Neurocomputing 675 (2026) 132976

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

Tuning metaheuristic parameters with the use of Large Language Models

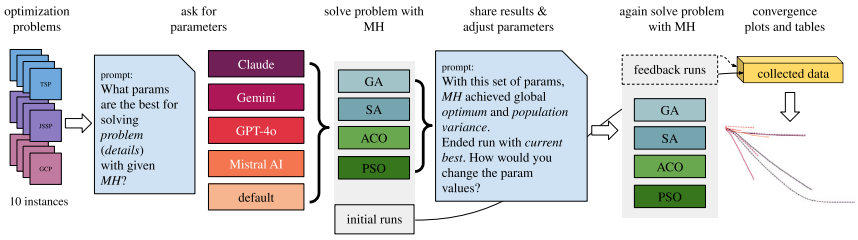
Alicja Martinek [a, b, *], Ewelina Bartuzi-Trokielewicz [b], Szymon Łukasik [a, b], Amir H. Gandomi [c, d]

[a] AGH University of Kraków, Cracow, Poland
[b] NASK - National Research Institute, Warsaw, Poland
[c] University of Technology Sydney, Sydney, Australia
[d] Obuda University, Budapest, Hungary

HIGHLIGHTS

- LLMs can tune metaheuristic parameters effectively without domain knowledge or specialized training.
- GPT-4o and Gemini suggest efficient parameter sets; Llama3 tends to overestimate key values.
- Simulated Annealing benefits most from LLM tuning due to temperature and schedule sensitivity.
- LLMs tune popular metaheuristics better, likely reflecting exposure to scientific literature.

GRAPHICAL ABSTRACT



ARTICLE INFO

Keywords:
Optimization
Large Language Models
Metaheuristics
Bio-inspired computation
Prompt engineering
Evolutionary algorithms

ABSTRACT

Since their explosion in popularity, the impact of Large Language Models (LLMs) has been evident in almost every aspect of life. This study examines whether LLMs can be utilized for tuning metaheuristic algorithms through the selection of their parameters. To verify this hypothesis, ten instances each of three well-known combinatorial optimization problems, Graph Coloring, Job-Shop Scheduling, and Traveling Salesman, were solved using heuristic optimizers guided by LLMs, including genetic algorithm, ant colony optimization, particle swarm optimization, and simulated annealing. Parameter values were generated by prompting several state-of-the-art LLMs with problem complexity descriptors and the set of tunable parameters. A two-stage procedure was employed: an initial run based on general problem characteristics, followed by a feedback run that used performance metrics such as average fitness, variance, and convergence behavior. Default settings from the Python-based Mealpy library served as the baseline for comparison.

Results, aggregated over 900 optimizer runs, show that LLMs are capable of proposing parameter configurations that outperform defaults in terms of final objective value and convergence speed. This effect is particularly pronounced in simulated annealing and Traveling Salesman problem settings. The findings suggest that LLMs possess a high degree of generalization and contextual understanding in the domain of optimization and can serve as practical assistants in heuristic algorithm design and tuning.

1. Introduction

The data revolution currently happening around the world is a motor for advancements of new methods for data analysis. It is driven by the wide availability of novel data sets and the interest of both academia and industry in the progression of innovative Machine Learning and Deep Learning algorithms.

* Corresponding author at: AGH University of Kraków, Cracow, Poland.
Email address: martinek@agh.edu.pl (A. Martinek).

https://doi.org/10.1016/j.neucom.2026.132976
Received 21 June 2025; Received in revised form 23 January 2026; Accepted 3 February 2026
Available online 5 February 2026

A huge amount of gathered language data facilitates the development of Large Language Models (LLMs). These methods have revolutionized the computing industry by finding countless use cases in almost every aspect of life. LLMs are transformer-based neural networks. They heavily rely on the attention mechanism introduced in 2017 in [1]. First widely used LLM, Google's BERT [2] was rolled out a year later and found applications in many tasks of Natural Language Processing. However, the actual transformation started in 2022, when OpenAI released its ChatGPT.

Despite the positive aspects associated with LLMs, it has to be remembered to use them cautiously. Models tend to fabricate content when they lack knowledge in a given area. This phenomenon is called hallucination [3]. Another reason not to take LLMs' answers for granted is that they can perpetuate stereotypes, share unethical content, or exhibit racial bias [4] - these are the justifications for extensive output filtering embedded in the contemporary LLMs.

Applications of Large Language Models include advanced chatbots, content creation, code writing, summarization, and many more [5]. The degree of attention devoted to LLMs is without precedent. Companies, researchers, and even governments are actively working on creation of their own Large Language Models.

Constantly growing data volumes and new algorithms provide an excellent playground for developing new Metaheuristic Algorithms (MHAs). Optimization in many areas is still an open question, as the search for faster and computationally cheaper methods will always be an exciting topic for scientists. Consequently, the number of MHAs is still growing, making it hard to stay up-to-date with all new developments. Recent classifications of heuristic optimizers result in collections of more than 500 distinguished methods [6,7].

Every metaheuristic has its set of distinct parameter values, advised applications, and additional information that characterizes that algorithm. Hence, it becomes tedious to grasp and remember all the details regarding MHAs. To manage such a large amount of material, comprehensive guidance is needed. An agent that could provide suggestions for selecting an algorithm and, moreover, its parameters suited for a given problem would be a game changer and could reduce the entry level barrier for appropriate utilization of metaheuristics.

The need for such a guide is a space where Large Language Models can thrive. Their ability to understand natural language and code can potentially be used as guidance for tuning heuristic optimizers. Such a possibility can provide a head start as opposed to default setups of MHAs and steer calculations in the already more desired direction.

The research presented in this paper is an extension of [8], which deepens the study by analyzing more problems and their instances. Presented research follows the same methodology; however, it involves multiple instances of three combinatorial problems and utilizes an extended list of Large Language Models and classic parameter search techniques including random and grid searches. The aforementioned methodology uses LLMs to select parameter values for optimization tasks with little knowledge about the problem domain. After solving the problems with suggested settings, LLMs are again asked to adjust the parameters based on initial results. The performance of the feedback-based round is then collected and analyzed. In addition, this study evaluates the cost of computation and seeks a correlation between computational expenses and the quality of the results achieved. This work contributes twofold. The first involves thorough experimentation to check whether current state-of-the-art large language models can tune parameters of MHAs in the context of combinatorial optimization. The second revolves around prompt engineering to effectively perform the aforementioned tuning. In addition, this work highlights the novel application of LLMs.

To evaluate the effectiveness of LLM-guided parameter tuning, this study focuses on three classical combinatorial optimization problems: Graph Coloring (GCP), Job-Shop Scheduling (JSSP), and Traveling Salesman Problem (TSP). These problems are well-established in the literature, known for their computational complexity and relevance in practical applications. Their diversity in structure and solution space makes them ideal testbeds for assessing the generalization capabilities of LLMs across different optimization scenarios.

In addition to experimental evaluation, the work also focused on developing command engineering techniques that enable the effective use of LLMs in heuristic optimization. This included creating detailed, configured queries that help language models generate useful outputs.

The growing complexity of metaheuristic algorithms makes parameter tuning a crucial yet costly task. Classical methods such as grid, random, or Bayesian search are computationally intensive and problem-specific. In contrast, Large Language Models, with their reasoning and language understanding capabilities, offer a new paradigm for analyzing algorithmic descriptions and proposing suitable parameter configurations. Despite their potential, the use of LLMs in heuristic optimization remains largely unexplored.

This paper is structured as follows: given the brief introduction and motivation, Section 2 follows with an explanation of concepts required to fully comprehend the experiments conducted in the study. Section 3 delivers the analysis of related work that lies at the intersection of LLMs and MHAs. The following Section 4 thoroughly describes the methodology undertaken in this research. Section 5 provides practical details on running calculations. Results and their analysis are contained in Section 6. Finally, Section 7 presents conclusions and outlines possibilities for further improvements.

2. Preliminaries

2.1. Large Language Models

Large Language Models (LLMs) represent a significant advancement in the field of artificial intelligence, particularly in natural language processing and generation. These models are primarily designed to understand and generate human-like text, leveraging vast amounts of data and sophisticated neural network architectures. This work considers the ChatGPT by OpenAI, Gemini by Google, Claude by Anthropic, and Le Chat by Mistral AI models, analyzing their potential use in tuning heuristic optimizers. The research attempted to assess how advanced natural language processing technologies can impact the efficiency and effectiveness of metaheuristic algorithms (MHAs) in the context of combinatorial optimization. All selected LLMs were used in their publicly available, off-the-shelf configurations, without any fine-tuning or domain-specific adaptation. Their accessibility via commercial APIs, strong benchmark performance, and ability to engage in structured prompt-based interaction make them well-suited for evaluating the potential of natural language-guided parameter tuning in metaheuristic optimization.

The **GPT-4o** [9] model, developed by OpenAI, belongs to the Generative Pre-trained Transformer (GPT) [10,11] family. GPT-4o and its predecessors are built upon the Transformer model, which uses self-attention mechanisms to process text. GPT-4o has approximately 200 billion parameters, allowing it to handle a wide range of natural language processing tasks with improved performance and accuracy. "o" in the model's name refers to its ability for multi-modal data understanding. The "omni" part of the model allows it to engage in more complex and diverse tasks.

Anthropic's **Claude Sonnet 3.7** [12] offers advanced reasoning with the possibility of controlling the length of model's thinking process. The number of trained parameters is not officially reported by Anthropic. The authors claim that Sonnet 3.7 is well fitted for coding and being an AI agent, which makes it a great candidate for tuning MHAs' parameters. High performance on benchmarks and reduced model's response time, are accredited to the dynamic neural architecture. This mechanism allows switching between deep and shallow layers depending on the complexity of the given task.

Gemini Flash 2.0 [13], developed by Google, represents a significant advancement in machine learning architectures by integrating the robust capabilities of the Transformer architecture with the innovative Mixture of Experts (MoE) [14] approach. The Gemini Flash 2.0 model we

utilized features a substantial 137 billion parameters, underscoring its capacity to handle complex, large-scale tasks across various domains. The novel aspect of Gemini's architecture is the incorporation of the MoE, which is a method designed to scale model capacity without a proportional increase in computational demands. It achieves this by distributing different parts of the input data to different 'expert' networks within the larger model framework, each specializing in different aspects of the problem.

The last model analyzed in our study is **Mistral Large 2** [15,16] by Mistral AI. It is trained with 123 billion parameters and has a context window of 128k. This model is specifically tailored to develop conversational intelligence, with an emphasis on maintaining long context and personalizing interactions. The ability of this model to appropriately process and adapt to complex contexts, while fitting responses to specific user scenarios, makes it very suitable for applications involving dynamic optimization environments where metaheuristic algorithms are used. One of the key innovations in Le Chat is the use of the Sliding Window Attention (SWA) technique [17], which addresses the challenges of standard attention mechanisms that scale quadratically with the length of token sequences. SWA limits the attention span of each token to a fixed window, which reduces computational costs and enables efficient processing of longer data sequences. Thanks to the use of SWA, Le Chat can more effectively manage relationships between tokens, which translates into better context preservation and deeper structural and semantic understanding of the text. This feature is particularly beneficial in the application of metaheuristic algorithms, where quick and efficient adaptation to changing conditions is crucial.

The analysis focused on the proficiency of these models to generate new, effective parameter configurations and their impact on accelerating and increasing the effectiveness of optimization processes. A key element of the study was the models' ability to interpret and synthesize information, which can translate into better adaptation of heuristic strategies over time.

2.2. Metaheuristic algorithms

Metaheuristic Algorithms (MHAs) are one of key concepts in the field of Computational Intelligence. This family of methods is distinctive for implementing real-life concepts in order to solve wide spectrum of optimization tasks. The versatility present in available implementations, as well as ongoing popularity and constant development of new heuristic optimizers, reflects their broad applicability. Heuristic optimizers deliver high-level frameworks for performing complex search processes. As nature is very efficient in optimizing a variety of routines, it is a great inspiration for developing MHAs. Surrounding environment, the behavior of animals, physical phenomena, sociology, and even music can be utilized as a blueprint for an optimizer [18–20]. Even though including heuristic optimizers substantially increases the computational load of a pipeline, they are still a go-to option for solving non-obvious and NP-hard optimization tasks. A wide span of existing MHAs allows us to find one that would suit the undertaken problem.

MHAs can be characterized by many aspects, with the origin of inspiration being the first one. Depending on chosen taxonomy MHAs can be divided into groups related to the main concepts they are built upon: biology, chemistry, physics, math, evolutionary processes, human behavior and finally, behavior of swarms [21,22].

Second feature that defines heuristic optimizers is associated with number of candidate solutions being used for problem solving. There are population based algorithms, like Genetic Algorithm, Differential Evolution, Particle Swarm Optimization [23], which maintain and upgrade over time a group of individuals. On the other side, there are single-solution algorithms, that include Simulated Annealing [24] or Tabu Search [25], which aim to iteratively improve a singular solution. This characteristic is extremely important when choosing an appropriate MHA, as the size of population can dramatically increase the computational cost of calculations.

Another basis for characterizing MHAs revolves around randomness [26]. There exist deterministic algorithms that always produce the same output for particular input setup. These include Tabu Search or Greedy Algorithms. Methods, that include randomness in the decision-making process are referred to as stochastic. They can be represented by Genetic Algorithm, Particle Swarm Optimization, Ant Colony Optimizer and many more.

Search space which is explored during calculations is another differentiating factor for MHAs. Algorithms like Simulated Annealing or Particle Swarm Optimization can operate on continuous spaces, whereas Genetic Algorithm or Ant Colony Optimizer move through a discrete one. The latter group is said to handle combinatorial optimization problems very well.

Even though there exist hundreds of heuristic optimizers, they are being mixed together, spanning hybrid versions of MHAs [27–29]. Such an approach takes the most interesting mechanisms from multiple algorithms and connects them to generate novel methods. As a consequence, such hybrids are often specialized in narrow areas, making them more difficult to be widely applied.

In this study, four well-known metaheuristic algorithms were selected to represent diverse paradigms of heuristic search. These include population-based (GA, PSO, ACO) and single-solution (SA) approaches, and they cover a range of inspiration sources, such as biology, physics, and swarm intelligence. Their inclusion provides a representative basis for evaluating LLM-guided tuning in various optimization scenarios.

2.2.1. Genetic algorithm

GA is the most popular example of bio-inspired computation. It draws from the phenomenal adaptation mechanisms of genes and techniques they developed to overcome their initial shortcomings [30–32]. GA uses operations of selection, mutation and crossover. These mechanisms ensure the diversity of the population and its evolution towards optimal settings.

This heuristic optimizer is an example of population-based, stochastic, discrete search space algorithm.

Selection strategy is key to determining which individuals in population will participate in reproduction, creating new generations, usually favors individuals with higher fitness, which increases the chance to passing on their traits of offspring. A common strategy is roulette-wheel selection, where the probability of selection is proportional to an individual's fitness.

Crossover and recombination are genetic operators used to combine the genetic information of two parents to generate new offspring. It is one of the key mechanisms for generating diversity within the population. Crossover is performed with a certain probability, called the crossover probability, and can take various forms, such as a single-point or two-point crossover.

Mutation introduces random alterations to the genetic material of the offspring, providing genetic diversity and enabling the algorithm to explore new areas of search space. The probability of a mutation controls how frequently these mutations occur. A typical mutation might involve flipping a bit in binary-encoded solution representation.

The number of generations defines the number of cycles the algorithm will run, with each cycle consisting of the selection of parents, crossover, and mutation to create a new generation. However, population size is the total number of individuals in the population. A larger population can explore more of the search space, but also requires more computational resources.

The overall GA process can be summarized to illustrate the cycle of operations:

- initialize a population of individuals randomly,
- evaluate the fitness of each individual in the population,
- repeat until termination condition is met (*e.g.,* number of generations reached):

- select parents from the population based on their fitness,
- crossover parents to create new offspring with a crossover probability P_C,
- mutate new offspring with mutation probability P_m,
- evaluate new offspring,
- replace the least fit individuals in the population with offspring,

• output the best found solution.

2.2.2. Ant Colony Optimizer

ACO implements the behavior of ant colonies [33–35]. Ants, treated as optimizing agents, have a natural ability to find the optimal (shortest) way to their desired destination. Ants communicate with each other through a pheromone that guide them along a path. ACO, due to the nature of its source, has found applications in routing problems and graph traversal tasks.

The key to the effectiveness of the ACO algorithm lies in the pheromone update rule, which is crucial for guiding the search process towards promising areas of the solution space:

$$\tau_{i,j}(t+1) = (1-\rho)\tau_{i,j}(t) + \triangle\tau_{i,j}, \tag{1}$$

where $\tau_{i,j}(t)$ is the pheromone level on path i, j at time t, ρ is a trail evaporation rate, and $\triangle\tau_{i,j}$ represents the amount of pheromone to be added, depending on the quality of the solution that the ants found using that path.

This rule mimics the way real ants deposit pheromones on the paths they traverse, leaving a chemical trail that influences the path choices of subsequent ants.

Ants in ACO algorithm make their path choices based on probabilistic decision rule. The probability that an ant will move from node i to node j depends on the amount of pheromone present on the connecting path ($\tau_{i,j}$), and possibly other heuristic information ($\eta_{i,j}$), such as the visibility or desirability of the next node:

$$P_{i,j} = \frac{\tau_{i,j}^{\alpha} \cdot \eta_{i,j}^{\beta}}{\sum_k \tau_{i,k}^{\alpha} \cdot \eta_{i,k}^{\beta}} \tag{2}$$

2.2.3. Particle Swarm Optimization

PSO is a robust and versatile optimization technique inspired by the social behavior of birds and fish [36–38]. This method belongs to the family of swarm intelligence algorithms and is widely used to solve global optimization problems in a continuous search space. PSO simulates the social dynamics of swarms, where each individual, referred to as a "particle," adjusts its flying trajectory based on both its own experience and that of its companions.

Each particle in the swarm symbolizes a potential solution to the optimization problem at hand. The particles are characterized by positions and velocities, which are dynamically adjusted on the basis of the best-known positions of the particle itself and of the entire swarm. The adjustment of velocities is governed by the velocity update rule, which incorporates the particle's current velocity, a cognitive component reflecting the particle's individual learning, and a social component derived from the swarm's collective knowledge. The formula for updating the velocity is given by the following:

$$v_i^{(t+1)} = \omega v_i^{(t)} + c_1 r_1 (p_i - x_i^{(t)}) + c_2 r_2 (g - x_i^{(t)}), \tag{3}$$

where $v_i^{(t)}$ is the velocity of particle i at time t, x_i is the position, p_i is the best known position of particle i, g is the best known position among all particles, ω represents the inertia weight that influences the degree to which the particle's previous velocity impacts its current one, c_1 and c_2 are acceleration coefficients that guide the particle towards its personal best and global best positions (learning factors), respectively, and r_1 and r_2 are random factors between 0 and 1 that introduce stochastic elements into the velocity update process.

Following the velocity updates, the positions of the particles are adjusted to reflect their new velocities:

$$x_i^{(t+1)} = x_i^{(t)} + v_i^{(t+1)} \tag{4}$$

2.2.4. Simulated Annealing

SA represents a group of physics-inspired metaheuristics. SA [24, 39,40], contrary to previous algorithms, is not a population-based algorithm. It is built upon a concept of controlled cooling of a material, taken directly from metallurgy. This method mimics the physical process in which a material is heated to a high temperature and then allowed to cool slowly, which helps in reducing defects and achieving a stronger structure. SA is often used in combinatorial optimization problems, including TSP [41]. This optimization technique is used to find a good approximation of the global minimum of a given function in a large search space.

The core decision rule of SA is that a new solution that decreases the energy of the system is always accepted. However, if the new solution increases the energy, it may still be accepted with a probability defined by the Boltzmann distribution:

$$P(e, e', T) = \exp \frac{-(e' - e)}{T}, \tag{5}$$

where e is the energy of the current solution, e' is the energy of the new solution, and T is the current temperature.

2.3. Combinatorial optimization problems

Combinatorial Optimization constitutes a special area at the intersection of mathematics and computer science. It is defined as finding the optimal solution from finite set of possible solutions. It can be reformulated as searching for the best entity in a group of feasible entities. As a consequence of such definition, heuristic optimization and search algorithms are a popular choice for handling combinatorial optimization [42]. Other approaches include linear programming, dynamic programming, approximation or generating search boundaries [43].

The variety of combinatorial problems and amount of interest devoted to solving them, explain the importance of this area of research. Most popular problems include but are not limited to: Knapsack Problem, Vehicle Routing Problem, Minimum Spanning Tree, Graph Coloring, Traveling Salesman and Job-Shop Scheduling Problems, which are deeply analyzed in this paper.

Aforementioned problems are nothing other than the formulations of real-life tasks, hence the wide applicability of them. Combinatorial Optimization is present in the fields of operations research, logistics, scheduling and many more.

In this study, we selected three combinatorial optimization problems: Graph Coloring (GCP), Job-Shop Scheduling (JSSP), and the Traveling Salesman Problem (TSP), which differ significantly in structure, representation, and search space characteristics. This diversity provides a robust testing ground for evaluating the effectiveness and generalization capabilities of LLM-driven metaheuristic parameter tuning strategies.

2.3.1. Graph Coloring Problem

GCP originates from the task of coloring the map in such a manner that adjacent countries (or states) do not have the same color. Later, it was redefined to match the graph representation [44]. In this manner, Graph Coloring (often called Vertex Coloring) becomes a task of assigning colors to the vertices with the possibly lowest number of colors. The adjacency constraint is still present in this task. Considering an undirected graph G, where V denotes its vertices and E represents the edges:

$$G = (V, E) \tag{6}$$

The aim of GCP is to assign a coloring to each V, so that the adjacent vertices connected through E are distinctively colored.

The minimal amount of distinct colors resulting from solving the GCP is called a chromatic number $\chi(G)$ and is one of the characteristics of the graph. The complexity of this task depends on the graph structure itself. Presence of cliques, cycles, triangles or wheel architecture can postpone the convergence to optimal solution. In popular benchmark instances of GCP there exists a set of graphs based on Mycielski transformation. These are said to be difficult to solve [45].

Among applications in the field of graph theory, solving GCP is present in real life as well. It can be utilized in the process of designing the surveillance systems, aircraft scheduling, timetabling and assigning frequencies in telecommunication networks [46]. Graph Coloring is an NP-complete problem.

2.3.2. Job-Shop Scheduling Problem

JSSP represents the task of assigning n jobs to m machines [47]. Each job has a different completion time and each machine has its own processing capabilities. This task aims to find the best scheduling sequence that minimizes the overall time required to run all procedures. JSSP can be described using the disjunctive graph model [48], where M and J are finite sets and represent machines and jobs respectively.

$$\begin{aligned} M &= \{M_1, M_2, \dots, M_m\} \\ J &= \{J_1, J_2, \dots, J_n\} \end{aligned} \tag{7}$$

Let χ denote all possible sets of job to machine assignments. Single solution $x \in \chi$ can be written as a matrix of size $n \times m$. In such notation columns represent consecutive machines M_i and rows correspond to jobs, stating the order of their execution. Total processing time, which is the cost function C in such problem definition is expressed as:

$$\begin{aligned} C &: \chi \to [0, +\infty] \\ C_{ij} &: M \times J \to [0, +\infty] \end{aligned} \tag{8}$$

The task in Job-Shop Scheduling Problem is to find such x that yields minimum $C(x)$, meaning that processing time of jobs J on set of machines M is as minimal.

JSSP popularity is caused by its wide applicability to production and manufacturing. Existence of wide range of variations of this task, shows how universal and important it is for the research as well as for the industry. JSSP also constitutes an NP-hard problem. Interestingly, Traveling Salesman Problem is a special instance of Job-Shop Scheduling, having single job and multiple machines (cities).

2.3.3. Traveling Salesman Problem

TSP [49] is a classic optimization problem defined as finding the minimal route between n cities. The data structure for this exercise can be represented as a matrix of distances between these cities or as a list of (x, y) coordinates for each of them. Regardless of representation, the algorithm has to find the order of the cities that yields the shortest route.

TSP can be formulated with:

- nodes (cities): $N = \{1, 2, \dots, n\}$;
- distance matrix: $D = [d_{ij}]_{n \times n}$, where d_{ij} corresponds to the distance between cities i and j, and it is a $n \times n$ matrix;
- decision variables: x_{ij}, where value equals 1 if path between cities i and j is a part of the solution and 0 in opposite scenario;
- objective function: $\min \sum_{i=1}^{n} \sum_{i=j}^{n} d_{ij} x_{ij}$, which minimizes the total traveled distance.

There exist many variations of TSP; they include adding constraints such as a particular starting point or the requirement to return to the starting point. Numerous versions of the classic problem try to adapt it to fit more real-life applications, resulting in kinetic-based, profit-based, symmetrical, or asymmetrical formulations of TSP [50].

The data representation for this problem can take two forms. The first one, more popular due to real-life applications, is a list of (x, y) coordinates for each city within the problem scope. The second representation has a matrix structure containing distances between cities. Regardless of the input data configuration, the TSP solution is a vector of city identifiers, which corresponds to the order of places on the minimal route.

Complication of TSP is dependent on a number of cities it's defined on. In terms of complexity, original TSP is an NP-hard problem.

3. Literature review

This section summarizes the most relevant research related to interaction between LLMs and metaheuristic optimization methods. It is divided into two parts: the first one discusses studies combining LLMs with heuristic and evolutionary algorithms, while the second reviews classical parameter tuning approaches, which form the conceptual foundation for this work.

3.1. General overview

Large Language Models and Heuristic Optimizers can interact with each other in two directions. The first research direction focuses on employing metaheuristics to enhance or optimize LLMs themselves, while the second investigates the potential of LLMs as intelligent agents supporting or even replacing traditional heuristic optimization processes. These paths are not equally explored, as due to LLMs' popularity, more interest is devoted to the field of optimizing LLMs with the employment of wide span of techniques, including MHAs.

3.1.1. Metaheuristics applied to LLMs

The first way involves optimizing LLMs. Such a route is more apparent in the literature, thus addressing the demand associated with LLMs' popularity. In this scenario, Evolutionary Algorithms (EA) can be utilized in the optimization of prompt formulation [51] or to boost the prompt learning process [52].

Another idea for re-purposing bio-inspired computation is presented in [53], where the proposed framework resembles a genetic algorithm. Such an approach is used to generate a population of candidate prompts that are further evolved to trigger the desired response from the model.

These approaches demonstrate that metaheuristics can serve as effective external controllers or search mechanisms for LLM performance enhancement.

3.1.2. LLMs applied to metaheuristics

An alternative and increasingly active research stream reverses this relationship, in this case Large Language Models can be used to assist or automate metaheuristic optimization process, for instance by generating populations of candidate solutions. This approach is present in LEO [54]. This LLM-Based Evolutionary Optimizer employs reasoning abilities of LLMs to generate two pools of candidate solutions. This dual-pool design aims to balance exploration and exploitation of the optimization. However, such framework has to be used cautiously, due to language models' tendency to hallucinate.

The concept of LLM-driven Evolutionary Algorithms (LMEA) is also present in [55]. In this work, LLM is embedded directly in the MHA and is tasked with performing selection and mutation. It is another example of incorporating language and data comprehension into the evolutionary process.

Language Hyper-Heuristics (LHHs), introduced in [56], use Reflective Evolution mechanism (ReEvo). In this method GPT-3.5 Turbo, GPT-4 Turbo and Llama 3 are used to verbalize knowledge and suggestions, called reflections, for further iterations. Based on LLM's output a population of heuristics is altered to achieve greater performance in solving combinatorial optimization problems. LHHs are compared with another notable method - Evolution of Heuristics (EoH) [57], which employs LLMs to translate concepts of heuristics formulated in natural language into runnable code.

Comparison of LLM-based methodology for hyperparameter optimization (HPO) with standard techniques like Bayesian optimization

and random search can be found in [58]. Similar approach to utilizing LLMs as parameter advisors and incorporation of feedback rounds is present in [59,60]. Although, this study is preliminary it sheds light on the new approach to parameter tuning. These works follow a very similar methodology to the original paper used as basis for this study [8]. Large Language Models have also found application in task of Neural Architecture Search, which is a narrow optimization problem of finding best neural network architecture. EvoPrompting [53] involves LLMs in adaptive crossover and mutation mechanisms.

Other methods can include utilizing LLMs for suggesting parameters based on analysis of optimization logs [61]. This study evaluated Llama2-70b and Mixtral in optimizing parameters in Evolution Strategies.

A thorough summary of interactions between LLMs and heuristic optimizers can be found in [62,63]. It underlines that this intersection of techniques is a compelling, and promising area of research, and the scarcity of methods and unresolved issues justifies a need for additional investigation.

The intersection of these two concepts provides an abundance of possible applications. The alternative direction of research is devoted to the use of LLMs as systems that can design novel metaheuristics [64].

3.1.3. LLMs for heuristic design and generation

A more recent and ambitious research direction investigates the ability of LLMs to autonomously design new heuristic algorithms. LLaMEA [65] is an interesting example, where a Large Language Model through the evolutionary process generates novel heuristic optimizers. MHAs designed with this framework are reported to outperform classical metaheuristics. Another interesting work presents the idea of Algorithm Evolution using a Large Language Model (AEL) [66]. This paper delves into the generation of new optimization algorithms through the evolutionary framework and evaluates them on a Traveling Salesman Problem. Guided Evolution technique [67] explores the ability of LLMs to directly alter the code through the self-sustaining feedback loop.

Optimizing by Prompting (OPRO) [68] is a framework from Google DeepMind. This research exploits natural language to generate new solutions, step by step. The prompts used in this approach contain extensive details about the tasks and the data along with corresponding fitness values from previous steps. Authors claim, that the goal of this research was not to outperform state-of-the-art methods, but to showcase LLMs' optimizing abilities.

3.2. Classical parameter tuning approaches

Apart from advanced tuning techniques involving LLMs and Evolutionary Computation, there exists a plethora of frameworks that one can follow in order to tune an algorithm's parameters. This idea is strongly present in the field of AutoML, where it becomes important to reduce human input (and bias) in the optimization process. Representative approaches include Deep Belief Networks, Sequential Model-based Global Optimization, and Gaussian Process methods, among others [69]. The following paragraphs briefly summarize the most influential and widely used strategies.

3.2.1. Grid and random search

A standard, and one of the simplest yet successful techniques is grid search, which systematically explores predefined combinations of parameter values. There exist numerous studies that dwell on application of this strategy in various problems [70–72]. However, computational cost of grid search increases exponentially with the number of parameters, a phenomenon commonly referred to as the curse of dimensionality [73].

A successor and improvement of search performed on a grid, Random Search [74], is a step forward. The biggest contribution and advancement introduced by this method is the reduction of human input. Since there is no grid, there is no need to design it or choose the appropriate values to fill in the mesh of possible value combinations. Random search is reported to achieve comparable results to hand-drafted Deep Belief Networks and outperforms grid search given proportionate computational resources.

Automated parameter optimization can be achieved by leveraging grid and random search together, in an iterative process as presented in [75].

3.3. Bayesian and gradient-based optimization

Bayesian Optimization and its improved versions utilize the concept of statistical modeling of the objective function and sampling. It is suited for continuous noisy problems. Its performance drops with increasing dimensionality of the problem, hence it is best suited for domains smaller than 20 dimensions [76]. Many improvements aim to address mentioned shortcoming, as in REMBO, through utilization of efficient embeddings [77].

Gradient calculations allow to streamline solving of many computational tasks. They can also be applied in the field of parameter optimization. These techniques are based on calculating parameter gradients. This can be achieved by reversing the dynamics of Stochastic Gradient Descent as in [78]. Forward and reverse propagation of such gradients and their impact on parameter tuning process are examined in [79].

Such methods provide high efficiency when gradient information is available but are less suitable for non-differentiable metaheuristics.

3.3.1. Domain-specific and hybrid methods

Optimization or tuning of MHAs constitutes an important task, since the impact of parameter values is irrevocable. Usually, such a tuning is performed manually and can be guided by expert knowledge. However, this approach is not scalable and does not align with trend for algorithm automation. Motivation for finding solutions that can overcome mentioned shortcomings results in development for many techniques of parameter tuning applied to MHAs.

For instance, combination of data envelopment analysis and response surface methodology yields a hybrid method called DSM. It is evaluated using Cuckoo Optimization Algorithm. Achieved results can compete with other approaches in terms of efficiency, computation time, and accuracy [80].

Gravitational Search Algorithm (GSA) was subject of research that aimed to establish a generic tuning framework [81]. Idea present in this paper exploits the knowledge about topological nature of the undertaken problem. Although, it tested performance of GSA on continuous CEC 2015 function suite, authors claim that presented idea can be reapplied to other heuristic optimizers regardless of the problem domain.

Autonomous parameter tuning can be accomplished with the use of racing based learning module [82]. Racing in such setting means incremental assessment of candidate solutions (in this case a solution is a set of parameter values). Important part of such an approach involves exclusion of worse performing individuals, as soon as enough evidence is collected to support such a claim. This technique allows for reducing equally distributed importance in brute force approaches, hence steering the search in more promising directions by ruling out weak sets of parameter values. Detailed classifications and descriptions of other notable parameter tuning strategies are present in [83], which provide an extensive taxonomy of existing approaches and their comparative performance across diverse optimization domains.

4. Proposed methodology

This study aims to investigate the feasibility of tuning the parameters of metaheuristic algorithms using state-of-the art LLMs in the context of combinatorial optimization. The research process consists of several stages, which are described in detail in the attached diagram (Fig. 1) and include the following steps:

1. Determining optimization problems - the analysis was performed for three optimization problems (GCP, JSSP, TSP), for which ten benchmark instances were selected, thus creating 30 unique test scenarios.
2. Selecting most suitable heuristic optimizers for solving chosen problems - LLMs were prompted to identify the most suitable metaheuristic algorithm for each problem. This was followed by majority voting to determine top three MHAs per task. LLMs were consistent with their selection - candidate pool for voting could be reduced to GA, SA, ACO, and PSO.
3. Querying for parameters - for each problem-instance-optimizer triplet, various LLMs such as Claude, GPT-4o, Gemini, and Mistral, in addition to a default configuration, were prompted to recommend parameter values.
4. Solving the problem using MHAs - three selected metaheuristic algorithms from the pool of: GA, SA, ACO, and PSO were used to address the problem with the suggested parameters; each configuration setup was independently run 30 times to ensure statistical reliability.
5. Results analysis and feedback - once the problem was solved using the initially suggested parameters, results were shared back with the language models for feedback; the models could suggest changes to the parameters based on the analysis of the outcomes, enabling further refinement of the strategies. LLMs could propose adjustments to the parameters, forming a single feedback iteration. Multiple feedback loops were intentionally avoided, as repeated LLM querying and rerunning the optimization would substantially increase computational and time costs.
6. Data collection and analysis - collected data was analyzed to evaluate the convergence and effectiveness of the various metaheuristic strategies in the context of the adjusted parameters.

LLMs used based this study were selected in on their the ELO rating [84] and the geographical availability. Four LLMs were chosen for the experiments: Google's Gemini, GPT-4o by OpenAI, Claude from Anthropic, and Mistral Large by Mistral AI. These models are well-established in the community, their data sources are considered transparent and ethical, and finally they are available via APIs, which makes multiple prompting much easier. The fact that these models are primarily trained with English data is also important, as scientific papers, which are the basis for the reasoning in this study, are written in English.

It has to be mentioned that these LLMs were not fine-tuned or adjusted to fit the optimization-oriented domain. Taking these systems off the shelf facilitates comparison of the models in a form, that fully presents differences and their abilities to effectively guide parameter tuning process.

Additionally, there exist scenarios where the Large Language Model could not decide on one exact value. In cases where it suggests a range of values, the middle one is chosen. What is more, in situations where LLM cannot decide on any value, the default one is taken.

The single round of feedback collected from LLMs was selected over contentious parameter refinement in a loop. Introducing multiple feedback iterations would lead to a substantial increase in computational and time costs, since each iteration requires additional LLM queries and repeated execution of the optimization process. Automating such loop-based prompting in a stable and reproducible way is also non-trivial, as model responses can vary between runs and require careful state management. Therefore, this study concentrates on evaluating the LLMs' ability to provide meaningful one-shot reasoning rather than performing a complete multi-stage optimization loop.

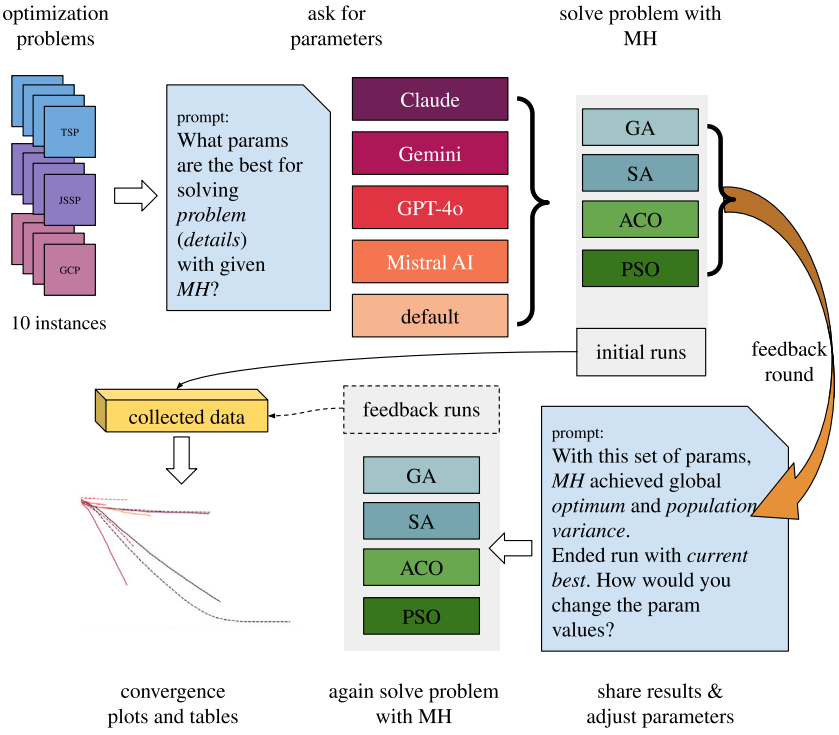


Fig. 1. Experimental pipeline. Prompts are simplified, the exact formulations can be found in Table 1. The framework starts with defining instances of optimization problems to be solved. In this case TSP, JSSP and GCP are available. In the next step, four LLMs are asked for best parameters for a *instance - MHA* pairs. Fifth set of parameters includes default values from mealpy library. Subsequently, problems are solved with the use of three out of GA, SA, ACO, PSO heuristic optimizers. These solutions constitute a *initial* round. Next, the LLMs are prompted for their feedback based on achieved performance. LLMs can suggest changes to the parameter values and calculations are run again - giving the *feedback* round results. There is no feedback round for default set of parameters. In the end, initial and feedback results are analyzed together and visualized as convergence plots.

Table 1
Exemplary prompts used in experiments. Text in italics is adjusted for each setup.

Type	Prompt
Selection	What are the best heuristic optimizer for solving *Graph Coloring Problem?*
Initial	I'm using mealpy library to solve *Job-Shop Scheduling Problem*. My problem instance has *20 jobs and 5 machines*. I want to use *Particle Swarm Optimization* to solve it. Mealpy's implementation of *PSO* takes following parameters: *epoch (integer smaller than 100000), pop_size (integer), c1 (float), c2 (float), w (float)*. In the brackets - I wrote possible parameter values or their type. Suggest me parameter values in this problem setup. Give exact values and present them in table.
Feedback	I'm solving a *TSP* of *29 cities* with the use of *GA*. I used parameter values previously suggested by you: *epoch = 10000, pop_size = 300, pc = 0.8, pm = 0.02, selection = roulette, crossover = uniform, mutation_multipoints = False, mutation = flip*. I did run it 30 times and averaged the results. I got global minimum of *16,168.2305* with std of *442.818*. I also measured variance of the population at the beginning and last epoch: *27.9753* and *27.38019* correspondingly, with std of *0.007* and *0.146*. The solution at last epoch had average fitness of *20,520.809* with std of *790.537*. What changes to the parameters would you suggest to improve performance? Keep the epoch*pop_size constant. Give exact values and present them in table.

4.1. Prompts

Selection of Metaheuristic Optimizers suited for given optimization problem was performed with guidance of Large Language Models. Initially, LLMs were asked for the most suitable MHAs for solving particular problem.

Once the triplets of *problem-instance-MHA* were established LLMs were fed with prompts called *initial*. These prompts included general information about problem instance: number of nodes and edges for GCP, quantity of jobs and machines in JSSP and number of cities in case of TSP. In addition, the prompts contained details of available parameters, their names, possible values and expected types.

It can be observed that the initial cue for an LLM contains only general information about the problem instance rather than a full formulation with actual data. Problems were generalized to numbers that dictate their complexity. Such an approach encourages LLMs to use their reasoning skills rather than finding *overfitted* parameter values directly associated with the provided data. The latter technique constitutes a different problem.

Prompts used to establish parameters for feedback runs contained additional information about achieved performance in initial round, expressed as average global minimum, average fitness at last epoch and corresponding standard deviations. In addition, for population based algorithms, variance of population at the first and last epoch along with standard deviations was included. Table 1 presents the exemplary prompts used in this study.

It is worth mentioning that an additional constraint was included in the feedback prompt. To minimize the chance of getting the obvious response of increasing the number of epochs and/or population size, it is required to keep the number of fitness function evaluations constant. This fosters fairness in comparison between runs and maintains the cost of calculations.

5. Experimentation

This section describes the experimental setup used in this study, including benchmark datasets, implementation details and evaluation metrics to provide a transparent and reproducible procedure for assessing the accuracy and effectiveness of LLM-based parameter tuning.

5.1. Data

To ensure reproducibility and focus on the methodology rather than the problems themselves, this study uses well-established benchmark instances. The use of publicly available data supports transparency and facilitates comparison with future work.

In the field of combinatorial optimization, OR-Library [85] is a scientifically acclaimed repository of test problems for a wide range of tasks. It gathers and references data from important research works. Another valuable source of benchmarks is called a TSPLIB [86]. It is a similar collection of problems, but strictly devoted to Traveling Salesman Problem.

In this paper, we selected 10 benchmark instances for each of the three problems: GCP, JSSP, and TSP, resulting in a total of 30 unique test cases. Graph Coloring tasks are represented by instances of Mycielski's (*myciel*) and *queen* graphs. JSSP is featured by instances gathered from variety of scientific works, including: *abz, ft, la* and *orb* examples. Finally, TSP is exemplified with problems such as *ulysses, chn, vrp* or *period*. The notation used in graphs further in this work follows convention of *problem - MHA - instance*. A detailed description of the selected instances is provided in Appendix A.

To ensure reproducibility, problem instances as well as the code used in the development of this project are available in the github repository found under https://github.com/alicjanna/LLMs_as_parameter_tuners.

5.2. Experimental details

All calculations were performed in Python. The important decision regarding the implementation of MHAs was also driven by LLMs. It was determined that all four models of our interest are aware of Mealpy library [87]. Mealpy is an open-source Python library that gathers implementations of a vast collection of metaheuristics and their variations. As it is a well-known and mature project, LLMs were most likely trained on Mealpy's documentation as a part of their training corpora. This not only reduces the risk of ambiguity associated with individual implementations of MHAs but also allows us to refer to Mealpy directly in the prompts.

Tuning process of parameters included all parameters available in the Mealpy's implementation of heuristic optimizers used in this study. The MHAs and their corresponding parameter names are listed below:

- Ant Colony Optimizer - epoch, pop_size, sample_count, intent_factor, zeta;
- Genetic Algorithm - epoch, pop_size, pc (crossover probability), pm (mutation probability), selection, crossover, mutation, mutation_multipoints;
- Particle Swarm Optimization - epoch, pop_size, c1 (cognitive coefficient), c2 (social coefficient), w (inertia weight);
- Simulated Annealing - epoch, temp_init, step_size.

Each parameter setup was run 30 times. It has to be pointed out that for Initial and Feedback runs, there are 900 unique *triplet* setups in total. The computational cost of these calculations was maintained by an upper limit on number of epochs implemented in Mealpy.

5.3. Evaluation metrics

Assessment of the results obtained requires special measures. Experimental settings and design of these tests impose the employment of following metrics:

- average total runtime;
- average objective value (in Mealpy called *global_best*);
- average L1 distance - measures variance in population, where applicable. L1 was utilized to derive the information about candidate solutions. As the representation of each individual is a vector, they can be compared in pairwise manner according to the formula:

$$L1(A, B) = \sum_{i=1}^{n} \delta(a_i, b_i), \tag{9}$$

where a_i and b_i are *i-th* elements of vectors A and B and δ represents a binary indicator function that equals 0 when $a_i = b_i$ and 1 otherwise (corresponds to the discrete L1 distance);

- count of times when Objective Value of LLM-guided setup achieved better performance than default or grid search-guided (comparison) setups. It is expressed as:

$$LLM_{count} = \sum_{i=1}^{30} OV(epoch_{max})_{LLM} > OV(epoch_{max})_{comp}, \tag{10}$$

where OV is the Objective Value and $epoch_{max}$ refers to the last epoch for given run.

Results for both rounds, Initial and Feedback, are averaged over 30 runs for each. This ensures statistical significance and reduces the risk of drawing conclusions based on random outcomes.

6. Results

The experimental findings demonstrate that Large Language Models can efficiently guide the tuning process for combinatorial problems. LLM-based runs of MHAs can exhibit superior performance compared to default runs and present the ability to improve their achievements with just one iteration of feedback.

Fig. 2 presents convergence plots for representative instances across all tested problems and MHAs. Each subplot illustrates how objective values evolve over the course of optimization. Due to the repetitive convergence patterns observed across instances, only one representative plot per problem–MHA combination is included. To facilitate comparison, convergence curves are differentiated by line style and color: solid lines correspond to the Initial tuning phase, dashed lines represent Feedback-based refinement, whereas the default configuration (used only in the Initial phase) is depicted as a distinct solid line. Colors are used to distinguish between LLMs and the default setup.

It is advised to look at the plot in terms of default performance compared against LLMs. The second angle of analysis includes comparison of lines within the same color. Ideally, a faster convergence of dashed lines would be expected.

For visualization clarity, most plots were truncated at 1000 or 3000 generations on the x-axis, even though runs were usually scheduled for up to 10,000 generations. This adjustment was made because the majority of convergence dynamics occur early in the optimization process, and it allows better visual contrast between different configurations.

6.1. LLMs' awareness of benchmark problems

At the beginning of the process of experiment conceptualization and pipeline design, the idea to test LLMs' awareness of benchmark instances was explored. This concept was later abandoned as the models hallucinated about the problems and their instances or presented just general knowledge, that was meaningless and did not imply an understanding of particular benchmarks. On the other hand, given that the potential awareness of problems and their corresponding data would not foster an environment to conclude reasoning skills of LLMs, but rather just test their training sets. It should be mentioned that, with the dramatically rapid development of language models, they could soon potentially extend their awareness of benchmarks and reduce the effect of hallucinations.

6.2. Model's laziness

The number of cities in selected Traveling Salesman Problem instances was not distributed evenly. An interesting observation was made during the prompting stage of the experiment. It appears that Large Language Models often fall into laziness and can suggest useless values. It was discovered via prompting models in two ways. First would attempt to ask for all parameters at once. The prompt contained information about all problem instances at the same time. In this way, LLM answered with a table where generations and population sizes would increase linearly. It would be expected not to increase dramatically the computational cost between problems defined on 16, 20 and 22 cities, yet Large Language Models fell into this trap. However, prompting about each problem instance separately mitigates this behavior of LLMs. It is essential to keep the number of fitness function evaluations as low as possible since it is a driver of computational cost. A valuable lesson learned from this is to always write separate prompts, as models tend to overestimate the numbers of generations and population sizes when asked for multiple parameters at once.

6.3. Comparison with default runs

Tables 2 and 3 present aggregated statistics for evaluating LLM-based runs against default ones. These tables follow convention of counting how many times given setup was better (+), worse (−) or there was no significant difference (0) between runs. This was performed with a t-student test for comparing distributions of Objective Values at particular epoch. Each vector had 30 elements, because each setup was run this many times. Counts within the problem-MHA pair add up to 10, as it is the number of problem instances solved in such a setting.

The key difference between Tables 2 and 3 is in the evaluation point. The first table shows results at the last epoch for LLM and default runs, which is not always a fair way of comparing results. Second table, inspired by looking at convergence plots, contrasts Objective Values at $\min(\max(epoch_{LLM}), \max(epoch_{default}))$. In this way, it is possible to grasp the convergence dynamics regardless of the number of epochs (generations).

Intuitively, a *fair* comparison increases LLM performance. Default setups are scheduled to run for 10,000 epochs, which usually exceed the epoch value for LLM-advised parameters. In addition, the number of nonsignificant differences rose from 42 to 195 - which implies that the convergence is aligned together.

6.4. Graph Coloring Problem

Fig. 2 shows convergence dynamics for 3 problem instances solved with different metaheuristics.

In the case of GA, almost all LLMs achieved rapid convergence in the early stages of optimization, outperforming the default configuration in terms of speed. However, this advantage did not translate into better final results. The default configuration ultimately reached a lower objective value than any of the LLM-guided runs. Only Gemini marginally improved its outcome in the Feedback round, while Claude and GPT-4o exhibited slightly worse performance after refinement.

For PSO – myciel6, GPT-4o stands out as the only model that produced a configuration outperforming the default after the Feedback round. The remaining LLMs either matched or underperformed relative to the baseline, particularly in the Feedback stage. Gemini in that case proposed small number of evaluations (epoch * pop_size) making it hard to count as promising solution. However, it achieved a huge improvement with feedback round.

The most consistent and positive results were observed in the SA–myciel4 scenario. All LLMs (except Gemini after feedback) managed to produce significantly better configurations than the default setup, both in terms of convergence speed and final objective value. Feedback refinement led to even better results in most cases, with Claude and GPT-4o achieving near-optimal solutions.

SA appears to benefit most from this approach, whereas GA and PSO reveal potential limitations of generic prompting and suggest the need for more adaptive or instance-specific strategies. Table 3 supports this hypothesis with 31 setups performing significantly better than default, which exceeds 3 setups for both, GA and PSO.

6.5. Job-Shop Scheduling Problem

In the ACO–la10 instance, GPR-4o achieved substantial improvements over the default configuration. The Feedback round further

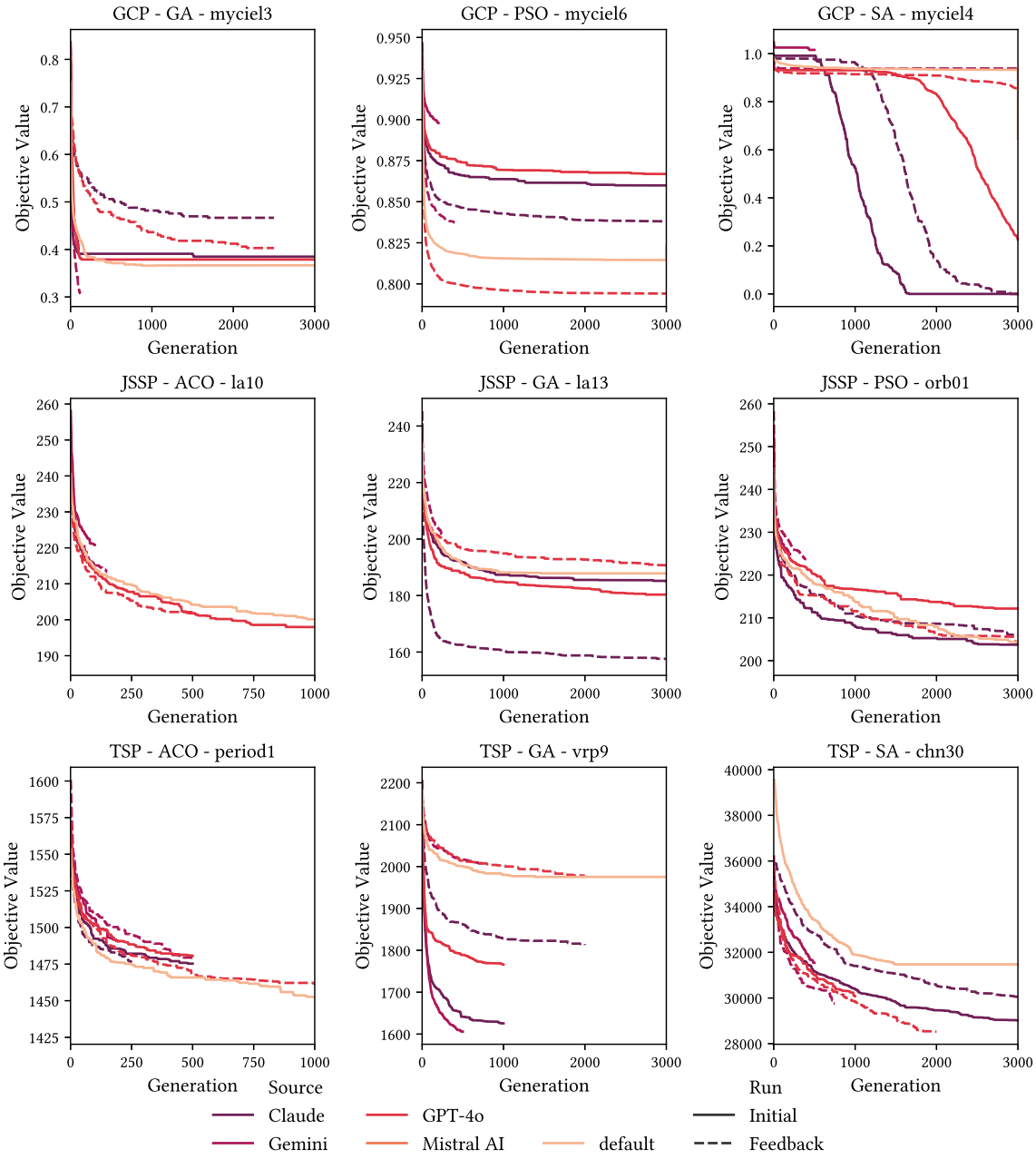


Fig. 2. Selected convergence plots for proposed procedure. These graphs present the dynamics of convergence to the global optimum. Colors denote source of parameter values (either a particular LLM or a default set) while line style refers to the run type - with solid line meaning initial run and dashed line presenting feedback. Each row of graphs depicts different optimization problem (GCP, JSSP, TSP). Within such row there is a one plot for each Heuristic Optimizer (ACO, GA, PSO, SA - displayed in alphabetical order) used to solve these. Instances were chosen randomly. Ideally, dashed lines should exhibit faster convergence than their solid counterparts.

enhanced performance, with Claude and GPT-4o achieving the lowest final objective values.

The GA – la13 highlighted the superior performance of Claude, whose Feedback configuration outperformed all others and reached a final objective value significantly lower than the default setup. While GPT-4o and Gemini models also produced better-than-default results, the benefits of the Feedback round were limited to Claude, suggesting a model-specific ability to refine GA's more complex configuration space.

PSO – orb01 - although LLMs enabled faster early convergence than the default configuration, final objective values converged to a similar range across all methods. Notably, Gemini and Claude experienced performance degradation in the Feedback round, implying that further tuning disrupted the optimizer's balance. PSO's behavior in JSSP is particularly sensitive to over-tuning and may benefit from more conservative feedback strategies.

Genetic Algorithm seems to be reacting most positively to LLM suggestions. Table 3 shows that 21 setups performed better than default. Interestingly, feedback round introduced increased performance only in the case of PSO and Mistral pair, with change from 1 to 2 in the counts.

6.6. *Traveling Salesman Problem*

The results for the TSP demonstrate the strong capability of LLMs to improve both convergence dynamics and final solution quality for GA and SA. In those two instances the LLM-guided parameter setups significantly outperformed the default configurations. Fair assessment

Table 2
Comparison of LLMs against default runs at last epoch. + indicates how many times LLM solution was better than default, 0 denotes no significant differences between 30 runs, and - refers to runs worse than default runs. Statistical significance was established using a t-student test.

			Claude		Gemini		GPT-4o		Mistral		Grand Total
			Initial	Feedback	Initial	Feedback	Initial	Feedback	Initial	Feedback	
GCP	GA	+				1					1
		0							1		1
		–	10	10	10	9	10	10	9	10	78
	PSO	+		1				2			3
		0		2		1				2	5
		–	10	7	10	9	10	8	10	8	72
	SA	+	5	4	1	1	8	7	4	3	33
		0		1		3				2	6
		–	5	5	9	6	2	3	6	5	41
JSSP	ACO	+									
		0									
		–	10	10	10	10	10	10	10	10	80
	GA	+	4	1			4	1	5	2	17
		0	2	2	1		2	1	1		9
		–	4	7	9	10	4	8	4	8	54
	PSO	+					1	1			2
		0	1	1							2
		–	9	9	10	10	9	9	10	10	76
TSP	ACO	+									
		0									
		–	10	10	10	10	10	10	10	10	80
	GA	+	6	4	3		6		4	2	25
		0	1		1			3	2	1	8
		–	3	6	6	10	4	7	4	7	47
	SA	+	7	3	4	6	6	6	6	7	45
		0		4	3	1	1	1	1		11
		–	3	3	3	3	3	3	3	3	24
	Total	+	22	13	8	8	25	17	19	14	126
	Total	0	4	10	5	5	3	5	5	5	42
	Total	–	64	67	77	77	62	68	66	71	552

of ACO is impossible, due to the limited number of epochs for LLM-guided runs. Looking at convergence curves for TSP - ACO - period1 is not definitive, as *shorter* runs could potentially outperform the default one.

Traveling Salesman Problem seems to be the most explored one, with regards to literature and attention, however this is not reflected by improvements in feedback runs. Only Gemini and GPT-4o scored better Objective Value in TSP - SA - chn30 triplet.

When analyzing Table 3 for TSP the Simulated Annealing achieves 69 counts globally. It is the highest result among all problem-MHA pairs. Result for TSP-GA of 28 is the second best. This highlights the meaning of academic presence of problems and heuristic optimizers on the *knowledge* of Large Language Models. Importantly, ACO is not underperforming compared to default runs - with 56 counts in the nonsignificant difference column.

6.7. Comparison with grid and random search procedures

In order to further evaluate LLMs' applicability as tuning agents additional experiments with grid search and random search procedures were conducted. Again, for compatibility, Mealpy's implementations of both were utilized. In this scenario, only limited amount of setups was analyzed - one *problem-instance-MHA* triplet for every problem. Such selection covers all tasks and heuristic optimizers. The selected triplets are:

- GCP - SA - myciel3,
- JSSP - PSO - abz5,
- TSP - GA - burma14.

Grids of possible parameter values were established with regard to the bounds present in LLM-based runs. Each obtained grid search setup was run 5 times and the best one was preserved to conduct the full calculation process (30 runs) to ensure fair comparability with other results. Mealpy's implementation of random search can repurpose these grids in its calculations.

Graph Coloring Problem instance *myciel3* was selected to be solved by Simulated Annealing. In a search process it was determined that temp_init = 500 and step_size = 0.95 yield best results. Grid search process for Traveling Salesman Problem, Genetic Algorithm and *burma14* instance resulted in selection of pop_size = 150, pc = 0.85 and pm = 0.2, whereas Job-Shop Scheduling Problem PSO for *abz5* instance gave the best results with pop_size = 100, c1 = 1.75, c2 = 1.5, and w = 0.7.

Random search procedure selected different parameter values for all 3 tested setups. *myciel3* had temp_init = 10 and step_size = 0.95, *burma14* ended with pop_size = 100, pc = 0.9 and pm = 0.1. PSO parameters selected for *abz5* were pop_size = 100, c1 = 2, c2 = 2.5, and w = 0.6.

Although, analysis of Table 4 shows the superiority of grid search and random search over LLM runs the computational cost has to be taken into consideration. Figures depicting average convergence dynamics present counter-intuitive (not supported by tabular results) outcomes. Looking at these plots one can analyze them in two separate manners. First covers contrasting searches with default runs (blue and green versus orange lines), whereas second includes comparisons with runs based on parameters suggested by LLMs.

Table 3
Comparison of LLMs against default runs at *min(max(epoch$_{LLM}$), max(epoch$_{default}$))*. + indicates how many times LLM solution was better than default, 0 denotes no significant differences between 30 runs, and − refers to runs worse than default runs. Statistical significance was established with a t-student test.

			Claude		Gemini		GPT-4o		Mistral		Grand Total
			Initial	Feedback	Initial	Feedback	Initial	Feedback	Initial	Feedback	
GCP	GA	+				2				1	3
		0		1	1				1		3
		−	10	9	9	8	10	10	9	9	74
	PSO	+		1		1		1			3
		0	2	4	1		2	3		2	14
		−	8	54	9	9	8	6	10	8	63
	SA	+	5	4	1	1	8	6	3	3	31
		0		1		3		1	1	2	8
		−	5	5	9	6	2	3	6	5	41
JSSP	ACO	+				1	1				2
		0	8	10	3	4	5	6	6	5	47
		−	2		7	5	4	4	4	5	31
	GA	+	5	3			4	1	6	2	21
		0	1	2	6	1	3	1		1	15
		−	4	5	4	9	3	8	4	7	44
	PSO	+	1				1	1	1	2	6
		0	7	3	4	1	2	5	3	6	31
		−	2	7	6	9	7	4	6	2	43
TSP	ACO	+		1				1			2
		0	9	9	5	5	7	8	7	6	56
		−	1		5	5	3	1	3	4	22
	GA	+	6	4	4		6		5	3	28
		0	2		1			3	3	2	11
		−	2	6	5	10	4	7	2	5	41
	SA	+	9	6	9	10	8	9	9	9	69
		0	1	3	1		2	1	1	1	10
		−		1							1
	Total	+	26	19	14	15	28	19	24	20	165
	Total	0	30	33	22	14	21	28	22	25	195
	Total	−	34	38	54	61	41	43	44	45	360

Table 4
Comparison of LLMs against default random and grid searches at $\min(\max(epoch_{LLM}), \max(epoch_{default}))$. + indicates how many times LLM solution was better than search-based, 0 denotes no significant differences across 30 runs, and - refers to runs worse than default runs. Statistical significance was established using a t-student test.

LLM	Run	Random Search									Grid Search		
		TSP GA			JSSP PSO			GCP SA			GCP SA		
		+	0	−	+	0	−	+	0	−	+	0	−
Claude	Initial	1				1				1	1		
Claude	Feedback		1				1			1	1		
GPT-4o	Initial	1						1		1	1		
GPT-4o	Feedback			1				1		1	1		
Gemini	Initial			1		1			1		1		
Gemini	Feedback			1			1			1		1	
Mistral	Initial		1			1				1		1	
Mistral	Feedback			1		1				1	1		
default	Initial		1					1		1			1

Fig. 3 illustrates convergence behavior across methods. In the GCP case, random search significantly outperforms most other configurations, achieving optimality rapidly. However, this trend is less consistent in the remaining cases:

- for JSSP, both grid/random and LLM-based feedback runs achieve comparable convergence rates,
- for TSP - LLM feedback (Claude) slightly outpaces grid and random search in early convergence.

Fig. 4 illustrates contrast, with average runtime for exhaustive search-based methods (grid and random search) significantly exceeding that of LLM-guided setups. Notably, for both GA and PSO, grid and default configurations incur the highest computational costs, with default PSO running over 450 s on average. LLM-based initial and feedback runs remain consistently efficient, with average runtimes rarely exceeding 10 s (with the exception of GPT-4o). Intuitively, random search works faster than a grid search. However, it should be mentioned that these times depend highly on the parameter landscape. This highlights a clear trade-off between performance and cost, emphasizing the practical appeal of LLM-driven tuning in time-constrained scenarios. GCP - SA - myciel3 was not included in Fig. 4, as for this setup all times are consistently low (around 3 s). JSSP - PSO - abz5, TSP - GA - burma14 triplets were used for this analysis.

The computational and environmental cost of prompting Large Language Models is also an important factor to consider when evaluating the feasibility of proposed solutions. Prompting LLMs does not introduce time overhead (when models are *not thinking*) as they work nearly in real time. Allowing newer models to work in the reasoning mode increases the computational time to obtain parameters, but is still contained within a minute per request. Compared with exhaustive search procedures, LLMs have a time advantage over them. It should be mentioned that reasoning mechanisms in Large Language Models were not utilized in this study and can be used to deepen the exploration of the presented topic.

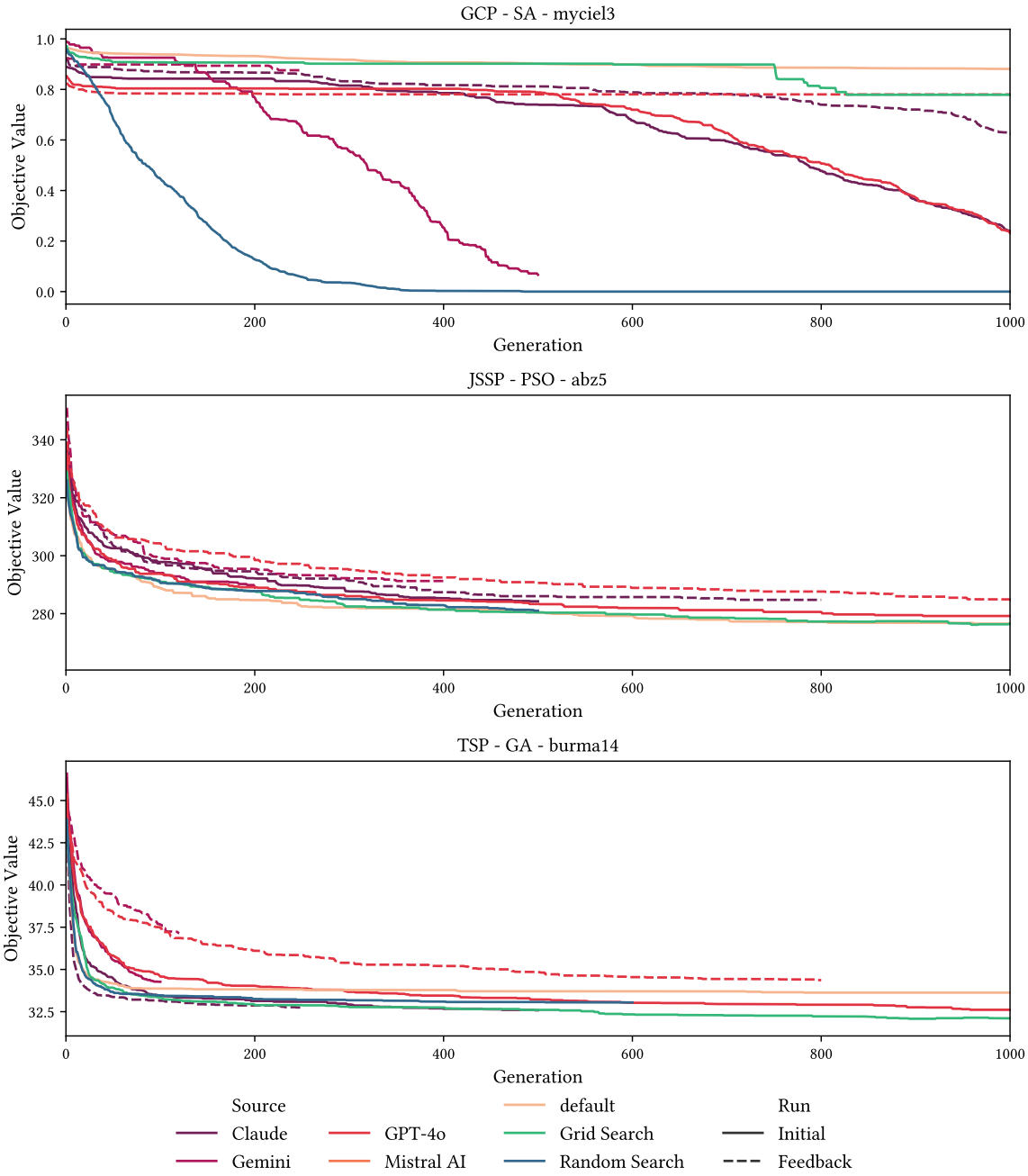


Fig. 3. Convergence plots with addition of Grid Search and Random Search procedures.

6.8. Additional remarks

Over the course of experimentation process some tendencies of selected large language models were observed. It was discovered that Le Chat by Mistral AI is likely to overestimate the parameters. Often times it would suggest the numbers of generations that exceed the limit implemented in the Mealpy library (100k generations). Runs based on Mistral AI suggestions are the most expensive in terms of computational cost. In contrast, Gemini tends to select substantially shorter experiments, by lowering the amount of suggested epochs.

It is worth mentioning, that at the early stage of experimentation Llama3 [88] was also considered as part of tested models. However, looking at parameters selected by Llama3 it can be said that this model does not exhibit awareness of heuristic optimizers at satisfactory level. Llama's suggestions were close to random and usually proposed maximum possible number of epochs. These were the reasons that supported the decision not to proceed with Llama3 further in the computation process.

Interestingly, models displayed completely different idea of what a *step_size* parameter in Simulated Annealing is. The range of values suggested is correlated with the Large Language Model. The parameter itself is a float with default value equal to 0.1. GPT-4o outputs values close to 3.6 on average, whereas values from models like Mistral AI or Claude oscillate around 26 and 1.17, respectively. Having these differences in mind, one can conclude that LLMs have considerably different understanding of the mechanisms in heuristic optimizers.

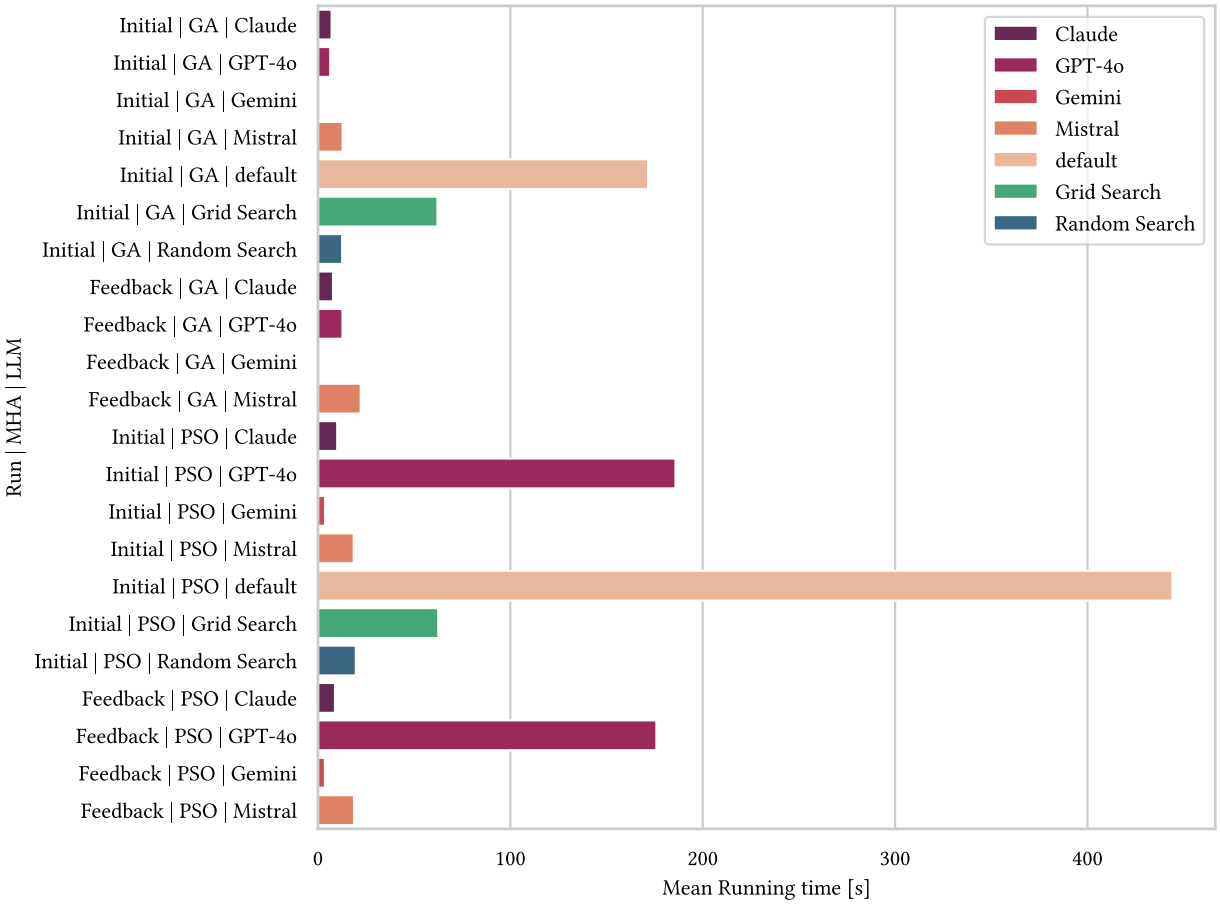


Fig. 4. Average time of calculations, gathered over 30 runs per each triplet configuration. GA runs were calculated based on *burma14* instance of a Traveling Salesman Problem, whereas PSO was gathered for *abz5* instance of JSSP. The values for Gemini-based runs for Genetic Algorithm are exceptionally low with 0.63 and and 1.03 s for initial and feedback runs correspondingly. It is due to the low amount of generations used in these runs.

7. Conclusions

Conducted experiments gathered substantial evidence that can support the claim stating that Large Language Models can cautiously guide the process of parameter tuning for MHAs. However, their performance is not always consistent and depends heavily on the level of exploration of a particular MHA in the research community. The popularity of the problem also plays an important role, as results show that the MHA-problem pairs that are well-established seem to be better understood by Large Language Models.

It was proven that prompts containing high-level information about the problem and achieved results are suitable bases for effective LLM decision-making. The exact data does not need to be fed to the model to get the desired recommendations. There are ways prompts could be further refined, especially those used for feedback rounds. Adding additional information about possible values of parameters could reduce the model's confusion.

A series of experiments showed that there is no single model that would outperform others. Yet, some of LLMs have general tendencies to overestimate certain values or fail to understand particular parameters of heuristic optimizers. Llama3 is an example of such a model, which proposed close to random values, that were additionally inconsistent in their logic.

Studies conducted on 10 instances of Graph Coloring Problem, Job-Shop Scheduling Problem, and Traveling Salesman Problem, utilizing 4 heuristic optimizers guided by 4 distinct LLMs in two rounds of prompting for parameter values, showed that Large Language Models can be helpful when working with MHAs. Results present that LLMs have abilities to draw conclusions from previous runs, yielding improvements in the feedback round of calculations. It is not consistent, but more careful choice of feedback prompts can potentially strengthen this behavior.

Other observations from the data suggest that LLMs may overfit or misestimate optimal values for smaller problem instances when prompted globally. Small problems are not always representative of Large Language Model's skills. The sensitivity of MHAs to parameter choices suggests that poor prompt design or overestimation of population parameters may have a negative impact on exploration process.

In order to overcome the shortcoming of LLMs' bound to presence of code documentation and research papers in their training set one could train a language model to specialize in the field of optimization. It can be beneficial because works in this area are constantly being published. A RAG or AI-optimizing agent distilling recent and canonical papers could have a better understanding of parameter meanings and suggest better values.

Another way of improvement is expansion of the idea of the feedback round. In this study, there was only a one iteration of it. Potential enhancement could involve more rounds of informed parameter suggestions, given the refinement of feedback prompt. Increasing the success rate between initial and feedback round is the crucial preliminary step for further improvements.

This study explored canonical problems and heuristic optimizers. It would be interesting to see, if language models can deal with less popular MHAs - as this would be a paramount experiment of their

reasoning. Still, one has to remember that this performance would be highly dependent on presence of data in the training set.

Both fields of MHAs and LLMs, are constantly growing, and it is exciting to see what their future integration will bring.

CRediT authorship contribution statement

Alicja Martinek: Writing – original draft, Visualization, Validation, Software, Methodology, Formal analysis, Data curation. **Ewelina Bartuzi-Trokielewicz:** Writing – original draft, Validation, Software. **Szymon Łukasik:** Supervision, Methodology, Conceptualization. **Amir H. Gandomi:** Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Appendix A. Benchmark instances of problems

See Table A.5.

Table A.5
Details of benchmark instances used in experiments.

problem	characteristic	instance	details	source
GCP	number of vertices and edges	myciel3	11 20	[45]
		myciel4	23 71	[45]
		myciel5	47 236	[45]
		myciel6	95 755	[45]
		myciel7	191 2360	[45]
		queen5_5	25 160	[45]
		queen6_6	36 290	[45]
		queen7_7	49 476	[45]
		queen8_8	64 728	[45]
		queen9_9	81 2112	[45]
JSSP	number of jobs and machines	abz5	10 10	[89]
		abz6	10 10	[89]
		ft06	6 6	[90]
		ft10	10 10	[90]
		la01	10 5	[91]
		la02	10 5	[91]
		la10	15 5	[91]
		la13	20 5	[91]
		la20	10 10	[91]
		orb01	10 10	[92]
TSP	number of cities	burma14	14	[86]
		ulysses16	16	[86]
		chn20	20	[93]
		ulysses22	22	[86]
		bays29	29	[86]
		chn30	30	[93]
		att48	48	[86]
		vrp8	50	[94]
		vrp9	75	[94]
		period1	127	[95]

Data availability

Data is publicly available.

References

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, L.U. Kaiser, I. Polosukhin, Attention is all you need, in: I. Guyon, U.V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, R. Garnett (Eds.), Advances in Neural Information Processing Systems, vol. 30, Curran Associates, Inc, 2017, pp. 5998–6008.

[2] J. Devlin, M.-W. Chang, K. Lee, K. Toutanova, BERT: Pre-training of deep bidirectional transformers for language understanding, arXiv preprint arXiv:1810.04805., 2019.

[3] R. Azamfirei, S.R. Kudchadkar, J. Fackler, Large language models and the perils of their hallucinations, Crit Care 27 (1) (2023) 120.

[4] A. Salinas, L. Penafiel, R. McCormack, F. Morstatter, "im not racist but…": discovering bias in the internal knowledge of large language models, arXiv preprint arXiv:2310.08780, 2023.

[5] J. Kaddour, J. Harris, M. Mozes, H. Bradley, R. Raileanu, R. McHardy, Challenges and applications of large language models, arXiv preprint arXiv:2307.10169, 2023.

[6] K. Rajwar, K. Deep, S. Das, An exhaustive review of the metaheuristic algorithms for search and optimization: taxonomy, applications, and open challenges, Artif. Intell. Rev. 56 (04 2023), https://doi.org/10.1007/s10462-023-10470-y

[7] R. Martí, M. Sevaux, K. Sörensen, 50 years of metaheuristics, Eur. J. Oper. Res. 321 (2) (2024) https://doi.org/10.1016/j.ejor.2024.04.004.

[8] A. Martinek, S. Łukasik, A. Gandomi, Large language models as tuning agents of metaheuristics, in: ESANN 2024 Proceedings, 2024, pp. 631–636, https://doi.org/10.14428/esann/2024.ES2024-209

[9] OpenAI, Hello GPT-4o, 2025, https://openai.com/index/hello-gpt-4o/ (accessed: 20 March 2025).

[10] G. Yenduri, G. Srivastava, P.K.R. Maddikunta, R.H. Jhaveri, W. Wang, A.V. Vasilakos, T.R. Gadekallu, Generative pre-trained transformer: a comprehensive review on enabling technologies, potential applications, emerging challenges, and future directions, arXiv preprint arXiv:2305.10435, 2023.

[11] OpenAI et al., Gpt-4 technical report, arXiv preprint arXiv:2303.08774, 2024

[12] Anthropic, Claude 3.7 sonnet and Claude Code, 2025. https://www.anthropic.com/news/claude-3-7-sonnet, (Accessed: 2025–march–20).

[13] Gemini Team et al., Gemini: a family of highly capable multimodal models, arXiv preprint arXiv:2312.11805, 2024

[14] W. Fedus, B. Zoph, N. Shazeer, Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity, arXiv preprint arXiv:2101.03961, 2022.

[15] Mistral AI, Le Chat: advanced conversational AI, 2024, https://mistral.ai (Accessed 20 March 2025).

[16] M. AI, Large enough, 2025, https://mistral.ai/news/mistral-large-2407 (Accessed 20, March, 2025).

[17] A.Q. Jiang, A. Sablayrolles, A. Mensch, C. Bamford, D.S. Chaplot, D. de las Casas, F. Bressand, G. Lengyel, G. Lample, L. Saulnier, L.R. Lavaud, M.-A. Lachaux, P. Stock, T.L. Scao, T. Lavril, T. Wang, T. Lacroix, W.E. Sayed, Mistral 7B, arXiv preprint arXiv:2310.06825, 2023.

[18] A. Ezugwu, A. Shukla, R. Nath, A. Akinyelu, O. Agushaka, H. Chiroma, P. Muhuri, Metaheuristics: a comprehensive overview and classiication along with bibliometric analysis, Artif. Intell. Rev. 54 (2021) 1–79, https://doi.org/10.1007/s10462-020-09952-0

[19] V. Tomar, M. Bansal, P. Singh, Metaheuristic algorithms for optimization: a brief review, Eng. Proc. 59 (1) (2024) 238.

[20] B. Benaissa, M. Kobayashi, M.A. Ali, T. Khatir, M.E.A.E. Elmeliani, Metaheuristic optimization algorithms: an overview, HCMCOU J. Sci. Adv. Comput. Struct. (2024) 33–61.

[21] M. Abdel-Basset, L. Abdel-Fatah, A.K. Sangaiah, Chapter 10 - metaheuristic algorithms: a comprehensive review, In A.K. Sangaiah, M. Sheng, Z. Zhang (Eds.), Computational Intelligence for Multimedia Big Data on the Cloud with Engineering Applications, Intelligent Data-Centric Systems, Academic Press, 2018, pp. 185–231, https://doi.org/10.1016/B978-0-12-813314-9.00010-4

[22] S. Almufti, R. Marqas, V. Ashqi, Taxonomy of bio-inspired optimization algorithms, J. Adv. Comput. Sci. Technol. 8 (2) (23 2019).

[23] Z. Beheshti, S.M.H. Shamsuddin, A review of population-based meta-heuristic algorithms, Int. J. Adv. Soft Comput. Appl. 5 (1) (2013) 1–35.

[24] S. Kirkpatrick, C.D. Gelatt Jr, M.P. Vecchi, Optimization by simulated annealing, Science 220 (4598) (1983) 671–680.

[25] F. Glover, Tabu Search: a tutorial, Interfaces 20 (4) (1990) 74–94.

[26] D.E. Kvasov, M.S. Mukhametzhanov, Metaheuristic VS. deterministic global optimization algorithms: the univariate case, Appl. Math. Comput., Recent Trends Numer. Comput.: Theory Algorithms. 318 (2018) 245–259, https://doi.org/10.1016/j.amc.2017.05.014, https://www.sciencedirect.com/science/article/pii/S0096300317303028.

[27] G.R. Raidl, A unified view on hybrid metaheuristics, in: International Workshop on Hybrid Metaheuristics, Springer, 2006, pp. 1–12.

[28] T. Ting, X.-S. Yang, S. Cheng, K. Huang, Hybrid metaheuristic algorithms: past, present, and future, Recent Adv. Swarm Intell. Evol. Comput. (2014) 71–83.

[29] A. Seyyedabbasi, W.Z.T. Tareq, N. Bacanin, An effective hybrid metaheuristic algorithm for solving global optimization algorithms, Multimed. Tools Appl. 83 (37) (2024) 85103–85138.

[30] S. Mirjalili, S. Mirjalili, Genetic algorithm, evolutionary algorithms and neural networks: theory and applications, Stud. Comput. Intell. (2019) 43–55.

[31] J.H. Holland, Genetic algorithms, Sci. Am. 267 (1) (1992) 66–73.

[32] A. Lambora, K. Gupta, K. Chopra, Genetic algorithm-a literature review, in: 2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon), IEEE, 2019, pp. 380–384.

[33] M. Dorigo, M. Birattari, T. Stutzle, Ant colony optimization, IEEE Comput. Intell. Mag. 1 (4) (2006) 28–39.

[34] M. Dorigo, G. Di Caro, Ant colony optimization: a new meta-heuristic, Proc. 1999 Congr. Evol. Comput.-CEC99 (Cat.99TH8406), Vol. 2 (1999) 1470–1477, IEEE.

[35] M. Dorigo, T. Stützle, Ant colony optimization: overview and recent advances, Handb. Metaheuristics (2018) 311–351.

[36] J. Kennedy, R. Eberhart, Particle swarm optimization, in: Proceedings of ICNN'95 - International Conference on Neural Networks, vol. 4, 1995, pp. 1942–1948, https://doi.org/10.1109/ICNN.1995.488968

[37] D. Wang, D. Tan, L. Liu, Particle swarm optimization algorithm: an overview, Soft Comput. 22 (2) (2018) 387–408.

[38] A.G. Gad, Particle swarm optimization algorithm and its applications: a systematic review, Arch. Comput. Methods Eng. 29 (5) (2022).

[39] D. Bertsimas, J. Tsitsiklis, Simulated annealing, Stat. Sci. 8 (1) (1993) 10–15.
[40] P.J. Van Laarhoven, E.H. Aarts, Simulated annealing, in: Simulated Annealing: Theory and Applications, Springer, 1987, pp. 7–15.
[41] P. Grabusts, J. Musatovs, V. Golenkov, The application of simulated annealing method for optimal route detection between objects, Proc. Comput. Sci., iCTE in Transportation and Logistics 2018 (ICTE 2018) 149 (2019) 95–101, https://doi.org/10.1016/j.procs.2019.01.112
[42] C. Blum, A. Roli, Metaheuristics in combinatorial optimization: overview and conceptual comparison, ACM Comput. Surv. 35 (3) (2003) 268–308.
[43] B.H. Korte, J. Vygen, B. Korte, J. Vygen, Combinatorial Optimization, vol. 1, Springer, 2011.
[44] P.M. Pardalos, T. Mavridou, J. Xue, The graph coloring problem: a bibliographic survey, Handbook of Combinatorial Optimization 1–3 Springer 1077–1141, (1998).
[45] M. Trick, Graph coloring instances, 2018, https://mat.tepper.cmu.edu/COLOR/instances.html (Accessed 20 March, 2024).
[46] S. Ahmed, Applications of graph coloring in modern computer science, Int. J. Comput. Inf. Technol. 3 (2) (2012) 1–7.
[47] R.L. Graham, Bounds for certain multiprocessing anomalies, Bell Syst. Tech. J. 45 (9) (1966) 1563–1581, https://doi.org/10.1002/j.1538-7305.1966.tb01709.x
[48] J. Błażewicz, E. Pesch, M. Sterna, The disjunctive graph machine representation of the job shop scheduling problem, Eur. J. Oper. Res. 127 (2) (2000) 317–331, https://doi.org/10.1016/S0377-2217(99)00486-5
[49] B. Gavish, S.C. Graves, The travelling salesman problem and related problems, Working paper or-078-78, Massachusetts Institute of Technology, Operations Research Center, Cambridge, MA, USA, 1978.
[50] K. Ilavarasi, K.S. Joseph, Variants of travelling salesman problem: a survey, in: International Conference on Information Communication and Embedded Systems (ICICES2014), 2014, pp. 1–7, https://doi.org/10.1109/ICICES.2014.7033850
[51] Q. Guo, R. Wang, J. Guo, B. Li, K. Song, X. Tan, G. Liu, J. Bian, Y. Yang, Connecting large language models with evolutionary algorithms yields powerful prompt optimizers, in: The Twelfth International Conference on Learning Representations, 2024, https://openreview.net/forum?id=ZG3RaNIsO8.
[52] R. Pan, S. Xing, S. Diao, W. Sun, X. Liu, K. Shum, R. Pi, J. Zhang, T. Zhang, Plum: Prompt learning using metaheuristic, arXiv preprint arXiv:2311.08364, 2024.
[53] Y. Chen, J. Arkin, Y. Hao, Y. Zhang, N. Roy, C. Fan, Prompt optimization in multi-step tasks (promst): Integrating human feedback and preference alignment, arXiv preprint arXiv:2402.08702, 2024.
[54] S. Brahmachary, S. Joshi, A. Panda, K. Koneripalli, A. Sagotra, H. Patel, A. Sharma, A. Jagtap, K. Kalyanaraman, Large language model-based evolutionary optimizer: reasoning with elitism Neurocomputing 622 (2025) 129272.
[55] S. Liu, C. Chen, X. Qu, K. Tang, Y.-S. Ong, Large language models as evolutionary optimizers, arXiv preprint arXiv:2310.19046, 2023.
[56] H. Ye, J. Wang, Z. Cao, F. Berto, C. Hua, H. Kim, J. Park, G. Song, Reevo: large language models as hyper-heuristics with reflective evolution, Advances in Neural Information Processing Systems, pp. 43571–43608, 2024, https://github.com/ai4co/reevo.
[57] F. Liu, X. Tong, M. Yuan, X. Lin, F. Luo, Z. Wang, Z. Lu, Q. Zhang, Evolution of heuristics: towards efficient automatic algorithm design using large language model, in: Proceedings of the 41st International Conference on Machine Learning,ICML'24, JMLR.org, 2024.
[58] M.R. Zhang, N. Desai, J. Bae, J. Lorraine, J. Ba, Using large language models for hyperparameter optimization, arXiv preprint arXiv:2312.04528, 2023.
[59] O. Kramer, Large language models for tuning evolution strategies, arXiv preprint arXiv:2405.10999, 2024.
[60] O. Kramer, Llama tunes cma-es, in: ESANN 2024 Proceedings, 2024, pp. 601–606, https://doi.org/10.14428/esann/2024.ES2024-136
[61] L.L. Custode, F. Caraffini, A. Yaman, G. Iacca, An investigation on the use of large language models for hyperparameter tuning in evolutionary algorithms, in: Genetic and Evolutionary Computation Conference (GECCO '24 Companion), GECCO '24 Companion, Association for Computing Machinery, New York, NY, USA, 2024, pp. 1838–1845, https://doi.org/10.1145/3638530.3664163. URL:10.1145/3638530.3664163
[62] X. Wu, S. hao Wu, J. Wu, L. Feng, K.C. Tan, Evolutionary computation in the era of large language model: survey and roadmap, arXiv preprint arXiv:2401.10034, 2024.
[63] S. Huang, K. Yang, S. Qi, R. Wang, When large language model meets optimization, arXiv preprint arXiv:2405.10098, 2024.
[64] M. Pluhacek, A. Kazikova, T. Kadavy, A. Viktorin, R. Senkerik, Leveraging large language models for the generation of novel metaheuristic optimization algorithms, in: Proceedings of the Companion Conference on Genetic and Evolutionary Computation, GECCO '23 Companion, Association for Computing Machinery, New York, NY, USA, 2023, pp. 1812–1820, https://doi.org/10.1145/3583133.3596401. URL:https://doi.org/10.1145/3583133.3596401
[65] N. van Stein, T. Bäck, LLaMEA: a large language model evolutionary algorithm for automatically generating metaheuristics, arXiv preprint arXiv:2405.20132, 2024.
[66] F. Liu, X. Tong, M. Yuan, Q. Zhang, Algorithm evolution using large language model, arXiv preprint arXiv:2311.15249, 2023.
[67] C. Morris, M. Jurado, J. Zutty, LLM guided evolution - the automation of models advancing models, in: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '24, Association for Computing Machinery, New York, NY, USA, 2024, pp. 377–384, https://doi.org/10.1145/3638529.3654178. URL:https://doi.org/10.1145/3638529.3654178
[68] C. Yang, X. Wang, Y. Lu, H. Liu, Q.V. Le, D. Zhou, X. Chen, Large language models as optimizers, arXiv preprint arXiv:2309.03409, 2024.
[69] J. Bergstra, R. Bardenet, B. Kégl, Y. Bengio, Algorithms for hyper-parameter optimization, in: Advances in Neural Information Processing Systems, vol. 24, Curran Associates, Inc., Granada, Spain, 2011, pp. 2546–2554.
[70] H. Alibrahim, S.A. Ludwig, Hyperparameter optimization: comparing genetic algorithm against grid search and Bayesian optimization, in: 2021 IEEE Congress on Evolutionary Computation (CEC), 2021, pp. 1551–1559, https://doi.org/10.1109/CEC45853.2021.9504761
[71] F. Pontes, G. Amorim, P. Balestrassi, A. Paiva, J. Ferreira, Design of experiments and focused grid search for neural network parameter optimization, Neurocomputing 186 (2016) 22–34, https://doi.org/10.1016/j.neucom.2015.12.061, https://www.sciencedirect.com/science/article/pii/S0925231215020184.
[72] K.B. Ensor, P.W. Glynn, Stochastic optimization via grid search, Lect. Appl. Math.-Am. Math. Soc. 33 (1997) 89–100.
[73] M. Claesen, J. Simm, D. Popovic, Y. Moreau, B.D. Moor, Easy hyperparameter search using optunity, arXiv preprint arXiv:1412.1114, 2014.
[74] J. Bergstra, Y. Bengio, Random search for hyper-parameter optimization, J. Mach. Learn. Res. 13 2012) 281–305.
[75] L. Zahedi, F.G. Mohammadi, S. Rezapour, M.W. Ohland, M.H. Amini, Search algorithms for automated hyper-parameter tuning, arXiv preprint arXiv:2104.14677, 2021.
[76] P.I. Frazier, A tutorial on bayesian optimization, arXiv preprint arXiv:1807.02811, 2018.
[77] Z. Wang, F. Hutter, M. Zoghi, D. Matheson, N. de Freitas, Bayesian optimization in a billion dimensions via random embeddings, arXiv preprint arXiv:1301.1942, 2016.
[78] D. Maclaurin, D. Duvenaud, R.P. Adams, Gradient-based hyperparameter optimization through reversible learning, arXiv preprint arXiv:1502.03492, 2015.
[79] L. Franceschi, M. Donini, P. Frasconi, M. Pontil, Forward and reverse gradient-based hyperparameter optimization, arXiv preprint arXiv:1703.01785, 2017.
[80] E. Shadkam, Parameter setting of meta-heuristic algorithms: a new hybrid method based on DEA and RSM, Environ. Sci. Pollut. Res. 29 (2021) 1–23, https://doi.org/10.1007/s11356-021-17364-y
[81] S.K. Joshi, J.C. Bansal, Parameter tuning for meta-heuristics, Knowl. Based Syst., 189 2020, 105094, https://doi.org/10.1016/j.knosys.2019.105094
[82] I. Pereira, A. Madureira, Racing based approach for metaheuristics parameter tuning, in: 2015 10th Iberian Conference on Information Systems and Technologies (CISTI), 2015, pp. 1–6, https://doi.org/10.1109/CISTI.2015.7170351
[83] C. Huang, Y. Li, X. Yao, A survey of automatic parameter tuning methods for metaheuristics, IEEE Trans. Evol. Comput. 24 (2) (2020) 201–216, https://doi.org/10.1109/TEVC.2019.2921598
[84] L. Zheng, W.-L. Chiang, Y. Sheng, S. Zhuang, Z. Wu, Y. Zhuang, Z. Lin, Z. Li, D. Li, E.P. Xing, H. Zhang, J.E. Gonzalez, I. Stoica, Judging llm-as-a-judge with mt-bench and chatbot arena, arXiv preprint arXiv:2306.05685, 2023.
[85] J.E. Beasley, Or-Library: distributing test problems by electronic mail, J. Oper. Res. Soc. 41 (11) (1990) 1069–1072, http://www.jstor.org/stable/2582903.
[86] J. Burkardt, TSP data for the traveling salesperson problem, 2019, https://people.sc.fsu.edu/jburkardt/datasets/tsp/tsp.html (Accessed: 1 March 2024).
[87] N. Van Thieu, S. Mirjalili, MEALPY: an open-source library for latest meta-heuristic algorithms in Python, J. Syst. Archit. 139 (2023), https://doi.org/10.1016/j.sysarc.2023.102871
[88] Meta AI, Llama3,2024 https://github.com/meta-llama/llama3.
[89] J. Adams, E. Balas, D. Zawack, The shifting bottleneck procedure for job shop scheduling, Manag. Sci. 34 (3) (1988) 391–401, http://www.jstor.org/stable/2632051.
[90] J. Muth, Probabilistic learning combinations of local job-shop scheduling rules, in: Industrial Scheduling, 1963, pp. 225–251.
[91] S. LAWRENCE, Resouce constrained project scheduling: an experimental investigation of heuristic scheduling techniques (supplement), in: Graduate School of Industrial Administration, Carnegie-Mellon University, 1984, https://cir.nii.ac.jp/crid/1571980073974705920.
[92] D. Applegate, W. Cook, A computational study of the job-shop scheduling problem, Inf. J. Comput. 3 (1991) 149–156, https://doi.org/10.1287/ijoc.3.2.149
[93] 10-zin, Ant-colony-tsp, Github repository, 2020. https://github.com/10-zin/ant-colony-tsp/tree/master/data., Accessed: March 20 2024.
[94] S. Eilon, C. Watson-Gandy, N. Christofides, R. de Neufville, Distribution management-mathematical modelling and practical analysis, systems, Man Cybern, IEEE Trans. 21 (1974) 589, https://doi.org/10.1109/TSMC.1974.4309370
[95] R. Russell, W. Igo, An assignment routing problem, Networks 9 (1) (1979) 1–17, https://doi.org/10.1002/net.3230090102, https://onlinelibrary.wiley.com/doi/abs/10.1002/net.3230090102.

Author biography

Alicja Martinek received the B.Eng. and M.Sc. degrees in Biomedical Engineering from AGH University of Kraków, Poland, in 2016 and 2018, respectively. In 2017, she also earned the M.Sc. degree in Advanced Computer Science from the University of Kent, Canterbury, United Kingdom. Since 2021, she has been pursuing the Ph.D. degree in Information Technology and Telecommunications at AGH University of Kraków under the supervision of Prof. Szymon Łukasik and Prof. Amir Gandomi. She joined NASK – National Research Institute in 2023, where she works as a Senior Machine Learning Specialist. Her main research interests, and her Ph.D. work, revolve around deepfake detection, explainable artificial intelligence, and bio-inspired computing.

Ewelina Bartuzi-Trokielewicz received the B.Eng. and the M.Sc. degree in Biomedical Engineering from the Warsaw University of Technology, Warsaw, Poland, in 2016 and 2017, respectively. She pursued her Ph.D. degree at Warsaw University of Technology. Her dissertation explored presentation attack-resistant identity recognition for mobile devices in unconstrained conditions. In 2016 she joined NASK - National Research Institute, Warsaw, Poland, where she currently is a Head of Department of Audiovisual Analysis and Biometric Systems working within AI Safety Research Center. Her main research interests include biometrics, presentation attacks, and deepfake detection, computer vision.

Szymon Łukasik received the M.Sc. degrees in Computer Science (2005) and in Automatic Control (2006) from the Cracow University of Technology, Kraków, Poland. He earned the Ph.D. degree in Computer Science from the Systems Research Institute of the Polish Academy of Sciences in 2012, and completed his habilitation in Technical Sciences in 2020. In 2022 he joined NASK – National Research Institute, Warsaw, Poland, where he is the Director of the AI Safety Research Center. He is also an associate professor at the AGH University of Krakow. His main research interests include computational intelligence, metaheuristics and swarm methods, and AI safety and trustworthy AI.

Amir H. Gandomi is among the world's most cited researchers for his work in the fields of global optimisation and big data analytics, in particular, using machine learning and evolutionary computations. In 2019 he joined the University of Technology Sydney (UTS), Australia, where he is a Professor of Data Science and an ARC DECRA Fellow. He has previously served as an Assistant Professor at Stevens Institute of Technology, USA, and as a distinguished research fellow at the BEACON Center, Michigan State University; he is also a Distinguished Professor affiliated with Óbuda University, Budapest. His main research interests include global optimisation, evolutionary computation, machine learning, and big data analytics.