IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 17 October 2024, accepted 24 December 2024, date of publication 30 December 2024, date of current version 3 January 2025.

Digital Object Identifier 10.1109/ACCESS.2024.3524176

RESEARCH ARTICLE

Metaheuristics and Large Language Models Join Forces: Toward an Integrated Optimization Approach

CAMILO CHACÓN SARTORI, CHRISTIAN BLUM, FILIPPO BISTAFFA, AND GUILLEM RODRÍGUEZ COROMINAS
Artificial Intelligence Research Institute (IIIA-CSIC), Bellaterra, 08193 Barcelona, Spain

Corresponding author: Camilo Chacón Sartori (cchacon@iiia.csic.es)

This work was supported by MCIN/AEI/10.13039/ 501100011033 under Grant TED2021-129319B-I00 and Grant PID2022-136787NB-I00.

ABSTRACT Since the rise of Large Language Models (LLMs) a couple of years ago, researchers in metaheuristics (MHs) have wondered how to use their power in a beneficial way within their algorithms. This paper introduces a novel approach that leverages LLMs as pattern recognition tools to improve MHs. The resulting hybrid method, tested in the context of a social network-based combinatorial optimization problem, outperforms existing state-of-the-art approaches that combine machine learning with MHs regarding the obtained solution quality. By carefully designing prompts, we demonstrate that the output obtained from LLMs can be used as problem knowledge, leading to improved results. Lastly, we acknowledge LLMs' potential drawbacks and limitations and consider it essential to examine them to advance this type of research further. Our method can be reproduced using a tool available at: https://github.com/camilochs/optipattern.

INDEX TERMS Combinatorial optimization, hybrid algorithm, metaheuristics, large language models.

I. INTRODUCTION

The advent of Large Language Models (LLMs) has altered the Natural Language Processing (NLP) landscape, empowering professionals across diverse disciplines with their remarkable ability to generate human-like text. Models like OpenAI's GPT [1], Meta's Llama [2], and Anthropic's Claude 3 [3] have become indispensable collaborators in many peoples' daily lives; giving rise to innovative products such as ChatGPT for general use, GitHub Copilot for code generation, DALL-E 2 for image creation, and a multitude of voice generators, including OpenAI's text-to-speech API and ElevenLabs's Generative Voice AI. Currently, LLMs are being experimentally applied across various fields, yielding mixed results [4]. While some applications seem questionable, others exhibit spectacular outcomes. One of the most contentious applications is using LLMs for tasks necessitating mathematical reasoning. Given LLMs' inherently probabilistic nature, this application was once deemed implausible. However, recent findings suggest a shift in perspective, particularly with LLMs boasting vast parameter counts [5]. As LLMs continue to scale, new capabilities emerge [6]. Crucially, these opportunities are contingent upon the thoughtful design of prompts, which helps mitigate the risk of LLMs providing irrelevant or inaccurate responses [7].

Whenever a new technology emerges, it is natural to wonder if it might enhance an existing one. In combinatorial optimization, metaheuristics (MHs) [8] have been established as effective approximate algorithms for tackling complex, NP-hard problems. While they excel in rapidly providing good-enough solutions, they depend heavily on domain-specific knowledge. To address this limitation, researchers have explored the integration of MHs with other approaches, including exact algorithms and Machine Learning (ML). While combining MHs with exact algorithms has shown promise [9], a successful integration demands significant technical expertise. Alternatively, incorporating ML techniques within MHs can provide valuable problem

The associate editor coordinating the review of this manuscript and approving it for publication was Bijoy Chand Chatterjee.

insights [10], but these approaches often require specialized know-how or, in the case of Deep Learning (DL), substantial datasets and computational resources for training [11]. This paper seeks another direction. We delve into the potential of LLMs to create a novel hybrid approach that combines the strengths of MHs and LLMs.

A. OUR CONTRIBUTION

We present a novel approach to enhance MHs performance utilizing LLMs' output. Our method diverges from existing techniques in two key aspects:

- Rather than employing LLMs to generate MHs (e.g., [12], [13], [14]), our approach leverages them as pattern recognition tools for problem instance metrics. This strategy allows for seamless integration into existing MHs by introducing an additional LLM-provided parameter.
- Unlike methods that use LLMs as direct optimizers for natural language-described problems [15], [16], [17]—a technique limited to simple optimization tasks due to LLMs' stochastic nature—our approach tackles complex optimization challenges by using LLMs to identify and track pertinent information within the problem instance.

This dual-faceted approach represents a significant advancement in the integration of LLMs with metaheuristic optimization techniques, offering a more robust and versatile framework for tackling a wide array of optimization challenges. Thus, we employ LLMs not as an oracle providing final answers (i.e., it is not an *end-to-end* approach according to the classification by Bengio et al. [18]) but as an *intermediate step*, assisting in pattern detection within the metrics' values (see Figure 2), i.e., it is an *hybrid* one.

We validate our proposed `MH+LLM` integration using the *Multi-Hop Influence Maximization in Social Networks* problem, demonstrating improved performance over the current state-of-the-art approach, which combines MH with deep learning (DL) [19]. Therefore, we believe this approach can unlock new possibilities for improving MHs by leveraging generative AI to tackle complex pattern recognition tasks.

The paper is organized as follows. Section II examines existing approaches for integrating ML into MHs and provides an overview of existing research on applying LLMs in optimization. Section III formally defines the NP-Hard combinatorial optimization problem that serves as an example for our study. Our proposed integration strategy for combining MH and LLMs is presented in Section IV. The empirical evaluation of our hybrid approach is detailed in Section V, where we introduce a comprehensive three-dimensional framework for assessment and provide a visual analysis of the algorithm's performance. Section VI identifies remaining open research questions and discusses the current limitations of LLMs. Finally, Section VII concludes the paper by summarizing our key findings. Moreover, future research directions are mentioned.

B. REPRODUCIBILITY

Recognizing the importance of reproducibility in our field [20], and the potential challenges introduced by new technologies, we have developed a tool called `OptiPattern` (*LLM-Powered Pattern Recognition for Combinatorial Optimization*) that automates our hybridization method—detailed in Section IV—to ensure greater ease and accuracy in replication. This tool allows researchers to input a problem instance, generating the full prompt in response. Furthermore, by incorporating an LLM API key, the tool can return node-specific probabilities, which can then be integrated into the metaheuristic.[1] This is essential, as replicating prompts can be complex and prone to errors.

II. BACKGROUND

A. MACHINE LEARNING FOR ENHANCING METAHEURISTICS IN COMBINATORIAL OPTIMIZATION

Metaheuristics (MHs) are approximate algorithms that have proven effective in solving complex optimization problems, especially combinatorial optimization problems (COPs). COPs are characterized by discrete variables and a finite search space. Although MHs have been shown to deliver good results in reduced computation times, they do not guarantee finding the optimal solution. Moreover, their success often hinges on the availability of problem-specific knowledge. Each problem instance is treated similarly, applying problem-specific heuristics and (generally) relying on a stochastic behavior. Along these lines, the community aims to innovate by integrating techniques from various domains to improve MHs' performance and address the limitations of MH techniques. In particular, *hybrid* approaches based on the combination with (1) exact algorithms [9] and (2) learning techniques [10] have been explored. Currently, the primary focus has shifted towards the second option. Especially the integration of machine learning (ML) techniques has recently resulted in a multitude of different hybrid approaches. Researchers have explored various strategies to integrate ML into MHs [10]. In particular, ML might be used for the following purposes in MHs: algorithm selection (determining the best MH for a given problem), fast approximate fitness evaluation in the presence of costly objective functions, initialization (generating high-quality initial solutions), and parameter configuration (optimizing the numerous parameters of an MH, which is crucial for its performance). These strategies utilize learning techniques to analyze numerous cases and scenarios, uncovering hidden patterns in the data. By identifying these patterns, MHs can extract general principles that apply to a broad range of situations. This enhances MH's decision-making process, increasing their performance and adaptability.[2]

Beyond classical ML methods (supervised learning: logistic regression, decision trees, support vector machines;

[1] https://github.com/camilochs/optipattern

[2] The reverse integration, which involves enhancing ML architectures with MH techniques, is beyond the scope of this study.

unsupervised learning: k-means clustering, principal component analysis), there are approaches from several research subfields within the discipline that have been leveraged and tailored for their use in MHs. Each subfield is characterized by its unique methods, strategies, and possibilities. For instance, deep learning (DL) and reinforcement learning (RL) are two areas that have proven particularly promising for their integration with MHs.

DL employs many-layered artificial neural networks to automatically learn complex data representations, whereas RL focuses on sequential decision-making to achieve good results using a trial-and-error learning process. Methods from both fields have found valuable applications in the realm of MHs, enhancing their ability to find high-quality solutions. In fact, a growing body of research demonstrates this hybrid approach's success. In the following, we provide short descriptions of exemplary hybridization approaches from three different categories:

- `MH+ML`: In a study by Sun et al. [21], ML techniques were incorporated into the metaheuristic Ant Colony Optimization (ACO) to address the orienteering problem. The authors improved the solution construction process of ACO by utilizing guided predictions based on engineered features. In [22], the authors developed a novel algorithm that combines a metaheuristic with decision trees to address the classic vehicle routing problem.
- `MH+DL`: Examples of this type of hybridization can be found abundantly in the literature of recent years. For instance, in [19], the authors presented a novel approach that uses a Graph Neural Network (GNN) for learning heuristic information that is then used by a Biased Random Key Genetic Algorithm (BRKGA) to translate random keys into solutions to the tackled problem. Another example concerns [23] where the authors apply different GNN architectures for parametrizing the neighbor selection strategy in Tabu Search (TS) and in Large Neighborhood Search (LNS).
- `MH+RL`: RL has recently been used in numerous hybrid approaches. For instance, in [24], the authors devised a method for learning the heuristic function of beam search in the context of two variants of the Longest Common Subsequence (LCS) problems. In [25], the authors proposed a hybrid approach comprised of an attention-based model trained with RL and combined with a more classical optimization method for the formation of collectives of agents in real-world scenarios, showing that it reaches the performance of state-of-the-art solutions while being more general. Furthermore, a variable neighborhood search (VNS) based on Q-learning was devised for a machine scheduling problem in [26]. Finally, we also mention [27], where RL is used for adapting the parameters of a BRKGA during the evolutionary process.

While these hybridization strategies offer potential solutions, they each come with their own set of drawbacks. For instance, the manual feature selection process in many `MH+ML` approaches relies heavily on the expertise of a specialist. Concerning `MH+DL` approaches, the system's generalization ability might be hindered by lacking a large and diverse enough dataset. As for `MH+RL` hybrids, the complexity lies in defining the action space, rewards, and learning policies in a clear and effective manner. Moreover, all three approaches share similar technical challenges: the complexity of replicating models, the time-consuming process of data collection and preparation, and the computational demands of generalization, especially for large-scale problems or complex optimization tasks [11].

To explore innovative methods for enhancing the performance of MHs and considering the usefulness and potential of LLMs, which will be discussed in the following subsection, in this paper, we explore a novel hybrid approach: `MH+LLM` (see Section IV). Utilizing the capabilities of LLMs—while being aware of their limitations—we aim to enhance the problem-solving capabilities of MHs and open up new avenues for tackling complex optimization problems.

B. LLMS AS PATTERN RECOGNITION ENGINES

LLMs have breathed new life into the field of NLP. These high-level language models employ billions of parameters and exhibit an outstanding ability to learn from data. Models like GPT-4o (OpenAI),[3] Claude-3-Opus (Anthropic),[4] Gemini 1.5 (Google),[5] Mixtral 8 × 22b (Mistral AI),[6] and Command-R+ (Cohere),[7] as well as tools built on top of them—such as ChatGPT, GitHub Copilot, and Bing Chat—have demonstrated that we are in the presence of a groundbreaking technology. Unlike previous advancements, this new wave of AI is no longer limited to experts; instead, it is accessible to anyone who can grasp its benefits.

LLMs are generative AI models that produce text sequentially, predicting each token based on the previous ones. This is made possible by the Transformer, a groundbreaking DL architecture that revolutionized the field of NLP. Proposed by Vaswani et al. [28], it introduces the concept of self-attention, allowing the model to contextually select the most suitable words. The Transformer derives its name from its ability to *transform* a set of vectors in a given representation space to a new set of vectors with identical dimensions but in a different space. By assigning varying weight values to each input, the attention mechanism leverages inductive biases related to sequential data [29]. This architecture also takes advantage of the capabilities of high-performance hardware due to its parallelizable nature. As a result, the Transformer generates words that seamlessly fit the context—although it lacks

[3] https://openai.com/index/hello-gpt-4o
[4] https://www.anthropic.com/news/claude-3-family
[5] https://deepmind.google/technologies/gemini
[6] https://mistral.ai/news/mixtral-8 × 22b
[7] https://docs.cohere.com/docs/command-r-plus

factual verification—marking a significant advancement in NLP tasks.

LLMs have found uses in various domains, including the interpretation of complex results like chemical compounds or images [30], where they provide insights and explanations. Additionally, LLMs are being applied as autonomous agents that leverage external tools and resources to accomplish tasks [31]. Furthermore, these models demonstrate progress in domains once thought to be out of the reach of their capabilities, including mathematical reasoning and optimization tasks [15]. While there are still numerous obstacles to overcome [5], these advancements showcase the potential of these models to address intricate cognitive problems.

Recent research on leveraging LLMs for optimization has primarily explored two paths: first, formulating optimization problems within the prompt and requesting the LLM to solve the described problem, typically for straightforward optimization tasks [15], [16], [17]; and second, automating code generation to enhance optimization algorithms [12], [13], [14]. While both strategies are effective in leveraging LLMs to address certain weaknesses of MHs, they fail to account for the significance of problem instances, as different instances can produce varying results in an MH. Our approach seeks to tackle this challenge, functioning as a complementary tool rather than a rival to current strategies. Thus, we present a novel integration that utilizes LLMs to enhance the effectiveness of metaheuristic search processes. Although a recent study has shown that LLMs can detect patterns across various tasks [32], no method has yet been developed that employs LLMs as *pattern recognition engines* in combinatorial optimization problems. Acknowledging the challenges related to LLMs (discussed in the subsequent Section II-B1), we identify numerous opportunities for progress in this area (see Section IV).

1) OBSTACLES AND OPPORTUNITIES

LLMs are an emerging technology that is still evolving. Despite demonstrating usefulness across various applications, as seen above, we believe that our research has mitigated two key risks associated with LLMs:

1) Training these models requires an immense amount of diverse data, from social media posts to books, and an equally vast amount of computing power. As a result, the leaders in the LLM industry tend to be well-funded private companies with significant resources and high valuations. This creates a high barrier to entry for startups or under-funded research centers, which often lack the necessary infrastructure to compete directly with these powerful players. To reduce these risks, a dual strategy can be implemented. On one side, this involves using more compact, open-source LLMs, which may, however, result in lower-quality outputs. On the other side, this entails using proprietary LLMs as software-as-a-service to end-users, which incurs financial expenses. For this research, we employed a mix of both strategies, highlighting their respective benefits and drawbacks.
2) LLMs have no built-in understanding of the world, as they cannot directly experience or "simulate" our environment. This is unlike humans, who constantly perceive information, be it sensory, visual, or auditory. In contrast, LLM operation depends on the data with which they are trained. Consequently, if the data is not meticulously selected by humans, the model's predictions may be unreliable or even generate false information, a phenomenon known as "hallucinations" in AI discourse.[8] We mitigated the problem of hallucinations with a special focus on prompt design. It has been observed that ambiguous prompts can lead to inconsistent results, with the same prompt potentially yielding different responses each time it is executed. As LLMs have evolved and increased in size, it has become evident that altering the prompt strategy can significantly improve the quality of the responses. This discovery has opened up new possibilities, enabling LLMs to tackle tasks that previously yielded negative or untested results. Carefully designing and adjusting prompts can improve the reliability and performance of LLMs [34]. This article focuses on creating prompts that produce desired outcomes.

III. PROBLEM DEFINITION

In this section, we provide the definition of the optimization problem we consider as an example in this paper, while in Section IV, we present our hybrid optimization approach. The considered optimization problem is from the realm of social networks. In fact, social network problems often serve as an interesting experimental laboratory for testing optimization techniques since they can be modeled as combinatorial problems based on directed graphs that become highly complex as the instance size grows.

Specifically, we consider Multi-Hop Influence Maximization, a social network problem proven NP-hard by Ni et al. [35] and Basuchowdhuri et al. [36]. In the literature, this problem has been studied using both metaheuristics and a state-of-the-art combination of a BRKGA with deep learning (DL) [19], which gives us a point of comparison.

A. MULTI-HOP INFLUENCE MAXIMIZATION IN SOCIAL NETWORKS

Many optimization problems in social networks can be formalized by modeling the social network as a directed graph $G = (V, A)$, where V represents the set of nodes and A represents the set of directed arcs. This is also the case

[8]The term "hallucination" in the context of LLMs was derived from the concept of "AI hallucination," which refers to incorrect responses generated by AI systems. In fact, this term was adopted due to the tendency of humans to anthropomorphize technology, attributing human qualities to it. As Floridi and Nobre have recently shown [33], such a tendency is common with disruptive technologies. It has also been shown that, as we become more familiar with these technologies, the tendency to anthropomorphize should decrease over time.

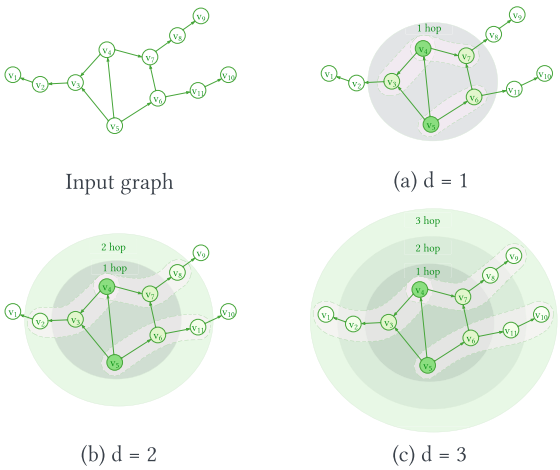

FIGURE 1. Multi-hop influence process. The given directed graph consists of 11 nodes and 12 arcs, and the task is to solve the k-dDSP with $k = 2$. The example solution U consists of two nodes: v_4 and v_5 (colored in green). The bottom row illustrates the concept of d-hop coverage: when $d = 1$, nodes v_3, v_7, v_6 are 1-hop covered by U; when $d = 2$, nodes $v_2, v_3, v_7, v_8, v_6, v_{11}$ are 2-hop covered by U; and when $d = 3$, all remaining nodes in the graph are 3-hop covered by U.

of the specific multi-hop influence maximization problem addressed in this paper, referred to as the k-d-Dominating Set Problem (k-dDSP).

The most crucial concept in this context is the *influence* $I_d(u) \subseteq V$ of a node $u \in V$, which is determined by two factors:

1) Parameter $d \geq 1$, which is an input to the problem and represents the maximum distance of influence.
2) A distance measure $dist(u, v)$ between nodes u and v. In this paper, $dist(u, v)$ is defined as the length (in terms of the number of arcs) of the shortest directed path from u to v in G.

Based on these factors, we can define the influence of a node u as follows:

$$I_d(u) := \{v \in V \mid dist(u, v) \leq d\} \tag{1}$$

In other words, $I_d(u)$ represents the set of all nodes in G that can be reached from u via a directed path with at most d arcs. We say that u influences (or covers) all nodes in $I_d(u)$. This definition can be naturally extended to sets of nodes as follows:

$$I_d(U) := \bigcup_{u \in U} I_d(u) \quad \forall U \subseteq V \tag{2}$$

That is, $I_d(U)$ represents the set of all nodes in G that are influenced by at least one node from the set U.

Valid solutions to the k-dDSP are all sets $U \subseteq V$ such that $|U| \leq k$, meaning that any valid solution can contain at most k nodes. The objective of the k-dDSP is to find a valid solution $U^* \subseteq V$ such that $|I_d(U^*)| \geq |I_d(U)|$ for all valid solutions U to the problem. In other words, the objective function value of a valid solution U is $|I_d(U)|$. Formally, the k-d DSP can be stated as follows:

$$\max_{U \subseteq V} |I_d(U)| \quad \text{s.t. } |U| \leq k \tag{3}$$

For an intuitive explanation of k-dDSP, see the toy example in Figure 1.

IV. INTEGRATION OF LLM OUTPUT INTO A METAHEURISTIC

Figure 2 depicts the framework of our proposed `MH+LLM` integration, comprising three automatic sequential steps:

1) **Prompt generation and execution by an LLM.** We begin by phrasing the k-dDSP in natural text and creating a small random graph with a high-quality solution. Next, we calculate five key metrics for each node of the graph, which enables the LLM to determine the most relevant metrics for this problem. We then compute the same metrics for a second (larger) graph in which we want to solve the k-dDSP problem. This graph is henceforth called the *evaluation graph*. Using the generated data, we design a prompt and ask the LLM to provide parameters for calculating the importance of each node in the evaluation graph. In essence, we leverage the LLM as a *pattern recognition engine* to identify correlations between node metrics and node importance in the context of the k-dDSP.
2) **Calculate probabilities for each node of the evaluation graph.** As explained in detail below, the LLM provides values for ten parameters that can be used to compute the probability of each node of the evaluation graph to form part of an optimal k-dDSP solution. We expect this information to offer excellent guidance to a MH.
3) **Utilizing the probabilities (guidance) within a MH.** Since the MH we use in this work is a BRKGA, we incorporate the probabilities calculated based on the LLM output into the decoder that translates random keys into valid solutions to the tackled optimization problem.

In what follows, these three steps are detailed in corresponding subsections.

A. PROMPT ENGINEERING

A prompt is an instruction provided as input to an LLM. The design of a prompt can greatly influence the quality of the LLMs' responses [6], [7]. Also, the specific response can vary depending on the LLM used. From the available prompt design techniques, we have opted for one-shot learning [37], also sometimes called few(1)-shot learning. According to Chen's findings [38], tabular data reasoning can achieve good results with a single example, eliminating the need for additional examples or fine-tuning. This method involves furnishing the LLM with an example to facilitate

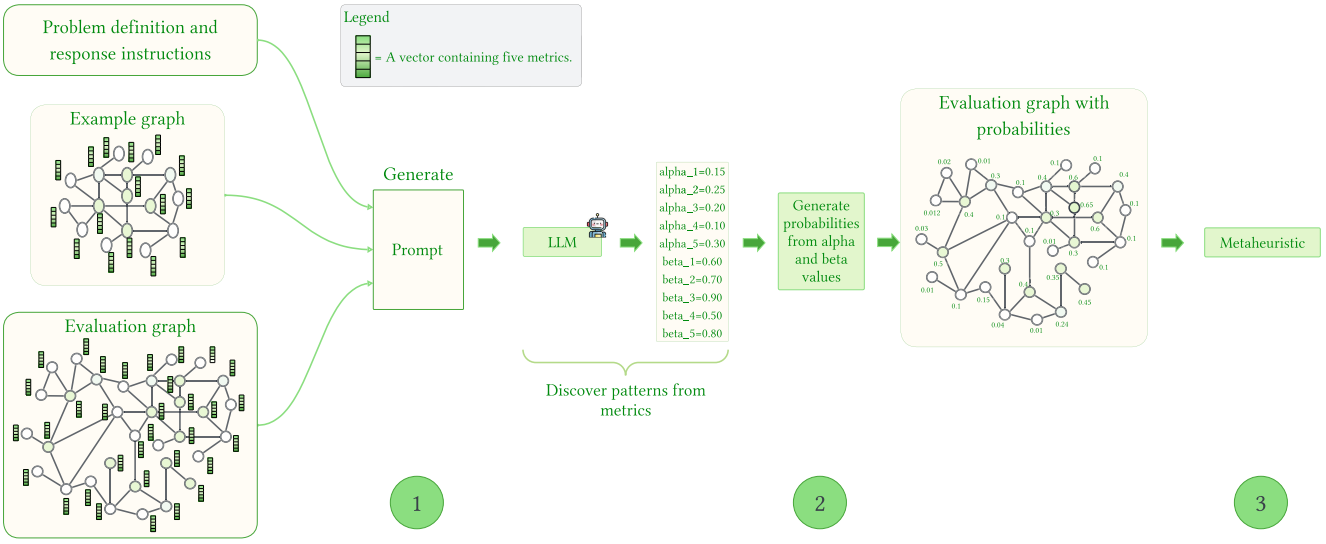

FIGURE 2. An overview of our approach to integrating MHs and LLMs: We employ LLMs to analyze problem instances and uncover hidden patterns. The patterns are then converted into useful information that guides the MH in its search for high-quality solutions.

pattern recognition in the response instructions that are to be requested.

1) DEFINITION

The prompt we have designed consists of four tags, defined as follows:

$$P := \text{prompt}(\textsf{Tag1}, \textsf{Tag2}, \textsf{Tag3}, \textsf{Tag4}) \tag{4}$$

where

- **Tag1** is the **[PROBLEM]** tag,
- **Tag2** is the **[EXAMPLE GRAPH]** tag,
- **Tag3** is the **[EVALUATION GRAPH]** tag, and
- **Tag4** is the **[RULES ANSWERING]** tag.

Hereby, the **[PROBLEM]** tag contains the description of the k-dDSP. Moreover, the example graph information is provided in the **[EXAMPLE GRAPH]** tag. Hereby, the example graph consists of 100 nodes, each characterized by the values of five metrics: **in-degree**, **out-degree**, **closeness**, **betweenness**, and **pagerank**.[9] In particular, these values are henceforth denoted by

$$m^{ex}_{i,1}, m^{ex}_{i,2}, m^{ex}_{i,3}, m^{ex}_{i,4}, m^{ex}_{i,5} \quad \text{for all } v_i \text{ of the example graph.} \tag{5}$$

Hereby, $m^{ex}_{i,1}$ is the value corresponding to metric **in-degree**, $m^{ex}_{i,2}$ corresponds to **out-degree**, etc. Note also that the metric values are normalized to the range [0, 1]. Furthermore, the high-quality k-dDSP solution of the example graph is computed using the pure BRKGA algorithm, which we adopted from our earlier work [19]. The solution is encoded as a vector of 32 nodes, separated by commas, corresponding to the k-dDSP parameter $k = 32$.

Next, the **[EVALUATION GRAPH]** tag contains the evaluation graph for which the k-dDSP must be solved. Each node of this graph is described by the values of the same five metrics described above. These evaluation graph values are henceforth denoted by

$$m^{eval}_{i,1}, m^{eval}_{i,2}, m^{eval}_{i,3}, m^{eval}_{i,4}, m^{eval}_{i,5} \quad \text{for all } v_i \text{ of the evaluation graph.} \tag{6}$$

Finally, the **[RULES ANSWERING]** tag specifies the details of the request to the LLM, which will be elaborated on in Section IV-A2.

After the prompt P is formulated, it is utilized by invoking the execute function, which takes three parameters: the prompt P, the selected LLM, and Θ, representing a set of values for the configuration parameters of the LLM. This results in the corresponding LLM output:

$$Output := \text{execute}(P, LLM, \Theta) \tag{7}$$

Specifically, Θ contains values for exactly two hyperparameters, regardless of the utilized LLM. The first, known as **temperature**, is a value between 0 and 1 that measures the model's response uncertainty, with lower values indicating a more deterministic output. While "more deterministic" does not imply that the LLM will generate identical responses to the same prompt every time, it does enhance the stability of the outputs, making it easier to replicate experiments [39].

[9] We selected these five metrics based on our understanding of the k-dDSP problem. However, in future work, leveraging the extensive knowledge LLMs possess from academic sources, we plan to use them to guide the selection of metrics for each specific problem. This could represent the extension of our method.

Therefore, we set the **temperature** to 0.[10] The second hyper-parameter is the **maximum number of output tokens**, which we have set to a moderate 1000 tokens. This choice is based on the prompt design, which consistently yields relevant outputs regardless of the evaluation graph's size, ensuring that the quality of the results is not compromised by a smaller token limit.

2) PROMPT STRUCTURE

Effective prompts are generally those with few language ambiguities. To achieve this, the four unique opening and closing tags mentioned in the previous section provide structure and coherence. We will now clarify the syntactic structure of each of these tags. A complete example of a prompt, along with each tag, can be found in Figure 3.

1) **Problem description.** The prompt starts by providing a concise definition of the k-dDSP utilizing LaTeX notation within the **[PROBLEM]** tag; see the top right of Figure 3.
2) **Example Graph.** The **[EXAMPLE GRAPH]** tag, as the name suggests, provides information about the example graph. Nestled within this tag are two additional tags: **[DATA]**, encompassing the metric values of each node of the example graph, and **[ANSWER]**, which provides a high-quality solution for the given graph.
   - **[DATA] tag**: A (directed) random graph with 100 nodes produced with the Erdös–Rényi model [40] was chosen as an example graph. The edge probability of the graph was 0.05. Subsequently, five before-mentioned metrics were calculated for each of the 100 nodes and incorporated into the prompt in a tabular data format, with rows and columns separated by commas. Each row corresponds to a node ID, while the columns represent the respective metric values for that particular node. The metric values are presented in scientific numerical notation to minimize token usage. The rationale behind this decision is discussed in the context of the empirical results; see Section V.
   - **[ANSWER] tag**: The solution to the example graph is computed using the BRKGA algorithm from [19]. However, note that this solution (which is not necessarily optimal) could potentially have been achieved through alternative means, such as employing a different metaheuristic or solving the problem via an exact method. The rationale behind including a high-quality solution is our expectation that—given the nodes belonging to a presumably high-quality solution—the LLM will be able to discern which metrics are more crucial than others and how the metric values of selected nodes interrelate.
3) **Evaluation Graph.** The **[EVALUATION GRAPH]** tag, much like the **[EXAMPLE GRAPH]** tag, utilizes a nested **[DATA]** tag to store the values of the five metrics for every node. However, we obviously do not provide any solution for the evaluation graph. This is because the objective is to request information from the LLM on the probability of nodes from the evaluation graph to pertain to an optimal solution.
4) **Rules Answering.** The **[RULES ANSWERING]** tag is crucial as it ties together all the information provided in the previous tags. In this part of the prompt, an equation is presented to the LLM to calculate the probability of each node of the evaluation graph to form part of an optimal solution. The equation requires 10 parameters: 5 alpha parameters and 5 beta parameters, which will be explained in more detail in Section IV-B. These parameters serve to assign weights to the metrics and correct potential errors. The LLM infers the values of these parameters by analyzing the metrics in the **[EVALUATION GRAPH]** tag and using the metrics and the solution from the **[EXAMPLE GRAPH]** tag as a guide.

For optimal prompt construction, we recommend using our `OptiPattern` tool, which generates the prompt automatically.

B. LLM OUTPUT

As described before, a prompt P provides the values of the following five metrics for each node of the example graph and the evaluation graph: **in-degree**, **out-degree**, **closeness**, **betweenness**, and **pagerank**. It is assumed that the most important metric for addressing the k-dDSP is the **out-degree**, that is, the number of neighbors that can be reached from a node via directed arcs. A node with a higher out-degree is generally more likely to form part of high-quality solutions. However, we assume that there are additional metrics (among the other four metrics) that might contribute valuable information. Consequently, we anticipate that the LLM will be able to identify this. To identify patterns in the values provided by the metrics, the LLM is requested (by means of the **[RULES ANSWERING]** tag) to return values for two sets of five parameters (one for each metric, in the order as given above), resulting in ten values. More specifically, upon executing a prompt P, the chosen LLM produces a set *Output* (see Eq. (7)) which is as follows:

$$Output = \{\alpha_1, \ldots, \alpha_5, \beta_1, \ldots, \beta_5\} \tag{8}$$

The first five of these values are henceforth called alpha values, while the last five values are named beta values. The heart concept of the proposed prompt is centered on the meaning of these values and how they are utilized.

- **alpha values**: These are weights that indicate the influence of each metric. The total sum of all alpha

[10] For text generation and paraphrasing tasks, it's recommended to increase the **temperature**. This adjustment promotes creativity, which is a desirable characteristic in these contexts. These applications differ from using the LLM as a pattern recognition tool, where consistent and stable outputs are preferable.

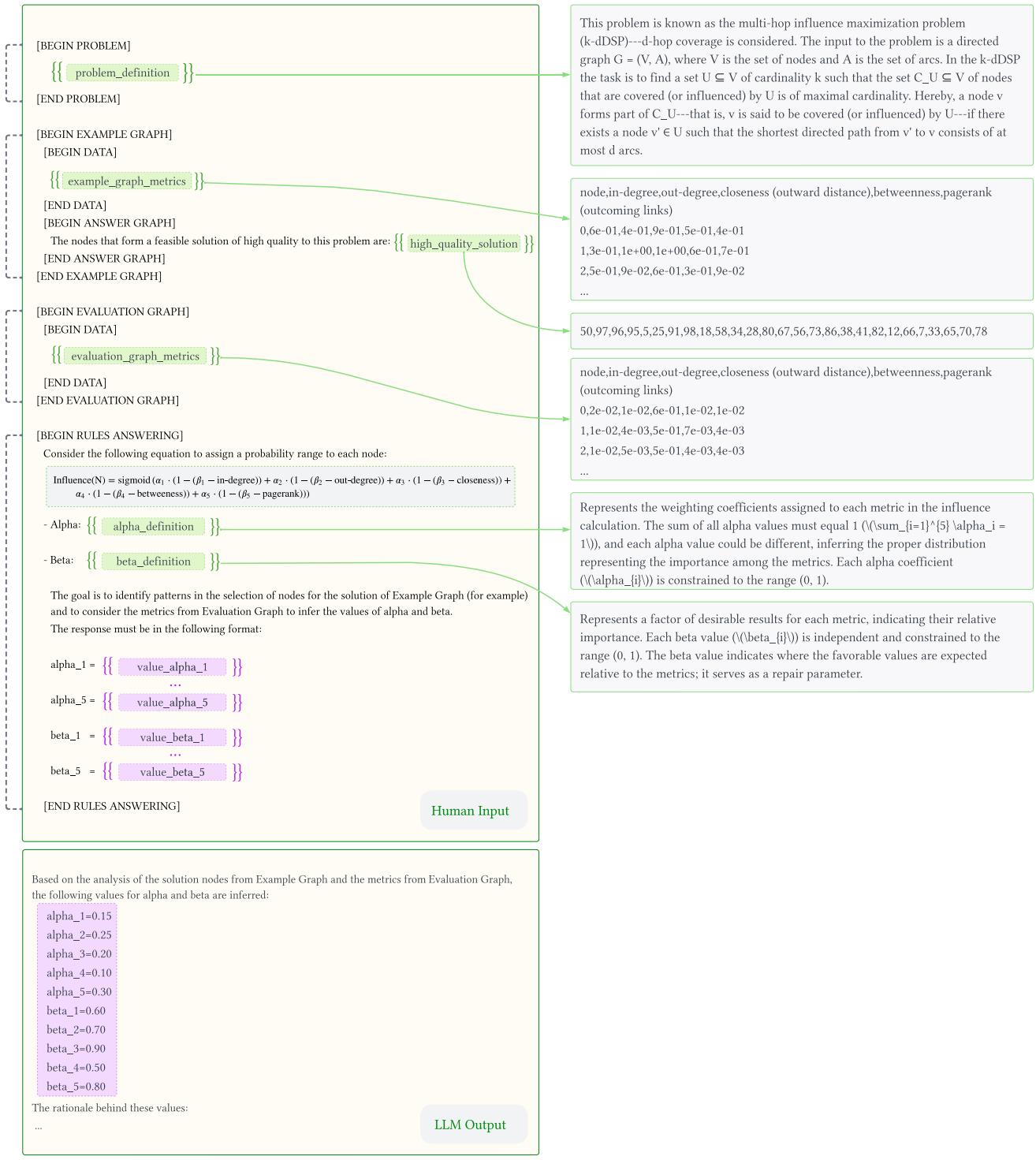

FIGURE 3. An example of a prompt and the corresponding LLM response. The prompt includes the problem definition, a graph example with node metrics and a high-quality solution, an evaluation graph, and instructions for the LLM for producing the output. Based on the patterns identified in the evaluation graph, the LLM provides the importance of each metric, represented by the set of alpha and beta values.

values should be equal to one ($\sum_{i=1}^{5} \alpha_i = 1$), and each alpha value can be unique. In other words, the alpha value $0 < \alpha_i < 1$ reflects the relative significance of the i-th metric (in the order as mentioned above).

- **beta values**: These five values are adjustment (or correction) parameters. Unlike the alpha values, beta values $0 < \beta_i < 1$ are independent of each other. Moreover, beta values do not represent relative weights

among the metrics. They rather indicate the best possible value of a node regarding a metric. This allows the LLM to identify where the best values are found with respect to their range [0, 1].

Based on these values from the LLM output, the probability for a node v_j of the evaluation graph is determined using the following formula:

$$\mathbf{p}^{\text{LLM}}(v_j) := \sigma\left(\sum_{i=1}^{5} \alpha_i \cdot (1 - (\beta_i - m_{j,i}^{eval}))\right) \quad (9)$$

Note that this formula introduces non-linearity into the node probabilities by applying the sigmoid function σ, which enables a more nuanced representation of the probability space.[11]

As shown in Figure 3, our proposed prompt thoroughly explains the alpha and beta values to the LLM, along with Eq. 9. By giving the LLM a clear understanding of the context surrounding the alpha and beta values, we simply ask the LLM to provide the corresponding values for the evaluation graph.

C. USING LLM OUTPUT TO GUIDE A METAHEURISTIC

In this section, we first describe the metaheuristic considered to test the quality of the LLM output. Subsequently, the way of incorporating the probability values into the metaheuristic is outlined.

1) THE CONSIDERED METAHEURISTIC: A BRKGA

The chosen metaheuristic is a so-called Biased Random Key Genetic Algorithm (BRKGA) designed for solving the k-d DSP in [19].[12] In fact, two algorithm variants were proposed in [19]: (1) a pure BRKGA variant and (2) an algorithm variant that makes use of a hand-designed deep learning framework for biasing the BRKGA. This will allow us to compare our proposal properly to existing algorithm versions.

In general, a BRKGA is problem-independent because it works with populations of individuals that are vectors of real numbers (random keys). The problem-dependent part of each BRKGA deals with how individuals are translated into solutions to the tackled problem. The problem-independent pseudocode of BRKGA is provided in Algorithm 1.

The algorithm begins by calling GenerateInitialPopulation(p_{size}, $seed$) to create a population P of p_{size} individuals. If $seed = 0$, all individuals are randomly generated, with each $\pi \in P$ being a vector of length $|V|$ (where V is the set of nodes from the input graph). The value at position i of π, $\pi(i)$, is randomly chosen from [0, 1] for all $i = 1, \ldots, |V|$. If $seed = 1$, $p_{size} - 1$ individuals are randomly generated, and the last individual is obtained by setting $\pi(i) := 0.5$ for all $i = 1, \ldots, |V|$. The initial population's individuals are then evaluated by transforming each individual $\pi \in P$ into a valid solution $U_\pi \subset V$ to the k-dDSP, with the value $f(\pi)$ defined as $f(\pi) := |U_\pi|$. The transformation process is discussed later.

Algorithm 1 The Pseudocode of BRKGA

Require: a directed graph $G = (V, E)$
Ensure: values for parameters $p_{size}, p_e, p_m, prob_{elite}, seed$
1: $P \leftarrow$ GenerateInitialPopulation(p_{size}, $seed$)
2: Evaluate(P) ▷ problem-dependent part (greedy)
3: **while** computation time limit not reached **do**
4: $P_e \leftarrow$ EliteSolutions(P, p_e)
5: $P_m \leftarrow$ Mutants(P, p_m)
6: $P_c \leftarrow$ Crossover(P, p_e, $prob_{elite}$)
7: Evaluate($P_m \cup P_c$) ▷ problem-dependent part (greedy)
8: $P \leftarrow P_e \cup P_m \cup P_c$
9: **end while**
10: **return** Best solution in P

At each iteration, the algorithm performs the following operations:

1) The best $\max\{\lfloor p_e \cdot p_{size} \rfloor, 1\}$ individuals are copied from P to P_e using EliteSolutions(P, p_e).
2) A set of $\max\{\lfloor p_m \cdot p_{size} \rfloor, 1\}$ mutants are generated by function Mutants(P, p_m) and stored in P_m. These mutants are random individuals generated the same way as those from the initial population.
3) A set of $p_{size} - |P_e| - |P_m|$ individuals are generated by crossover using Crossover(P, p_e, $prob_{elite}$) and stored in P_c. The crossover, which involves combining two solutions, serves as the mechanism that enhances the search process, concentrating on transferring the superior traits of parents to their offspring.

The evaluation of an individual (see lines 2 and 7 of Algorithm 1) is the crucial problem-dependent aspect of the BRKGA algorithm from [19]. This evaluation function—often termed the *decoder*—utilizes a straightforward greedy heuristic. The heuristic is based on the notion that nodes with a higher out-degree—that is, more neighbors—are likely to yield a higher influence.

For a node $v_j \in V$, the set of neighbors, $N(v_j)$, comprises nodes reachable via a directed arc from v_j: $N(v_j) = \{v_i \in V \mid (v_j, v_i) \in A\}$. The greedy value $\phi(v_j)$ for each $v_j \in V$ is calculated as:

$$\phi(v_j) := |N(v_j)| \cdot \pi(j) \quad (10)$$

In other words, in this equation, the greedy value of a node v_j is determined by multiplying its out-degree with the corresponding numerical value from the individual being translated into a solution. The final solution, U_π, is obtained by selecting the k nodes with the highest greedy values.

The following will modify the greedy function ϕ to create our hybrid algorithm.

[11] The sigmoid function has been used for many purposes in neural networks. But also in metaheuristics, for example, for significantly accelerating the convergence of a genetic algorithm [41].

[12] BRKGA's are well-known GA variants, mostly for solving combinatorial optimization problems.

2) HYBRID ALGORITHM

The proposed hybrid algorithm—referred to as BRKGA+LLM—begins with two offline steps. Given an evaluation graph $G = (V, A)$, a prompt is generated as outlined in the previous section and sent to an LLM. Based on the alpha and beta values obtained from the LLM's response, the probability $\mathbf{p}^{\text{LLM}}(v_j)$ for each node $v_j \in V$ is determined using Eq. (9). Next, the original greedy function $\phi()$ from Eq. (10) is substituted with a modified version that integrates the node probabilities derived from Eq. (9):

$$\phi_{\text{Influence}_{LLM}}(v_j) := |N(v_j)| \cdot \pi(j) \cdot \mathbf{p}^{\text{LLM}}(v_j) \quad \forall v_j \in V \quad (11)$$

We hypothesize that with suitable predictions from the LLM, the algorithm can be guided/biased to explore more promising areas of the search space. These areas are believed to contain high-quality solutions that the BRKGA would have been unable to discover on their own without the guidance of these predictions. In other words, using an LLM to discover patterns in metric values (see Section IV-A), rather than relying solely on the out-degree, might enhance the algorithm's performance.

V. EMPIRICAL EVALUATION

This section presents empirical evidence demonstrating the benefits of integrating MHs and LLMs. The following algorithm variants are considered for the comparison:

- `BRKGA`: the pure BRKGA variant already published in [19] and described above in Section IV-C1.
- `BRKGA+FC`: the BRKGA hybridized with a hand-designed GNN called `FastCover` (FC) that was used to derive the probability values (last term of Eq. (11)) in [19].
- `BRKGA+LLM`: the BRKGA enhanced with LLM output as described in the previous section.

Note that both `BRKGA` and `BRKGA+FC` underwent a general parameter tuning in [19] depending on the value of k. The well-known `irace` tool [42] was used for this purpose. In this work, we adopt the corresponding parameter settings of `BRKGA` for `BRKGA+LLM`. In this way, we can be sure that any difference in their performance is caused by the guidance of the probabilities computed from the LLM outputs. In any case, a specific tuning of `BRKGA+LLM` could only further improve its results.

Apart from comparing the three approaches mentioned above, we show results for different LLMs and provide evidence for the quality of LLM output. Additionally, we support our analysis with a visual examination, providing additional insight into why the hybrid `BRKGA+LLM` outperforms the other algorithm variants.

A. EXPERIMENTAL SETUP

The BRKGA was implemented in `C++`, whereas the prompt construction process, which entails extracting metrics from graph instances, was conducted using Python 3.11. Regarding the choice of LLMs, we utilized two proprietary language models, GPT-4o and Claude-3-Opus, as well as two open-source models, Command-R+ and Mixtral-8 × 22b-Instruct-v0.1. We selected these models based on the Chatbot Arena—a platform developed by LMSYS members and UC Berkeley SkyLab researchers—which provides an Arena Leaderboard,[13] a community-driven ranking system for LLMs [43].[14] Table 1 presents a comprehensive overview of the models, including their ranking in the Chatbot Arena Leaderboard (as of May 2024), corresponding version numbers, licenses, maximum context windows, and crucially, the test environment employed for each model.

1) EXECUTION ENVIRONMENT

We utilized the OpenRouter API[15] to execute prompts in their corresponding LLMs, except for Claude-3-Opus, which we used through the Anthropic API (see Table 1). Finally, all experiments involving the three BRKGA variants were conducted on a high-performance computing cluster comprising machines powered by Intel Xeon CPU 5670 processors with 12 cores running at 2.933 GHz and a minimum of 32 GB of RAM.

2) DATASET

Our evaluation is based on two categories of k-dDSP instances (evaluation graphs). The first consists of rather small, synthetic social network graphs with 500 and 1000 nodes, generated using three configuration methods developed by Nettleton [44]. The corresponding graph generator requires four real-valued parameters, whose values are reflected by the instance names.[16] The second instance set comprises four real-world social network graph instances obtained from the well-established SNAP (Stanford Network Analysis Project) repository [45]. Moreover, note that the k-dDSP can be solved in each graph for different values of d and k. In this work we solved all evaluation graphs with $d \in \{1, 2, 3\}$ and $k \in \{32, 64, 128\}$. All datasets (synthetic and real), prompts, and results can be found in the `supplementary material/` folder in the repository: https://github.com/camilochs/optipattern.

Restrictions. The size of the graphs poses a constraint on the prompts we have designed for the LLMs, which is limited by two factors:

[13] https://lmarena.ai

[14] Please be aware that our experiments took place between February and May 2024, and the LLMs ranking classification in Chatbot Arena may have changed by the time of reading.

[15] https://openrouter.ai

[16] The instance names are obtained by a concatenation of the utilized parameter values: examples are **0.4-0.15-0.15-0.3**, **0.3-0.0-0.3-0.4**, and **0.2-0.0-0.3-0.5**. The parameters labeled a, b, c, and d, respectively, define communities weights (a and d) and link weights between communities (b and c), $a + b + c + d \approx 1$. These parameters influence the topology of the network, specifically the total number of connections and the density.

TABLE 1. Summary of the assessed LLMs, which have been used via the OpenRouter API. This is except for Claude-3-Opus, the first LLM considered. At that point, we had yet to become familiarized with OpenRouter.

Model	Chatbot Arena Ranking	Version	License	Maximum Context Window	Test Environment (API)
OpenAI/GPT-4o	#1	may2024	private	128,000	OpenRouter
Anthropic/Claude-3-Opus	#2	march2024	private	200,000	Anthropic
Cohere/Command-R+	#10	april2024	CC-BY-NC-4.0	128,000	OpenRouter
MistralAI/Mixtral-8x22b-Instruct-v0.1	#19	april2024	Apache 2.0	32,768	OpenRouter

TABLE 2. Number of input/output tokens and the associated cost of processing the input prompts concerning Claude-3-Opus. The costs correspond to March 2024.

Instance	Input (tokens)	Output (tokens)	Cost (USD/EUR)
soc-wiki-elec	181,719	463	2,78/2,58
soc-advogato	160,812	410	2,46/2,28
sign-bitcoinotc	66,406	303	1,02/0,95
soc-hamsterster	17,097	438	0,29/0,27

1) The maximum context window of LLMs is still relatively small.[17] For instance, the largest evaluation graph we use, **soc-wiki-elec**, results in an input prompt size of 181 719 tokens, which is close to the 200 000 tokens limit of Claude-3-Opus [3], the LLM which offers the currently largest context window.
2) The cost of processing larger instances is prohibitively high. For example, executing the prompt regarding the **soc-wiki-elec** evaluation graph on Claude-3-Opus exceeds €2.5.

Table 2 provides a detailed breakdown of the constraints for the largest evaluation graphs considered in this work. Although these limitations currently restrict us to testing with smaller instances, we anticipate that this constraint will soon be alleviated as the maximum context window increases and processing costs decrease (see Section VI for more information on this).

B. ANALYSIS OF LLM OUTPUT

This section aims to validate the outputs of the LLMs and their usefulness for utilizing them as guidance within the BRKGA algorithm. The aim is to demonstrate that they are not arbitrary or devoid of significance. We have conducted three sets of experiments to achieve this, aiming at different aspects. Figure 4 shows the custom-designed, three-dimensional experimental framework developed specifically for this evaluation.[18]

Before starting with the main experimental evaluation, we must choose the most suitable LLM for the considered task. For this purpose, we produced all prompts concerning the six synthetic graphs from the dataset (as described before) for all considered combinations of $d \in \{1, 2, 3\}$ and $k \in \{32, 64, 128\}$. These prompts were fed into the following LLMs: GPT-4o, Claude-3-Opus, Command-R+, and Mixtral-8 × 22b-Instruct-v0.1. The obtained probabilities were then directly used to produce solutions containing the k nodes with the largest probability values. The same was done concerning metric **out-degree**. That is, solutions were produced that consist of the k nodes with the highest **out-degree** values. The results shown in Table 3 allow us to make the following observations. First, although no LLM always outperforms the other LLMs, Claude-3-Opus shows advantages over the other LLMs, especially for increasing values of k. When comparing the results obtained with **out-degree** (the most popular greedy heuristic for the k-dDSP) to the results obtained with Claude-3-Opus, we can observe that **out-degree** seems to work slightly better for $k = 32$, while the opposite is the case for $k = 128$. Based on these results, we use Claude-3-Opus for the remainder of our experiments.

1) DIMENSION 1 OF THE EVALUATION FRAMEWORK: RESULT QUALITY

In the first set of experiments, we decided to compare the pure `BRKGA` approach with `BRKGA+LLM` in the context of the six synthetic graphs (and for all combinations of $d \in \{1, 2, 3\}$ and $k \in \{32, 64, 128\}$). Both algorithms were applied 10 times to each case, with a computation time limit of 900 CPU seconds per run. The results, which are shown in Table 4, clearly show that `BRKGA+LLM` outperforms the pure `BRKGA` algorithm most of the time. Considerable improvements can be observed in the context of the last graph; see instance **0.2-0.0-0.3-0.5** with 1000 nodes and 8000 arcs. Only in three cases, the result obtained by `BRKGA+LLM` is slightly inferior to the one of `BRKGA`. While

[17] The maximum context window of an LLM sets the maximum amount of text it can process simultaneously when generating a response. This constraint determines the scope of contextual information the LLM can draw upon when answering a question or completing a task. The response quality will likely degrade if the input prompt exceeds this limit. Given that our prompt design requires each metric for each node to be equally important, the LLM needs to consider as much context as possible to deliver reasonable and trustworthy results.

[18] Future research could expand the framework's dimensions to better justify LLMs' response quality and integration with MHs and to address unexplored aspects.

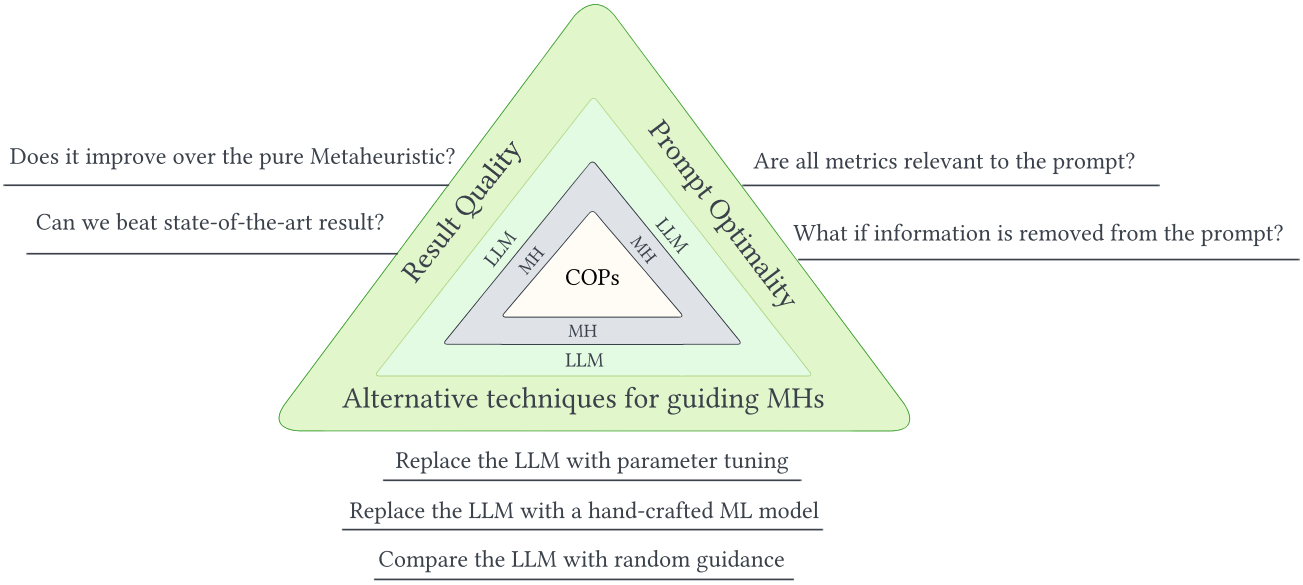

FIGURE 4. A comprehensive evaluation framework was used to assess the usefulness of integrating MHs with LLMs for solving combinatorial optimization problems (COPs) across the three dimensions shown in the graphic.

TABLE 3. Solution qualities obtained when turning the probabilities computed based on the LLM's output directly into solutions. In addition, the same is done for the **out-degree** metric. Considered LLMs are GPT-4o, Claude-3-Opus, Command-R+, and Mixtral-8 × 22b-Instruct-v0.1. The six synthetic graphs are chosen as a test bed. Green cells highlight the best LLM results, while gray cells indicate the top **out-degree** metric results.

				$k = 32$					$k = 64$					$k = 128$				
Instance	$\|V\|$	$\|E\|$	d	**GPT-4o**	**Claude-3-Opus**	**Command-R+**	**Mixtral-8x22b**	out-degree	**GPT-4o**	**Claude-3-Opus**	**Command-R+**	**Mixtral-8x22b**	out-degree	**GPT-4o**	**Claude-3-Opus**	**Command-R+**	**Mixtral-8x22b**	out-degree
0.4-0.15-0.15-0.3	500	3000	1	240	241	240	241	**256**	338	339	332	330	**361**	425	428	430	428	**439**
			2	458	458	**461**	457	**461**	481	481	483	479	**485**	**494**	**494**	**494**	**494**	492
			3	**494**	**494**	**494**	**494**	493	494	495	**496**	494	495	**496**	**496**	**496**	**496**	**496**
0.3-0.0-0.3-0.4	500	3000	1	302	316	308	288	**323**	391	**393**	392	375	384	435	**449**	439	429	448
			2	366	377	371	358	**379**	416	**421**	418	404	410	442	**455**	447	439	453
			3	369	380	374	360	**381**	416	**421**	418	404	410	442	**455**	447	439	453
0.2-0.0-0.3-0.5	500	3000	1	164	**168**	163	163	155	236	**250**	244	244	225	321	**332**	331	331	297
			2	198	**202**	201	201	179	253	**269**	264	264	237	328	339	**340**	**340**	301
			3	202	**206**	203	203	179	254	**269**	264	264	237	328	339	**340**	**340**	301
0.4-0.15-0.15-0.3	1000	8000	1	381	379	378	n.a.	**392**	534	551	552	n.a.	**560**	727	732	734	n.a.	**748**
			2	926	925	925	n.a.	**931**	**973**	971	971	n.a.	**973**	988	988	989	n.a.	**990**
			3	992	992	991	n.a.	**993**	993	993	993	n.a.	**994**	994	**995**	**995**	n.a.	**995**
0.3-0.0-0.3-0.4	1000	8000	1	523	561	509	n.a.	**571**	701	720	692	n.a.	**723**	823	**848**	819	n.a.	842
			2	743	**762**	728	n.a.	755	804	810	799	n.a.	**812**	857	**880**	855	n.a.	874
			3	758	773	747	n.a.	**775**	808	815	803	n.a.	**819**	858	**880**	856	n.a.	874
0.2-0.0-0.3-0.5	1000	8000	1	232	**237**	230	n.a.	223	343	**349**	341	n.a.	316	501	**512**	**512**	n.a.	470
			2	308	311	**322**	n.a.	269	394	399	**402**	n.a.	343	526	535	**537**	n.a.	482
			3	313	320	**328**	n.a.	272	396	401	**405**	n.a.	345	526	535	**537**	n.a.	482

these results are promising, it is important to recognize that they are based on relatively small instances.

In the next set of experiments, we applied the BRKGA+FC [19], in addition to BRKGA and BRKGA+LLM, to the four larger real-world social networks. Remember that BRKGA+FC is a hybrid approach that uses a hand-crafted GNN approach for biasing the search process of BRKGA. The results are shown—again for each combination of $d \in \{1, 2, 3\}$ and $k \in \{32, 64, 128\}$—in Table 5. The computation time limit for the three approaches was 900 CPU

TABLE 4. Comparison of the pure BRKGA with BRKGA+LLM on the six synthetic social networks. For each network, the algorithms were applied for each combination of $d \in \{1, 2, 3\}$ and $k \in \{32, 64, 128\}$. Average results over 10 algorithm runs are shown. Green cells indicate the best quality metrics results—higher values are better in this maximization problem.

				$k = 32$		$k = 64$		$k = 128$	
Instance	$\|N\|$	$\|E\|$	d	BRKGA	BRKGA+LLM	BRKGA	BRKGA+LLM	BRKGA	BRKGA+LLM
0.4-0.15-0.15-0.3	500	3000	1	309.6	**309.9**	433.9	**437.2**	499.0	**500.0**
			2	496.0	**499.9**	497.4	**500.0**	**500.0**	**500.0**
			3	496.0	**500.0**	498.0	**500.0**	**500.0**	**500.0**
0.3-0.0-0.3-0.4	500	3000	1	**353.7**	**353.1**	432.9	**434.1**	489.0	**498.8**
			2	**413.0**	**413.0**	453.0	**454.0**	489.0	**500.0**
			3	**417.0**	416.9	455.0	**456.0**	489.0	**500.0**
0.2-0.0-0.3-0.5	500	3000	1	203.8	**204.4**	287.6	**291.4**	373.0	**380.0**
			2	247.0	**252.6**	316.0	**323.8**	386.0	**394.0**
			3	247.0	**254.9**	316.0	**323.8**	386.0	**394.0**
0.4-0.15-0.15-0.3	1000	8000	1	446.6	**447.4**	679.4	**680.5**	907.6	**915.9**
			2	982.0	**985.0**	996.0	**1,000.0**	996.0	**1,000.0**
			3	**996.0**	**996.0**	996.0	**1,000.0**	996.0	**1,000.0**
0.3-0.0-0.3-0.4	1000	8000	1	604.0	**604.6**	773.8	**774.7**	908.7	**909.2**
			2	**808.8**	808.0	**880.0**	879.6	948.8	**949.8**
			3	**824.3**	**824.8**	888.6	**891.0**	955.0	**955.9**
0.2-0.0-0.3-0.5	1000	8000	1	296.0	**296.4**	438.0	**441.3**	596.1	**612.2**
			2	390.4	**404.7**	506.7	**523.2**	630.0	**658.9**
			3	404.1	**424.7**	511.8	**531.2**	635.7	**662.0**

TABLE 5. Numerical comparison of three algorithms—BRKGA, BRKGA+FC (results extracted from [19]), and our hybrid approach BRKGA+LLM—on a total of four real-world social network instances. For each network, the algorithms were applied 10 times to each combination of $d \in \{1, 2, 3\}$ and $k \in \{32, 64, 128\}$. Green cells indicate the best quality metrics results—higher values are better in this maximization problem.

				$k = 32$			$k = 64$			$k = 128$		
Instance	$\|V\|$	$\|E\|$	d	BRKGA	BRKGA+FC	BRKGA+LLM	BRKGA	BRKGA+FC	BRKGA+LLM	BRKGA	BRKGA+FC	BRKGA+LLM
soc-hamsterster	2426	16630	1	1,230.0	962.7	**1,238.9**	1,455.3	1,184.5	**1,478.0**	1,627.8	1,376.5	**1,731.3**
			2	1,751.0	1,682.1	**1,783.2**	1,779.6	1,778.7	**1,892.0**	1,811.0	1,857.0	**2,115.6**
			3	1,788.0	1,799.6	**1,876.0**	1,816.8	1,850.2	**1,947.3**	1,828.0	1,877.4	**2,180.2**
sign-bitcoinotc	5881	35592	1	**3,479.0**	**3,479.0**	**3,479.0**	4,040.3	4,041.0	**4,054.7**	4,599.9	4,618.0	**4,606.2**
			2	5,632.0	5,632.6	**5,650.2**	5,716.4	5,715.2	**5,752.2**	5,769.0	5,781.2	**5,835.1**
			3	5,838.0	5,838.0	**5,852.7**	5,839.0	5,839.1	**5,863.4**	5,842.0	5,844.0	**5,868.0**
soc-advogato	6551	51332	1	2,464.1	2,469.1	**2,485.9**	2,949.8	2,948.9	**2,952.2**	3,342.2	3,372.3	**3,385.1**
			2	4,142.6	4,132.3	**4,144.9**	4,208.7	4,207.8	**4,223.1**	4,268.5	4,251.1	**4,330.1**
			3	4,280.3	4,275.5	**4,318.6**	4,284.4	4,280.0	**4,359.9**	4,301.2	4,284.0	**4,431.9**
soc-wiki-elec	7118	107071	1	2,167.0	2,176.7	**2,188.0**	2,265.6	2,268.6	**2,286.1**	2,367.7	2,366.5	**2,408.8**
			2	2,354.7	2,355.1	**2,365.0**	2,390.0	2,388.0	**2,409.7**	2,454.5	2,427.5	**2,478.6**
			3	2,357.1	2,357.3	**2,366.2**	2,389.5	2,389.7	**2,406.4**	2,452.2	2,426.5	**2,474.2**

seconds, as in the previously outlined experiments. Moreover, the numbers in the tables are averages over 10 algorithm runs. The results show that BRKGA+LLM outperforms both approaches, with higher margins than those observed in the context of smaller synthetic networks. This holds especially for a growing value of k. This is interesting as the prompts only contained an example solution for $k = 32$. This indicates the LLM's ability to uncover meaningful patterns in the example graph, respectively, in the solution provided in the prompts.

Finally, we aimed to test how meaningful the LLM output really is. For this purpose, we produced two additional variants of BRKGA+LLM: the one called static is obtained by replacing the LLM output with probabilities obtained by random alpha and beta values. The second one, called dynamic, is a similar variant in which the LLM output is replaced with probabilities re-computed at each iteration based on newly determined random alpha and beta values. All three algorithm variants were applied 10 times to each of the four real-world social networks for each combination

of $d \in \{1, 2, 3\}$ and $k \in \{32, 64, 128\}$. The results averaged over the 10 runs are shown in Table 6. They clearly indicate that guidance by the LLM output is much more useful than random guidance.

We can conclude that guidance by LLM output clearly leads to an improved algorithm and results. In other words, these experiments demonstrate that an LLM can provide valuable information to inform a metaheuristic. In fact, we would expect an even higher benefit through LLM guidance in the context of even larger networks. However, this is currently not possible due to the reasons outlined before.

2) DIMENSION 2 OF THE EVALUATION FRAMEWORK: ALTERNATIVE TECHNIQUES FOR GUIDING MHS

To assess the reliability of the LLM output in an alternative way, we decided to compare `BRKGA+LLM` to an algorithm variant in which the alpha and beta values are obtained by an explicit parameter tuning procedure using the `irace` tool [42]. This algorithm variant will henceforth be called `BRKGA+irace`. In particular, for each of the four large social networks—mentioned in Section V-A2 and already used, for example, in Table 6—we applied a tuning procedure for obtaining well-working alpha and beta values in the following way. First, for every combination of $d \in \{1, 2, 3\}$ and $k \in \{32, 64, 128\}$ a training instance was generated. Second, `irace` was applied with a budget of 1000 algorithm runs, using (as before) a computation time limit of 900 CPU seconds per run. After obtaining the final alpha and beta values from `irace` for each network, `BRKGA+irace` was applied under the same conditions as `BRKGA+LLM` to each of the four problem instances. The results are shown in Table 8. We observe that while `BRKGA+irace` outperforms `BRKGA+LLM` in the **soc-hamsterster** and **soc-wiki-elec** instances, the opposite is the case for the **sign-bitcoinotc** instance. In the **soc-advogato** instance, performance is similar except for $d = 3$, where `BRKGA+irace` shows better results. For putting these results into perspective, consider that the alpha and beta values of `BRKGA+irace` were obtained through a specific tuning process that required significant computational time—approximately 59460 minutes (single-thread) or 2696.7 minutes (parallel jobs on SLURM) for the sum of the four problem instances (see Table 7).[19] In contrast, the LLM output (Claude-3-Opus) is obtained in just 5.3 minutes via an API call over HTTP. While this is not a fully fair comparison due to the different hardware setups, the substantial difference in runtime highlights the efficiency of the LLM-based approach. In some scenarios, it may be preferable to pay for API access rather than maintain dedicated servers that require significantly more time to achieve comparable results.

We used Pearson's correlation coefficient [46] to identify relationships between the alpha and beta parameters obtained from `irace` and those from the LLM output (see the rows labeled with $\rho_{\text{irace,LLM}}$ in Table 9). A value close to -1 indicates a negative correlation, meaning that when one value increases, the other decreases. A value close to $+1$ indicates a positive correlation, where both values tend to move together. A value close to 0 means there is no clear relationship between the two series. Our observations are as follows:

1) In the instances where `BRKGA+irace` clearly dominates—**soc-hamsterster** and **soc-wiki-elec**—there is a moderate to strong negative correlation concerning the alpha values. However, concerning the beta values, there is no clear relationship in the context of **soc-hamsterster**, whereas there is a negative correlation in the context of **soc-wiki-elec**. Thus, negative correlations are predominant. This suggests that the values determined by `irace` and the LLM tend to move in opposite directions, and based on the results, the direction chosen by `irace` appears to be the better one.
2) In contrast, in the **sign-bitcoinotc** instance, where `BRKGA+LLM` outperforms `BRKGA+irace`, there is no correlation between the two methods concerning the alpha values. The LLM's prediction is unique and unrelated to `irace`'s, demonstrating that the LLM could find good results that `irace` missed.
3) The results for the **soc-advogato** instance are inconclusive since, for each $k \in \{32, 64, 128\}$, there is no definitive winner between `BRKGA+irace` and `BRKGA+LLM`. A negative correlation concerning the alpha values but a positive correlation in the beta values can be observed. This suggests that either `irace` has identified the alpha values correctly but not the beta values, or the LLM has identified the beta values correctly but not the alpha values.

We believe that, although `BRKGA+LLM′s` results are not favorable compared to those obtained with the help of `irace` in the context of two out of four problem instances, our approach benefits from a significantly reduced computational effort (see Table7). This still holds when the computation time required to extract five metric values to build the prompts is taken into account. Additionally, with model improvements or prompt adjustments—for example, the LLM currently, by its own internal decision, produces values divisible by 0.05, leading to a loss of precision compared to `irace`—greater precision could enable the LLM to improve in the future.

3) DIMENSION 3 OF THE EVALUATION FRAMEWORK: PROMPT QUALITY

In the final dimension, we transition from a numerical analysis—as conducted in the previous two dimensions—to a more interpretative approach, where we discuss the complex process of designing a well-working prompt tailored to the considered optimization problem. To accomplish this, we first examine the five selected metrics already introduced

[19]The irace logs can be found in the `supplementary material/` folder in the repository.

TABLE 6. Comparison of the LLM output with random values. `static` **refers to a variant of** `BRKGA+LLM` **in which the LLM output is replaced by probabilities computed based on random alpha and beta values.** `dynamic` **refers to a very similar** `BRKGA+LLM` **variant in which the random values for the alpha's and beta's are dynamically changed at each iteration. Green cells indicate the best quality metrics results—higher values are better in this maximization problem.**

				$k = 32$			$k = 64$			$k = 128$		
Instance	$\|V\|$	$\|E\|$	d	static	dynamic	BRKGA+LLM	static	dynamic	BRKGA+LLM	static	dynamic	BRKGA+LLM
soc-hamsterster	2426	16630	1	1,226.5	1,227.1	**1,238.9**	1,419.7	1,418.5	**1,500.2**	1,605.6	1,609.0	**1,791.5**
			2	1,744.8	1,746.3	**1,783.2**	1,777.5	1,781.3	**1,972.5**	1,811.0	1,811.0	**2,150.2**
			3	1,788.0	1,788.0	**1,876.0**	1,816.0	1,811.8	**2,056.6**	1,825.2	1,822.4	**2,211.0**
sign-bitcoinotc	5881	35592	1	**3,479.0**	**3,479.0**	**3,479.0**	4,037.6	4,038.5	**4,061.0**	4,593.9	4,594.1	**4,628.2**
			2	5,632.1	5,632.1	**5,650.2**	5,715.3	5,715.3	**5,767.7**	5,761.8	5,734.4	**5,842.0**
			3	5,838.0	5,838.0	**5,852.7**	5,839.0	5,839.0	**5,873.9**	5,842.0	5,842.0	**5,874.0**
soc-advogato	6551	51332	1	2,463.7	2,464.1	**2,485.9**	2,949.1	2,949.4	**2,958.5**	3,338.5	3,339.2	**3,402.0**
			2	4,141.1	4,138.7	**4,144.9**	4,207.2	4,206.1	**4,234.8**	4,267.2	4,266.4	**4,357.3**
			3	4,279.0	4,276.6	**4,318.6**	4,281.3	4,283.4	**4,377.8**	4,301.1	4,299.7	**4,451.5**
soc-wiki-elec	7118	107071	1	2,166.6	2,169.1	**2,188.0**	2,265.0	2,264.7	**2,295.0**	2,367.1	2,366.5	**2,417.1**
			2	2,354.8	2,354.8	**2,365.0**	2,389.0	2,389.8	**2,418.6**	2,454.4	2,454.8	**2,484.0**
			3	2,357.3	2,357.1	**2,366.2**	2,389.8	2,390.1	**2,419.6**	2,451.9	2,451.0	**2,485.0**

TABLE 7. Comparison of computational time (in seconds) for `irace` **(single-threaded and parallel on SLURM) vs. LLM (Claude-3-Opus) via API, highlighting significant time differences despite hardware variations.**

	Time (seconds)		
	irace		LLM
Instance	Single-Thread (Serial)	Multiple Jobs in SLURM (Parallel)	Anthropic API (HTTP)
soc-wiki-elec	891900	46422	119
soc-advogato	891900	40002	114
sign-bitcoinotc	891900	37292	51
soc-hamsterster	891900	38087	36
Total time (minutes)	59460.0	2696.7	5.3

in Section IV-A. This is crucial because every additional metric enlarges the prompt. And the larger the prompt, the smaller the maximum network size that—due to the limited size of the context window—can be passed to the LLM. Also, larger prompts are associated with increased financial costs. Hence, it is essential to determine whether similar or even superior results can be achieved using less information. To investigate this, we perform two experiments: the first assesses the correlation between each pair of metrics, and the second removes information from the prompt to analyze the impact on the LLM results.

a: CORRELATION BETWEEN METRICS

For this analysis, we utilize a matrix of plots provided in Figure 5 concerning the **soc-hamsterster** instance for which `BRKGA+LLM` significantly outperformed `BRKGA`; see Table 5. Hereby, the plots in the upper triangle of the matrix are scatter plots that display the values of all pairs of metrics. For example, the second plot in the first matrix row shows the scatter plot concerning the values of metrics `out-degree` (x-axis) and `in-degree` (y-axis). In contrast, the plots in the lower triangle are kernel density estimation (KDE) plots for each pair of metrics. KDE plots provide a smoothed representation of the underlying data distribution, aiding in the identification of patterns such as clusters, outliers, and non-linear relationships. Lastly, the diagonal showcases univariate KDE plots for each variable, analogous to a histogram, depicting the distribution of each variable independently. Note that a corresponding graphic concerning the other problem instances is provided in the GitHub repository, whose link can be found in Section I-B. The following observations can be made based on Figure 5:

- The upper triangle reveals that all pairs of metric comparisons exhibit a non-linear pattern, albeit to varying degrees. For instance, the relationship between `in-degree` and `closeness` indicates a complex interaction between the metrics, suggesting that each metric contributes additional information about the problem. This finding indicates that none of the considered metrics is superfluous.
- The KDE plots in the lower triangle highlight non-linear relationships that may not be as apparent in the scatter plots. The relationship between `pagerank` and all other metrics (bottom row) indicates mainly a more linear relationship with other metrics, which was not as evident in the scatter plots. Still, the outliers shown in the scatter plots do not allow `pagerank` to be excluded from the prompt.
- Finally, the diagonal of univariate KDE plots reveals that certain metrics, such as `betweenness`, contain more frequently occurring values—observe the peak in the plot. In other words, some **betweenness** values repeat across many nodes in this instance. This insight suggests developing a prompt design strategy to further decrease the prompt size.

Upon examining each metric, we conclude that all are potentially relevant and contribute to the outcome. However, it is still possible that adjusting the set of metrics by removing certain metrics or adding alternative ones could lead to even better results.[20]

[20]Financial limitations prevent from conducting extensive experiments and from studying every aspect of the prompt, especially for large-size, challenging scenarios; hence, careful consideration is essential.

TABLE 8. A numerical comparison of `BRKGA+LLM` and `BRKGA+irace`. In the latter algorithm, the alpha and beta values are determined by tuning through `irace`. Green cells indicate the best quality metrics results—higher values are better in this maximization problem.

				$k = 32$		$k = 64$		$k = 128$	
Instance	$\|V\|$	$\|E\|$	d	BRKGA+irace	BRKGA+LLM	BRKGA+irace	BRKGA+LLM	BRKGA+irace	BRKGA+LLM
soc-hamsterster	2426	16630	1	**1,242.4**	1,238.9	**1,487.9**	1,478.0	**1,763.2**	1,731.3
			2	**1,816.3**	1,783.2	**1,956.9**	1,892.0	**2,128.7**	2,115.6
			3	**1,931.4**	1,876.0	**2,038.4**	1,947.3	**2,192.5**	2,180.2
sign-bitcoinotc	5881	35592	1	3,478.8	**3,479.0**	4,054.5	**4,054.7**	4,606.0	**4,606.2**
			2	5,642.3	**5,650.2**	5,749.1	**5,752.2**	5,823.5	**5,835.1**
			3	5,851.6	**5,852.7**	5,860.4	**5,863.4**	5,867.1	**5,868.0**
soc-advogato	6551	51332	1	**2,487.3**	2,485.9	2,951.9	**2,952.2**	3,380.9	**3,385.1**
			2	4,141.3	**4,144.9**	**4,224.6**	4,223.1	4,329.2	**4,330.1**
			3	**4,320.1**	4,318.6	**4,366.8**	4,359.9	**4,441.0**	4,431.9
soc-wiki-elec	7118	107071	1	**2,188.7**	2,188.0	**2,293.2**	2,286.1	**2,411.5**	2,408.8
			2	**2,375.3**	2,365.0	**2,417.5**	2,409.7	**2,482.90**	2,478.6
			3	**2,378.8**	2,366.2	**2,415.3**	2,406.4	**2,478.60**	2,474.2

TABLE 9. The alpha and beta values as determined by `irace` and the `LLM` for each case. Pearson's correlation coefficient ($\rho_{\text{irace,LLM}}$) is used to quantify the relationships between the two sets of values.

	soc-hamsterster		sign-bitcoinotc		soc-advogato		soc-wiki-elec	
	irace	LLM	irace	LLM	irace	LLM	irace	LLM
α_1	0.40	0.10	0.28	0.15	0.21	0.15	0.08	0.15
α_2	0.08	0.30	0.14	0.25	0.21	0.25	0.05	0.25
α_3	0.03	0.20	0.30	0.20	0.12	0.35	0.21	0.20
α_4	0.40	0.10	0.18	0.30	0.34	0.15	0.25	0.30
α_5	0.09	0.30	0.10	0.10	0.12	0.10	0.41	0.10
$\rho_{\text{irace,LLM}}$	-0.85		0.02		-0.27		-0.38	
β_1	0.78	0.60	0.61	0.70	0.31	0.60	0.61	0.70
β_2	0.83	0.60	0.21	0.60	0.65	0.70	0.92	0.60
β_3	0.65	0.90	0.01	0.80	0.83	0.90	0.19	0.90
β_4	0.01	0.60	0.51	0.05	0.75	0.60	0.80	0.60
β_5	0.75	0.60	0.50	0.10	0.86	0.70	0.51	0.50
$\rho_{\text{irace,LLM}}$	0.08		-0.55		0.55		-0.67	

b: REMOVAL OF INFORMATION

During our investigation, we discovered that modifying the prompt has the following effects:

- When removing the graph metrics from the prompt (by removing the **[EXAMPLE GRAPH]** tag), the LLM is still capable of generating useful response values. This implies that the LLM's extensive pre-training enables it to infer alpha and beta values, leveraging its prior knowledge of relevant metrics for social network node coverage problems. Despite this prior knowledge, providing an example graph (in terms of the five metric values per node) allows the LLM to refine the alpha and beta values to better suit the considered k-dDSP problem, resulting in improved output.
- Expressing metric values for each node in scientific numerical notation does not compromise the quality of the LLM's response (**[DATA]** tag). This approach offers a dual benefit: it enables greater precision while reducing the character count and, therefore, the number of tokens in the prompt.
- The beta values play a crucial role in shaping the response quality. That is, omitting them significantly reduces the quality of the LLM's output. By assigning importance weights to each metric (alpha values) and requesting an expected value (beta), we apparently enable the LLM to uncover more subtle patterns in the evaluation graph's metrics (**[EVALUATION GRAPH]** tag), ultimately leading to enhanced results.

To summarize, while the LLM possesses prior knowledge, it appears insufficient to enable the model to independently identify patterns in tabular numerical data.

c: DIFFERENCES IN NODE SELECTION

Finally, we wanted to study how the use of LLM output leads to the selection of different nodes for solutions produced by

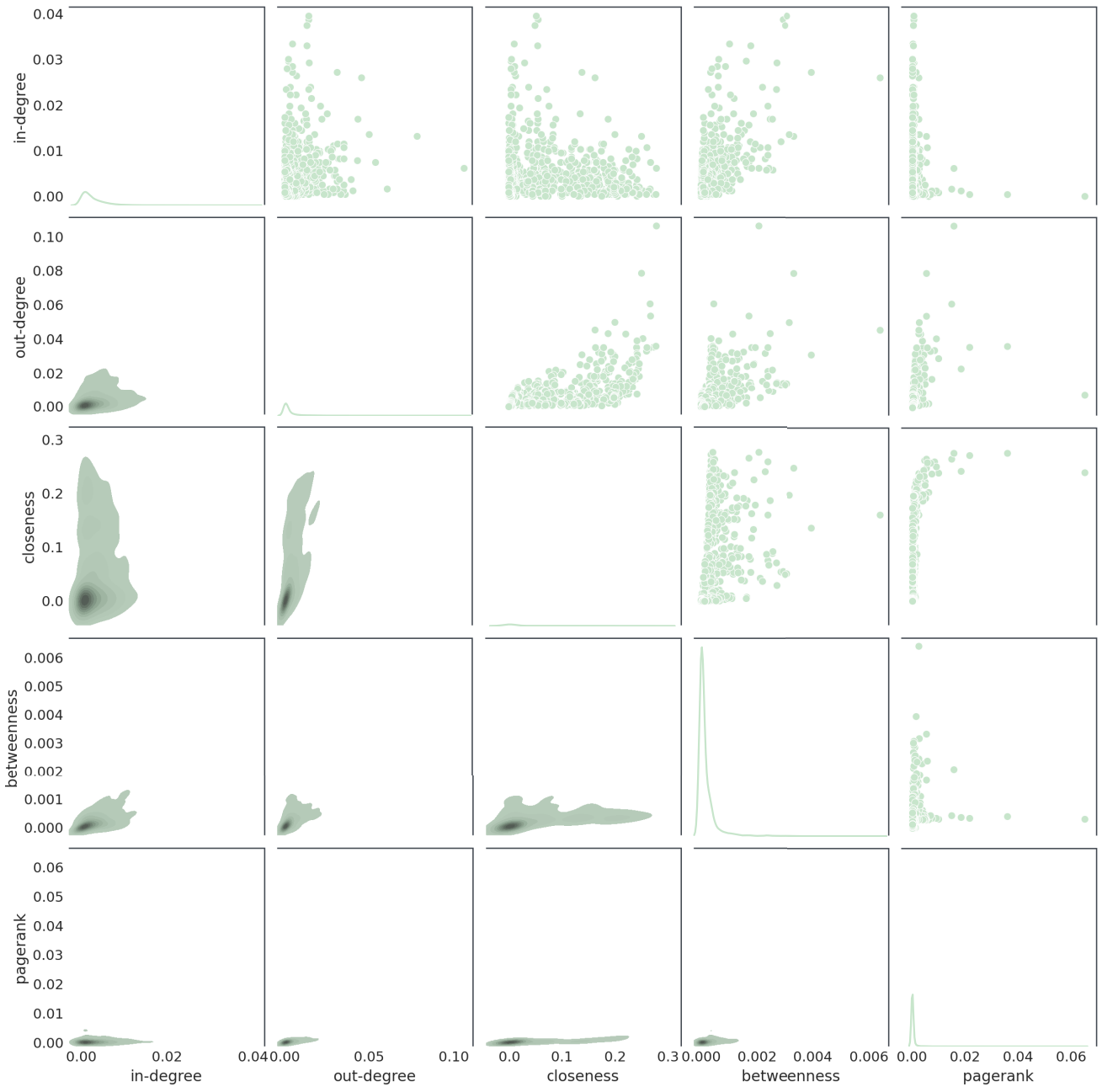

FIGURE 5. **Correlations between all pairs of the five considered metrics concerning the soc-hamsterster network.**

BRKGA+LLM in comparison to BRKGA. For this purpose, Figure 6 shows the node probabilities computed based on the alpha and beta values (black line) in relation to the (normalized) values of the five metrics, exemplary in the context of the synthetic graph **0.2-0.0-0.3-0.5**. The x-axis ranges over all 500 graph nodes, ordered by a non-increasing LLM-probability. Moreover, by means of horizontal lines the graphic marks the nodes chosen for the best BRKGA solution (dotted), the best BRKGA+LLM solution (solid), and their intersection (dashed). In the following, we point out three specific cases highlighted as (a), (b), and (c) in Figure 6:

1) One of the nodes selected by BRKGA+LLM (second solid green line) has a much higher **closeness** metric value than it has an **out-degree** metric value. Remember that **out-degree** is the standard metric used by BRKGA. This indicates that the LLM can identify more suitable nodes by blending information from several available metrics.

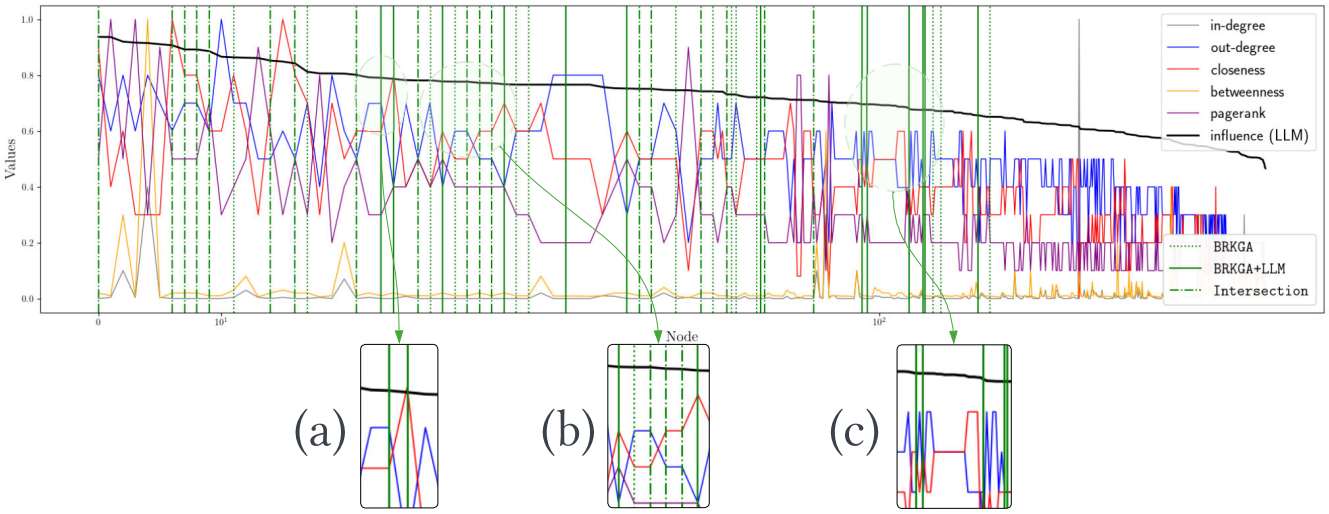

FIGURE 6. Analysis of the probabilities computed based on the alpha and beta values (black line) in relation to the (normalized) values of the five metrics. The x-axis ranges of all 500 nodes of the synthetic graph 0.2-0.0-0.3-0.5 ordered by a non-increasing LLM-probability. Moreover, the graphic marks the nodes chosen for the best BRKGA solution, the best BRKGA+LLM solution, and their intersection.

2) Similar to (a), it can be observed that for the first and last node selected by BRKGA+LLM, the closeness value is higher than the one of out-degree. In contrast, in the context of those nodes that are shared by both BRKGA and BRKGA+LLM (green dashed lines), closeness is high, but so is out-degree.
3) Cases in which closeness is high and out-degree is relatively lower in the context of nodes selected by BRKGA+LLM can also be seen in this example. This indicates that, for the 0.2-0.0-0.3-0.5 network, the best solution is achieved due to the LLM's ability to recognize the importance of closeness for certain nodes, an importance that pure BRKGA cannot detect due to a lack of information.

C. VISUAL COMPARATIVE ANALYSIS

Numerical analysis can fall short of capturing the full complexity of a metaheuristics' search process due to its stochastic nature. Visual tools have emerged in recent years to address this limitation. They were developed to provide a more comprehensive understanding and additional insight. One such tool is STNWeb [47], which generates directed graphs from algorithm trajectories to visualize how these algorithms navigate the search space. This allows to compare and justify the performance of different algorithms. A visual analysis is presented in this section to understand better the advantages of our BRKGA+LLM approach over BRKGA and BRKGA+FC.

Figure 7 shows a STNWeb plot displaying 10 runs of each BRKGA+LLM, BRKGA, and BRKGA+FC when applied to the soc-hamsterster instance. The following analysis highlights the insights that can be obtained from this visualization. However, first of all, let us explain the technical elements of the plot:

1) Each of the 30 algorithm runs is shown as a trajectory—that is, a directed path—in the search space. Hereby, the trajectories of the three algorithms are distinguished by color: BRKGA (cyan ●), BRKGA+FC (magenta ●), and BRKGA+LLM (green ●).
2) Trajectory starting points are marked by a yellow square (■). Moreover, trajectory end points are generally marked by a black triangle (▶).
3) A trajectory consists of multiple solutions (nodes, ●, ●, and ●) connected by directed edges, each with an associated fitness value. Since this is a maximization problem (k-dDSP), the fitness increases as it approaches the end of the trajectory ▶.
4) The size of each node indicates the number of trajectories that have passed through it.
5) A red node ● includes a best solution found by all 30 algorithm trajectories. Note that different best solutions might have been found.

Several interesting observations can be made based on Figure 7. First of all, only BRKGA+LLM can find best solutions (see the two red dots). Moreover, the two best solutions found by BRKGA+LLM are rather different from each other, as they are found in different areas of the search space. The three algorithms seem to be attracted by different areas of the search space. Moreover, while BRKGA and BRKGA+FC clearly converge to solutions that are close to each other, this is not so much the case for BRKGA+LLM, which does not show a clear convergence behavior towards a single area of the search space. Finally, note that the search trajectories of BRKGA are much shorter than those of the two hybrid approaches. Additional STN plots concerning other problem instances can be found in the repository whose link was provided in Section I-B.

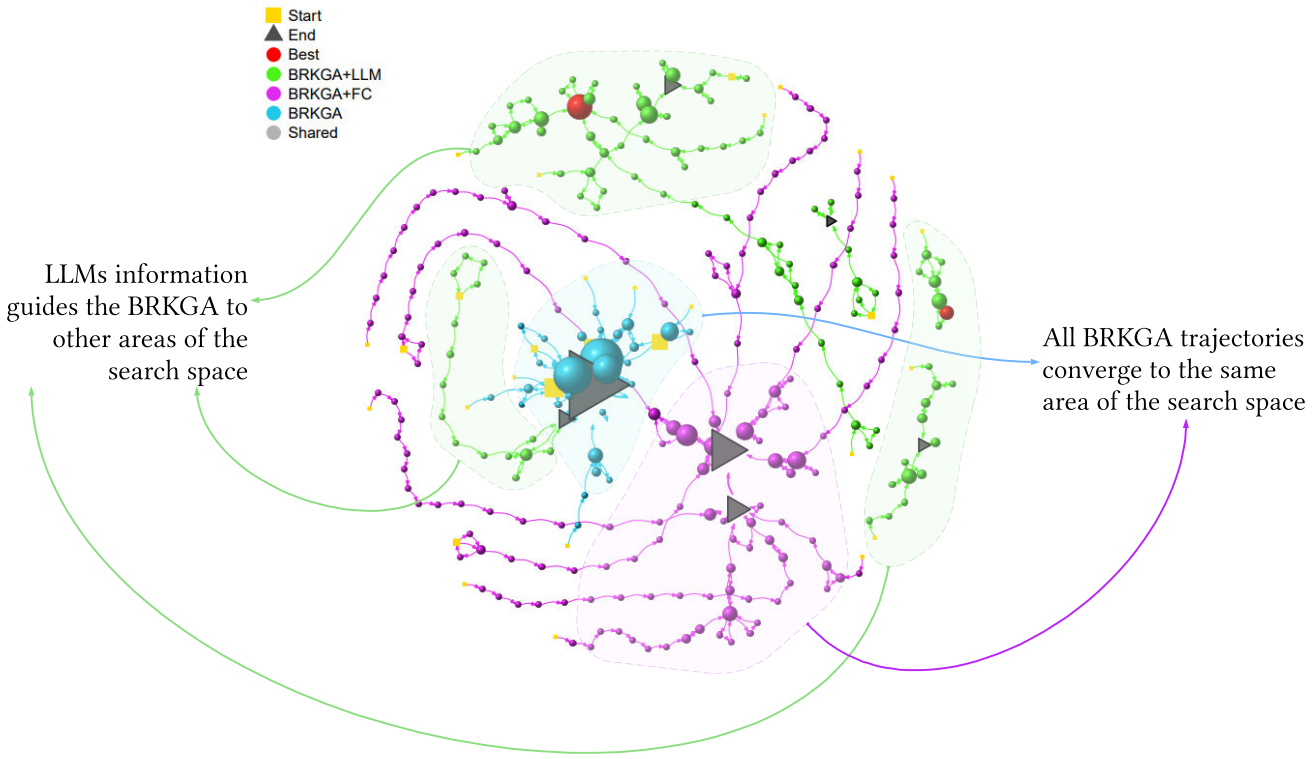

FIGURE 7. STNWeb-generated plot comparing the trajectories of `BRKGA` (cyan), `BRKGA+FC` (magenta) and `BRKGA+LLM` (green) over 10 runs on the soc-hamsterster instance (with $d = 1$ and $k = 32$). This plot was generated using the so-called agglomerative clustering partitioning method available in STNWeb, with the number of clusters set to approximately 20% of the total, allowing for a visualization focusing on the essential characteristics.

Following the empirical study of our prompt's design and quality (Section V-B) and following a visual analysis, we can conclude that our proposed hybridization achieves its intended objective: showing that LLMs can generate heuristic information that can be used to improve the search process of a metaheuristic. Interestingly, our approach has also outperformed an alternative hybridization scheme that used heuristic information produced by a hand-crafted, specifically trained, graph neural network from [19]. However, successfully integrating LLMs into MHs involves addressing several critical issues and open questions, detailed in the next section.

VI. DISCUSSION AND OPEN QUESTIONS

The LLM frenzy continues to gain momentum, with new papers appearing daily praising their virtues. LLMs are being applied far and wide, from tackling complex problems to simplifying mundane tasks. While it is uncertain whether their utility is universally applicable, our approach reveals the potential of LLMs to serve as pattern recognizers, uncovering hidden patterns and providing researchers with insights to boost their optimization algorithms. Our study gives rise to several open questions, including the following ones:

- **Are LLMs merely stochastic parrots or black boxes capable of reasoning?** In the influential paper by Bender et al. [48], the authors raise concerns about the risks of using LLMs and question their necessity. They argue that LLMs are trained on vast amounts of data based on probability distributions but lack any reference to meaning, earning them the label of *stochastic parrots*. However, significant progress has been made since the paper's publication in 2021. LLMs have improved their capabilities, and it is questionable if they can still be called "stochastic parrots." Instead, they might be seen as black boxes that can reason within a certain context. Our research demonstrates the utility of LLMs in one such context of reasoning. Moreover, several other studies have confirmed the abil of LLMs to reason in other specific contexts [5], [32]. On the other hand, other —more sensitive— contexts such as legal or moral reasoning [4] need more thoughtful human oversight, since a careless integration of LLMs could potentially have an unpredictably disruptive effect. Thus, we suggest approaching LLMs with caution and letting the experiments speak for themselves. After all, technology can improve rapidly, and it is essential to avoid hasty dismissal or overhyping LLMs as a silver bullet.
- **Are private LLMs the sole providers of superior outcomes?** In the past, it would have been accurate to affirm this, as models such as GPT-4 and Claude-3-Opus were recognized for their top-tier response quality. In fact, our research demonstrates that Claude-3-Opus

outperforms its competitors. However, the scene is evolving rapidly. Recent models, such as Cohere's Command-R+, Mistral's Mistral models, and Meta's Llama 3 (which we did not incorporate in our study, because its context limit is very low: 8192), are delivering results comparable to some of the before-mentioned models (e.g., [49], [50], [51]). These models also come with open licenses, although with varying levels of permissiveness. Nonetheless, it is important to note that these models are all products of private entities with substantial financial resources. The prospect of a public entity or small business independently creating an LLM from scratch appears remote, largely due to the computational demands and associated costs. This could potentially pose a risk to the transparency of their design processes [52].

- **What are the primary obstacles to adopting the strategy proposed in our research?** We identify two significant hurdles: cost constraints and technical limitations. While leveraging LLMs as software-as-service can alleviate the need for in-house cluster training, a substantial financial burden is incurred by processing large volumes of tokens. Our investigation revealed that even with moderate-sized graph instances of around 7000 nodes, we quickly reach the token limit of what the most permissive LLM can handle within its context window (see subsection V-A2). Having said that, recent studies (see, for example, [53]) have proposed the possibility of ''infinite'' context windows. Additionally, Google's Gemini Pro 1.5 model boasts an impressive 2 800 000-token context window limit [54]. However, to successfully apply a strategy similar to ours in the context of massive graphs—or, more generally, in the context of large-scale problem instances—these two limitations must be significantly mitigated. Alternatively, a novel prompt strategy could be developed to detect patterns in metrics while reducing token counts, potentially through prompt compression [55] or relevant node metric filtering. Neither of these alternatives has been investigated in our research.
- **As researchers, what other aspects of LLMs should we be cautious about?** Earlier, we touched on the issue of LLMs' capability for reasoning in certain contexts. However, many argue that they are incapable of reasoning altogether. This discrepancy stems from the ambiguity surrounding the term ''reasoning''. In our context, reasoning refers to the ability to infer and identify useful patterns in metrics associated with each node of a graph. In particular, we demonstrated that LLMs do not produce random or uninformative values but rather follow the given instructions. However, if we had left the concept of ''reasoning'' or ''pattern discovery'' too vague, our research would be susceptible to misinterpretation. Therefore, we recommend that researchers be aware of the underlying philosophical discussions surrounding this technology when working with LLMs. After all, technological advancements can shape the way we express ourselves. A good starting point may be to engage with the works of Floridi (e.g., [33], [56]). Particularly since the boundaries of LLMs' capabilities remain undefined and are a subject of ongoing debate.
- **Are there ways to improve the integration between MHs and LLMs?** As discussed in Section II-B, there are already a few hybridization approaches. These include creating new MHs by leveraging LLMs to generate code and employing LLMs as solvers for optimization problems described in natural language. In contrast, our approach utilizes LLMs as pattern detectors for complex instances. These patterns are then used to bias the search process of the metaheuristic. However, an intriguing integration could involve combining all these hybridization techniques within a single software framework. We believe that these approaches are not mutually exclusive but rather complementary. By unifying them, we may unlock even greater contributions to the field of MHs. One potential approach to achieving this integration could be through the use of agents [57], which would be responsible for orchestrating the three hybridization methods. Note that agents are currently a topic of significant interest in the LLM community [31].

VII. CONCLUSION

This paper showcased the potential of leveraging Large Language Models (LLMs) as *pattern search engines* to enhance metaheuristics (MH) by integrating the information they provide. We demonstrate the effectiveness of this approach on a combinatorial optimization problem in the realm of social networks. An important aspect of our work is prompt engineering. In fact, useful LLM answers are only obtained with well-designed prompts. In the context of the considered social networks problem, the LLM output is used to compute a probability for each node of the input graph to belong to an optimal solution. These node probabilities are then used to bias the search process of a biased random key genetic algorithm (BRKGA). We could show that our hybrid approach outperforms both the pure BRKGA and the state-of-the-art BRKGA variant, whose search process is biased by the output of a hand-designed (and trained) graph neural network model. To address this, we created a tool named `OptiPattern` (*LLM-Powered Pattern Recognition for Combinatorial Optimization*), which implements this hybrid method and is available at: https://github.com/camilochs/optipattern.

This pioneering approach paves the way for further exploration, including extending LLM-assisted pattern recognition to problem instances in a broader range of optimization problems.

REFERENCES

[1] J. Achiam et al., ''GPT-4 technical report,'' 2023, *arXiv:2303.08774*.

[2] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample, "LLaMA: Open and efficient foundation language models," 2023, *arXiv:2302.13971*.

[3] *The Claude 3 Model Family: Opus, Sonnet, Haiku*, Anthropic, San Francisco, CA, USA, 2024.

[4] G. F. C. F. Almeida, J. L. Nunes, N. Engelmann, A. Wiegmann, and M. D. Araújo, "Exploring the psychology of LLMs' moral and legal reasoning," *Artif. Intell.*, vol. 333, Aug. 2024, Art. no. 104145.

[5] J. Ahn, R. Verma, R. Lou, D. Liu, R. Zhang, and W. Yin, "Large language models for mathematical reasoning: Progresses and challenges," 2024, *arXiv:2402.00157*.

[6] J. Wei, Y. Tay, R. Bommasani, C. Raffel, B. Zoph, S. Borgeaud, D. Yogatama, M. Bosma, D. Zhou, D. Metzler, E. H. Chi, T. Hashimoto, O. Vinyals, P. Liang, J. Dean, and W. Fedus, "Emergent abilities of large language models," 2022, *arXiv:2206.07682*.

[7] Z. Wan, X. Wang, C. Liu, S. Alam, Y. Zheng, J. Liu, Z. Qu, S. Yan, Y. Zhu, Q. Zhang, M. Chowdhury, and M. Zhang, "Efficient large language models: A survey," 2023, *arXiv:2312.03863*.

[8] M. Gendreau and J.-Y. Potvin, Eds., *Handbook Metaheuristics*, 3rd ed., Cham, Switzerland: Springer, 2019.

[9] C. Blum, J. Puchinger, G. R. Raidl, and A. Roli, "Hybrid metaheuristics in combinatorial optimization: A survey," *Appl. Soft Comput.*, vol. 11, no. 6, pp. 4135–4151, Sep. 2011. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1568494611000962

[10] M. Karimi-Mamaghan, M. Mohammadi, P. Meyer, A. M. Karimi-Mamaghan, and E.-G. Talbi, "Machine learning at the service of meta-heuristics for solving combinatorial optimization problems: A state-of-the-art," *Eur. J. Oper. Res.*, vol. 296, no. 2, pp. 393–422, Jan. 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0377221721003623

[11] S. Barocas, M. Hardt, and A. Narayanan, *Fairness and Machine Learning: Limitations and Opportunities*. Cambridge, MA, USA: MIT Press, 2023.

[12] M. Pluhacek, A. Kazikova, T. Kadavy, A. Viktorin, and R. Senkerik, "Leveraging large language models for the generation of novel metaheuristic optimization algorithms," in *Proc. Companion Conf. Genetic Evol. Comput.* New York, NY, USA: Association for Computing Machinery, Jul. 2023, pp. 1812–1820, doi: 10.1145/3583133.3596401.

[13] S. Liu, C. Chen, X. Qu, K. Tang, and Y.-S. Ong, "Large language models as evolutionary optimizers," 2023, *arXiv:2310.19046*.

[14] Z. Ma, H. Guo, J. Chen, G. Peng, Z. Cao, Y. Ma, and Y.-J. Gong, "LLaMoCo: Instruction tuning of large language models for optimization code generation," 2024, *arXiv:2403.01131*.

[15] C. Yang, X. Wang, Y. Lu, H. Liu, Q. V. Le, D. Zhou, and X. Chen, "Large language models as optimizers," in *Proc. 12th Int. Conf. Learn. Represent.*, Jan. 2023, pp. 1–42. [Online]. Available: https://openreview.net/forum?id=Bb4VGOWELI

[16] P.-F. Guo, Y.-H. Chen, Y.-D. Tsai, and S.-D. Lin, "Towards optimizing with large language models," 2023, *arXiv:2310.05204*.

[17] R. Ma, X. Wang, X. Zhou, J. Li, N. Du, T. Gui, Q. Zhang, and X. Huang, "Are large language models good prompt optimizers?" 2024, *arXiv:2402.02101*.

[18] Y. Bengio, A. Lodi, and A. Prouvost, "Machine learning for combinatorial optimization: A methodological tour d'horizon," *Eur. J. Oper. Res.*, vol. 290, no. 2, pp. 405–421, Apr. 2021.

[19] C. C. Sartori and C. Blum, "Boosting a genetic algorithm with graph neural networks for multi-hop influence maximization in social networks," in *Proc. 17th Conf. Comput. Sci. Intell. Syst. (FedCSIS)*, Sep. 2022, pp. 363–371.

[20] M. López-Ibáñez, J. Branke, and L. Paquete, "Reproducibility in evolutionary computation," *ACM Trans. Evol. Learn. Optim.*, vol. 1, no. 4, pp. 1–21, Oct. 2021.

[21] Y. Sun, S. Wang, Y. Shen, X. Li, A. T. Ernst, and M. Kirley, "Boosting ant colony optimization via solution prediction and machine learning," *Comput. Oper. Res.*, vol. 143, Jul. 2022, Art. no. 105769. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0305054822000636

[22] F. Lucas, R. Billot, M. Sevaux, and K. Sörensen, "Reducing space search in combinatorial optimization using machine learning tools," in *Learning and Intelligent Optimization*, I. S. Kotsireas and P. M. Pardalos, Eds., Cham, Switzerland: Springer, 2020, pp. 143–150.

[23] D. Liu, V. Perreault, A. Hertz, and A. Lodi, "A machine learning framework for neighbor generation in metaheuristic search," *Frontiers Appl. Math. Statist.*, vol. 9, Jul. 2023, Art. no. 1128181.

[24] M. Huber and G. R. Raidl, "Learning beam search: Utilizing machine learning to guide beam search for solving combinatorial optimization problems," in *Proc. Int. Conf. Mach. Learn., Optim., Data Sci.*, Jan. 2022, pp. 283–298.

[25] A. Fenoy, F. Bistaffa, and A. Farinelli, "An attention model for the formation of collectives in real-world domains," *Artif. Intell.*, vol. 328, Mar. 2024, Art. no. 104064. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0004370223002102

[26] M. Alicastro, D. Ferone, P. Festa, S. Fugaro, and T. Pastore, "A reinforcement learning iterated local search for makespan minimization in additive manufacturing machine scheduling problems," *Comput. Oper. Res.*, vol. 131, Jul. 2021, Art. no. 105272.

[27] A. A. Chaves and L. H. N. Lorena, "An adaptive and near parameter-free BRKGA using Q-learning method," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Jun. 2021, pp. 2331–2338.

[28] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, Jun. 2017, pp. 5998–6008.

[29] C. M. and Hugh, *Deep Learning: Foundations and Concepts*, 1st ed., Cham, Switzerland: Springer, Nov. 2023.

[30] C. Singh, J. P. Inala, M. Galley, R. Caruana, and J. Gao, "Rethinking interpretability in the era of large language models," 2024, *arXiv:2402.01761*.

[31] T. Guo, X. Chen, Y. Wang, R. Chang, S. Pei, N. V. Chawla, O. Wiest, and X. Zhang, "Large language model based multi-agents: A survey of progress and challenges," 2024, *arXiv:2402.01680*.

[32] S. Mirchandani, F. Xia, P. Florence, B. Ichter, D. Driess, M. G. Arenas, K. Rao, D. Sadigh, and A. Zeng, "Large language models as general pattern machines," 2023, *arXiv:2307.04721*.

[33] L. Floridi and A. C. Nobre, "Anthropomorphising machines and computerising minds: The crosswiring of languages between artificial intelligence and brain & cognitive sciences," *Minds Mach.*, vol. 34, no. 1, p. 5, Apr. 2024, doi: 10.1007/s11023-024-09670-4.

[34] S. Qiao, Y. Ou, N. Zhang, X. Chen, Y. Yao, S. Deng, C. Tan, F. Huang, and H. Chen, "Reasoning with language model prompting: A survey," 2022, *arXiv:2212.09597*.

[35] R. Ni, X. Li, F. Li, X. Gao, and G. Chen, "FastCover: An unsupervised learning framework for multi-hop influence maximization in social networks," 2021, *arXiv:2111.00463*.

[36] P. Basuchowdhuri and S. Majumder, "Finding influential nodes in social networks using minimum k-hop dominating set," in *Applied Algorithms*, P. Gupta and C. Zaroliagis, Eds., Cham, Switzerland: Springer, 2014, pp. 137–151, doi: 10.1007/978-3-319-04126-1_12.

[37] T. B. Brown et al., "Language models are few-shot learners," in *Proc. Adv. Neural Inf. Process. Syst.*, Jan. 2020, pp. 1877–1901.

[38] W. Chen, "Large language models are few(1)-shot table reasoners," 2022, *arXiv:2210.06710*.

[39] B. Atil, A. Chittams, L. Fu, F. Ture, L. Xu, and B. Baldwin, "LLM stability: A detailed analysis with some surprises," 2024, *arXiv:2408.04667*.

[40] P. Erdős and A. Rényi, "On random graphs. I.," *Publicationes Mathematicae*, vol. 6, nos. 3–4, pp. 290–297, Jul. 2022.

[41] Y. Xing, Z. Chen, J. Sun, and L. Hu, "An improved adaptive genetic algorithm for job-shop scheduling problem," in *Proc. 3rd Int. Conf. Natural Comput. (ICNC)*, 2007, pp. 287–291.

[42] M. López-Ibáñez, J. Dubois-Lacoste, L. P. Cáceres, M. Birattari, and T. Stützle, "The irace package: Iterated racing for automatic algorithm configuration," *Oper. Res. Perspect.*, vol. 3, pp. 43–58, Jan. 2016.

[43] L. Zheng, W.-L. Chiang, Y. Sheng, S. Zhuang, Z. Wu, Y. Zhuang, L. Zi, Z. Li, D. Li, E. P. Xing, H. Zhang, J. E. Gonzalez, and I. Stoica, "Judging LLM-as-a-judge with MT-bench and chatbot arena," in *Proc. Adv. Neural Inf. Process. Syst.*, Jan. 2023, pp. 46595–46623.

[44] D. F. Nettleton, "A synthetic data generator for online social network graphs," *Social Netw. Anal. Mining*, vol. 6, no. 1, p. 44, 2016, doi: 10.1007/s13278-016-0352-y.

[45] J. Leskovec and A. Krevl. (Jun. 2014). *SNAP Datasets: Stanford Large Network Dataset Collection*. [Online]. Available: http://snap.stanford.edu/data

[46] J. Benesty, J. Chen, Y. Huang, and I. Cohen, *Pearson Correlation Coefficient*. Berlin, Germany: Springer, 2009, pp. 1–4, doi: 10.1007/978-3-642-00296-0_5.

[47] C. C. Sartori, C. Blum, and G. Ochoa, "STNWeb: A new visualization tool for analyzing optimization algorithms," *ACM SIGEVOlution*, vol. 16, no. 3, Sep. 2023, Art. no. 100558. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2665963823000957

[48] E. M. Bender, T. Gebru, A. McMillan-Major, and S. Shmitchell, "On the dangers of stochastic parrots: Can language models be too big?" in *Proc. ACM Conf. Fairness, Accountability, Transparency*. New York, NY, USA: Association for Computing Machinery, Mar. 2021, pp. 610–623, doi: 10.1145/3442188.3445922.
[49] C. Davis, A. Caines, Ø. Andersen, S. Taslimipoor, H. Yannakoudakis, Z. Yuan, C. Bryant, M. Rei, and P. Buttery, "Prompting open-source and commercial language models for grammatical error correction of English learner text," 2024, *arXiv:2401.07702*.
[50] Z. Liu et al., "LLM360: Towards fully transparent open-source LLMs," 2023, *arXiv:2312.06550*.
[51] H. Chen, F. Jiao, X. Li, C. Qin, M. Ravaut, R. Zhao, C. Xiong, and S. Joty, "ChatGPT's one-year anniversary: Are open-source large language models catching up?" 2023, *arXiv:2311.16989*.
[52] B. Harandizadeh, A. Salinas, and F. Morstatter, "Risk and response in large language models: Evaluating key threat categories," 2024, *arXiv:2403.14988*.
[53] T. Munkhdalai, M. Faruqui, and S. Gopal, "Leave no context behind: Efficient infinite context transformers with infini-attention," 2024, *arXiv:2404.07143*.
[54] G. Team, "Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context," 2024, *arXiv:2403.05530*.
[55] H. Jiang, Q. Wu, C.-Y. Lin, Y. Yang, and L. Qiu, "LLMLingua: Compressing prompts for accelerated inference of large language models," 2023, *arXiv:2310.05736*.
[56] L. Floridi, "Ai as agency without intelligence: On ChatGPT, large language models, and other generative models," *Philosophy Technol.*, vol. 36, no. 1, pp. 1–7, 2023.
[57] Z. Xi et al., "The rise and potential of large language model based agents: A survey," 2023, *arXiv:2309.07864*.

CAMILO CHACÓN SARTORI is currently pursuing the Ph.D. degree in AI with the Artificial Intelligence Research Institute (IIIA-CSIC), Bellaterra, Spain. His research interests include establishing a connection between computational optimization, metaheuristics, visualization tools for understanding algorithm behavior, and generative models.

CHRISTIAN BLUM received the Ph.D. degree in applied sciences from the Free University of Brussels, Brussels, Belgium, in 2004. He is currently a Senior Research Scientist with the Artificial Intelligence Research Institute (IIIA-CSIC), Bellaterra, Spain. His research interests include solving difficult optimization problems using swarm intelligence techniques and combinations of metaheuristics with exact techniques.

FILIPPO BISTAFFA received the Ph.D. degree in computer science from the University of Verona, in 2016. He is currently a Tenured Researcher (former Marie Skłodowska-Curie Fellow) with the Artificial Intelligence Research Institute (IIIA-CSIC), Bellaterra, Spain. His research interests include optimization applied to complex real-world problems (e.g., sustainable mobility, and team formation for cooperative learning) and, more recently, ethical and trustworthy AI.

GUILLEM RODRÍGUEZ COROMINAS is currently pursuing the joint Ph.D. degree in computer science with the Polytechnic University of Catalonia (UPC), Barcelona, Spain, and the Artificial Intelligence Research Institute (IIIA), Bellaterra, Spain. His research interests include tackling complex combinatorial optimization problems using metaheuristics, with a special focus on hybridizing these methods with exact algorithms and machine learning techniques.