_threshold:
            # Apply a bonus to underused edges
            boost = bonus_factor * (1 + (boost_threshold - usage_count) * 0.1)
            updated_edge_distance[current_node, next_node] *= boost
            updated_edge_distance[next_node, current_node] *= boost  # Ensure symmetry

        # Dynamic scaling based on average usage
        if usage_count > avg_usage:
            adjustment_factor = scaling_factor / (1 + decay_factor ** (usage_count - avg_usage))
        else:
            adjustment_factor = scaling_factor * (1 + decay_factor ** (avg_usage - usage_count))

        # Update the distance with adjustment and ensure non-negative distances
        updated_edge_distance[current_node, next_node] = max(min_distance,
            updated_edge_distance[current_node, next_node] * adjustment_factor)

    return updated_edge_distance
```

Figure 3.12: The best heuristic for TSP found by HSEvo (continue).

```python
import numpy as np


def heuristics_v2(node_attr: np.ndarray, edge_attr: np.ndarray, node_constraint: float) -> np.ndarray:
    """
    Enhanced heuristics incorporating contextual adjustments, multi-dimensional scoring,
    and adaptive responsiveness to edge conditions.
    """
    normalization_epsilon = 1e-8
    influence_threshold = 0.9
    adaptability_factor = 2.0
    non_linearity_base = 2.5
    n = node_attr.shape[0]
    score_matrix = np.zeros_like(edge_attr)

    # Normalize node attributes (maintain zero division safety)
    total_node_attr = np.sum(node_attr) + normalization_epsilon
    normalized_node_attr = node_attr / total_node_attr

    for i in range(n):
        for j in range(n):
            if i == j:
                continue  # Skip self-loops

            # Calculate contextual adjustment for edge attributes
            dynamic_edge = edge_attr[i, j] ** adaptability_factor
            adjusted_edge = dynamic_edge / (node_constraint + normalization_epsilon)

            # Multi-dimensional scaling based on edge and node attributes
            if adjusted_edge > influence_threshold:
                scaling_factor = np.sqrt(adjusted_edge + normalization_epsilon)
            else:
                scaling_factor = influence_threshold / (adjusted_edge + normalization_epsilon)



            # ...
```

Figure 3.13: The best heuristic for OP found by HSEvo.

```python
            # ...


            # Layered logic for combined node influence using non-linear model
            combined_node_influence = (
                (normalized_node_attr[i] ** non_linearity_base) +
                (normalized_node_attr[j] ** non_linearity_base)
            ) ** 1.5  # Further amplify the influence

            # Score calculation with contextual penalties (non-linearity)
            score = combined_node_influence * scaling_factor

            if dynamic_edge > 0:
                penalty_base = np.log1p(dynamic_edge)  # Non-linear penalty
                penalty_factor = 1 / (1 + penalty_base ** 2)  # Stronger sensitivity with edges
                score *= penalty_factor

            # Normalization to retain meaningful scale
            score_matrix[i, j] = score / (dynamic_edge + normalization_epsilon)

    return score_matrix
```

Figure 3.14: The best heuristic for OP found by HSEvo (continue).

CHAPTER 4. NUMERICAL RESULTS

To unveil the potential of the HSEvo framework. This chapter presents a series of experiments designed to benchmark its performance against existing LLM-EPS methodologies. By tackling a range of optimization challenges, including BPO, TSP, and OP, this thesis examines how effective HSEvo is in improving heuristic design and evolutionary strategies.

4.1 Experimental settings

Benchmarks: To evaluate the diversity and objective scores of HSEvo in comparison to earlier LLM-EPS frameworks, the same benchmarks outlined in Section 2.4 were employed to conduct experiments on three distinct AHD problems: BPO [42], TSP [43], and OP [38].

- BPO: packing items into bins of fixed capacity in real-time without prior knowledge of future items. In this benchmark, LLM-EPS frameworks need to design deterministic constructive heuristics to solve.
- TSP: find the shortest possible route that visits each city exactly once and returns to the starting point. In this setting, LLM-EPS frameworks need to design heuristics to enhance the perturbation phase for the GLS solver.
- OP: find the most efficient path through a set of locations, maximizing the total score collected within a limited time or distance. This setting requires LLM-EPS frameworks to design herustics used by the ACO solver.

Experiment settings: All frameworks were executed under identical environmental settings listed in Table 4.1. The heuristic search processes across the LLM-EPS frameworks and the evaluations of heuristics were conducted using a single core of an Xeon Processors CPU.

4.2 Benchmark dataset details

4.2.1 Bin packing online

Definition. The objective of this problem is to effectively assign a diverse collection of items, each with its own specific size and weight, into the smallest possible number of containers, each having a fixed capacity denoted by C. This challenge is particularly focused on the online packing scenario, where items are introduced one at a time and must be packed immediately upon arrival without prior knowledge of future items. This contrasts with the offline scenario, where all items are available beforehand and can be optimized together.

Table 4.1: Summary of parameters settings

Description of Setting	Value
LLM (generator and reflector)	gpt-4o-mini-2024-07-18
LLM temperature (generator and reflector)	1
Maximum budget tokens	425K tokens
Population size (for EoH, ReEvo and HSEvo)	30 (initial stage), 10 (other stages)
Mutation rate (for ReEvo and HSEvo)	0.5
# of islands, # of samples per prompt (for FunSearch)	10, 4
Number of independent runs per experiment	3
HS size, HMCR, PAR, bandwidth, max iterations (for Harmony Search)	5, 0.7, 0.5, 0.2, 5
Maximum evaluation time for each heuristic	100 sec (TSP-GLS), 50 sec (Other)

Dataset generation. Following [13], this experiment randomly generates five Weibull instances of size 5k with a capacity of $C = 100$. The objective score is set as the average $\frac{lb}{n}$ of five instances, where lb represents the lower bound of the optimal number of bins computed [44] and n is the number of bins used to pack all the items by the evaluated heuristic.

Solver. LLM-EPS is used to design heuristic functions that are able to solve this problem directly without any external solver.

4.2.2 Traveling salesman problem

Definition. The TSP is a well-known combinatorial optimization issue in the field of computer science and operations research. It involves a scenario where a salesman must visit a given set of cities, with the objective of finding the shortest possible route that allows him to visit each city exactly once before returning to his starting point, which is often referred to as the origin city.

Dataset generation. Following [37], this experiment generates a set of 64 TSP instances with 100 nodes (TSP100). The node locations in these instances are randomly sampled from $[0, 1]^2$. This means that the nodes are positioned within a square area bounded by (0, 0) and (1, 1). The average gap from the optimal solution, generated by Concorde [45], is used as the objective score.

Solver. GLS was used as the solver for this benchmark. GLS explores the solution space using local search operations guided by heuristics. The idea behind using this solver is to explore the potential of search penalty heuristics for LLM-

Table 4.2: Problems parameters used for heuristic evaluations

Problem	Problem size	# of iterations	Other
BPO	5000	-	C=100
TSP	100	1000	-
OP	50	50	-

EPS. In this experiment, the traditional GLS algorithm was modified by including perturbation phases [46], where edges with higher heuristic values are given priority for penalization. Settings parameters are listed in Table 4.2, and the number of perturbation moves is 1.

4.2.3 Orienteering problem

Definition. In the OP at hand, the primary aim is to maximize the cumulative score achieved by visiting a series of designated nodes. This objective is subject to the condition that the total length of the tour does not exceed a specified maximum limit. The nodes represent key points of interest, each contributing a certain score, and the challenge lies in strategically planning the route to optimize the overall score while adhering to the constraints of the maximum allowable tour length.

Dataset generation. Following the process of DeepACO [38] during the generation of this synthetic dataset. In each problem instance, uniform distribution is used to sample 50 nodes (OP50), including the depot node, from the unit interval $[0, 1]^2$. This means that the nodes are positioned within a square area bounded by (0, 0) and (1, 1). Especially, a challenging prize distribution [47] also adopt:

$$p_i = \left(1 + \frac{99 \cdot \frac{d_{0i}}{\max_{j=1}^{n} d_{0j}}}{100}\right), \tag{4.1}$$

where d_{0i} represents the distance between the depot and node i, and the maximum tour length constraint is 3.

Solver. For this problem, ACO is used as a solver for this benchmark. ACO is an evolutionary algorithm that integrates solution sampling with pheromone trail updates. It employs stochastic solution sampling, which is biased towards more promising solution spaces based on heuristics. The population size is set at 20 for ACO to solve OP50.

Table 4.3: Experiment results on different AHD problems

Method	BPO		TSP		OP	
	CDI (↑)	Obj. (↓)	CDI (↑)	Obj. (↓)	CDI (↑)	Obj. (↓)
FunSearch	4.97 ± 0.24	2.05 ± 2.01	5.24 ± 0.14	0.09 ± 0.06	-	-
EoH	$\mathbf{5.86 \pm 0.49}$	3.17 ± 2.97	$\mathbf{5.81 \pm 0.23}$	1.09 ± 3.11	$\mathbf{6.17 \pm 0.42}$	-14.62 ± 0.22
ReEvo	4.91 ± 0.53	2.48 ± 3.74	5.15 ± 0.19	0.05 ± 0.06	5.02 ± 0.13	-14.54 ± 0.21
HSEvo	5.68 ± 0.35	$\mathbf{1.07 \pm 1.11}$	5.41 ± 0.21	$\mathbf{0.02 \pm 0.03}$	5.67 ± 0.41	$\mathbf{-14.62 \pm 0.12}$

4.3 Experimental results

Table 4.3 present experiment results on all three AHD problems[1]. From the table, it was observed that while HSEvo still hasn't obtained better CDI than EoH, HSEvo is able to achieve the best objective score on all tasks. On BPO, HSEvo outperforms both FunSearch and ReEvo by a huge margin. This highlights the importance of my findings from the analysis in Section 2.5, where it is crucial to improve diversity in order to optimize the population better.

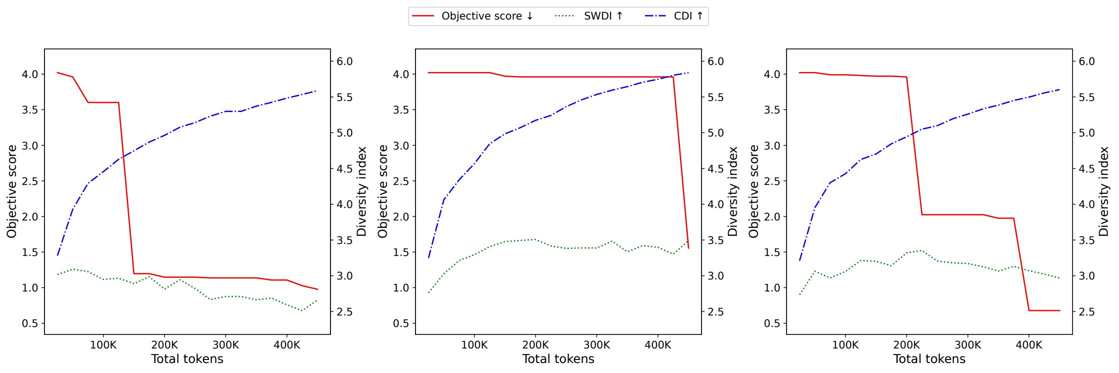


Figure 4.1: Diversity indices and objective scores of HSEvo framework on BPO problem through different runs.

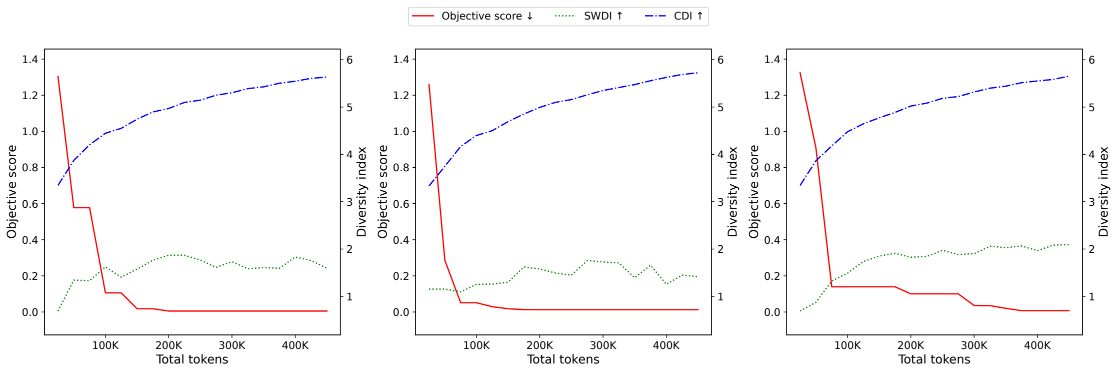


Figure 4.2: Diversity indices and objective scores of HSEvo framework on TSP problem through different runs.

To investigate the impact of my framework on the diversity of the population, Figure 4.1, 4.2, and 4.3 was plotted to illustrate the diversity metrics and objective

[1]Extending FunSearch to solve the OP problem with an ACO solver caused conflicts that could not be resolved with reasonable effort.

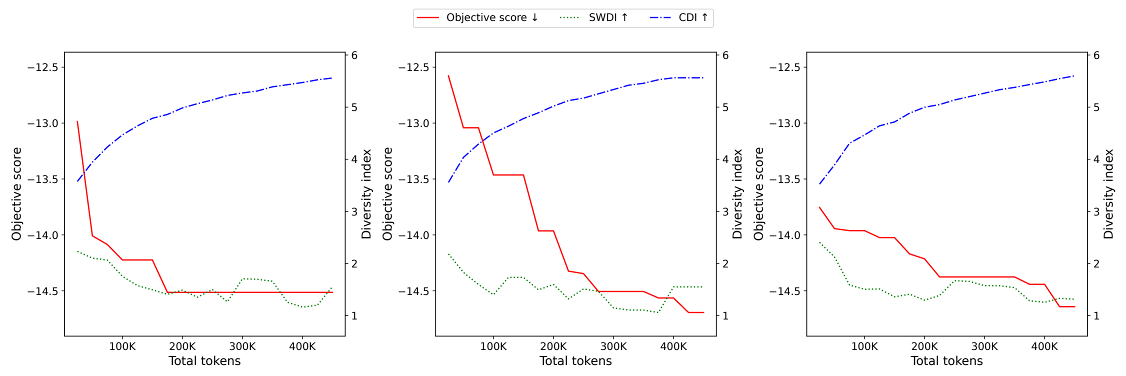


Figure 4.3: Diversity indices and objective scores of HSEvo framework on OP problem through different runs.

Table 4.4: Ablation results on harmony search

Method	OP	
	CDI (↑)	Obj. (↓)
ReEvo	5.02 ± 0.13	-14.54 ± 0.21
ReEvo + HS	5.11 ± 0.27	-14.58 ± 0.38
HSEvo	$\mathbf{5.67 \pm 0.41}$	$\mathbf{-14.62 \pm 0.12}$

score of HSEvo through different runs on BPO, TSP, OP problems, respectively. It can draw a similar observation with findings in Section 2.4, where with a high SWDI and CDI, the objective score can be optimized significantly. One thing to note here is that in HSEvo first run in Figure 4.1 and ReEvo third run in Figure 2.1, both have the SWDI decreasing over time. However, in HSEvo, the SWDI is at around 3.0 when the objective score improves significantly, then only decreases marginally after that. In ReEvo, the objective score improves when SWDI is at around 3.0 and 2.7, and the magnitude of the improvement is not as large as HSEvo, which implies the importance of diversity in the optimization of the problem and also the impact of the proposed harmony search.

4.4 Ablation study

To gain a better understanding of novel framework HSEvo, The experiment extends to conduct ablation studies on proposed components, the harmony search, and flash reflection.

4.4.1 Harmony search analysis

As harmony search is a vital component in HSEvo framework, it will not be eliminated from HSEvo; instead, an experimental setup will incorporate harmony search into the ReEvo framework. The results of the experiment concerning the OP problem are summarized in Table 4.4. The findings indicate that HSEvo surpasses both ReEvo and the variant of ReEvo that includes harmony search, in terms of both

Table 4.5: Ablation results on flash reflection

Method	OP	
	CDI (↑)	Obj. (↓)
ReEvo	4.43 ± 0.23	-13.84 ± 1.13
ReEvo + F.R.	4.63 ± 0.37	$\mathbf{-14.36 \pm 0.19}$
HSEvo	$\mathbf{4.77 \pm 0.46}$	-14.07 ± 0.38

objective score and CDI. Here, notice that harmony search can only marginally improve ReEvo. This can be explained that ReEvo does not have any mechanism to promote diversity, therefore it does not benefit from the advantage of harmony search process.

4.4.2 Flash reflection analysis

An additional experiment was conducted in which the reflection component used in ReEvo was replaced with a flash reflection component. Since flash reflection is more cost-effective than the original reflection, reducing the number of tokens utilized for optimization to 150K tokens serves as a method to validate this. Table 4.5 presents experiment results on OP problem. The results show that given a smaller number of the timestep, ReEvo with flash reflection mechanism can outperform HSEvo in optimization performance while obtaining comparable results on CDI. However, when running with a larger number of tokens, ReEvo with flash reflection cannot improve on both objective score and CDI, while HSEvo improves both metrics to 5.67 and -14.62, respectively. This implies that without a diversity-promoting mechanism, flash reflection is not enough to improve the optimization process of LLM-EPS.

The experiments confirm that HSEvo consistently achieves superior objective scores across BPO, TSP, and OP benchmarks while maintaining strong diversity indices. Additionally, the ablation studies highlight the critical roles of harmony search and flash reflection in enhancing HSEvo's performance. These results validate HSEvo's effectiveness and its potential to advance LLM-EPS frameworks in solving complex optimization problems.

To sum up, experimental findings presented in this chapter underscore the significant advantages of the HSEvo in addressing COPs. By systematically benchmarking HSEvo against established LLM-EPS methodologies across three distinct AHD problems. The results reveal HSEvo's ability to consistently achieve superior objective scores while maintaining robust diversity indices. These outcomes validate the hypothesis that diversity-driven optimization strategies, as embodied in HSEvo, are instrumental in advancing AHD and evolutionary processes.

Furthermore, the ablation studies emphasize the pivotal contributions of the harmony search and flash reflection components in enhancing performance of HSEvo. While harmony search bolsters the optimization process by promoting diversity, flash reflection offers a cost-effective yet impactful mechanism for refining solutions. Together, these components elevate HSEvo's capabilities, establishing it as a robust framework for optimizing heuristic design. The results of this chapter lay a solid foundation for future exploration of HSEvo's potential in broader optimization domains, reinforcing its role as a transformative advancement in LLM-EPS.

CONCLUSIONS

This thesis provides the pivotal role of population diversity in enhancing the performance of LLM-EPS for AHD. Two novel diversity measurement metrics, the SWDI and CDI, were introduced to analyze and monitor population diversity quantitatively. A comprehensive review of existing LLM-EPS frameworks reveals a common oversight: the failure to adequately balance population diversity with optimization effectiveness, often leading to suboptimal results.

In response to existing boundaries, the introduction of HSEvo a new state-of-the-art LLM-EPS framework marks a notable advancement in this field. HSEvo leverages a diversity-driven harmony search and genetic algorithm to maintain an optimal balance between exploration and exploitation within heuristic search spaces. By incorporating mechanisms like flash reflection and elitist mutation, HSEvo achieves high diversity indices and competitive objective scores across various optimization problems, such as the BPO, TSP, and OP. Ablation studies also confirmed the effectiveness of HSEvo's components, underscoring its capability to enhance diversity and performance.

This work also establishes a new standard in the domain of LLM-EPS, highlighting the important relationship between diversity and optimization. The results not only fill the voids in current frameworks but also create opportunities for future studies to expand on the suggested metrics and methods. By enhancing the comprehension of heuristic search spaces and promoting automated design techniques. This research offers significant insights and resources for tackling intricate combinatorial optimization problems.

Furthermore, creating effective stopping criteria for the search process is a promising area for research. By integrating diversity metrics and evaluating objective scores. It may be possible to reduce the computational costs inherent in LLM operations, thereby enhancing efficiency without compromising performance.

However, the thesis acknowledges certain limitations, particularly in evaluating LLM-EPS applications to AHD. Currently, experiments are only confined to GPT-4o-mini model due to the high costs associated with utilizing closed-source APIs. Expanding evaluations to incorporate additional models such as Qwen, LLaMA, Gemini, and others could significantly reinforce the robustness of the findings of this thesis. Another area of concern is parameter sensitivity. The harmony search component introduces additional hyperparameters HMCR, PAR, and bandwidth that influence performance. This is a gap that persists across the LLM-EPS, including in prior works like FunSearch, EoH, and ReEvo. Addressing parameter sensitivity represents a promising avenue for future research.

Publication

Parts of the work presented in this thesis have been accepted at the 39th Annual AAAI Conference on Artificial Intelligence (AAAI-25):

1. **Pham Vu Tuan Dat**, Long Doan, and Huynh Thi Thanh Binh, "HSEvo: Elevating Automatic Heuristic Design with Diversity-Driven Harmony Search and Genetic Algorithm Using LLMs", *The 39th Annual AAAI Conference on Artificial Intelligence*, (AAAI-25), 2025. (accepted, **rank A***).

REFERENCE

[1] N. Pillay and R. Qu, *Hyper-heuristics: theory and applications*. Springer, 2018.

[2] E. K. Burke, M. Gendreau, M. Hyde, *et al.*, "Hyper-heuristics: A survey of the state of the art," *Journal of the Operational Research Society*, vol. 64, no. 12, pp. 1695–1724, 2013.

[3] R. Qu, G. Kendall, and N. Pillay, "The general combinatorial optimization problem: Towards automated algorithm design," *IEEE Computational Intelligence Magazine*, vol. 15, no. 2, pp. 14–23, 2020.

[4] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio, "Neural combinatorial optimization with reinforcement learning," *arXiv preprint arXiv:1611.09940*, 2016.

[5] S. Liu, Y. Zhang, K. Tang, and X. Yao, "How good is neural combinatorial optimization? a systematic evaluation on the traveling salesman problem," *IEEE Computational Intelligence Magazine*, vol. 18, no. 3, pp. 14–28, 2023.

[6] D. Drakulic, S. Michel, F. Mai, A. Sors, and J.-M. Andreoli, "Bq-nco: Bisimulation quotienting for efficient neural combinatorial optimization," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[7] M. Nejjar, L. Zacharias, F. Stiehle, and I. Weber, "Llms for science: Usage for code generation and data analysis," *Journal of Software: Evolution and Process*, e2723, 2023.

[8] J. Austin, A. Odena, M. Nye, *et al.*, "Program synthesis with large language models," *arXiv preprint arXiv:2108.07732*, 2021.

[9] K. Mahowald, A. A. Ivanova, I. A. Blank, N. Kanwisher, J. B. Tenenbaum, and E. Fedorenko, "Dissociating language and thought in large language models," *Trends in Cognitive Sciences*, 2024.

[10] S. Liu, C. Chen, X. Qu, K. Tang, and Y.-S. Ong, "Large language models as evolutionary optimizers," in *2024 IEEE Congress on Evolutionary Computation (CEC)*, IEEE, 2024, pp. 1–8.

[11] E. Meyerson, M. J. Nelson, H. Bradley, *et al.*, "Language model crossover: Variation through few-shot prompting," *ACM Transactions on Evolutionary Learning*, vol. 4, no. 4, pp. 1–40, 2024.

[12] A. Chen, D. Dohan, and D. So, "Evoprompting: Language models for code-level neural architecture search," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[13] B. Romera-Paredes, M. Barekatain, A. Novikov, *et al.*, "Mathematical discoveries from program search with large language models," *Nature*, vol. 625, no. 7995, pp. 468–475, 2024.

[14] F. Liu, T. Xialiang, M. Yuan, *et al.*, "Evolution of heuristics: Towards efficient automatic algorithm design using large language model," in *Forty-first International Conference on Machine Learning*, 2024.

[15] H. Ye, J. Wang, Z. Cao, and G. Song, "Reevo: Large language models as hyper-heuristics with reflective evolution," *arXiv preprint arXiv:2402.01145*, 2024.

[16] Y. J. Ma, W. Liang, G. Wang, *et al.*, "Eureka: Human-level reward design via coding large language models," *arXiv preprint arXiv:2310.12931*, 2023.

[17] E. Hemberg, S. Moskal, and U.-M. O'Reilly, "Evolving code with a large language model," *Genetic Programming and Evolvable Machines*, vol. 25, no. 2, p. 21, 2024.

[18] Q. Guo, R. Wang, J. Guo, *et al.*, "Connecting large language models with evolutionary algorithms yields powerful prompt optimizers," *arXiv preprint arXiv:2309.08532*, 2023.

[19] M. Yuksekgonul, F. Bianchi, J. Boen, *et al.*, "Textgrad: Automatic" differentiation" via text," *arXiv preprint arXiv:2406.07496*, 2024.

[20] A Kenneth, *De jong. evolutionary computation. a unified approach*, 2006.

[21] M. Melanie, "An introduction to genetic algorithms," *A Bradford Book The MIT Press. Cambridge, Massachusetts• London, England. Fifth printing.[GS SEARCH]*, 1999.

[22] D. B. Fogel, *Evolutionary computation: toward a new philosophy of machine intelligence*. John Wiley & Sons, 2006.

[23] W. Gao and Z. Yin, "Modern intelligent bionics algorithm and its applications," *Science Pres: Beijing, China*, 2011.

[24] J. Koza, "On the programming of computers by means of natural selection," *Genetic programming*, 1992.

[25] A Vaswani, "Attention is all you need," *Advances in Neural Information Processing Systems*, 2017.

[26] Y. Tay, M. Dehghani, D. Bahri, and D. Metzler, "Efficient transformers: A survey," *ACM Computing Surveys*, vol. 55, no. 6, 1–28, Dec. 2022, ISSN: 1557-7341. DOI: `10.1145/3530811`. [Online]. Available: `http://dx.doi.org/10.1145/3530811`.

[27] R. Zhang, F. Liu, X. Lin, Z. Wang, Z. Lu, and Q. Zhang, "Understanding the importance of evolutionary search in automated heuristic design with large

language models," in *International Conference on Parallel Problem Solving from Nature*, Springer, 2024, pp. 185–202.

[28] F. Liu, X. Tong, M. Yuan, and Q. Zhang, "Algorithm evolution using large language model," *arXiv preprint arXiv:2311.15249*, 2023.

[29] E. J. Solteiro Pires, J. A. Tenreiro Machado, and P. B. de Moura Oliveira, "Diversity study of multi-objective genetic algorithm based on shannon entropy," in *2014 Sixth World Congress on Nature and Biologically Inspired Computing (NaBIC 2014)*, 2014, pp. 17–22. DOI: `10.1109/NaBIC.2014.6921898`.

[30] E. Pires, J. Tenreiro Machado, and P. Moura Oliveira, "Dynamic shannon performance in a multiobjective particle swarm optimization," *Entropy*, vol. 21, p. 827, Aug. 2019. DOI: `10.3390/e21090827`.

[31] L. Wang and Y. Chen, "Diversity based on entropy: A novel evaluation criterion in multi-objective optimization algorithm," *International Journal of Intelligent Systems and Applications*, vol. 4, no. 10, 113–124, Sep. 2012, ISSN: 2074-9058. DOI: `10.5815/ijisa.2012.10.12`. [Online]. Available: `http://dx.doi.org/10.5815/ijisa.2012.10.12`.

[32] K. Nolan and J. Callahan, "Beachcomber biology: The shannon-weiner species diversity index," *Proc. Workshop ABLE*, vol. 27, Jan. 2006.

[33] L. Jost, "Entropy and diversity," *Oikos*, vol. 113, no. 2, pp. 363–375, 2006.

[34] Y. Sheng, G. Shi, and D. A. Ralescu, "Entropy of uncertain random variables wi h application to minimum spanning tree problem," *International journal of uncertainty, fuzziness and knowledge-based systems*, vol. 25, no. 04, pp. 497–514, 2017.

[35] A. Ben-Naim, *Entropy and the second law: interpretation and misss-interpretationsss*. World Scientific Publishing Company, 2012.

[36] C. H. Heip, P. M. Herman, K. Soetaert, *et al.*, "Indices of diversity and evenness," *Oceanis*, vol. 24, no. 4, pp. 61–88, 1998.

[37] C. Voudouris and E. Tsang, "Guided local search and its application to the traveling salesman problem," *European journal of operational research*, vol. 113, no. 2, pp. 469–499, 1999.

[38] H. Ye, J. Wang, Z. Cao, H. Liang, and Y. Li, "Deepaco: Neural-enhanced ant systems for combinatorial optimization," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[39] Y. Wang, H. Le, A. D. Gotmare, N. D. Bui, J. Li, and S. C. Hoi, "Codet5+: Open code large language models for code understanding and generation," *arXiv preprint arXiv:2305.07922*, 2023.

[40] N. Shinn, F. Cassano, A. Gopinath, K. Narasimhan, and S. Yao, “Reflexion: Language agents with verbal reinforcement learning,” *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[41] M. Shanahan, K. McDonell, and L. Reynolds, “Role play with large language models,” *Nature*, vol. 623, no. 7987, pp. 493–498, 2023.

[42] S. S. Seiden, “On the online bin packing problem,” *Journal of the ACM (JACM)*, vol. 49, no. 5, pp. 640–671, 2002.

[43] K. L. Hoffman, M. Padberg, G. Rinaldi, *et al.*, “Traveling salesman problem,” *Encyclopedia of operations research and management science*, vol. 1, pp. 1573–1578, 2013.

[44] S. Martello and P. Toth, “Lower bounds and reduction procedures for the bin packing problem,” *Discrete applied mathematics*, vol. 28, no. 1, pp. 59–70, 1990.

[45] B. Tüű-Szabó, P. Földesi, and L. T. Kóczy, “Analyzing the performance of tsp solver methods,” in *Computational Intelligence and Mathematics for Tackling Complex Problems 2*, Springer, 2022, pp. 65–71.

[46] F. Arnold and K. Sörensen, “Knowledge-guided local search for the vehicle routing problem,” *Computers & Operations Research*, vol. 105, pp. 32–46, 2019.

[47] W. Kool, H. Van Hoof, and M. Welling, “Attention, learn to solve routing problems!” *arXiv preprint arXiv:1803.08475*, 2018.