# Using Large Language models to solve optimization problems

A taxonomy of methods with evaluation on traveling salesman and vehicle routing problems.

Master's thesis in Computer science and engineering

Oliver Olsson

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2024

Master's thesis 2024

# Using Large Language models to solve optimization problems

A taxonomy of methods with evaluation on traveling salesman and vehicle routing problems.

Oliver Olsson

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2024

Using Large Language models to solve optimization problems
A taxonomy of methods with evaluation on traveling salesman and vehicle routing problems.
Oliver Olsson

© Oliver Olsson, 2024.

Supervisor: Morteza Haghir Chehreghani, Computer Science and Engineering
Advisor: Leo Laine, Volvo GTT, Adjunct Professor at Mechanics and Maritime Sciences.
Examiner: Peter Damaschke, Computer Science and Engineering

Master's Thesis 2024
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2024

Using Large Language models to solve optimization problems
A taxonomy of methods with evaluation on traveling salesman and vehicle routing problems.
Oliver Olsson
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

# Abstract

This thesis explores the use of Large Language models (LLMs) to solve optimization problems. This is achieved by defining, developing and evaluating three method which utilize the LLM in different ways, namely, as algorithm implementer, optimizer and hyperparameter optimizer. Evaluation is mainly performed on the traveling salesman problem and capacitated vehicle routing problems. Performance, limitations and challenges are discussed for the three methods. Results shows that LLM is able to effectively implement simple algorithm and heuristics, but not algorithms that achieve state-of-the-art performance. LLM is also shown to be an effective optimizer for small scale optimization problems but lacks the ability to effectively perform optimization for problems of higher complexity. However, the performance is able to be improved by the use of self reflection and problem reformulations. LLM used as hyperparameter optimizer achieves the highest performance, although relies on preexisting high performance algorithms or heuristics. LLMs are also shown to outperform other simple hyperparameter optimization methods such as grid search and random search.

Keywords: Computer, science, computer science, engineering, thesis, large language models, LLMs, optimization, fleet optimization, transport mission planning.

# Acknowledgements

I would like to thank my academic supervisor Morteza Haghir Chehreghani, my industrial supervisor Leo Laine and my examiner Peter Damaschke for feedback, guidance and support in writing this thesis. I would also like to thank Michelle Kim and Dhasarathy Parthasarathy for valuable discussions during the time of the project.

Oliver Olsson, Gothenburg, 2024-06-19

# Contents

# List of Figures

# List of Tables

# 1 Introduction

For most of language modeling history, language models were intended to perform simple natural language processing (NLP) tasks, such as information retrieval, text classification and sentiment analysis [1]. Until recently, this is all language models were believed to be able to achieve. However, this would change due to several key findings, most notably the transformer architecture [2] and pre-training of models on a large text corpus, and later fine-tuning according to downstream tasks [1]. The scaling of these models, originally called pre-trained language models (PLMs) lead to models that are now called Large language models (LLMs). The scaling of model parameter size and pre-training corpus lead to significant improvements and emergent capabilities, such as the ability to solve tasks through in-context learning [3] [4]. Unlike past language models, LLMs could be used as general-purpose task solvers, able to solve real-world tasks.

Since the release of the popular LLM ChatGPT, the number of arXiv papers containing the keyword ”large language model” published each day has increased from 0.4 to 8.58 papers per day [1]. Today, LLMs are used for many complex NLP tasks such as summarizing, translation, text generation and code generation but can also be used to solve tasks involving arithmetic, commonsense, logical and symbolic reasoning [5]. However, LLMs are still not able to solve most optimization problems, especially ones of high complexity. Solving complex optimization problems requires optimization, which is the process of finding the best feasible solution to a given problem. For many optimization problems, such as NP-complete problems, this process can be both complicated and computationally heavy due to the search space.

This thesis aims to explore methods that allow LLMs to solve optimization problems, which are not possible to solve through direct prompting. The methods include comprehensive frameworks that allow the LLM to perform optimization and iteratively produce better solutions. However, they can differ drastically depending on the specific use of tools, internal control flows and solution generation strategies. For this reason, three methods are defined which serve as categories for the many variations that can be implemented. These are (1) LLM as algorithm implementer, (2) LLM as optimizer and (3) LLM as hyperparameter optimizer. LLM as algorithm implementer produces algorithms, iterative methods or heuristics in code that is subsequently executed by a code interpreter. LLM as the optimizer involves using the LLM in a prompting framework that iteratively produce better solutions by feeding back past solutions. LLM as hyperparameter optimizer, as the name suggests, uses the LLM to produce hyperparameter configurations for an already existing algorithm or method that solves the optimization problem. The methods are evaluated on two case study problems. The first case study problem is the traveling salesman problem (TSP), a popular combinatoric optimization problem, and the second is a variation of a vehicle routing problem (VRP) called capacitated VRP (CVRP). TSP was chosen since it serves as a standardized benchmark problem for optimization problems and CVRP in order to highlight the potential limitations that comes from higher complexity problems, typically found in real world applications.

## 1.1 Prior work

The main inspiration of this thesis is a paper by Google DeepMind called "Using Large language models as optimizers" [6]. This paper introduces the idea of using LLMs to solve optimization problems by letting the LLM act as a gradient-free optimizer in a framework named OPRO (Optimization by PROmpting). OPRO is shown to outperform simple human built algorithms for small scale instances of linear regression and traveling salesman [6]. OPRO is used as a starting point and serves as a proof of concept of the idea that LLMs could be used to solve optimization problems. Other articles also support this idea, most notably, Mirchandani et. al. [7] which shows that LLMs can serve as general sequence modelers by autoregressively completing complex token sequences.

In addition to OPRO, there was also a recent study using LLMs for hyperparameter optimization (HPO) [8]. The study found that LLMs could perform comparably or better than traditional HPO methods such as random search and Bayesian optimization on standard benchmarks. Another notable study used LLMs to implement simple algorithms and evolved them using an evolutionary algorithm. This method was shown to outperform simple hand-crafted and LLM-generated heuristics [9]. To the best of found knowledge, there are no survey or taxonomy articles discussing the topic of using LLMs to solve optimization problems.

This thesis aims to categorize the different methods that exist which allow LLM to solve optimization problems. It also aims to compare these methods using case study problems in order to highlight their respective performance, limitations and challenges. To achieve this, some experiments found in the literature are reproduced, specifically applying the LLM as optimizer to solve TSP and using LLM as hyperparameter optimizer to solve benchmark test functions. The methods in these experiments are then built on further and expended to new case evaluations. Finally this thesis also aims to investigate if the use of techniques such as reflection can improve the optimization of problems with high complexity.

## 1.2 Motivation and applicability

The utilization of using LLMs to solve optimization problems is of interest for several reasons. Firstly, it would not require any advanced knowledge in mathematical modeling or algorithm development to use, allowing anyone to be able to solve optimization problems on their own using an LLM. Secondly, it could reduce the time and cost of the development of an algorithm to solve the given optimization problem. Thirdly, LLMs as optimizers could be used to solve derivative-free optimization problems where the derivative of the objective function is unavailable or impractical to obtain. Finally, LLMs also have the innate ability to incorporate natural language as variables in the problem instance, which is especially useful for optimization problems involving language, such as prompt optimization.

It is also crucial to address the limitations of using LLMs to solve optimization problems. LLMs are next token predictors, which means they try to find the most likely continuation of the provided sequence using a probabilistic decoding strategy. This makes LLMs inherently stochastic, which means optimal solutions can never be guaranteed. However, this is also true for any stochastic algorithm. Additionally, the objective function is always considered to be given, thus the feasibility of any solution provided by LLM can always be ensured by checking the solution with the objective function (assuming that the objective function is deterministic). LLMs are also not currently capable of outperforming state-of-the-art optimization methods, however, may be employed in specific use cases where adaptability is preferred over performance. Nevertheless, it is likely that LLMs will continue improving in the future, allowing them to solve optimization problems with higher accuracy.

## 1.3 Scope

The comparison between the methods is of importance and thus the LLM models are kept constant in all evaluations. For this reason, task-specific fine-tuning and training are considered out of the scope of this thesis. However, for real-world applications, these techniques should be considered in order to increase performance. Evaluation of fine-tuned models would also serve as an interesting direction for future work.

Additionally, problem specific changes to the methodology is not considered, outside of the objective function and the input prompts. This is because the methodology should be general and applicable to any optimization problem, without changes. The main benefits, as mention in Section 1.2, is that LLMs should be both time efficient and simple to implement, which would not be the case if the methodology would have to be changed for every optimization problem. Nevertheless, implementing problem specific changes such as for instance, dividing the problem into sub-problems, could lead to better performance and should be considered if optimal performance is desired.

## 1.4 Research questions

As previously mentioned, the aim is to present an overview of different methods for using LLMs to solve optimization problems evaluated on the traveling salesman problem and capacitated vehicle routing problems. The results of this evaluation should answer the following four research questions.

- How do the methods perform in comparison to standard benchmarking algorithms?
- What are the main limitations of each method and how can they be addressed?
- How does increased problem complexity affect the performance of the method?
- Which method is best suited for a given situation, such as for large/small problem scales and high/low complexity tasks?

# 2

# Theory

This chapter will discuss the theory regarding large language models, optimization and complexity that is relevant for this thesis. For LLMs the focus is on tokenization and how LLMs predict the next token. The section on optimization introduces relevant terminology and classical ways of solving optimization problems. It also introduces the two optimization problems in which the methodology is applied to. Finally complexity is discussed since it affects the LLMs ability to solve the given optimization problem.

## 2.1 Large Language models

Large language models (LLMs) are artificial neural networks trained on large corpora of text in order to learn the statistical relationships within text. From these statistical relationships, LLMs are able to predict the next token or word of a sequence of text. This section will focus on how the underlying model learn the relationships between tokens and how the model generates an output, i.e the next token.

Machine learning algorithms process numbers, thus the text must be converted from letters and words into numbers. This is done by first creating a vocabulary containing all unique tokens of the corpus. Tokens can be either words, letters or some arbitrary choice of sub-words, thus there are infinitely many ways of dividing a corpus into tokens, a process called tokenization. All tokenizations are however not equal. Letter or symbol tokenization, yields a small vocabulary, since there are not that many letters and symbols. A small vocabulary is desirable for efficiency, however, letters and symbols fail to capture semantic meanings, often resulting in worse performance. In contrast, words do have a much clearer semantic meaning than letters, but tokenizing the corpus into words usually leads to a very large vocabulary. Additionally, the most rare words will be infrequent in the corpus leading to worse performance when these words are involved. Sub-word tokenization is therefore a good trade-off between keeping the semantic meaning of tokens relatively intact while making the vocabulary as small as possible while doing so [10]. The overwhelming majority of LLMs use some sort of sub-word tokenization [1]. All tokens of the vocabulary are then indexed to a unique integer representation. Figure 2.1 shows tokenization of a sentence on a character, word and subword level.

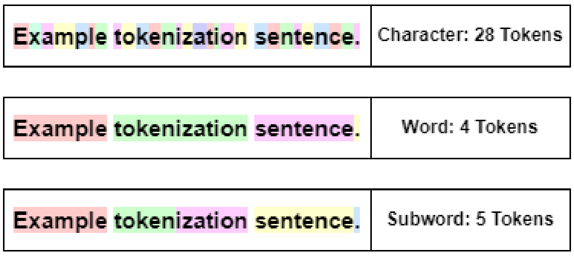



Figure 2.1: Character, word and subword tokenization of an example sentence resulting in different total amount of tokens.

The LLM is however not trained on integer representations of tokens, but instead on high-dimensional vectors called word embeddings. These vectors have the property that similar words have similar vector representations, i.e one can compute a similarity between words (or tokens). This property is essential when training language models, since it gives a quantitative measurement of how close a generated output was to the desired output. Every token of the vocabulary has a word embedding associated with it and this embedding is learned during the training phase. A typical dimension of word embeddings for state-of-the-art models, i.e the size of the vector representation ranges from 50 to 300 elements, where each element is a floating point [11]. Tokenization is a fundamental part of LLMs and will be discussed later with regard to techniques used to achieve higher output accuracy.

Once a corpus of text has been tokenized, the LLM can be trained to perform next token prediction given an input sequence, also known as input prompt. The model architecture of LLMs are primarily decoder-only transformers, however other architectures such as Mambas, a recent state space model architecture, can also be used [12]. This thesis exclusively considers decoder-only transformers as they are widely employed in state-of-the-art models and consistently demonstrate superior performance across various task solving benchmarks [13]. Transformers are able to learn statistical and context specific correlations in sequential data through the use of a mechanism called self-attention. The self-attention mechanism works by calculating ”soft” weights for each token in the input sequence. These soft weight can be computed in parallel and are dynamically changed at runtime. This allows the model to understand context specific correlations between tokens in the input sequence, which allows for better prediction capabilities. A decoder-only transformer uses a special attention mechanism called masked self-attention. The masked self-attention is computed for every token in the input sequences, and for a given token, only tokens to the left are allowed to be compared with and used in the attention computations. The correlations of tokens appearing after our token of interest are masked, since these are the tokens we want the model to generate, without knowing them in advance. This is explained in Figure 2.2.

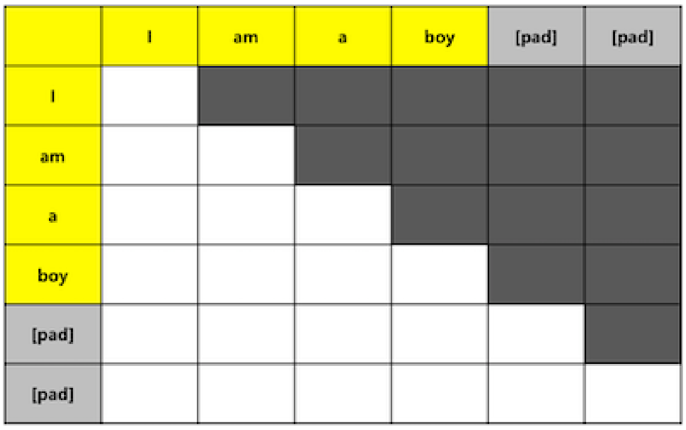



Figure 2.2: Self-attention mask for the sequence of tokens ”I am a boy”. Columns show each token for which the attention weights are computed, and the rows indicate if the row token is part of the attention computation or not. Only tokens appearing before the current token are used. For example the word ”am’ only has the attention weight of ”I” and itself (”am”).

After training, the LLM is able take an input sequence of tokens and give the probability that a certain token is the next token, for every token in the vocabulary. One way of selecting the next token is to simply pick the token with the highest probability of being the next token. This decoding strategy is called greedy decoding, and it is deterministic. Alternatively one could also randomly sample the probability distribution and choose tokens depending on their individual probability. This introduces an element of randomness which can be desired, especially when it comes to solving tasks, since the most probable next token might not always lead to the optimal answer. The level of randomness is determined by a chosen hyperparameter called temperature. After the first token has been selected according to some decoding strategy such as greedy or random sampling, it is appended to the input sequence and the process can be repeated until the LLM generates an end of sequence token, representing the end of the response.

## 2.2 Optimization problems as a task

An optimization problem is the task of finding the best solution from the set of all feasible solutions with respect to a goal or objective function. Optimization is generally divided into two categories, continuous or discrete, depending on the problem variables. Formally, an optimization problem can be defined as follows:

**Definition 1.** *Given: a function* $f : A \to \mathbb{R}$ *from some set* $A$ *to the real numbers*

*Sought: an element* $x_0 \in A$ *such that* $f(x_0) \leq f(x)$ *for all* $x \in A$ *(”minimization”) or such that* $f(x_0) \geq f(x)$ *for all* $x \in A$ *(”maximization”).*

where $f$ is the objective (or goal) function. $A$ is typically called the choice set or search space and is determined by the constraints of the problem. Elements of $A$

are called feasible solutions, and a feasible solution that minimizes (or maximizes) the objective function is by definition, the optimal solution.

There are essentially three main categories of solving optimization problems. These are algorithms, iterative methods and heuristics. Algorithms are defined as a finite sequence of rigorous instructions, in order to perform a specific computation or to solve a problem. Algorithms that are not guaranteed to produce optimal solutions are commonly refereed to as approximation algorithms, and an example of such an algorithm is nearest neighbour for solving TSP. Iterative methods are methods that converge on a solution by iterative evaluation of Hessians, gradients or function values. These methods tends to work well when, for example, gradients can be easily computed and varies sufficiently smoothly. An example of an iterative method is gradient descent, commonly used in back-propagation when training neural networks. Unfortunately, many optimization problems have function landscapes that are too "bumpy" or gradients that are too computationally heavy for iterative methods to be utilized. Finally there are heuristics which are similar to algorithms, however they are not always guaranteed to terminate and are typically aimed at providing approximate solutions that are useful in practical purposes. Examples of heuristic are genetic algorithms, particle swarm optimization, simulated annealing and ant colony optimization.

Clearly there are many different ways of attempting to solve optimization problems, and since they are able to be described in natural language, they can be described in a prompt and given to an LLM tasked with solving the problem.

### 2.2.1 Traveling salesman problem

The traveling salesman problem (also known as traveling salesperson problem or TSP) is a classical combinatorical optimization problem with many known algorithms that provide exact solutions as well as heuristic that provide approximate solutions [14]. The problem is highly studied in the literature and commonly used as a benchmark for optimization methods. The problem can be formulated in slightly different ways, one of which being the following, which Figure 2.3a also depicts an instance of.

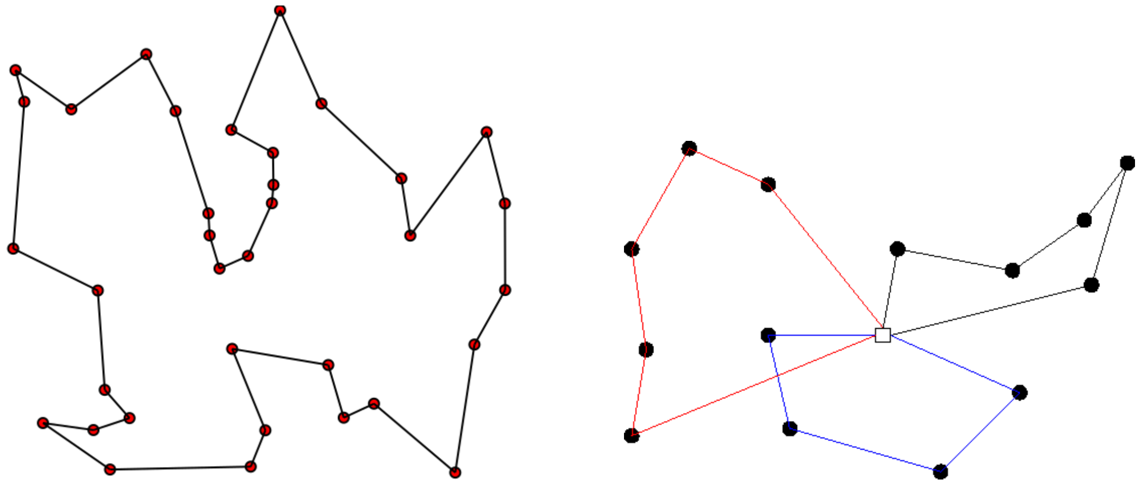
**Problem 1.**

*Given: A set of nodes with their coordinates in the plane.*

*Sought: The shortest route that traverses all nodes from the starting node and finally returns to the starting node.*

### 2.2.2 Vehicle routing problem

The vehicle routing problem (VRP) is a generalization of TSP, first formulated by George Dantzig and John Ramser in 1959 [15]. The problem can be formulated as follows.



(a) TSP optimal tour (b) VRP with 3 vehicles

Figure 2.3: Example solutions of instances of (a) TSP and (b) VRP.

**Problem 2.**

*Given: A set of nodes with coordinates in the plane, a fleet of vehicles and a set of customers with delivery demands.*

*Sought: An optimal set of routes for the fleet to traverse that satisfies all demands.*

In this problem, optimal refers to the minimization (or maximization) of the total route cost, where the cost could be a function of many variables such as the driver, fuel, energy, delivery, wear and tear to name a few. VRP has many variants that have different considerations such as VRP with pick up and delivery, VRP with time windows, VRP with carrying capacity constraints and many more.

This thesis focuses on a VRP with vehicle carrying constraint (CVRP), i.e every vehicle has a maximum load. When every vehicle can have a unique carrying capacity the problem is called mixed fleet CVRP or heterogeneous fleet CVRP. It also assumes that all pickups of goods are made at a singular node, commonly called the depot. Any given route in the set of routes that solve the problem must always start and end at the depot. Further details are described in the method section and an example problem solution is shown in Figure 2.3b.

## 2.3 Complexity

The word complexity has many definitions depending on its context, and the definition may sometimes be ambiguous. For instance, in every-day situations, the term can simply be used to describe something that is difficult to answer or understand. For this thesis there are two categories of complexity that are of interest, system complexity and computational complexity.

System complexity, refers to a system or model with many interacting components or parts. The interacting components of the system is sometimes also refereed to as subunits. These systems, called complex systems, exhibit what are called emergent properties. Emergent properties, are properties of the system that cannot be reduced to any properties of the subunits and their interactions [16]. Emergent properties are sometimes refereed to as properties that are greater than the sum of its parts. Examples of complex systems are Earth's global climate, the human brain, living cells, transportation systems and, relevant to this thesis, LLMs [17] [18].

The second category of complexity is computational complexity. Computational complexity considers the resource usage required to solve a problem, where the resource often being the time or space needed for an algorithm to solve the problem. In computational complexity theory, problems are classified according to this resource usage, and these classes, called complexity classes, are used to relate and compare problems [19]. Computational complexity theory is a deep topic that considers many different complexity classes and their respective relationships, however, this thesis is only concerned with the classes P and NP. P stands for polynomial time and the class contains all decision problems [1] that can be solved by a deterministic Turing machine using a polynomial amount of computation time. What this mean in practice, is that problems in P are generally efficiently solvable by algorithms. NP stands for non-deterministic polynomial time, and contains all decision problems solvable in polynomial time by a non-deterministic Turing machine. What this means in practice is that provided solutions where the answer is ”yes”, can be verified in polynomial time. Problems in NP do not necessarily have polynomial time algorithms. P is a subset of NP, thus all problems in P are also in NP, see Figure 2.4. It is widely believed by researchers that there are problems in NP that are not in P ($N \neq NP$) [20], the hardest of which are called NP-complete. Any algorithm that would solve an NP-complete problem in polynomial time would also solve every problem in NP through reduction, and prove that $N = NP$, this is however, as previously mentioned, not believed to be the case. Instead NP-complete problems are believed to be unsolvable by polynomial time algorithms.

[1] Every computational problem can be turned into a decision problem, by changing the problem into a ”yes or no” answerable question, i.e a decision problem [19].

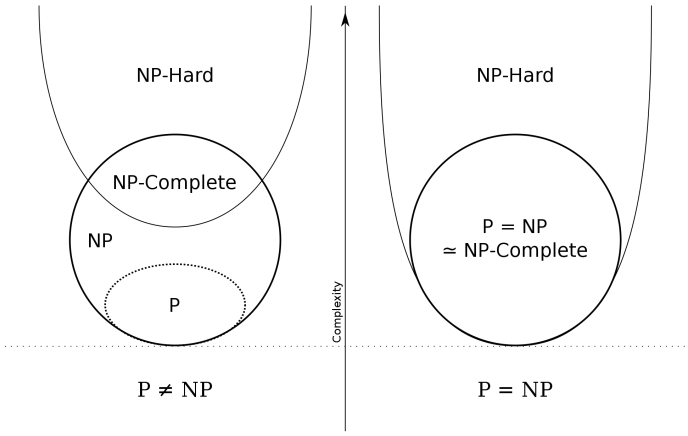



Figure 2.4: Complexity classes P, NP, NP-complete and NP-hard for the two cases $N \neq NP$ and $N = NP$. $N \neq NP$ (Left) is widely believed to be correct, indicating that not all problem in NP have polynomial time algorithms, especially NP-complete problems.

The two problems considered in this thesis, the traveling salesman (TSP) and the vehicle routing problem (VRP), [2] both have a decision formulation that is NP-complete. This means that there are no polynomial-time algorithms that solve these problems [3]. Nevertheless, this does not eliminate the use of approximation algorithms, iterative methods or heuristics to produce approximate solutions, which are generally satisfactory for practical applications. Although, both TSP and VRP are in the same complexity class they are not equally complex. VRP is significantly more complex than TSP, due to several reasons such as the additional constraints of charging/fueling, capacity, time frames, and type of vehicle. VRP also allows revisiting nodes any number of times which drastically increases the set of feasible solutions. In fact, VRP is a generalization of TSP. These additional complexities can be understood as system complexities since they introduce more components that are able to interact with each others, which by definition makes the problem more complex.

[2]Technically we use CVRP, however it is a subclass of VRP so it also holds true

[3]The non-existence of polynomial-time algorithms for problems in NP assumes that $N \neq NP$ [20].

# 3

# Methods

The methodology is presented in two parts. Firstly, by describing the experimental setup used to generate the results and secondly, by defining the methods along with their respective case evaluations.

## 3.1 Experimental setup

This section outlines the experimental setup used to generate the results. Specifically it describes which LLMs where used and how TSP and CVRP were formulated and implemented. Finally the evaluation metrics are described.

### 3.1.1 Models

The LLMs used in this thesis to generate the results were Mixtral 8x7b (56b) and LLAMA3-70b. They are both state-of-the-art open source LLMs with notable performance in comparison to models of equal size as of May 2024. Mixtral 8x7b is a mixture of experts model containing eight different seven billion parameters models which are each specialized to solving a specific task. The architecture then delegates the generation of tokens to different models depending on a weighting function, which is learned in the training process. LLAMA3-70b was chosen since it was the best performing open-source model on math, reasoning and task solving benchmarks as of May 2024.

### 3.1.2 Traveling salesman problem

TSP was defined as a set of $n$ nodes with coordinates in the x, y plane. Coordinates were randomly sampled over a square grid ranging from -100 to 100, as integers. All nodes were considered to be connected to all other nodes, i.e a complete graph. The starting node of the tour was not provided, thus the model could choose any node as starting point.

A solution to TSP is a route that visit all nodes and finally returns to the starting node, with minimal total Euclidean distance. Solutions were thus represented as a list of nodes, and from this list and coordinates, the total route length could be computed. Infeasible solutions provided by the LLM were disregarded. These could for instance be paths that fail to visit all nodes exactly once or visit non existing nodes, which can occur from hallucinations in the LLM. The prompts used to generate the results can be found in Appendix B.

### 3.1.3 Heterogeneous fleet CVRP

The hetrogeneous fleet capacitated vehicle routing problem (CVRP) was implemented similarly to that of TSP. Nodes were sampled on a grid ranging from -100 to 100, as integers. Node 0 was chosen to be the depot, i.e the starting point of all routes. Like TSP, the graph of nodes was chosen to be fully connected. For simplicity all evaluation was performed for a fleet of two vehicles. These vehicles were two trucks with carrying capacity 20 and 12 pallets respectively. These values were chosen to represent typical carrying capacities for 40 feet and 20 feet trucks. Evaluation was performed for three test cases which are shown in Figure 3.1, where one customer is located at every node with a specific delivery demand. The objective function was defined in two ways, presented in the following subsections. The first definition minimizes the total cost while fulfilling all deliveries, while the second definition maximizes vehicle revenue while not requiring every delivery to be made. The purpose of the second definition is to relax the constraint that the LLM has to provide a solution that fulfills all demands, which proved to be difficult. The prompts used to generate the results can be found in Appendix B.

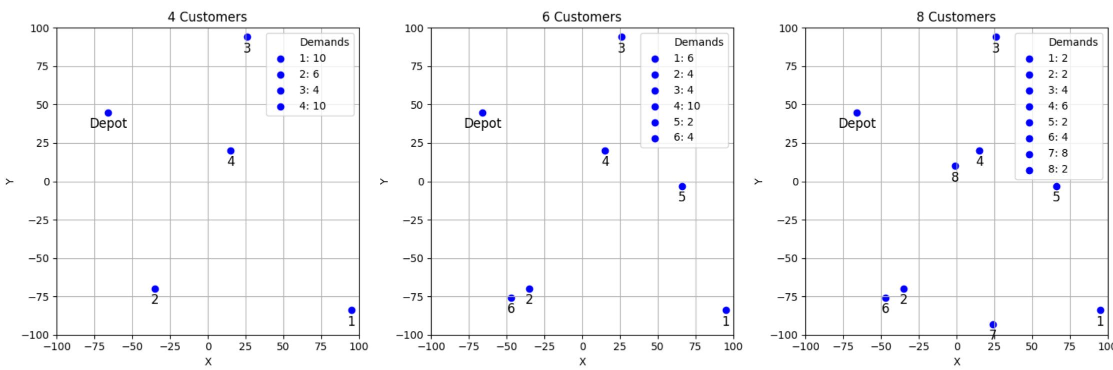



Figure 3.1: CVRP test cases for different number of customers ($n$ = 4, 6 and 8). The demands of the customers are number of pallets and is shown in the legend.

### Total cost minimization with vehicle flow constraint

$$\min \quad \sum_{i=0}^{n+1}\sum_{j=0}^{n+1} c_{ij}x_{ij} \tag{3.1}$$

$$\text{s.t.} \quad \sum_{\substack{j=1 \\ j\neq i}}^{n+1} x_{ij} = 1 \quad i = 1, ..., n \tag{3.2}$$

$$\sum_{\substack{i=0 \\ i\neq h}}^{n} x_{ih} - \sum_{\substack{j=1 \\ j\neq h}}^{n+1} x_{hj} = 0 \quad h = 1, ..., n \tag{3.3}$$

$$\sum_{j=1}^{n} x_{0j} \leq K \tag{3.4}$$

$$y_j \geq y_i + q_j x_{ij} - Q_v(1 - x_{ij}) \quad i, j = 0, ..., n+1 \tag{3.5}$$

$$q_i \leq y_i \leq Q_v \quad i = 0, ..., n+1 \tag{3.6}$$

$$x_{ij} \in \{0, 1\} \quad i, j = 0, ..., n+1 \tag{3.7}$$

This objective minimizes the total cost of traveling while fulfilling all deliveries. $c_{ij}$ is the cost of traveling from node $i$ to node $j$ and $x_{ij}$ is a decision variable that indicates if any route traverses directly from $i$ to $j$. Constraint 3.2 enforces that each customer is visited exactly once. Constraint 3.3 makes sure that any route that visits a node $h$ must also subsequently leave $h$. 3.4 limits the number of outgoing nodes from the depot to $K$ where $K$ is the number of vehicles. Constraint 3.5 and 3.6 together enforce that the carrying capacity is never violated where $Q_v$ is the maximum capacity of vehicle $v$. $q_i$ is the pallet demand of node $i$ and $y_i$ is the cumulative demand for the vehicle visiting node $i$ (including the demand at node $i$). Constraint 3.7 simply states that x is a decision variable which only takes on the values 0 or 1.

### Vehicle revenue maximization with unconstrained flow

$$\max \quad \sum_{i=0}^{n+1}\sum_{j=0}^{n+1} rq_j x_{ij} - c_{ij}x_{ij} \tag{3.8}$$

$$\text{s.t.} \quad \sum_{\substack{j=1 \\ j\neq i}}^{n+1} x_{ij} \leq 1 \quad i = 1, ..., n \tag{3.9}$$

$$\sum_{\substack{i=0 \\ i\neq h}}^{n} x_{ih} - \sum_{\substack{j=1 \\ j\neq h}}^{n+1} x_{hj} = 0 \quad h = 1, ..., n \tag{3.3}$$

$$\sum_{j=1}^{n} x_{0j} \leq K \tag{3.4}$$

$$y_j \geq y_i + q_j x_{ij} - Q_v(1 - x_{ij}) \quad i, j = 0, ..., n+1 \tag{3.5}$$

$$q_i \leq y_i \leq Q_v \quad i = 0, ..., n+1 \tag{3.6}$$

$$x_{ij} \in \{0, 1\} \quad i, j = 0, ..., n+1 \tag{3.7}$$

This objective instead maximizes the vehicle revenue for every vehicle in the fleet. The revenue of every delivered pallet was assumed to be constant and denoted as $r$. In this formulation the objective is simply changed to a maximization. The first term in Equation 3.8 is the raw revenue while the second term in the cost of traveling. The only constraint that is changed in this formulation is Constraint 3.9 where the equality sign is changed to an inequality, relaxing the requirement that all customers have to be visited. Instead, each customer can be visited at most one time. The remaining constraints and variables are explained in the previous section.

### 3.1.4 Evaluation metrics

To evaluate the methods, two main metrics were used, optimality gap and feasibility rate. Optimality gap (OG) is a metric that measures the accuracy of a solution with regards to the optimal solution. The optimality gap is calculated as follows.

$$\mathrm{OG} = |\frac{S - S^*}{S^*}|, \tag{3.10}$$

where $S$ is the score of the generated solution and $S^*$ is the score of the optimal solution. Optimality gap is dimensionless and typically given in percent. Feasibility rate is also given as a percentage and represents the fraction of solutions that do not violate any constraints. Infeasible solutions always violate at least one constraint, which means their score cannot be computed by the objective function.

## 3.2 Description of methods

From the starting point of an optimization problem $P$ with a given objective function $F$, how can LLMs be employed to find an optimal solution $S$, or an approximate optimal solution $S^*$? This is the main question of the thesis, and it only makes the assumption that the optimization problem $P$ and objective function $F$ is known, which is often the case, especially for problems in NP (See Section 2.3). An approximate optimal solution is a solution that is considered sufficiently close to the optimal solution, and could be the result of an approximation algorithm, iterative method or heuristic. This section lays out three methods which can be seen in Figure 3.2. It also gives the detailed implementation of these methods that was used to generate the results.

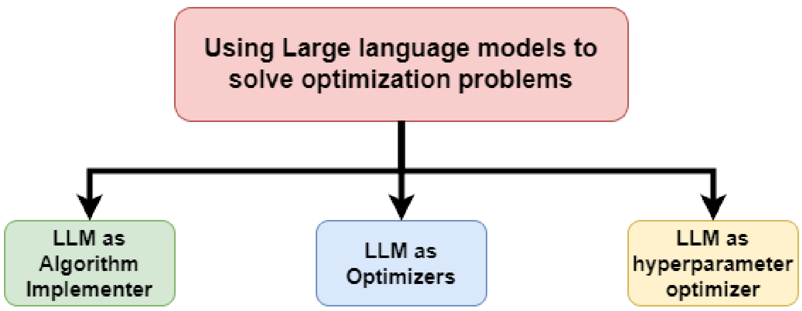



Figure 3.2: Overview of method to solve optimization problems using LLMs.

### 3.2.1 LLM as algorithm implementer

One way of using LLMs to solve optimization problems is to task the LLM with the implementation of algorithms in executable code and subsequently run the code in an external interpreter. Additionally, one can also implement a framework connecting the LLM and the code interpreter in order to correct or improve parts of the code. Such implementations are shown in Figure 3.3. In all variations of this method, the end results is a step-by-step instruction of how to solve the problem. This does not necessarily have to be an algorithm, it could also be an iterative method or a heuristic and the step-by-step instructions could be executable code, pseudo-code or natural language.

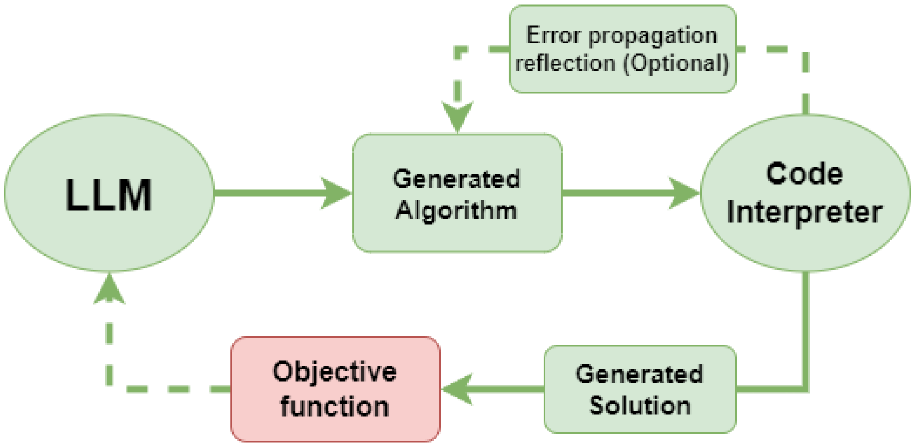



Figure 3.3: General framework of using LLM as algorithm implementer. The LLM generates an algorithm that is executed by a code interpreter and solutions are evaluated by the objective function. Error propagation and feedback of solutions to the LLM are optional variations and therefore have dashed lines.

Evaluation of this method was performed for TSP and CVRP, for problem instances of 5, 10, 20, 50 and 100 nodes. Two prompting strategies were explored, one in which no requirement was placed on the choice of algorithm and the other requesting the most efficient algorithm to solve the problem. The choice of algorithm directly effects the performance, which is why no requirement was placed on which algorithm could and could not be implemented. This was also done to allow the LLM to base the choice of algorithm on the problem instance. For example, problem instances with a small number of nodes could be solved using a brute force method while problems in the range of 100 nodes would require approximation algorithms. The framework that was implemented is shown in Figure 3.3 with the exclusion of the error propagation reflection, since this was out of the scope of this thesis.

### 3.2.2 LLM as optimizer

In this method, the LLM is used as the optimizer, thus the LLM is tasked with producing new solutions in the iteration process. This is performed in a framework, originally called OPRO (Optimization by PROmpting) and developed by Google

DeepMind [6]. The framework is shown in Figure 3.4 which shows the iteration loop including a meta-prompt, LLM as optimizer and an objective function evaluator. In addition to these three components, a reflection mechanism is also added which can be used to correct false solutions provided by the LLM.

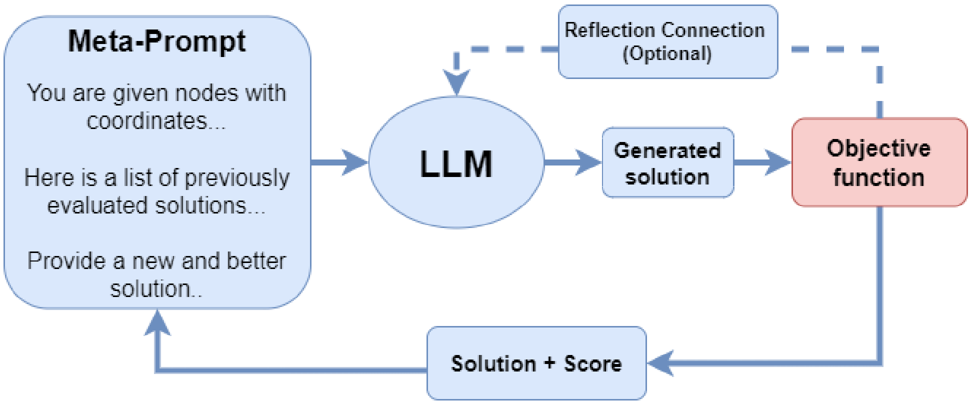



Figure 3.4

The framework was implemented as shown in Figure 3.4 both with and without the reflection connection. Additionally several design parameter choices were explored. These parameter choices are shown in Table 3.1 along with the variations explored and the final values that were used to generate the results. Evaluation of optimality gap and feasibility was performed on TSP and both formulations of CVRP outlined in Section 3.1.3.

Table 3.1: Design parameters of using LLM as optimizer. History refers to the solution-score history in the meta-prompt and solutions per iteration refers to the number of LLM calls in every iteration for the first 10 iterations. This is done to ensure more stable convergence.

| Parameter | Explored variations | Used in results |
|---|---|---|
| LLM Temperature | 0, 0.5, 1 | 1 |
| Decoding strategy | Greedy decoding | Greedy decoding |
| History ordering | Sorted, Random | Random |
| History length | 5, 10, 20 | 10 |
| Max iterations | 50, 100, 200 | 100 |
| Solutions per iteration (Early steps) | 1, 10, 20 | 10 |

### 3.2.3 LLM as hyperparameter optimizer

In this method the LLM was tasked with providing hyperparameter configurations for an external optimization algorithm that in turn performs the optimization of the objective function. This is shown in Figure 3.5.

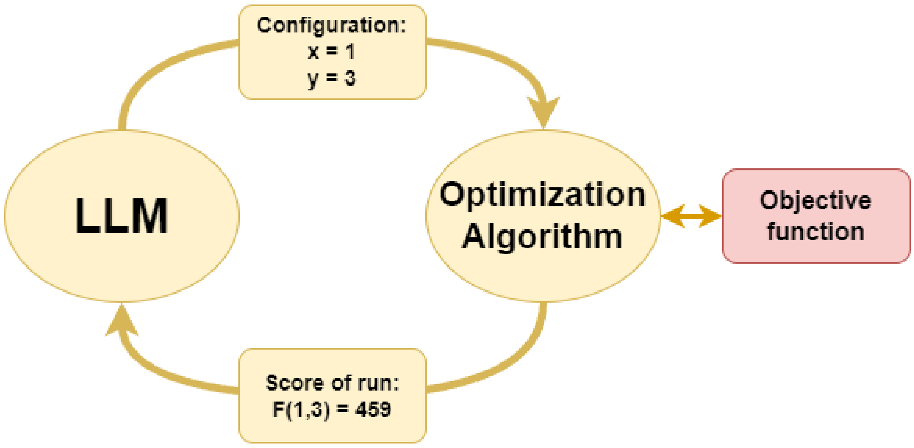
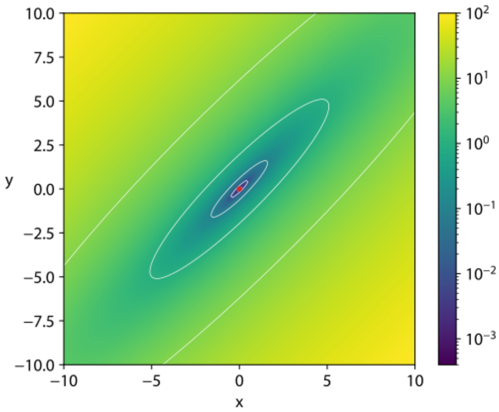



Figure 3.5: Simple schematic of LLM being used to provide hyperparameter configurations to an optimization algorithm in a feedback loop. The optimization algorithm uses the objective function to evaluate solution candidates.

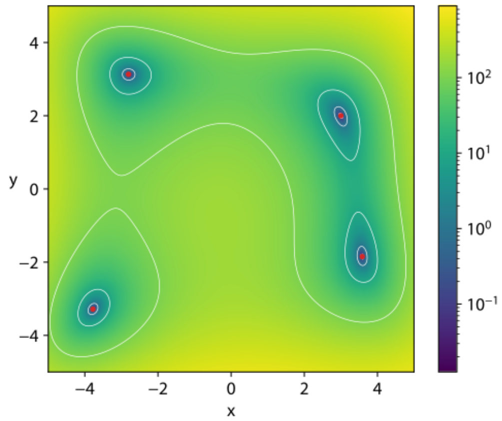
This method could be viewed as a special case of using the LLM as optimizer, where the optimization problem is finding the optimal hyperparameters. Hyperparameters is typically set by the user and they are not learned by the optimization algorithm or model. The optimal values that the hyperparameters take on are also dependent on the problem instance, making it difficult to predict them without experimentation. Two simple, yet common hyperparameter optimization methods are grid search and random search [21]. Grid search creates a grid of possible parameter configurations covering the search space and evaluates them all. Random search, as the name suggests, randomly chooses configurations and evaluates them.

To evaluate the method of using LLM as hyperparameter optimizer it was compared to grid search and random search in two test cases. The first test case was finding the inputs that minimizes a function of two variables. This is a common way of benchmarking hyperparameter optimization methods where the two input variables serve as hyperparameters. The second test case was finding the hyperparameters to an ant colony optimization (ACO) algorithm to solve the traveling salesman problem, i.e to minimize the total distance of the tour.

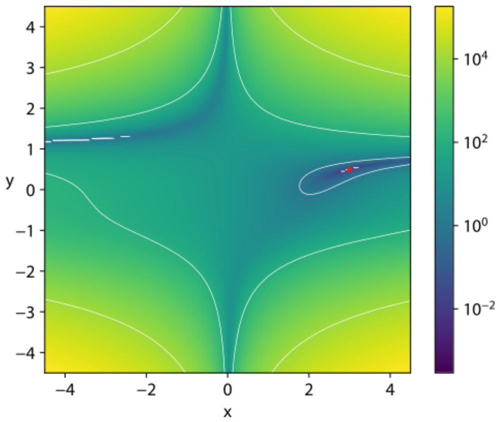
For the first test case, four different two-variable-functions were used in the evaluation. These functions were all common benchmarking functions with relatively smooth function space, see Figure 3.6. All functions have at least one minimum function value of zero within their respective search space. The search budget, i.e the number of function evaluations was set to 10 for the LLM, grid and random search. All prompts used to generate the results is shown in Appendix B.



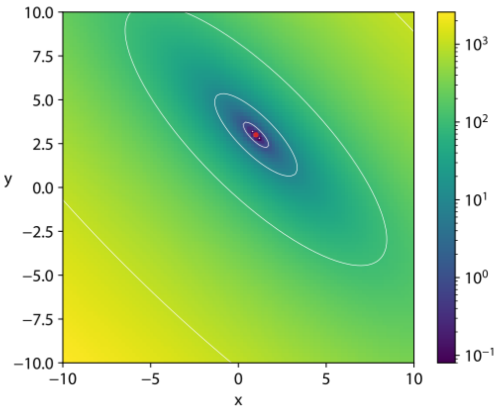

**Rosenbrock** function




**Himmelblau** function




**Beale** function




**Booth** function

Figure 3.6: Contour plot of the test functions used over their respective search space. Generated by Nschloe - Own work, CC BY-SA 4.0, `https://commons.wikimedia.org/w/index.php?curid=114939712`

In the second test case the LLM was tasked with finding three hyperparameters in an ACO algorithm, specifically Ant System (AS), when applied to solving TSP. The hyperparameters are shown in Table 3.2. Two prompting strategies were evaluated. The first explicitly mentioned Ant colony optimization and described the parameters as they are described in the literature, and the second simply described the parameters as $x_1, x_2$ and $x_3$ without additional context. The prompts used to generate the result are shown in Appendix B.

Table 3.2: Parameters of Ant Colony Optimization

| **Parameter** | **Description** | **Typical Value Range** |
|---|---|---|
| $\alpha$ | Pheromone factor | 0 - 5 |
| $\beta$ | Heuristic factor | 1 - 5 |
| $\rho$ | Pheromone evaporation rate | 0 - 1 |

# 4

# Results

The results are presented for each method separately. Additional parameter sweeps can be found in Appendix A and prompts used to generate the results in Appendix B. Resources to the code used to generate the results is also provided in Appendix A.

## 4.1 LLM used as algorithm implementer

Figure 4.1 shows optimality gap of LLM used as algorithm implementer applied to TSP for varying number of nodes. Nearest Neighbour algorithm is also shown for reference. A maximum of 10 algorithms were implemented for each problem instance, these are shown in Table 4.1. A maximum of 30 seconds was allocated to each algorithm, if no solution was provided within this time window, no solution was recorded. LLM as algorithm implementer is able to produce multiple algorithms which achieve a lower optimality gap than Nearest Neighbour. For small problem scales, computationally heavy brute-force algorithms produce the lowest optimality gap while heuristics such as k-opt and ant colony optimization achieve the lowest optimality gap in the larger problem scales. Figure 4.2 shows the results of using the LLM as algorithm implementer when the problem instance was given in the prompt. This was done for multiple problem instances of different number of nodes.

Table 4.1: Zero-shot feasibility of implementing various algorithms and heuristics to solve TSP. Infeasibility can be caused either by code not executing (bugs) or implementation errors causing poor performance.

| **Implemented algorithm** | **Feasibility rate** |
|---|---|
| Greedy | 94 % |
| Brute force | 84 % |
| Nearest Neighbour | 100 % |
| Two-Opt | 90 % |
| K-Opt | 81 % |
| Ant system (ACO) | 40 % |
| Simulated Annealing | 0 % |
| Genetic Algorithm | 0 % |

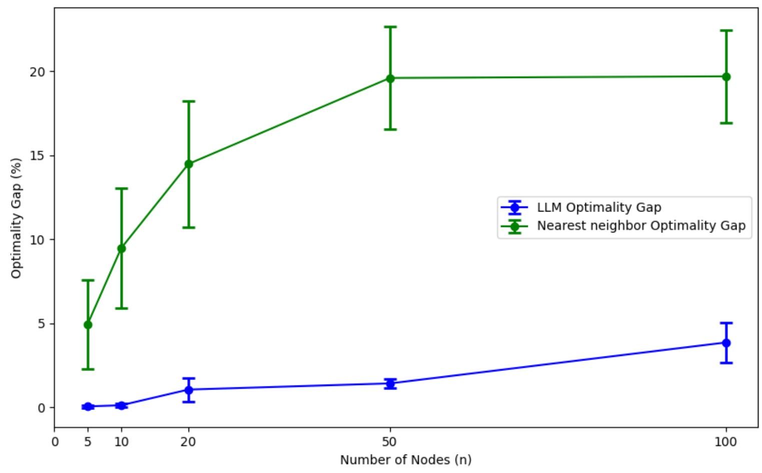



Figure 4.1: Optimality gap of LLM as Algorithm implementer and Nearest Neighbour for TSP problems of 5, 10, 20, 50 and 100 nodes. Data point are mean of 5 seeds and error bars show standard deviation.

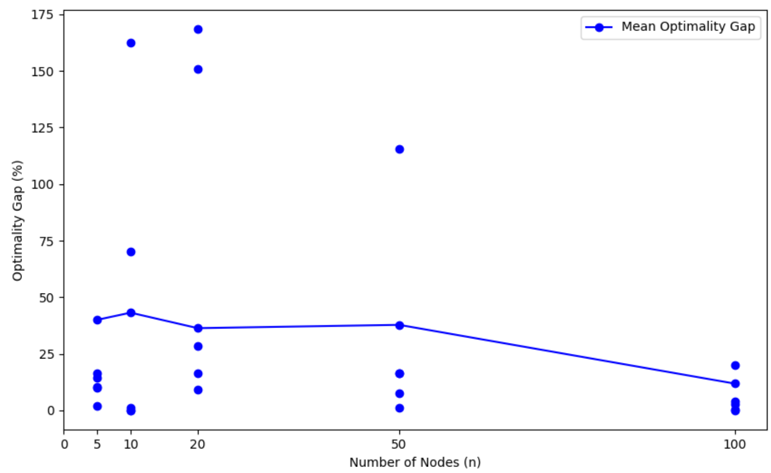



Figure 4.2: Optimality gap of LLM as algorithm implementer when problem instance was given in the prompt. Data points show optimality gap of 6 different algorithms generated by the LLM for each $n$.

## 4.2 LLM used as optimizer

The LLM used as optimizer is shown for three cases, TSP, Cost minimization CVRP and fleet revenue maximization CVRP. Figure 4.3 shows that LLM as optimizer applied to TSP performs better than Nearest Neighbour for 7 nodes, performs slightly better for 10 nodes and is outperformed at 15 nodes. LLM significantly outperforms random search, showcasing the fact that solutions were not found by brute force.

The results were generated using the parameter settings shown in Table 3.1 and the LLM was prompted to only respond with the solution, without reasoning and reflection.

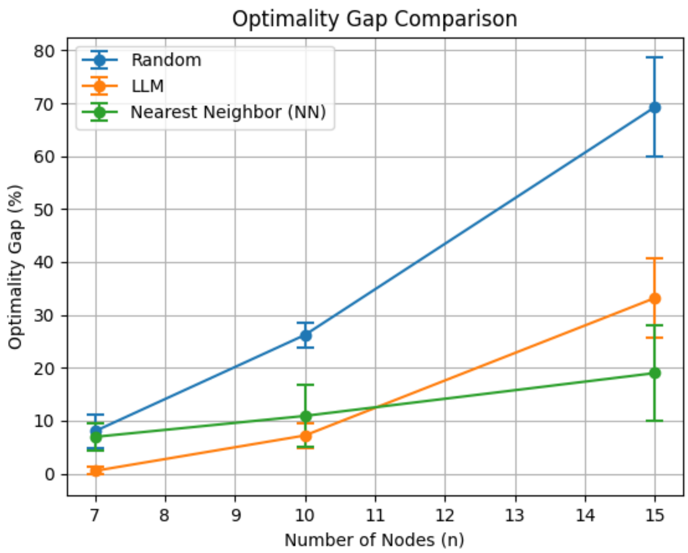



Figure 4.3: Mean Optimality Gap for traveling salesman problem of LLM (Mixtral-8x7b), random search and nearest neighbor for $n$ = 7, 10 and 15. Errorbars show standard deviation of 150 independent runs.

Figure 4.4 shows the optimality gap and feasibility rate for the cost minimization CVRP, both with and without the use of reflection. The reflection connection was used when the LLM provided an infeasible solution to the objective function, allowing the LLM to provide a correction to the provided solution. Optimality gap significantly increase with the number of customers. The use of reflection had varying effects on the performance, however, increased the feasibility rate. The feasibility rate still remained low, at best 40 percent, meaning more than half of all solutions provided by the LLM were infeasible. No comparison method was used due to the problem scale being so small. Heuristics designed to solve CVRP problems are able to find the optimal solution for all problem instances of this problem size.

Figure 4.5 shows the same metrics, but for the vehicle revenue maximization formulation of CVRP. Overall the optimality gap is significantly lower, both with and without the use of reflection. The feasibility rate, with the use of reflection also shows an increasing trend, indicating the LLM was easily able to find feasible solutions, given a chance to correct itself at most one time (1 reflection).

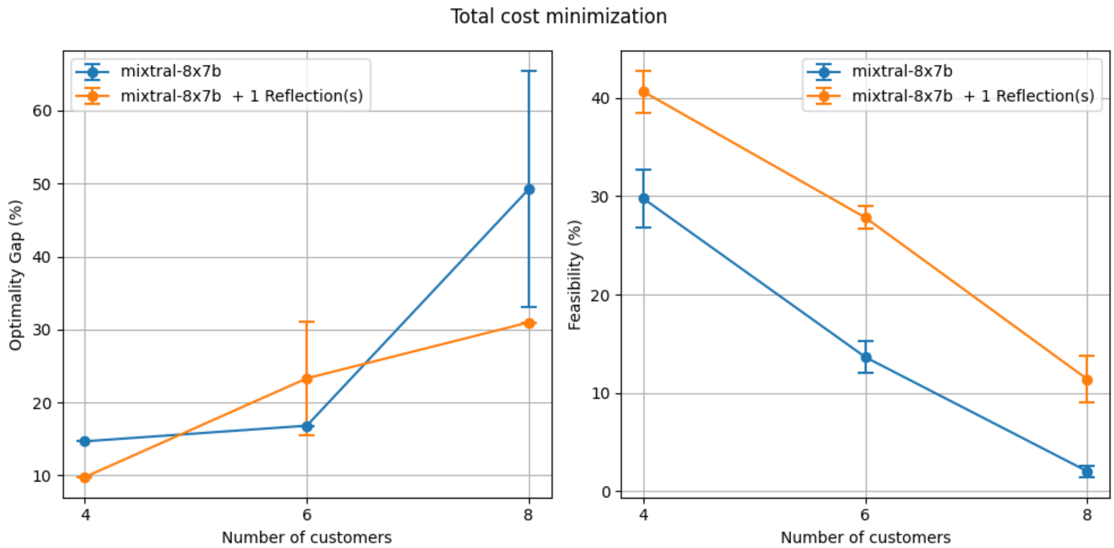



Figure 4.4: Optimality gap and feasibility rate for total cost minimization of CVRP. LLM used was Mixtral 8x7b using a maximum of 50 iterations, with and without the use of reflection. Results were averaged over 3 runs for 3 different seeds.

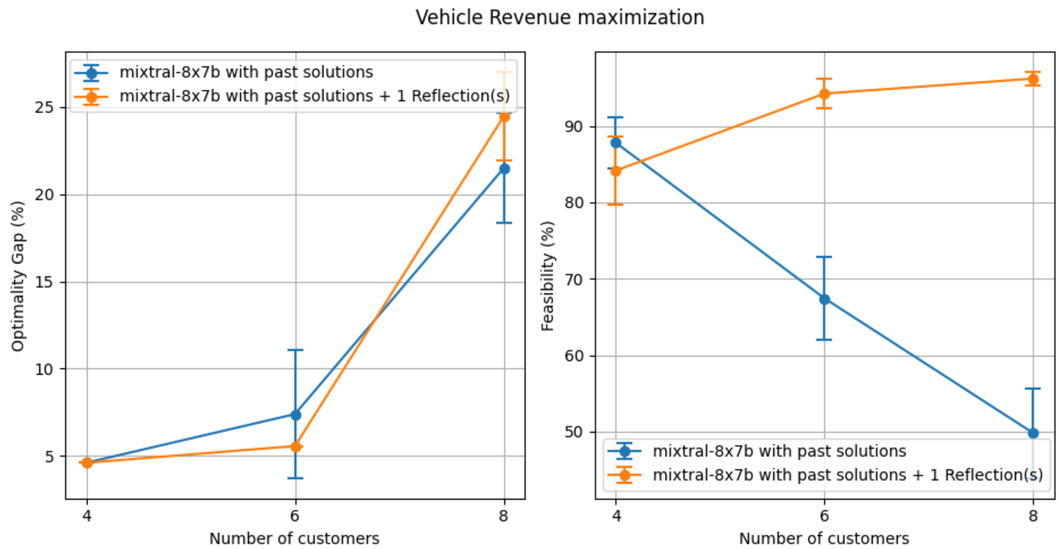
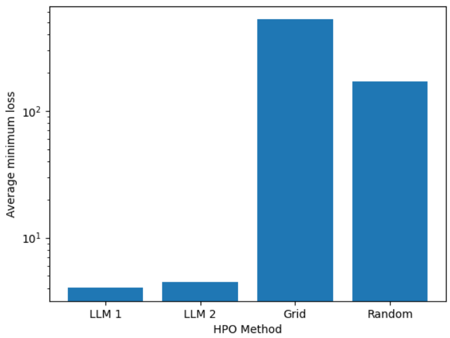


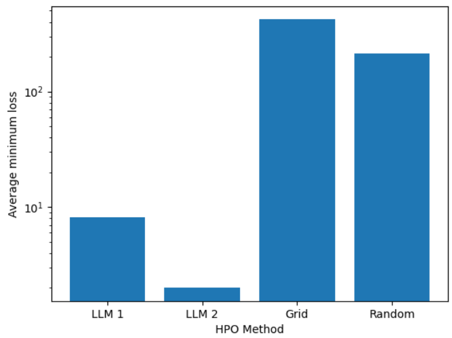

Figure 4.5: Optimality gap and feasibility rate for vehicle revenue maximization of CVRP. LLM used was Mixtral 8x7b using a maximum of 50 iterations, with and without the use of reflection. Results were averaged over 3 runs for 3 different seeds.

## 4.3 LLM used as hyperparameter optimizer

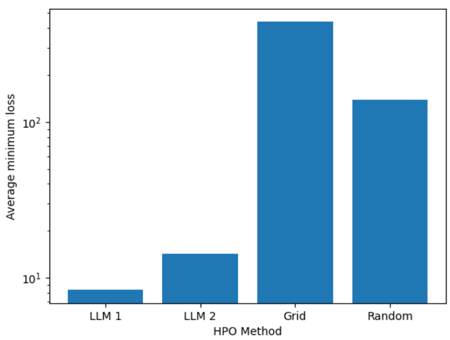
Figure 4.6 shows the results of using LLM as hyperparameter optimizer when optimizing the four different test functions described in the method. The LLM significantly outperforms both grid search and random search for all test functions, note the logarithmic scale on the y-axis. The results were averaged over five different seeds using LLAMA-3-70b set at a temperature of 0 with a maximum of 10 function evaluations. Grid search and random search also had the same budget of 10 function evaluations. Two prompting strategies were used by the LLM, prompt 1 gave no instructions on search strategy while prompt 2 instructed the LLM to prioritize exploration in its first function evaluations. Prompt 1 had the highest average performance however prompt 2 was superior for the Himmelblau function. It should be noted that while the LLMs outperformed both grid search and random search, they did not always find the global optima, instead the optimization process often converged to local minima.



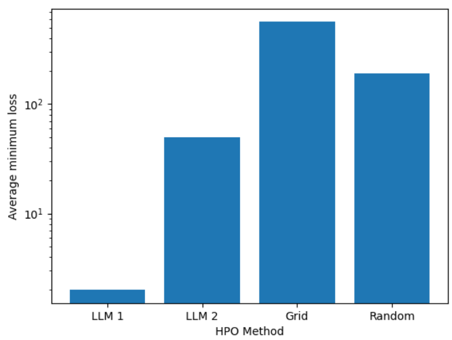

**Rosenbrock** function




**Himmelblau** function




**Beale** function




**Booth** function

Figure 4.6: Average minimum loss achieved by LLM using two prompts (LLM 1 and LLM 2), grid search and random search. All methods used 10 evaluations and the LLM used was LLAMA3-70b.

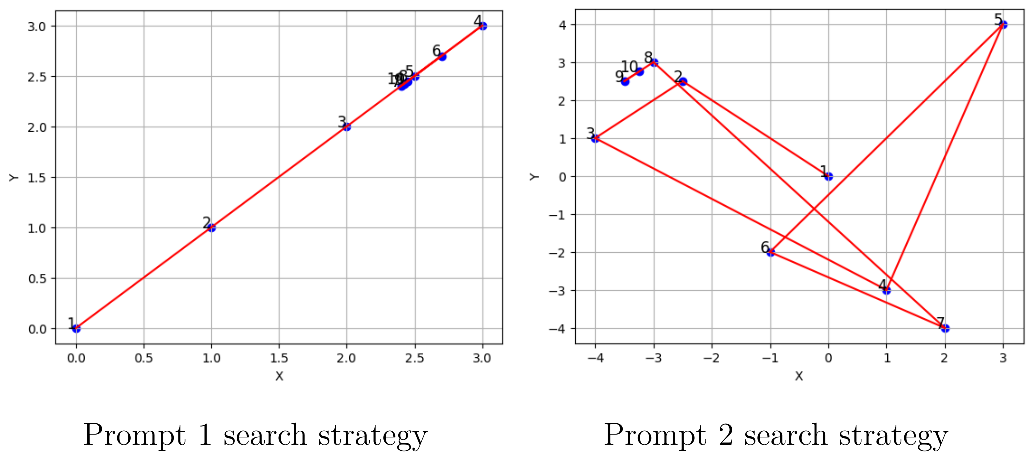



Figure 4.7: Search strategy of two different prompts for Himmelblau. Prompt 1 only ask to find parameters that minimize a function while prompt 2 explicitly mention to prioritize exploration in early function evaluations.

Figure 4.8 shows the optimality gap of using LLM as hyperparameter optimizer to optimize Ant system, an ant colony optimization method (ACO). The ACO is applied to a TSP instances of 50 nodes, since trial and error testing showed that the choice of hyperparameters had a large effect on performance for this problem scale. Two prompting strategies were evaluated along with a random search as a benchmark. Both prompting strategies outperformed random search, but with a much smaller margin compared to the previous experiments optimizing test functions. Results also show that providing ACO context in the prompt (LLM + ACO context) increased the optimality gap. The best performance was achieved by providing the variables as $x_1$, $x_2$ and $x_3$ without any explanation of their effect on the function being minimized.

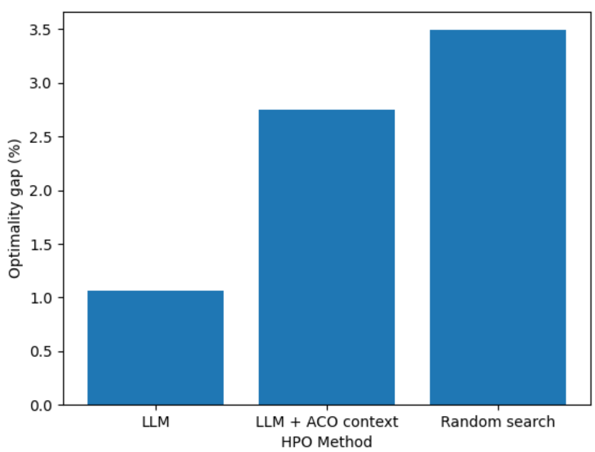



Figure 4.8: Optimality gap of using ant colony optimization to solve a TSP problem of 50 nodes for (1) LLM (without context), (2) LLM with ACO context provided in the prompt and (3) random search. Results were averaged over five seeds.

# 5

# Discussion

This thesis explores three methods to solve optimization problems using LLMs and this section will address the results with respect to the research question outlined in introduction. Performance, limitations and scaling with complexity is discussed for each method separately. Then, a discussion of suitable applications of the methods is provided followed by future research directions.

## LLM as Algorithm implementer

From the results in Figure 4.1 we can observe that the LLM is able to implement algorithms which significantly outperform Nearest Neighbour. These algorithm are shown in Table 4.1. It is able to implement both brute force methods for small scale instances and heuristics such as Ant Colony Optimization for larger scale problems.

In Figure 4.2 the LLM was tasked with implementing algorithms to solve a specific problem instance given in the input prompt. This was done in order to investigate if the LLM was able to adapt its choice in algorithm depending on the given problem instance. However, Figure 4.2 shows that the performance of the implemented algorithms were very unreliable and had worse performance compared to not giving the problem instance in the prompt. This was especially the case for smaller problem scales, around 5 and 10 nodes, where the LLM rarely chose to implement brute-force algorithms, even though the problem scale allowed for it. For large problem scales the LLM did not attempt to implement any algorithms that exceeded the maximum time limit, instead heuristics were implemented in all test cases.

It is possible that the LLM could adapt better to the problem instance if instead, an agentic framework would be used. In this framework, the problem scale is instead given as a description of the number of nodes and edges. Additionally the LLM could potentially adapt its choices in algorithms if it was able to get feedback from executing the code in each iteration. For example, if past provided algorithms by the LLM produced no solutions due to exceeding maximum runtime, then more efficient algorithms could be prioritized. If, on the other hand, more time was available, then slower and more accurate algorithms could be prioritized instead.

One of the main limitations with this method is to produce error-free algorithms. Feasibility of implementing algorithms by the LLM using a single prompt shown in Table 4.1. These errors include both code execution errors and errors in the algorithm. Consistency can however be improved using reflection where the errors in the code is fed back to the LLM. Also, allowing the LLM to produce multiple algorithms significantly increases the consistency since only one of them has to be implemented correctly. Some algorithms appear to be more difficult to implement than others, these are usually algorithms that are less common in the literature. This highlights another main limitation of this method, which is that it relies on having seen the algorithms in its pre-training phase. Algorithms that have been implemented countless times, for example, Nearest Neighbour is significantly more likely to be implemented without error compared to a genetic algorithm.

## LLM as Optimizer

Figure 4.3 shows the performance for TSP of different problem scales. For small problem scales, around seven nodes, the LLM outperforms nearest neighbour and finds the optimal solution for the majority of problem instances, however, as the number of nodes is increased, the LLM is not able to cope with the increased computational complexity and is instead outperformed by nearest neighbour. This is also observed in Figures 4.4 and 4.5 where the optimality gap significantly increases with number of customers. As expected, CVRP is more difficult to solve than TSP. For eight customers, the optimality gap of CVRP is around 25 and 30 percent for the two formulations with reflection, compared to an optimality gap of 7 percent for TSP of 10 nodes. Thus we can see that both computational and systematic complexities heavily effect the performance of the method.

There are, however, two ways to aid the LLM in dealing with high problem complexity. Firstly, by using reflection, the LLM is given a chance to correct its mistakes, which leads to a higher feasibility rate and more explored unique solutions. As a result, the optimality gap also decreases, which can be seen in both Figure 4.4 and 4.5. The second way to deal with problem complexity is to formulate the optimization problem in a way that reduces the number of constraints. The vehicle revenue maximization formulation, as described in the method, allows solutions where not all customers are visited. This drastically increases the space of possible solutions, but it also allows the LLM to more easily find feasible solutions. These solutions can then be evaluated and added to the meta-prompt and later improved upon in future iterations. In the total cost minimization formulation, the LLM has harder time to produce solutions that satisfy all constraints, leading to the majority of LLM calls to be essentially wasted.

## LLM as Hyperparameter optimizer

In this method the LLM was used as hyperparameter optimizer. Figure 4.6 show that the LLM significantly outperforms both grid search and random search when applied to smooth function landscapes. When applied to finding the hyperparameters to an Ant Colony Optimization (ACO), the results show that the LLM only marginally outperforms random search. This is likely due to the ”bumpy” function landscape of the ACO, making the search process significantly harder. Figure 4.8 also shows that providing additional context of the ACO hyperparameters increases the optimality gap. This result was unexpected, since intuitively, additional context of the parameters would allow the model to choose values based on past research. One explanation of why additional context results in worse performance could be that the LLM places too much emphasis on typical parameters values used in the literature, instead of searching for ones that seem to perform well for the given problem instance.

A key observation of this method was that the prompt heavily effects the search strategy, see Figure 4.7. Without specifying any requirements in the prompt regarding the generation of hyperparameters, the LLM tends to quickly converge to local minima, without adequately exploring the search space. However, this can easily be counteracted by prompting the LLM to prioritize exploration in the first couple of function evaluations. This trade-off between exploration and exploitation in the search space is a very common dilemma for solving optimization problem, and would be an interesting continuation of exploring this method.

## 5.1 Suitable applications of methods

While LLMs are able to implement various efficient algorithms for solving optimization problems such as TSP, they are not able to implement state-of-the-art human designed algorithms or algorithms which outperform human designed ones. Therefore, in the current state of LLMs, they are likely best utilized as hyperparameter optimizers for all use cases where high performance is the priority.

However, hyperparameter optimization builds on the existence of efficient and high performance algorithms or heuristics that solve the given optimization problem, which is not always the case. An example of such a case is prompt optimization, which is the task of finding an input prompt to an LLM that maximize some given objective function. Prompt optimization is typically solved by trial and error or by using LLMs, since there are no algorithms or heuristics available. This is where LLMs are best used as optimizers, since it requires no knowledge of implementing algorithm or any gradient evaluation of the objective function. Still, using LLMs as optimizers is heavily limited by the problem complexity, as shown in the result. It is therefore unlikely that LLMs, in their current state, can be used as optimizers to solve problems with already existing human designed algorithm. Although, this may very well change in the future, if the capabilities of LLMs continues to expand.

Finally, we have the method of using LLMs as algorithm implementer. Since LLMs are not able to beat state-of-the-art human designed algorithms and the use of implementing algorithms on problems without preexisting algorithms remains unexplored, it leaves little room for applicable use cases. However, the method can still be used as a tool to generate functioning algorithms for testing and experimentation.

## 5.2 Future research directions

There are many interesting future research directions to consider when using LLMs to solve optimization problems. When using the LLM as algorithm implementer, the first possible improvement would be to create an agentic framework, as previously described. This could improve the LLMs ability to adapt to the problem scale and also allow for implementation of more complex algorithms through debugging feedback. From a broader perspective, it would also be interesting to evaluate if LLMs are able to implement algorithms for problems without existing algorithms in the literature. Both TSP and VRP are highly studied problems with countless algorithms and heuristics. This results in the LLM mainly implementing these exact human designed algorithms. It would therefore be of interest to investigate if LLMs are able to generalize the implementation of algorithms to unseen problems in its training data. This could also give an indication if LLMs are ever going to be able to surpass human designed algorithms or if they appear to be limited by the algorithms in their training data. If it is the case that LLMs will not be able to surpass human designed algorithms, then perhaps a different use case for LLMs would be finding reductions between problems, that is, a translation of a problem to another. This would allow us to use our human designed algorithms to solve more problems, without requiring LLMs to be able to implement algorithms that outperform human designed ones.

For the method of using LLMs as optimizer, future research should mainly focus on the complexity scaling problem. As LLMs continue to improve, it is likely that they will be able to solve more and more complex problems, but there are also other techniques that could be explored to deal with the complexity, one of which being fine-tuning. To the best of found knowledge, no evaluation has been done using a task specific fine-tuned LLM as optimizer. Fine-tuning could potentially aid the model to better understand and provide solutions to the given optimization problem. Another area of exploration is the representation of past solutions in the meta-prompt. It is not clear exactly how the LLM uses past solutions in order to generate better solutions, other than it is performing some pattern recognition. It is is possible that there are other ways of representing past solutions that better guide the LLM towards finding higher quality solutions. One could for instance include explanation or hypotheses of why the past solutions gave a good evaluation, in order to guide the search for better solutions.

Then there is the method of using LLM as hyperparameter optimizer. This method has the potential to be widely applicable, however it requires more research into how prompting affects the search strategy, and which search strategy performs the best. This could for example be implemented as an agentic framework, that finds a balance between exploration and exploitation. It would also be relevant to finding ways to prevent the LLM to converge on local optima.

# 6

# Conclusion

In conclusion, this thesis has outlined three methods that use Large Language models (LLM) to solve optimization problems by using the LLM as algorithm implementer, optimizer and hyperparameter optimizer. The performance of these methods are mainly evaluated on the traveling salesman problem and the results indicate that LLMs are not yet able to implement state-of-the-art human designed algorithms. Instead, the current state of LLMs are likely best served as optimizers when considering problems without known human designed algorithms, and hyperparameter optimizers for problems where such human designed algorithms do exist. The use of LLMs to solve optimization problems is a novel research area with many promising future research directions.

# Bibliography


[1] W. X. Zhao, K. Zhou, J. Li, *et al.*, *A survey of large language models*, 2023. arXiv: 2303.18223 [cs.CL].

[2] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, *Attention is all you need*, 2023. arXiv: 1706.03762 [cs.CL].

[3] T. B. Brown, B. Mann, N. Ryder, *et al.*, *Language models are few-shot learners*, 2020. arXiv: 2005.14165 [cs.CL].

[4] J. Kaplan, S. McCandlish, T. Henighan, *et al.*, *Scaling laws for neural language models*, 2020. arXiv: 2001.08361 [cs.LG].

[5] S. Qiao, Y. Ou, N. Zhang, *et al.*, *Reasoning with language model prompting: A survey*, 2023. arXiv: 2212.09597 [cs.CL].

[6] C. Yang, X. Wang, Y. Lu, *et al.*, *Large language models as optimizers*, 2023. arXiv: 2309.03409 [cs.LG].

[7] S. Mirchandani, F. Xia, P. Florence, *et al.*, *Large language models as general pattern machines*, 2023. arXiv: 2307.04721 [cs.AI].

[8] M. R. Zhang, N. Desai, J. Bae, J. Lorraine, and J. Ba, *Using large language models for hyperparameter optimization*, 2023. arXiv: 2312.04528 [cs.LG].

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

[10] T. Kudo and J. Richardson, *Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing*, 2018. arXiv: 1808.06226 [cs.CL].

[11] M. E. Peters, M. Neumann, L. Zettlemoyer, and W.-t. Yih, *Dissecting contextual word embeddings: Architecture and representation*, 2018. arXiv: 1808.08949 [cs.CL].

[12] A. Gu and T. Dao, *Mamba: Linear-time sequence modeling with selective state spaces*, 2023. arXiv: 2312.00752 [cs.LG].

[13] A. Srivastava, A. Rastogi, A. Rao, *et al.*, "Beyond the imitation game: Quantifying and extrapolating the capabilities of language models," *Transactions on Machine Learning Research*, 2023, ISSN: 2835-8856. [Online]. Available: https://openreview.net/forum?id=uyTL5Bvosj.

[14] M. Jünger, G. Reinelt, and G. Rinaldi, "Chapter 4 the traveling salesman problem," in *Network Models*, ser. Handbooks in Operations Research and Management Science, vol. 7, Elsevier, 1995, pp. 225–330. DOI: https://doi.org/10.1016/S0927-0507(05)80121-5. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0927050705801215.

[15] G. B. Dantzig and J. H. Ramser, "The truck dispatching problem," *Management Science*, vol. 6, no. 1, pp. 80–91, 1959. DOI: 10.1287/mnsc.6.1.80.

[16] D. Rickles, P. Hawe, and A. Shiell, "A simple guide to chaos and complexity," en, *J Epidemiol Community Health*, vol. 61, no. 11, pp. 933–937, Nov. 2007.

[17] J. Wei, Y. Tay, R. Bommasani, *et al.*, *Emergent abilities of large language models*, 2022. arXiv: 2206.07682 [cs.CL].

[18] A. Holtzman, P. West, and L. Zettlemoyer, *Generative models as a complex systems science: How can we make sense of large language model behavior?* 2023. arXiv: 2308.00189 [cs.LG].

[19] S. Arora and B. Barak, *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009.

[20] L. Fortnow, "The status of the p versus np problem," *Commun. ACM*, vol. 52, pp. 78–86, Sep. 2009. DOI: 10.1145/1562164.1562186.

[21] P. Liashchynskyi and P. Liashchynskyi, *Grid search, random search, genetic algorithm: A big comparison for nas*, 2019. arXiv: 1912.06059 [cs.LG].

# A

# Appendix A

The code used to generate the result can be found at `https://github.com/Oliver777int/llm-optimization`.

## A.1 LLM parameter sweep

Table A.1: Temperature Parameter Sweep using Mixtral 8x7B to solve a 7 node traveling salesman problem using at most 50 iterations.

| T | Optimal Solution Found | Optimality Gap (%) | Avg. Iteration Steps |
|---|---|---|---|
| 0.1 | 15/30 (0.5) | $9.4 \pm 1.038$ | 29 |
| 0.25 | 12/30 (0.4) | $10.7 \pm 1.008$ | 26 |
| 0.5 | 15/30 (0.5) | $7.1 \pm 0.758$ | 23 |
| 1.0 | 22/30 (0.73) | $4.3 \pm 0.738$ | 23 |
| 1.5 | 18/30 (0.6) | $5.5 \pm 0.699$ | 22 |

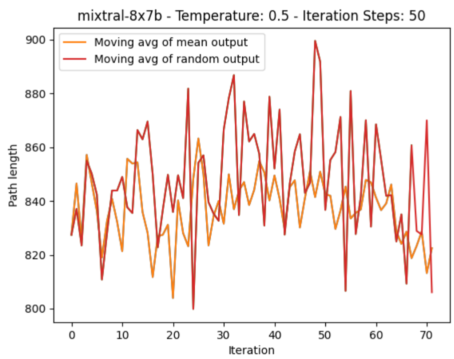



Figure A.1: Sorted solution-score list. Average best LLM: 678.5 +- 68.4.

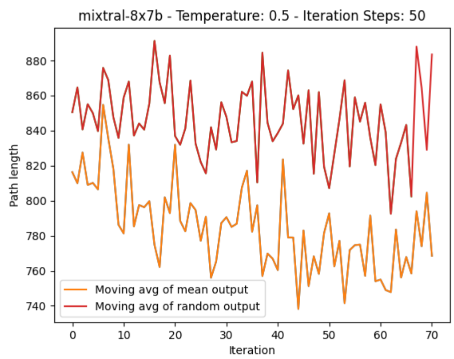



Figure A.2: Shuffled solution-score list. Average best LLM: 592.9 +- 47.2.

Figure A.3: Comparison of sorted and shuffled solution-score lists for one run. Score (path length) is averaged over 30 seeds.

# B

# Appendix B

This Chapter shows the prompts used to generate the results.

**LLM as Algorithm Implementer - Prompt 1**

Your task is to implement an algorithm that solves the following problem:
Given a set of nodes with coordinates in the plane, find the shortest route that visits all nodes exactly once and returns to the starting node.

The algorithm should take a dictionary as input where keys are nodes and values are coordinates,
example: graph = {graph}.

The algorithm should be written as a function called shortest_route and print "Shortest path: [LIST_OF_NODES]". Write ```python before providing the code and only use standard packages.

**LLM as Algorithm Implementer - Prompt 2**

I have a dictionary where keys are nodes and values are coordinates,
graph = {graph}

Write an algorithm that prints the nodes of the shortest route from a starting node, that visits all nodes exactly once and returns to the starting node.

The algorithm should not include any non-standard packages. Explicitly write the graph in the code, write ```python before the code and format your output as: "Shortest path: [LIST_OF_NODES]".

**Prompt used between iterations**

Provide another algorithm that is different from the previous ones. Write ```python before the code and format the output print as: "Shortest path: [LIST_OF_NODES]".

Figure B.1: Prompts used to generate results of LLM as algorithm implementer. Prompt 1 was used to generate results in Figure 4.1 and Prompt 2 for 4.2. Prompt used between iterations was used for both.

## LLM as Optimizer - Prompt 1 TSP

You are given a list of points with coordinates below:
0: (-3, -15), 1: (-41, -58), 2: (-2, 47), 3: (76, -28), 4: (41, -30), 5: (-2, 87), 6: (95, 47), 7: (-98, 99), 8: (-37, 69), 9: (-96, 12).

Below are some previous traces and their lengths. Lower length is better.
<trace> 2,9,5,7,8,1,3,4,0,6 </trace> length: 928
<trace> 4,5,8,2,7,1,9,0,3,6 </trace> length: 919
<trace> 9,7,1,4,2,5,6,3,0,8 </trace> length: 904

Give me a new trace that is different from all traces above, and has a length lower than any of the above. The trace should traverse all points exactly once. Do not write code. The trace should start with <trace> and end with </trace>.

## LLM as Optimizer - Prompt 2 CVRP Total cost minimization

You are given a set of nodes with x, y coordinates in the plane below, node 0 is the depot:
{0: (-66, 45), 1: (95, -84), 2: (-35, -70), 3: (26, 94), 4: (15, 20)}

There are customers with demands at the following locations:
Node 1 with demand of 10 pallets.
Node 2 with demand of 6 pallets.
Node 3 with demand of 4 pallets.
Node 4 with demand of 10 pallets.

You have the following vehicles able to make deliveries. They all start from the depot (node 0) and return to the depot after making all deliveries.

Vehicle a with capacity of 20 pallets.
Vehicle b with capacity of 12 pallets.

Output an optimal set of routes for the fleet of vehicles that minimize the total cost of traveling. The set of routes should fufill all demands without violating the carrying capacity of any vehicle in the fleet. The cost of traveling is only dependent on the traveling distance between nodes. Vehicles cannot be used multiple times. Make sure to also provide the total minimized cost.

## LLM as Optimizer - Prompt 3 CVRP Fleet revenue maximization

You are given a set of nodes with x, y coordinates in the plane below, node 0 is the depot:
{0: (-66, 45), 1: (95, -84), 2: (-35, -70), 3: (26, 94), 4: (15, 20)}

There are customers with demands at the following locations:
Node 1 with demand of 10 pallets.
Node 2 with demand of 6 pallets.
Node 3 with demand of 4 pallets.
Node 4 with demand of 10 pallets.

You have the following vehicles able to make deliveries. They all start from the depot (node 0) and return to the depot after making all deliveries. Each delivered pallet yields a revenue of 100.

Vehicle a with capacity of 20 pallets.
Vehicle b with capacity of 12 pallets.

The revenue of a vehicle is equal to 100 for every delivered pallet subtracted by the route cost. The route cost is equal to the total distance traveled, starting from the depot, to the customers and back to the depot.

Your objective is to output an optimal set of routes for the fleet of vehicles that maximize the fleet revenue. Each vehicle can only traverse one route. Provide an optimal solution and maximum fleet revenue and make sure not to violate the maximum capacity of the vehicles.

Figure B.2: Prompts used to generate results of LLM as optimizer for TSP and VRP, TSP prompt was obtained from ORPO [6]

**Prompt 1 - 2 variable test functions**

You are optimizing a function with two inputs.

x1 must be in range {xlim}.
x2 must be in range {ylim}.

I want you to predict values that minimize the loss of a function. I will tell you the value of the function before you try again. Do not put new lines or extra characters in your response. Format your output with json as follows: {format}

**Prompt 2 - 2 variable test functions**

You are optimizing a function with two inputs.

x1 must be in range {xlim}.
x2 must be in range {ylim}.

I want you to predict values that minimize the loss of a function. I will tell you the value of the function before you try again. Early you should favour exploration. If the values of the parameters seem to converge it means you found a minimum, in that case search somewhere else, do not let the parameters converge for too long. Do not put new lines or extra characters in your response. Format your output with json as follows: {format}

**Prompt used between iterations**

x1=3 and x2=3.5 gave an evaluation of 859, provide the next values of x1 and x2.

Figure B.3: Prompts used to generate results of LLM as hyperparameter optimizer for two variable test functions, see Figure 4.6. Prompt format was obtained from Zhang et al. [8]

### Prompt 1 - Ant colony optimization

You are optimizing a function with three inputs.

x1 must be in range {xlim}.
x2 must be in range {ylim}.
x3 must be in range {zlim}.

I want you to predict values that minimize the loss of a function. I will tell you the value of the function before you try again. Do not put new lines or extra characters in your response. Format your output with json as follows: {format}

### Prompt 2 - Ant colony optimization

You are helping me find the best hyperparameters to my ant colony optimization with two inputs, alpha, beta and the pheromone level rho.

alpha must be in range {xlim}.
beta must be in range {ylim}.
rho must be in range {zlim}.

I want you to find values of alpha, beta and rho that minimize the loss function. I will tell you the value of the loss function before you try again. Do not put new lines or extra characters in your response. Format your output with json as follows: {format}

### Prompt 1 used between iterations

x1=2.5, x2=3.2 and x3=0.7 gave an evaluation of 796,
provide the next values of x1, x2 and x3.

### Prompt 2 used between iterations

alpha=2.9, beta=2.9 and rho=0.6 gave an evaluation of 925,
provide the next values of alpha, beta and rho.

Figure B.4: Prompts used to generate results of LLM as hyperparameter optimizer for Ant Colony Optimization, see Figure 3.5