

ADVERTIMENT. L'accés als continguts d'aquesta tesi queda condicionat a l'acceptació de les condicions d'ús estableties per la següent llicència Creative Commons:  <https://creativecommons.org/licenses/?lang=ca>

ADVERTENCIA. El acceso a los contenidos de esta tesis queda condicionado a la aceptación de las condiciones de uso establecidas por la siguiente licencia Creative Commons:  <https://creativecommons.org/licenses/?lang=es>

WARNING. The access to the contents of this doctoral thesis is limited to the acceptance of the use conditions set by the following Creative Commons license:  <https://creativecommons.org/licenses/?lang=en>



Improving Optimization Algorithms via Machine Learning and Visualization Tools

A dissertation presented in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy in Computer Science

Camilo Chacón Sartori

Universitat Autònoma de Barcelona
Department of Computer Science
PhD Thesis in Computer Science

Barcelona, September 2025

Improving Optimization Algorithms via Machine Learning and Visualization Tools

A dissertation presented in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy in Computer Science

Camilo Chacón Sartori

Supervisor: Christian Blum

Senior Researcher

Co-supervisor: Filippo Bistaffa

Tenured Researcher

Universitat Autònoma de Barcelona
Department of Computer Science
PhD Thesis in Computer Science

Barcelona, September 2025

Acknowledgements

To my supervisor, Christian Blum, who gave me the most valuable things a PhD student can receive: ideas, trust, and above all, *time*.

To all the people who hired me to do some kind of work (courses, talks, private lessons, consulting) throughout these years—thank you; it was thanks to you that I was able to support myself during the PhD.

To my family—my mother, my sisters, and my nieces—for their constant support in all the projects that this hopelessly passionate one (me) insists on keeping alive.

Finally, to Metallica and Dream Theater, whose music accompanied me throughout the PhD years: during my routine visits to Sandwichez (my favorite coffee shop), on my way to and from my go-to bar, during my compulsive book-buying at my favorite bookstore (Alibri), and through the endless coffees I drank at the IIIA; filling me with motivation and nostalgia—especially for the romantic dates I never had.

Thank you all!

Agradecimientos

A mi director de tesis, Christian Blum, quien me dio las cosas más valiosas que un estudiante de doctorado puede recibir: ideas, confianza y, sobre todo, *tiempo*.

A todas las personas que me contrataron para realizar algún tipo de trabajo (cursos, charlas, clases particulares, consultoría) a lo largo de estos años, gracias; fue gracias a ustedes que pude mantenerme durante el doctorado.

A mi familia —mi madre, mis hermanas y mis sobrinas— por su apoyo constante en todos los proyectos que este apasionado sin remedio (yo) insiste en mantener vivos.

Finalmente, a Metallica y Dream Theater, cuya música me acompañó durante todos los años del doctorado: en mis visitas rutinarias a Sandwichez (mi cafetería favorita), en mis trayectos de ida y vuelta a mi bar de cabecera, durante mis compras compulsivas de libros en mi librería favorita (Alibri), y a través de los interminables cafés que me tomé en el IIIA; llenándome de motivación y nostalgia, especialmente por las citas románticas que nunca tuve.

¡Gracias a todos!

Abstract

This thesis addresses two major challenges in metaheuristics: the lack of information about the problem instance and the lack of transparency in understanding their results.

To tackle the first issue, I propose integrating two Machine Learning techniques into metaheuristic algorithms: Graph Neural Networks and, more notably, Large Language Models. These techniques enhance the quality of solutions obtained for NP-hard combinatorial problems when compared to using the metaheuristic alone.

To address the second issue, the thesis introduces a visualization tool called STNWeb, designed to support researchers in the comparative analysis of metaheuristics by facilitating the understanding of how problem instances influence their behavior.

Thus, the thesis is structured in two parts: the first focuses on improving the solution quality of metaheuristics through the incorporation of information extracted via modern Machine Learning techniques; the second explores the use of visualization as a means to better understand the behavior of metaheuristics.

Keywords: Metaheuristics, Combinatorial Optimization, Machine Learning, Large Language Models, Visual Tools

Resumen

Esta tesis aporta soluciones a dos problemas presentes en las metaheurísticas: la ausencia de información sobre la instancia del problema y la falta de transparencia para entender sus resultados.

Para abordar el primer problema, se propone integrar dos técnicas de Machine Learning dentro de las metaheurísticas: las Graph Neural Networks y, especialmente, los recientes Modelos Masivos de Lenguaje. Estas técnicas mejoran la calidad de las soluciones obtenidas en problemas combinatorios NP-hard, en comparación con el uso exclusivo de la metaheurística.

Para el segundo problema, se propone una herramienta de visualización llamada STNWeb, diseñada para apoyar a los investigadores en el análisis comparativo de metaheurísticas, facilitando la comprensión de cómo la instancia del problema influye en el comportamiento de estas técnicas.

Así, la tesis se divide en dos partes: la primera está dedicada a mejorar la calidad de las soluciones generadas por metaheurísticas mediante la incorporación de información proveniente de técnicas modernas de Machine Learning; la segunda se enfoca en el uso de la visualización como medio para comprender el comportamiento de las metaheurísticas.

Palabras Clave: Metaheurísticas, Optimización Combinatoria, Aprendizaje Automático, Modelos Masivos de Lenguaje, Herramientas visuales

Resum

Aquesta tesi aporta solucions a dos problemes presents en les metaheurístiques: l'absència d'informació sobre la instància del problema i la manca de transparència per entendre els seus resultats.

Per abordar el primer problema, es proposa integrar dues tècniques de Machine Learning dins de les metaheurístiques: les Graph Neural Networks i, especialment, els recents Models Massius de Llenguatge. Aquestes tècniques milloren la qualitat de les solucions obtingudes en problemes combinatoris NP-hard, en comparació amb l'ús exclusiu de la metaheurística.

Per al segon problema, es proposa una eina de visualització anomenada STNWeb, dissenyada per donar suport als investigadors en l'anàlisi comparativa de metaheurístiques, facilitant la comprensió de com la instància del problema influeix en el comportament d'aquestes tècniques.

Així doncs, la tesi es divideix en dues parts: la primera està dedicada a millorar la qualitat de les solucions generades per metaheurístiques mitjançant la incorporació d'informació provinent de tècniques modernes de Machine Learning; la segona se centra en l'ús de la visualització com a mitjà per comprendre el comportament de les metaheurístiques.

Paraules clau: Metaheurístiques, Optimització Combinatòria, Aprendentatge Automàtic, Models Massius de Llenguatge, Eines visuals

Contents

<i>List of Figures</i>	xxii
<i>List of Tables</i>	xxx
<i>Glossary</i>	xxxiv

1 Introduction	1
1.1 Overview of Optimization Algorithms	1
1.1.1 Metaheuristics	2
1.2 Contributions	3
1.2.1 Algorithmic Improvement	3
1.2.2 Interpretability Enhancement through Visualization Tools	4
1.2.3 Philosophical Implications	6
1.3 Publications Resulting from this Thesis	6
1.4 Organization	8
1.4.1 Part I - Algorithmic Improvement with Machine Learning (Graph Neural Networks & Large Language Models)	9
1.4.2 Part II - Visualization Tools for Algorithm Analysis	9

Part I - Machine Learning (Graph Neural Networks & Large Language Models)

2 Introduction	18
2.1 Metaheuristics and Machine Learning	18
2.2 Metaheuristics and Deep Learning	19
2.2.1 Deep Reinforcement Learning (DRL) in Metaheuristics	20
2.2.2 Graph Neural Networks for Combinatorial Optimization	20
2.3 Metaheuristics and Large Language Models	20
2.3.1 LLMs for Improving Solution Quality	21
3 Boosting a Genetic Algorithm using Graph Neural Networks	24
3.1 Introduction	25
3.2 Problem Definition	25
3.3 Methodology	27
3.3.1 Biased Random Key Genetic Algorithm	27
3.3.2 Graph Neural Network Framework	29

3.3.3	The Hybrid BRKGA Algorithm	30
3.4	Experimental Evaluation	31
3.4.1	Data Preparation and Tuning Process	37
3.4.2	Experimental Evaluation	37
3.4.3	Analysis	40
3.5	Conclusion	40
4	Improving Ant Colony Optimization supported by Deep Learning	42
4.1	Introduction	43
4.2	Problem Definition	43
4.3	Methodology	45
4.3.1	Solution Construction in MMAS	46
4.3.2	Integrating Deep Learning via Q-Learning	47
4.4	Generating Deep Learning-Based Node Information	48
4.4.1	Selected Features and Training Instances	48
4.4.2	SAGE Training	49
4.5	Experimental Evaluation	53
4.5.1	Experimental setting	54
4.5.2	Algorithm tuning	54
4.5.3	Numerical results	54
4.6	Conclusion	56
5	Large Language Models as Assistants for Enhancing Metaheuristics	57
5.1	Introduction	57
5.1.1	Our Contribution	58
5.2	Background	59
5.2.1	LLMs as Pattern Recognition Engines	59
5.3	Problem Definition	60
5.3.1	Multi-Hop Influence Maximization	60
5.4	Integration of LLM Output into a Metaheuristic	61
5.4.1	Prompt Engineering	61
5.4.2	LLM Output	66
5.4.3	Using LLM Output to Guide a Metaheuristic	67
5.5	Empirical Evaluation	68
5.5.1	Experimental Setup	68
5.5.2	Analysis of LLM Output	70
5.5.3	Visual Comparative Analysis	78
5.6	Discussion and Open Questions	80
5.7	Conclusion	82

6 Enhancing a CMSA Heuristic for the MIS Problem with Large Language Models	84
6.1 Introduction	84
6.2 Background	85
6.2.1 Code Generation with LLMs	85
6.2.2 Maximum Independent Set (MIS) Problem	87
6.2.3 CMSA	87
6.3 LLM-Enhanced CMSA for MIS	89
6.3.1 Discovering New Heuristics	89
6.3.2 Code Optimization Strategies	93
6.3.3 Reproducibility	95
6.4 Empirical Evaluation	95
6.4.1 Preliminary	95
6.4.2 Numerical Results	98
6.5 Discussion	100
6.6 Conclusions	101
7 Improvement of Optimization Algorithms with LLMs by Non-expert Users	102
7.1 Introduction	102
7.2 Background	103
7.2.1 Large Language Models in Combinatorial Optimization	103
7.2.2 Problem Definition	106
7.2.3 Traditional Optimization Algorithms for the TSP	107
7.2.4 Selected Implementations	109
7.3 Methodology	109
7.3.1 Enhancing Traditional Optimization Algorithms with Large Language Models	109
7.3.2 Prompt Design: A Focus on Simplicity and Accessibility	111
7.4 Experimental evaluation	113
7.4.1 Setup	113
7.4.2 Benchmark Datasets and Evaluation Metrics	114
7.4.3 Experimental Design	114
7.4.4 Comparative Analysis with Original Algorithm Codes	116
7.4.5 Key Insights in Code Generation	120
7.4.6 Code complexity	123
7.5 Discussion	124
7.5.1 Limitations and Methodological Considerations	125
7.5.2 Directions for Future Research	126
7.6 Conclusion	126

8	Introduction	133
8.1	LLMs for Automated Analysis in Optimization Tools	134
8.2	Future Directions: Leveraging LVLMs for Enhanced STNWeb Analysis	135
9	Search Trajectory Networks Meet the Web	136
9.1	Introduction	136
9.2	Background: Search Trajectory Networks	137
9.2.1	Limitations	139
9.3	Integration Into the Web	140
9.3.1	New system architecture	141
9.3.2	New Features	143
9.4	Case Studies	145
9.4.1	Case 1: A Simple Study	147
9.4.2	Case 2: Comparison of Two Algorithms	147
9.4.3	Case 3: Complex Analysis	148
9.5	Conclusion	149
10	STNWeb: A new visualization tool for analyzing optimization algorithms	150
10.1	Introduction	150
10.2	STNWeb Architecture	151
10.2.1	STNWeb Frontend	152
10.2.2	STNWeb Backend	152
10.2.3	REST API	152
10.2.4	Search Space Partitioning Strategy	153
10.3	Limitations	154
10.4	Conclusion	154
11	Enhancing the Explainability of STNWeb with Large Language Models	156
11.1	Introduction	156
11.2	Background	157
11.2.1	Search Trajectory Networks (STNs)	157
11.2.2	Large Language Models (LLMs)	160
11.3	Integrating LLMs into STNWeb	160
11.3.1	Prompt Engineering	161
11.3.2	Feature Extraction	163
11.4	Empirical Evaluation	165
11.4.1	Setup	166
11.4.2	Methodology	166
11.4.3	Results	168
11.5	Discussion	168
11.6	Conclusion	169

12 Improving STNWeb Graphical via HAC of the Search Space	171
12.1 Introduction	171
12.2 Contextual Overview: Search Trajectory Networks	172
12.2.1 Search Space Partitioning Schemes	172
12.2.2 Standard Strategies for Partitioning	173
12.3 Partitioning By Hierarchical Agglomerative Clustering	174
12.4 Experimental Results	176
12.4.1 Methodology and Setup	176
12.4.2 Continuous Optimization Problems	181
12.5 Conclusion	181
13 A Benchmark Generator for Assessing Variability in Graph Analysis Using LVLMs	183
13.1 Introduction	183
13.1.1 Contributions	185
13.2 VisGraphVar: A benchmark generator	185
13.2.1 A Custom Synthetic Dataset	185
13.2.2 Tasks	186
13.2.3 Dataset Configuration and Statistics	194
13.2.4 Metrics	195
13.2.5 Prompt design	196
13.3 Experiments and Evaluation	197
13.3.1 Environment Setup and LVLM Configuration	197
13.3.2 Results	197
13.3.3 Observations	203
13.4 Discussion and Open Questions	209
13.5 Conclusions	210
14 Conclusion	212
14.1 Discussion of Main Contributions	212
14.1.1 Part I – Algorithmic Enhancements	212
14.1.2 Part II – Enhanced Interpretability	213
14.2 Limitations and Challenges	214
14.2.1 Algorithmic Improvement	214
14.2.2 Interpretability Enhanced	216
14.3 Future Research Directions	216
14.3.1 STNWeb 3D	216
14.3.2 Path-Dependent Runtime Heuristic Steering (PathSteer)	217

A A Brief Guide to Optimization	223
A.1 Types of Optimization Problems	223
A.1.1 Continuous Optimization	223
A.1.2 Discrete or Combinatorial Optimization	224
B A Brief Introduction and Defense of Metaheuristics	225
B.1 An Invitation to Metaheuristics	227
<i>Bibliography</i>	231

List of Figures

1.1	A visual map of the thesis structure, where the connections between nodes show the influence of each chapter on the others. For example, node 10 represents the work in which we developed STNWeb; this tool is later used to enrich the analysis in Chapter 5, creating a feedback loop between both parts of the thesis.	12
3.1	Multi-hop influence process. Given is a directed graph with 11 nodes and 12 arcs (top). Let us assume the k - d DSP is solved with $k = 2$. The two purple nodes (v_4 and v_5) form part of the example solution U . If $d = 1$ (bottom left), then nodes $\{v_3, v_7, v_6\}$ are 1-hop covered by U . If $d = 2$ (bottom center), then nodes $\{v_2, v_3, v_7, v_8, v_6, v_{11}\}$ are 2-hop covered by U . Finally, if $d = 3$ (bottom right), then all remaining nodes of the graph are 3-hop covered by U	26
3.2	Hybridization Process. The integration of BRKGA with FC starts with two offline steps concerning FC as follows. The training phase begins by using 15 random graphs (Erdős–Rényi). This provides us with a trained version of FC (called GNN Framework in the graphic). Then, the social network in which the k - d DSP is to be solved is presented to FC, which returns probabilities for all nodes of the network to belong to the optimal solution. Finally, the final phase consists of integrating these probabilities into the BRKGA (called Genetic Algorithm in the graphic).	30
3.3	Data preparation and pipeline. The pipeline starts by training three FC models, one for each $k \in \{32, 64, 128\}$. Random graphs (Erdős–Rényi) were used for this purpose. Next, the evaluation of the FC models is performed for each of the 19 instances (social networks), for each value of parameter $d \in \{1, 2, 3\}$. Finally, the obtained probabilities (FC output) are exported and stored in text files.	38

3.4 Search trajectory analysis of BRKGA and BRKGA+FC. The three plots display 10 execution trajectories of BRKGA (orange) and BRKGA+FC (pink) on three instances: <i>gplus</i> , <i>twitter-follows</i> , and <i>themarker</i> . The parameter z controls the granularity of search space partitioning used to generate these visualizations (see [155]). Yellow squares mark trajectory start points, gray triangles denote endpoints, light gray circles indicate regions visited by both algorithms, and red circles highlight the best solutions found. (a) BRKGA outperforms BRKGA+FC on <i>gplus</i> . (b) Both algorithms achieve comparable results on <i>twitter-follows</i> . (c) BRKGA+FC outperforms BRKGA on <i>themarker</i> . All visualizations use a force-directed layout based on physical analogies, without assuming any prior network structure.	39
4.1 General framework of the proposed approach.	43
4.2 Diffusion process for threshold $\theta(v) = \left\lceil \frac{\deg(v)}{2} \right\rceil$ for all $v \in V$. The initial target set (single node) is shown in (a).	44
4.3 Feature value distributions for all nodes in scale-free network ($\lambda = 2.25, l = 10$). The x-axis indicates the feature values, and the y-axis shows the number of nodes with each value.	51
4.4 Process of extracting the node probabilities for an unseen graph. First, the feature values (for the five selected features) are calculated for each node of the graph. Then, each node is given as input to the SAGE network, which then provides a node probability as output.	52
4.5 Evolution of the value of the GA elite individual during training. In addition, the objective function value on the basis of the Erdös graphs is used to detect overfitting.	53
5.1 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.	62
5.2 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.	65
5.3 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.	71
5.4 Correlations between all pairs of the five considered metrics concerning the soc-hamsterster network.	78

5.5	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.	79
5.6	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 focused on the essential characteristics.	81
6.1	: A dialogue showing how a chatbot applies our approach to improving optimization algorithms.	85
6.2	Examples of maximum independent sets.	87
6.3	Two LLM interaction patterns: (a) a direct request to improve a heuristic using CMSA's age parameter, and (b) an iterative dialogue to enhance both heuristic quality and C++ performance through error correction. Both use in-context learning as a prompting strategy [58, 115].	92
6.4	Comparative analysis of solution quality: Original CMSA vs. LLM-CMSA variants (V1 and V2).	96
6.5	Critical Difference (CD) plots for different graph types. Algorithms connected by the same horizontal bar do not exhibit a statistically significant difference in performance. (a) Barabási-Albert : The top-performing group, consisting of LLM-CMSA-V1 and LLM-CMSA-V1-PERF, significantly outperforms the lower group (LLM-CMSA-V2 and CMSA). Within each group, performances are statistically equivalent. LLM-CMSA-V2-PERF ranks last. (b) Watts-Strogatz and (c) Erdős-Rényi : In both graph types, LLM-CMSA-V1 and LLM-CMSA-V1-PERF are the top performers and do not differ significantly from each other. Both significantly outperform the other methods, while LLM-CMSA-V2 and CMSA exhibit the lowest performance.	97
6.6	Examples of algorithm evolution over time.	99
7.1	A non-expert user's interaction with an LLM can enhance an existing genetic algorithm by incorporating modern techniques.	105
7.2	Comparison of the metaheuristic codes generated by the five LLMs with the original codes. Remember that the TSP is a minimization problem, that is, the lower the values, the better. The y-axes are shown in a logarithmic scale. 117	117

7.3 Comparison of the reinforcement learning (RL) codes generated by the five LLMs with the original codes. Remember that the TSP is a minimization problem, that is, the lower the values, the better. The y-axes is shown in logarithmic scale.	118
7.4 Comparison of the deterministic heuristic codes generated by the five LLMs with the original codes. The bar plots show the performance gaps (in percent) relative to the original codes. Note that a positive value indicates that the LLM-generated code produces a better solution.	119
7.5 Comparison of the BB codes generated by the five chosen LLMs with the original BB code (in terms of computation time). Each code was applied 30 times, and the y-axis is shown in a logarithmic scale.	120
7.6 Cyclomatic Complexity	124
9.1 Examples of STN graphics comparing three different algorithms applied 10 times to the same problem instance.	138
9.2 The graphic shows the workflow of the original STNs tool. It is divided into three phases (from left to right): (a) a folder must be created, containing a result file for each algorithm to be included in the comparison; (b) then two different R scripts must be executed in a given order, and depending on how many algorithms are included in the comparison (i.e., one versus multiple); (c) finally, the corresponding STN graphics are generated and provided in terms of a PDF file.	139
9.3 At the top is the old input format containing redundant information. The new and simpler version (as implemented in our web application) is below.	140
9.4 The architecture of the original STNs tool, consisting of R scripts, along with its limitations is displayed in the upper part. At the bottom, the new STNs architecture is characterized, consisting additionally of Python scripts, a Rest API, and a website along with additional improvements.	141
9.5 The web application architecture incorporates a Rest API and new features implemented in Python to the original STNs tool. Unlike the original STNs tool, this version requires just one query to the API in order to execute the three phases of the formerly manual process: creating a folder with the input files, executing the corresponding R scripts, and generating a PDF or metrics file.	142
9.6 The layout of the web application in the web browser has three main parts. In the upper part on the left the user can specify the type of optimization problem (maximization vs. minimization) and a number of advanced options (see text). At the bottom left, the user can add one algorithm after the other (clicking the ADD button). For each algorithm, a name, colour and the input data file must be provided. Finally, the right-hand side includes an embedded PDF viewer that shows the STN visualization once the user has clicked on the GENERATE button.	144

9.7 This graphic presents three case studies with different percentages of search space partitioning generated by STNWeb. First, a single algorithm when applied 10 times to the same problem instance is analyzed in the upper row. In the middle row, we can see the behaviour of two algorithms by means of different levels of search space partitioning. Finally, the new STNWeb feature was tested with four algorithms (the original STN tool did not support more than the three algorithms), which is graphically shown in the bottom row.	146
10.1 After completing the configuration form and uploading the required files for each algorithm to be compared (frontend), the user can request the generation of visualizations. The REST API (backend) receives the request and invokes the selected algorithm to partition the space and generate a PDF visualization based on the provided configuration. The resulting PDF is then displayed within the embedded viewer in the frontend.	152
10.2 Interacting with STNWeb requires just three steps. First, generate separate data files with the search trajectories of the algorithms to be compared. Second, configure the analysis and upload the files on the STNWeb interface. Third, generate and download the visualization in PDF format.	153
11.1 Overview of automating graphics interpretation with Large Language Models (LLMs).	158
11.2 Example STN generated by STNWeb for the <i>Rastrigin function</i>	159
11.3 Prompt templates automatically generated by STNWeb for each task.	162
11.4 Example prompts and GPT-4-turbo outputs for the STN in Figure 11.2.	164
11.5 Methodology for evaluating LLMs across tasks.	168
12.1 Example illustrating single-linkage HAC. At each step, the two clusters with the minimum distance (according to a distance metric) are merged. The illustration starts after the first two steps are already performed (to shorten the procedure).	172
12.2 Comparison of standard partitioning vs. agglomerative clustering for MDS. .	178
12.3 Standard search space partitioning vs. agglomerative clustering in the context of 2E-EVRP-TW results.	179
12.4 Standard search space partitioning vs. agglomerative clustering in the context of continuous optimization problems.	180
13.1 A general overview of the seven tasks covered by VisGraphVar (1-7), each representing a different challenge for LVLMs, enabling us to conduct a more detailed performance comparison and evaluation.	187
13.2 Available configurations for generating graph images to evaluate node and edge detection capabilities.	188
13.3 LVLM execution of Task 1 with overlapping nodes and prompt input.	189

13.4 Seven different types of graphs.	190
13.5 Networks with an increasing number of nodes and a single cut-edge: the graph on the left has cut-edge (6, 7); the one in the center has (1, 19); and the most complex one to detect, on the right, has (4, 23).	190
13.6 Three graphs with different types of patterns.	191
13.7 Three types of graphs with different numbers of nodes for which the LVLM is expected to predict a missing link/edge. The missing link on the left is (4,2), the one in the center is (2, 4), and the most complex case is (3, 5), on the right.	192
13.8 Three graphs with varying levels of interpretive difficulty in identifying shortest paths. (a) and (b) are simpler due to the lack of overlap between nodes and edges, whereas (c) makes it very hard to locate each node along a shortest path due to element overlap.	193
13.9 Graph pairs are shown with the goal for the LVLM to identify matches on the left and distinctions on the right. Note that, in this work, two graphs in the same image are said to match if their structure (including node labels) is equal; that is, only their display style might differ. For this reason, the two graphs on the right do not match, even though they are isomorphic.	194
13.10 An overview of LVLM performance across the seven tasks (complete dataset).	199
13.11 The distribution of average scores across the six LVLMs for each task. The violin plot is configured with <code>bw_adjust = 0.5</code> (which adjusts the bandwidth of the kernel density estimation, making the plot more detailed) and <code>cut = 0</code> (which ensures the plot is limited to the range of the data without extending beyond it) using the Seaborn library in Python.	200
13.12 Average LVLM performance (best to worst from left to right) regarding the <code>VisGraphVar</code> dataset.	201
13.13 Average performance of Claude-3.5-Sonnet for each task from the <code>VisGraphVar</code> dataset.	201
13.14 Average scores for each task by prompt strategy, Chain-of-Thought (top) and 0-shot (bottom). Green indicates strong results, while red denotes poor results.	202
13.15 Average performance of Claude-3.5-Sonnet on Task 1 for each considered graph layout.	204
13.16 Image from our dataset (Task 1), showcasing a spectral layout with randomly colored nodes, directed edges, and 10 nodes with 20 edges.	205
13.17 Image from the dataset concerning Task 7 (Matching), containing two structurally equal graphs.	206
13.18 Comparison of the average model performance for graphs with labeled nodes (pink) and graphs with unlabeled nodes (green) in Task 1.	207
13.19 (a) A node-labeled graph with 10 nodes and 16 edges and random node-colors. (b) A similar, un-labeled graph. Both belong to the dataset of Task 1.	208

List of Tables

2.1	Evolution of Combining Large Language Models (LLMs) with Metaheuristics	23
3.1	Tuning configuration. Final parameter setting for BRKGA and BRKGA+FC (for $k \in \{32, 64, 128\}$)	31
3.2	Numerical results obtained by FC, the BRKGA, and our hybrid algorithm BRKGA+FC on 19 well-known social networks. For each network the algorithms were applied for $d \in \{1, 2, 3\}$ and $k \in \{32, 64, 128\}$. For $k = 32$ BRKGA+FC wins in 73% of the cases; for $k = 64$ in 71%; and for $k = 128$ in 66%.	32
4.1	Target Set sizes obtained by the five features for the 20 scale-free networks (training set).	50
4.2	Numerical results for 27 social networks	55
5.1	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.	68
5.2	Number of input/output tokens and the associated cost of processing the input prompts concerning Claude-3-Opus. The costs correspond to March 2024.	70
5.3	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-8x22b-Instruct-v0.1. The six synthetic graphs are chosen as a test bed.	72
5.4	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.	74
5.5	Numerical comparison of three algorithms—BRKGA, BRKGA+FC (results extracted from [32]), 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 $k \in \{32, 64, 128\}$.	75

5.6 Comparison of the LLM output with random values. <code>static</code> refers to a variant of BRKGA+LLM in which the LLM output is replaced by probabilities computed based on random alpha and beta values. <code>dynamic</code> 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.	75
5.7 A numerical comparison of BRKGA+LLM and BRKGA+irace. In the latter algorithm, the alpha and beta values are determined by tuning through <code>irace</code>	76
5.8 The alpha and beta values as determined by <code>irace</code> and the LLM for each case. Pearson's correlation coefficient ($\rho_{\text{irace}, \text{LLM}}$) is used to quantify the relationships between the two sets of values.	76
7.1 Expanded Comparative Analysis of LLM-based Algorithm Design Approaches	104
7.2 Overview of Selected Algorithms for Solving the Travelling Salesman Problem (TSP)	108
7.3 Analysis of the Code Generation Process	111
7.4 Parameter values obtained by tuning with <code>irace</code> . Ranges show minimum/- maximum values considered for tuning.	115
7.5 Average Cyclomatic Complexity of the codes	123
11.1 LLM Tasks for STN Interpretation: Guidelines and Expected Inferences . .	163
11.2 LLMs evaluations for tasks.	167
13.1 Dataset distribution by task.	194
13.2 Performance percentage for each LVLM with the spectral layout.	205

Glossary

k-d Dominating Set Problem (k-dDSP)	An NP-hard influence maximization problem on a directed graph. The goal is to select a set of k nodes that maximizes the total number of unique nodes reachable within a distance of d hops. It is a generalization of the Minimum Dominating Set Problem. (p. 27, 60)
Ant Colony Optimization (ACO)	A metaheuristic inspired by the foraging behavior of ants. It uses a probabilistic mechanism where solutions are constructed based on 'pheromone' values, which represent learned information about the quality of solution components. A specific variant mentioned is the <i>MAX-MIN</i> Ant System (MMAS). (p. 43, 108)
Biased Random-Key Genetic Algorithm (BRKGA)	A metaheuristic that represents solutions as vectors of random keys (real numbers). Its core components include a population of individuals, elite solutions, mutants, and a crossover operator. A problem-specific 'decoder' maps the random keys to a feasible solution. (p. 20, 25, 60)
chain-of-thought reasoning	An emergent ability of LLMs, accessed via prompting, where the model is guided to explain its reasoning step-by-step to arrive at a solution. This is presented as an alternative to zero-shot prompting for improving reasoning in complex tasks. (p. 160)
Construct, Merge, Solve & Adapt (CMSA)	A hybrid metaheuristic (matheuristic) that combines probabilistic construction heuristics with exact solvers, such as Integer Linear Programming. Its four phases are: generating candidate solutions (Construct), aggregating promising components into a subproblem (Merge), finding an optimal solution for the subproblem (Solve), and updating parameters based on the result (Adapt). (p. 87)

cyclomatic complexity	A software metric used to measure the structural complexity of a program by counting the number of linearly independent paths through its source code. Lower values generally indicate simpler, more maintainable code. (p. 123)
decoder	The problem-dependent component of a Biased Random-Key Genetic Algorithm (BRKGA). It is a function that translates an individual's vector of random keys into a valid solution for the specific optimization problem being solved. (p. 27, 61)
few-shot	An emergent ability of Large Language Models (LLMs) where the model is provided with a small number of examples ('shots') within the prompt to guide its response. The text mentions one-shot learning as a specific case where a single illustrative example is given to help the model follow instructions more effectively. (p. 160)
Graph Neural Networks (GNNs)	A type of deep learning model designed to work directly on graph-structured data. GNNs iteratively refine node representations by aggregating information from neighbors, enabling them to learn patterns for tasks like node classification or prediction.. (p. 27, 43)
GraphSAGE	A specific type of Graph Neural Network (GNN) that generates node embeddings by aggregating feature information from a node's local neighborhood. It is used in the text to assign importance scores (probabilities) to nodes. (p. 48)
Hierarchical Agglomerative Clustering (HAC)	A bottom-up clustering strategy used as a search space partitioning method in STNWeb. It starts with each solution as an individual cluster and iteratively merges the two closest clusters based on a distance metric (e.g., single-linkage), subject to constraints on cluster size and volume. (p. 172)
Jaccard Index	A similarity metric used to evaluate the performance of LVLMs on the reasoning task (shortest path finding). It measures the similarity between the set of nodes in the ground truth path and the set of nodes in the path predicted by the model. (p. 196)

Large Language Models (LLMs)	Advanced AI models, typically based on the Transformer architecture, capable of understanding and generating human-like text. In this context, they are used as pattern recognition engines to analyze problem metrics or as assistants to improve existing algorithm code. (p. 57, 102, 156)
Large Vision-Language Models (LVLMs)	Multimodal models capable of analyzing both visual information, such as images of graphs, and text prompts. The text evaluates their ability to perform graph interpretation tasks like detection, classification, segmentation, and reasoning. (p. 184)
Maximum Independent Set (MIS) Problem	A classic NP-hard problem on an undirected graph. The objective is to find the largest possible subset of vertices where no two vertices in the subset are connected by an edge. (p. 85)
prompt engineering	The practice of designing and refining input instructions (prompts) to guide a Large Language Model (LLM) toward producing a desired, high-quality output. The text utilizes a one-shot learning approach as part of this practice. (p. 61, 106, 157)
Q-learning	An off-policy reinforcement learning algorithm. In the text, it is used as an adaptive mechanism to dynamically choose between different solution construction strategies in an Ant Colony Optimization algorithm based on their past performance. (p. 47, 125)
Search Trajectory Networks (STNs)	A visualization technique used to analyze and compare the behavior of metaheuristics. It represents the search process of an algorithm as a directed graph (a trajectory) in the solution space, allowing for insights into exploration and convergence patterns. (p. 136, 151, 157, 181)
Shannon Entropy	A standard method used in STNWeb for partitioning the search space of discrete optimization problems. It involves calculating the entropy for each decision variable based on its values across all solutions, and then removing a percentage of the variables with the lowest information content to merge solutions. (p. 153, 173)

STNWeb	A web application that automates the generation of Search Trajectory Network (STN) graphics. It is designed to improve the usability and accessibility of the original R-script-based STN tool by providing a user-friendly interface and integrating features like advanced search space partitioning for both discrete and continuous optimization problems. (p. 141, 157, 171)
Target Set Selection (TSS)	An optimization problem on an undirected graph where each node has a threshold. The goal is to find a minimum-sized initial set of 'influenced' nodes (a target set) such that influence propagates throughout the entire network according to a diffusion model, like the Linear Threshold (LT) model. (p. 43)
temperature	A parameter that controls the randomness and creativity of the output generated by a Large Language Model. A lower temperature value (e.g., 0.2) makes the output more deterministic and focused, which is suitable for tasks requiring factual accuracy or pattern detection. A higher temperature value (e.g., 0.8 or above) increases the diversity and unpredictability of the response, making it better for creative tasks like writing or brainstorming. (p. 63, 98, 110)
Travelling Salesman Problem (TSP)	A canonical NP-hard combinatorial optimization problem. Given a set of cities and the distances between them, the goal is to find the shortest possible route that visits each city exactly once and returns to the origin city. (p. 103)
VisGraphVar	A configurable benchmark generator designed to create graph images with controlled variations in structure and style. It is used to evaluate the robustness of Large Vision-Language Models (LVLMs) across seven distinct visual graph interpretation tasks, such as node detection, classification, and reasoning. (p. 184)
zero-shot	An emergent ability of Large Language Models (LLMs) to perform a task without being given any specific examples in the prompt. In this approach, the LLM must make its decision based solely on the instructions and data provided. (p. 160, 196)

1

Introduction

While metaheuristics are a type of optimization algorithm extremely powerful for addressing NP-hard optimization problems across a wide range of domains, their effectiveness is fundamentally limited by two core challenges: their lack of insight into the underlying structure of the problem, and their inherent opacity, which complicates meaningful analysis. This thesis addresses both issues by introducing novel methodologies: machine learning techniques are employed to embed structural awareness into metaheuristics, while advanced visualization tools are developed to enhance interpretability. Together, these contributions aim to create more powerful, transparent, and adaptable optimization algorithms.

1.1 Overview of Optimization Algorithms

Not all optimization problems have the same complexity. In fact, many of them are classified as NP-hard, which means that there is no known algorithm capable of solving them efficiently—that is, in polynomial time—while guaranteeing the discovery of the optimal solution. However, it is not only the type of problem that determines its difficulty, but also the specific instance. For example, finding the shortest possible route that visits a set of cities exactly once (as in the Traveling Salesman Problem) is vastly different when dealing with 10 cities compared to 100 or 1,000. As the size of the instance grows, the difficulty posed to the algorithm increases exponentially, resulting in dramatically higher computational time.

Because of this exponential growth, exact algorithms—those that guarantee finding the best possible solution—are often only feasible for small instances or in cases where time constraints are not a concern (which is rare in real-world applications). As a result, over the past few decades, approximation algorithms and heuristic methods have gained significant importance. These approaches aim to find good-enough or near-optimal solutions in a reasonable amount of time, trading some accuracy for much greater efficiency—a compromise that is often essential in practice.

1.1.1 Metaheuristics

Metaheuristics are stochastic approximation algorithms that, by relying on heuristics, forgo the guarantee of finding the global optimum (as exact algorithms do) in exchange for a significant reduction in computational time [22]. This trade-off makes them highly useful for a wide range of optimization problems, hence the prefix “meta”, which implies generality. Unlike purely heuristic algorithms, which are tailored to specific problems, metaheuristics are problem-independent frameworks; thus, a metaheuristic designed for one problem can often be adapted to another with relatively little technical effort, as its underlying principles remain consistent. However, the No Free Lunch theorem [226] dictates that no single metaheuristic can consistently outperform others across all instances of a given optimization problem, regardless of test instance size. Consequently, experimentation takes precedence over theory in this field, with the primary goal being to discover new strategies that improve existing metaheuristics to surpass the current state of the-art. In principle, there is always room for novel approaches to enhance metaheuristic performance.

Thus, over the past years, knowledge from other fields has been increasingly integrated into metaheuristics, giving rise to the area known as hybrid metaheuristics. These hybrid approaches combine different techniques, including but not limited to:

- Integration with exact methods (often referred to as matheuristics), which combine metaheuristic frameworks with exact optimization techniques to improve solution quality and computational efficiency [23].
- Incorporation of machine learning techniques (sometimes called learnheuristics [27]), where learning algorithms are used to adapt, guide, or enhance the metaheuristic search process based on problem characteristics or past experience [16, 102].

This thesis focuses on the second category: leveraging machine learning to enhance existing metaheuristics through various integration strategies. Metaheuristics, by design, lack inherent knowledge of problem structure or specific instance characteristics. This absence prevents them from utilizing historical data to identify patterns that could guide the search toward more promising solution space regions, thus accelerating the discovery of good or near-optimal solutions [16, 102].

Limitations

Metaheuristics exhibit two main drawbacks that prevent them from being more robust algorithms:

1. **Lack of contextual information.** The computational efficiency of metaheuristics largely derives from their use of pseudo-random operators. This inherent stochasticity allows for broad exploration of the search space but also presents

a key limitation: metaheuristics typically lack awareness of the problem’s structure, recurring patterns across instances, or the specific features of the instance at hand. Although some adaptive variants attempt to learn from the search history, they often do so through simplistic performance metrics rather than by explicitly modeling the problem’s topology. In the absence of prior knowledge or contextual clues, these algorithms must rely heavily on exploration—often at the cost of increased computational time compared to strategies guided by problem-specific insights [102].

2. **Lack of interpretability in result analysis.** As mentioned above, metaheuristics often lack strong theoretical guarantees due to their stochastic nature. This makes experimental validation essential, typically involving statistical analyses of multiple runs to compare the performance of one metaheuristic against another. However, numerical comparison alone is not sufficient. Metaheuristics usually lack interpretability—that is, they do not provide clear explanations of *how* or *why* a particular result was achieved, whether it is good or poor [155].

1.2 Contributions

This thesis proposes novel methodologies for integrating metaheuristics with modern strategies. These strategies include approaches from other fields, such as machine learning (specifically Graph Neural Networks and Large Language Models), as well as the development of tools to enhance interpretability.

Thus, the primary focus is not on the specific optimization problem being addressed, but rather on the proposed hybridization strategies themselves, which may later be applied to tackle more complex optimization problems. Nevertheless, as demonstrated throughout this thesis, many of the proposed approaches resulted in improvements over the current state-of-the-art for specific problems.

In general, this thesis makes contributions in two main areas: algorithmic enhancement and the interpretability of optimization algorithm comparisons. These two domains form a feedback loop: interpretability tools support and justify the algorithmic improvements developed in the first area, while those algorithmic advances in turn create new challenges that drive the refinement of interpretability methods (see Figure 1.1). The thesis concludes with a study that explores the philosophical implications of code generation with generative models.

1.2.1 Algorithmic Improvement

Graph Neural Networks (GNNs)

During my first doctoral year, I demonstrated the potential to achieve state-of-the-art results in challenging combinatorial optimization problems, specifically *Multi-Hop Influence Maximization in Social Networks* and *Target Set Selection*. This was accomplished by integrating Graph Neural Networks (GNNs) with two distinct metaheuristics: the Bi-

ased Random-Key Genetic Algorithm (BRKGA) and Ant Colony Optimization (ACO). For detailed methodologies and results, please refer to Chapters 3 and 4.

Large Language Models (LLMs)

Large Language Models (LLMs) are a class of systems based on the transformer architecture, a type of deep neural network that fundamentally differs from Graph Neural Networks (GNNs) [101]. These models are trained on vast corpora of data, enabling them to generate sequences of text tokens. When extended to additional modalities such as audio, images, or video, they are referred to as multimodal models.

In the third year of my doctoral research, I began exploring how LLMs could be used as collaborative assistants for optimization algorithm designers. In Chapter 5, I demonstrate that LLMs can act as powerful pattern recognition engines, particularly for tabular numerical data such as graph-based metrics. By leveraging this capability, we identified novel heuristics that were then integrated into a Biased Random-Key Genetic Algorithm (BRKGA). This hybrid approach outperformed the previous GNN-based solution and achieved state-of-the-art results for the *Multi-Hop Influence Maximization in Social Networks* problem.

In the final stage of my PhD, I shifted the focus of my LLM-related research toward code generation and algorithm enhancement [95]. While most existing studies rely on LLMs to synthesize entirely new metaheuristics through prompt-based programming [165], our approach centered on the augmentation of existing, expert-written code. Specifically, we used the Construct, Merge, Solve & Adapt (CMSA) algorithm—a hybrid metaheuristic implemented by an expert—as contextual input. The LLM was then employed to identify heuristic code fragments that could be optimized or replaced to improve algorithmic performance (see Chapter 6).

Recently, we extended this methodology to a broader range of algorithms, including reinforcement learning, classical heuristics, and exact optimization methods. All experiments were conducted on the benchmark Traveling Salesman Problem, ensuring consistency and comparability across techniques. This line of research demonstrated not only the versatility of LLMs in enhancing diverse optimization strategies, but also their potential to democratize algorithm development. By enabling non-experts to improve complex optimization code through guided suggestions, this approach significantly lowers the barrier to entry in combinatorial optimization. The full study is presented in Chapter 7.

1.2.2 Interpretability Enhancement through Visualization Tools

My work on improving the interpretability of metaheuristic results focuses on the development of a web-based tool called STNWeb, an extended and more user-friendly version of the original Search Trajectory Networks (STNs) tool. STNWeb enables researchers to better justify and enrich their heuristic-based algorithms through intuitive and informative visualizations.

STNWeb

The optimization community, particularly within the realm of stochastic algorithms like metaheuristics, has long highlighted the critical need for visual tools to justify and understand algorithmic results [71, 22]. In this context, Search Trajectory Networks (STNs) emerged as one of the pioneering tools enabling the comparison of multiple algorithm executions on a single problem instance, regardless of whether the problem is discrete or continuous, minimization or maximization. Thus, in cases where numerical comparisons are inconclusive or a clear winner is not apparent, STNs provide an invaluable additional perspective by revealing how a specific problem instance *affects* the behavior of a metaheuristic. This distinction is crucial: STNs do not merely describe how an algorithm generally behaves, but rather illustrate the nuanced impact of particular problem instances on that algorithm. This is vital because a single algorithm can exhibit substantially different behaviors across various optimization problems, and even across different instances of the same problem.

Originally implemented in R, STNs were only accessible via the command line, requiring a certain level of technical expertise to use [155]. A key contribution of this thesis was the development of a web-based version of STNWeb (see Chapters 9 and 10), initiated in the second year of the PhD. This new version not only significantly enhances usability but also introduces several features that expand its potential for broader adoption:

- **Enhanced Interpretability of Visualizations:** We improved the clarity and insight provided by STN graphics. This was achieved, in part, through the integration of Large Language Models (LLMs) to generate natural language explanations and summaries of the visual patterns observed in the networks (see Chapter 11)—a line of work that emerged at the beginning of the third year of the PhD. This integration allows users to quickly grasp complex behavioral dynamics without requiring prior domain knowledge.
- **Improved Visualization Quality:** The visual quality of the STN plots was enhanced by incorporating a novel clustering algorithm. This new algorithm (detailed in Chapter 12) provides a more effective search space partitioning strategy, leading to clearer and less cluttered visualizations, especially for large or complex problem instances. This work was carried out concurrently with the integration of LLMs into STNWeb.
- **Potential for Visual-Spatial Analysis with LVLMs:** STNWeb opens the door for integrating Large Vision-Language Models (LVLMs) to perform direct visual-spatial analysis of the generated images. For instance, an LVLM could analyze an STNWeb image and generate a textual description such as: "In the bottom-right corner, Algorithm_A's trajectories overlap and appear trapped in a sub-optimal region, whereas Algorithm_B's trajectories are dispersed across the image, indicating greater exploratory capacity, which leads to better results." This type of rich textual analysis, derived directly from passing the STNWeb-generated im-

age to an LVLM, is impossible to achieve with only textual prompts (without images). Our initial exploration of the capabilities of LVLMs for analyzing graph images—particularly relevant given that STNWeb produces directed graphs—is presented in Chapter 13. This line of work was developed during the final year of the PhD.

1.2.3 Philosophical Implications

Finally, the progression of research on LLMs over the last two years of the PhD (Chapters 5 to 7, 11 and 13) naturally led to an exploration of the philosophical implications of generative AI. This gave rise to a paper titled *Architectures of Error: A Philosophical Inquiry into AI-Generated and Human-Generated Code*, which is currently under review. In it, I argue for the conceptual importance of distinguishing between the errors produced by LLMs during code generation and those made by human—a distinction that, I contend, has practical implications for programmers. Due to its purely philosophical nature, I wrote this paper independently, thus concluding the body of publications that emerged from my doctoral research.

1.3 Publications Resulting from this Thesis

The publications produced during my PhD are grouped into two categories, reflecting my main research lines. This trajectory culminates in a final paper of a philosophical nature, arising from my broader exploration of AI generative models.

Part I - Algorithmic Improvement

This line of research focuses on leveraging Machine Learning, particularly Graph Neural Networks and Large Language Models, to directly enhance the performance and design of existing metaheuristics.

Preprints / Under Review

- [1] **Combinatorial Optimization for All: Using LLMs to Aid Non-Experts in Improving Optimization Algorithms** (March 2025) [182]. arXiv.
Authors: **Camilo Chacón Sartori**, Christian Blum.
DOI: [10.48550/arXiv.2503.10968](https://doi.org/10.48550/arXiv.2503.10968)
Submitted to a journal for review.
- [2] **Improving Existing Optimization Algorithms with LLMs** (February 2025) [183]. arXiv.
Authors: **Camilo Chacón Sartori**, Christian Blum.
DOI: [10.48550/arXiv.2502.08298](https://doi.org/10.48550/arXiv.2502.08298)
Submitted to a conference for review.

Journal Papers

[3] **Metaheuristics and Large Language Models Join Forces: Toward an Integrated Optimization Approach** (2025) [185]. IEEE Access.
Authors: **Camilo Chacón Sartori**, Christian Blum, Filippo Bistaffa, and Guillem Rodríguez Corominas.
[Code]
DOI: [10.1109/ACCESS.2025.3333333](https://doi.org/10.1109/ACCESS.2025.3333333)

Conference Papers

[4] **Large Language Models for the Automated Analysis of Optimization Algorithms** (2024) [35]. Genetic and Evolutionary Computation Conference (GECCO) 24. **Core A**
Authors: **Camilo Chacón Sartori**, Christian Blum, Gabriela Ochoa.
DOI: [10.1145/3638529.3654086](https://doi.org/10.1145/3638529.3654086)

[5] **Q-Learning Ant Colony Optimization supported by Deep Learning for Target Set Selection** (2023) [171]. Genetic and Evolutionary Computation Conference (GECCO) 23. **Core A**
Authors: Jairo Enrique Ramírez Sánchez, **Camilo Chacón Sartori**, Christian Blum.
DOI: [10.1145/3583131.3590396](https://doi.org/10.1145/3583131.3590396)

[6] **Boosting a Genetic Algorithm with Graph Neural Networks for Multi-Hop Influence Maximization in Social Networks** (2022) [31]. Conference on Computer Science and Intelligence System (FedCSIS). **Core B. Best paper award.**
Authors: **Camilo Chacón Sartori**, Christian Blum.
DOI: [10.1109/FedCSIS55175.2022.9909110](https://doi.org/10.1109/FedCSIS55175.2022.9909110)

Part II - Interpretability Enhancement

This line of research focuses on developing and improving tools and methodologies for better understanding and visualizing the behavior of optimization algorithms.

Journal Papers

[7] **VisGraphVar: A Benchmark Generator for Assessing Variability in Graph Analysis Using Large Vision-Language Models** (2025) [184]. IEEE Access.
Authors: **Camilo Chacón Sartori**, Christian Blum, Filippo Bistaffa.
[Homepage / Dataset / Code]
DOI: [10.1109/ACCESS.2025.4444444](https://doi.org/10.1109/ACCESS.2025.4444444)

[8] **STNWeb: A new visualization tool for analyzing optimization algorithms** (2023) [33]. Software Impacts.
Authors: **Camilo Chacón Sartori**, Christian Blum, Gabriela Ochoa.

[Code]

DOI: [10.1016/j.si.2023.100095](https://doi.org/10.1016/j.si.2023.100095)

Conference Papers

- [9] **An Extension of STNWeb Functionality: On the Use of Hierarchical Agglomerative Clustering as an Advanced Search Space Partitioning Strategy** (2024) [34]. Genetic and Evolutionary Computation Conference (GECCO) 24. **Core A**
Authors: **Camilo Chacón Sartori**, Christian Blum, Gabriela Ochoa.
DOI: [10.1145/3638529.3654084](https://doi.org/10.1145/3638529.3654084)
- [10] **STNWeb for the Analysis of Optimization Algorithms: A Short Introduction** (2024) [181]. Metaheuristics International Conference (MIC).
Authors: **Camilo Chacón Sartori**, Christian Blum.
DOI: [10.1007/978-3-031-62922-8_29](https://doi.org/10.1007/978-3-031-62922-8_29)
- [11] **Search Trajectory Networks Meet the Web: A Web Application for the Visual Comparison of Optimization Algorithms** (2023) [30]. International Conference on Software and Computer Applications (ICSCA).
Authors: **Camilo Chacón Sartori**, Christian Blum, Gabriela Ochoa.
DOI: [10.1145/3587828.3587843](https://doi.org/10.1145/3587828.3587843)

Philosophical Synthesis and Reflection

Preprint / Under Review

- [12] **Architectures of Error: A Philosophical Inquiry into AI-Generated and Human-Generated Code** (May 2025) [180]. SSRN.
Author: **Camilo Chacón Sartori**
DOI: [10.2139/ssrn.5265751](https://doi.org/10.2139/ssrn.5265751)
Submitted to a journal for review.

Total number of produced texts: 12

1.4 Organization

This thesis is structured into two main parts, reflecting my primary research lines (see Figure 1.1). The first part focuses on the algorithmic improvement of metaheuristics using Graph Neural Networks and, predominantly, Large Language Models. The second part is dedicated to the research, creation, and enhancement of STNWeb, a tool designed for analyzing the behavior of metaheuristics in relation to specific problem instances. Each part begins with an introductory chapter that provides a concise review of the state-of-the-art in the respective fields.

1.4.1 Part I - Algorithmic Improvement with Machine Learning (Graph Neural Networks & Large Language Models)

Chapter 2 begins this section with an overview of the different types of machine learning techniques applied to metaheuristics, examining their emergence, strengths, and drawbacks. This helps set the context for what was happening in the field at the time we began proposing new integration methods.

- **Chapter 3: Boosting a Genetic Algorithm using Graph Neural Networks.** This chapter presents my initial work on integrating GNNs with metaheuristics, demonstrating how Graph Neural Networks can enhance the performance of the Biased Random-Key Genetic Algorithm (BRKGA) for a specific combinatorial optimization problem—achieving state-of-the-art results in the *Multi-Hop Influence Maximization problem in Social Networks*.
- **Chapter 4: Improving Ant Colony Optimization supported by Deep Learning.** Building on the previous work, this chapter explores another successful integration, showing how Deep Learning (specifically Q-Learning) can support Ant Colony Optimization (ACO) to achieve state-of-the-art results in the *Target Set Selection* problem.
- **Chapter 5: Large Language Models as Assistants for Enhancing Metaheuristics.** This chapter marks my transition toward the use of LLMs, showcasing their novel application as pattern recognition engines for tabular data derived from graph instances. It introduces a new symbiosis between metaheuristics and LLMs, which led to the discovery of heuristics that significantly boost the performance of BRKGA—ultimately achieving state-of-the-art results (Chapter 3) in the *Multi-Hop Influence Maximization problem in Social Networks*.
- **Chapter 6: Enhancing a CMSA Heuristic for the Maximum Independent Set Problem with Large Language Models and Chapter 7: Improvement of Optimization Algorithms with Large Language Models by Non-expert Users.** These chapters collectively explore the use of LLMs for code generation and improvement of existing optimization algorithms. Chapter 6 details how LLMs can act as collaborators for algorithm designers, suggesting improvements to expert-written code. Chapter 7 expands this concept into a systematic study, showing how LLMs can enable non-experts to enhance a wide range of optimization algorithms for problems like the *Traveling Salesman Problem*.

1.4.2 Part II - Visualization Tools for Algorithm Analysis

This section begins with a brief Chapter 8, which emphasizes the critical role of visual tools in our field. It explains why Search Trajectory Networks (STNs) are not meant to replace numerical comparisons but rather to complement and enrich researchers' analytical capabilities.

- **Chapter 9: Search Trajectory Networks Meet the Web: A Web Application for**

the Visual Comparison of Optimization Algorithms. This foundational chapter presents the initial proposal and development of STNWeb, our web-based application for visualizing and analyzing STNs. It addresses the usability limitations of the original R-script implementation, detailing how STNWeb automates the generation of STN graphics and enhances accessibility for the broader research community.

- **Chapter 10: STNWeb: A New Visualization Tool for Analyzing Optimization Algorithms.** Building upon the initial proposal, this chapter introduces the publicly available version of STNWeb. It highlights how this web-based tool not only overcomes the usability limitations of the original R-script implementation but also incorporates additional functionalities designed to further enhance its adoption and utility for the research community.
- **Chapter 11: Enhancing the Explainability of STNWeb with Large Language Models.** This chapter explores the integration of LLMs directly into STNWeb to automate the analysis of optimization algorithm behavior. It demonstrates how LLMs can generate natural language explanations and insights from STN visualizations, making complex algorithmic dynamics more accessible.
- **Chapter 12: Improving STNWeb Graphical via Hierarchical Clustering-Based Search Space Partitioning.** This chapter details a significant enhancement to STNWeb, introducing a novel search space partitioning scheme based on hierarchical agglomerative clustering to improve the clarity and interpretability of STN visualizations, especially for complex problems.
- **Chapter 13: A Benchmark Generator for Assessing Variability in Graph Analysis Using Large Vision-Language Models.** This chapter introduces a novel benchmark generator designed to rigorously evaluate the visual graph comprehension capabilities of Large Vision-Language Models (LVLMs). It explores how visual variability in graph representations impacts LVLM performance, laying groundwork for future visual analysis applications within STNWeb.

The thesis concludes with Chapter 14, where I provide a brief summary of the work presented, a discussion of the main contributions, limitations, and ongoing research directions.

Note

Each chapter in Parts I and II begins with the research's origin and ends with reflections on its impact—how it influenced later work or shifted its direction. Collectively, these notes highlight the evolving and unpredictable nature of research.

Support

To support readers less familiar with the field, I have included two appendices: Appendix A provides a brief overview of optimization problems, and Appendix B introduces metaheuristics.

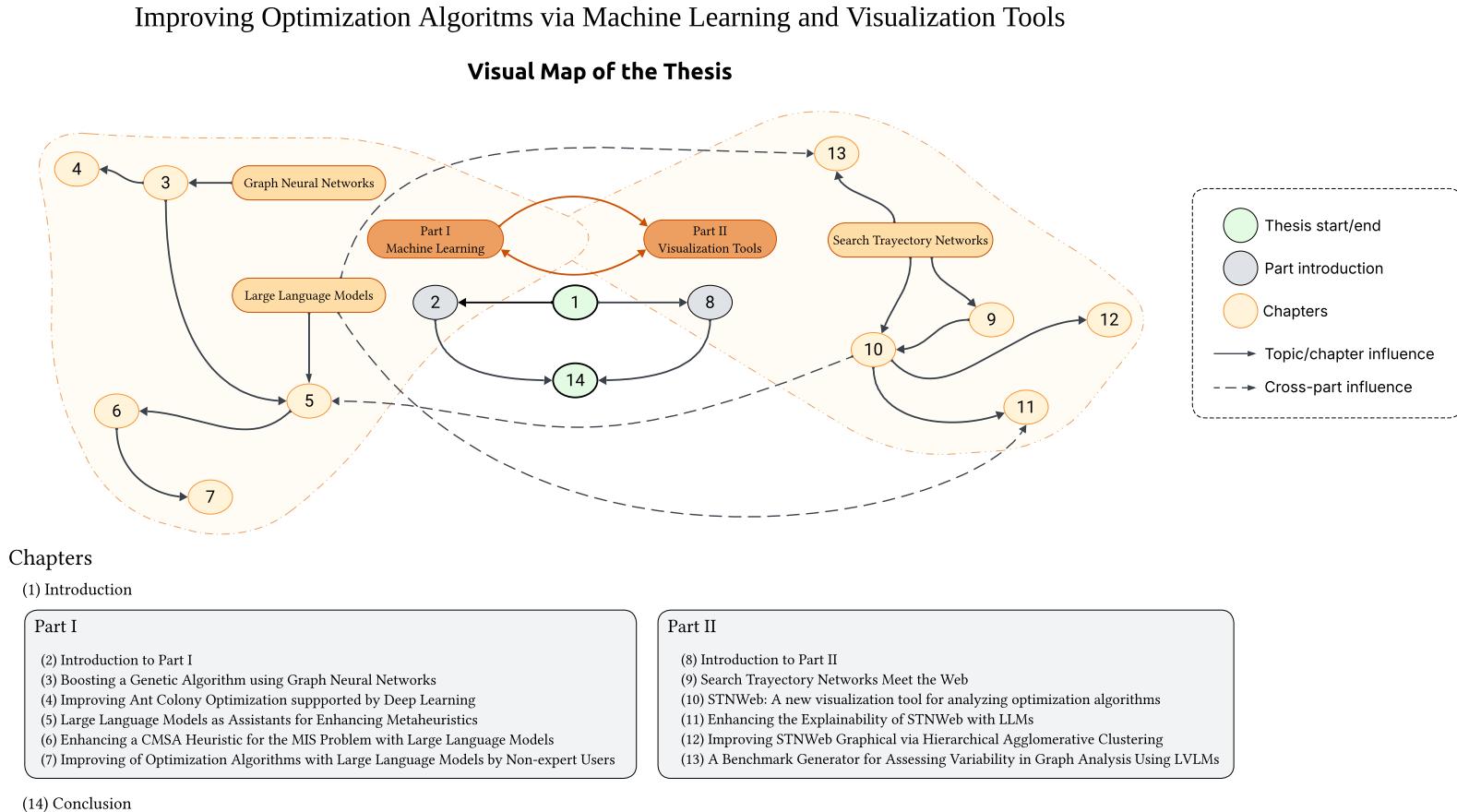


Figure 1.1: A visual map of the thesis structure, where the connections between nodes show the influence of each chapter on the others. For example, node 10 represents the work in which we developed STNWeb; this tool is later used to enrich the analysis in Chapter 5, creating a feedback loop between both parts of the thesis.

Part I
Machine Learning
(Graph Neural Networks & Large Language
Models)

2

Introduction

In this chapter, I begin with a brief review of established machine learning (ML) techniques applied within optimization algorithms, particularly metaheuristics (MH) (Section 2.1). These traditional approaches often rely on expert-driven manual feature selection when constructing models. Section 2.2 introduces deep learning (DL), which replaces this manual effort with automatic pattern extraction from large datasets, enabling neural networks to learn rich representations. Within this section, we also discuss approaches that incorporate reinforcement learning (RL), and especially graph neural networks (GNNs)—a branch of DL tailored to operate directly on graph structures, making it particularly well-suited for combinatorial optimization problems on graphs. My first contribution to the state-of-the-art involves integrating GNNs into metaheuristics, before I transitioned to working with foundational models. Finally, Section 2.3 explores how the emergence of Large Language Models (LLMs) is transforming the field of metaheuristics.

2.1 Metaheuristics and Machine Learning

Optimization algorithms operate based on an objective function and a set of constraints to minimize or maximize a given value. However, they have no prior knowledge of the problem instances before execution and do not incorporate past experience into their operation. In contrast, ML does.

ML serves as a powerful complement to optimization algorithms by filling this gap, providing insights into the structure of the problem. For example, it can analyze the topology of graph instances to determine whether valuable patterns can be learned. This knowledge becomes crucial when tackling larger instances of the same problem—or even real-world scenarios!

Researchers have proposed various strategies to integrate ML into MHs. As [Karimi-Mamaghan et al. \[102\]](#) point out, key approaches for incorporating ML into MHs to solve combinatorial optimization problems (COPs) include:

- **Algorithm Selection:** Using ML to predict the performance of different MHs, enabling the choice of the most suitable algorithm or algorithm portfolio for each problem instance.
- **Fitness Evaluation:** Employing ML to reduce the computational cost of fitness assessments, either by approximating expensive fitness functions (functional approximation) or by minimizing the number of evaluations required (fitness reduction).
- **Initialization:** Leveraging ML to generate high-quality initial solutions or populations, which, in some cases, fosters faster convergence and enhances diversity. This can involve full or partial solution construction based on prior knowledge, or decomposition of the problem space.
- **Evolution:** Integrating ML within the search process to enhance its intelligence through:
 - *Operator Selection:* Dynamically choosing the most effective search operators during optimization based on their past performance.
 - *Learnable Evolution Models (LEMs):* Creating new populations by learning rules from high-quality solutions rather than relying solely on traditional stochastic operators.
 - *Neighbor Generation:* Utilizing insights from good solutions to guide the creation of new neighboring candidates, focusing the search on promising regions.
- **Parameter Setting:** Applying ML techniques to configure MH parameters either offline (parameter tuning before execution) or online (parameter control during search), adapting parameters dynamically to the evolving optimization process.
- **Cooperation:** Enhancing cooperative MH frameworks by enabling intelligent behavior adaptation and information exchange among algorithms, exploiting their complementary strengths.

However, beyond traditional ML methods, recent research has investigated approaches based on DL, GNNs, and LLMs, as explained below.

2.2 Metaheuristics and Deep Learning

Unlike traditional ML, which often relies on extensive manual feature engineering, DL replaces this laborious process with automatic feature extraction from large-scale training data. Its multiple neural network layers are designed to automatically capture complex, hierarchical patterns that are often beyond the reach of simpler ML models. In essence, DL excels at processing vast data volumes and uncovering intricate structures within them, making it a powerful tool for various applications, including optimization.

2.2.1 Deep Reinforcement Learning (DRL) in Metaheuristics

Deep Reinforcement Learning (DRL) combines the learning power of deep neural networks with the decision-making framework of RL. This synergy has proven highly effective for enhancing metaheuristics in several key areas established previously:

- **Initialization:** A prominent application of DRL is in generating high-quality initial solutions. For instance, [Miki et al. \[145\]](#) applied DRL to the Traveling Salesman Problem (TSP), training a model on millions of instances to learn a policy that significantly improved the starting quality for traditional metaheuristics.
- **Learning Heuristic Functions and Policies:** Beyond initialization, DRL can learn complex search policies. [Huber et al. \[90\]](#) used it to guide beam search for the Longest Common Subsequence problem. In a more advanced application, [Fenoy et al. \[64\]](#) combined an RL-trained attention model with classical optimization to form agent collectives, allowing for coordinated search strategies.
- **Dynamic Operator and Parameter Control:** DRL is well-suited for dynamic adaptation during the search. Notable works include RL-guided variable neighborhood search [\[5\]](#), where RL selects the most effective neighborhood structures, and its use in adaptive parameter tuning for algorithms like Biased Random-Key Genetic Algorithm (BRKGA) [\[38\]](#).

2.2.2 Graph Neural Networks for Combinatorial Optimization

GNNs are a specialized DL architecture ideal for problems with inherent graph structures. By operating directly on non-Euclidean data, they can learn rich representations of the problem's topology, providing powerful guidance to metaheuristics.

- **Guiding the Search Process:** A key example, presented in this thesis (see Chapter 3), involves training a GNN to guide a BRKGA. The GNN exploits the graph's structural properties to steer the search toward more promising regions. A related approach integrates Q-learning with an Ant Colony System to similarly enhance search guidance [\[171\]](#) (see Chapter 4).
- **Enhancing Search Operators:** GNNs can also improve core components of local search. [Liu et al. \[122\]](#) employed GNNs to enhance neighborhood selection in Tabu Search and Large Neighborhood Search, where the GNNs learned to identify more effective moves based on the underlying graph structure.

2.3 Metaheuristics and Large Language Models

Over the past two years, the optimization community has explored LLMs with the hope that their vast pre-training knowledge can beneficially contribute to improving or even generating optimization algorithms. As we will demonstrate, LLMs have been leveraged to guide optimization processes, detect patterns and key features in problem

instances, refine search spaces, and generate new problem-specific heuristics. Furthermore, LLMs offer valuable insights by explaining optimization results, making them versatile tools for both problem-solving and interpretation.

2.3.1 LLMs for Improving Solution Quality

For a concise overview, Table 2.1 presents the main contributions in this emerging hybridization approach.

Early Explorations (2023): Direct Solving and Simple Integration

A foundational work on LLM-optimization synergy is “Large Language Models as Optimizers” by Google DeepMind [235], published in September 2023. This influential paper introduced Optimization by PROmpting (OPRO), demonstrating LLMs’ ability to solve small-scale optimization problems through natural language instructions.

However, directly tasking an LLM with complex or large-scale optimization remains challenging. Consequently, researchers sought alternative methods to leverage LLMs for more demanding optimization tasks. A month later, the Large Language Model-based Evolutionary Algorithm (LMEA) emerged [126], which integrated an LLM within an evolutionary algorithm, shifting the paradigm from a standalone solver to an intelligent component in a hybrid system.

Advanced Hybrid Approaches (2024): LLMs as Core Components

The landscape shifted significantly in early 2024 with the advent of larger, more capable LLMs. The first half of the year saw two key hybrid methodologies emerge, both successfully benchmarked against state-of-the-art algorithms.

As part of the research for this doctoral thesis, in May 2024, *OptiPatterns* [185] was published, a work that leverages leading LLMs as pattern-detection engines. In our approach, instead of direct solution generation, the LLM analyzes problem instances to extract relevant structural information and distill it into parameters that guide an existing genetic algorithm. To validate this methodology, we integrated it with the BRKGA for the *Multi-Hop Influence Maximization* problem, evaluating it on real-world social networks of up to 7,000 nodes. The results showed that our method surpassed the previous state-of-the-art (our GNN-enhanced BRKGA). This work also pioneered the incorporation of open-weight LLMs in experiments, although at the time, they did not match the quality of proprietary models.

Days after the publication of *OptiPatterns*, LLAMEA (Large Language Model Evolutionary Algorithm for Automatically Generating Metaheuristics) [194] appeared on arXiv. It adopts a different but complementary strategy, using an LLM-driven evolutionary process to automatically generate entirely new metaheuristics.

Emerging Paradigms: Augmentation vs. Generation

These developments, **including our contribution with *OptiPatterns*** (see Chapter 5), have crystallized into two distinct integration areas. The first is Algorithm Augmentation, where an LLM enhances a pre-existing, expert-designed algorithm by providing it with data-driven insights. The second is Algorithm Generation, where an LLM-powered framework autonomously creates novel metaheuristic algorithms from scratch (see Chapters 6 and 7).

Table 2.1: Evolution of Combining Large Language Models (LLMs) with Metaheuristics

Study / Acronym	Timeline	Key Publication(s)	Core Methodology	Integration Paradigm
<i>2023: Early Explorations</i>				
OPRO	Sep 2023	Yang et al. [235]	Uses an LLM to directly solve small-scale optimization problems through natural language instructions (prompting).	Direct Solving
LMEA	Oct 2023	Liu et al. [126]	Integrates an LLM as an intelligent component within a traditional evolutionary algorithm framework.	Algorithm Augmentation
<i>2024: Advanced Hybrid Approaches</i>				
OptiPatterns	May 2024	Our article [185]	Leverages an LLM as a pattern-detection engine to analyze problem instances and distill guiding parameters for a metaheuristic (e.g., BRKGA).	Algorithm Augmentation
LLaMEA	May 2024	Stein et al. [195]	Employs an LLM-driven evolutionary process to automatically generate entirely new metaheuristic algorithms from scratch.	Algorithm Generation

3

Boosting a Genetic Algorithm using Graph Neural Networks

Foundational Work for This Chapter

This chapter is based on the following publication:

- **Title:** *Boosting a Genetic Algorithm with Graph Neural Networks for Multi-Hop Influence Maximization in Social Networks*
- **Published in:** Federated Conference on Computer Science and Information Systems (FedCSIS) - Core B
- **Type:** Conference Paper
- **Year:** 2022
- **Metaheuristic used:** Biased Random Key Genetic Algorithm (BRKGA)
- **Main contribution:** Leveraging Graph Neural Networks to identify patterns that steer BRKGA towards more promising search spaces
- **Problem addressed:** Multi-Hop Influence Maximization in Social Networks
- **Type of contribution:** Algorithmic & Methodological
- **Has it improved the state-of-the-art?** Yes
- **DOI:** <http://dx.doi.org/10.15439/2022F78>
- **Current number of citations in Google Scholar:** 6

This work received the *Best Paper Award* at the Workshop on Computational Optimization at FedCSIS 2022, held in Sofia, Bulgaria.

3.1 Introduction

This work marked my first research project in which I integrated ML techniques—specifically, Deep Learning methods such as Graph Neural Networks (GNNs)—into a metaheuristic. The initial idea stemmed from a recommendation by my PhD supervisor, who pointed me to a recent framework named *FastCover* [151]. This framework leverages GNNs to solve the *multi-hop influence maximization* problem in social networks. My contribution involved adapting FastCover and embedding it into a Biased Random-Key Genetic Algorithm (BRKGA).¹ The resulting hybrid algorithm outperformed the state-of-the-art for this problem. However, as I will discuss in Chapter 5, there is a simpler approach that does not rely on GNNs and achieves even better results.

This work received the *Best Paper Award* at the Workshop on Computational Optimization at FedCSIS, held in Sofia, Bulgaria.

Optimization problems over graphs are central to many real-world applications, from social network analysis and epidemic modeling to infrastructure planning and recommendation systems. Among these, selecting a small yet influential subset of nodes capable of exerting a wide-reaching effect over the network is a fundamental challenge. This is especially relevant in scenarios where spreading information, influence, or control efficiently is critical. The problem we address in this work belongs to this category and extends classical formulations by incorporating directional constraints and influence dynamics. To better understand the underlying challenge addressed by our algorithm, we now provide a formal definition of the optimization problem in question. It belongs to the family of Influence Maximization (IM) problems, and can be seen as a generalization of the classical Minimum Dominating Set Problem (MDSP) on a directed graph $G = (V, A)$. In the MDSP, the goal is to find a subset $U \subseteq V$ of minimum size such that every node $v \in V$ is either in U or directly reachable from some $v' \in U$ —i.e., $(v', v) \in A$. In other words, MDSP focuses on *one-hop coverage*.

The specific variant we tackle—known as the *k-d Dominating Set Problem* (*k-dDSP*)—extends this concept to *multi-hop influence*. Instead of minimizing the size of U , we fix a budget k and a hop distance d , and aim to select k nodes such that the number of influenced nodes is maximized. A node v is considered influenced (or covered) if it lies within d hops from at least one node in U .

3.2 Problem Definition

As an illustrative example, consider Figure 3.1, where $k = 2$ and the selected nodes (colored in purple) are $U = \{v_4, v_5\}$. If $d = 1$, the covered set is $C_U = \{v_4, v_5, v_3, v_6, v_7\}$, since those nodes lie within one hop of a node in U , yielding an objective value of 5.

¹ The choice of BRKGA over other metaheuristics is purely empirical. In our experiments, it proved to be easy to implement and capable of delivering satisfactory results for this problem.

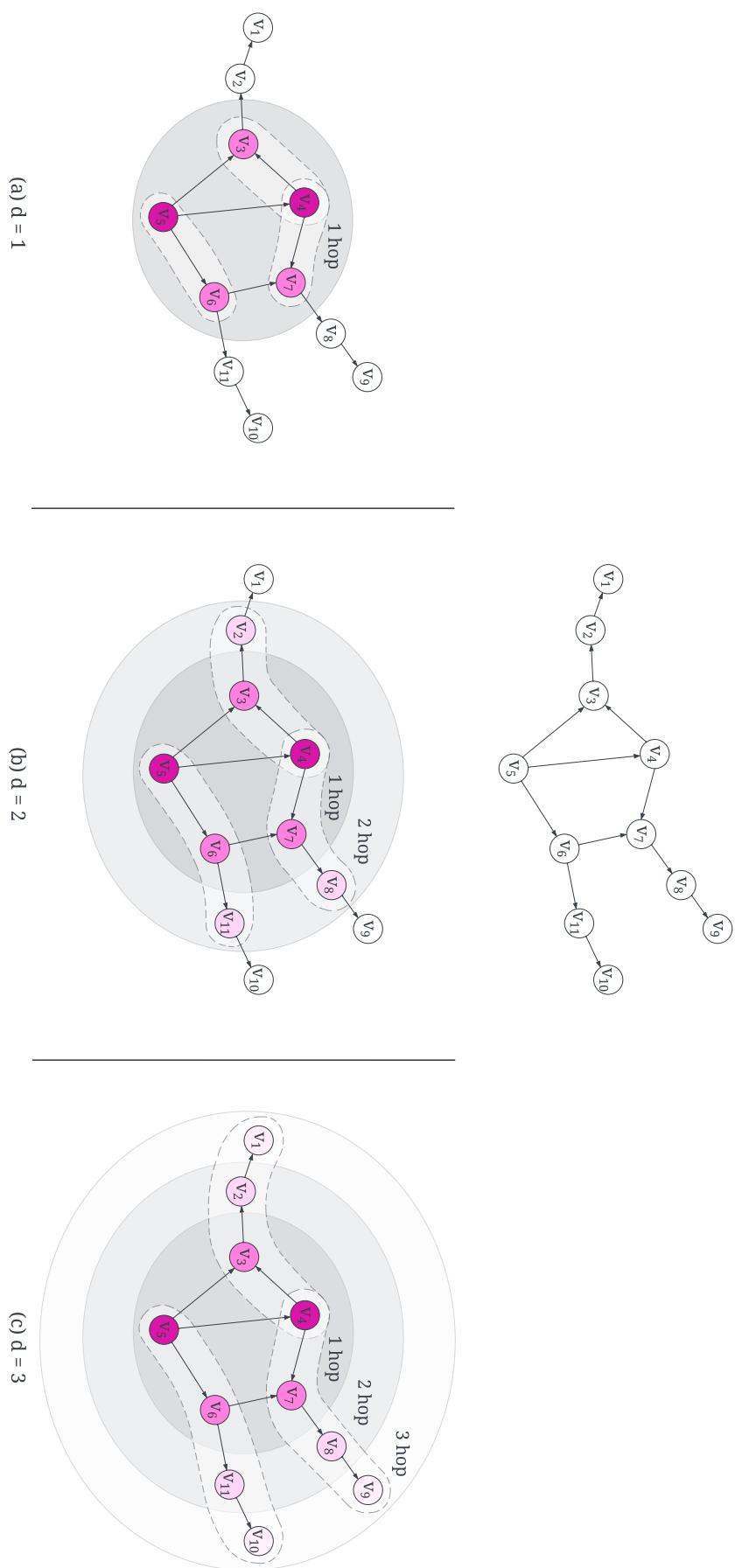


Figure 3.1: Multi-hop influence process. Given is a directed graph with 11 nodes and 12 arcs (top). Let us assume the k -dDSP is solved with $k = 2$. The two purple nodes (v_4 and v_5) form part of the example solution U . If $d = 1$ (bottom left), then nodes $\{v_3, v_7, v_6\}$ are 1-hop covered by U . If $d = 2$ (bottom center), then nodes $\{v_2, v_3, v_7, v_8, v_6, v_{11}\}$ are 2-hop covered by U . Finally, if $d = 3$ (bottom right), then all remaining nodes of the graph are 3-hop covered by U .

If $d = 2$, the covered set becomes $C_U = \{v_4, v_5, v_3, v_6, v_7, v_2, v_8, v_{11}\}$, increasing the objective to 8. Finally, if $d = 3$, all nodes in V are covered, and the objective reaches its maximum value of 11.

Many optimization problems in social networks can be modeled by representing the network as a directed graph $G = (V, A)$, where V is the set of nodes and A the set of arcs. This is the case for the multi-hop influence maximization problem addressed in this work, known as the k - d Dominating Set Problem (k - d DSP), as mentioned already above.

The core notion is the *influence set* $I_d(u) \subseteq V$ of a node $u \in V$, defined as:

$$I_d(u) := \{v \in V \mid \text{dist}(u, v) \leq d\} \quad (3.1)$$

where $\text{dist}(u, v)$ is the length (number of arcs) of the shortest directed path from u to v . That is, $I_d(u)$ contains all nodes reachable from u within d hops.

This definition extends naturally to a set of nodes $U \subseteq V$:

$$I_d(U) := \bigcup_{u \in U} I_d(u) \quad (3.2)$$

A node is considered influenced if it belongs to $I_d(U)$.

A solution to the k - d DSP is any subset $U \subseteq V$ with $|U| \leq k$. The objective is to find a set U^* that maximizes the number of influenced nodes:

$$\begin{aligned} \max_{U \subseteq V} \quad & |I_d(U)| \\ \text{s.t.} \quad & |U| \leq k \end{aligned} \quad (3.3)$$

The k - d DSP is known to be NP-hard [151, 13].

3.3 Methodology

In this section, we introduce a novel hybrid algorithm that combines a Biased Random-Key Genetic Algorithm (BRKGA) [74] with a Graph Neural Networks (GNNs) framework to solve the k - d DSP in social networks. We begin by briefly describing each algorithmic component. Then, we detail the hybridization strategy that integrates them into a unified optimization approach.

3.3.1 Biased Random Key Genetic Algorithm

We implemented a BRKGA, a well-established variant of genetic algorithms for combinatorial optimization. BRKGA is problem-independent in its core structure, as it operates on populations of individuals represented by vectors of real numbers (random keys). The problem-specific component lies in the decoder, which maps these vectors to feasible solutions of the target problem. The general, problem-independent procedure of BRKGA is outlined in Algorithm 1.

Algorithm 1 The pseudo-code of BRKGA

Require: a directed graph $G = (V, E)$
Ensure: values for params. $p_{size}, p_e, p_m, prob_{elite}, seed$

- 1: $P \leftarrow \text{GENERATEINITIALPOPULATION}(p_{size}, seed)$
- 2: $\text{EVALUATE}(P)$ ▷ dependent part (greedy)
- 3: **while** computation time limit not reached **do**
- 4: $P_e \leftarrow \text{ELITESOLUTIONS}(P, p_e)$
- 5: $P_m \leftarrow \text{MUTANTS}(P, p_m)$
- 6: $P_c \leftarrow \text{Crossover}(P, p_e, prob_{elite})$
- 7: $\text{EVALUATE}(P_m \cup P_c)$ ▷ dependent part (greedy)
- 8: $P \leftarrow P_e \cup P_m \cup P_c$
- 9: **end while**
- 10: **return** Best solution in P

In the following, we first describe the independent or generic part of the algorithm. It starts by invoking function $\text{GenerateInitialPopulation}(p_{size}, seed)$, which generates a population P formed by p_{size} individuals. In case $seed = 0$, all p_{size} individuals are randomly generated. Hereby, each individual $\pi \in P$ is a vector of length $|V|$, where V is the set of nodes from the input graph. For this purpose, the value at position i of π , denoted by $\pi(i)$, is chosen uniformly at random from $[0, 1]$, for all $i = 1, \dots, |V|$. In case $seed = 1$, only $p_{size} - 1$ individuals are randomly generated. The last individual is obtained by defining $\pi(i) := 0.5$ for all $i = 1, \dots, |V|$. Next, the individuals from the initial population are evaluated. This means, each individual $\pi \in P$ is transformed into a valid solution U_π to the k -dDSP, and the value $f(\pi)$ of π is defined as follows: $f(\pi) := |U_\pi|$. The transformation of individuals to valid solutions is discussed below.

Then, at each iteration of the algorithm, the operations to be performed are as follows. First, the best $\max\{\lfloor p_e \cdot p_{size} \rfloor, 1\}$ individuals are copied from P to P_e in function $\text{EliteSolutions}(P, p_e)$. Second, a set of $\max\{\lfloor p_m \cdot p_{size} \rfloor, 1\}$ so-called mutants are generated and stored in P_m . These mutants are random individuals generated in the same way as the random individuals from the initial population. Finally, a set of $p_{size} - |P_e| - |P_m|$ individuals are generated by crossover in function $\text{Crossover}(P, p_e, prob_{elite})$ and stored in P_c .

Each such individual is generated as follows: (1) an elite parent π_1 is chosen uniformly at random from P_e , (2) a second parent π_2 is chosen uniformly at random from $P \setminus P_e$, and (3) an offspring individual π_{off} is generated on the basis of π_1 and π_2 and stored in P_c . In the context of the crossover operator, value $\pi_{off}(i)$ is set to $\pi_1(i)$ with probability $prob_{elite}$, and to $\pi_2(i)$ otherwise. After generating all new offspring in P_m and P_c , these new individuals are evaluated in function $\text{Evaluate}()$; see line 7. Note that the individuals in P_e are already evaluated. Finally, the population of the next generation is determined to be the union of P_e with P_m and P_c .

The evaluation of an individual (see lines 2 and 7 of Algorithm 1) is the problem-dependent part of our BRKGA algorithm. The function that evaluates an individual is often called the *decoder*. In our case, we make use of a simple greedy heuristic which

is based on the intuition that nodes with a higher degree (number of neighbors) are more likely to have a high influence than nodes with a lower degree. Hereby, the set of neighbors $N(v_i)$ of a node $v_i \in V$ is defined as follows: $N(v_i) := \{v_j \in V \mid (v_i, v_j) \in A\}$, that is, neighbors of v_i are only those nodes that can be reached via a directed arc from v_i . The greedy value $\phi(v_i)$ of each $v_i \in V$ is defined as follows:

$$\phi(v_i) := |N(v_i)| \cdot \pi(i) \quad (3.4)$$

In other words, the greedy value of a node v_i is computed as the product of its degree and the value at position i in the individual being decoded. The solution U_π is then constructed by selecting the k nodes with the highest greedy values.

As we will see in Section 3.3.3, the greedy function ϕ will be modified to incorporate information from the GNN, resulting in a hybrid algorithm.

3.3.2 Graph Neural Network Framework

Graph Neural Networks (GNNs) [227, 229, 233] aim to automatically learn meaningful patterns from data represented as graphs. Unlike classical deep learning models, which operate on *Euclidean domains* such as images (regular grids) or sequences (ordered vectors in \mathbb{R}^n), GNNs are designed to work directly with *non-Euclidean, graph-structured data*. This allows them to make predictions at the level of nodes, edges, or subgraphs without requiring preprocessing steps that flatten or distort the underlying graph topology.

The core idea behind GNNs is to iteratively refine the representation of each node by aggregating information from its neighbors and combining it with its own representation. Given a graph $G = (V, A)$, each GNN layer $l \in \{0, 1, \dots, L\}$ maintains a node feature matrix $H^l \in \mathbb{R}^{|V| \times C}$, where each row corresponds to the representation of a node, and C is the number of features. The goal of the GNN is to learn expressive node embeddings through a series of message-passing iterations.

At each layer, two operations are performed:

1. *Aggregate*: gather information from neighboring nodes;
2. *Combine*: update the node's current representation using the aggregated information.

This process can be formalized as:

$$\begin{aligned} a_v^l &= \text{AGGREGATE}^l\{H_u^{l-1} \mid u \in N(v)\} \\ H_v^l &= \text{COMBINE}^l(H_v^{l-1}, a_v^l) \end{aligned}$$

where $N(v)$ denotes the set of neighbors of node v . Once trained, the final node representations H^L can be used for various downstream tasks.

In the context of the k -dDSP, GNNs can be trained to estimate the likelihood that each node belongs to an optimal solution. As mentioned earlier, such an approach

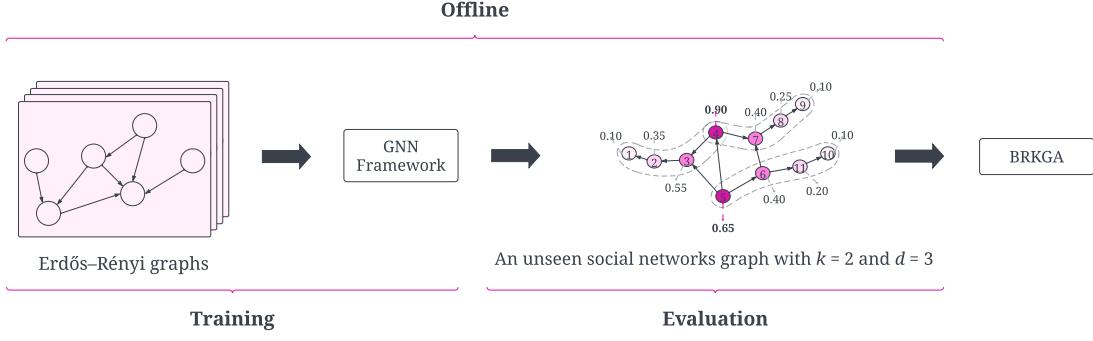


Figure 3.2: Hybridization Process. The integration of BRKGA with FC starts with two offline steps concerning FC as follows. The training phase begins by using 15 random graphs (Erdős–Rényi). This provides us with a trained version of FC (called GNN Framework in the graphic). Then, the social network in which the k - d DSP is to be solved is presented to FC, which returns probabilities for all nodes of the network to belong to the optimal solution. Finally, the final phase consists of integrating these probabilities into the BRKGA (called Genetic Algorithm in the graphic).

was proposed in [151], introducing the FASTCOVER (FC) framework—an unsupervised GNN tailored for this problem. FC is based on a *graph reversed attention network* (GRAT) [151], and it operates as follows: (1) node features are embedded in a vector space, and all arcs in the graph are reversed; (2) the GRAT computes, via a multi-layer GNN, a real-valued score in $[0, 1]$ for each node; and (3) a differentiable loss function over the node scores guides the learning during training.

The core innovation of FC lies in the design of the GRAT layer. Unlike standard Graph Attention Networks (GATs) [211], which apply attention at the destination nodes, GRAT applies it at the source nodes. This shift emphasizes the idea that nodes with greater influence should receive stronger reinforcement, thereby increasing their potential score in the solution space.

3.3.3 The Hybrid BRKGA Algorithm

Our hybrid algorithm, BRKGA+FC, begins with two offline steps. First, given a network for the k - d DSP, node probabilities $p_i \in (0, 1]$ are extracted from the pretrained FC model (training details follow in the next section). Then, the original greedy function $\phi()$ (Eq. 3.4) is modified to incorporate these probabilities:

$$\phi_{FC}(v_i) := |N(v_i)| \cdot \pi(i) \cdot p_i, \quad \forall v_i \in V \quad (3.5)$$

The intuition is that accurate predictions guide the search toward regions containing optimal or near-optimal solutions. Additionally, the FC probabilities help correct the bias from node degree, which can sometimes be misleading. The integration process is illustrated in Figure 3.2.

3.4 Experimental Evaluation

This section is organized into three parts. First, we describe data preparation for training and evaluation, along with parameter tuning. Next, we present the experimental setup and numerical results for three algorithms: FC, BRKGA, and the hybrid BRKGA+FC. Note that FC can also be used standalone by selecting the k nodes with the highest probabilities. Finally, we provide a graphical analysis of the algorithms using STNs.

Table 3.1: Tuning configuration. Final parameter setting for BRKGA and BRKGA+FC (for $k \in \{32, 64, 128\}$)

Parameters	Tuning domain	BRKGA			BRKGA+FC		
		$k = 32$	$k = 64$	$k = 128$	$k = 32$	$k = 64$	$k = 128$
P_{size}	[50, 250]	113	162	132	183	198	137
P_e	[0.1, 2.0]	0.17	0.24	0.25	0.19	0.22	0.2
P_m	[0.3, 5.0]	0.27	0.22	0.14	0.3	0.21	0.21
$prob_{elite}$	[0.01, 0.1]	0.6	0.59	0.58	0.57	0.67	0.67
$seed$	{0, 1}	0	0	0	1	1	1

Table 3.2: Numerical results obtained by FC, the BRKGA, and our hybrid algorithm BRKGA+FC on 19 well-known social networks. For each network the algorithms were applied for $d \in \{1, 2, 3\}$ and $k \in \{32, 64, 128\}$. For $k = 32$ BRKGA+FC wins in 73% of the cases; for $k = 64$ in 71%; and for $k = 128$ in 66%.

Instance	V	E	d	k = 32			k = 64			k = 128		
				FC	BRKGA	BRKGA+FC	FC	BRKGA	BRKGA+FC	FC	BRKGA	BRKGA+FC
advogato	6551	51332	1	2338	2464.13	2469.13	2865	2948.67	2948.90	3313	3340.17	3372.33
			2	4069	4139.83	4132.30	4153	4206.83	4207.77	4220	4266.97	4251.13
			3	4268	4279.67	4275.47	4275	4281.80	4280.00	4277	4301.07	4284.00
anybeat	12645	67053	1	8556	8566.80	8570.70	9045	8981.40	9002.83	9650	9537.60	9626.47
			2	11104	11177.10	11205.63	11209	11300.53	11305.83	11384	11371.47	11400.77
			3	11507	11527.17	11526.00	11515	11531.00	11530.33	11556	11542.27	11546.87
brightkite	56739	212945	1	1266	1714.33	1808.67	1954	2483.03	2640.27	3023	3448.00	3711.63
			2	4018	4160.63	4671.50	5444	5088.90	5910.40	6795	6075.13	6891.07
			3	6094	5699.57	6535.13	7530	6349.90	7614.10	8650	7178.47	8189.63
delicious	536108	1365961	1	8522	10860.00	10864.83	12431	15793.53	15792.90	19483	21044.43	21170.37
			2	21119	22341.80	22481.57	26018	26909.07	26995.33	32248	32811.27	33414.13
			3	32000	33041.13	33175.07	36112	36039.43	36328.03	40309	40418.77	41722.63
douban	154908	327162	1	1093	1503.83	1482.30	2117	2649.93	2637.90	3950	4557.10	4565.73

Table 3.2 – continued from previous page

Instance	V	E	d	k = 32			k = 64			k = 128		
				FC	BRKGA	BRKGA+FC	FC	BRKGA	BRKGA+FC	FC	BRKGA	BRKGA+FC
epinions	26588	100120	2	4147	6809.23	6743.50	6801	9516.80	9594.53	10583	12950.27	13093.53
			3	11686	13988.10	14448.00	15938	17548.57	17866.60	20720	21277.43	22368.23
			1	1532	1753.27	1774.70	2198	2333.10	2413.17	3019	3000.47	3170.93
			2	3645	3711.43	3853.17	4271	4086.47	4416.07	4904	4549.13	4897.77
			3	4500	4487.73	4634.20	4948	4661.37	5002.83	5430	4973.10	5334.60
gowalla	196591	950327	1	1998	3296.13	3384.73	3415	4869.60	5122.30	5553	6976.07	7403.00
			2	7509	9723.47	10754.63	10734	12534.07	13628.10	15386	14888.50	17251.67
			3	14247	14913.00	16657.80	18418	17386.73	18966.73	23692	19064.10	22800.10
gplus	23628	39242	1	17498	18077.00	17896.00	22138	22496.93	22167.90	23543	23628.00	23567.00
			2	21277	23077.20	22726.63	23200	23562.93	23172.73	23628	23628.00	23628.00
			3	21636	23271.37	22884.60	23271	23559.80	23169.00	23628	23628.00	23628.00
			1	8778	9041.33	9047.27	11232	11719.57	11722.57	14749	15058.97	15128.27
loc-brightkite	58228	214078	2	37295	38212.10	38267.47	40161	41258.13	41190.97	42929	43777.03	43827.50
			3	52335	52645.00	52744.00	53272	53600.17	53469.27	53783	54123.80	54134.60

Table 3.2 – continued from previous page

Instance	V	E	d	k = 32			k = 64			k = 128		
				FC	BRKGA	BRKGA+FC	FC	BRKGA	BRKGA+FC	FC	BRKGA	BRKGA+FC
sign-Slashdot081106	77350	516575	1	7162	8087.00	8087.00	11362	11999.33	12003.93	17046	17514.80	17524.33
			2	37521	42221.13	42352.03	44291	47782.33	47827.97	47747	51839.47	51796.63
			3	59456	60393.67	60367.30	60709	61148.47	61148.07	61333	61683.10	61701.40
sign-Slashdot090216	81867	545671	1	7232	8127.87	8128.00	11385	12094.27	12108.50	16949	17592.80	17613.43
			2	39841	43723.57	43781.93	46021	49774.13	49832.23	49661	54244.47	54141.53
			3	62964	63840.83	63817.00	64209	64710.97	64723.77	64912	65375.13	65399.37
sign-Slashdot090221	82140	549202	1	7182	8129.00	8129.00	11421	12129.03	12126.33	17010	17642.67	17641.80
			2	39220	43869.37	43982.53	46410	49972.23	49968.17	49917	54408.47	54334.93
			3	62958	64062.17	64036.97	64473	64935.37	64953.90	65145	65589.77	65598.23
sign-bitcoinotc	5881	35592	1	3455	3479.00	3479.00	4010	4038.17	4040.97	4615	4595.70	4617.97
			2	5568	5631.97	5632.60	5645	5715.37	5715.20	5761	5761.83	5781.17
			3	5814	5838.00	5838.03	5834	5839.00	5839.10	5844	5842.00	5844.00
sign-epinions	131828	841372	1	17765	18690.03	18693.50	22933	23569.77	23609.43	28969	29052.87	29284.87
			2	56849	59372.80	59411.70	60208	62238.23	62288.90	63070	64153.33	64309.37

Table 3.2 – continued from previous page

Table 3.2 – continued from previous page

Instance	V	E	d	k = 32			k = 64			k = 128		
				FC	BRKGA	BRKGA+FC	FC	BRKGA	BRKGA+FC	FC	BRKGA	BRKGA+FC
	2	2328	2354.73		2355.10		2341	2390.00	2388.03	2366	2454.50	2427.53
				3	2331	2357.10	2357.27	2344	2389.53	2389.67	2366	2452.23

3.4.1 Data Preparation and Tuning Process

We conducted experiments for three values of k : $\{32, 64, 128\}$. Accordingly, we trained one FC model per k , all using a fixed hop parameter $d = 1$ (see Figure 3.3). This single model was applied to FC and BRKGA+FC for all $d \in \{1, 2, 3\}$ to reduce computational cost. However, as our analysis will show, this choice slightly degraded the accuracy of node probabilities for larger d values.

Each FC model was trained on 15 Erdős–Rényi graphs [62] with 4000 nodes, following the setup in [151]. After training, node probabilities for all 19 social networks used in the final evaluation (across $d \in \{1, 2, 3\}$) were extracted and saved.

To ensure fair evaluation, BRKGA and BRKGA+FC were tuned separately for each k on 10 Erdős–Rényi test graphs², each with $n = 25,000$ nodes and arc probability $p = 10/n$. Tuning was performed using `irace` [129], with parameter domains and selected values listed in Table 3.1. The graph size was chosen to reflect the average network size in the final experiments (Section 3.4.2), as BRKGA’s population size depends on graph size. FC parameters remained as in [151].

Note that both FC training and BRKGA tuning used random graphs to promote generality.

3.4.2 Experimental Evaluation

In this subsection, we evaluate the three approaches—FC, BRKGA, and BRKGA+FC—on 19 real-world social networks sourced from the SNAP library [111]. Each network is a directed, unweighted graph. The sizes of these graphs are summarized in Table 3.2 (columns $|V|$ and $|A|$).

We tested three values of $k \in \{32, 64, 128\}$ and three values of the multi-hop parameter $d \in \{1, 2, 3\}$. The choice of $k = 64$ follows the evaluation in FC [151], while 32 and 128 were added for broader analysis. Values of d beyond 3 were omitted, as no significant differences were observed past $d = 3$.

Since FC is deterministic post-training, it was applied once per network for each (k, d) pair. In contrast, BRKGA and BRKGA+FC were run 30 times per network and parameter combination, each with a 900-second CPU time limit. Experiments ran on Intel(R) Xeon(R) Silver 4210 CPUs at 2.20 GHz. FC was implemented in Python 3, while BRKGA and BRKGA+FC were coded in C++.³

Table 3.2 compares FC with the average results of BRKGA and BRKGA+FC over 30 runs. Key observations are:

- Both BRKGA variants generally outperform FC, which itself surpasses existing heuristics [151]. Exceptions occur in 1 case for $k = 64$ and 9 cases for $k = 128$, indicating FC’s relative performance improves as k increases.

² Available at <https://github.com/camilochs/genetic-algorithm-with-gnn>

³ The difference in programming language arises because the authors of FC implemented it in Python 3. To mitigate the performance limitations of Python as an interpreted language, FC is implemented with PyTorch, an efficient deep learning library that leverages C++ and CUDA bindings to accelerate computations.

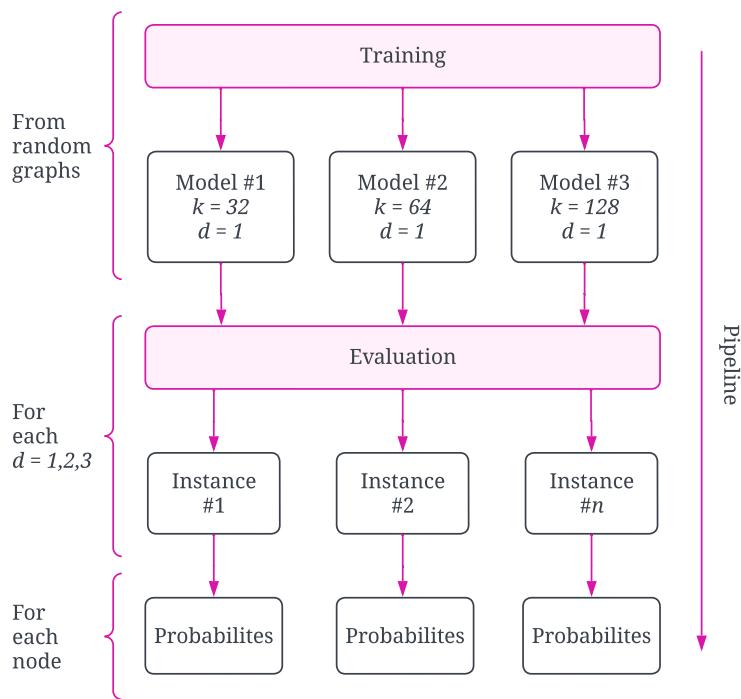


Figure 3.3: Data preparation and pipeline. The pipeline starts by training three FC models, one for each $k \in \{32, 64, 128\}$. Random graphs (Erdős–Rényi) were used for this purpose. Next, the evaluation of the FC models is performed for each of the 19 instances (social networks), for each value of parameter $d \in \{1, 2, 3\}$. Finally, the obtained probabilities (FC output) are exported and stored in text files.

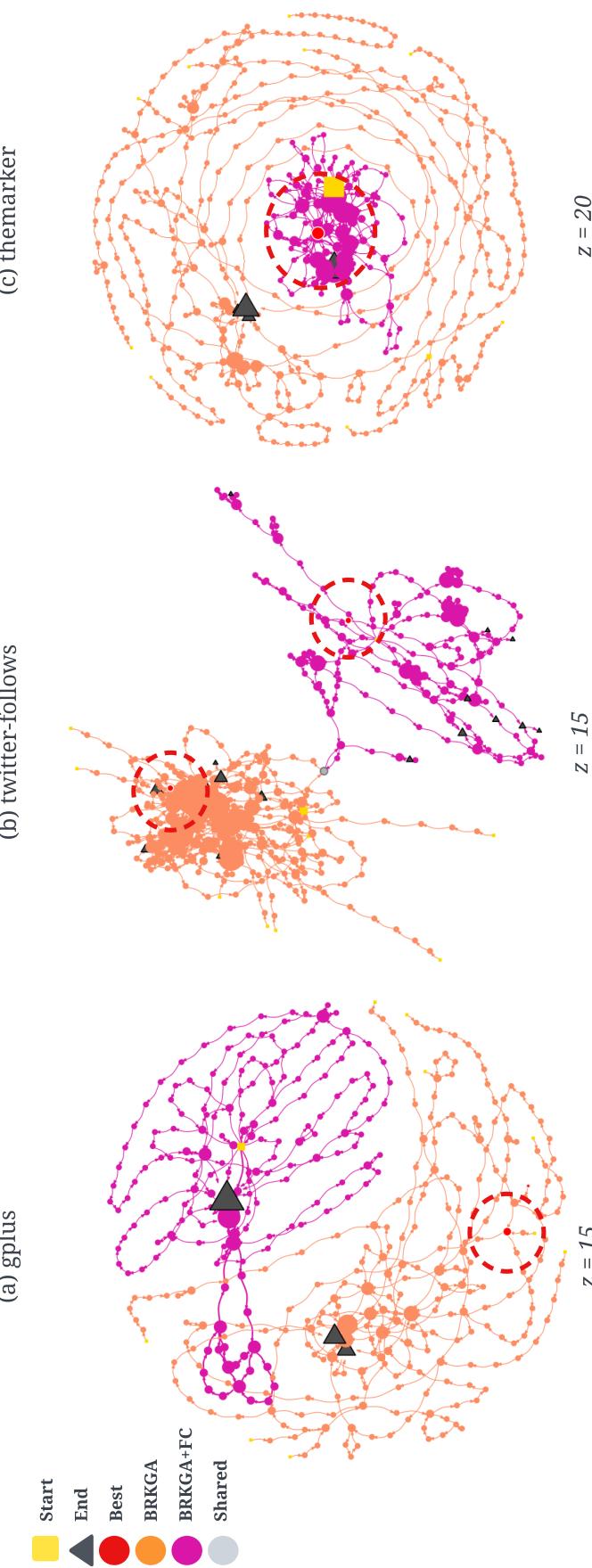


Figure 3: Search trajectory analysis of BRKGA and BRKGA+FC. The three plots display 10 execution trajectories of BRKGA (orange) and BRKGA+FC (pink) on three instances: gplus, twitter-follows, and themarker. The parameter z controls the granularity of search space partitioning used to generate these visualizations (see [155]). Yellow squares mark trajectory start points, gray triangles denote endpoints, light gray circles indicate regions visited by both algorithms, and red circles highlight the best solutions found. (a) BRKGA outperforms BRKGA+FC on gplus. (b) Both algorithms achieve comparable results on twitter-follows. (c) BRKGA+FC outperforms BRKGA on themarker. All visualizations use a force-directed layout based on physical analogies, without assuming any prior network structure.

- While BRKGA typically outperforms FC, the hybrid BRKGA+FC benefits from incorporating FC’s probability information when decoding individuals. This advantage is most pronounced at $d = 1$, where BRKGA+FC outperforms BRKGA in 73% of cases.
- The weakest BRKGA+FC performance is observed at $k = 32$ and $d = 3$ (47% superiority), possibly because (1) FC struggles to detect patterns at small k , and (2) all FC models were trained at $d = 1$, suggesting potential gains from training FC specifically for each d .

In summary, incorporating information from the GNN framework FC into our BRKGA significantly enhances the algorithm’s performance.

3.4.3 Analysis

In some cases, our hybrid algorithm does not outperform BRKGA or shows similar results. To investigate these situations, we employed Search Trajectory Networks (STNs) [155], which visualize algorithm trajectories in the search space and enable comparison of metaheuristic behaviors. We selected three networks illustrating different cases, shown in Figure 3.4. The following observations arise:

1. Figure 3.4 (a) depicts a case where BRKGA+FC underperforms relative to BRKGA. The trajectories focus on distinct regions, with BRKGA attracted to a specific area. However, the best solution (red dot) lies away from this attraction zone (larger grey triangles), suggesting that FC’s node probabilities may be misleading here.
2. Figure 3.4 (b) shows comparable performance between the algorithms, with mostly separate search areas and minimal trajectory overlap (light gray dot). Although the best solutions have equal quality, they differ substantially (two red dots).

In most cases, however, BRKGA+FC outperforms BRKGA, as illustrated in Figure 3.4 (c). Here, BRKGA+FC’s trajectory is more concentrated and less dispersed than BRKGA’s, with the best solution found within the region it explores—indicating that FC’s guidance is effective.

3.5 Conclusion

In this work, we developed a hybrid algorithm that integrates a Biased Random-Key Genetic Algorithm with the Graph Neural Network framework `FASTCOVER`, targeting an NP-hard combinatorial optimization problem of influence maximization in social networks. Our approach leverages the probability estimates from `FASTCOVER` to guide the decoding of individuals into feasible solutions. Experimental results on 19 real-world social networks demonstrate that, in most cases, the hybrid algorithm outperforms both standalone components.

A promising direction for future work is to extend this hybridization approach to other combinatorial problems, especially by harnessing recent advances in graph representation learning.

Note

This potential is further explored in the following chapter, where we demonstrate that machine learning techniques can be successfully integrated not only with BRKGA, but also with metaheuristics based on different paradigms—such as Ant Colony Optimization—while addressing a distinct optimization.

4

Improving Ant Colony Optimization supported by Deep Learning

Foundational Work for This Chapter

This chapter is based on the following publication:

- **Title:** *Q-Learning Ant Colony Optimization supported by Deep Learning for Target Set Selection*
- **Published in:** Genetic and Evolutionary Computation Conference (GECCO) - Core A
- **Type:** Conference Paper
- **Year:** 2023
- **Metaheuristic used:** Ant Colony Optimization (ACO)
- **Main contribution:** Using Deep Learning to detect patterns that guide ACO towards better search spaces
- **Problem addressed:** Target Set Selection
- **Type of contribution:** Algorithmic & Methodological
- **Has it improved the state-of-the-art?** Yes
- **DOI:** <https://doi.org/10.1145/3583131.3590396>
- **Current number of citations in Google Scholar:** 3

This was the only publication during my PhD in which I was not the first author, as a Master's student led the implementation and experimentation. Notably, the paper was nominated for the Best Paper Award in the *Evolutionary Combinatorial Optimization and Metaheuristics* (ECOM) track at GECCO 2023, held in Lisboa, Portugal.

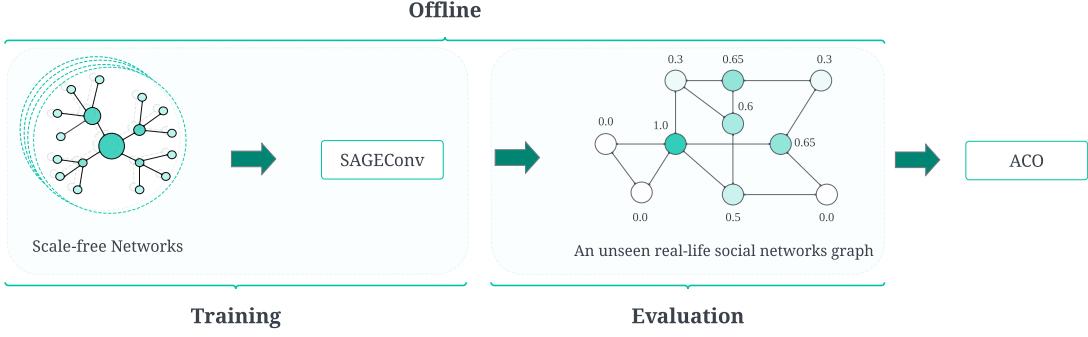


Figure 4.1: General framework of the proposed approach.

4.1 Introduction

In this work, we propose a more ambitious strategy that goes beyond previous approaches, ambitious in the sense that it enables the integration of metaheuristics with novel deep learning techniques. Specifically, we employ a Graph Neural Network of the SAGE type to generate node-level probability scores, which guide the Ant Colony Optimization algorithm toward the most promising regions of the search space. This offline mechanism parallels the approach discussed in the previous chapter. In addition, the method incorporates an adaptive component based on Q-learning, which dynamically refines the search strategy by rewarding configurations that produce better solutions—thus adjusting the initial static guidance provided by SAGE throughout the algorithm’s execution.

To tackle the Target Set Selection problem more effectively, we propose a hybrid approach that leverages both machine learning and combinatorial optimization. The core idea is to combine the structural awareness provided by Graph Neural Networks (GNNs) with the adaptive search capabilities of Ant Colony Optimization (ACO). By integrating learned node importance scores into the construction phase of ACO, we aim to guide the search towards more promising regions of the solution space. The full methodology is outlined in Figure 4.1. In the offline phase, a SAGE Graph Convolutional Network [81] is trained using scale-free networks to learn structural patterns and assign importance scores to nodes. These node-level probabilities are then used during the ACO’s solution construction process, guiding the probabilistic selection of nodes in each iteration. This hybrid design—blending offline learning and online adaptation—aims to enhance the overall performance of ACO on the Target Set Selection problem.

4.2 Problem Definition

We consider the Target Set Selection (TSS) problem following the notation of [36, 128]. Let $G = (V, E)$ be an undirected graph modeling, for instance, a social network. Each

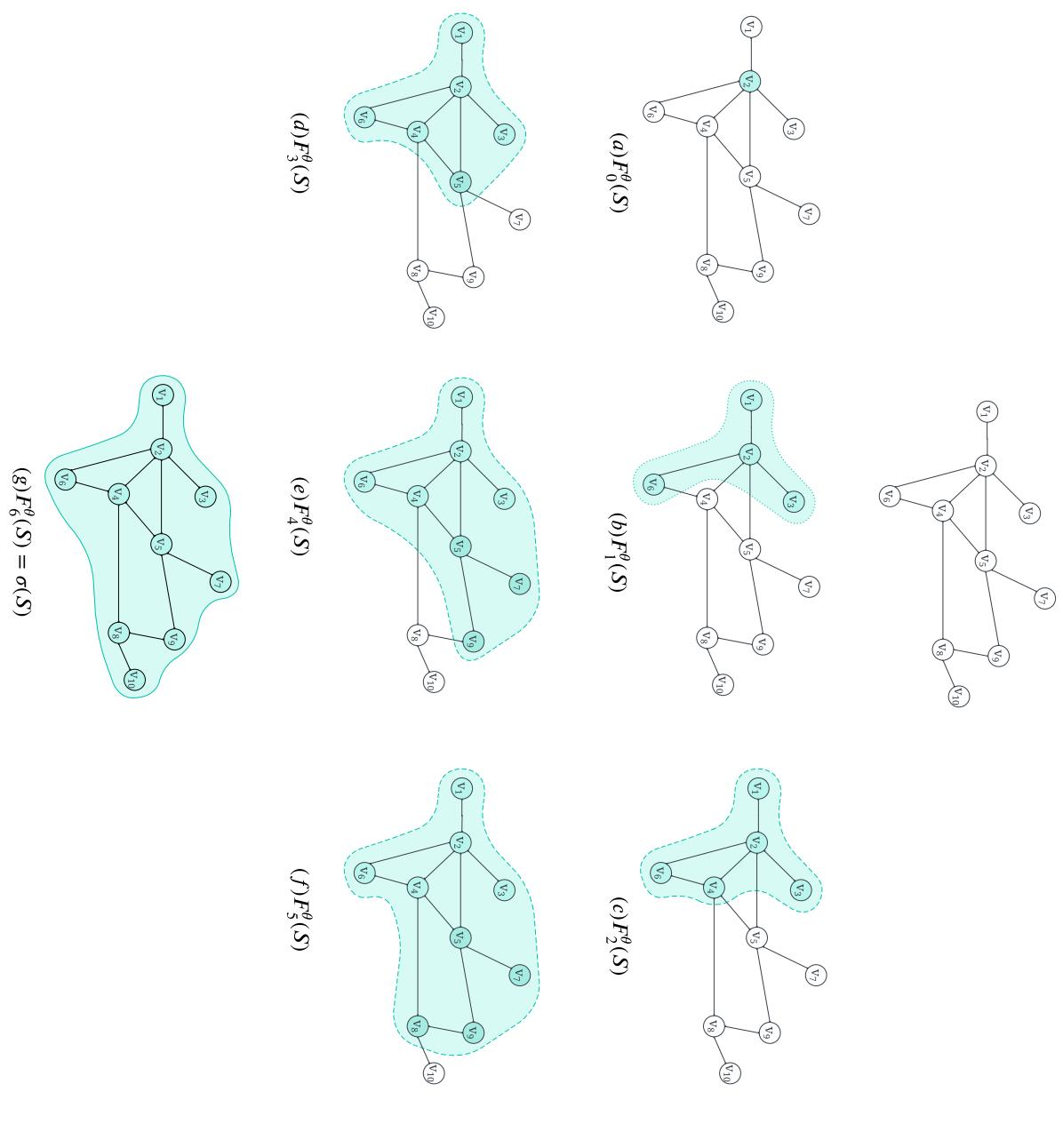


Figure 4.2: Diffusion process for threshold $\theta(v) = \left\lceil \frac{dg(v)}{2} \right\rceil$ for all $v \in V$. The initial target set (single node) is shown in (a).

node $v \in V$ has an associated threshold $\theta(v) \in \mathbb{N}$ with $\theta(v) \leq \deg(v)$, where $\deg(v)$ denotes the degree of node v . Recall that $\deg(v) := |N(v)|$, where $N(v) = \{u \in V \mid (u, v) \in E\}$ is the neighborhood of v .

A subset $S \subseteq V$ is called a *target set*, whose nodes are initially influenced. To verify if S is a valid solution, we apply a diffusion process based on the Linear Threshold (LT) model [75]. This generates a sequence of influenced node sets:

$$S = F_0^\theta(S) \subseteq F_1^\theta(S) \subseteq F_2^\theta(S) \subseteq \dots \subseteq \sigma^\theta(S) \subseteq V, \quad (4.1)$$

where $F_t^\theta(S)$ is the set of nodes influenced at time t . A node v becomes active at step t if (1) it was inactive at $t - 1$, and (2) at least $\theta(v)$ of its neighbors were active at $t - 1$. The initial active set $F_0^\theta(S)$ is the target set S .

Formally, the diffusion update at step t is:

$$F_t^\theta(S) = F_{t-1}^\theta(S) \cup \left\{ v \in \bigcup_{u \in F_{t-1}^\theta(S)} N(u) \mid |N(v) \cap F_{t-1}^\theta(S)| \geq \theta(v) \right\}, \quad \forall t \in \mathbb{N}. \quad (4.2)$$

This process guarantees the nested sequence in Equation (4.1).

A target set S is *valid* if there exists some finite t_0 such that $F_{t_0}^\theta(S) = V$, i.e., all nodes are influenced within finite time. Figure 4.2 illustrates this process for $\theta(v) = \left\lceil \frac{\deg(v)}{2} \right\rceil$, with an initial target set of a single node v_2 . After seven diffusion steps, the entire graph is influenced.

Additionally, as proven in [188], the following property holds: for any target set $S \subseteq V$ and node $v \in V$,

$$\sigma(S \cup \{v\}) = \sigma(\sigma(S) \cup \{v\}).$$

This means applying diffusion to $S \cup \{v\}$ yields the same influenced set as applying it to $\sigma(S) \cup \{v\}$. Since $\sigma(S)$ is already fully diffused, the latter computation is faster. We exploit this property to efficiently construct valid solutions within our Ant Colony Optimization framework.

4.3 Methodology

The proposed methodology integrates ACO with an external source of problem knowledge—here, derived from deep learning—applied to the TSS problem using a *MAX-MIN* Ant System (MMAS) within the Hypercube Framework. Our implementation follows the MMAS algorithm detailed in [153] for the minimum dominating set problem. To avoid redundancy, we omit the basic MMAS description, referring readers to [153]. The key differences in our approach lie in the pheromone model and the solution construction process.

Algorithm 2 Solution Construction Procedure**Require:** Graph $G = (V, E)$

```

1:  $S \leftarrow \emptyset$ 
2:  $C \leftarrow \emptyset$ 
3: while  $C \neq V$  do
4:    $v \leftarrow \text{ChooseFrom}(V \setminus C)$ 
5:    $C \leftarrow \sigma(C \cup \{v\})$ 
6:    $S \leftarrow S \cup \{v\}$ 
7: end while
8: return  $S$ 

```

4.3.1 Solution Construction in MMAS

The pheromone model associates a pheromone value τ_v to each node $v \in V$, following the standard approach for combinatorial optimization problems where solutions are subsets of nodes. Motivated by the strong performance of the maximum degree heuristic (MDH) [31], we use the node degree as a greedy function within the construction process.

The solution construction procedure (Algorithm 2) initializes the solution set S and the influenced node set C as empty. At each iteration, the function $\text{ChooseFrom}(V \setminus C)$ selects one node v from the uninfluenced nodes as follows: a random number r is drawn uniformly from $[0, 1]$. If $r \leq d_{\text{rate}}$ (the determinism rate), the node v maximizing the product of pheromone and degree plus one is chosen deterministically:

$$v := \arg \max_{v' \in V \setminus C} \{\tau_{v'} \cdot (\deg(v') + 1)\}. \quad (4.3)$$

Otherwise, a candidate list $L \subseteq V \setminus C$ of size l_{size} is formed, containing the nodes with the highest values of $\tau_{v'} \cdot (\deg(v') + 1)$:

$$\tau_{v'} \cdot (\deg(v') + 1) \geq \tau_{v''} \cdot (\deg(v'') + 1) \quad \forall v' \in L, v'' \in (V \setminus C) \setminus L. \quad (4.4)$$

Node v is then selected from L via roulette wheel selection with probabilities

$$p(v') := \frac{\tau_{v'} \cdot (\deg(v') + 1)}{\sum_{v'' \in L} \tau_{v''} \cdot (\deg(v'') + 1)} \quad \forall v' \in L. \quad (4.5)$$

The parameters d_{rate} (determinism rate) and l_{size} (candidate list size) play crucial roles in balancing exploration and exploitation. Additional MMAS parameters tuned experimentally include the number of solution constructions per iteration n_a and the pheromone evaporation rate $\rho \in [0, 1]$.

The construction process terminates once all nodes in the graph are influenced.

4.3.2 Integrating Deep Learning via Q-Learning

This research pursues two main goals: (1) to obtain problem-specific knowledge that improves upon the current best heuristic (node degree), and (2) to integrate this additional knowledge with the pheromone and greedy components of MMAS. While the next section (Section 4.4) details how this knowledge is generated, here we explain how it is integrated into the algorithm.

Assume that, for each node $v \in V$, we have precomputed a value $0 \leq l_v \leq 1$ representing the estimated usefulness of including v in a solution. These values are later used during hybrid solution construction.

Since this additional knowledge may not be equally useful for all graphs, we introduce a Q-learning mechanism [217] to dynamically balance between two solution construction modes:

- **Standard mode:** Based on $\tau_v \cdot (\deg(v) + 1)$.
- **Hybrid mode:** Incorporates deep learning knowledge as $\tau_v \cdot (\deg(v) + 1) \cdot l_v$, where l_v is a value between 0 and 1 estimated by a deep learning model, representing the usefulness of including node v in the solution.

At the beginning of the algorithm, both probabilities p_{std} and p_{hyb} are initialized to 0.5. At each solution construction, one of the two modes is chosen based on these probabilities. Once all n_a solutions are built in an MMAS iteration, the success of each mode is evaluated using a rank-based reward scheme.

Let the n_a constructed solutions be sorted by non-increasing size into $\{S_1, \dots, S_{n_a}\}$. We assign a quality score z_i to each S_i as:

$$z_1 := 1, \quad z_i := \begin{cases} z_{i-1} & \text{if } |S_i| = |S_{i-1}| \\ i & \text{otherwise} \end{cases}$$

Let \mathcal{S}_{std} and \mathcal{S}_{hyb} be the sets of solutions constructed in standard and hybrid mode, respectively. The reward for standard mode is then:

$$r_{\text{std}} := \frac{\sum_{S_i \in \mathcal{S}_{\text{std}}} \frac{1}{z_i}}{\sum_{i=1}^{n_a} \frac{1}{z_i}}, \quad \text{and} \quad r_{\text{hyb}} := 1 - r_{\text{std}}$$

Using these rewards, we update the probabilities for the next iteration via the Q-Learning update rule:

$$p_{\text{std}} := \alpha \cdot p_{\text{std}} + (1 - \alpha) \cdot r_{\text{std}}, \quad p_{\text{hyb}} := 1 - p_{\text{std}}$$

Here, $\alpha \in [0, 1]$ is the learning rate. This mechanism gradually favors the more effective construction strategy depending on problem instance and learning performance.

4.4 Generating Deep Learning-Based Node Information

To generate informative node probabilities for the TSS problem, we adopt a deep learning approach based on Graph Neural Networks (GNNs) [186]. Traditional machine learning methods struggle to capture node interactions in non-Euclidean domains, especially in graphs with large variability in size and structure. GNNs are specifically designed for such data, preserving permutation invariance and leveraging node, edge, and graph-level features. Among their variants, Graph Convolutional Networks (GCNs) use convolution operators to aggregate neighbourhood information, transforming node features \vec{x}_v into latent representations \vec{x}'_v .

Several GCN architectures exist—such as GAT, GIN, and SAGE—each using different aggregation strategies. In this work, we employ GraphSAGE [81], implemented in PyTorch Geometric [65], to assign importance scores (probabilities) to nodes. These probabilities serve as a proxy for estimating each node’s likelihood of belonging to a minimal target set. The exploration of alternative architectures is left for future work.

GraphSAGE Aggregation. Let $G = (V, E)$ be an undirected, unweighted graph, and \vec{x}_v the k -dimensional feature vector of node $v \in V$. The embedding \vec{x}'_v is computed via:

$$\vec{x}'_v = \kappa \left(W_1 \vec{x}_v + W_2 \cdot \frac{1}{|N(v)|} \sum_{u \in N(v)} \vec{x}_u \right) \quad (4.6)$$

Here, W_1 and W_2 are learnable weight matrices for the node and its neighbourhood, respectively, and κ is a ReLU activation. Adding multiple layers expands the receptive field of each node, aggregating information from more distant neighbours. However, stacking too many layers leads to the well-known over-smoothing problem [41, 26], where node embeddings become indistinguishable. To mitigate this, we use a single GraphSAGE layer.

The following sections describe the selected node features, the training data, and the learning process in more detail.

4.4.1 Selected Features and Training Instances

Feature selection plays a critical role in the performance of the SAGE model. To capture different structural aspects of each node, we extracted five graph-based metrics using the NetworkX library in Python [80]:

- **Betweenness centrality:** Measures how often a node appears on shortest paths between other nodes. High values indicate strong control over information flow.
- **Closeness centrality:** Reflects how close a node is to all others in terms of shortest paths. Central nodes have smaller average path lengths.
- **Eigenvector centrality:** Captures influence through transitivity. Nodes linked to other high-scoring nodes receive higher scores.

- **PageRank:** Assesses node importance based on the quantity and quality of incoming links, favoring connections from authoritative or selective nodes.
- **Degree:** Counts the number of neighbors of a node, capturing its direct connectivity.

Since real-world social networks often follow a scale-free structure, we trained SAGE using synthetic scale-free networks. We generated 20 such networks with 1000 nodes each, varying both density ($|E| = l|V|$ for $l \in \{5, 10, 20, 30\}$) and degree distribution exponent $\lambda \in \{2.0, 2.25, 2.5, 2.75, 3.0\}$, which controls the prevalence of highly connected hubs.

To evaluate each feature's standalone predictive power for TSS solutions, we applied a deterministic solution construction heuristic (see Section 4.3.1) to each of the 20 networks. In this setting, each node v was evaluated solely by its feature value, replacing the standard metric $\tau_v \cdot (\deg(v) + 1)$. Table 4.1 reports the resulting solution sizes. As expected, the degree is a strong indicator, but PageRank occasionally performs even better—e.g., for the instance ($\lambda = 2.5, l = 5$).

To further illustrate how feature values are distributed across a graph, Figure 4.3 shows the value distributions for the five metrics on one example graph ($\lambda = 2.25, l = 10$). Each subplot presents the feature values on the x-axis and the frequency of nodes with that value on the y-axis. Before being fed into SAGE, all values were normalized to the range $[0, 1]$.

4.4.2 SAGE Training

Obtaining the optimal target set sizes for the 20 training instances is computationally infeasible. Therefore, instead of using gradient-based optimization to train SAGE, we opted for a genetic algorithm (GA), a well-known alternative for training deep neural networks [57, 54]. The GA runs for 100 generations with a population of 50 individuals, using one elite individual, uniform crossover (rate 0.5), and mutation (rate 0.4). Each individual I encodes the 22 real-valued weights of the matrices W_1 and W_2 in SAGE.

To evaluate an individual I , the weights encoded in I are used in SAGE to generate node probabilities for each of the 20 training graphs, denoted by \mathcal{G} . For a graph $G = (V, E) \in \mathcal{G}$, the corresponding set of node probabilities is denoted $L(G, I)$, where $l_v \in [0, 1]$ for each $v \in V$. Then, using the construction heuristic from Section 4.3.1, a target set $S_{L(G, I)}$ is constructed deterministically by evaluating each node v with $l_v \cdot (\deg(v) + 1)$ in place of the standard pheromone expression.

The fitness of an individual I is computed as:

$$f(I) := \frac{1}{|\mathcal{G}|} \sum_{G \in \mathcal{G}} \frac{\alpha \cdot |S_{L(G, I)}| + (1 - \alpha) \cdot C(L(G, I))}{|V|} \quad (4.7)$$

where

Table 4.1: Target Set sizes obtained by the five features for the 20 scale-free networks (training set).

Graph		Centrality Measures				
λ	l	Betweenness	Closeness	Eigenvector	PageRank	Degree
2.25	10	97	102	100	97	96
	20	129	129	129	128	128
	30	145	144	144	144	143
	5	85	85	86	84	81
2.5	10	126	135	135	126	127
	20	160	161	158	159	156
	30	176	174	176	173	176
	5	107	115	114	97	100
2.75	10	143	155	159	143	141
	20	183	183	182	180	181
	30	197	198	201	200	198
	5	107	114	116	104	105
2.0	10	16	16	16	16	16
	20	24	24	24	24	24
	30	34	34	34	34	34
	5	46	46	46	46	46
3.0	10	101	105	103	97	97
	20	127	129	132	127	127
	30	156	149	153	149	149
	5	73	81	80	71	71

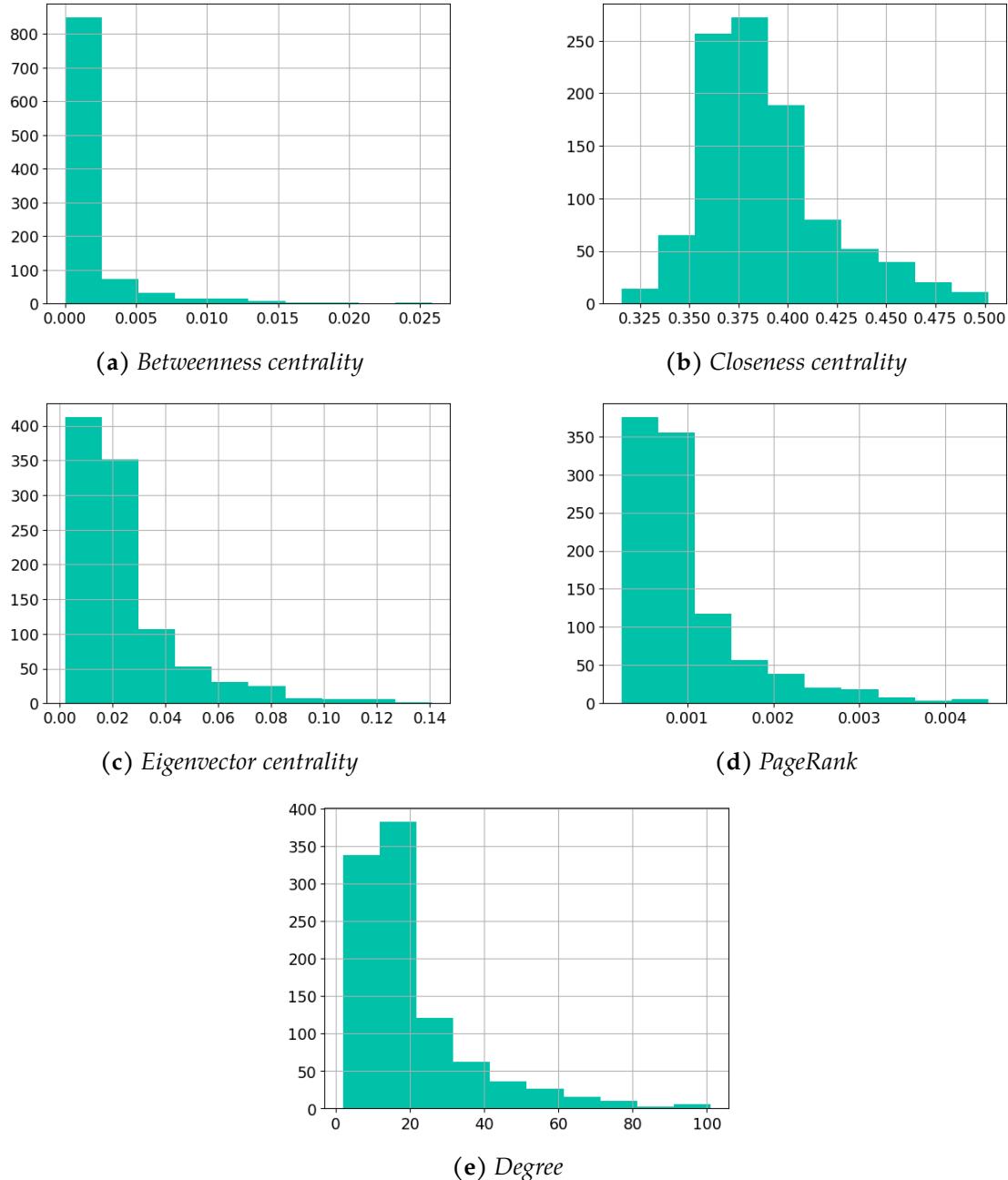


Figure 4.3: Feature value distributions for all nodes in scale-free network ($\lambda = 2.25, l = 10$). The x-axis indicates the feature values, and the y-axis shows the number of nodes with each value.

$$C(L(G, I)) = \sum_{v \in V} \begin{cases} (1 - l_v) \cdot w_1 & \text{if } v \in S_{L(G, I)} \\ l_v \cdot w_0 & \text{if } v \notin S_{L(G, I)} \end{cases} \quad (4.8)$$

Here, w_1 and w_0 are inverse-frequency weights, used to counterbalance class imbalance between nodes in and out of the target set. The parameter α (set to 0.7 in our experiments) controls the trade-off between minimizing the target set size and enforcing confidence in the node probabilities. The function $C()$ penalizes probabilities

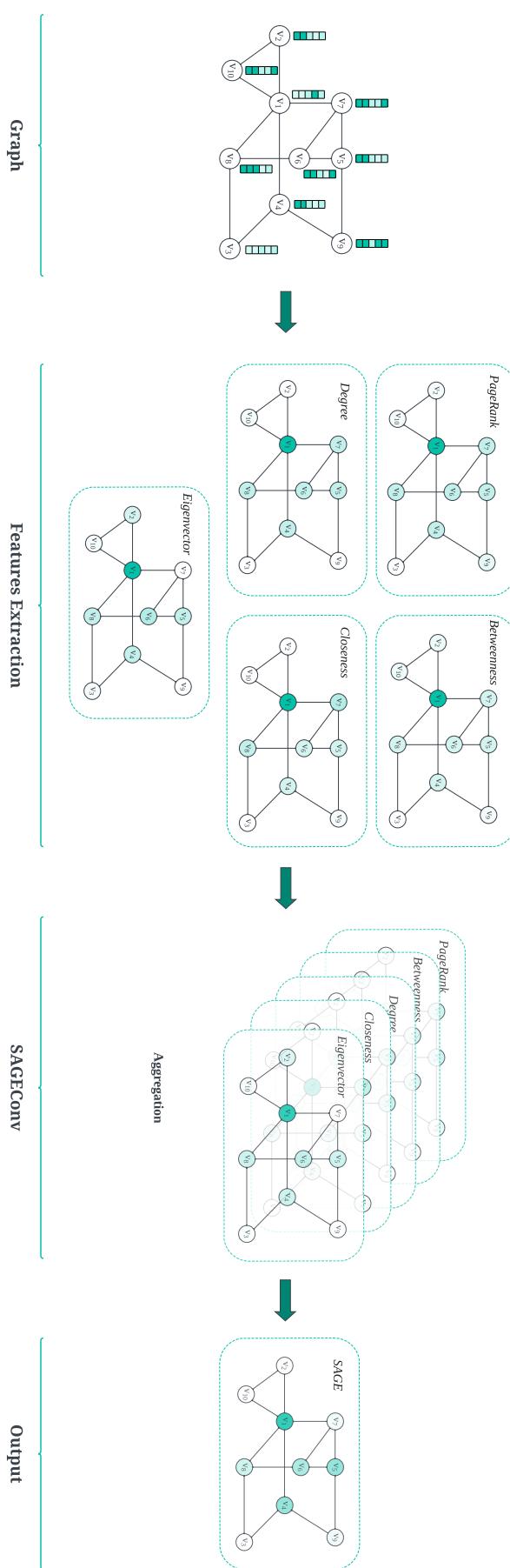


Figure 4.4: Process of extracting the node probabilities for an unseen graph. First, the feature values (for the five selected features) are calculated for each node of the graph. Then, each node is given as input to the SAGE network, which then provides a node probability as output.

that deviate from ideal binary values (0 or 1), promoting sharper distinction between selected and non-selected nodes.

To monitor potential overfitting, we also evaluated the elite individual's performance on a separate set of 18 Erdős–Rényi graphs, generated with sizes $|V| \in \{1000, 2000, 5000\}$ and edge densities of $10/|V|$, $15/|V|$, and $20/|V|$. At each GA generation, we computed the objective function for both the training set (scale-free graphs) and this test set. Figure 4.5 shows both curves. As no overfitting was detected—i.e., the test curve did not show a sustained increase—there was no need for early stopping.

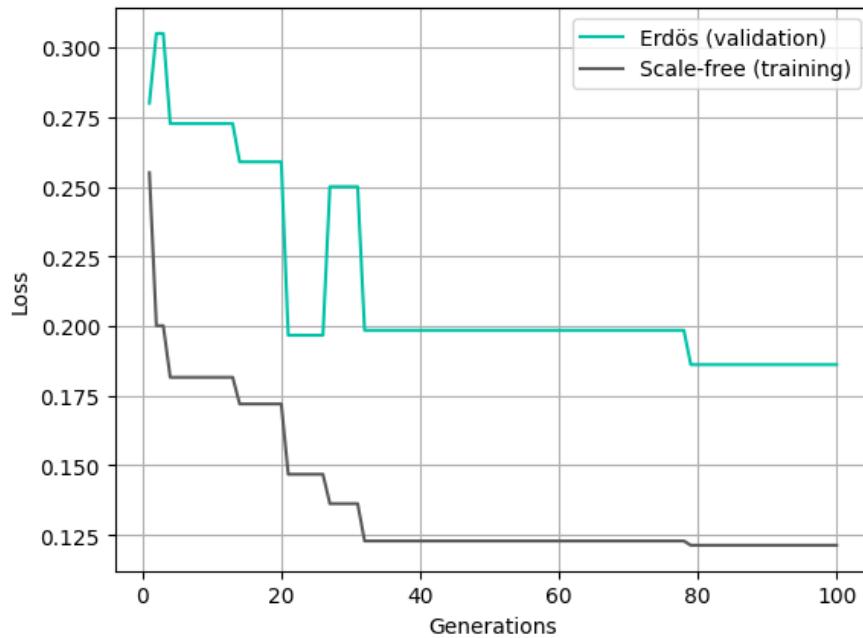


Figure 4.5: Evolution of the value of the GA elite individual during training. In addition, the objective function value on the basis of the Erdős graphs is used to detect overfitting.

4.5 Experimental Evaluation

This section presents the experimental evaluation of the standard and hybrid MMAS approaches, referred to as MMAS and MMAS-LEARN, respectively. We also assess the effectiveness of the node probabilities produced by SAGE by applying the construction heuristic from Section 4.3.1 deterministically: each node $v \in V$ is evaluated using $l_v \cdot (deg(v) + 1)$ instead of $\tau_v \cdot (deg(v) + 1)$, where l_v is the node probability output by SAGE. This heuristic will be denoted as SAGE. For comparative purposes, we also report results from the well-known maximum degree heuristic (MDS). The feature extraction and training of SAGE were implemented in Python, while the ACO variants were developed in C++. All experiments were conducted on a computing cluster with Intel® Xeon® 5670 CPUs (12 cores at 2.933 GHz) and at least 32 GB of RAM.

4.5.1 Experimental setting

We tested our methods on a set of 27 real-world social networks, some of which were sourced from the open-access SNAP repository [110], widely used in graph-based combinatorial optimization. The benchmark includes small-, medium-, and large-scale networks, enabling a study of algorithm performance across different scales.

Following prior work on a BRKGA for the TSS problem [188], we adopted the threshold function $\theta(v) = \lceil \deg(v)/2 \rceil$ for all $v \in V$. This choice ensures our results are directly comparable to those obtained by BRKGA, a current state-of-the-art approach. Moreover, fixed thresholds have also been used in other related studies [50, 66].

To ensure scalability, a dynamic time limit was imposed on MMAS and MMAS-LEARN runs: $\max \{100, |V|/100\}$ CPU seconds. This provides more computation time for larger graphs, while ensuring at least 100 seconds for each run.

4.5.2 Algorithm tuning

The parameter values for MMAS were optimized using the automated configuration tool `irace` [129], with a budget of 2000 runs and a precision of two decimal places. The tuning was conducted on three representative instances: CA-AstroPh, socfb-Mich67, and Amazon0302. The resulting parameters were: $n_a = 4$ (solutions per iteration), $d_{\text{rate}} = 0.0$, and $\rho = 0.22$. For fairness, MMAS-LEARN used the same configuration, with the exception of the Q-learning rate α , which was tuned separately and set to $\alpha = 0.32$.

4.5.3 Numerical results

Table 4.2 reports the numerical results. The first three columns list the network name, number of nodes, and number of edges. The largest networks contain hundreds of thousands of nodes and millions of edges. For Mds and SAGE, we report the resulting target set size (column “Result”) and the runtime (column “Time”). Note that SAGE’s training time is excluded. For BRKGA [188], MMAS, and MMAS-LEARN, we show the best and average results across 10 runs, along with the average time required to obtain the best solution in each run. The final row summarizes the average best results for each technique.

Several insights emerge from the results. First, SAGE outperforms Mds in 13 out of 27 cases, ties in 4, and is outperformed in 10. Notably, SAGE shows a clear advantage in the six largest instances. On average, SAGE yields solutions of size 8347.33, compared to 8535.52 for Mds.

Second, both MMAS and MMAS-LEARN outperform BRKGA—which was run under identical conditions as in its original study—on the majority of instances. With the exception of a few medium-sized graphs (such as the socfb-* family and networks like `musae_git`, `gemsec_facebook_artist`, and `deezer_HR`), the ACO-based approaches consistently perform better. These exceptions might be related to specific structural properties of the graphs, which could be examined in future work.

Table 4.2: Numerical results for 27 social networks

Networks	V	E	MDS		SAGE		BRKGA		MMAS		MMAS-LEARN				
			Result	Time	Result	Time	Best	Average	Avg. Time	Best	Average	Avg. Time	Best		
Karate	62	159	8	< 0.01	8	< 0.01	6	6.0	< 0.01	6	6.0	< 0.01	6	6.0	< 0.01
Football1	115	613	31	< 0.01	32	< 0.01	22	22.3	29.6	23	23.0	22.0	22	23.0	30.3
Dolphins	34	78	3	< 0.01	3	< 0.01	3	3.0	< 0.01	3	3.0	< 0.01	3	3.0	< 0.01
Jazz	198	2742	31	< 0.01	33	< 0.01	20	20.4	10.0	20	20.0	4.7	20	20.0	12.9
CA-AstroPh	18772	198050	1638	0.05	1701	0.03	1500	1508.6	182.7	1405	1412.5	173.1	1405	1413.0	172.6
CA-GrQc	5242	14484	1033	0.01	997	0.01	942	947.4	90.0	898	900.1	64.2	897	899.4	72.3
CA-HepPh	12008	118489	1529	0.03	1482	0.02	1394	1402.6	115.2	1289	1297.2	109.2	1289	1298.4	110.5
CA-HepTh	9877	25973	1388	0.01	1335	0.01	1307	1312.	76.4	1179	1186.2	84.6	1182	1189.2	82.5
CA-CondMat	23133	93439	2938	0.05	2753	0.05	2760	2777.6	216.4	2416	2422.3	220.4	2419	2428.0	222.8
Email-Enron	36692	183831	2881	0.1	2829	0.1	2745	2759.9	355.6	2679	2686.0	287.4	2692	2699.4	299.6
ncstrlwg2	6396	15872	1108	0.01	1100	0.01	1027	1045.7	64.3	1000	1003.5	80.4	997	1001.2	82.6
actors-data	10042	145682	1014	0.02	1060	0.03	937	943.4	95.6	900	907.8	89.3	895	907.0	92.1
ego-facebook	4039	88234	528	0.01	547	0.01	493	501.1	65.2	478	482.8	65.7	478	481.9	68.1
socfb-Brandeis99	3898	137567	395	0.02	394	0.01	338	358.0	80.8	365	369.2	68.1	368	370.2	59.3
socfb-nips-ego	2888	2981	10	< 0.01	10	< 0.01	10	10.0	< 0.01	10	10.0	< 0.01	10	10.0	< 0.01
socfb-Mich67	3748	81903	202	0.01	300	0.01	168	172.2	73.6	177	179.3	68.3	177	179.3	67.5
soc-gplus	23628	39194	69	0.01	69	0.01	62	62.9	56.5	61	61.5	28.8	61	61.9	10.2
musae_grit	37700	289003	196	0.08	230	0.07	176	182.7	308.0	202	205.9	202.9	199	206.4	172.7
loc_gowalla_edges	196591	950327	5670	0.7	5833	0.69	5546	5580.5	1835.3	5180	5195.1	728.0	5155	5177.6	1113.5
gemsec_facebook_artist	50515	819090	790	0.2	930	0.17	702	716.0	475.9	726	744.7	383.6	735	748.4	330.8
deezer_HR	54573	498202	2346	0.1	3044	0.12	2223	2252.9	468.9	2231	2247.0	335.9	2240	2255.9	283.9
com_youtube	1134890	2987242	39090	12.45	38774	12.23	39023	39037.3	10897.5	37399	37418.0	961.1	37154	37177.3	923.6
com dblp	317080	1049866	37197	4.1	35009	4.04	37017	37056.2	2970.3	32981	33016.9	1749.1	32364	32397.7	1726.5
Amazon0302	262111	899792	35766	2.4	34940	2.39	35685	35717.1	2148.5	30291	3034.7	1768.8	29553	29618.0	1705.4
Amazon0312	400727	2349869	31165	3.5	30298	3.47	31085	31096.8	3263.4	26280	26317.8	1747.4	26186	26201.2	1733.1
Amazon0505	410236	2439437	31926	3.7	30962	3.65	31842	31857.7	3702.5	26945	27000.9	1746.3	26801	26871.2	1695.6
Amazon0601	403394	2443408	31507	3.5	30705	3.47	31455	31475.2	3558.5	26665	26708.3	1726.5	26511	26568.0	1744.0
			8535.52		8347.33		8462.52		7474.41		7400.70				
AVERAGE															

Finally, while MMAS and MMAS-LEARN perform similarly on small- and medium-scale graphs, the hybrid MMAS-LEARN shows a clear advantage on the largest instances. It achieves an average solution size of 7400.70, compared to 7474.41 for MMAS, highlighting the benefit of integrating learning into the search process.

4.6 Conclusion

In conclusion, both objectives of this research have been successfully achieved. First, the fact that SAGE outperforms Mds—particularly on large-scale graphs—demonstrates our ability to extract and encode generic problem knowledge more effectively than the best-known greedy heuristic. Second, we have shown that incorporating this learned knowledge into an ant colony optimization algorithm leads to a measurable improvement in performance. As future work, we intend to validate the generality of these findings by applying the approach to other classes of challenging optimization problems.

Note

This will be the last project involving Graph Neural Networks (GNNs) in the context of my PhD. The main reason for this decision is that I found alternative modern techniques that offer two key advantages: easier implementation and improved performance.

Nevertheless, GNNs remain a valid option for integration within metaheuristic frameworks—especially in scenarios where modern alternatives, such as Large Language Models, present limitations. These may include high monetary cost or contextual constraints that prevent them from efficiently processing graphs with millions of nodes. For further discussion, see Chapter 5.

5

Large Language Models as Assistants for Enhancing Metaheuristics

Foundational Work for This Chapter

This chapter is based on the following publication:

- **Title:** *Metaheuristics and Large Language Models Join Forces: Toward an Integrated Optimization Approach*
- **Published in:** IEEE Access
- **Type:** Journal Paper
- **Year:** 2025 (The preprint was published in May 2024)
- **Metaheuristic used:** Biased Random Key Genetic Algorithm (BRKGA)
- **Main contribution:** Use LLMs as pattern-matching systems to identify and generate novel heuristics, thereby enhancing the performance of BRKGA
- **Problem addressed:** Multi-Hop Influence Maximization in Social Networks
- **Type of contribution:** Algorithmic & Methodological
- **Has it improved the state-of-the-art?** Yes
- **DOI:** <https://doi.org/10.1109/ACCESS.2024.3524176>
- **Current number of citations in Google Scholar:** 9

5.1 Introduction

This work presents what I consider to be the most original and personally meaningful idea developed during my PhD. In early 2024, the integration of metaheuristics (MHs) with Large Language Models (LLMs) primarily focused on explicit code generation. The prevailing approach involved crafting prompts for LLMs to generate heuris-

tics [165, 194], which would then be refined through an evolutionary process to develop new metaheuristics. This initial direction did not appeal to me at the time, primarily because it aimed to generate algorithms from scratch rather than leveraging existing ones. However, as the context window of LLMs expanded, the potential for providing richer context—such as extensive existing code—opened up more exciting possibilities.

In particular, the following promising avenue for advancing existing approaches emerged: using LLMs not as code generators, but as pattern detection engines based on metrics extracted from an optimization problem instance. One day, I opened ChatGPT and fed it hundreds of rows of numerical data—and to my surprise, it was able to identify several patterns that could be useful for an algorithm designer. Of course, it turned out not to be as simple as I initially thought. The concept was refined and matured through the invaluable feedback of my co-authors. My PhD supervisor, Christian Blum, not only suggested asking the LLM to optimize a linear model, but also provided the ideas for useful experiments. My co-supervisor, Filippo Bistaffa, proposed additional experiments; and my PhD colleague, Guillem Rodríguez, contributed the final insight that significantly improved our results: prompting the LLM to suggest repair parameters (which we refer to as *beta values*). This chapter presents the approach we developed through that collaboration.

Whenever a new technology emerges, it is natural to ask whether it can enhance existing methods. In the field of combinatorial optimization, MHs have proven to be powerful approximate algorithms for solving complex, NP-hard problems [72]. While they are effective at quickly generating good-enough solutions, their performance often relies on domain-specific knowledge. To overcome this limitation, researchers have explored integrating MHs with other techniques, such as exact algorithms and Machine Learning (ML). The combination of MHs with exact methods has yielded promising results [23], but it typically demands a high level of technical expertise. Similarly, embedding ML techniques within MHs can yield valuable problem-specific insights [102], yet such approaches frequently require specialized knowledge or, in the case of Deep Learning (DL), large datasets and significant computational resources [12]. In this work, we explore a different path: investigating the potential of Large Language Models (LLMs) to develop a novel hybrid approach that leverages the strengths of both MHs and LLMs.

5.1.1 Our Contribution

We introduce a novel approach to enhance MH performance by leveraging LLM output. Our method distinguishes itself from existing techniques in two key ways:

- Instead of using LLMs to generate MHs (e.g., [165, 126, 133]), we employ them as pattern recognition tools for problem instance metrics. This allows seamless integration into existing MHs via an LLM-provided parameter.

- Unlike methods that use LLMs as direct optimizers for simple, natural language-described problems [235] (limited by LLMs' stochastic nature), our approach tackles complex optimization problems by using LLMs to identify and track pertinent information within the problem instance.

This dual-faceted approach significantly advances LLM-MH integration, offering a more robust and versatile framework for diverse optimization problems. We utilize LLMs not as an end-to-end oracle (following the classification proposed by Bengio et al.[16]) but as an *intermediate, hybrid step* for pattern detection within metric values (see Figure 5.1). We validate our proposed MH+LLM integration on the *Multi-Hop Influence Maximization in Social Networks* problem, demonstrating improved performance over the current state-of-the-art MH-Deep Learning (DL) approach [32] (see Chapter 3). We believe this method unlocks new possibilities for enhancing MHs by leveraging generative AI for complex pattern recognition.

5.2 Background

5.2.1 LLMs as Pattern Recognition Engines

LLMs have revolutionized Natural Language Processing (NLP), offering unprecedented capabilities in understanding and generating human language. Models like GPT-4o (OpenAI)¹, Claude-3-Opus (Anthropic)², Gemini 1.5 (Google)³, Mixtral 8x22b (Mistral AI)⁴, and Command-R+ (Cohere)⁵, along with tools like ChatGPT, GitHub Copilot, and Bing Chat, have made advanced AI accessible beyond the expert community.

Built on the Transformer architecture [209], LLMs generate text token-by-token by modeling dependencies through self-attention. This mechanism enables the model to weigh and prioritize contextual information, producing coherent and contextually relevant outputs. Its parallelizable nature also aligns well with modern hardware, leading to efficient training and inference at scale. Despite their lack of built-in factual verification, LLMs represent a significant leap in NLP performance.

Beyond language, LLMs are increasingly applied to tasks requiring reasoning and abstraction, such as interpreting chemical structures or images [191], acting as autonomous agents with external tools [79], and solving problems in math and optimization [235]. Although challenges remain [2], these applications highlight the growing potential of LLMs in complex cognitive domains.

So far, LLM-based optimization has followed two main paths: (i) formulating optimization tasks via prompting and having the model propose solutions [235], and (ii) automating code generation to improve algorithmic design [133, 165]. However, the

¹ <https://openai.com/index/hello-gpt-4o>.

² <https://www.anthropic.com/news/claude-3-family>.

³ <https://deepmind.google/technologies/gemini>.

⁴ <https://mistral.ai/news/mixtral-8x22b>.

⁵ <https://docs.cohere.com/docs/command-r-plus>.

idea of using LLMs explicitly as pattern recognition engines in combinatorial optimization remains largely unexplored, despite evidence of their pattern-finding capabilities across tasks [147].

This work proposes a novel hybrid approach that integrates LLMs into metaheuristic search as pattern detectors, guiding the optimization process.

5.3 Problem Definition

This section introduces the combinatorial optimization problem used as a case study in our approach (described later in Section 5.4). We focus on a social network optimization problem—an ideal testbed for metaheuristics due to its graph-based structure and scalability challenges.

Specifically, we consider the Multi-Hop Influence Maximization problem, shown to be NP-hard [151, 13]. This problem has been tackled using various metaheuristics and, more recently, through a hybrid of Biased Random-Key Genetic Algorithm (BRKGA) and deep learning [32], which serves as a strong benchmark (see Chapter 3).

5.3.1 Multi-Hop Influence Maximization

Note

This is the same problem used in Chapter 3, in Section 3.2, so we now explain it only briefly.

Let the social network be modeled as a directed graph $G = (V, A)$, with V as nodes and A as directed arcs. The goal is to select a subset $U \subseteq V$ with at most k nodes, maximizing the number of nodes influenced within d hops. This problem is formally known as the k - d Dominating Set Problem (k - d DSP).

We define the influence of a node $u \in V$ as:

$$I_d(u) := \{v \in V : \text{dist}(u, v) \leq d\} \quad (5.1)$$

where $\text{dist}(u, v)$ is the length of the shortest directed path from u to v in G . For a set of nodes $U \subseteq V$, the total influence is:

$$I_d(U) := \bigcup_{u \in U} I_d(u) \quad (5.2)$$

The objective is to find a set $U^* \subseteq V$ such that $|U| \leq k$ and $|I_d(U)|$ is maximized:

$$\max_{U \subseteq V} |I_d(U)| \text{ s.t. } |U| \leq k \quad (5.3)$$

Figure 3.1 illustrates this influence process for different values of d in a small exam-

ple graph.

5.4 Integration of LLM Output into a Metaheuristic

Figure 5.1 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.

5.4.1 Prompt Engineering

A prompt is an input instruction given to a LLM, and its design plays a critical role in shaping the quality of the model’s output [219, 214] (also called prompt engineering). Moreover, the generated response may vary depending on the specific LLM used. Among the various prompting strategies, we adopted a one-shot learning approach—also referred to as few(1)-shot learning[25]. This involves providing the LLM with a single illustrative example to guide its response. Notably, Chen et al.[45] showed that for tabular data reasoning tasks, even a single example can yield strong results, removing the need for additional examples or model fine-tuning. This strategy enables the LLM to recognize patterns and follow instructions more effectively in the target task.

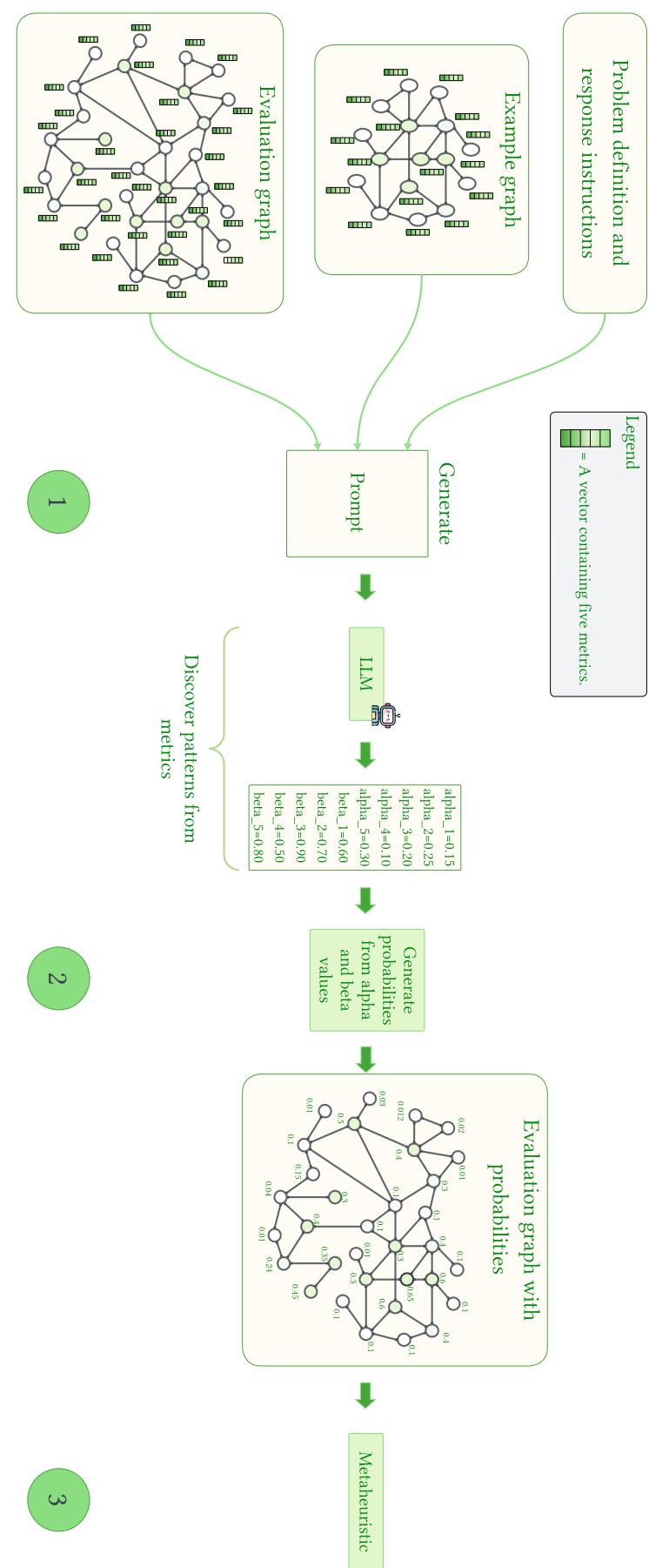


Figure 5.1: An overview of our approach to integrating MHs and LLMs: We employ LLMs to analyze problem instances and uncover hidden patterns. The patterns are then converted into useful information that guides the MH in its search for high-quality solutions.

Definition

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

$$P := \text{PROMPT}(\text{Tag1}, \text{Tag2}, \text{Tag3}, \text{Tag4}) \quad (5.4)$$

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 - d DSP. 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. In particular, these values are henceforth denoted by

$$m_{i,1}^{ex}, m_{i,2}^{ex}, m_{i,3}^{ex}, m_{i,4}^{ex}, m_{i,5}^{ex} \quad \text{for all } v_i \text{ of the example graph.} \quad (5.5)$$

Hereby, $m_{i,1}^{ex}$ is the value corresponding to metric in-degree, $m_{i,2}^{ex}$ corresponds to out-degree, etc. Note also that the metric values are normalized to the range $[0, 1]$. Furthermore, the high-quality k - d DSP solution of the example graph is computed using the pure BRKGA algorithm, which we adopted from our earlier work [32] (see Chapter 3). The solution is encoded as a vector of 32 nodes, separated by commas, corresponding to the k - d DSP parameter $k = 32$.

Next, the [EVALUATION GRAPH] tag contains the evaluation graph for which the k - d DSP 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

$$m_{i,1}^{eval}, m_{i,2}^{eval}, m_{i,3}^{eval}, m_{i,4}^{eval}, m_{i,5}^{eval} \quad \text{for all } v_i \text{ of the evaluation graph.} \quad (5.6)$$

Finally, the [RULES ANSWERING] tag specifies the details of the request to the LLM, which will be elaborated on in Section 5.4.1.

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:

$$\text{Output} := \text{EXECUTE}(P, \text{LLM}, \Theta) \quad (5.7)$$

Specifically, Θ contains values for exactly two parameters, 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. Since a more deterministic response is desirable when searching for patterns in data, we set the temperature to 0.⁶ The second 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.

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 5.2.

1. **Problem description.** The prompt starts by providing a concise definition of the k - d DSP utilizing LaTeX notation within the [PROBLEM] tag; see the top right of Figure 5.2.
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 Erdős–Rényi model [161] 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 5.5.
 - [ANSWER] tag: The solution to the example graph is computed using the BRKGA algorithm from [32] (see Chapter 3). 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.

⁶When generating text or paraphrasing, it is advisable to increase the temperature. This is because creativity is a welcome characteristic in these scenarios.

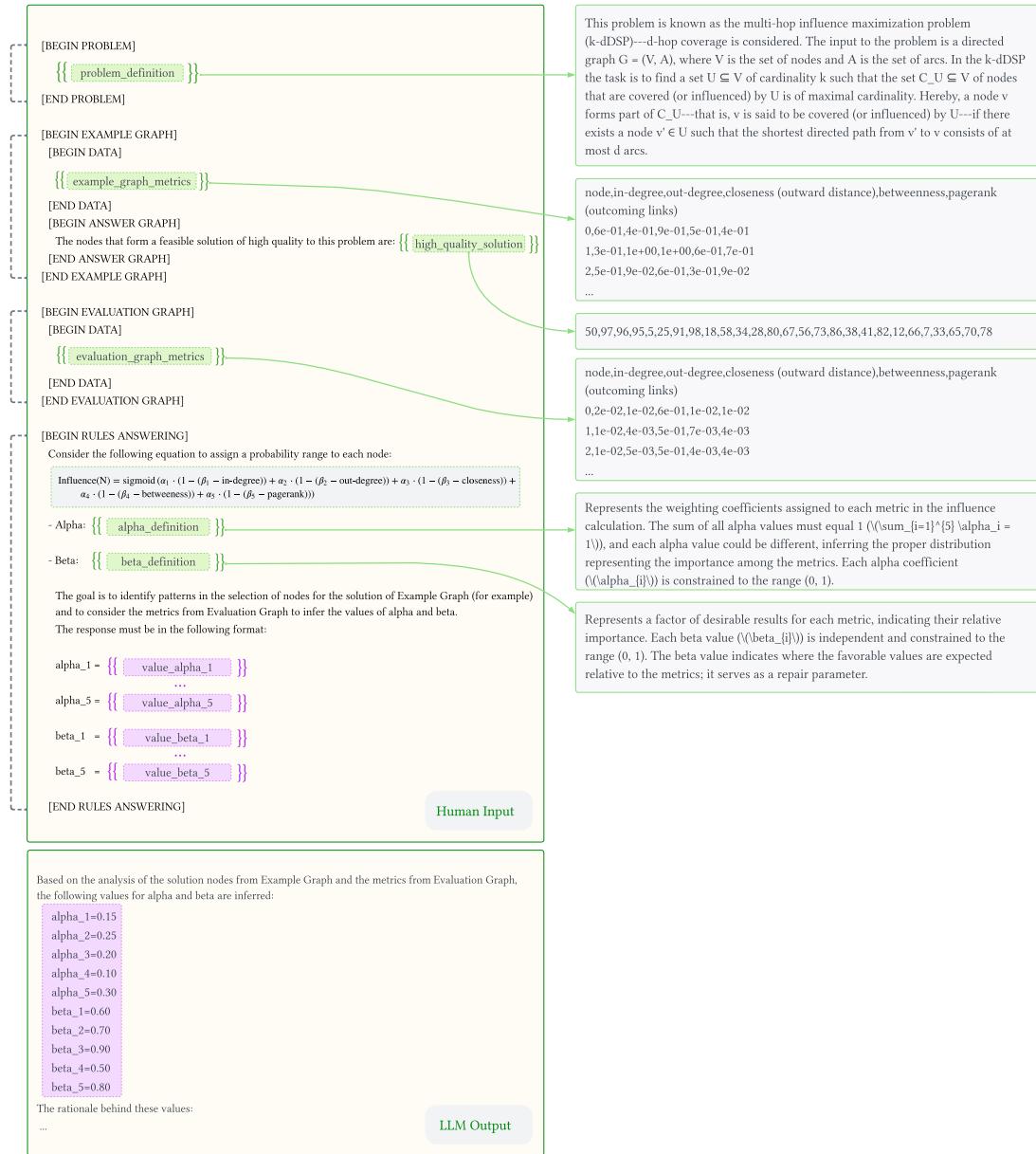


Figure 5.2: An example of a prompt and the corresponding LLM response. The prompt includes the problem definition, a graph example with node metrics and a high-quality solution, an evaluation graph, and instructions for the LLM for producing the output. Based on the patterns identified in the evaluation graph, the LLM provides the importance of each metric, represented by the set of alpha and beta values.

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 5.4.2. 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.

5.4.2 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. (5.7)) which is as follows:

$$Output = \{\alpha_1, \dots, \alpha_5, \beta_1, \dots, \beta_5\} \quad (5.8)$$

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 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:

$$p^{\text{LLM}}(v_j) := \sigma \left(\sum_{i=1}^5 \alpha_i \cdot (1 - (\beta_i - m_{j,i}^{\text{eval}})) \right) \quad (5.9)$$

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.⁷

As shown in Figure 5.2, our proposed prompt thoroughly explains the alpha and beta values to the LLM, along with Eq. 5.9. 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.

5.4.3 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.

The Considered Metaheuristic: A BRKGA

Since the BRKGA employed in this study is identical to the one presented in Chapter 3, Section 3.3, we refrain from redefining it here.

Hybrid Algorithm

The proposed hybrid algorithm—referred to as BRKGA+LLM—starts with two offline pre-processing steps. Given an evaluation graph $G = (V, A)$, a prompt is generated as described in the previous section and sent to an LLM. Based on the resulting α and β values extracted from the model’s output, the probability $p^{\text{LLM}}(v_j)$ for each node $v_j \in V$ is computed using Eq. (5.9).

Next, the original greedy function $\phi()$ defined in Eq. (3.4, see page 29) is replaced with a modified version that incorporates the LLM-derived probabilities:

$$\phi_{\text{INFLUENCE}_{\text{LLM}}}(v_j) := |N(v_j)| \cdot \pi(j) \cdot p^{\text{LLM}}(v_j), \quad \forall v_j \in V \quad (5.10)$$

⁷ 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 [232].

Table 5.1: 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	Version	License	Maximum	Test Environment (API)
				Arena	
				Context	
Model	Chatbot	Version	License	Maximum	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

We hypothesize that appropriate predictions from the LLM can bias the algorithm toward more promising regions of the search space. These regions are assumed to contain high-quality solutions that the BRKGA alone would be unlikely to identify. In this sense, leveraging LLM-discovered patterns in metric values (see Section 5.4.1), rather than relying solely on out-degree, may lead to improved performance.

5.5 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 [32] and described above in Section 5.4.3.
- 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. (5.10)) in [32].
- BRKGA+LLM: the BRKGA enhanced with LLM output as described in the previous section.

In this chapter, we adopt the corresponding parameter settings of BRKGA obtained by tuning in Chapter 3 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.

5.5.1 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. Re-

garding 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-8x22b-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,⁸ a community-driven ranking system for LLMs [246].⁹ Table 5.1 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.

Execution Environment

We utilized the OpenRouter API¹⁰ to execute prompts in their corresponding LLMs, except for Claude-3-Opus, which we used through the Anthropic API (see Table 5.1). 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.

Dataset

Our evaluation is based on two categories of k - d DSP 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 [150]. The corresponding graph generator requires four real-valued parameters, whose values are reflected by the instance names.¹¹ The second instance set comprises four real-world social network graph instances obtained from the well-established SNAP (Stanford Network Analysis Project) repository [110]. Moreover, note that the k - d DSP 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 $k \in \{32, 64, 128\}$.

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

1. The maximum context window of LLMs is still relatively small.¹² For instance,

⁸ <https://chat.lmsys.org>.

⁹ 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 this thesis.

¹⁰ <https://openrouter.ai>.

¹¹ 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.

¹² 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

Table 5.2: 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

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 [7], 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 5.2 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 5.6 for more information on this).

5.5.2 Analysis of LLM Output

This section evaluates the usefulness of LLM outputs as guidance within the BRKGA algorithm, aiming to show they are meaningful rather than arbitrary. To this end, we conducted three sets of experiments, each targeting a different aspect. Figure 5.3 presents the custom three-dimensional experimental framework designed for this evaluation.¹³

Before the main evaluation, we selected the most suitable LLM. We generated prompts for six synthetic graphs across all $d \in \{1, 2, 3\}$ and $k \in \{32, 64, 128\}$ combinations, feeding them to GPT-4o, Claude-3-Opus, Command-R+, and Mixtral-8x22b-Instruct-v0.1. Probabilities from these LLMs (and from the out-degree metric) were directly used to generate solutions. Table 5.3 shows Claude-3-Opus generally excels, especially for larger k . While out-degree performs slightly better for $k = 32$, Claude-3-Opus is superior for $k = 128$. Thus, we selected Claude-3-Opus for all subsequent experiments.

¹³each node to be equally important, the LLM needs to consider as much context as possible to deliver reasonable and trustworthy results.

¹³Future work may extend this framework to further assess LLM response quality and their integration with metaheuristics.

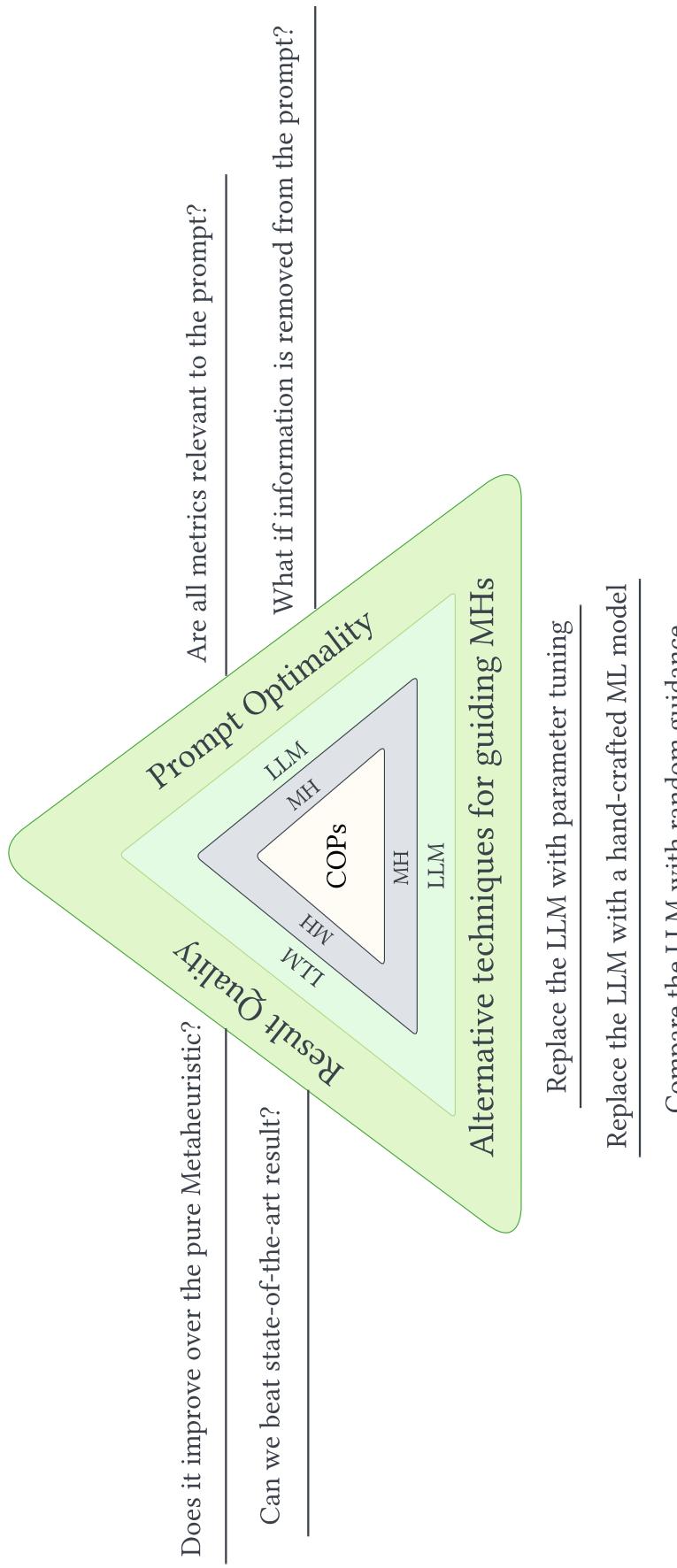


Figure 5.3: A comprehensive evaluation framework was used to assess the usefulness of integrating MHs with LLMs for solving combinatorial optimization problems (COPs) across the three dimensions shown in the graphic.

Table 5.3: 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-8x22b-Instruct-v0.1. The six synthetic graphs are chosen as a test bed.

Instance	V	E	d	k = 32				k = 64				k = 128						
				GPT-4o		Claude-3-Opus		Command-R+		Mixtral-8x22b		GPT-4o		Claude-3-Opus				
				out-degree		out-degree		out-degree		out-degree		out-degree		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

Dimension 1 of the Evaluation Framework: Result Quality

In the first experiment set, we compared BRKGA with BRKGA+LLM on six synthetic graphs (all $d \in \{1, 2, 3\}$ and $k \in \{32, 64, 128\}$ combinations). Both were executed 10 times, with a 900 CPU second limit. Table 5.4 shows BRKGA+LLM generally outperforms BRKGA, with significant improvements on the largest graph (0.2-0.0-0.3-0.5, 1000 nodes, 8000 arcs). Only three cases showed slightly inferior BRKGA+LLM results. While promising, these results are from relatively small instances.

Next, we applied BRKGA+FC [32] (a hybrid GNN-biased approach), BRKGA, and BRKGA+LLM to four larger real-world social networks (Table 5.5). All three had a 900 CPU second limit, averaged over 10 runs. BRKGA+LLM consistently outperformed both, with greater margins on larger networks, especially for increasing k . This is notable as prompts only included an example solution for $k = 32$, indicating the LLM’s ability to uncover meaningful patterns and adapt.

Finally, to assess the meaningfulness of LLM output, we created two BRKGA+LLM variants: `static` (random alpha/beta probabilities) and `dynamic` (random alpha/beta re-computed iteratively). All three ran 10 times on the four real-world networks for all $d \in \{1, 2, 3\}$ and $k \in \{32, 64, 128\}$ combinations. Table 5.6 clearly shows LLM guidance is significantly more useful than random guidance.

In conclusion, LLM output guidance consistently improves algorithm performance, demonstrating its value in informing metaheuristics. We anticipate even greater benefits for larger networks, though current limitations prevent such extensive testing.

Dimension 2 of the Evaluation Framework: Alternative Techniques for Guiding MHS

To assess LLM output reliability, we compared BRKGA+LLM with BRKGA+irace, an algorithm variant where alpha and beta values are obtained via explicit parameter tuning using `irace` [129]. For each of the four large social networks (Section 5.5.1), we performed a tuning procedure: for every $d \in \{1, 2, 3\}$ and $k \in \{32, 64, 128\}$ combination, a training instance was generated, and `irace` was applied with a 1000-run budget and a 900 CPU second time limit per run. After obtaining the optimal alpha and beta values, BRKGA+irace was run under the same conditions as BRKGA+LLM.

Table 5.7 shows that BRKGA+irace outperforms BRKGA+LLM in `soc-hamsterster` and `soc-wiki-elec`, while the opposite holds for `sign-bitcoinotc`. Performance is similar for `soc-advogato`, except for $d = 3$ where BRKGA+irace is better. Crucially, BRKGA+irace’s alpha and beta values required 250 hours of computing time per instance, whereas Claude-3-Opus’s output was obtained in under a minute without specific training.

We used Pearson’s correlation coefficient [15] ($\rho_{\text{irace}, \text{LLM}}$ in Table 5.8) to analyze the relationship between alpha and beta parameters from `irace` and the LLM.

1. Where BRKGA+irace dominates (`soc-hamsterster` and `soc-wiki-elec`), alpha values show a moderate to strong negative correlation. Beta values show no clear rela-

Table 5.4: 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.

Instance	N	E	d	k = 32		k = 64		k = 128	
				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
	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
	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
	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
	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

tionship for soc-hamsterster but a negative correlation for soc-wiki-elec. Predominant negative correlations suggest irace’s chosen direction is superior.

2. Conversely, in sign-bitcoinotc, where BRKGA+LLM excels, alpha values show no correlation. This indicates the LLM found unique, effective predictions missed by irace.
3. For soc-advogato, results are inconclusive. Alpha values show negative correlation, while beta values show positive correlation. This suggests either irace or the LLM correctly identified one set of values but not the other.

While BRKGA+LLM’s results are not always superior to irace in two of four instances, our approach offers significantly reduced computational effort, even accounting for prompt generation time. Future LLM improvements or prompt adjustments (e.g., addressing current precision limitations due to 0.05 divisibility) could further enhance its performance.

Table 5.5: Numerical comparison of three algorithms—*BRKGA*, *BRKGA+FC* (results extracted from [32]), 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 $k \in \{32, 64, 128\}$.

Instance	V	E	d	k = 32			k = 64			k = 128		
				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

Table 5.6: 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.

Instance	V	E	d	k = 32			k = 64			k = 128		
				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

Dimension 3 of the Evaluation Framework: Prompt Quality

This final dimension shifts from numerical analysis to an interpretive discussion on designing effective prompts for optimization problems. We first examine the five selected metrics (Section 5.4.1). As larger prompts increase context window limitations and financial costs, we investigate if less information can yield similar or superior results. This involves two experiments: assessing metric correlations and analyzing the impact of information removal.

Correlation between metrics. Figure 5.4 displays a matrix of plots for the soc-

Table 5.7: A numerical comparison of BRKGA+LLM and BRKGA+irace. In the latter algorithm, the alpha and beta values are determined by tuning through *irace*.

Instance	V	E	d	k = 32		k = 64		k = 128	
				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 5.8: The alpha and beta values as determined by *irace* and the LLM for each case. Pearson’s correlation coefficient ($\rho_{\text{irace}, \text{LLM}}$) 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
$\rho_{\text{irace}, \text{LLM}}$	-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
$\rho_{\text{irace}, \text{LLM}}$	0.08		-0.55		0.55		-0.67	

hamsterster instance, where BRKGA+LLM significantly outperformed BRKGA (Table 5.5). The upper triangle shows scatter plots of metric pairs (e.g., out-degree vs. in-degree). The lower triangle presents kernel density estimation (KDE) plots, revealing smoothed data distributions, clusters, outliers, and non-linear relationships. The diagonal shows univariate KDE plots for individual metric distributions. Observations from Figure 5.4:

- All metric pairs in the upper triangle exhibit non-linear patterns, suggesting each metric contributes unique information and none are superfluous.
- Lower triangle KDE plots highlight non-linear relationships less apparent in scatter plots. pagerank shows a more linear relationship with other metrics, but outliers prevent its exclusion.
- Univariate KDE plots on the diagonal reveal frequently occurring values for certain metrics (e.g., betweenness), suggesting potential prompt design strategies to reduce size.

All metrics appear relevant. However, adjusting the metric set might yield better results.¹⁴

Removal of information. Our investigation revealed the following effects of prompt modification:

- Removing graph metrics ([EXAMPLE GRAPH] tag) still allows the LLM to generate useful responses, leveraging its pre-training knowledge of social network metrics. However, providing an example graph refines alpha and beta values, improving output for the k - d DSP problem.
- Using scientific notation for metric values ([DATA] tag) maintains LLM response quality while reducing character count and tokens.
- Beta values are crucial for response quality; their omission significantly degrades LLM output. Assigning importance weights (alpha values) and requesting an expected value (beta) enables the LLM to uncover subtle patterns in evaluation graph metrics ([EVALUATION GRAPH] tag), leading to enhanced results.

In summary, while LLMs possess prior knowledge, it is insufficient for independent pattern identification in tabular numerical data.

Differences in node selection. Finally, we analyzed how LLM output influences node selection in BRKGA+LLM versus BRKGA. Figure 5.5 shows node probabilities (black line) relative to normalized metric values for the synthetic graph 0.2-0.0-0.3-0.5. The x-axis orders 500 nodes by decreasing LLM-probability. Horizontal lines mark nodes chosen by best BRKGA (dotted), best BRKGA+LLM (solid), and their intersection (dashed). Specific cases (a), (b), and (c) in Figure 5.5 highlight:

- (a) A BRKGA+LLM-selected node (second solid green line) has significantly higher closeness than out-degree (the standard BRKGA metric). This shows the LLM's ability to identify suitable nodes by blending multiple metrics.

¹⁴Financial limitations prevent extensive prompt experimentation, especially for large, challenging scenarios; thus, careful consideration is essential.

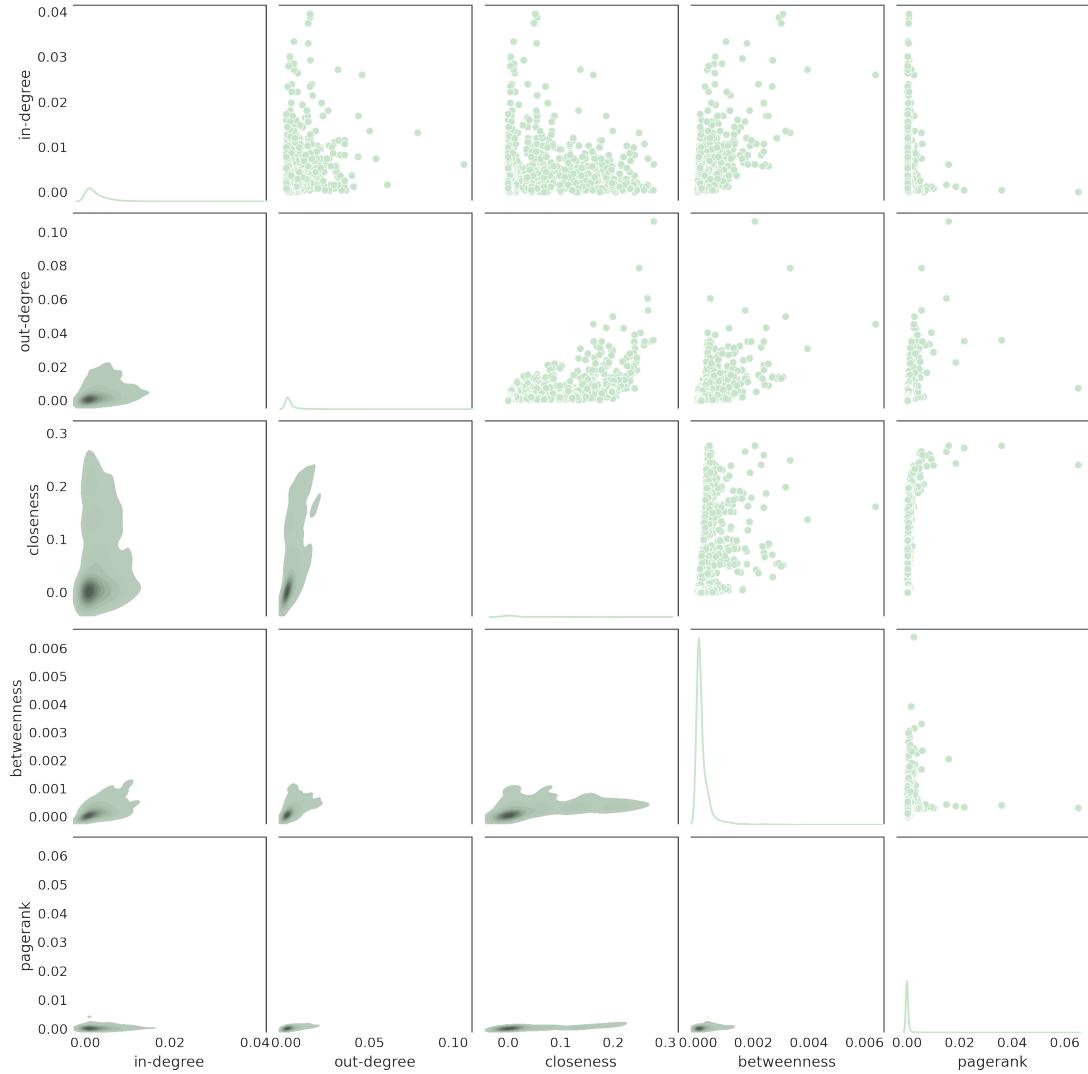


Figure 5.4: Correlations between all pairs of the five considered metrics concerning the soc-hamsterster network.

- (b) Similar to (a), the first and last BRKGA+LLM-selected nodes show higher closeness than out-degree. Conversely, nodes shared by both (BRKGA and BRKGA+LLM, green dashed lines) have high closeness and out-degree.
- (c) This example also shows cases where closeness is high and out-degree is relatively lower for BRKGA+LLM-selected nodes. This indicates that for the 0.2-0.0-0.3-0.5 network, the LLM's ability to recognize the importance of closeness for certain nodes, which pure BRKGA cannot detect, leads to the best solution.

5.5.3 Visual Comparative Analysis

Numerical analysis often fails to capture the full complexity of a metaheuristic's stochastic search process. Visual tools, like STNWeb [33] (see Chapters 9 and 10), address this by generating directed graphs from algorithm trajectories, offering deeper

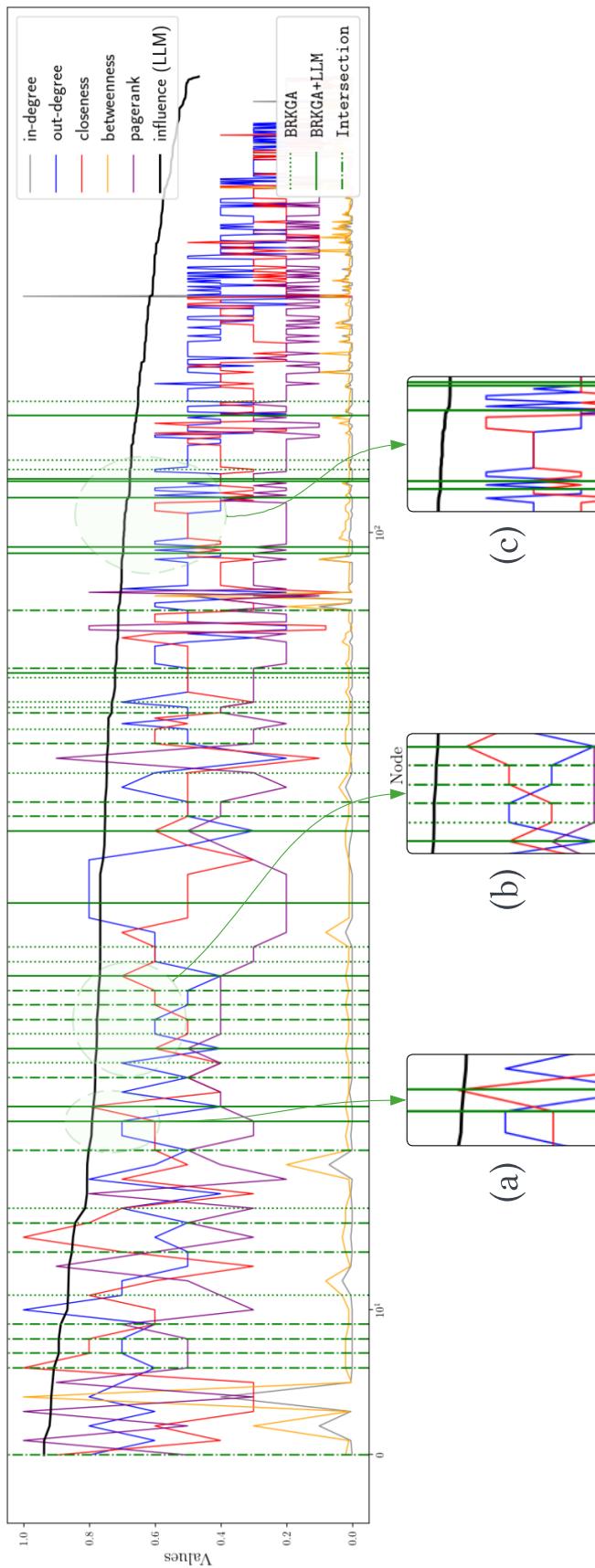


Figure 5.5: Analysis of the probabilities computed based on the alpha and beta values (black line) in relation to the (normalized) values of the five metrics. The x-axis ranges of all 500 nodes of the synthetic graph 0.2-0.0-0.3-0.5 ordered by a non-increasing LLM-probability. Moreover, the graphic marks the nodes chosen for the best BRKGA solution, the best BRKGA+LLM solution, and their intersection.

insights into search space navigation and enabling performance comparisons. This section presents a visual analysis to better understand BRKGA+LLM’s advantages over BRKGA and BRKGA+FC.

Figure 5.6 displays STNWeb plots for 10 runs of BRKGA+LLM, BRKGA, and BRKGA+FC on the soc-hamsterster instance. The following explains the plot’s technical elements and highlights key insights:

1. Each of the 30 algorithm runs is a directed trajectory in the search space, color-coded: BRKGA (cyan ●), BRKGA+FC (magenta ●), and BRKGA+LLM (green ●).
2. Trajectory starting points are yellow squares (■); end points are black triangles (►).
3. Trajectories comprise solutions (nodes, ●, ●, and ●) connected by directed edges, each with an increasing fitness value (for this maximization problem).
4. Node size indicates the number of trajectories passing through it.
5. Red nodes ● represent the best solutions found across all 30 trajectories. Multiple best solutions may exist.

Figure 5.6 reveals several interesting observations. Only BRKGA+LLM finds best solutions (two red dots), located in distinct search space areas. The three algorithms are attracted to different regions. While BRKGA and BRKGA+FC converge to nearby solutions, BRKGA+LLM shows no clear convergence to a single area. Notably, BRKGA’s search trajectories are much shorter than those of the two hybrid approaches.

Following empirical prompt analysis (Section 5.5.2) and this visual study, our proposed hybridization successfully demonstrates that LLMs can generate heuristic information to improve metaheuristic search. Our approach even outperformed an alternative hybridization using a hand-crafted, specifically trained graph neural network [32]. However, successful LLM integration into MHs still involves critical issues and open questions, detailed in the next section.

5.6 Discussion and Open Questions

The current enthusiasm surrounding LLMs continues to grow, with new applications emerging daily across a wide range of domains—from solving complex problems to automating everyday tasks. While their universal utility remains debatable, our work highlights their potential as pattern recognizers capable of uncovering latent structures and guiding metaheuristics. This opens several questions:

- **Are LLMs still just “stochastic parrots”?** Bender et al. [14] famously argued that LLMs lack true understanding, merely mimicking language without grounding in meaning. However, models have advanced considerably since then. Our study, among others [2, 147], shows that LLMs can reason within structured contexts such as optimization, suggesting their utility goes beyond imitation. That said, their application in sensitive areas—like law or ethics [6]—still demands careful

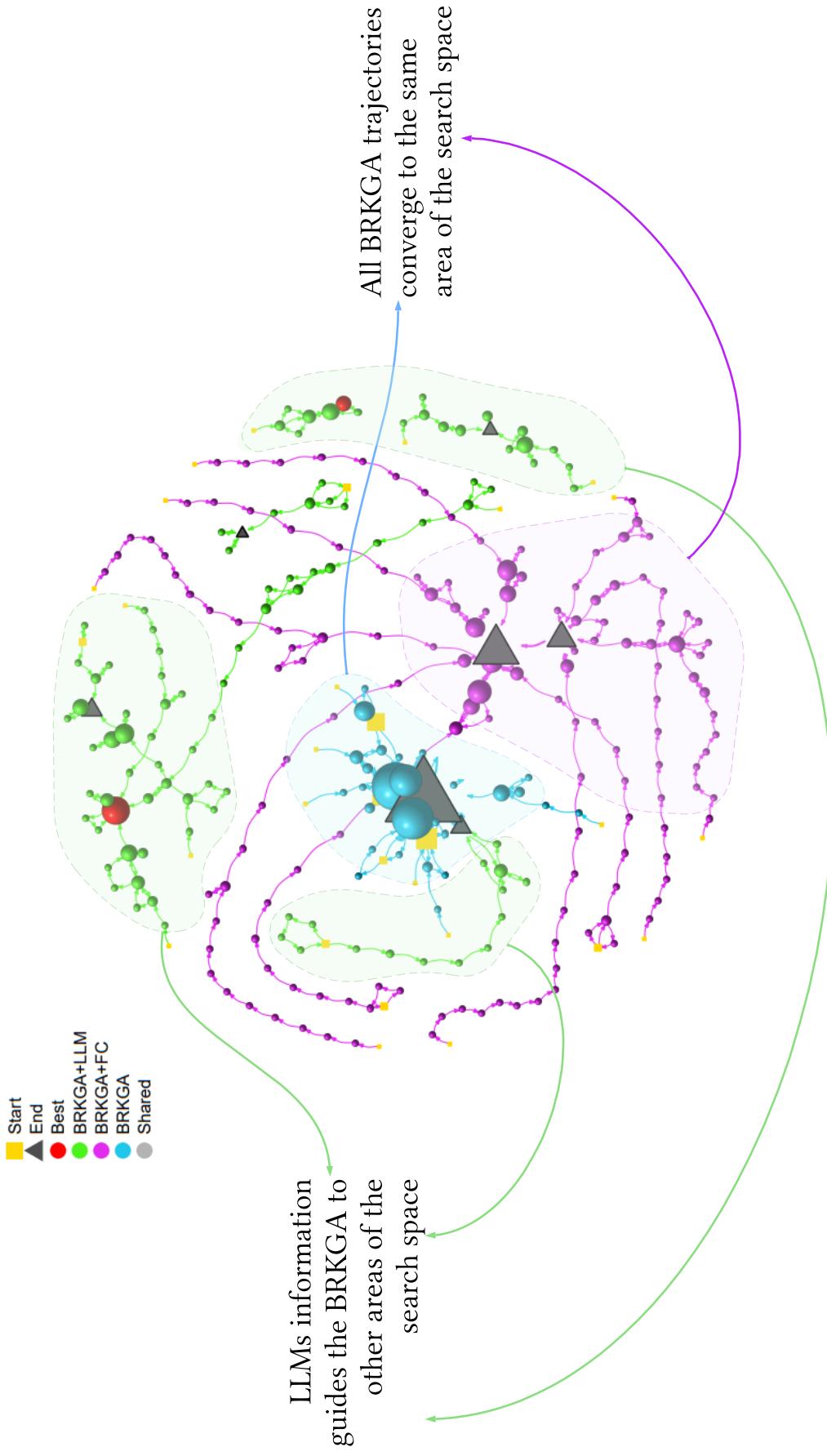


Figure 5.6: STNWeb-generated plot comparing the trajectories of BRKGA (cyan), BRKGA+FC (magenta), and BRKGA+LLM (green) over 10 runs on the *soc-hamsterster* instance (with $d = 1$ and $k = 32$). This plot was generated using the so-called agglomerative clustering partitioning method available in STNWeb, with the number of clusters set to approximately 20% of the total, allowing for a visualization focused on the essential characteristics.

oversight. As with any powerful tool, LLMs should be approached with caution, letting empirical results guide adoption.

- **Do private LLMs have a monopoly on performance?** Historically, the best-performing models (e.g., GPT-4, Claude-3-Opus) were proprietary. Our results confirm Claude’s advantage. However, recent open-source alternatives—such as Cohere’s Command-R+, Mistral’s Mixtral, and Meta’s LLaMA 3—are rapidly closing the gap [55, 127, 42]. Still, even these models are backed by well-funded private entities. Training high-performing LLMs from scratch remains out of reach for small teams, raising concerns about transparency and accessibility [82].
- **What are the main barriers to adopting our approach?** Two stand out: computational cost and context limitations. Even modest graph instances (7,000 nodes) can hit the token limits of current models (see Section 5.5.1). While models like Gemini 1.5 offer extended context windows (up to 2.8M tokens [204]), general adoption requires either more efficient prompt strategies or context compression techniques [94], neither of which we explored in this study.
- **How should researchers approach LLM-based reasoning?** Claims about LLMs’ reasoning abilities often hinge on how “reasoning” is defined. In our context, it refers to identifying useful patterns in node metrics. We’ve shown that LLMs follow structured instructions and yield non-random results. Still, it’s crucial to remain aware of the philosophical and epistemological implications. For a deeper foundation, works by Floridi [67, 68] offer a valuable starting point.
- **Can MH–LLM integration be improved?** Several hybrid approaches already exist—using LLMs to write code, solve problem descriptions, or, as we propose, detect structural patterns. We believe these are complementary rather than exclusive. A unified framework, possibly agent-based [230, 79], could orchestrate all these methods and further elevate MH research.

5.7 Conclusion

This work explored the novel use of Large Language Models (LLMs) as *pattern recognition engines* to guide and enhance metaheuristics (MHs). We applied this concept to a combinatorial optimization problem in the domain of social networks, demonstrating that LLM-generated information can meaningfully bias the search of a Biased Random Key Genetic Algorithm (BRKGA). A key component of our method is prompt engineering: the effectiveness of LLM responses critically depends on well-crafted prompts. In our case, the LLM outputs are used to assign probabilities to each node in the input graph, indicating their likelihood of belonging to an optimal solution. These probabilities then influence the MH search process.

Our hybrid method outperformed both the baseline BRKGA and a state-of-the-art BRKGA variant augmented with a hand-crafted, trained graph neural network. This result highlights the potential of LLMs to serve as powerful, general-purpose tools for structure-aware optimization without the need for expensive model training or task-

specific feature engineering.

This initial exploration opens new research avenues, particularly in extending LLM-guided optimization to a wider range of problem domains and further refining prompt strategies to maximize interpretability and performance.

Note

This study was among the earliest to combine metaheuristics with Large Language Models and was published in IEEE Access. Whether it will have any impact or be cited in the future remains uncertain, but I thoroughly enjoyed working on it!

6

Enhancing a CMSA Heuristic for the Maximum Independent Set Problem with Large Language Models

Foundational Work for This Chapter

This chapter is based on the following publication:

- **Title:** *Improving Existing Optimization Algorithms with LLMs*
- **Published in:** arXiv
- **Type:** Conference Paper (under revision)
- **Year:** 2025
- **Hybrid Metaheuristic used:** Construct, Merge, Select & Adapt (CMSA)
- **Main contribution:** Enhance the performance of CMSA through the use of LLM-generated code improvements
- **Problem addressed:** Maximum Independent Set
- **Type of contribution:** Methodological
- **DOI:** <https://arxiv.org/abs/2502.08298>
- **Current number of citations in Google Scholar:** 3

6.1 Introduction

Since 2024, and after completing the work presented in Chapter 5, I had resisted using LLMs for code generation in my research. There were several reasons for this hesitation—perhaps because I anticipated skepticism or resistance from some peers. But after running a series of experiments, and taking advantage of the latest LLM versions (which evolve week by week!), I grew curious: what if I provided a sophisticated, non-trivial algorithm as context and used an LLM as an assistant to discover

new heuristics tailored to that algorithm? Not to invent heuristics never seen before, but to generate original ones specific to that setting. I chose the Construct, Merge, Solve & Adapt (Construct, Merge, Solve & Adapt (CMSA)) algorithm, originally designed by my PhD supervisor. And when the results turned out to be promising, I thought, “We have to publish this!”

In this chapter, we demonstrate how LLMs like GPT-4o can be leveraged to enhance a sophisticated optimization algorithm, specifically the CMSA hybrid meta-heuristic [21, 24, 23]. Starting with an expert-developed C++ implementation of CMSA for the Maximum Independent Set (MIS) Problem (roughly 400 lines of code), we employed an in-context prompting strategy combined with interactive dialogue (see Figure 6.1). Our findings show that the LLM successfully understood the intricate logic and parameter interactions within the CMSA implementation, offering insightful suggestions that led to novel, algorithm-specific heuristics and improvements to the codebase. This case highlights the potential of LLMs as effective assistants for refining and extending complex optimization algorithms.

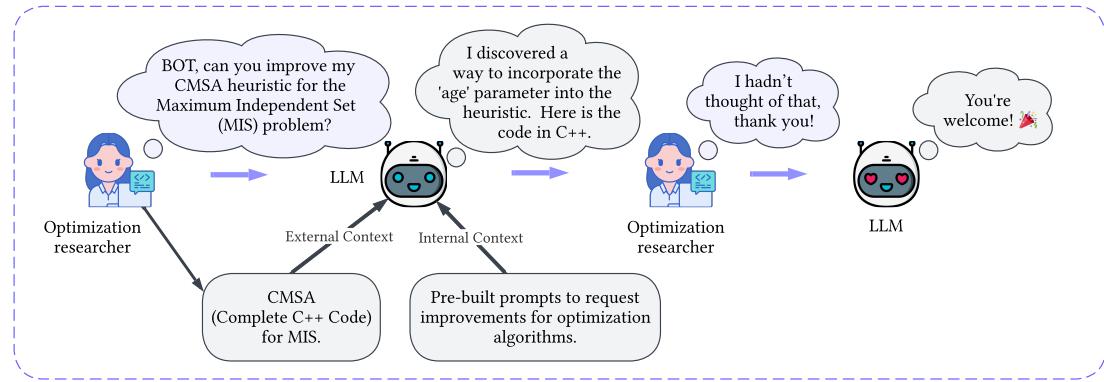


Figure 6.1: A dialogue showing how a chatbot applies our approach to improving optimization algorithms.

6.2 Background

6.2.1 Code Generation with LLMs

Code generation has emerged as a prominent research area within the field of LLMs [95, 98, 43]. The core idea is simple: given a natural language prompt (e.g., “I need an algorithm to sort a list of numbers in Python”), an LLM can return a working implementation (e.g., Quicksort) in the specified language. This capability is made possible by the extensive training of LLMs on vast datasets that include source code from GitHub, StackOverflow, technical documentation, academic papers, and publicly available books.

The specificity of a prompt often influences the sophistication of the output. For instance, a more detailed prompt such as *“I need an efficient sorting algorithm for a list of numbers in Python, capable of parallel execution and using modern optimization techniques”* might lead GPT-4o to generate a ParallelMergeSort. Further improvements can be obtained through iterative prompting (e.g., *“Find new ways to improve it”*), which may result in versions incorporating NumPy-based memory optimization, hybrid sorting techniques, or more efficient merging strategies. However, the output could vary if the goal shifts (e.g., prioritizing memory efficiency over parallelism), highlighting the importance of both prompt design and interactive refinement [248].

The quality of generated code is closely tied to the training data and scale of the LLM. As both factors improve, so does performance [238]. Still, no LLM can consistently generate flawless code. For example, even a correct ParallelMergeSort implementation may include subtle bugs. Debugging through model interaction (e.g., pasting runtime errors) remains a necessary step. In some cases, LLMs can even generate their own test cases to validate outputs. To evaluate LLMs on such tasks, standardized benchmarks like HUMAN EVAL [44] are commonly used. These include problems covering algorithmic thinking, basic programming, and mathematics—akin to coding interviews.

Code generation, however, spans a wide range of domains: data science (e.g., statistical modeling [222]), systems programming (e.g., memory or low-level operations [97, 52]), frontend development (e.g., UI prototyping [231]), and optimization. Each domain presents unique challenges and constraints. Our work focuses on the latter, specifically metaheuristic optimization algorithms.

Within this domain, automatic code generation intersects with a central theoretical insight: the *No Free Lunch Theorem* [226], which asserts that no single metaheuristic performs best across all optimization problems. This insight motivates the idea of LLMs as tools for generating tailored metaheuristics that adapt to specific problems. As black-box models, LLMs could assist in discovering novel operators or algorithmic components that outperform conventional approaches [194].

A seminal contribution in this direction was *FunSearch* [175], which used LLMs to evolve novel heuristics for the Bin Packing Problem. However, recent evaluations [190] suggest that the heuristics discovered lacked generalization. *FunSearch* begins from an incomplete base implementation and relies on trial-and-error exploration. In contrast, our approach starts from a complete, expert-designed implementation and uses LLMs to discover meaningful variations.

Other notable contributions include *LlaMEA* [194], a framework combining evolutionary algorithms and LLMs to generate metaheuristics in real-time; *LLM-GP* [84], which evolves genetic operators through LLM-assisted programming; and the decomposition of swarm intelligence techniques via prompting, as explored in [165].

Despite their promise, these works mostly focus on generating new algorithms from scratch. However, the optimization research community already possesses a vast library of high-quality, handcrafted algorithms. A quick search for “optimization algo-

arithm” on GitHub yields tens of thousands of repositories. We believe this motivates a complementary direction: using LLMs to improve existing algorithms.

In this research, we adopt this perspective by treating LLMs as research assistants capable of working with expert-developed code. Our goal is not to replace but to augment human insight—helping identify improvements, propose new heuristics, or simplify complex logic within sophisticated optimization frameworks. This paradigm leverages the knowledge embedded in LLMs through pretraining and unlocks new opportunities for collaboration between human designers and generative models.

6.2.2 Maximum Independent Set (MIS) Problem

To validate our hypothesis that LLMs can enhance existing optimization algorithms, we focus on solving the *Maximum Independent Set* (MIS) problem—an NP-hard combinatorial problem with well-established applications in network design, scheduling, and bioinformatics.

Formally, given an undirected graph $G = (V, E)$, the MIS problem seeks the largest subset $S \subseteq V$ such that no two vertices in S are adjacent, i.e., for all $u \neq v \in S$, there is no edge $(u, v) \in E$.

Figure 6.2 illustrates three example graphs, each with an optimal MIS solution (highlighted as non-white nodes).

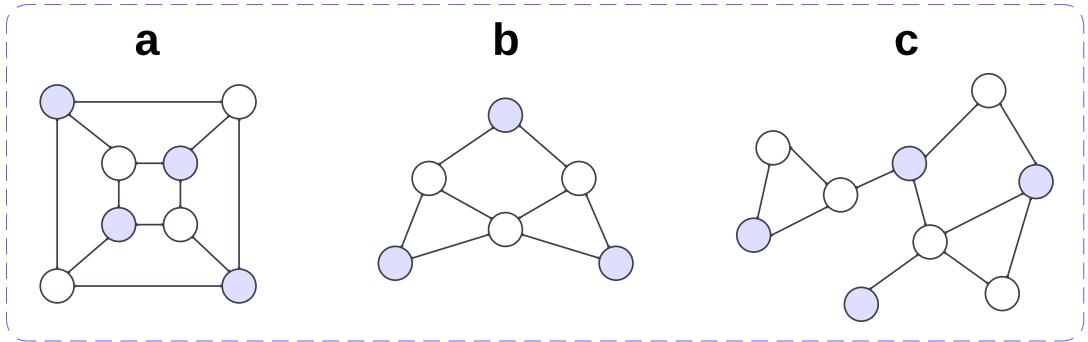


Figure 6.2: Examples of maximum independent sets.

6.2.3 CMSA

Construct, Merge, Solve & Adapt (CMSA) is a hybrid metaheuristic—also known as a matheuristic—that integrates classical metaheuristics with exact methods, such as Integer Linear Programming (ILP) solvers, for tackling combinatorial optimization problems [24]. Each iteration of CMSA follows four key phases:

1. **Construct:** In this initial phase, new candidate solutions are generated. This is typically achieved through a probabilistic greedy mechanism. Instead of making purely deterministic choices, the algorithm introduces an element of randomness,

where components are selected based on probabilities derived from heuristic information. This allows for exploration of the search space beyond purely greedy paths. The quality and diversity of the initial solutions generated in this phase significantly impact the overall performance of the CMSA framework. **This crucial step is revisited and enhanced in the next section.**

2. **Merge:** Following the construction of new solutions, the Merge phase aggregates promising components from these and potentially other high-quality solutions found so far. The goal is to identify a subset of variables or solution elements that appear frequently in good solutions. This aggregation process defines a reduced, yet representative, subproblem that captures the most critical aspects of the original problem, making it amenable to exact solution.
3. **Solve:** The core of the matheuristic approach lies in this phase. The reduced subproblem, defined in the Merge phase, is passed to an exact solver. For the Maximum Independent Set (MIS) problem, this typically involves an Integer Linear Programming (ILP) solver. The exact solver then finds an optimal solution for this smaller, well-defined subproblem. This ensures that the most promising parts of the solution space are explored with guaranteed optimality, leveraging the power of exact methods.
4. **Adapt:** In the final phase, the CMSA framework updates its internal parameters and data structures based on the quality of the solution obtained from the Solve phase. This adaptive mechanism is crucial for the metaheuristic's learning capability. It might involve adjusting the probabilities used in the Construct phase, refining the criteria for the Merge phase, or updating other parameters that guide the search. This feedback loop allows the algorithm to learn from its successes and failures, continuously improving its search strategy over iterations.

This architecture balances the exploration strengths of metaheuristics with the precision of exact solvers applied to smaller, focused subspaces.

The behavior of CMSA is governed by several key parameters. Each solution component has an associated *age*, initialized to zero when added to the subproblem. If a component is not included in the optimal solution returned by the solver, its age increases. The parameter age_{max} defines the threshold at which aged-out components are removed, promoting diversity and avoiding stagnation.

Additional parameters include:

- n_a : Number of solution constructions per iteration.
- t_{max} : Total allowed runtime.
- t_{limit} : Time limit for the ILP solver per iteration.
- $0 \leq d_{rate} \leq 1$: Degree of determinism in the construction phase—higher values yield more deterministic solutions, while lower values increase randomness.

Proper tuning of these parameters enables effective exploration, exploitation, and efficient use of computational resources.

CMSA was selected for this study due to its implementation complexity compared to simpler metaheuristics. Implementing CMSA for the Maximum Independent Set (MIS) problem in C++ requires precise definition and integration of all four phases, as well as careful attention to performance and design. This complexity grows with more challenging optimization problems, demanding expertise in both metaheuristics and exact methods, as well as advanced programming skills.

For our experiments, we used the original C++ implementation of CMSA for MIS, developed by the algorithm’s creator and made available on his website (<https://www.iiia.csic.es/~christian.blum/>). This choice serves a dual purpose: ensuring a robust, expert-level baseline and creating a realistic setting to test whether LLMs can meaningfully improve code written by specialists. Our results are discussed in the next section.

6.3 LLM-Enhanced CMSA for MIS

This section introduces a methodology for leveraging LLMs to enhance existing, expert-crafted optimization algorithms. Unlike prior approaches that often generate code from scratch or rely on simplified implementations, our method focuses on dialog-based interaction with an LLM via a chatbot interface. This interaction aims to uncover novel code improvements that even seasoned experts can benefit from. We detail this process in the following subsections.

6.3.1 Discovering New Heuristics

LLMs excel as pattern-recognition machines [147, 185], and code provides a rich source of structured textual patterns. This enables LLMs to identify underutilized code elements—such as variables or functions—that could play a strategic role within a heuristic. Such insights might be overlooked by human experts, particularly in complex codebases. These discoveries demand a deep syntactic and semantic understanding of both the code and the optimization problem, allowing LLMs to suggest meaningful modifications to heuristic strategies that transcend superficial code changes. In this capacity, LLMs serve as advanced assistants for optimization algorithm designers, offering novel heuristic improvements informed by reasoning over their vast pretraining knowledge.

To demonstrate our methodology, we employ an LLM to improve a heuristic within the CMSA code, specifically for solving the Maximum Independent Set (MIS) problem. The function `generation_solution()` implements the solution construction phase of the CMSA framework [21]:

```
while (int(positions.size()) > 0) {
    double dec = standard_distribution(generator);
    int position = 0;

    if (dec <= determinism_rate) {
        position = *(positions.begin());
```

```

} else {
    int max = candidate_list_size;
    if (max > int(positions.size())) {
        max = int(positions.size());
    }

    double rnum = standard_distribution(generator);
    int pos = produce_random_integer(max, rnum);

    set<int>::iterator sit2 = positions.begin();
    for (int i = 0; i < pos; ++i) {
        ++sit2;
    }
    position = *sit2;
}

greedy_sol.score += 1;
greedy_sol.vertices.insert(increasing_degree_order[position]);

if (age[increasing_degree_order[position]] == -1) {
    age[increasing_degree_order[position]] = 0;
}

positions.erase(position);

for (auto sit = neigh[increasing_degree_order[position]].begin();
     sit != neigh[increasing_degree_order[position]].end(); ++sit) {
    positions.erase(position_of[*sit]);
}
}
}

```

Listing 6.1: Probabilistic greedy algorithm for MIS in CMSA.

It implements a greedy randomized construction heuristic that selects exactly one vertex $v_i \in \tilde{V}$ at each step (where \tilde{V} is the set of nodes that can feasibly be chosen), until the MIS solution is complete:

$$v_i = \begin{cases} v_{\min} & \text{if } r \leq \alpha \\ v_{\text{random}} \in \text{CL}(k) & \text{otherwise} \end{cases}$$

where

- v_{\min} is the vertex with minimum degree (among the ones from \tilde{V})
- r is a random number between $[0, 1]$
- α is the determinism rate
- $\text{CL}(k)$ is a candidate list of size k
- v_{random} is a randomly selected vertex from the candidate list

The heuristic combines deterministic greedy selection (based on vertex degree) with randomization to create diverse solutions. It selects vertices either by taking the best available vertex (greedy choice) with probability α , or by randomly selecting from a restricted candidate list with probability $(1 - \alpha)$.

New Heuristic from the LLM

The original greedy heuristic, while computationally efficient, notably overlooked the age variable—a fundamental component of the CMSA framework. This variable was solely used for solution restarts, not for guiding the crucial candidate selection mechanism. In Figure 6.3, we present two illustrative dialogues demonstrating our interaction approach with the LLM.

In the first dialogue, case (a), the LLM proposes an enhanced heuristic that intelligently combines both node degrees and current age values. The vertex selection mechanism suggested by the LLM can be formally described as follows:

$$v_i = \begin{cases} \operatorname{argmin}\{P_w(v_j) \mid v_j \in \tilde{V}\} & \text{if } r \leq \alpha \\ \text{roulette-wheel selection w.r.t. } P_w(\cdot) \text{ values} & \text{otherwise} \end{cases}$$

where $P_w(v_j)$ is a weighted probability:

$$P_w(v_j) = \frac{w(v_j)}{\sum_{v_l \in V} w(v_l)}, \quad w(v) = \frac{1}{2 + \operatorname{age}(v)} + \frac{1}{1 + \operatorname{degree}(v)} \quad (6.1)$$

The weight function $w(v)$ is designed to favor vertices with lower age and degree values, thereby promoting diversity in the selection process. This demonstrates the LLM’s ability to modify the original heuristic by integrating age values to diversify node selection, while simultaneously balancing this with degree information through a composite weight function. Intuitively, decreasing the probability of selecting solution components with high age values for newly constructed solutions is a sound strategy.

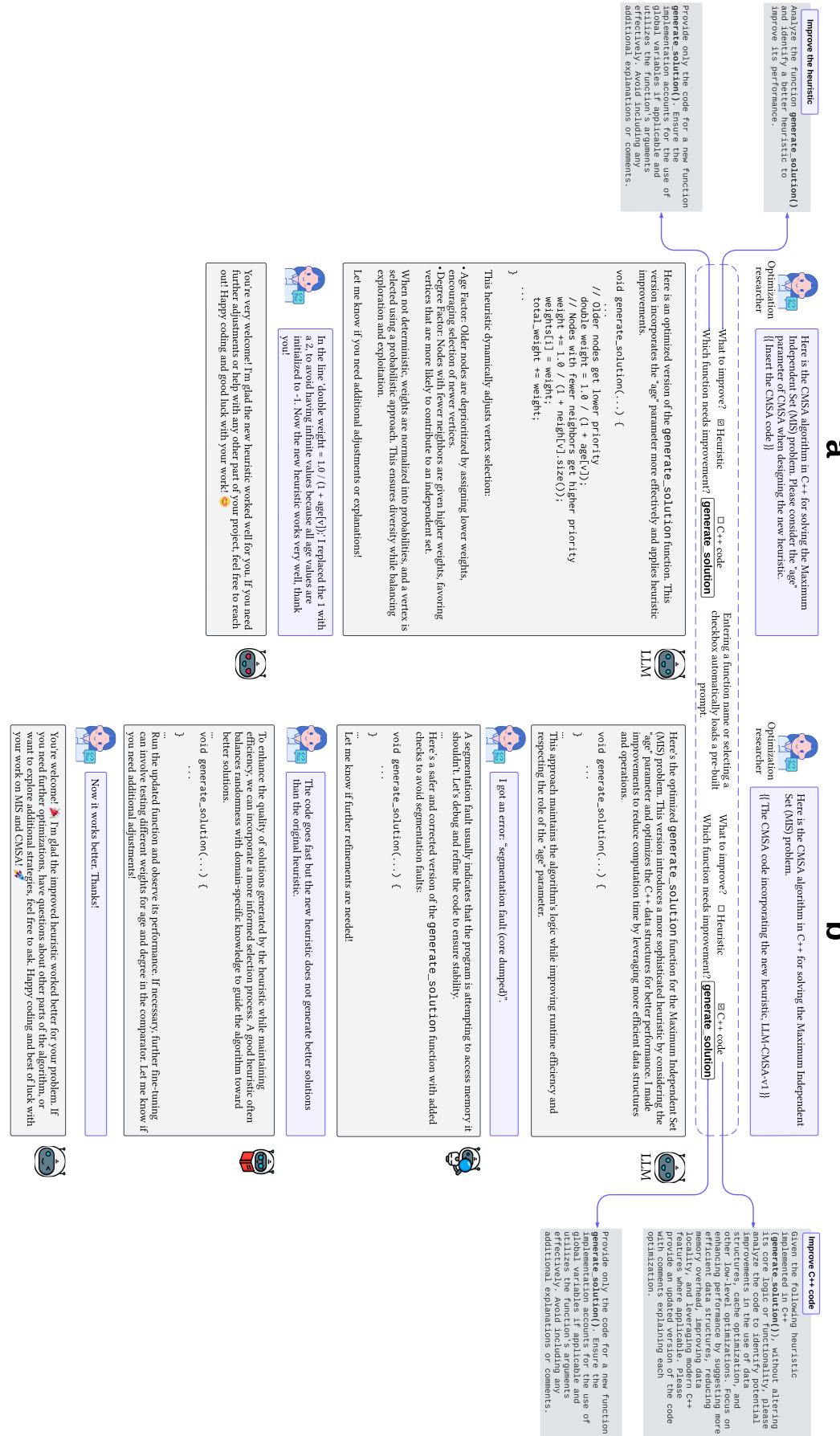
Despite successfully identifying this previously overlooked application of the age variable within the CMSA construction heuristic, any dialog-based system inherently requires human feedback [39]. Consequently, the LLM occasionally makes errors. For instance, as depicted in Figure 6.3 (a), the line `double weight = 1.0 / (1 + age[v])` could lead to a division by zero, as age values in CMSA are set to -1 for solution components not belonging to the subproblem. The human-provided fix involved replacing `1 + age[v]` with `2 + age[v]`. Remarkably, this specific utilization of the age variable—despite its initial flaw and the subsequent human correction—had never been considered by any researchers working on CMSA algorithms before.

This new CMSA variant is henceforth called LLM-CMSA-V1. It was obtained by replacing the `generate_solution()` of the original CMSA implementation with the new function provided by the LLM.

Improving LLM-CMSA-V1 with the LLM

Since our methodology relies on dialog-based interactions with the LLM via a chatbot interface, it is also possible to request improvements to previously generated code—still

Figure 6.3: Two LLM interaction patterns: (a) a direct request to improve a heuristic using CMSA's *age* parameter, and (b) an iterative dialogue to enhance both heuristic quality and C++ performance through error correction. Both use in-context learning as a prompting strategy [58, 115].



available in the current session context. For example, one might ask: “*Are there ways to enhance the dynamic selection heuristic to allow for a more diverse and advanced search?*” The LLM responds with several suggestions, one of which involves incorporating the concept of entropy. The C++ code provided by the LLM is characterized by a corresponding change, which involves replacing the definition of $P_w(\cdot)$ (see Equation 6.1) with the following entropy-adjusted probabilities:

$$P_H(v_j) = \frac{P_w(v_j) + H}{\sum_{v_l \in V} P_w(v_l) + H} ,$$

where

$$H = - \sum_{v_l \in V} P_w(v_l) \log(P_w(v_l)) .$$

The entropy adjustment increases selection diversity by adding the system’s uncertainty to each probability.

This CMSA variant is henceforth called LLM-CMSA-V2. We directly replaced the `generate_solution()` function of the original CMSA with the LLM-generated code.

6.3.2 Code Optimization Strategies

After discovering the two new CMSA variants with the help of the LLM, it is also possible to request a different kind of improvement—not in terms of proposing new algorithmic heuristics, but rather in enhancing the underlying C++ code. These enhancements may involve the use of more efficient data structures, changes in data types, or other low-level optimizations that preserve the algorithmic logic. For instance, one might ask: “*Could the LLM create an improvement at the C++ code level?*” In other words, is there a more efficient way to implement the new CMSA variants without altering their core behavior?

It is reasonable to assume that, since the LLM has been pretrained on vast amounts of source code, it could suggest highly optimized C++ implementations—even for contexts such as optimization algorithm design (e.g., metaheuristics), where low-level improvements to data structures or code organization are not typically the focus of human designers. This leads us to the next prompt, which we apply to both previously generated heuristics, LLM-CMSA-V1 and LLM-CMSA-V2:¹

Human prompt

Given the following heuristic implemented in C++ (`generate_solution()`), without altering its core logic or functionality, please analyze the code to identify potential **improvements in the use of data structures, cache optimization, and**

¹ This prompt uses the “C++ code” checkbox, as shown in Figure 6.3 (b).

other low-level optimizations. Focus on enhancing performance by suggesting more efficient data structures, reducing memory overhead, improving data locality, and leveraging modern C++ features where applicable. Please provide an updated version of the code with comments explaining each optimization.

However, dialog-based interactions with the LLM are not free from hallucinations—that is, the model may generate code with bugs or memory management issues in C++ (as shown in Figure 6.3 (b)). One effective way to address these problems is through human feedback, engaging in a trial-and-error refinement process with the model—a common practice in code generation workflows (see [39]). Eventually, the LLM produces code that compiles successfully. Upon reviewing the final output, it becomes evident that the generated code is more complex, less readable, and incorporates unconventional or less familiar programming techniques. For example, it replaces the `set` and `vector` data structures with more advanced, low-level alternatives.²

```
#include <bitset>
...
// Constants for optimization
constexpr size_t BLOCK_SIZE = 64;
...
// Bitset for boolean operations
std::bitset<32768> available;
// Adjust size based on max n_of_vertices
available.set();
// Aligned vector to optimize cache usage
alignas(BLOCK_SIZE) std::vector<int> active_vertices;
active_vertices.reserve(n_of_vertices);
```

Listing 6.2: Fragment code optimization suggested by LLM.

Meanwhile, the original code is shown in the following listing.

```
set<int> positions;
vector<int> position_of(n_of_vertices, 0);
for (int i = 0; i < n_of_vertices; ++i) {
    positions.insert(i);
    position_of[increasing_degree_order[i]] = i;
}
```

Listing 6.3: Fragment of original code from CMSA.

The logic remains the same, except for changes in variable names: `positions` is replaced with `available`, and `position_of` is substituted with `active_vertices`.³

Although the changes do not affect the heuristic’s logic but rather the underlying C++ structures, they do not always lead to measurable improvements in efficiency, specifically in reducing the runtime, which in practice would allow us to explore better candidates when building a valid CMSA solution. Nevertheless, the generated code compiles and runs correctly.⁴

² The comments in the code were generated by the LLM.

³ Explicitly instructing the prompt to retain the variable names might have avoided this issue.

⁴ Visit our project website for more details on the C++ optimizations suggested by GPT-4o.

The two new CMSA variants with these performance improvements will be named: LLM-CMSA-V1-PERF and LLM-CMSA-V2-PERF.

This iterative process with the LLM, asking it to identify underutilized variables or functions in the existing code, demonstrates its potential as a sophisticated assistant for optimization experts. For instance, in this case, the LLM proposes a new CMSA construction heuristic utilizing the `age` parameter. Unlike simple code generation, the LLM enhances not only the algorithm itself but also the existing code (e.g., C++ implementations), suggesting improvements without altering its logic. This opens the door to using LLMs not just for developing optimization algorithms but also for updating and refining sophisticated legacy code, leveraging the extensive knowledge embedded in LLMs.

6.3.3 Reproducibility

Although it is not possible to replicate the exact results of an LLM, due to their autoregressive nature that predicts the most probable token based on a probability distribution (with the next token being determined stochastically) [101], it is possible to reproduce similar responses by using the same prompts, the same LLM, and its parameters. For this reason, our repository (<https://imp-opt-algo-llms.surge.sh/>) includes a *chatbot* that implements the same prompts used in our research (as shown in Figure 6.3). In fact, each element in Figure 6.3 (textbox and checkbox) loads pre-built prompts, known as in-context prompts [58, 115], to eliminate the need for manual input. Thus, our chatbot features two types of in-context prompts: (1) external ones, related to the C++ CMSA code for the MIS, and (2) internal ones, focused on improving the heuristic, the C++ code, and specifying which function in the code requires enhancement. Next, we will assess the quality of the heuristics proposed by the LLM.

6.4 Empirical Evaluation

This section is divided into two parts: the preliminary phase (setup, benchmark, and CMSA parameter tuning) and the results.

6.4.1 Preliminary

Our experimental setup involved selecting an appropriate LLM and defining the benchmark and computational environment.

LLM Selection and Parameters

We initially explored various LLMs using Chatbot Arena [47] to identify a suitable model without incurring costs.⁵ Ultimately, we selected GPT-4o (version: 2024-11-20)

⁵ <https://lmarena.ai/>

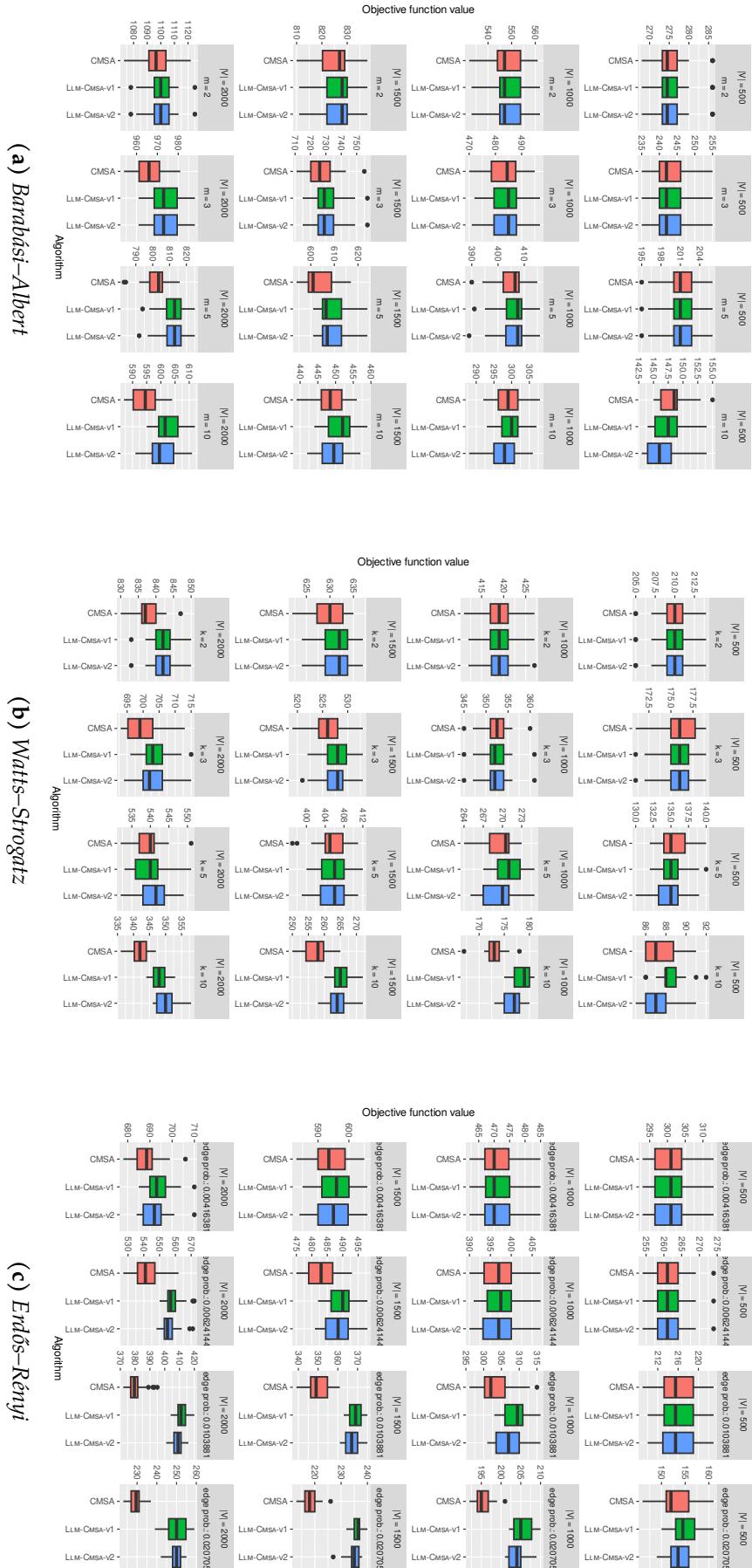


Figure 6.4: Comparative analysis of solution quality: Original CMSA vs. LLM-CMSA variants (V1 and V2).

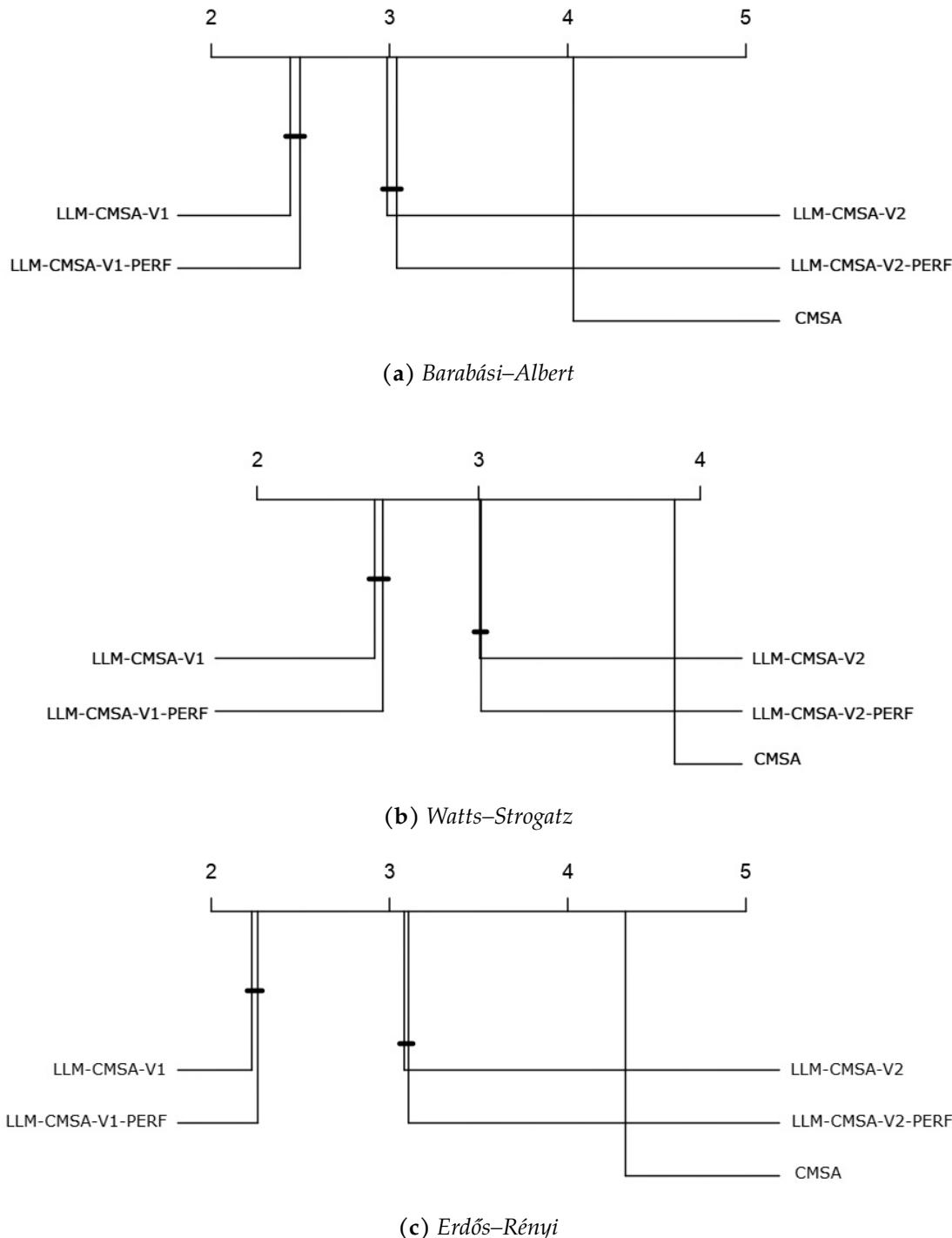


Figure 6.5: Critical Difference (CD) plots for different graph types. Algorithms connected by the same horizontal bar do not exhibit a statistically significant difference in performance. (a) **Barabási-Albert:** The top-performing group, consisting of LLM-CMSA-V1 and LLM-CMSA-V1-PERF, significantly outperforms the lower group (LLM-CMSA-V2 and CMSA). Within each group, performances are statistically equivalent. LLM-CMSA-V2-PERF ranks last. (b) **Watts-Strogatz** and (c) **Erdős-Rényi:** In both graph types, LLM-CMSA-V1 and LLM-CMSA-V1-PERF are the top performers and do not differ significantly from each other. Both significantly outperform the other methods, while LLM-CMSA-V2 and CMSA exhibit the lowest performance.

for our experiments, given its status as one of the top-performing models currently available. The LLM was used with its default parameters:

- `temperature` = 0.7 (controlling response randomness),
- `top-p` = 1 (nucleus sampling threshold),
- `max-output-tokens` = 2048 (maximum output tokens)

Computational Environment and Benchmarks

Experiments for algorithm variants CMSA, LLM-CMSA-V1, and LLM-CMSA-V2 were conducted on a cluster equipped with Intel® Xeon® CPU 5670 processors (12 cores at 2.933 GHz) and 32 GB of RAM.

Our benchmark set comprises three graph types: Barabási-Albert, Watts-Strogatz, and Erdős-Rényi. For each type, we used four different sizes and four density levels (see Figure 6.4). This resulted in 1 tuning instance and 30 testing instances for each size-density combination, totaling 48 tuning instances and 1440 testing instances. The parameters for all three CMSA variants (CMSA, LLM-CMSA-V1, LLM-CMSA-V2) were tuned using `irace` [129]. Computation time limits were set at 150, 300, 450, and 600 CPU seconds for the four different graph sizes, respectively.

6.4.2 Numerical Results

In addition to the three tuned algorithms, we evaluated two variants, LLM-CMSA-V1-PERF and LLM-CMSA-V2-PERF, derived from the efficiency-improved C++ codes (Section 6.3.2). Each of the five algorithms was run once per testing instance. Figure 6.4 presents box plots for the main three variants, indicating graph size and density. A key observation is that both LLM-generated CMSA variants consistently outperform the standard CMSA as graph size and density increase. This strongly supports the LLM’s suggestion to utilize age values for solution component selection during construction.

To establish statistical significance, we generated critical difference (CD) plots (Figure 6.5). These plots are based on the **Nemenyi post-hoc test**, which follows a **Friedman test** for multiple comparisons. The Friedman test first ranks algorithms across multiple problem instances. The Nemenyi test then identifies statistically significant differences between these ranked algorithms. In the CD plots, whiskers represent average algorithm rankings across problem instances. Crucially, algorithms whose whiskers are connected by a bold horizontal bar are considered statistically equivalent according to the Nemenyi test at a significance level of $\alpha = 0.05$.

Figure 6.5 shows three CD plots, one per graph type, including the efficiency-optimized LLM-generated CMSA variants. Observations: Both LLM-CMSA-V1 and LLM-CMSA-V2 significantly outperform CMSA across all graph types. However, LLM-CMSA-V1 consistently outperforms LLM-CMSA-V2, indicating that using entropy of selection probabilities was not beneficial. Furthermore, the efficiency-optimized

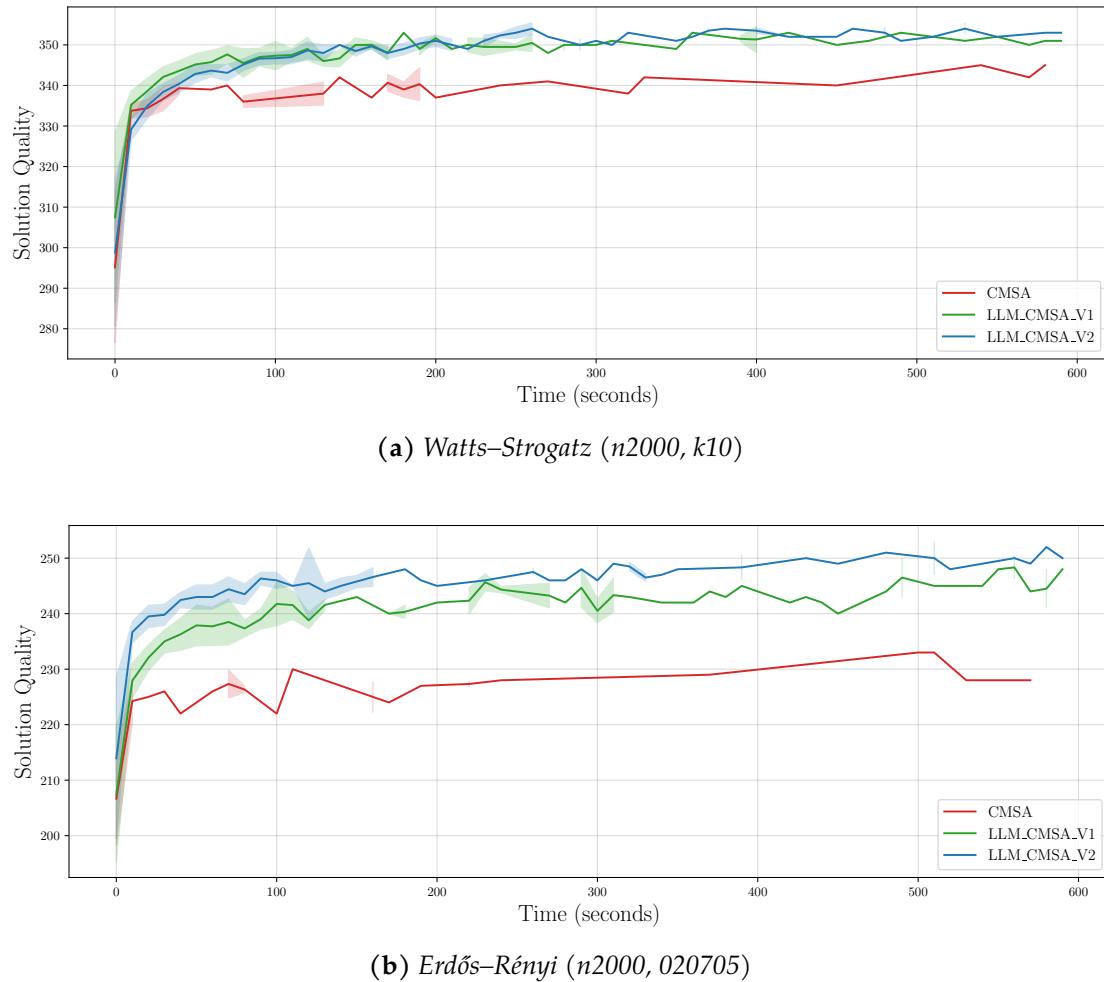


Figure 6.6: Examples of algorithm evolution over time.

variants are statistically equivalent to their non-optimized counterparts. This suggests that while they might offer RAM savings (an untested hypothesis), they do not yield better results. Preliminary checks also showed no significant runtime improvements from these C++ optimizations in our test environment, implying the original expert implementation was already highly efficient or the specific low-level changes had minimal impact.

Finally, Figure 6.6 displays two representative convergence examples from 10 runs of the three main CMSA variants. In both cases, the LLM-generated CMSA variants rapidly achieve solution qualities that standard CMSA does not reach even by the end of its runs.

6.5 Discussion

Our study demonstrates LLMs' capacity for algorithmic reasoning, successfully identifying conceptual enhancements for CMSA. Specifically, the LLM-CMSA-V1 heuristic outperformed the expert baseline. Interestingly, the LLM's subsequent, more complex suggestion incorporating entropy (LLM-CMSA-V2), while plausible, proved less effective. This highlights that added complexity may not always be beneficial, reinforcing the need for rigorous empirical validation of any proposed heuristic. Our study, however, has limitations: we only tested GPT-4o for CMSA improvements due to space constraints; future work should compare additional LLMs, especially open-weight ones, and explore enhancing other complex algorithms for diverse optimization problems. These limitations could be addressed in an extended article. Despite these, our comprehensive analysis of CMSA with new heuristics for MIS instances opens promising new research avenues:

1. **Specialized Benchmarks.** Our study, while demonstrating LLMs' potential in enhancing CMSA for the MIS problem, highlights a broader gap in the field: the scarcity of benchmarks specifically designed for LLMs in optimization. General-purpose code generation benefits immensely from standardized benchmarks that allow for consistent evaluation and comparison of models. Similarly, to rigorously assess and advance LLMs' ability to discover and implement improved heuristics for complex optimization problems, we urgently need domain-specific benchmarks. These benchmarks should encompass a diverse range of problem types, instance sizes, and performance metrics, moving beyond simple code correctness to evaluate true algorithmic innovation and efficiency.
2. **LLM-Based Agent Integration.** Figure 6.3 depicts a manual, human-in-the-loop interaction with the LLM. However, the potential for autonomous agents to handle repetitive or complex tasks, such as code execution, debugging, and iterative refinement, is immense. Imagine a sophisticated platform where multiple LLM-powered agents collaborate, each specializing in a different aspect of algorithm improvement (e.g., one for heuristic discovery, another for code optimization, a third for performance testing). Such a collaborative multi-agent system could significantly accelerate the discovery and refinement of existing optimization algorithms, potentially leading to major breakthroughs in efficiency and solution quality [96, 124]. This paradigm shift could transform how optimization algorithms are designed and deployed.
3. **Programming Language Translation.** Optimization algorithms, especially high-performance ones, are often implemented in languages like C++ for speed, but might need to be translated to other languages (e.g., Python for prototyping, Java for enterprise systems) for broader applicability, easier integration, or improved maintainability. This translation process is typically labor-intensive and prone to errors. LLMs, with their advanced understanding of code syntax and seman-

tics, can significantly assist in this process [160, 109]. They could automate large parts of the translation, identify potential performance bottlenecks in the target language, and even suggest idiomatic translations that preserve or enhance efficiency. While LLMs offer a powerful starting point, domain-specific fine-tuning on optimization codebases might be necessary to achieve highly optimized and bug-free translations.

An important question arises: should the LLM receive credit for discovering a superior heuristic compared to a human expert's best implementation? While prompt design influences outcomes, questions concerning authorship, ownership, and AI's role in scientific discovery warrant further consideration [215].

6.6 Conclusions

Our research demonstrates the effective application of LLMs to enhance existing optimization algorithms. Using the non-trivial C++ implementation of the Construct, Merge, Solve, and Adapt (CMSA) algorithm for the Maximum Independent Set problem, GPT-4o successfully understood its operational context via in-context prompts. It then proposed conceptually new heuristics for the probabilistic construction phase. A thorough comparative analysis showed these LLM-suggested heuristics outperformed expert-implemented ones, highlighting LLMs' potential not merely as tools, but as intelligent collaborators in complex algorithm design.

Note

This study demonstrates the feasibility of using LLMs as assistants in the design of complex hybrid metaheuristics like CMSA. Leveraging their semantic analysis capabilities, LLMs can detect underutilized variables or structures within the code and suggest meaningful ways to integrate them into other parts of the implementation.

Once this work was completed, in early 2025, I wanted to continue along this line of research. I believed (and still believe) that it could be highly valuable not only for researchers but also for people without a strong theoretical background in computational optimization. This led to my next investigation—see the following chapter.

7

Improvement of Optimization Algorithms with Large Language Models by Non-expert Users

Foundational Work for This Chapter

This chapter is based on the following publication:

- **Title:** *Combinatorial Optimization for All: Using LLMs to Aid Non-Experts in Improving Optimization Algorithms*
- **Published in:** arXiv
- **Type:** Journal Paper (under revision)
- **Year:** 2025
- **Optimization algorithms used:** Ant Colony Optimization (ACO)
- **Main contribution:** Boost the effectiveness of optimization algorithms by assisting non-expert users in generating better code through Large Language Models
- **Problem addressed:** Travelling Salesman Problem
- **Type of contribution:** Methodological
- **DOI:** <https://arxiv.org/abs/2503.10968>

7.1 Introduction

Large Language Models (LLMs) possess vast knowledge, including the implementation details of a wide range of algorithms. One might therefore assume that if an LLM can meaningfully assist in improving a lesser known algorithm like CMSA—with limited examples available online (see previous Chapter 6)—its suggestions for well-known classical algorithms could be even more effective.

Thus, we decided to go beyond testing a single metaheuristic and instead evaluate multiple optimization algorithms for a well-known combinatorial problem: the Travelling Salesman Problem (TSP). Our selection includes metaheuristics, reinforcement learning algorithms, deterministic heuristics, and exact methods, evaluated using various state-of-the-art LLMs. As such, this work adopts a benchmarking perspective, systematically assessing the capabilities of LLMs to improve or modernize classic implementations across a diverse set of algorithmic families.

I must admit that working on this project was incredibly enjoyable—not only because of what we discovered (that LLMs can successfully update optimization algorithm implementations written in Python!) but also because it suggests a broader implication: when properly integrated, an LLM could meaningfully assist individuals without theoretical knowledge of combinatorial optimization.

In this work, we present the first large-scale, systematic evaluation of LLMs' ability to upgrade classical algorithms across diverse families—including metaheuristics, reinforcement learning, deterministic heuristics, and exact methods. Our study centers on the canonical Travelling Salesman Problem (TSP), using a simple and reproducible prompting strategy via a chatbot interface. Although our experiments focus on the TSP, its foundational nature as a permutation problem makes it a representative case for a wide range of combinatorial problems, such as the Vehicle Routing Problem, Hamiltonian Cycle Problem, and Sequential Ordering Problem. All of these share a core challenge: determining the optimal order in which tasks should be performed to minimize total cost.

Our findings show that LLMs can propose and implement meaningful enhancements, including modernizing algorithmic components and optimizing data structures—often reducing code complexity compared to the original implementation. Importantly, these improvements are applied not to isolated heuristic snippets, as in prior work, but to full algorithmic codebases. We also analyze instances where the suggested improvements were marginal or ineffective, highlighting the current boundaries of the approach.

Ultimately, this work demonstrates a practical methodology that enables practitioners without deep expertise in optimization to access high-performance algorithmic designs, harnessing the extensive embedded knowledge of modern LLMs. A conceptual overview of our framework using a Genetic Algorithm is shown in Figure 7.1.

7.2 Background

7.2.1 Large Language Models in Combinatorial Optimization

Large Language Models (LLMs) have recently shown promise in optimization tasks [235, 126, 89] by exploiting the vast knowledge gained during their pre-training

Table 7.1: Expanded Comparative Analysis of LLM-based Algorithm Design Approaches

Feature	Our Work	Improving Optimization Algorithms [183]	Existing Algo-AlphaEvolve [152]	LLaMEA/ReEvo [194, 236]	FunSearch/Evo-Heuristics [175, 123]
Main Goal	To provide a large-scale benchmark on enhancing <i>complete</i> implementations of <i>existing</i> algorithms from diverse families.	To demonstrate a proof of concept through the improvement of a <i>single</i> , complex heuristic in a state-of-the-art algorithm.	To autonomously evolve a population of algorithms starting from a seed program, leveraging LLM-based agents to discover improved variants.	To generate new meta-heuristics or heuristic components using an evolutionary computation loop.	To discover specific mathematical functions or heuristics from scratch.
LLM Input	A complete, functional code file from a well-known framework [162].	The complete code of a single, high-performance algorithm.	An initial “seed” algorithm and a fitness function.	A set of prompts representing algorithm components (e.g., operators).	A problem description and a code skeleton for a specific function.
Improvement Process	Interactive LLM-driven prompting and refinement of full algorithms.	Interactive LLM prompting to refine a specific heuristic or function.	Evolutionary loop where the LLM acts as an advanced mutation operator to generate new programs.	Evolutionary loop where the LLM acts as a crossover/mutation operator on algorithm components.	Program search tree where the LLM proposes new function implementations.
User's Role	Practitioner/Non-expert: Provides base code, validates final solution.	Expert: Seeks to enhance a specific, already-strong algorithm.	Expert: Designs the evolutionary process and fitness function.	Expert: Designs the evolutionary framework and component prompts.	Expert: Defines the problem, evaluator, and code structure.
Key Contribution	A systematic benchmark demonstrating that a single, prompt-based methodology can holistically improve algorithms from diverse families without specialized theoretical knowledge.	A case study demonstrating the feasibility of LLM-based enhancement on a complex, expert-level algorithm.	An evolutionary framework for automated program search and algorithm improvement.	A framework for the automatic generation of algorithms.	A method for the automatic discovery of functions/heuristics.

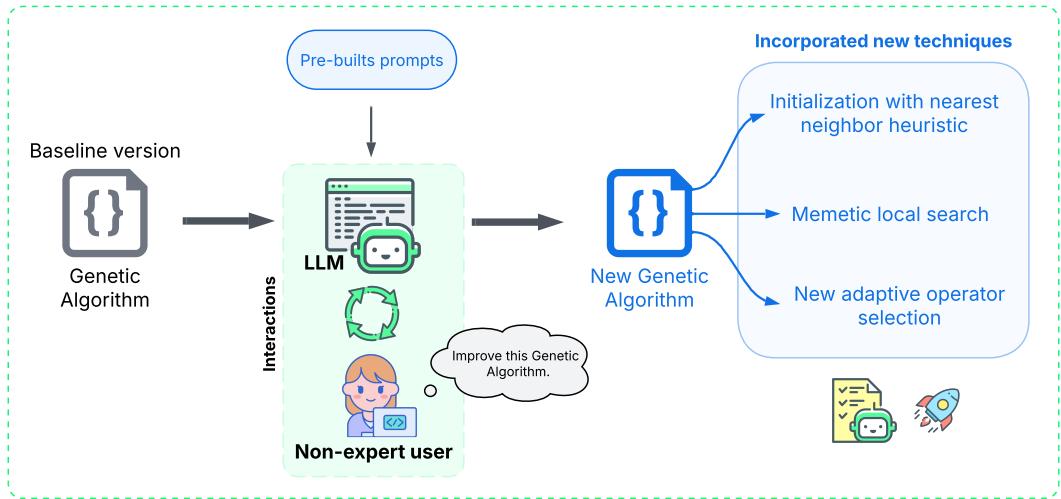


Figure 7.1: A non-expert user's interaction with an LLM can enhance an existing genetic algorithm by incorporating modern techniques.

phase. Beyond guiding the optimization process, LLMs excel at detecting patterns, identifying key features in problem instances, and refining search spaces. They have also shown to be able to generate new heuristics tailored to specific problems. Furthermore, LLMs offer valuable insights by explaining the results of optimization problems, making them versatile tools for both solving and interpreting complex tasks.

As in previous chapters of this thesis, we focus on a specific type of optimization problem: combinatorial problems. These problems have unique properties that set them apart: many valid solutions (including the existence of equivalent or similar solutions), decomposability (some problems can be broken down into smaller, more manageable subproblems), constraint handling (a set of rules that define valid solutions), and search space structure (the presence of multiple local optima, requiring well-chosen search strategies to avoid getting trapped in suboptimal solutions), among others.

As a result, researchers in this field must not only be knowledgeable about combinatorial optimization problems but also highly proficient in implementing optimization algorithms. Given the importance of computational efficiency, factors such as memory optimization, effective data structure management, minimizing unnecessary abstractions, and carefully selecting the right programming language play a crucial role in the design and development of these algorithms.

The rapid advancements in using LLMs as black-box collaborators for optimization demand a clear positioning of new methodologies. Table 7.1 delineates our work's unique contributions in this crowded landscape. Specifically, our approach is distinguished by:

- **Systematic Scope over Anecdotal Evidence.** Where prior work often provides proofs-of-concept on a limited set of algorithms, we deliver a systematic bench-

mark across ten algorithms spanning four distinct families. We prove that a straightforward, reproducible prompting strategy is robust enough to yield significant performance gains across this diverse set without compromising code stability.

- **Practitioner-Centric Design over Expert-Centric Frameworks.** Our methodology marks a fundamental departure from expert-centric systems that require designing complex evolutionary loops or prompt engineering strategies. It is uniquely designed for the practitioner—a user with programming skills but lacking deep theoretical expertise. Our work is the first to demonstrate a path for these users to upgrade complete, monolithic codebases, rather than just optimizing isolated functions. This holistic approach significantly lowers the barrier for applying state-of-the-art optimization techniques.

To introduce this novel research direction, the next two subsections present a classic combinatorial optimization problem along with ten traditional optimization algorithms, grouped into four distinct categories. Then, in the methodology section, Section 7.3, we demonstrate how LLMs can be leveraged to enhance these ten algorithms, improving both performance and efficiency while streamlining their implementations.

7.2.2 Problem Definition

The Travelling Salesman Problem (TSP) is one of the most iconic and extensively studied problems in combinatorial optimization, serving as a foundational pillar in both Artificial Intelligence and Operations Research. Its simplicity in definition belies its profound computational complexity, making it a benchmark for evaluating new algorithmic paradigms. Formally, let $C = \{c_1, c_2, \dots, c_n\}$ be a set of n cities, and let $D : C \times C \rightarrow \mathbb{R}_{\geq 0}$ be a function that assigns a non-negative distance between each pair of cities. This distance function can represent Euclidean distances, road distances, or any other relevant metric. The objective of the TSP is to find a permutation π of the indices $\{1, 2, \dots, n\}$ that minimizes the total travel distance of a closed tour, where each city is visited exactly once before returning to the starting city. This objective is formally defined as:

$$\min_{\pi \in S_n} \left\{ \sum_{i=1}^{n-1} D(c_{\pi(i)}, c_{\pi(i+1)}) + D(c_{\pi(n)}, c_{\pi(1)}) \right\},$$

where S_n denotes the set of all possible permutations of $\{1, 2, \dots, n\}$. The problem is known to be NP-hard, meaning that no polynomial-time algorithm is known to find the optimal solution for large instances, necessitating the use of heuristics and meta-heuristics.

For this study, we specifically selected the TSP due to the vast amount of high-quality implementations available online. This includes extensive public code repositories (e.g., GitHub), numerous textbooks, and a multitude of scientific articles detailing

various exact and approximate solution techniques. This widespread availability of code and conceptual knowledge strongly suggests that LLMs likely possess extensive pre-training knowledge regarding effective strategies and implementations for solving the TSP [132]. This makes it an ideal candidate for exploring LLMs' capabilities in code generation and optimization within a well-understood domain.

7.2.3 Traditional Optimization Algorithms for the TSP

Algorithms for solving the TSP encompass a broad variety of forms and approaches, reflecting decades of research in combinatorial optimization. For this study, we have carefully chosen ten distinct optimization algorithms, summarized in Table 7.2. These algorithms are specifically categorized into four fundamental groups: metaheuristic methods (which are stochastic, iterative optimization algorithms), reinforcement learning (representing policy-driven approaches), deterministic heuristics (simple, rule-based methods), and an exact algorithm (guaranteeing optimality). These four algorithm categories can be briefly characterized as follows:

1. **Metaheuristics** [22] are high-level problem-independent algorithmic frameworks that provide a set of guidelines or strategies to develop specific optimization algorithms. They are versatile optimization methods that use heuristic and stochastic principles to explore vast solution spaces. While they do not guarantee finding the global optimum, they are highly effective in efficiently finding high-quality, near-optimal solutions for complex, NP-hard problems within reasonable computational time. Examples include Genetic Algorithms, Simulated Annealing, and Ant Colony Optimization.
2. **Reinforcement Learning (RL)** [224] is a machine learning paradigm where an autonomous agent learns optimal decision-making by interacting with an environment. The agent performs actions, receives feedback in the form of rewards or penalties, and iteratively adjusts its policy to maximize cumulative rewards over time. In the context of combinatorial optimization, RL agents can learn to construct or improve solutions by making sequential decisions, often outperforming traditional heuristics by adapting to problem structures.
3. **Deterministic heuristics** [139] are among the most basic and computationally inexpensive algorithms for combinatorial optimization. They operate by following a fixed set of rules or a greedy strategy at each step to construct a solution from scratch. Unlike metaheuristics, they do not involve randomness or iterative improvement. They generally produce only one solution, making a myopic, deterministic decision at each step. While fast, they often get stuck in local optima and do not guarantee solution quality. Examples include Nearest Neighbor and Cheapest Insertion.
4. **Exact algorithms** [139] are designed to find the provably optimal solution for a given problem instance. They ensure optimality by exhaustively exploring the entire solution space, either explicitly or implicitly (e.g., through pruning tech-

Table 7.2: Overview of Selected Algorithms for Solving the Travelling Salesman Problem (TSP)

Algorithm	Characteristics	Application to TSP
Metaheuristic		
Ant Colony Optimization (ACO) [59]	Probabilistic, pheromone-based learning	Simulates ants' foraging behavior where solutions (routes) are constructed based on pheromone trails left by previous solutions.
Genetic Algorithm (GA) [167]	Population-based, crossover, mutation	Generates a population of routes and evolves them through selection, crossover, and mutation to find near-optimal solutions.
Adaptive Large Neighborhood Search (ALNS) [176]	Adaptive destruction and reconstruction	Iteratively destroys and reconstructs routes, dynamically adjusting strategies based on previous performance.
Tabu Search (TABU) [73]	Use of memory structures (tabu lists)	Iteratively modifies routes while keeping a list of features of previously visited solutions to prevent revisits.
Simulated Annealing (SA) [104]	Probabilistic, temperature-based search ^a	Iteratively refines a route by accepting worse solutions with a decreasing probability to escape local optima.
Reinforcement Learning		
Q_Learning [216]	Value-based learning, exploration-exploitation trade-off	Learns an optimal routing policy by iteratively updating action-value functions based on rewards from different paths.
SARSA [216]	On-policy learning, continuous updates	Uses an on-policy approach to learning optimal routing strategies based on real-time interactions with the environment.
Deterministic Heuristic		
Christofides [48]	Guarantees 1.5-optimality, MST-based	Constructs a minimum spanning tree, finds perfect matching, and combines them to form a tour.
Convex Hull [61]	Geometric approach	Starts with a convex hull and incrementally inserts remaining points in a way that minimizes travel distance.
Exact	Branch and Bound (BB) [148]	Explores all possible solutions while pruning suboptimal paths to guarantee optimality.

niques like branch-and-bound). While they guarantee the best possible solution, their computational complexity typically grows exponentially with problem size, making them impractical for large-scale instances of NP-hard problems like the TSP.

The deliberate choice of algorithms from these four diverse categories guarantees that our LLM-based improvement framework is tested across a broad spectrum of methodologies. This is crucial because, although all selected algorithms address the same problem (TSP), they vary fundamentally in their underlying principles, search strategies, and computational characteristics, providing a robust testbed for LLM capabilities.

7.2.4 Selected Implementations

To minimize the possibility of implementation errors in the ten algorithms for solving the TSP, and to ensure that these implementations are being utilized by the community, we employed `pyCombinatorial` [162].¹ This Python library, which includes a whole range of optimization algorithms for solving the TSP, has received over 100 stars on GitHub and was created by [Valdecy Pereira](#). Each implementation is based on a standard algorithm variant, providing us with an excellent testing environment for attempting to improve them using LLMs. In the next section, we detail our methodology for improving optimization algorithms.

7.3 Methodology

7.3.1 Enhancing Traditional Optimization Algorithms with Large Language Models

We introduce a methodology based on LLM interactions to generate enhanced versions of the 10 before-mentioned algorithms, building upon and extending the approach proposed by [183] (see Figure 7.1). This process can be replicated using the chatbot available at the project URL.²

Given the initial set $\{A_i \mid i = 1, \dots, 10\}$ of 10 original algorithm codes and an LLM, the algorithm improvement process can technically be stated as follows. First, a prompt P_i is generated based on our general prompt template T (shown in the second column of this page), the algorithm name N_i , the signature of the main function S_i , and the algorithm code A_i :

$$P_i = \text{PRODUCE_PROMPT}(T, N_i, S_i, A_i), \quad i = 1, \dots, 10 \quad (7.1)$$

Then, the generated prompt P_i is executed by the LLM given a set of hyperparameters

¹ <https://github.com/Valdecy/pyCombinatorial>

² <https://camilochs.github.io/comb-opt-for-all/>

θ , resulting in a changed/updated implementation A'_i :

$$A'_i = \text{LLM}_{\text{execute}}(P_i, \theta), \quad i = 1, \dots, 10$$

To ensure correctness, each A'_i undergoes a validation process (see below). If an output fails validation, the refinement process is repeated iteratively until a valid version is obtained.

Code Validation

The validation of A'_i may fail for two reasons:

1. **Execution errors:** These lead to immediate failures during code runtime.
2. **Logical inconsistencies:** The algorithm executes without errors but produces invalid TSP solutions.

In the first case, an error message e is generated, and the LLM refines the code based on this feedback:

$$A'_i = \text{LLM}_{\text{execute}}(A'_i, \theta, e)$$

For the second case, where the execution is error-free but the generated solutions are invalid,³ an explicit prompt requesting a correction is passed to the LLM, such as:

“The provided code generates invalid solutions; please verify and return a corrected version.”

The refinement loop proceeds until a valid code A'_i is obtained. This process is carried out through an interactive conversation with an LLM-based chatbot. To increase the chances of generating a valid code with each retry, we begin with a high-temperature setting, which is then progressively lowered in each iteration of the process.

In LLMs, *temperature* controls the randomness of the model’s output. A higher temperature (e.g., 1.0 or 2.0) makes the model’s responses more diverse and less predictable, while a lower temperature (e.g., 0.2) makes the output more deterministic and stable.

Table 7.3 shows the results of this procedure for the five selected LLMs.⁴ A green checkmark (✓) indicates that the first obtained A'_i was valid, while a red cross (✗) signifies that corrections were necessary due to either of the two code failures. A double red cross (✗✗) indicates that both failures occurred. Moreover, in the case of corrections, the number of necessary corrections is provided in a second column. Note that Table 7.3, below the LLM names, also indicates the initial temperature setting.

³ To validate whether a solution is invalid, one must use a function that checks its feasibility. For example, in the case of the TSP, a requirement could be that each city appears exactly once in the tour.

⁴ The reasoning behind selecting these five LLMs (rather than others) will be explained in Section 7.4.

Table 7.3: Analysis of the Code Generation Process

Algorithm	Claude-3.5-Sonnet		Gemini-exp-1206		Llama-3.3-70B		GPT-O1		DeepSeek-R1	
	(temp = 1.0)		(temp = 2.0)		(temp = 1.0)		(temp = 1.0)		(temp = 1.0)	
	Success		Success		Success		Success		Success	
	1st Try	# Attempts	1st Try	# Attempts	1st Try	# Attempts	1st Try	# Attempts	1st Try	# Attempts
ACO	✗	1	✓	-	✓	-	✓	-	✓	-
GA	✓	-	✗	1	✓	-	✗	3	✓	-
ALNS	✗	1	✓	-	✓	-	✓	-	✗	3
TABU	✓	-	✓	-	✓	-	✓	-	✓	-
SA	✓	-	✓	-	✓	-	✓	-	✓	-
Q_Learning	✗	-	✓	-	✓	-	✓	-	✓	-
SARSA	✗	-	✓	-	✓	-	✓	-	✓	-
Christofides	✗	-	✓	-	✓	-	✓	-	✓	-
Convex Hull	✗	1	✓	-	✓	-	✓	-	✓	-
Branch and Bound	✓	-	✗	4	✓	-	✓	-	✓	-

7.3.2 Prompt Design: A Focus on Simplicity and Accessibility

A central hypothesis of this study is that significant algorithmic improvements can be achieved without requiring users to possess deep expertise in prompt engineering. To test this, we developed a single, standardized prompt template. The design of this prompt is *intentionally basic*.

Our objective is not to engage in an exhaustive search for the “optimal” algorithm variant through elaborate prompting, which constitutes a separate research direction. Instead, our goal is to establish a reproducible baseline for what LLMs can achieve when prompted by a non-expert user. This approach directly aligns with our focus on accessibility and allows us to isolate the LLM’s intrinsic ability to enhance code.

Despite its simplicity, the prompt’s formulation provides clear, high-level directives. It focuses the LLM on two critical performance axes—(1) improving solution quality and (2) accelerating convergence—and explicitly encourages the integration of state-of-the-art techniques to achieve these goals. The template is shown below:

Prompt Template

```
You are an optimization algorithm expert.

I need to improve this {{algorithm
    name}} implementation for the travelling salesman problem (TSP
) by incorporating state-of-the-art techniques. Focus on:

1. Finding better quality solutions
2. Faster convergence time
```

Requirements:

- Keep the main function signature: {{the signature of an the main function}}
- Include detailed docstrings explaining:
 - * What improvement is implemented
 - * How it enhances performance
 - * Which state-of-the-art technique it is based on
- All explanations must be within docstrings, no additional text
- Check that there are no errors in the code

IMPORTANT:

- Return ONLY Python code
- Any explanation or discussion must be inside docstrings
- At the end, include a comment block listing unmodified functions from the original code

Current implementation:

```
{{algorithm code}}
```

The prompt template begins with “*You are an optimization algorithm expert*,” a technique known as “role prompting”, which has been empirically shown to guide LLMs toward a specific behavior or specialization [108, 11]. By setting a clear context from the outset, it enhances both the relevance and quality of the model’s response.

This approach aligns with “in-context learning” [58, 115, 187], combining an external context (the user-provided code in the prompt) with an internal one (the LLM’s knowledge of various techniques to improve the TSP algorithm).

An important insight: supplying a complete algorithm code within the prompt works like a ‘map,’ guiding the model on how to update the code. The LLM must preserve the overall structure, making modifications only in the relevant sections without breaking the logic. Without this external context (the provided algorithm), the model’s solution would likely be more constrained and less effective, as it would have to generate everything from scratch. In contrast, with an initial codebase, the model can focus on refining and improving specific areas rather than rebuilding the entire algorithm code from scratch. In other words, the provided code *influences* the update proposed by the LLM [58, 183].

As technically indicated already in Eq. 7.1, the prompt template receives three dynamic variables that are placed in the corresponding positions in the prompt template (enclosed within {{ ... }}):

- **Algorithm’s name:** steers the LLM toward a specific context.
- **Main function’s signature:** ensures the initial function’s input arguments, output values, and name remain unchanged, preventing unintended modifications

that could affect compatibility with the original code.

- **Algorithm code:** the optimization algorithm’s original implementation in Python.

Additionally, we explicitly instruct the LLM to report the modifications it makes (“*Include detailed docstrings explaining: ...*”). This step is essential, as it enables us—as shown in Section 7.4—to understand why a particular LLM’s code outperforms the original or another model’s code.

7.4 Experimental evaluation

In this section, we present experiments with the 10 previously mentioned optimization algorithm codes taken from `pyCombinatorial`. We describe our setup for utilizing LLMs, the parameter tuning of the stochastic algorithms, the TSP datasets and evaluation metrics employed, and the comparative analysis of the results. We highlight key details from the generation process and conclude with an analysis concerning code complexity.

7.4.1 Setup

LLMs Environment

We selected five leading code-generation LLMs: Anthropic Claude-3.5-Sonnet [202], Google Gemini-exp-1206 [204], Meta Llama-3.3-70b [206], OpenAI GPT-O1 [156], and DeepSeek-R1 [203]. For simplicity, we will refer to the models as Claude, Gemini, Llama, O1, and R1 throughout the remainder of this work. Note that these LLMs rank among the top models in the LiveBench benchmark [223], which is immune to both test set contamination and the biases of LLM-based and human crowdsourced evaluations (**as of February 2025**). Using the OpenRouter API, we executed identical prompts across all models, enabling straightforward model switching for transparent experimentation. This produced 50 new algorithm codes which, combined with the 10 original algorithm codes from the `pyCombinatorial` framework, gave us 60 Python files ready for evaluation.

Hardware Environment

All experiments, including parameter tuning, are conducted on a cluster equipped with Intel® Xeon® CPU 5670 processors (12 cores at 2.933 GHz) and 32 GB of RAM.

Parameter Tuning

While the deterministic heuristics and the branch and bound method are parameterless, the seven probabilistic approaches (five metaheuristics and two reinforcement learning algorithms) require careful parameter tuning to perform well. Consequently, we tuned all 42 stochastic algorithm codes: the 7 original versions and the 35 new

variants generated by the LLMs. To ensure a fair and robust comparison, we employed `irace` [129], a well-established automatic algorithm configuration tool. The tuning process was executed as parallel jobs on the SLURM cluster described in Section 7.4.1. For each algorithm execution during the tuning phase, the CPU time limit was set to the number of cities in the instance (in seconds). Table 7.4 details the parameter ranges considered for tuning each algorithm variant, as well as the final best configurations selected by `irace`.

Parameter tuning in stochastic algorithms with parameters is essential, as suboptimal configurations can lead to poor performance regardless of the algorithm's inherent quality. For instance, in ACO, we tune key parameters— m (number of ants), α , β , and ρ (pheromone decay)—for all six code variants (five LLM-generated ones plus the original) to ensure they operate under optimal conditions for the TSP problem. This guarantees a fair evaluation, as each code variant is assessed using its best possible configuration.

7.4.2 Benchmark Datasets and Evaluation Metrics

To evaluate all algorithm codes, we use problem instances from the well-known TSPLib library [172]. We select 10 instances from the available ones, ranging from a small instance with 99 cities to a large one with 1084 cities.⁵ This selection ensures a comparison across a diverse set of problem instance sizes.

As an evaluation metric, we used the objective function value of the best-found solution in all cases except for the Branch and Bound (BB) codes. This is because BB is an exact algorithm that, if given enough computation time, will always find an optimal solution. Therefore, we use runtime as the evaluation metric in the case of BB. Moreover, as the runtime of BB for the 10 selected problem instances is very high, we instead generate 10 random TSP instances with 10 to 15 cities for the evaluation of the BB codes. Interestingly, as we will see in the comparative analysis subsection, some LLM-generated versions of BB incorporate heuristic mechanisms during algorithm initialization, leading to significant improvements in runtime performance.

7.4.3 Experimental Design

The experiments were designed as follows:

- **Stochastic Algorithms.** Each of the metaheuristics and reinforcement learning codes is applied 30 times independently to each of the 10 problem instances. The output of each run is the best solution found. Performing 30 independent algorithm executions for each problem instance is a common practice in the optimization community to obtain a reliable estimate of the algorithm's performance.

⁵ TSPLib names of the selected TSP instances: `RAT99`, `BIER127`, `d198`, `A280`, `F417`, `AL1535`, `GR666`, `U724`, `PR1002`, and `VM1084`.

Table 7.4: Parameter values obtained by tuning with *irace*. Ranges show minimum/maximum values considered for tuning.

Algorithm	Parameter	Range	Original	Claude-3.5-Sonnet	Gemini-exp-1206	Llama-3.3-70B	GPT-O1	DeepSeek-R1
Metaheuristics								
ACO	m (ants)	(2, 20)	7	4	2	3	17	20
	α (alpha)	(1.0, 2.0)	1.34	1.72	1.67	1.46	1.72	1.22
	β (beta)	(1.0, 2.0)	1.59	1.24	1.98	1.97	1.93	1.55
	ρ (decay)	(0.01, 0.3)	0.24	0.12	0.24	0.29	0.06	0.05
GA	N (population size)	(5, 100)	97	14	97	84	55	58
	μ (mutation rate)	(0.01, 0.2)	0.02	0.04	0.16	0.16	0.01	0.13
	e (elite)	(1, 5)	4	2	5	2	3	5
ALNS	λ (removal fraction)	(0.05, 0.3)	0.27	0.05	0.26	0.29	0.22	0.29
	ρ (rho)	(0.01, 0.3)	0.27	0.25	0.04	0.2	0.27	0.02
TABU	T (tabu tenure)	(3, 30)	8	12	30	15	10	9
SA	T_0 (initial temperature)	(1, 50)	12	49	9	30	35	50
	T_f (final temperature)	(0.0001, 0.1)	0.0547	0.0464	0.074	0.056	0.0433	0.048
	α (cooling rate)	(0.8, 0.99)	0.9895	0.8732	0.8956	0.8131	0.9154	0.8777
Reinforcement Learning								
RL_QL	lr (learning rate)	(0.01, 0.5)	0.44	0.15	0.26	0.49	0.46	0.34
	df (decay factor)	(0.8, 0.99)	0.97	0.82	0.98	0.87	0.98	0.82
	ϵ (epsilon)	(0.01, 0.3)	0.09	0.28	0.03	0.24	0.21	0.13
	E (episodes)	(1000, 10000)	4266	1082	4906	2474	1294	1989
SARSA	lr (learning rate)	(0.01, 0.5)	0.04	0.36	0.49	0.19	0.41	0.29
	df (decay factor)	(0.8, 0.99)	0.86	0.91	0.80	0.88	0.83	0.87
	ϵ (epsilon)	(0.01, 0.3)	0.23	0.18	0.16	0.08	0.12	0.16
	E (episodes)	(100, 5000)	105	156	1850	137	124	1711

Moreover, the CPU time limit for each algorithm execution is set to the number of cities (in seconds) of the tackled problem instance. For example, the run-time limit for `RAT99` is 99 seconds. This method, which aligns execution time with the instance size, is a common practice for comparing algorithms that solve the TSP.

- **Deterministic Heuristics.** Christofides and Convex Hull, since they are deterministic heuristics, always yield the same result for a given problem instance. Therefore, all corresponding codes are executed exactly once per instance.

- **Branch and Bound.** As previously mentioned, since Branch and Bound is an exact algorithm, the focus is on runtime rather than solution quality. All Branch and Bound codes are applied 30 times to each of the small problem instances specifically generated for the evaluation of the Branch and Bound codes.

7.4.4 Comparative Analysis with Original Algorithm Codes

The comparative analysis between the LLM-updated algorithm codes and the original algorithm codes (referred to as 'original' from now on) is studied in the following in a separate way depending on the type of optimization algorithm.

Metaheuristics

The results of all metaheuristic codes are shown by means of boxplots in Figure 7.2. Note that the y-axes are shown in a logarithmic scale.

1. **ACO:** Apart from the first problem instance (`RAT99`) where the LLM-generated codes of Gemini, O1, and R1 perform similarly to the original code, in all other problem instances the LLM-generated codes of the mentioned three LLMs outperform the original code with statistical significance. It also appears that the LLM-generated codes of O1 and R1 are somewhat more robust than the original code, which can be seen in smaller boxes. In contrast, the code generated by Claude, apart from the first problem instance, performs always worse than the original code. Finally, the Llama-generated code generally performs similarly to the original code, with the exception of the first problem instance.
2. **GA:** The original code exhibits very low robustness for the first two problem instances, which can be seen by the large boxes. Generally, most codes (except for R1) are quickly trapped in local optima which they cannot escape. The R1-generated code clearly outperforms all others, including the original.
3. **ALNS:** Generally, the best-performing codes are those by Claude and Gemini, with a slight advantage for Gemini in the last two instances. Another noteworthy aspect is the low robustness of the O1-generated code in this case.
4. **TABU:** Like in the case of GA, also the TABU code generated by R1 significantly outperforms the remaining codes. Only the O1-generated code can compete for the smallest two problem instances. This suggests that R1 excels at generating efficient optimization algorithms.
5. **SA:** The Claude-generated codes clearly show the weakest performance here. In contrast, the R1-generated code again outperforms the remaining ones.

Reinforcement Learning (RL)

In Figure 7.3, the RL codes generated by the LLMs are compared with the original RL codes. The displayed results differ from the metaheuristics case presented before in two key aspects. First, some LLM-generated codes fail to produce a result for all problem

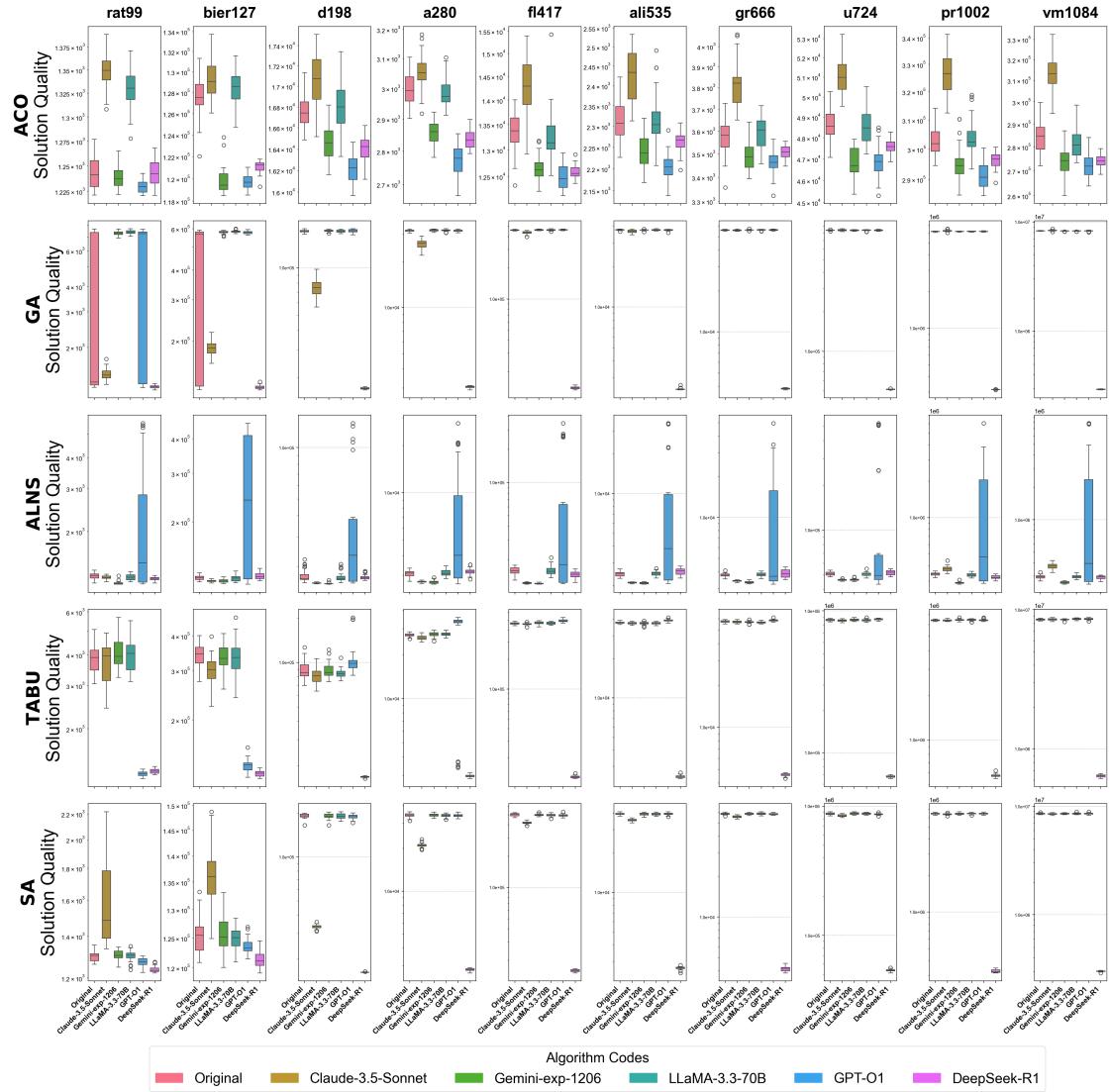


Figure 7.2: Comparison of the metaheuristic codes generated by the five LLMs with the original codes. Remember that the TSP is a minimization problem, that is, the lower the values, the better. The y-axes are shown in a logarithmic scale.

instances within the time limit. These cases are marked as N/A. Second, especially in the case of Q_Learning the original code performs more competitively. We analyze these aspects below:

6. **Q_learning:** The original code is actually the best-performing one in this case. The Llama-generated code is the only one that achieves a nearly comparable performance—an unexpected result given Llama’s poor performance in the case of the metaheuristics. In addition, the Claude-generated code outperforms the original one on the RAT99 and A280 instances.
7. **SARSA:** The general picture here aligns more with that observed in the case of the metaheuristics. The original code struggles (in comparison to some LLM-generated codes) as instance sizes grow larger. The R1-generated code fails to produce results for the largest three instances. The only code maintaining a stable

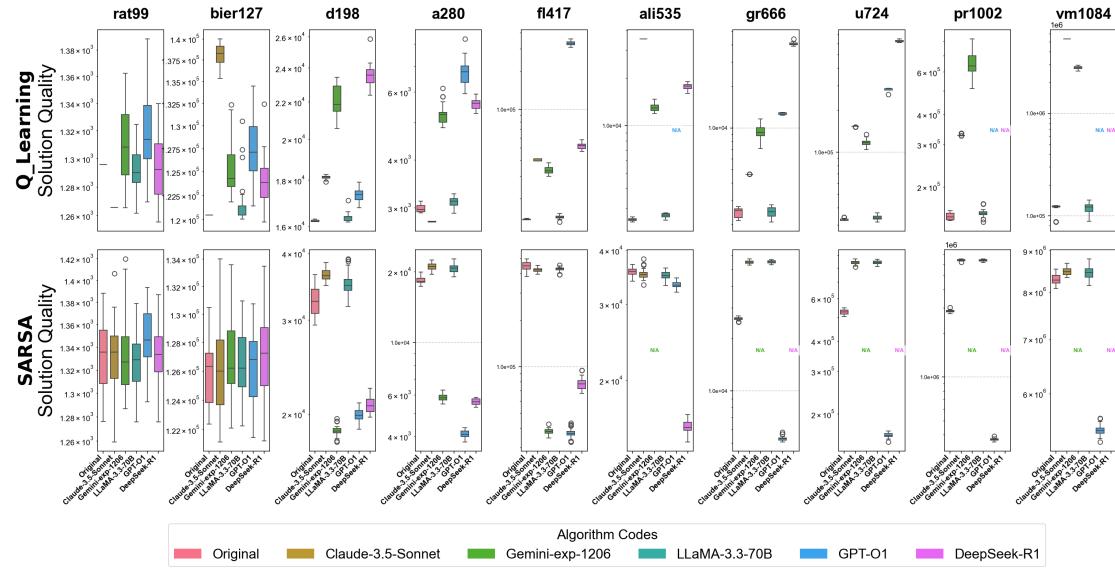


Figure 7.3: Comparison of the reinforcement learning (RL) codes generated by the five LLMs with the original codes. Remember that the TSP is a minimization problem, that is, the lower the values, the better. The y-axes is shown in logarithmic scale.

and strong performance across all 10 instances is the O1-generated one.

Deterministic/Heuristic

Remember that, as the chosen heuristics are deterministic, there is no need to analyze the distribution of their results over multiple runs. Therefore, in Figure 7.4, we simply compare the GAP of the results produced by the LLM-generated codes (in percent) relative to the results of the original codes. A positive value indicates that the respective LLM-generated code outperforms the original, while a negative value suggests the opposite.

8. **Christofides** (see Figure 7.4 (a)): While the Llama-generated code produces very similar results to the original code over the whole instance range, the Gemini-generated code is (apart from instance d198) always inferior. Moreover, its relative performance decreases as instance size grows. Concerning O1 and R1, it can be stated that the performance of their codes is slightly inferior to the one of the original code for rather small problem instances. However, with growing instance size, they clearly outperform the original code.
9. **Convex Hull** (see Figure 7.4 (b)): In contrast to Christofides, the Convex Hull codes generated by O1 and R1 perform rather poorly. In fact, the best code for Convex Hull is the one generated by Gemini. This code has slight disadvantages for smaller instances but increasingly outperforms the original code with growing instance size. The code generated by Claude shows the opposite pattern. While it outperforms the original code for smaller problem instances, its performance strongly decreases with growing instance size.

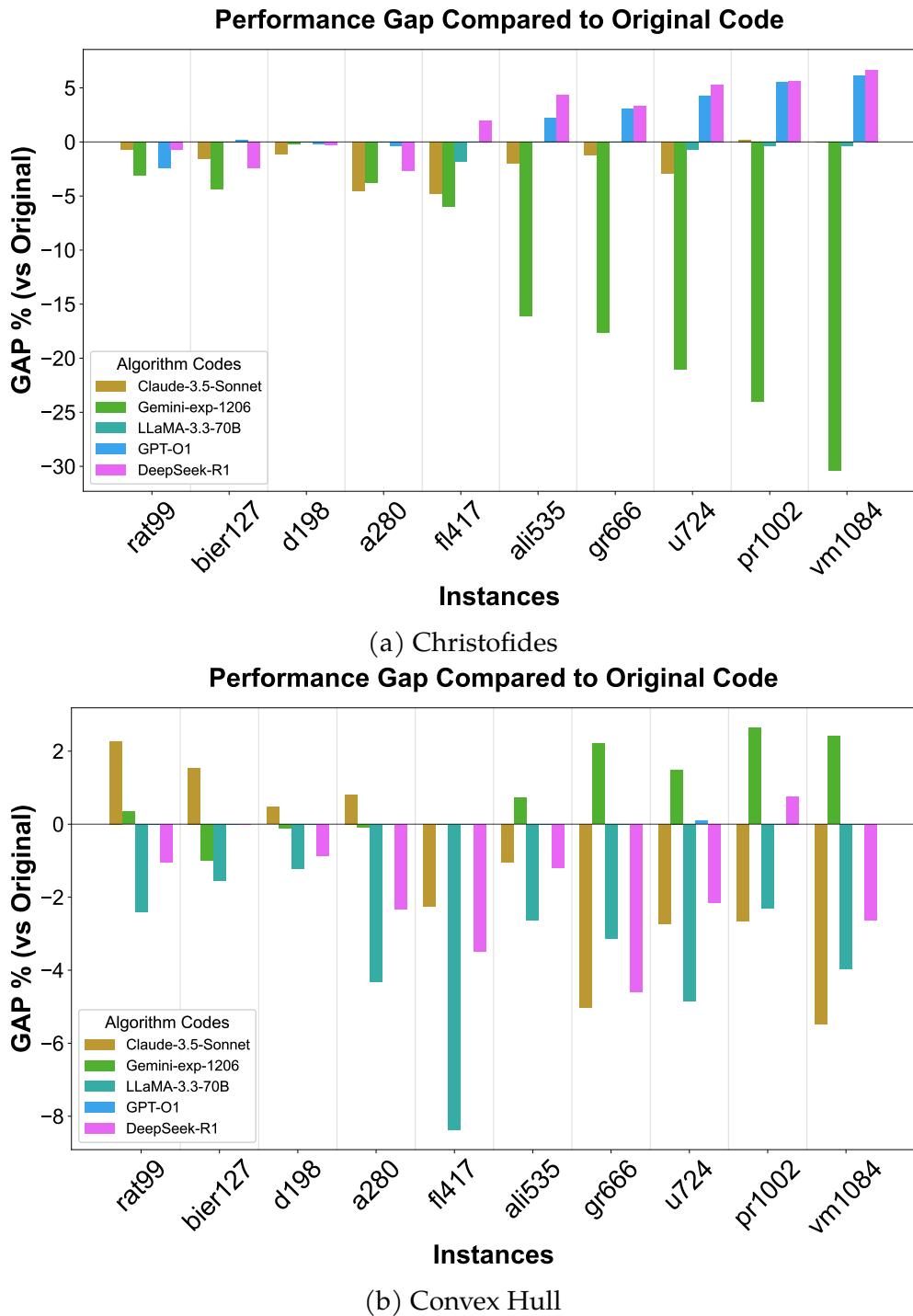


Figure 7.4: Comparison of the deterministic heuristic codes generated by the five LLMs with the original codes. The bar plots show the performance gaps (in percent) relative to the original codes. Note that a positive value indicates that the LLM-generated code produces a better solution.

Exact Approach

As already mentioned in Section 7.4.3 (Experimental Design), in the context of the exact BB method, the comparison is based on computation time.

10. **Branch and Bound:** Figure 7.5 shows that the codes generated by O1 and R1 outperform both the original code. Notably, R1—an open-weight LLM—achieves the best performance, surpassing all proprietary models. In contrast to O1 and R1, the other LLM-generated codes perform worse than the original code.

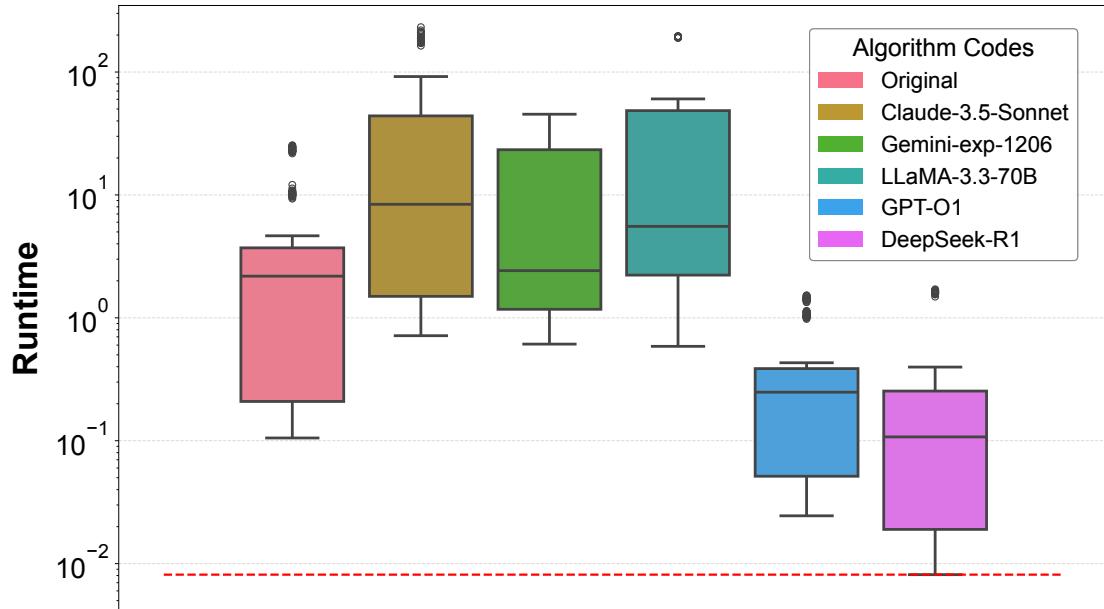


Figure 7.5: Comparison of the BB codes generated by the five chosen LLMs with the original BB code (in terms of computation time). Each code was applied 30 times, and the y-axis is shown in a logarithmic scale.

Summary: A notable conclusion is that LLMs can produce improved versions of baseline algorithms, resulting in performance improvements without necessitating specialized expertise in each algorithm. In the following subsection, we showcase examples of code improvements achieved, for example, by integrating more sophisticated algorithmic components.

7.4.5 Key Insights in Code Generation

Next, we explore why certain LLM-generated (or LLM-updated) codes outperform the original ones. Our focus was to understand if this was due to optimized data structures, for example, or due to adding different algorithmic components. In particular, we analyze four cases to address these questions on the basis of the LLM-generated codes.

Case 1: GA (R1-generated code)

The R1-generated version of GA features the following improvements, as stated by the model itself by means of a docstring in the code, as requested in the prompt.

```
Improvements:
1. Hybrid initialization with nearest neighbor heuristic
2. Rank-based fitness + tournament selection
3. Adaptive operator selection (OX, ER, BCR)
4. Memetic local search with stochastic 2-opt
5. Diversity preservation mechanisms
```

In particular, R1-generated GA is the only LLM-generated code that introduces a modification to the population initialization by incorporating the nearest neighbor heuristic. In contrast, both the original code and all other LLM-generated variants use the following initialization function:

```
# Function: Initial Population
def initial_population(population_size, distance_matrix):
    population = []
    for i in range(0, population_size):
        seed = seed_function(distance_matrix)
        population.append(seed)
    return population
```

Instead, R1-generated GA features the following initialization that leads to an improved performance.⁶

```
1 def initial_population(population_size, distance_matrix):
2     """Initialize population with mix of random and heuristic solutions. Combines
        diversity (random) with quality (NN) for better exploration. Implements hybrid
        population initialization from modern metaheuristics."""
3     population = []
4     if population_size >= 5: # Include 20% NN seeds
5         for _ in range(max(1, population_size//5)):
6             population.append(nearest_neighbor_seed(distance_matrix))
7
8     ...
9     return population
10 def nearest_neighbor_seed(distance_matrix):
11     """Generate initial solution using Nearest Neighbor heuristic. Provides high-
        quality initial seeds to accelerate convergence. Based on constructive heuristic
        methods commonly used in TSP."""
12     ...
```

In particular, the GA is initialized with 20% nearest neighbor solutions for population sizes of at least five individuals. This well-known TSP heuristic significantly speeds up convergence. In this way, R1 shows its ability to ‘dig’ into its knowledge base to choose an alternative population initialization method and implement it effectively.

Case 2: SA (R1-generated code)

Also in the case of SA, R1 identifies and utilizes two well-known mechanisms recognized for their efficiency in solving the TSP: (1) the Lundy-Mees adaptive cool-

⁶ Note that, in all Python code snippets shown in this work, ‘...’ indicates omitted parts that are not relevant.

ing schedule for improved temperature control, introduced years after the original SA [131], and (2) the nearest neighbor heuristic for TSP. In the latter case, R1 integrates the nearest neighbor heuristic for the TSP in a way similar to what it did in the case of the GA, demonstrating a consistent pattern in leveraging effective initialization strategies.

Case 3: SARSA (O1-generated code)

The O1-generated SARSA code achieved the best results among the competitors. This is due to being the only code to make use of *Boltzmann Exploration* (see code below). Unfortunately, LLMs do not have the capacity to identify the exact source (book, scientific article, etc) from which the information about Boltzmann Exploration was extracted. However, after reviewing the code, it is likely that it was sourced from a 2017 paper (see [9]), which suggests a Boltzmann operator for SARSA applied to the TSP.

In fact, the code below shows that, unlike the original code, O1 not only applies a random operator to select the next unvisited city but also assigns a probability—derived from the `q_table` data structure—to this choice (line 8), making the selection more dynamic. Moreover, it avoids unnecessary abstractions (e.g., extra data structures) that could slow down the Python code.

```

1 ...
2 while len(visited) < num_cities:
3     unvisited = [city for city in range(num_cities) if city not in visited]
4     # Boltzmann exploration
5     q_values = q_table[current_city, unvisited]
6     exp_q = np.exp(q_values / temperature)
7     probabilities = exp_q / np.sum(exp_q)
8     next_city = np.random.choice(unvisited, p=probabilities)
9     reward = -distance_matrix_normalized[current_city, next_city]
10    visited.add(next_city)
11    route.append(next_city)
12    ...
13 ...

```

Case 4: BB (R1-generated code)

When studying why the BB code of R1 was faster than the original code, first we noticed that, like in cases 7.4.5 and 7.4.5, R1 made use of the nearest neighbor heuristic for initialization. Moreover, R1 modified the `explore_path` function of BB by dynamically sorting the next candidates by edge weight to prioritize the cheapest/nearest extensions first. Both updates are not trivial. R1 notes the following in the code comments: “*Enhancements reduce unnecessary branching and accelerate convergence through early solution bias.*”⁷

In addition, the code snippet below is not present in the original code. O1 introduces `current_node` and `candidates` efficiently, using slicing (lines 4 and 5) and sorting with a lambda function (line 6) to enhance path exploration in BB. This new array

⁷ This can be seen in line 48 of file `bb_deepseek_r1.py` of our online repository URL.

based data structure is both efficient and implemented in a Pythonic style to improve performance.

```

1 def explore_path(route, distance, distance_matrix, bound, weight, level, path, visited,
2     min1_list, min2_list):
3     ...
4     current_node = path[level - 1]
5     candidates = [i for i in range(distance_matrix.shape[0])
6                     if distance_matrix[current_node, i] > 0 and not visited[i]]
7     candidates = sorted(candidates, key=lambda x: distance_matrix[current_node, x])
8     ...

```

7.4.6 Code complexity

In the previous subsection, we analyzed the LLM-generated codes in terms of their performance. But do these codes also offer better readability and reduced complexity in comparison to the original codes? To address this question, we evaluate their *cyclomatic complexity*—a metric that quantifies the number of independent paths through a program’s source code. Through empirical research, Chen [40] demonstrated that a high cyclomatic complexity correlates with an increased bug prevalence. For our measurements, we employ the RADON library for Python.⁸

As shown in Table 7.5, the Claude-generated codes have the lowest average cyclomatic complexity (5.60 points), which improves code readability but, as shown before, comes at the cost of performance. The other models’ codes and the original code have complexity scores between 6.84 (O1) and 7.51 (R1), which is still considered low and well-structured according to standard software engineering metrics. The values in the Risk Category column are taken from the documentation of the RADON library.⁹

Finally, Figure 7.6 reveals that there are cases—such as the R1-generates codes in the case of GA and Christofides, or the O1-generated code for SARSA—in which the LLM-generated codes not only outperform the original codes, but also decrease the cyclomatic complexity.

Table 7.5: Average Cyclomatic Complexity of the codes

Algorithm Codes	Average Complexity	Risk Category
Original	6.95	B (Low - Well structured)
LLMs	Claude-3.5-Sonnet	A (Low - Simple)
	Gemini-exp-1206	B (Low - Well structured)
	Llama-3.3-70b	B (Low - Well structured)
	GPT-O1	B (Low - Well structured)
	DeepSeek-R1	B (Low - Well structured)

⁸ <https://pypi.org/project/radon/>.

⁹ <https://radon.readthedocs.io/en/latest/commandline.html#the-cc-command>

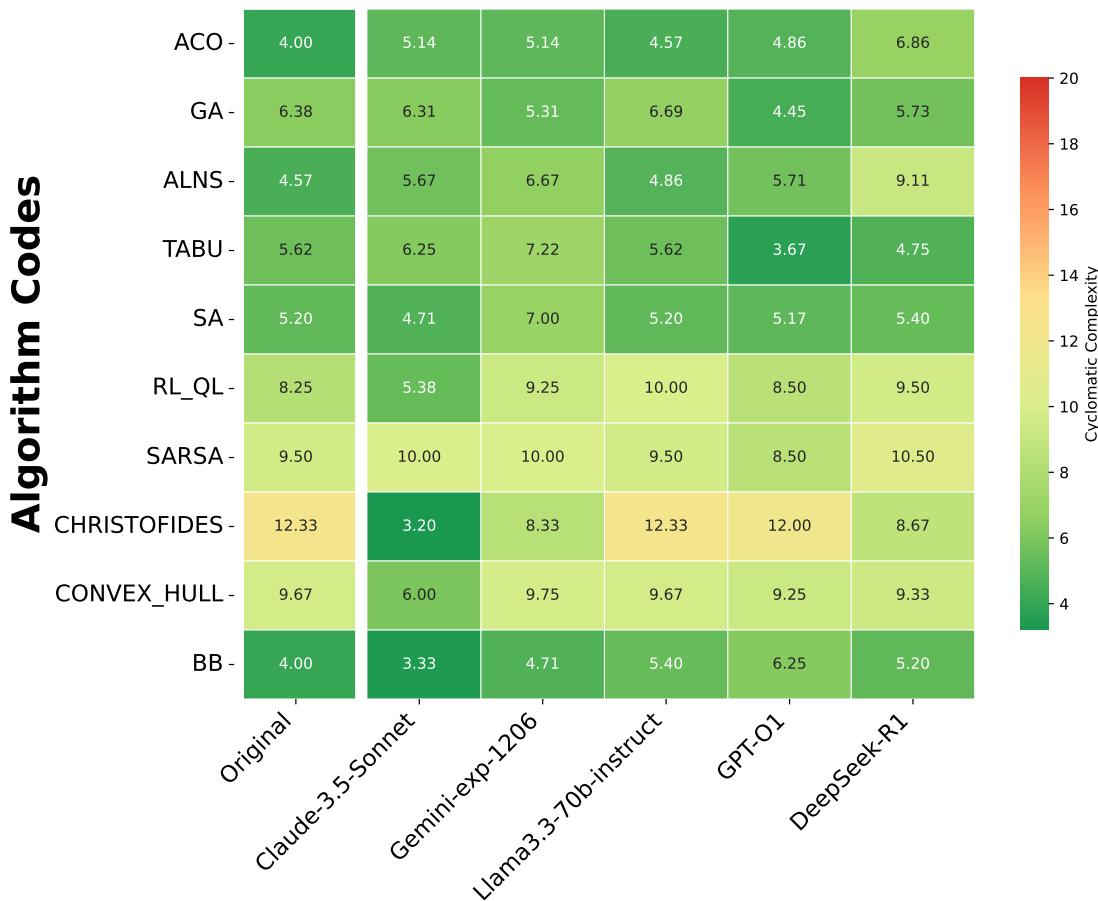


Figure 7.6: Cyclomatic Complexity

Summary: Based on all evaluations presented in this research, we can state that among the five LLMs tested, R1 generally delivered the best results, followed by O1. Gemini performed well in certain cases—such as ACO and ALNS—but underperformed in others, notably in Christofides. Among all tested models, Claude exhibited the weakest performance.

In summary, *LLM-enhanced code versions clearly outperformed the original implementations in nine out of ten cases/algorithms*. Only for Q_Learning were none of the models able to improve the original code. In that case, Llama matched the performance of the original implementation.

7.5 Discussion

Recent research has convincingly demonstrated that LLMs can be powerful tools for creating and enhancing complex optimization algorithms [183, 152, 194, 236, 175, 123]. To situate our current work within this rapidly evolving field and clearly delineate its unique contributions, we provide a comprehensive comparative analysis of these state-of-the-art approaches in Table 7.1. Building upon our own initial proof-of-concept [183], this work significantly expands the investigation by conducting a

large-scale, systematic study across 10 classical algorithms from four diverse families. Our findings confirm that the principle of LLM-driven improvement is not an isolated phenomenon but a broadly applicable technique for the Travelling Salesman Problem. However, our analysis also reveals several critical limitations and methodological considerations that warrant a deeper discussion.

7.5.1 Limitations and Methodological Considerations

While our results are promising, a nuanced understanding of the challenges is crucial for the practical application and future development of this approach.

- **The “Black Box” Nature and Its Risks.** A primary challenge, highlighted in [103], is that LLMs cannot precisely trace the sources of their suggestions. This opacity makes it difficult to pinpoint which specific modifications led to performance gains. More importantly, it introduces the risk of the LLM generating “hallucinated” algorithmic components or incorrectly combining concepts from different sources, potentially leading to subtle logical flaws that are not immediately apparent through basic testing. Exploring the question of “*where specifically does the generated code come from?*” is therefore not just a fascinating research avenue but a critical step towards building more reliable systems.
- **The Practical Costs of Iterative Refinement.** Our “simple prompt” strategy successfully lowers the barrier to entry, but the overall process is not without cost. As shown in Table 7.3, several algorithms required multiple manual correction rounds to become functional. This iterative cycle of generating, testing, and providing feedback demands significant user time and effort. This reveals a key trade-off: the ease of initial prompting versus the manual cost of validation. For this methodology to be truly effective in practice, the efficiency of this human-in-the-loop refinement process must be considered.
- **Performance Variability and Failure Analysis.** Our results reveal significant performance variability among LLMs and highlight specific failure cases, as promised in our introduction. For instance, the general failure of all tested LLMs to improve upon the Q-learning baseline (Figure 7.3) suggests that current models may struggle with algorithms whose performance is highly sensitive to specific state-space representations or reward structures. Similarly, the weaker performance of some models on complex algorithms like ALNS may point to a scarcity of high-quality, specialized training data for such intricate heuristics. Understanding these failure modes is essential for defining the boundaries of where LLM-based improvement is most effective. Conversely, it is particularly notable that improved performance does not always correlate with increased code complexity. In cases such as the GA code generated by R1, our analysis shows that the LLM simultaneously enhanced the solution quality while decreasing the cyclomatic complexity (Figure 7.6), suggesting that these models can also act as effective code refactorizers—a significant benefit beyond pure algorithmic enhance-

ment.

- **Positioning within the State-of-the-Art.** As detailed in Table 7.1, our work is methodologically distinct from other recent approaches. While prominent frameworks like LLaMEA/ReEvo, AlphaEvolve, and FunSearch employ complex, automated systems—such as evolutionary loops or program search trees—to generate or discover *new* algorithmic components, our approach uses a simple, interactive prompting strategy. This reflects a fundamental difference in objective and user role: expert-centric frameworks focus on automated algorithm discovery or generation, requiring users to design a sophisticated search process. In contrast, our practitioner-centric method centers on the collaborative enhancement of *complete, existing codebases*. Furthermore, while the work [183] established the feasibility of this concept in a single case study, the present research validates its effectiveness systematically across a broad and diverse range of algorithms.

7.5.2 Directions for Future Research

The limitations identified above naturally lead to several exciting directions for future research.

- **Broadening the Problem Scope.** A crucial next step, addressing a limitation of our current study, is to validate these findings beyond the TSP. Applying this methodology to combinatorial optimization problems with different structures, such as the Vehicle Routing Problem (VRP) or the Knapsack Problem, is essential to determine the true generality of this approach.
- **Automating the Refinement Loop.** To mitigate the manual effort of validation and correction, future work should focus on automating the feedback cycle. This could involve developing systems where an LLM agent can not only generate code but also autonomously execute it against a benchmark suite, analyze the results (including errors and performance metrics), and iteratively refine its own suggestions.
- **Specialized Models and Benchmarks.** As LLMs evolve, it will be necessary to continuously test their capabilities in a structured way. The creation of specialized benchmarks with baseline implementations for multiple optimization problems would be invaluable. Furthermore, developing fine-tuned LLMs specialized in optimization could overcome the data scarcity issues observed for complex or less-common algorithms, potentially leading to even more significant and reliable improvements. This aligns with trends towards “reasoning models” that prioritize response quality over speed [192].

7.6 Conclusion

This benchmark study demonstrates that Large Language Models (LLMs) can serve as effective collaborators for enhancing classical optimization algorithms. Through a

simple and reproducible prompting strategy, we improved ten baseline algorithms for the Travelling Salesman Problem—frequently achieving better solution quality, faster execution times, and reduced code complexity. Notably, these improvements were applied holistically across full codebases, often through the integration of modern heuristics and more efficient data structures.

Our methodology is fully reproducible and accessible via the chatbot interface on our project website (<https://camilochs.github.io/comb-opt-for-all/>), offering a practical tool for practitioners—especially those without deep theoretical expertise—to upgrade complex algorithms.

Looking ahead, we aim to extend this approach to a broader range of combinatorial optimization problems to assess its generalizability. Additionally, to address the cost of manual prompting and refinement, future work will explore the use of LLM-driven agents for automating and iteratively improving existing algorithmic implementations.

Note

At the time of writing this thesis (June 2025), this work and the previous one (Chapter 6) are still under peer review, following initial rejections. In the note at the end of Chapter 5, I pointed out that this shift in direction might raise doubts about the practical value of such an integration. The main criticism has centered on the simplicity of the approach, something I consider not a weakness, but rather a strength. It is evident that if I had developed a sophisticated framework for generating optimization algorithms using LLMs—something that only a few could reproduce—the reception might have been more favorable. Yet not all scientific contributions must be complex to be valuable. It always depends on the problem at hand, and in this case, I believe the goal was simply to demonstrate how LLMs can enhance optimization algorithms in a straightforward way. Perhaps simple enough that someone might think: “Isn’t this too easy?”

Part II

Visualization Tools

8

Introduction

In recent years, visualization has emerged as a powerful and increasingly necessary tool in the analysis and development of optimization algorithms, particularly meta-heuristics [71, 22]. As these algorithms often operate as black boxes, understanding their behavior beyond numerical performance metrics remains a significant challenge. Visualization techniques aim to bridge this gap by offering intuitive, interpretable representations of algorithm dynamics across the search space.

A notable contribution in this direction is the concept of Search Trajectory Networks (STNs), a method designed to trace and visualize the behavior of optimization algorithms on specific problem instances, whether combinatorial or continuous [154, 155]. The key strength of STNs lies in their ability to reveal how the structure of a problem instance *shapes* the search behavior of an algorithm, thereby enabling researchers to gain insights into why certain methods succeed or fail. This information is critical for refining algorithm design, identifying structural biases, and understanding performance bottlenecks.

STNs go beyond traditional performance plots by representing the sequence of explored solutions as a directed graph, where—in the most basic form—nodes correspond to unique solutions and edges represent transitions driven by the algorithm. This graphical abstraction allows researchers to analyze patterns of convergence, stagnation, or exploration across the landscape, revealing subtleties not easily detected or interpreted through numerical analysis alone.

Building upon this idea, **our work introduced STNWeb** [33] (see Chapters 9 and 10), a web-based platform that automates the generation and analysis of STNs. In addition to facilitating visualization, STNWeb includes features such as automatic textual descriptions of the generated plots, making interpretation accessible even to non-expert users. This enhances its pedagogical value and supports rigorous comparative analysis, enabling users to systematically demonstrate, for example, the superiority of a newly proposed algorithm over baselines in structurally diverse scenarios.

Thus, visualization tools like STNWeb are not merely illustrative; they function as analytical instruments that complement statistical evaluation and support deeper

methodological insight. They represent a shift toward explainability and interpretability in optimization, areas that are increasingly vital as algorithmic complexity and problem diversity grow.

8.1 LLMs for Automated Analysis in Optimization Tools

A parallel and significant development involves integrating Large Language Models (LLMs) to enhance the explainability of optimization algorithm results. While prior work in this thesis focused on LLMs improving metaheuristic solution quality, a trend emerged in 2024 to leverage LLMs for better understanding and analyzing algorithm behavior.

A prime example is our paper "**Large Language Models for the Automated Analysis of Optimization Algorithms**" [35] (see Chapter 11), published in GECCO 2024. This work integrates an LLM into STNWeb [33], a web-based tool utilizing Search Trajectory Networks (STNs) for metaheuristic analysis. The application addresses a critical question: how to determine the superiority of one stochastic/heuristic optimization algorithm over another for a given problem instance.

STNWeb visualizes algorithm behavior as a graph, but its interpretation requires specialized knowledge. To overcome this, we implemented an LLM-powered interpretation layer. Lacking widely available powerful multimodal LLMs at the time, we devised the following solution: the system programmatically extracts key visual features from the graph to construct a detailed text prompt. The LLM then interprets this prompt to generate a natural language explanation. Consequently, users receive not just a complex image, but a clear analysis of the data—e.g., why one algorithm outperforms another. The LLM acts as a virtual expert, verbalizing insights from the STNWeb visualization. This paper stimulated subsequent work incorporating LLMs as analysis systems to facilitate diverse optimization tools.

Since 2024, LLM integration with AI explainability within metaheuristics has primarily focused on genetic programming (GP) or other evolutionary techniques. Representative works include:

- In March 2024, [Maddigan et al. \[135\]](#) introduced a tool to improve the interpretability of GP-based non-linear dimensionality reduction. Their web-based interface, GP4NLDR, combines GP with an LLM-powered chatbot to provide users with a coherent understanding and explanation of dimensionality within a genetic process.
- The more recent work, "Code Evolution Graphs" [196], adds an interpretability layer to code generation within an evolutionary framework. The authors, also behind LLaMEA [194], identified this opportunity to incorporate interpretability into their framework (though broadly applicable). They showed that LLMs tend to produce increasingly complex code with repeated prompts, which can sometimes degrade algorithmic performance. Their approach enables traceability of

each LLM response, allowing for detailed analysis of generated code quality over time.

8.2 Future Directions: Leveraging LVLMs for Enhanced STNWeb Analysis

Currently, STNWeb’s LLM-based interpretation depends on converting visual features into textual prompts. However, the advent of advanced Large Vision-Language Models (LVLMs) presents exciting new opportunities. As we investigated in our work on VisGraphVar (see Chapter 13), LVLMs are increasingly capable of performing direct visual-spatial reasoning on graph images. This points to a promising research direction for STNWeb: incorporating LVLMs to analyze STN visualizations directly. Such models could interpret complex visual patterns—like overlapping trajectories, distinct cluster structures, or the distribution of optimal solutions—and produce detailed natural language explanations without relying on explicit programmatic feature extraction. For example, an LVLM might observe:

Example

“In the lower-right section, Algorithm_A’s trajectories cluster tightly, indicating entrapment in a suboptimal area, while Algorithm_B’s paths spread widely, reflecting stronger exploration and superior outcomes.”

This kind of direct visual comprehension would greatly deepen STNWeb’s analytical capabilities and make it more accessible, steering it toward an intuitive and intelligent visualization platform.

9

Search Trajectory Networks Meet the Web

Foundational Work for This Chapter

This chapter is based on the following publication:

- **Title:** *Search Trajectory Networks Meet the Web: A Web Application for the Visual Comparison of Optimization Algorithms*
- **Published in:** International Conference on Software and Computer Applications (ICSCA)
- **Type:** Conference Paper
- **Year:** 2023
- **Main contribution:** Details the integration process of STNs within a web-based architecture
- **Problem addressed:** Facilitate the use of STNs
- **Type of contribution:** Tool/software
- **DOI:** <https://doi.org/10.1145/3587828.3587843>
- **Current number of citations in Google Scholar:** 6

9.1 Introduction

This work initiated the second research line of my doctoral project. My involvement in developing the tool presented in this chapter originated from a suggestion by my supervisor: to enhance the accessibility of Search Trajectory Networks (STNs). As a software engineer before becoming a researcher, I viewed this challenge not as overly complex, but as a practical opportunity to contribute to the research community and build confidence in my thesis. My goal was to bring STNs to the web, leading to the creation of STNWeb.

In this chapter, we analyze the integration of STNs into a web-based environment. It is, in many ways, more of an engineering effort than a scientific one, but one that is likely to be more useful to scientists than to engineers.

A popular adage states that *a picture is worth a thousand words*. While not universally true, this certainly applies to the visual tool for comparing and understanding optimization algorithm behavior, which is the subject of this work. For decades, the standard approach to comparing optimization algorithms has relied on numerical result tables and classical data visualizations (e.g., line, bar, and scatter plots). However, to properly understand the behavior of complex optimization algorithms, such as metaheuristics [71, 22]—which are typically non-deterministic due to their stochastic components—researchers have increasingly emphasized the use of tools that enhance interpretability, with a particular focus on visual analytics [155].

Early techniques for visualizing search algorithm behavior were presented in [49, 166, 144, 130]. These approaches utilized dimensionality reduction to map search spaces into two or three dimensions, thereby tracking search progress. The latest advancement, *Search Trajectory Networks (STNs)*, was introduced in [154, 155]. STNs share a similar aim with earlier methods, but fundamentally differ by representing search trajectories as graph objects with nodes and edges that can be analyzed and visualized, rather than reducing the entire search space to a Cartesian plane. Specifically, STNs graphically display trajectories of different runs of optimization algorithms applied to the same problem instance as a directed network, relying heavily on graph-based visualization. The authors of [155] provided a set of scripts written in R, a free software environment for statistical computing and graphics.

Nevertheless, the original STNs tool, in its R script form, presented several inconveniences, most notably its difficulty of use. Generating the final graphics required multiple manual steps involving the execution of a series of R scripts. In this work, we alleviate this problem—among other improvements—by developing a web application that automates this entire process, significantly enhancing usability and accessibility.

9.2 Background: Search Trajectory Networks

Understanding and comparing stochastic optimization algorithms, like metaheuristics, is complex and cannot be fully achieved through numerical result tables alone. To address this, some co-authors of this research introduced Search Trajectory Networks (STNs) [154, 155], a methodology for graphically comparing algorithm trajectories. R scripts for generating STN graphics are available at <https://github.com/gabro8a/STNs>.

Figure 9.1 shows STN examples comparing three algorithms applied to a p-median problem instance, displaying 10 runs per algorithm. Network vertices correspond to either single solutions (Figure 9.1a) or defined solution subsets (Figure 9.1b).

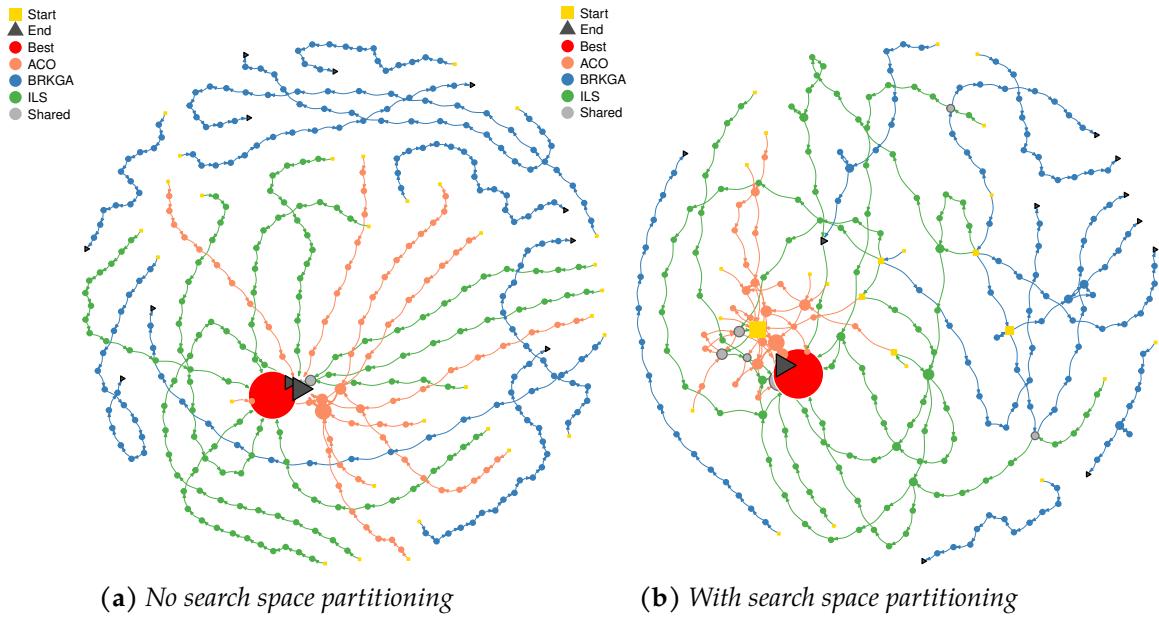


Figure 9.1: Examples of STN graphics comparing three different algorithms applied 10 times to the same problem instance.

- Algorithm trajectories are color-coded (e.g., ACO: orange, BRKGA: blue, ILS: green).
- Trajectory starting points are yellow squares.
- Endpoints are dark-grey triangles (not best solution) or red circles (best solution).
- Light-grey vertices indicate trajectories from at least two different algorithms passing through.
- Vertex size indicates the number of trajectories passing through it (larger size = more trajectories).

The difference between Figure 9.1a and Figure 9.1b lies in search space partitioning. Figure 9.1a displays STNs without partitioning, where vertices are individual solutions. Figure 9.1b shows the same 30 trajectories with partitioning, simplifying STNs by grouping solutions into search states. Advantages of partitioning are evident:

- Vertex count reduction, useful for cluttered STNs.
- Clearer identification of overlaps (multiple trajectories passing through the same vertex/solution set). For instance, ACO trajectories show minimal overlap without partitioning (Figure 9.1a) but significant overlap after partitioning (Figure 9.1b), indicating convergence to the same search area.
- Lack of overlaps within an algorithm's trajectories (even after partitioning) indicates poor robustness, as each run explores a different search space area (e.g., BRKGA in Figure 9.1).
- Overlaps only within the same trajectory (after partitioning) suggest the algorithm struggles to escape low-quality local optima.

STN graphics in Figure 9.1 use the Fruchterman-Reingold [69] layout from R's

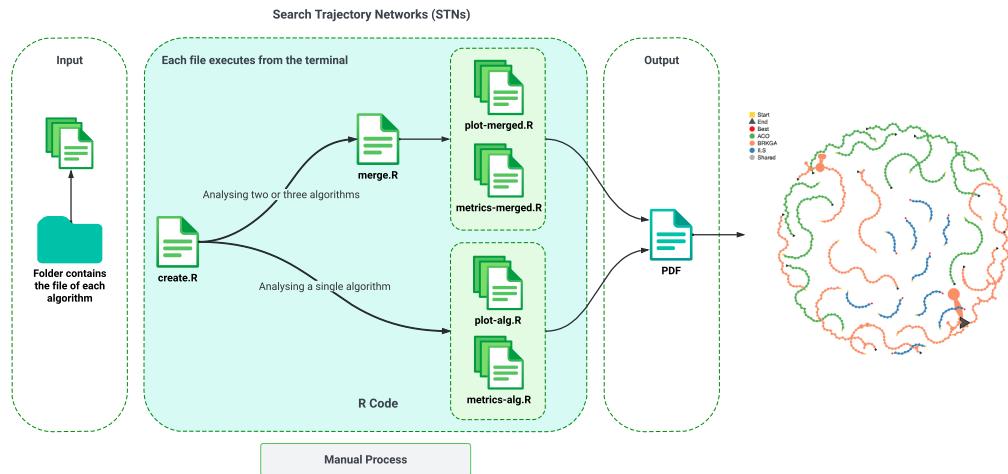


Figure 9.2: The graphic shows the workflow of the original STNs tool. It is divided into three phases (from left to right): (a) a folder must be created, containing a result file for each algorithm to be included in the comparison; (b) then two different R scripts must be executed in a given order, and depending on how many algorithms are included in the comparison (i.e., one versus multiple); (c) finally, the corresponding STN graphics are generated and provided in terms of a PDF file.

igraph package [51]; the tool also offers Kamada-Kawai [100] layout, with suitability depending on STN structure.

In the following, we outline the limitations of the original R scripts (<https://github.com/gabro8a/STNs>) before describing our developed web application.

9.2.1 Limitations

The current STNs tool based on R scripts has both limitations in terms of its usability and in terms of features. Here we describe three relevant issues.

Hard to Use

There are three main difficulties concerning the usability of the R scripts:

1. The workflow for producing STNs involves the iterative use of several R scripts (see Figure 9.2). It assumes that for generating the graphics for displaying the behaviour of one or multiple algorithms, the user must execute (from a terminal) a series of files in a given order. As a consequence, this process is error-prone and, at the same time, rather slow.
2. Given the need to execute R scripts, the user is supposed to have the same version of all packages installed in his local environment. Hence, if the user has a different operating system or a different R version, unintentional errors may occur due to having different prerequisites than those needed to use the current STNs tool.
3. If we look at Figure 9.3, we can see that the input file format is verbose and contains redundant information. For example, the FITNESS2 and SOLUTION2 columns

The diagram illustrates the transformation of an input data format. At the top, a table represents the 'Old Input Data Format'. It has columns for 'Run', 'Fitness1', 'Solution1', 'Fitness2', and 'Solution2'. The first row shows 'Run' 1 with 'Fitness1' 5 and 'Solution1' 0101100011. The second row shows 'Run' 1 with 'Fitness1' 6 and 'Solution1' 0101101011. The third row shows 'Run' 1 with 'Fitness1' 6 and 'Solution1' 0101101011. The fourth row shows 'Run' 1 with 'Fitness1' 7 and 'Solution1' 1101101011. The fifth row is an ellipsis. The 'Fitness2' and 'Solution2' columns are also present but are not highlighted. A pink arrow labeled 'Redundant Information' points to the second and third rows, which are identical. Below this, a large pink arrow labeled 'New Input Data Format' points down to a simplified table. This new table has columns for 'Run', 'Fitness', and 'Solution'. It contains the same data as the first four rows of the old table, but the 'Fitness2' and 'Solution2' columns are omitted. The fifth row is an ellipsis.

Run	Fitness1	Solution1	Fitness2	Solution2
1	5	0101100011	6	0101101011
1	6	0101101011	7	1101101011
...

Run	Fitness	Solution
1	5	0101100011
1	6	0101101011
1	7	1101101011
...

Figure 9.3: At the top is the old input format containing redundant information. The new and simpler version (as implemented in our web application) is below.

are unnecessary because they are repeated under FITNESS1 and SOLUTION1 in the subsequent row.

Limitation to maximally three algorithms

The R scripts of the original STNs tool do not allow comparing more than three algorithms. That is, the scripts do not accept more than three input files. Naturally, this presents a limitation as it is not uncommon in the area of metaheuristics to compare more than three algorithms. Therefore, this is neither scalable nor flexible.

Search space partitioning is not integrated

Search space partitioning is a central feature for taking profit from the STNs tool (see the previous section). However, the current R scripts do not perform this process automatically. Instead, this important feature is only described in [154] and users of the current STNs tool have to implement search space partitioning themselves in order to produce the corresponding input data files for producing the STN graphics. This makes it a challenging endeavour to take full profit of the advantages offered by STNs for the understanding of stochastic optimization algorithm behaviour.

9.3 Integration Into the Web

The section is divided into two parts. First, we will present the architecture of the new STNs web application. Then, we will describe the features that the web application incorporates.

One of the strong points of a web application is that the user does not have to install all the necessary prerequisites in his local system. The user only requires a mod-

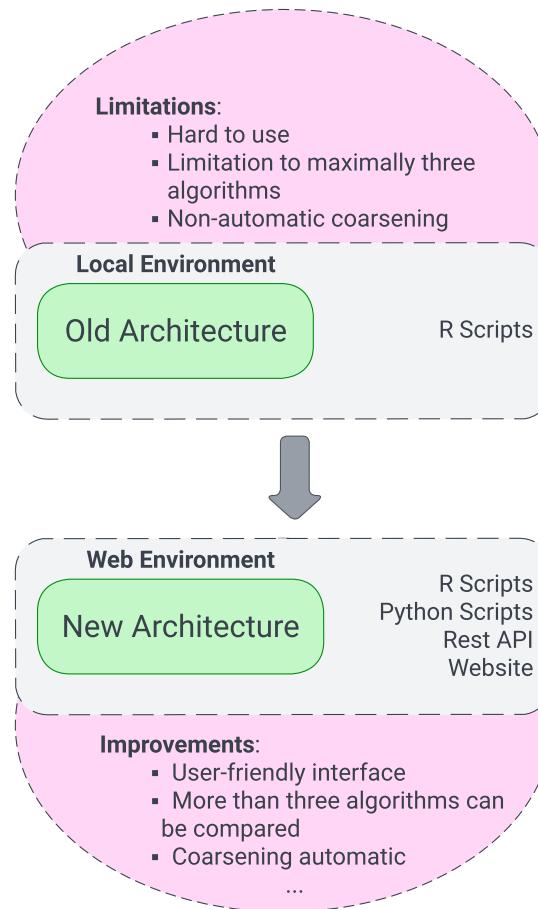


Figure 9.4: The architecture of the original STNs tool, consisting of R scripts, along with its limitations is displayed in the upper part. At the bottom, the new STNs architecture is characterized, consisting additionally of Python scripts, a Rest API, and a website along with additional improvements.

ern web browser and the URL of the web application in order to get started. In other words, a web environment furnishes interoperability and easy maintenance, even for small applications [213]. In order to achieve this in the case of the original STNs tool, it is required to change the system architecture in order to bridge the R scripts with a web application. This is mandatory, not only for new features but also for additional changes in the future. Henceforth, we refer to our new web application as *Search Trajectory Networks on the Web (STNWeb)*.¹

9.3.1 New system architecture

The new architecture incorporates Python scripts, a Rest API, and a website. The first two components are part of the backend, while the website is part of the frontend. Figure 9.4 shows the old architecture with a synthesis of its limitations and the new architecture with its improvements. All components of the new architecture are outlined

¹ The URL of the web application can be found in the following repository: <https://github.com/gabro8a/STNs>

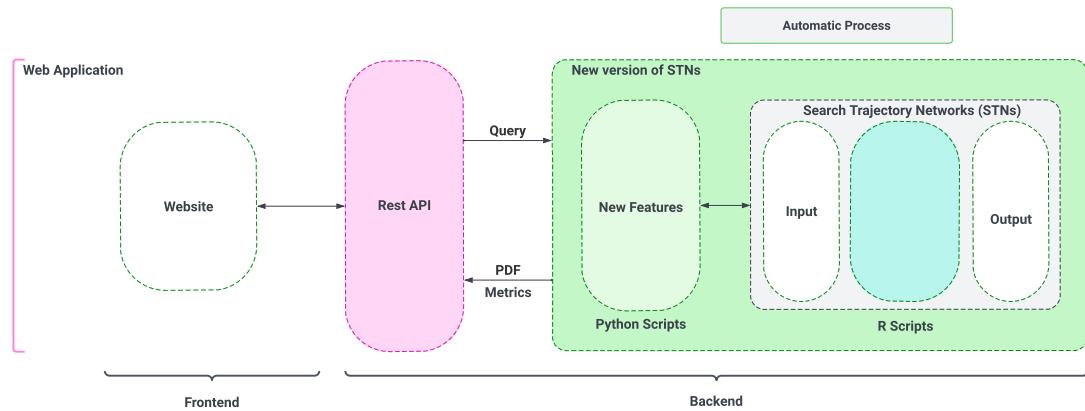


Figure 9.5: The web application architecture incorporates a Rest API and new features implemented in Python to the original STNs tool. Unlike the original STNs tool, this version requires just one query to the API in order to execute the three phases of the formerly manual process: creating a folder with the input files, executing the corresponding R scripts, and generating a PDF or metrics file.

in more detail in the following.

Backend: Rest API and new features implemented in Python

We chose Python for the implementation of the new features because of its rich ecosystem of libraries [163]. In particular, this choice allowed the use of the same programming language (Python) to create the Rest API and for implementing the new source code, for example, for automatic search space partitioning. We used Flask to implement the Rest API, a micro web framework for Python [77]. See also Figure 9.5, which shows that the new features are separated from the original STNs R scripts and a Rest API manages the automation of the whole process.

The Rest API opens a TCP/IP socket with a predefined port, which will receive requests from users. Moreover, it can handle multiple requests in parallel, as each new request generates a unique hash from a different user. Thus, it avoids undesired collisions, and later, the Rest API will return the data stream in PDF format, which will materialize in an embedded form on the website. Another advantage of using a Rest API is that it reduces application coupling by separating the user requests from the STNs source code.

Next we present more details about the endpoints of Rest API, both of which are POST requests:

- /stn - The parameters appended to this endpoint are those shown on the left-hand side of Figure 9.6. The invocation of this request occurs when clicking on the GENERATE button. In particular, parameters include the type of optimization problem (maximization vs. minimization), the value of the best-known solution for the considered problem instance (if known), the number of algorithms runs

to be used from the input data,² the relative size of the vertices in STN graphic, the layout of the STN graphic (normal layout vs. tree layout), the percentage of search space partitioning, and the algorithm names, colours and input data files.

- `/metrics` - The hash generated from the previous endpoint becomes the parameter of this endpoint. This implies that this endpoint can only be reached after the `/stn` endpoint is executed. The hash allows finding and returning the metrics file created before by clicking on the GENERATE button.

Frontend: Website

We used Angular 10 to build the web platform and Bootstrap 5 for styling. The app is designed to have all elements on one page with a bunch of essential HTML elements: a form, input types (e.g., text, button, range, colour, checkbox, and data file), select, and an embedded PDF viewer which is a component of Angular 5+ named `ng2-pdf-viewer`. In short, the website enables two operations after clicking on the GENERATE button: downloading the embedded PDF file, and downloading a stylesheet file in `.xls` format which contains the metrics of the displayed STN. All of this is indicated in Figure 9.6.

9.3.2 New Features

The new functionalities offered by STNWeb in comparison to the original STNs tool are described in the following.

Updated Data Input Format

As mentioned in the previous section and as shown in Figure 9.3, the input data format has slightly changed by removing redundant information. In particular, while the original STNs tool required an arc-based data format which contained two solutions (the origin and destination) at each row, the STNWeb tool simply requires a sequence of solutions (together with their fitness values) as input. This had the advantage that the files to be uploaded are significantly smaller, which is especially beneficial when the internet connection of the user is not that powerful.

Multiple data type formats

The application accepts not only `.txt` files, but also `.csv` files. The most important point is that `tar` compressed files are accepted. This is very beneficial for those situations in which solutions are very large (that is, consist of values for a large number of variables). Once the Rest API receives such a compressed file, it will decompress it and dispatch it to the STN tool process.

² In case this information is not provided by the user, all possible runs—that is, algorithm executions—from the input files are utilized.

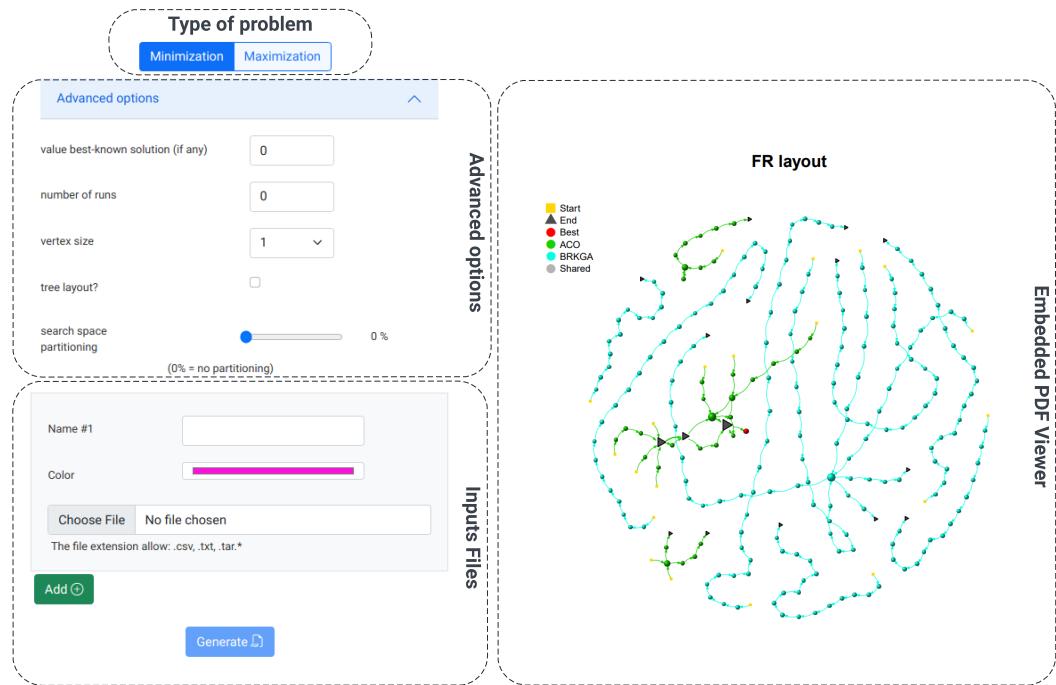


Figure 9.6: The layout of the web application in the web browser has three main parts. In the upper part on the left the user can specify the type of optimization problem (maximization vs. minimization) and a number of advanced options (see text). At the bottom left, the user can add one algorithm after the other (clicking the ADD button). For each algorithm, a name, colour and the input data file must be provided. Finally, the right-hand side includes an embedded PDF viewer that shows the STN visualization once the user has clicked on the GENERATE button.

A User-friendly Interface

The implemented interface allows the user to customize the generation of the visualizations. This concerns, for example, the possibility for the user to choose the trajectories' colour and the nodes' size. In the original version of the STNs tool, these issues were only possible by modifying the R scripts directly. Moreover, all the additional information that is required for the generation of the STNs can be provided by the user via the website: type of problem (maximization vs. minimization), the value of the best-known solution, the number of algorithm runs to be used, the type of the STN layout, and the percentage of search space partitioning, are included in the STNWeb.

In addition, a **HELP** button opens a pop-up window with instructions for the use and the application of the tool in case of doubts.

Possible Comparison of More Than Three Algorithms

The R scripts from the original STNs tool were modified in order to accept more than three algorithms for the comparisons. The strategy used for this was to apply reflection. This is a metaprogramming technique that allows changing a program's structure without the need to add extra code [120]. Thus, this gave flexibility to the scripts mentioned above.

Automatized Search Space Partitioning

As mentioned before, search space partitioning to display STNs in a partitioned search space in order to improve their interpretability was completely left to the user in the original STNs tool. This is, the user had to implement the code for transforming the original algorithm trajectories into trajectories of a partitioned search space. Differently to the original STNs tool, our new STNWeb tool implements this function in Python. Moreover, the web interface allows the user to provide a value between 0% (deactivated) to 100% of search space partitioning through a slider. The higher this percentage the higher the degree of search space partitioning.

9.4 Case Studies

In this section, we offer some case studies, ranging from a rather simple one to a more complex one, in order to show the utility of the use of this tool.

The case studies discussed below use a force-directed layout algorithm—in particular, Fruchterman–Reingold [69]—to visualize the network in the integrated PDF viewer. Note that this is an algorithm based on physical simulations. In particular, it does not rely on any prior assumptions about the structure of the networks. The STNWeb application provides us with different options. We may require a rather simple analysis of a single algorithm, for example. On the other side, STNWeb also allows a much more complex analysis of multiple runs of multiple algorithms.

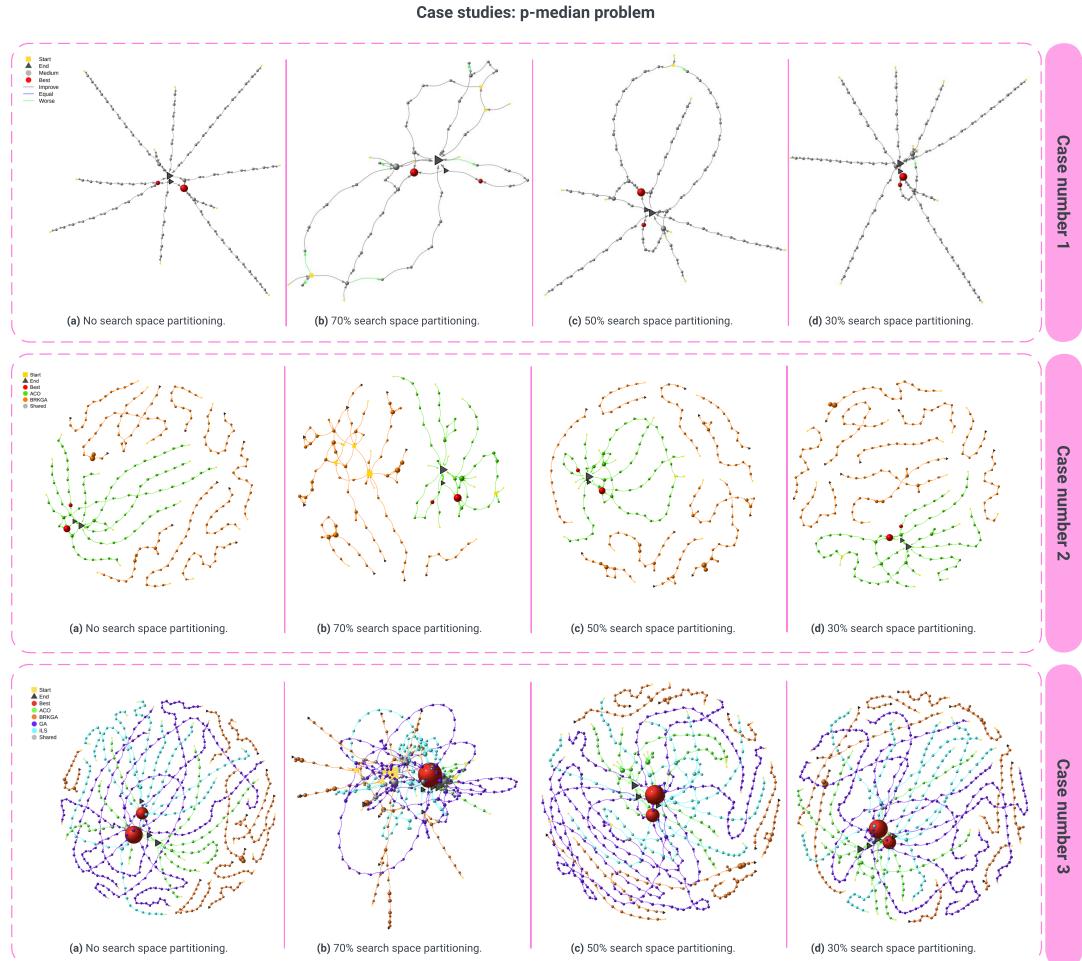


Figure 9.7: This graphic presents three case studies with different percentages of search space partitioning generated by STNWeb. First, a single algorithm when applied 10 times to the same problem instance is analyzed in the upper row. In the middle row, we can see the behaviour of two algorithms by means of different levels of search space partitioning. Finally, the new STNWeb feature was tested with four algorithms (the original STN tool did not support more than the three algorithms), which is graphically shown in the bottom row.

The three cases presented hereafter make use of three levels of search space partitioning: 70%, 50%, and 30%, in addition to the default case which does not make use of search space partitioning. All test cases are from an example application known as the *p-median* problem. The first (simple) case uses only the ACO algorithm; the BRKGA algorithm is added to ACO in the second case; and finally, ACO, BRKGA, ILS, and a standard genetic algorithm (called GA) are compared in the last case.

9.4.1 Case 1: A Simple Study

In STNWeb, when a single algorithm is analysed, the resulting STN graphic(s) essentially shows(s) a representation of the algorithms' behaviour. In the upper row of graphics in Figure 9.7 (case 1) only the ACO algorithm is analyzed. Notice that, in comparison to the other case studies that deal with the comparison of two or more algorithms, the colour of the arcs has a specific meaning in the case of a single algorithm comparison. In particular, grey arcs indicate transitions to better search states (solutions, or subsets of solutions), while blue arcs show transitions to equally good search states and green arcs show transitions to search states worse than the current one. In contrast, when multiple algorithms are compared, the colour of the arcs indicates the algorithm which has produced the corresponding trajectory.

1. Figure 9.7 (a) - upper row. This graphic was produced without search space partitioning, that is, nodes correspond to solutions. There are several interesting things to be observed. First, all algorithm trajectories seem to be attracted by the same area of the search space. Second, there are two best solutions (red circles) than are identified by the algorithm. In particular, two trajectories end up in the smaller red circle (more to the left), and three trajectories end up in the larger red circle (more on the right). All the trajectories keep improving (gray arrows). This indicates that the algorithm does not easily get trapped in a local optimum. Finally, there are not many overlaps between the algorithms' trajectories, at least not during the early stages of the search process.
2. Figure 9.7 (b-d) - upper row. In the others cases, different percentages of search space partitioning are utilized. The first aspect that deserves being mentioned is that the number of trajectory overlaps increases with an increasing percentage of search space partitioning. This indicates that, even though the different algorithm runs take different routes to the best-found solutions, they still pass through similar areas of the search space. Green arcs can be explained by the fact that nodes now represent subsets of solutions (instead of single solutions). Therefore, better solutions are joined with worse solutions, causing green arcs to appear.

9.4.2 Case 2: Comparison of Two Algorithms

In contrast to the previous case, in this case we compare two algorithms: ACO (green arcs) and BRKGA (orange arcs). This example is very illustrative for the usefulness of

STNWeb because the STN graphics allow immediately to confirm that ACO is, in this specific case, a much better algorithm than BRKGA. The best solutions (red circles) are only found by ACO trajectories. This indicates that ACO can discover areas of better quality without falling into local optima. In fact, the overlaps within the same trajectory of BRKGA (Figure 9.7 (b) - middle row) indicate that the algorithm tends to get stuck in certain areas of the search space.

9.4.3 Case 3: Complex Analysis

Apart from the automatization and the ease of use, one of the main advantages of STNWeb over its original non-web version is that it can analyze more than three algorithms. Moreover, search space partitioning is an essential aspect for the comparison of rather many algorithms.

1. Figure 9.7 (a) - lower row. The first thing that can be observed is that BRKGA cannot compete with the other algorithms. In addition, it can be seen that there is a shared solution (grey circle besides the large red circle) between ACO and GA. Although ACO, GA, and ILS find best solutions for this problem (the two red circles), there is no clear winner between ACO, GA, and ILS as their paths that do not find best solutions end up close (black triangles) to the best solutions.
2. Figure 9.7 (c-d) - lower row. When the search space partition is equal to or less than 50% (cases c-d) the resulting STNs are not so different from the STN produced without search space partitioning. Nevertheless, one interesting aspect is that the trajectories of GA, for example, neither become shorter nor do they show overlaps with a search space partitioning of 30% and 50%. This means the path taken by GA to find good solutions is, every time, rather different. Moreover, the algorithm tends to make rather large steps in the search space. On the other side, the trajectories of ACO clearly become shorter and show overlaps, for example, at a search space partitioning of 50%. This is because ACO algorithms include a strong heuristic bias which makes them start their trajectories in the same area of the search space. GA, on the other side, may start at any point of the search space.
3. Figure 9.7 (b) - lower row. This graphic shows that a search space partitioning of 30% is clearly too much for this case. In fact, there are so many trajectories overlaps now that the graphic becomes very hard to be analysed. However, it is interesting to see that there are three rather large grey circles (and some smaller ones). These large grey circles indicate that there are common attractors in the search space for several algorithms. However, all algorithms that are attracted by these regions are able to leave them in favour of regions containing even better solutions.

9.5 Conclusion

In this work, we introduced a new web-based implementation of the Search Trajectory Networks (STNs) tool. The application enables the generation of informative visualizations to support the comparison of stochastic optimization algorithms.

Our web implementation offers several key advantages over the original STNs tool. First, it automates previously manual processes, increasing usability and reducing the risk of human error. Second, it provides greater customization options—for example, in the choice of colors used to distinguish algorithms—allowing users to tailor visualizations to their needs. Third, and importantly, it supports the comparison of more than three algorithms, extending the tool’s applicability.

Future improvements include transitioning from 2D to 3D rendering and exploring alternative search space partitioning strategies. In particular, using diverse partitioning schemes may reveal new visual patterns that enrich the analysis and deepen our understanding of algorithmic behavior.

Note

From this work, I especially remember my trip to Kuantan, Malaysia. We presented it at a relatively “unknown” conference, but one that gave me the opportunity to discover a different world. I crossed the Kuantan jungle and witnessed a marching battalion of ants, led by the giant forest ant of Asia (*Dinomyrmex*), much to the surprise of my thesis supervisor—perhaps because he had spent years working on Ant Colony Optimization algorithms. All around us, the singular sounds of birds and other creatures formed a strange and perfect orchestra in that curious place. There were also monkeys near the beach, watching us closely, seemingly ready to steal something at the first chance.

It was, in short, a once-in-a-lifetime experience.

This research would become the first of three works related to STNWeb and marked the beginning of the second research line in my dissertation. A line that left me quite satisfied—though not completely (see the final note of Chapter 12).

10

STNWeb: A new visualization tool for analyzing optimization algorithms

Foundational Work for This Chapter

This chapter is based on the following publication:

- **Title:** *Search Trajectory Networks Meet the Web: A Web Application for the Visual Comparison of Optimization Algorithms*
- **Published in:** Software Impacts
- **Type:** Journal Paper
- **Year:** 2023
- **Main contribution:** Provide a web-accessible version of the Search Trajectory Networks (STNs)
- **Problem addressed:** Lower the barriers to adoption of STNs
- **Type of contribution:** Tool/software
- **DOI:** <https://doi.org/10.1016/j.simpa.2023.100558>
- **Current number of citations in Google Scholar:** 13

10.1 Introduction

Unlike the previous chapter, where STNWeb was presented from a more technical, software engineering perspective, this shorter chapter offers a concise overview of STNWeb's functionalities rather than its implementation details. All of my work on STNWeb originated as a suggestion from my thesis supervisor, following my earlier efforts with GNNs and metaheuristics (Chapters 3 and 4). He suspected I wasn't particularly enthusiastic about that research direction (he was right! LLMs hadn't yet entered my life).

As mentioned before, STNWeb was, overall, more of an engineering effort than a

scientific one—perhaps with the exception of the next chapter. I have always been good at building tools. This was important because it gave me the confidence to pursue new publications and, more importantly, to feel that the PhD had meaning for me. Without purpose, I would rather do nothing.

Visual representations significantly enhance our comprehension of complex digital information across Computer Science and AI. However, the combinatorial optimization community has historically lagged in developing visual tools, despite a growing need for new methods to compare optimization algorithms. For decades, algorithm comparison relied on numerical data tables and classical charts (e.g., line, bar, scatter plots), often complemented by statistical analysis. Yet, a consensus has emerged: truly understanding stochastic optimization algorithms, like metaheuristics [71, 22], requires additional, user-friendly graphical tools.

While visual tools for optimization algorithms have been scarce, some attempts include [49, 166, 144, 130], which used dimensionality reduction for rudimentary search progress tracking. The most advanced tool, *Search Trajectory Networks* (STNs) [154, 155], was introduced more recently. STNs utilize directed graph objects to visualize and analyze search progress, displaying multiple trajectories from various optimization algorithms applied to the same problem instance. This graph-based visualization approach comes with R scripts provided by the authors at <https://github.com/gabro8a/STNs>.

However, the original STNs tool’s R script implementation presented significant usability challenges, primarily due to its multi-step manual process for generating graphics. To overcome this and add new features, we developed STNWeb [30], a web application that automates this process. STNWeb can be run locally via Docker from Code Ocean (see code metadata) or accessed online at:

<https://www.stn-analytics.com/>

For an introduction to STNWeb through practical examples, see the previous chapter.

10.2 STNWeb Architecture

STNWeb’s architecture is composed of three core components: the frontend, the backend, and the search space partitioning module. The user-friendly frontend enables users to efficiently input all necessary information regarding the algorithms to be analyzed. The backend communicates with the partitioning module via a REST API to generate the visualizations. The partitioning module contains specialized algorithms that divide the search space into meaningful regions—potentially grouping solutions to reveal patterns not immediately apparent. Figure 10.1 illustrates the interaction between these three components. Below, we describe each component in detail.

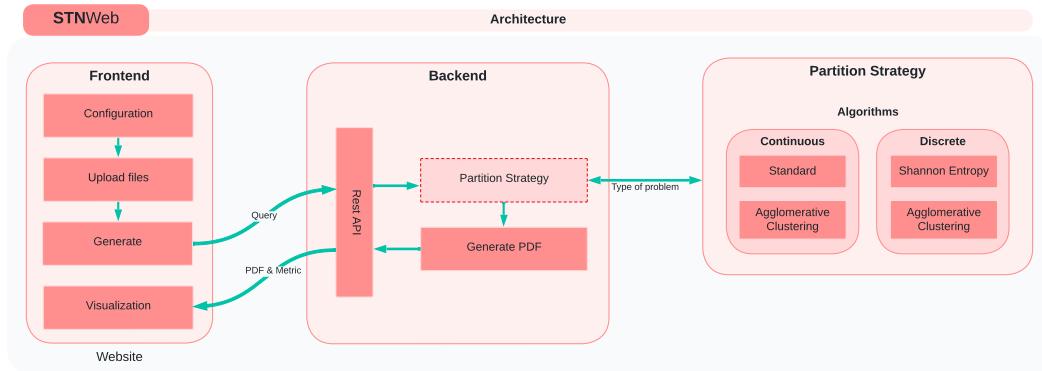


Figure 10.1: After completing the configuration form and uploading the required files for each algorithm to be compared (frontend), the user can request the generation of visualizations. The REST API (backend) receives the request and invokes the selected algorithm to partition the space and generate a PDF visualization based on the provided configuration. The resulting PDF is then displayed within the embedded viewer in the frontend.

10.2.1 STNWeb Frontend

The frontend was developed using Angular¹ and TypeScript², with Bootstrap³ for styling. It is designed as a single-page application. The interface allows users to select the type of optimization problem (discrete or continuous), choose a search space partitioning algorithm, and adjust configuration parameters. Users can also customize the visualization by modifying node colors and sizes. Once configured, the user uploads the trajectory data files for each algorithm and clicks the GENERATE button to produce the visualization. For instructions on how to format these files, please refer to the STNWeb tutorial (<https://github.com/camilochs/stnweb>).

10.2.2 STNWeb Backend

The backend was developed using both Python⁴ and R⁵. Python, with the Flask⁶ micro-framework, handles the REST API and partitioning module, while R is responsible for generating the final visualizations in PDF format. Upon receiving the configuration and trajectory files, the backend validates the input and applies the selected partitioning strategy. The processed data is then passed to R for visualization rendering.

10.2.3 REST API

The REST API provides the following two endpoints:

¹ <https://angular.io/>

² <https://www.typescriptlang.org/>

³ <https://getbootstrap.com/>

⁴ <https://www.python.org/>

⁵ <https://www.r-project.org/>

⁶ <https://flask.palletsprojects.com>

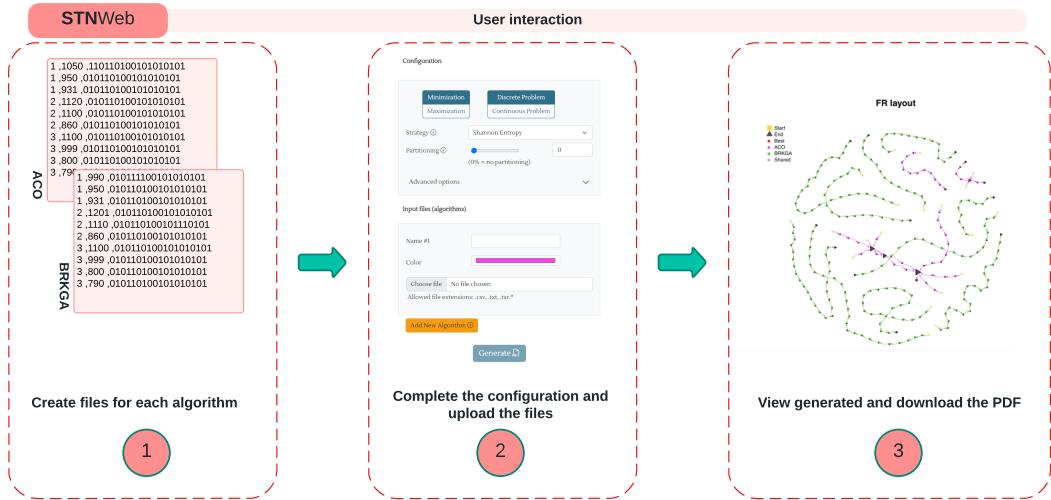


Figure 10.2: Interacting with STNWeb requires just three steps. First, generate separate data files with the search trajectories of the algorithms to be compared. Second, configure the analysis and upload the files on the STNWeb interface. Third, generate and download the visualization in PDF format.

- `/stn` – Accepts a POST request containing configuration settings and trajectory files. It returns a PDF visualization.
- `/metric` – Allows users to download the metrics file (provided in spreadsheet format) after the `/stn` endpoint has been called.

Using a REST API abstracts the logic from the user interface, making it easier to update or redesign the frontend without affecting the core backend functionality. This modularity ensures a seamless and consistent user experience (see Figure 10.2).

10.2.4 Search Space Partitioning Strategy

In some cases—due to either the problem instance or the algorithm used—search trajectories may become excessively long, resulting in cluttered STN visualizations that hinder analysis. To address this, STNWeb offers several strategies for partitioning the search space into clusters or regions that group multiple solutions. These strategies improve the clarity and informativeness of the resulting graphics. More details can be found in [155].

Note that search space partitioning is optional. The default configuration does not apply any partitioning. However, when used, it can significantly improve interpretability by focusing the visualization on the essential dynamics of the algorithms.

Partitioning strategies depend on the type of optimization problem. For instance, some methods—such as *Shannon Entropy*—are specific to discrete problems, while others—such as *Agglomerative Clustering*—can be applied to both discrete and continuous problems.

Partitioning Strategies for Discrete Optimization Problems

STNWeb offers two strategies for discrete optimization problems:

- *Shannon Entropy*: This method computes entropy values for each decision variable. Variables are then ranked by entropy, and the least informative ones are discarded from the visualization. See [155] for full details.
- *Agglomerative Clustering*: STNWeb implements a custom version of agglomerative clustering that groups solutions based on their distance in the search space and cluster characteristics such as size and density (refer to Chapter 12).

Partitioning Strategies for Continuous Optimization Problems

For continuous optimization problems, STNWeb provides the following options:

- *Standard Partitioning (Hypercubes)*: The search space is divided into hypercubes. While intuitive, this approach is only practical for problems with fewer than 5 decision variables and requires box-constrained domains.
- *Agglomerative Clustering*: Similar to the discrete case, but adapted to continuous distance metrics.

10.3 Limitations

A current limitation of STNWeb lies in its response time when generating visualizations, especially when comparing more than four algorithms or making minor adjustments (e.g., resizing or color changes). This delay arises because all visualizations are rendered server-side. Future versions will address this by incorporating a real-time web viewer capable of client-side rendering to reduce latency.

10.4 Conclusion

This work presented STNWeb, a web-based application designed to support optimization research by enhancing the usability of the original STNs tool introduced in [155]. In addition to simplifying data input and visualization generation, STNWeb incorporates search space partitioning strategies suitable for both discrete and continuous optimization problems. Future work will explore the following enhancements:

1. Incorporating 3D visualization technologies to reveal structural properties not visible in 2D projections.
2. Enabling real-time, client-side graph modifications to improve interactivity and analysis depth.
3. Designing and integrating new partitioning strategies to expand analytical capabilities.

Note

This was my first journal paper. Although it might not seem like it because it was a software paper, and they are typically short. Anyway, it still counts. As of today (June 2025), the paper on which this chapter is based is my most cited paper. And it makes me feel good, especially knowing that researchers are using the tool.

STNWeb in the Literature. Some studies have employed STNWeb to analyze optimization behavior. Notable examples include:

1. Zermani et al. “FPGA-based hardware implementation of chaotic opposition-based arithmetic optimization algorithm.” *Applied Soft Computing* [239].
2. Babaagba et al. “Exploring the use of fitness landscape analysis for understanding malware evolution.” *Genetic and Evolutionary Computation Conference* [10].
3. Akbay et al. “Two examples for the usefulness of STNWeb for analyzing optimization algorithm behavior.” *Metaheuristics International Conference* [3].

11

Enhancing the Explainability of STNWeb with Large Language Models

Foundational Work for This Chapter

This chapter is based on the following publication:

- **Title:** *Large Language Models for the Automated Analysis of Optimization Algorithms*
- **Published in:** Genetic and Evolutionary Computation Conference (GECCO) - Core A
- **Type:** Conference Paper
- **Year:** 2024
- **Main Contribution:** Generate a natural language explanation of the STNWeb graphics using a Large Language Model
- **Problem Addressed:** Enhancing the explainability of STNWeb graphics through the use of Large Language Models
- **Type of contribution:** Methodological
- **DOI:** <https://doi.org/10.1145/3638529.3654086>
- **Current number of citations in Google Scholar:** 9

11.1 Introduction

In late 2023, a compelling talk on Large Language Models (LLMs) at an AI event in Brussels sparked my interest. Shortly after, having completed research on STNWeb (see Chapters 9 and 10), I experimented with ChatGPT. I quickly realized LLMs could solve a key STNWeb challenge: generating explanatory written reports for its visualizations. This is complex, as STNWeb graphics use a visual terminology requiring prior understanding. A simple test confirmed LLMs could significantly simplify this task,

as they could lower the entry barrier for researchers using STNWeb. This initial experiment allowed me to complete the research rather fast, with subsequent work focusing on methodology.

This work initiated further research integrating LLMs into algorithmic interpretation and improvement within computational optimization (see subsequent chapters). This approach adheres to two principles: (1) designing a concise prompt that effectively contextualizes the problem, and (2) integrating the prompt's output into an existing system or algorithm.

Visualization tools, such as Search Trajectory Networks (STNs), are increasingly used in optimization to analyze metaheuristic algorithms [71, 22]. STNs reveal how problem characteristics influence algorithm behavior, aiding deeper understanding and informed algorithm design [154, 155]. The web-based STNWeb [33] automated STN generation but requires guidance for interpreting the resulting graphics. This research explores using Large Language Models (LLMs) to address this interpretation gap.

LLMs have revolutionized Natural Language Processing (NLP) with high-quality, versatile outputs [168]. Models like ChatGPT and DALL-E 2 set industry standards [157, 170], while GitHub Copilot automates code generation [199, 208]. Beyond applications like sentiment analysis, text classification, and machine translation [243, 200, 241], LLMs are now designing algorithms, particularly metaheuristics [165]. A key challenge is LLM "hallucinations" (unrealistic data), mitigated by prompt engineering. This technique, by carefully crafting inputs, enhances response reliability and explainability [174, 63, 245], effectively enabling "natural language programming."

This work integrates LLMs into STNWeb to automatically generate user prompts, improving comprehension of graphics and algorithm behavior. We also demonstrate LLMs' ability to generate basic plots (e.g., bar charts), augmenting the natural language reports. The integration process involves three tasks (A, B, C), as depicted in Figure 11.1.

11.2 Background

11.2.1 Search Trajectory Networks (STNs)

Note

Even though the following introduction to STNs is largely redundant, we decided to keep it due to the example used is different to the one used in earlier chapters.

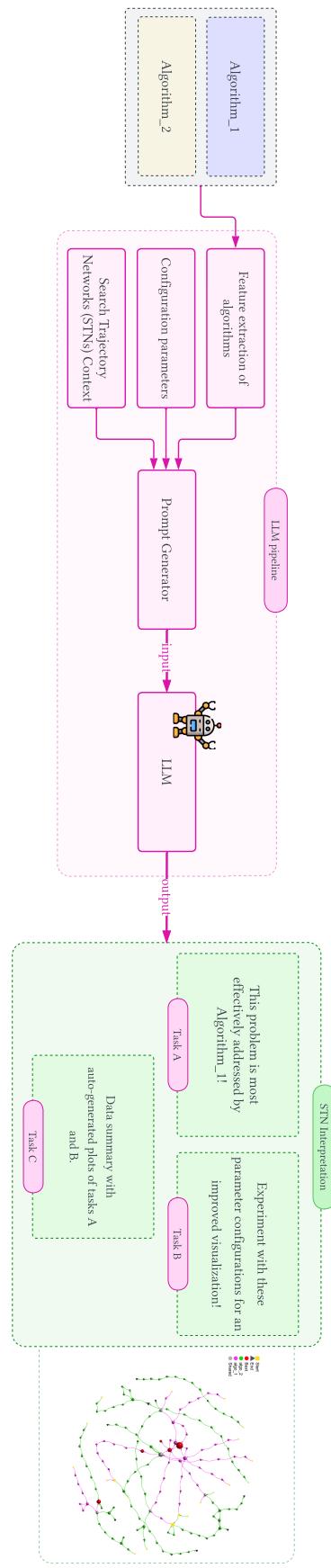


Figure 11.1: Overview of automating graphics interpretation with Large Language Models (LLMs).

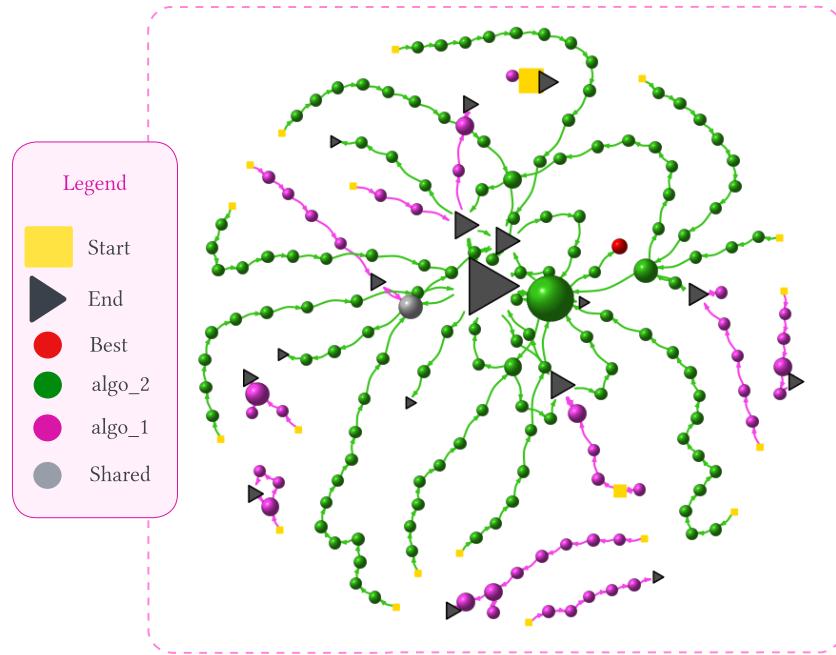


Figure 11.2: Example STN generated by STNWeb for the Rastrigin function.

STNs visualize directed graphs to analyze stochastic optimization algorithms, particularly metaheuristics. They enable researchers to deeply understand algorithm behavior on specific problem instances, revealing traits like attraction to local optima or limited exploration. The interpretability of STNs, however, hinges on understanding their visual language. This section details STN interpretation and how LLMs can simplify it.

Figure 11.2 illustrates an STN for two optimization algorithms on the *Rastrigin function*. Each vertex represents a search space chunk (formed by clustering), and dots indicate solutions. The STN's elements are:

- **Trajectories:** Depicted in distinct colors (e.g., `ALGO_1` in green, `ALGO_2` in purple).
- **Start Points:** Marked with yellow squares.
- **End Points:** Dark grey triangles (non-best solutions) and red dots (best solutions).
- **Shared Solutions:** Pale grey dots represent chunks visited by multiple algorithms.
- **Vertex Size:** Indicates the number of trajectories passing through it; larger vertices mean more traffic.

The STN in Figure 11.2 highlights `ALGO_2`'s attraction to a specific region, with seven of its ten trajectories converging there, unlike `ALGO_1`. The lack of overlap in `ALGO_1`'s trajectories suggests low robustness; in other words, `ALGO_1`'s trajectories tend to diverge and fail to consistently reach the best solution, indicating a more unstable behavior compared to `ALGO_2`.

While STNs offer rapid insights compared to traditional tables, their interpretation requires familiarity with the original STN literature [155, 30, 33], a time-consuming prerequisite. To enhance STN usability by eliminating this knowledge barrier, this work integrates LLMs to automatically generate natural language reports and supplementary plots.

11.2.2 Large Language Models (LLMs)

Large Language Models (LLMs) like GPT-4 [157], Mixtral [93], Gemini [205], and Llama 2 [207] are advanced, parameter-rich models capable of understanding and generating human language. Practical applications such as ChatGPT, Bing Chat, and Google Bard demonstrate their significant value.

At their core, LLMs utilize the self-attention mechanism within transformers [209], the fundamental unit for predicting tokens in language sequences. Transformers have revolutionized Natural Language Processing (NLP) by efficiently completing text and leveraging parallel processing on modern hardware to grasp extensive textual dependencies, thereby enhancing contextual quality. As LLMs scale in parameters, emergent abilities have surfaced [219], including few-shot learning [25], zero-shot problem-solving [107, 179], chain-of-thought reasoning [220], instruction following [158], directional stimulus [119], and retrieval-augmented generation [113]. These abilities are accessed via prompts, the primary interface for interacting with LLMs. Prompt engineering [248] is crucial for selecting the appropriate ability to elicit desired responses.

Our approach leverages pre-trained LLMs, allowing us to achieve our objectives through prompt adjustment [168, 125]. This bypasses the need for costly fine-tuning or maintaining our own LLM infrastructure, instead utilizing readily available models, both paid (e.g., OpenAI's GPT-4) and free (e.g., from Hugging Face).

By selecting suitable emergent abilities, we will develop three distinct prompts. Each prompt is designed to fulfill a specific function, providing users with information to interpret STNWeb graphics. This integration aims to simplify STNWeb's usage and significantly reduce the prerequisite knowledge needed for interpreting its visualizations.

11.3 Integrating LLMs into STNWeb

Our objective is to leverage Large Language Models (LLMs) to generate interpretations of Search Trajectory Network (STN) graphics, making them accessible to users without specialized knowledge. To achieve this, STNWeb has been extended to interact with LLMs by generating prompts and displaying LLM-generated interpretations (natural language summaries and supplementary plots). This transforms STNWeb into a more user-friendly and effective analysis tool for both beginners and experts.

The integration process involves two stages: prompt template engineering and feature extraction. STNWeb autonomously executes both, gathering data on algorithm

behavior and search space partitioning parameters from input files. It then creates tailored prompts for the LLM, which generates the interpretation.

11.3.1 Prompt Engineering

Prompting offers an intuitive interface for interacting with generalist models like LLMs. However, effective LLM guidance requires careful engineering, as their interpretation can differ from human understanding [218]. Prompt engineering is thus akin to "programming in natural language" [174].

We developed three prompt templates (Tasks A, B, and C) to leverage STN graphic information for the following:

- (A) Determining a clear winner among compared algorithms.
- (B) Suggesting improved agglomerative clustering parameters for more interpretable STN graphics.
- (C) Generating an automated summary with plots for Tasks A and B.

These templates are automatically instantiated by STNWeb. As shown in Figure 11.3, Tasks A and B templates use tags with static (cyan) and dynamic (orange) instructions. Dynamic tags adapt to STN features and user parameters. This structured approach aims to mitigate LLM hallucinations and enhance response coherence.

Task A: Algorithm Comparison

The prompt begins with a [CONTEXT] tag describing STNWeb, setting the LLM's understanding of terminology. The [RULES] tag outlines algorithm superiority criteria (detailed in Table 11.1). The dynamic [DATA] tag includes three algorithm-specific features extracted from the STN. Finally, the [QUERIES] tag guides the LLM on response formatting and content.

Task B: Parameter Tuning

Similar to Task A, this template starts with [CONTEXT]. The [PARAMETERS DEFINITIONS] tag describes the four agglomerative clustering parameters (Table 11.1), followed by the [DATA] tag containing user-selected parameter values. The [QUERIES] tag guides the response format.

Task C: Summary and Plots

This task uses two concise, static templates. Each prompts the LLM to generate a plot based on instructions and a CSV data file. The first template generates a grouped bar plot comparing algorithm performance (Task A data), while the second visualizes parameter settings (Task B data). This task leverages Chat2VIS [137, 136] for plot generation.

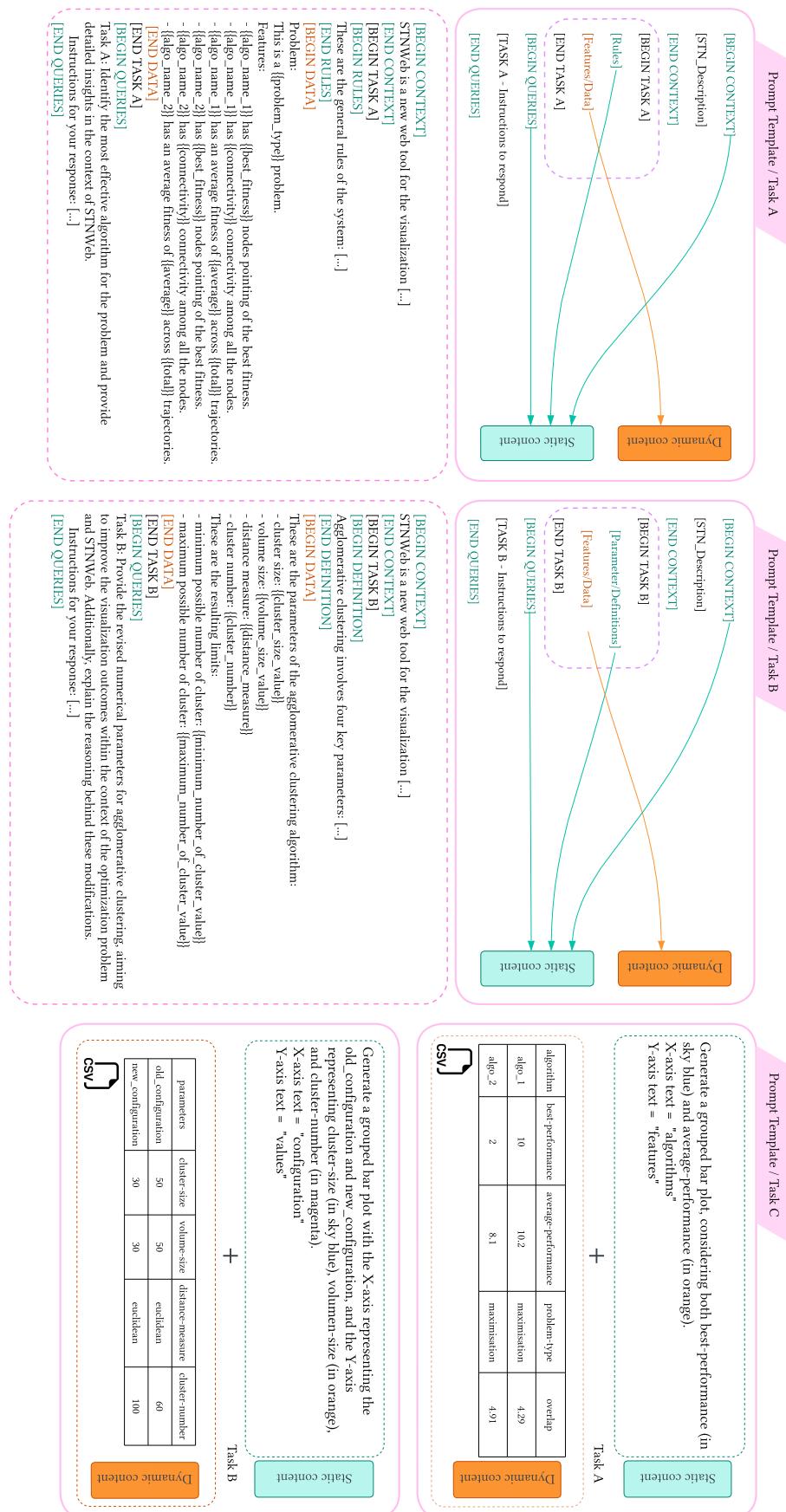


Figure 11.3: Prompt templates automatically generated by STNWeb for each task.

Table 11.1: LLM Tasks for STN Interpretation: Guidelines and Expected Inferences

Task A: Algorithm Winner Determination	
<i>Inference:</i> LLM identifies the best algorithm based on three rules and STNWeb-computed metrics (quality, robustness, average fitness). It acknowledges ties if algorithms perform similarly.	
Rule	Prompt Instruction
1	<i>Higher quality: More nodes pointing to best-fitness nodes.</i>
2	<i>More robust: Higher inter-trajectory connectivity, if finding best-fitness nodes.</i>
3	<i>Superiority: Favors lower average fitness (minimization) or higher (maximization).</i>
Task B: Agglomerative Clustering Parameter Tuning	
<i>Inference:</i> LLM suggests improved parameters for search space partitioning based on four control parameters and user-provided values.	
Parameter	Description
1	<i>Cluster size (%): Max cluster size relative to total solutions.</i>
2	<i>Volume size (%): Max cluster size relative to search space volume.</i>
3	<i>Distance measure: Solution distance metric (Hamming, Euclidean, Manhattan).</i>
4	<i>Cluster number: Number of clusters; ideal is above minimum but not maximum.</i>
Task C: Automated Summary with Plots	
<i>Inference:</i> LLM generates plots and a summary, suggesting parameters for Task B to demonstrate changes. It uses prompts and STNWeb data to create visualizations.	
Plot Type	Specification
1	<i>Grouped bar plot: Best performance (sky blue) vs. average performance (orange).</i>
2	<i>Grouped bar plot: X-axis (old/new config), Y-axis (cluster-size, volume-size, cluster-number).</i>

Table 11.1 details the STNWeb-specific rules and expected LLM inferences for each task. While these prompts are basic, they serve as a proof of concept for LLM-driven STN interpretation.

These prompts are designed for zero-shot learning [107, 179], meaning the LLM must make decisions based solely on the provided rules and data, without explicit examples.

11.3.2 Feature Extraction

The dynamic elements of our prompt templates ([DATA] tags for Tasks A/B, CSV files for Task C) require feature extraction from STNWeb’s data.

Task A: Algorithm Performance Features

STNWeb extracts three key features to help users identify the better-performing algorithm. The LLM uses these to automatically determine a winner or indicate a [draw]



Figure 11.4: Example prompts and GPT-4-turbo outputs for the STN in Figure 11.2.

if no clear preference exists.

The extracted features and their corresponding prompt sentences are:

Feature 1: Total Best Global Fitness. Counts trajectories ending in the globally best-found fitness. Higher counts suggest a better algorithm.

$$\text{TotalBestGlobalFitness}_{\text{ALGO}_K} = \sum_{i=1}^{|M_k|} \mathbb{I}(\text{BestFitness}_{k,i} = \text{BestGlobalFitness}), \quad (11.1)$$

Example sentence: ALGO_2 has 1 nodes pointing to nodes with the best fitness.

Feature 2: Inter-trajectory Connectivity. Measures overlap between an algorithm's distinct trajectories. Higher connectivity suggests greater robustness.

$$\text{Connectivity}_{\text{ALGO_K}} = \frac{\# \text{ pairs } T \neq T' \in M_k \text{ with an overlap}}{(|M_k| \cdot (|M_k| - 1))/2}, \quad (11.2)$$

Example sentence (for a value of 0.62): ALGO_2 has 0.62 connectivity among all the nodes.

Feature 3: Average Fitness. The mean of the best fitness values from all trajectories of an algorithm. A better average fitness generally indicates a superior algorithm.

$$\text{AvgFitness}_{\text{ALGO_K}} = \frac{1}{|M_k|} \sum_{i=1}^M \text{BestFitness}_{k,i} \quad (11.3)$$

Example sentence (for a value of 78.081 with 10 trajectories): ALGO_2 has an average fitness of 78.081 across 10 trajectories.

Task B: Clustering Parameter Configuration

This task extracts user-selected agglomerative clustering parameters for search space partitioning. The prompt includes these parameters and their ranges.

Agglomerative clustering parameters:

- cluster size: 5%
- volume size: 5%
- distance measure: Euclidean
- cluster number: 400 (range: [207, 574])

The LLM uses this information to suggest improved parameters.

Task C: Plot Generation

Prompts are static, but CSV files containing data for Tasks A and B are dynamic. Using Chat2VIS [137, 136], the LLM generates plots based on these prompts and files. Examples are in Figure 11.3 and Figure 11.4.

LLMs can process numerical data and keywords to link them to rules. However, complex mathematical reasoning can be challenging [114, 220]. Our approach uses simple, task-specific rules to enhance LLM reasoning accuracy.

11.4 Empirical Evaluation

We evaluate prompt quality for Tasks A, B, and C through a combined system and human evaluation approach. System evaluation compares LLM outputs against expected results and specific query formats. Human evaluation, conducted by STN ex-

perts (the authors of this paper), assesses prompt clarity, specificity, and effectiveness. This evaluation is crucial because LLMs can produce text that varies widely in detail and organization—some LLMs may generate lengthy outputs without conveying relevant information. For example, an LLM might return a well-written description of an STNWeb graph that, nevertheless, provides an incorrect or inadequate interpretation of the data.

11.4.1 Setup

We utilize web platforms for generating multiple LLM outputs.

Tasks A and B: LLM Comparison

For Tasks A and B, we use Chatbot Arena [246], an open-source platform from LMSYS and UC Berkeley SkyLab, offering over twenty LLMs. We selected GPT-4-turbo [157], Mixtral-8x7b-instruct-v0.1 [93], and Tulu-2-dpo-70b [92]. Chatbot Arena’s Leaderboard ranks LLMs based on user-voted response quality. We chose GPT-4-turbo (currently ranked first) and the next two highest-ranking open-source models to ensure a comprehensive performance overview.

Task C: Plot Generation

As mentioned, we use Chat2VIS [137, 136], a tool for generating data visualizations via natural language. It supports OpenAI’s GPT models [25, 157] and Meta’s Code Llama [177]. Plot generation relies on LLM-generated Python code, highlighting their coding abilities [247].

11.4.2 Methodology

We created four prompts for Tasks A and B (two easy, two hard) based on selected STNWeb graphics and templates. Prompt difficulty is determined by the dynamic [DATA] section. Each prompt is run five times per LLM to account for output stochasticity. Task C uses a single prompt with human evaluation only, as its output (bar plots) is not natural language. The methodology is illustrated in Figure 11.5.

Task A: Winner Determination

Easy prompts compare two algorithms with a clear winner; hard prompts compare three with two performing comparably. The [QUERIES] tag specifies output format: [ALGORITHM_NAME] for a winner or [DRAW] for ties.

- **System Evaluation:** Score $LLM_i = \frac{correct}{5}$ per LLM, where $correct$ is the number of accurate outputs out of five.
- **Human Evaluation:** Score $LLM_i = \frac{wins}{5}$ per LLM, where $wins$ is the number of times the LLM was judged superior.

Table 11.2: LLMs evaluations for tasks.

		Evaluation			
		System		Human	
Task	Prompt Type	LLM	Score (↑)	LLM	Score (↑)
A	1/Easy	GPT-4-turbo	1	GPT-4-turbo	0.6
		mixtral-8x7b-instruct-v0.1	0.8	mixtral-8x7b-instruct-v0.1	0.4
		tulu-2-dpo-70b	1	tulu-2-dpo-70b	0
A	2/Hard	GPT-4-turbo	0.8	GPT-4-turbo	1
		mixtral-8x7b-instruct-v0.1	0.4	mixtral-8x7b-instruct-v0.1	0
		tulu-2-dpo-70b	0.1	tulu-2-dpo-70b	0
B	1/Easy	GPT-4-turbo	1	GPT-4-turbo	1
		mixtral-8x7b-instruct-v0.1	0	mixtral-8x7b-instruct-v0.1	0
		tulu-2-dpo-70b	0	tulu-2-dpo-70b	0
B	2/Hard	GPT-4-turbo	0.6	GPT-4-turbo	1
		mixtral-8x7b-instruct-v0.1	0	mixtral-8x7b-instruct-v0.1	0
		tulu-2-dpo-70b	0	tulu-2-dpo-70b	0

Human evaluation		
Task	LLM	Score (↑)
C	GPT-4	1
	CodeLlama-34b-Instruct-hf	1

Task B: Parameter Tuning

Similar to Task A, we use two prompts (easy/hard). System evaluation checks LLM output compliance with task rules (Table 11.1) against the prompt's parameter settings. Both system and human evaluations are employed.

Task C: Plot Quality

Plot quality and correctness are assessed solely through human evaluation, as the output is not text-based.

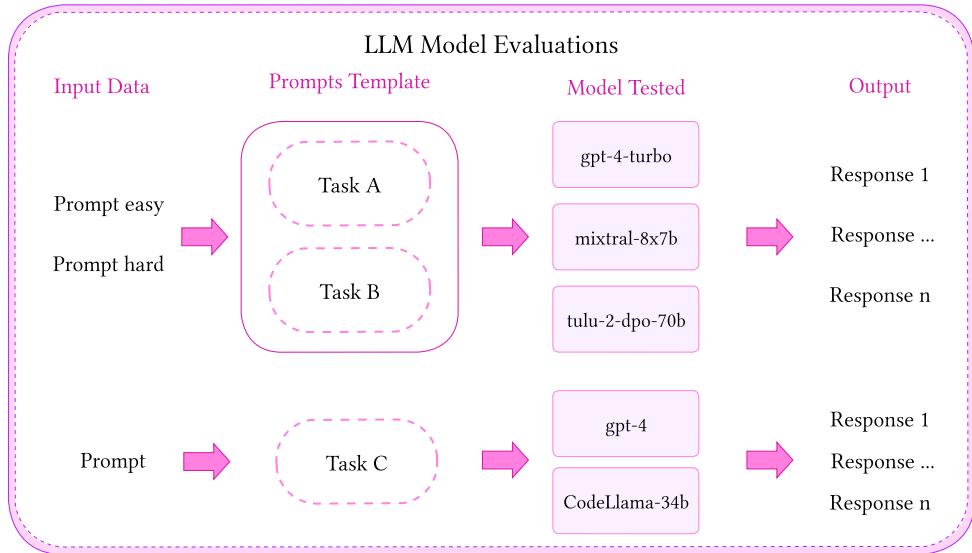


Figure 11.5: Methodology for evaluating LLMs across tasks.

11.4.3 Results

This section summarizes quantitative evaluation results. Detailed prompts, LLM outputs, and response variations are in the supplementary material and at <https://github.com/camilochs/explainability-LLM-stnweb>. Figure 11.4 shows example LLM responses.

Table 11.2 presents the evaluation. GPT-4-turbo excels in Tasks A and B, handling both easy and hard prompts proficiently. For hard Task A prompts (three algorithms, two similar), GPT-4-turbo was the only model to correctly identify a draw, unlike others' consistent inaccuracies. For easy Task A prompts, while all models performed similarly, GPT-4-turbo's response was judged superior by human evaluators for its conciseness and directness. It provided the answer first, then justification, unlike others that explained first. A partial score of 0.6 for GPT-4-turbo on an easy Task A prompt (human evaluation) reflects its correct winner identification but non-standard parenthetical referencing of other algorithms.

In Task B, only GPT-4-turbo adhered to the specified format for reporting improved parameters: '[parameter_name=new_value]'. Other models used ":" instead of "=" and omitted parentheses. Figure 11.4 shows a complete GPT-4-turbo response.

For Task C, GPTs and Code Llama via Chat2VIS showed no significant performance differences, likely due to the simplicity of our prompts and data.

11.5 Discussion

Applying LLMs to optimization tools like STNWeb involves three key considerations:

1. **Limitations and Trustworthiness:** LLMs have inherent limitations in complex mathematical reasoning [114, 220] and can produce inaccuracies or hallucinations.

tions, especially with unstructured or subjective prompts [88]. Robust benchmarking frameworks [249, 121, 197] are essential for evaluating LLM performance and ensuring trustworthiness.

2. **Open-Source LLMs:** While proprietary models like GPT-4 often outperform open-source alternatives [25, 174], their capabilities can be enhanced through few-shot learning, prompt tuning, and advanced prompt engineering [112]. Future research should focus on improving open-source LLM performance and accessibility.
3. **Explainability:** LLMs can bridge the knowledge gap by enhancing tool usability and explainability, potentially through multimodal outputs [237]. However, our findings show prompt engineering effectiveness is tied to the LLM’s reasoning ability; GPT-4’s strength with simple rules was key. Integrating external information, such as related articles via Retrieval-Augmented Generation (RAG) [70], can further improve LLM accuracy and explainability by providing context not present in the prompt itself.

11.6 Conclusion

This research leveraged Large Language Models (LLMs) to automate the generation of text and plots for STNWeb. This significantly reduces the prerequisite knowledge needed to analyze optimization algorithm comparisons, making STNWeb more accessible for both discrete and continuous problems. Our findings demonstrate that meticulous prompt engineering can yield LLM-generated reports with enhanced trustworthiness and explainability, even when dealing with LLMs’ known limitations in complex mathematical reasoning.

Future work will focus on augmenting the extracted algorithmic features to improve LLM insights and explainability. Additionally, we plan to explore multimodal LLMs to directly interpret STNWeb graphs, enriching textual descriptions with visual analysis.

Note

I presented this work at GECCO 2024, held in Melbourne, Australia, which, unknown to me at the time, would be my last conference before submitting this thesis. I remember a strong excitement surrounding this research, coinciding with the peak of discussions on large language models (LLMs). Unintentionally, we were at the forefront of a new application: leveraging LLMs as assistants to enhance the interpretation of optimization tools' results, a development that gave me great personal satisfaction.

This enthusiasm arose from identifying a clearly impactful use case: employing LLMs to simplify and clarify complex textual information—an ability easily verified by anyone who has used ChatGPT or other LLMs. Although I could have chosen to continue this line of work, I wanted to explore other applications of LLMs in optimization (life is too short!); this was the last project in my PhD where I focused on interpretability using LLMs.

In contrast, my future research with LLMs, such as code generation or optimization (as a continuation of Chapters 6 and 7), will require considerable effort, not so much in implementation, which is relatively straightforward, but in convincingly demonstrating its practical value. This is still viewed by many.

12

Improving STNWeb Graphical via Hierarchical Agglomerative Clustering of the Search Space

Foundational Work for This Chapter

This chapter is based on the following publication:

- **Title:** *An Extension of STNWeb Functionality: On the Use of Hierarchical Agglomerative Clustering as an Advanced Search Space Partitioning Strategy*
- **Published in:** Genetic and Evolutionary Computation Conference (GECCO) - Core A
- **Type:** Conference Paper
- **Year:** 2024
- **Main Contribution:** Integration of an advanced search space partitioning strategy into STNWeb
- **Problem Addressed:** In some instances, optimization algorithm results fail to result in useful visualizations in STNWeb
- **Type of contribution:** Algorithmic & Methodological
- **DOI:** <https://doi.org/10.1145/3638529.3654084>
- **Current number of citations in Google Scholar:** 1

12.1 Introduction

This contribution focuses not on modifying the STNWeb interface itself, but on improving the underlying behavior of the STN model embedded within it. The work was driven by the need to improve the interpretability of STNWeb's visualizations, which, in certain cases, failed to provide meaningful insights. Specifically, the absence of a

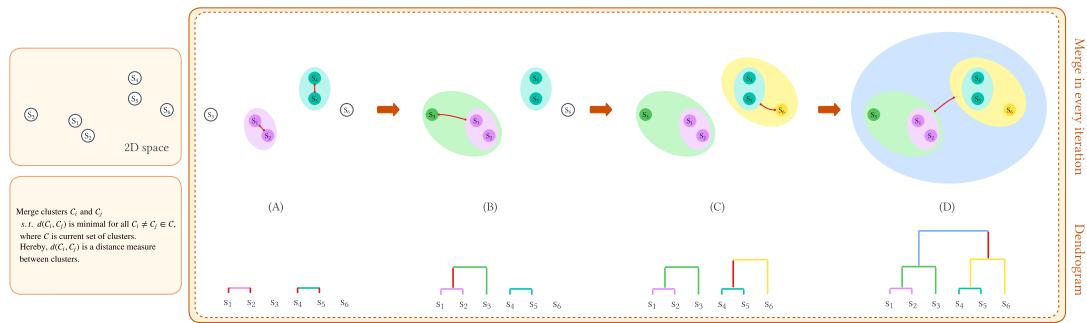


Figure 12.1: Example illustrating single-linkage HAC. At each step, the two clusters with the minimum distance (according to a distance metric) are merged. The illustration starts after the first two steps are already performed (to shorten the procedure).

more advanced partitioning algorithm often resulted in visualizations that lacked relevant structure, making them ineffective for researchers. This limitation, initially identified by my PhD supervisor, motivated the enhancement presented here. To address it, I integrated the Hierarchical Agglomerative Clustering (HAC) algorithm into the STN construction process. This integration allowed the model to reveal deeper structural patterns in complex instances, significantly enhancing the clarity and usefulness of the resulting visualizations.

In this work, we enhance STNWeb's interpretive capabilities by introducing a novel search space partitioning strategy. This innovation enables the creation of improved visualizations, specifically addressing challenges posed by large discrete solutions or an increased number of dimensions in continuous optimization problems.

12.2 Contextual Overview: Search Trajectory Networks

12.2.1 Search Space Partitioning Schemes

STNs based on original search trajectories are sometimes hard to visualize and/or interpret. The reasons for this are manifold. When computation times are long, or algorithms conduct rather small steps at each iteration, search trajectories might be very long, leading to cluttered STN visualizations. In addition, in continuous optimization problems, for example, hardly any solution is repeated in different search trajectories, because the floating point values of decision variables hardly ever coincide exactly. Therefore, often no overlaps are detected between different search trajectories. Search space partitioning schemes are therefore essential for extracting and visualizing the essential features of STNs.

This concept is analogous to zooming in on an image, enabling the observation of intricate details such as pixels and specific error types that might otherwise be challenging to discern. In the context of STNs, the concept of search space partitioning aligns

with this analogy. In this scenario, the “image” is the entirety of representative solutions across all considered search trajectories, and the “pixels” signify a condensed set of solutions modelling the search space after merging different representative solutions into common locations.

12.2.2 Standard Strategies for Partitioning

The initial STN article [155] proposed a standard search space partitioning (respectively, solution merging) scheme both for discrete and continuous problems. In the discrete case, Shannon Entropy is employed for this operation, along with a parameter representing the partitioning percentage. In the continuous case, search space partitioning is accomplished through a division of the search space into hypercubes, based on a so-called partitioning factor. Below we provide short descriptions of both cases. However, for a more in-depth understanding, we refer the interested reader to [155]. For simplicity, we shall refer to both of these original strategies as the *standard partitioning strategies*.

Discrete/Combinatorial Case. The principal idea of the standard partitioning strategy is to achieve a merging of representative solutions from the search trajectories by removing certain decision variables from the search space. More specifically, the Shannon entropy of all decision variables is calculated based on their value settings in all representative solutions found in the considered search trajectories. Variables are then sorted according to non-decreasing Shannon entropy values. Note, in this context, that variables that have the same value in many representative solutions have a low Shannon entropy, that is, their information content is low, and they might be considered for being removed. Accordingly, variables are sorted with respect to non-increasing Shannon entropy values and a percentage (PP) of the variables with the lowest Shannon entropy are removed.

Continuous Case. The standard partitioning strategy in the continuous case was defined for problems with box constraints. That is, we assume that the values for the variable of each dimension are limited to $[x_{\min}, x_{\max}]$. The search space is divided into hypercubes of width 10^{PF} in each dimension, where PF is the so-called partitioning factor. All representative solutions that fall into the same hypercube are then assigned to the same location. In [155], PF was formulated as a function of the domain range ($x_{\max} - x_{\min}$) and the problem dimension (D) as follows: PF was set to $n - 2$, where n is the largest integer satisfying the condition $(x_{\max} - x_{\min}) \times D \geq 10^n$. For example, for a twenty-dimensional problem with a box constraint of $[-100, 100]$ in all dimensions, the PF would be set to 1, given that $200 \times 20 \geq 10^3$.¹ From the offset, these standard search space partitioning methods have the following disadvantages. First, discrete and continuous problems are handled in very different ways. It might be convenient to have a method that works well in both cases. Second, the method based on Shannon entropy in the discrete case will not work very well, for example, for problems based on a few

¹ In STNWeb, the user interface automatically computes the potential range for PF .

variables with large domains. The fewer variables a problem has, the less fine-grained is the search space partitioning method. Third, identification of the hypercubes (locations) in the continuous case becomes impractical with a growing problem dimension.

12.3 Partitioning By Hierarchical Agglomerative Clustering

Hierarchical agglomerative clustering (HAC) is a clustering approach that adopts a bottom-up strategy [149]. Initially, each data point is treated as an individual “cluster”. The algorithm systematically operates according to a designated linkage criterion, potentially employing a range of different metrics—such as complete linkage, single linkage, average linkage, or Ward linkage—to decide which clusters should be merged at each step. For search space partitioning, we use single-linkage HAC extended with a mechanism that filters out those pairs of current clusters whose merging would result in a cluster that is too large with respect to, at least, one out of two different size measures (defined below).

In the following, we first introduce standard single-linkage HAC before we outline our extension. HAC is an algorithm for hierarchical clustering that constructs a tree of clusters (see Figure 12.1). The algorithm iteratively merges the closest clusters until a single cluster, encompassing all data points, is formed. Here is a detailed breakdown of the algorithm:

1. **Initialization:** Start by treating each data point (solution) as a separate singleton cluster.
2. **Identify Closest Clusters:** Determine the two closest clusters based on a distance measure $d(.,.)$ between clusters.
3. **Merge Closest Clusters:** Combine the two nearest clusters into a new cluster. The original two clusters are replaced by the new one.
4. **Iterate:** Repeat steps 2 to 4 until only a single cluster remains, forming a hierarchy or dendrogram.

Note that, in practice, distances between clusters are stored in memory and, at each iteration, only the distances between the new cluster (the result of the merging process) and the remaining old clusters must be newly calculated. The outcome of the process described above is a tree-like structure (dendrogram) visually representing relationships between clusters at varying levels of granularity. The algorithm is provided in more technical terms as a pseudo-code in Algorithm 3.

When applying this algorithm to search space partitioning in the context of STNs, the set of data points (S) consists of all unique representative solutions from the set of considered search trajectories, that is, S does not contain any duplicates of representative solutions.

Algorithm 3 HAC with Single Linkage

Require: Set of data points $S := \{x_1, x_2, \dots, x_n\}$
Ensure: Dendrogram representing the hierarchical clustering

- 1: Initial cluster set: $C := \{\{x_1\}, \{x_2\}, \dots, \{x_n\}\}$
- 2: $\text{stop} \leftarrow \text{false}$
- 3: **while** $\text{stop} = \text{false}$ **do**
- 4: $P \leftarrow \{(C_k, C_l) \mid C_k \neq C_l, C_k, C_l \in C\}$
- 5: **if** $P = \emptyset$ **then**
- 6: $\text{stop} \leftarrow \text{true}$
- 7: **else**
- 8: $(C_i, C_j) \leftarrow \arg \min \{d(C_k, C_l) \mid (C_k, C_l) \in P\}$
- 9: $C' \leftarrow C_i \cup C_j$
- 10: $C \leftarrow C \setminus \{C_i, C_j\} \cup \{C'\}$
- 11: **end if**
- 12: **end while**
- 13: **return** Dendrogram representation of the clustering

Non-Numeric Representations in Discrete Optimization

Note that if solutions are non-numeric vectors, which happens in the case of some discrete problems, they must first be converted to numerical vectors to be processed by the visualization and analysis tools. STNWeb's data loaders handle this conversion automatically.

Discrete Solution Spaces:

For discrete problems where solutions are represented as strings of characters (e.g., permutations like "C-A-B", symbolic sequences, or bitstrings), a direct character-to-integer mapping is applied. Functionally, the conversion mechanism processes each solution string character by character, mapping it to its corresponding integer ordinal value (such as its standard ASCII or Unicode code point).

- **Example:** A solution string "CAB" is transformed into the numerical vector [67, 65, 66], where 67, 65, and 66 are the ASCII values for 'C', 'A', and 'B' respectively.
- **Outcome:** This process yields a numerical representation for each solution, enabling distance calculations and projection into the 2D space.

However, sets of data points extracted from several search trajectories have specific characteristics. At the start of the search process, for example, search algorithms tend to make rather large steps. In contrast, search trajectories often converge to areas of attraction, causing most of the produced solutions to be rather close together in a confined area of attraction. We conjecture that such data point distributions would favor the creation of a super-cluster in the area of attraction while clustering at the beginning

of search trajectories would basically not happen.

Therefore, we added the following mechanism to the HAC algorithm to adapt it to the characteristics of search trajectories. In particular, we add the following instruction between lines 6 and 7 of Algorithm 3:

$$P := \text{Filter}(P)$$

This function filters out (removes) a cluster pair (C_k, C_l) from P if one of the two following conditions is fulfilled.

1. $\frac{|C_k \cup C_l| \cdot 100}{|S|} > CS$, where $CS \in [0, 100]$ is a user-defined parameter called cluster size limit. Hereby, the size of a cluster is the number of solutions it contains.
2. $\frac{\|M_{kl}\|_F \cdot 100}{\|M_S\|_F} > VS$. Hereby, M_{kl} is the matrix obtained by including all solutions from C_k and C_l as rows (or columns), and M_S is the matrix obtained by adding all solutions from S . Moreover, $\|\cdot\|_F$ is the Frobenius norm, which is a measure of the size, respectively the magnitude, of a matrix. Finally, $VS \in [0, 100]$ is a user-defined parameter called the cluster volume limit.

The filtering operation based on these two measures avoids the creation of clusters that are too large.

12.4 Experimental Results

The extended version of HAC was implemented in Python 3.11 and incorporated into STNWeb,² which served as a test lab for the experiments described in the following. In this context, note that in addition to using STNWeb online, it is also possible to run STNWeb locally, following the instructions outlined in the GitHub repository <https://github.com/camilochs/stnweb/>.

12.4.1 Methodology and Setup

To evaluate the proposed search space partitioning method, we employ both visual and quantitative comparisons across discrete and continuous optimization problems. All STNWeb visualizations follow a consistent encoding:

- Trajectories from different algorithms are shown in distinct colors (as per the legend).
- Yellow squares indicate starting points.
- Endpoints are marked as dark grey triangles (non-optimal) or red dots (best-found solutions).
- Pale grey dots highlight shared locations traversed by at least two algorithms.
- Node size reflects the number of trajectories passing through it: larger nodes represent more frequent traversal.

² <https://www.stn-analytics.com>

Figures 12.2 and 12.3 (discrete problems) are structured into four rows:

1. **Row 1:** Plot (A) shows the original STN (without partitioning), along with a table reporting STN metrics.
2. **Row 2:** Plots (B), (C), and (D) display STNs produced using standard partitioning, with increasing partitioning intensity from left to right.
3. **Row 3:** Plots (E), (F), and (G) show STNs generated via hierarchical agglomerative clustering (HAC), matching the partitioning levels of Row 2.
4. **Row 4:** Bar plots indicate, for each column, the number of nodes in the partitioned STNs (Rows 2–3) that contain multiple solutions from the original STN (Row 1). See the next section for interpretation. These plots provide a quantitative view of how our partitioning method organizes solutions, showing the way it groups multiple original solutions within single nodes.

Figure 12.4 (continuous problems) follows a similar structure:

1. **Row 0:** Tables with benchmark functions (left) and STN metrics (right).
2. **Rows 1–3:** Original STNs and their partitioned versions using standard (A–B) and HAC (C–D), shown under two partitioning levels.

Case 1: Minimum Dominating Set (MDS)

Given an undirected graph G , the MDS problem seeks the smallest subset D of vertices such that every node not in D is adjacent to at least one node in D . We evaluated three algorithms over 10 runs each: ALGO_1 (magenta), ALGO_2 (green), and ALGO_3 (cyan).

(B) vs. (E): Low partitioning. ALGO_1 and ALGO_3 consistently find the best solutions, unlike ALGO_2. Agglomerative clustering identifies structural similarities across solutions, grouping them into only 3 clusters, compared to the standard method which removes 96% of variables but fails to capture this.

(C) vs. (F): Medium partitioning. Agglomerative clustering condenses best solutions into 2 STN nodes, while standard partitioning still produces 8, despite removing 97% of variables. Shared nodes (gray dots) suggest convergence to a common region for ALGO_1 and ALGO_3.

(D) vs. (G): High partitioning. With 98% variable reduction, standard partitioning loses essential structural insight. Agglomerative clustering retains relevant information with only 120 STN nodes.

Notably, clustering preserves the *strength* (normalized incoming weight) of key nodes, unlike standard partitioning. Final bar plots show clustering effectively groups solutions near attractors, which standard partitioning fails to reveal.

Case 2: 2E-EVRP-TW

This logistics problem aims to optimize deliveries via electric vehicles using over 14,000 binary variables [4]. Figure 12.3 shows results for 10 runs each of ALGO_1 (magenta)

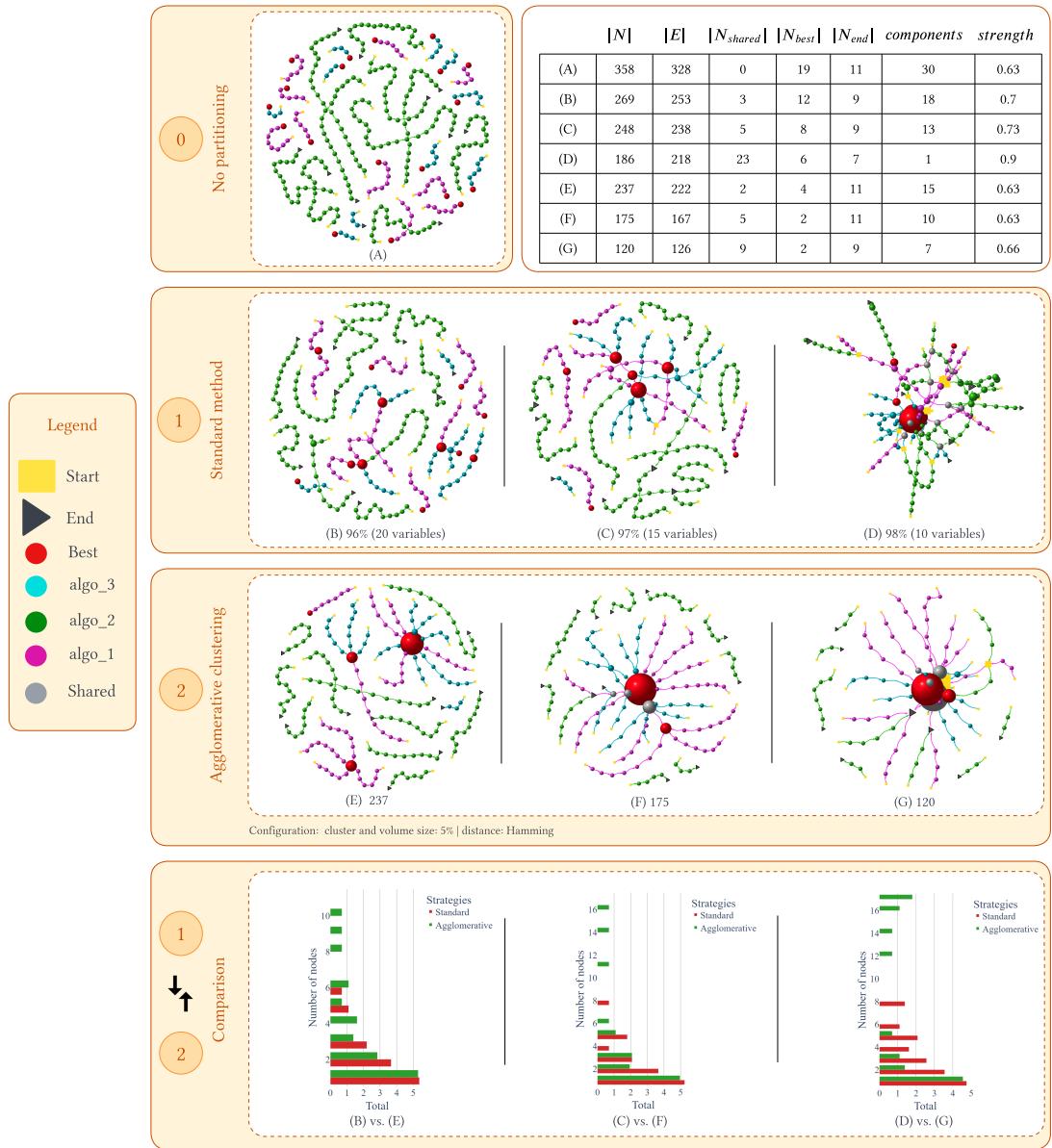


Figure 12.2: Comparison of standard partitioning vs. agglomerative clustering for MDS.

and ALGO_2 (green). In (A), ALGO_2 finds the best solution in 9 out of 10 runs, while ALGO_1 fails to reach it.

(B) vs. (E): Low partitioning. Clustering reveals trajectory overlap and proximity of ALGO_1 solutions, which is obscured by standard partitioning.

(C) vs. (F): Medium partitioning. Clustering shows attraction of ALGO_1 to a region near the optimum, even though it is not reached. Standard partitioning fails to reflect this.

(D) vs. (G): High partitioning. Clustering continues to highlight meaningful structure, whereas standard partitioning does not.

The bar plots (bottom row of Figures 12.2 and 12.3) confirm the robustness of agglomerative clustering across problem domains.

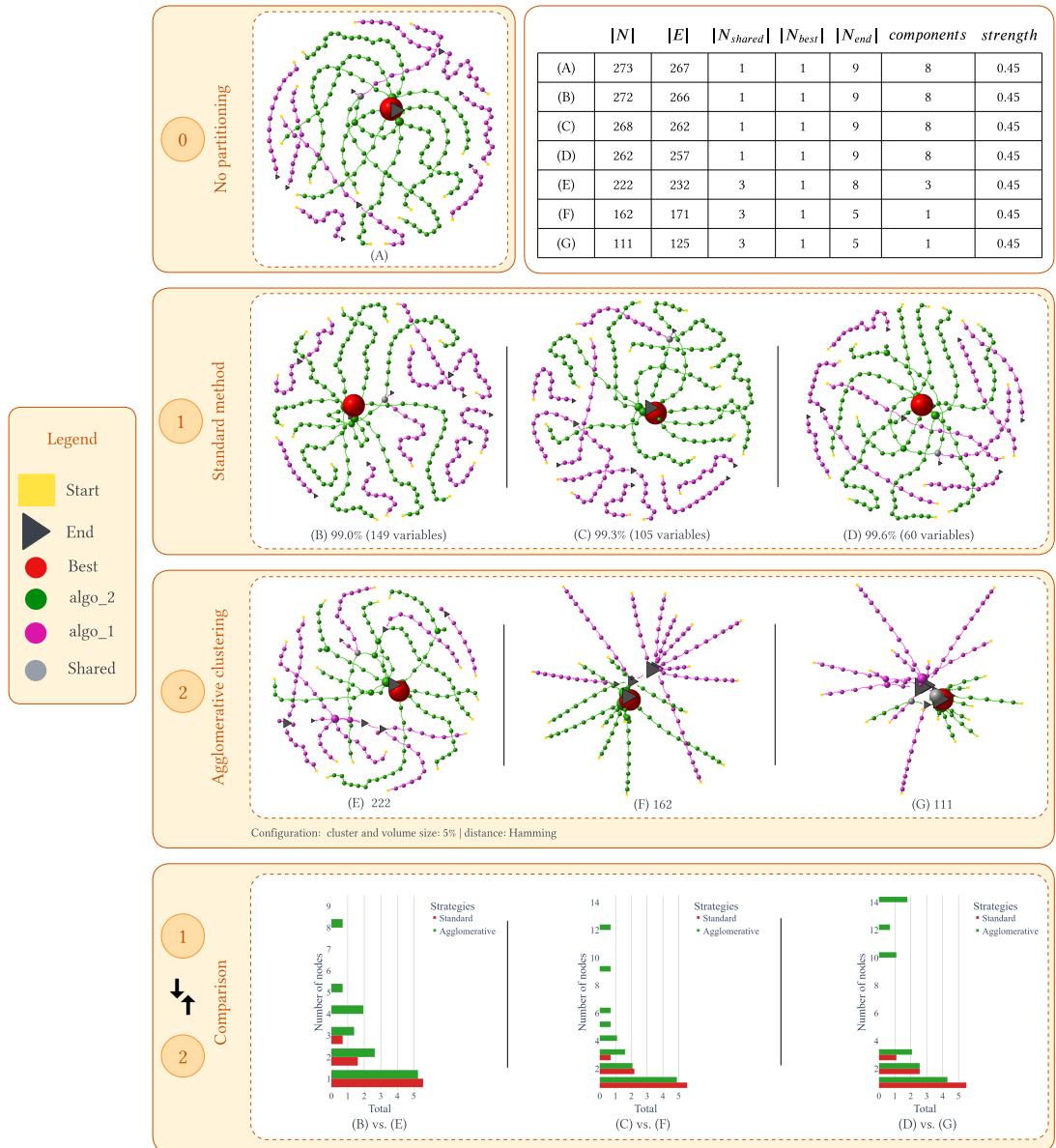
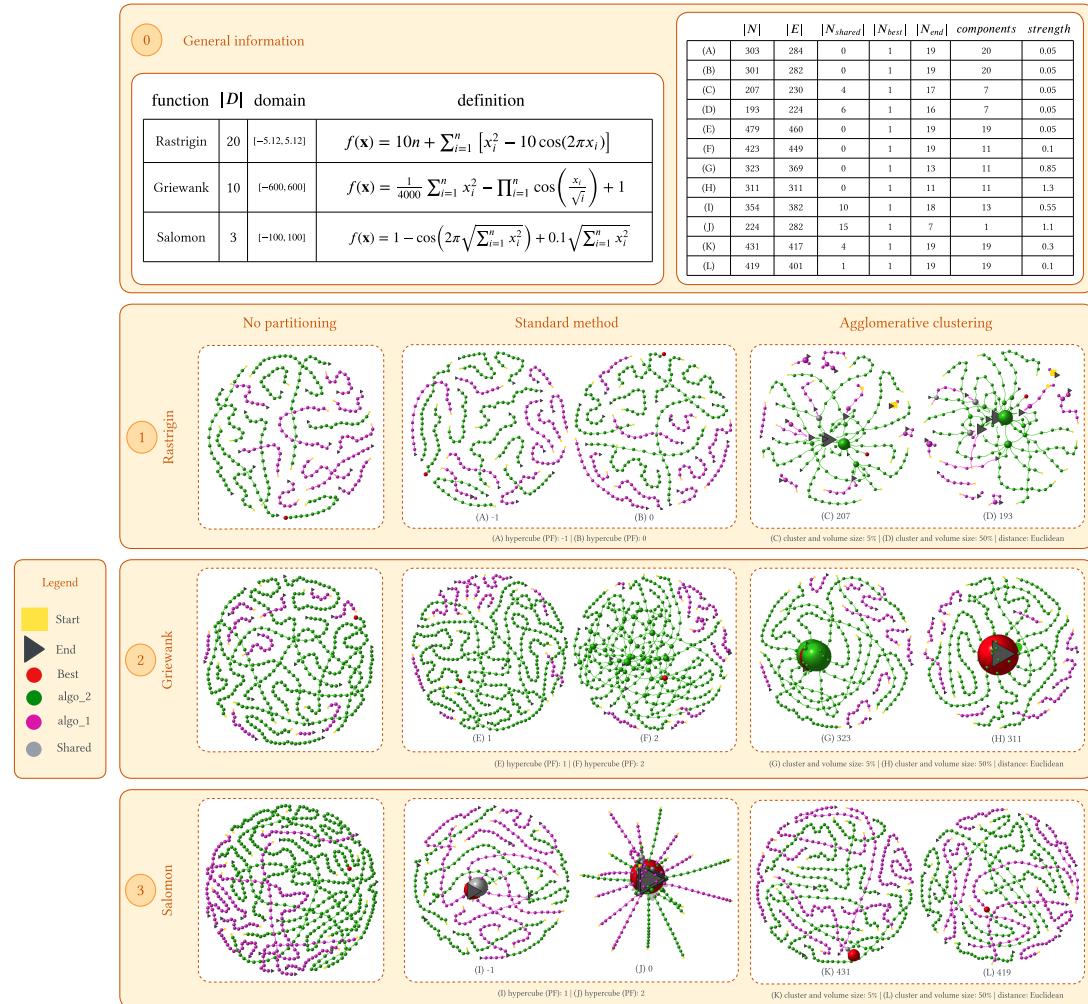


Figure 12.3: Standard search space partitioning vs. agglomerative clustering in the context of 2E-EVRP-TW results.



12.4.2 Continuous Optimization Problems

To evaluate the effectiveness of agglomerative clustering in continuous domains, we consider three well-known minimization functions: Rastrigin (20D), Griewank (10D), and Salomon (3D) [164]. Each was optimized using 10 runs of two nature-inspired algorithms, denoted as `ALGO_1` (`magenta`) and `ALGO_2` (`green`), implemented via the `Metaheuristics.jl` framework [142]. Results are shown in Figure 12.4, following the structure described in Section 12.4.1.

Case 1: Rastrigin (20D)

The Rastrigin function is highly multimodal and non-convex, with numerous local minima and a rugged search space. Standard partitioning (plots A and B; $PF = 0$ and -1) yields STNs similar to the unpartitioned baseline, indicating its limited effectiveness in high dimensions. In contrast, agglomerative clustering (C and D) reveals clear trajectory convergence of `ALGO_2` to an attractor region, highlighting its superior performance. Trajectories from `ALGO_1`, however, remain dispersed.

Case 2: Griewank (10D)

Also multimodal and non-convex, the Griewank function has fewer dimensions. Here, standard partitioning (E, F) begins to capture some structure—particularly `ALGO_2`'s trajectory overlap in (F). Yet, clustering-based partitioning (G, H) provides a clearer picture: `ALGO_2` trajectories consistently converge to the same region in the search space.

Case 3: Salomon (3D)

The Salomon function is smooth and unimodal, lacking the complexity of the previous two. This is the only case where standard partitioning (I, $PF = -1$) performs comparably or better. While `ALGO_2` still finds the best solution, standard partitioning also reveals that several `ALGO_1` runs end near that optimum—a detail lost in clustering-based STNs (K, L).

Summary: Agglomerative clustering outperforms standard partitioning in higher-dimensional, rugged landscapes—typical of real-world problems—by better revealing structural patterns in the search space.

12.5 Conclusion

This study introduced a novel search space partitioning scheme for Search Trajectory Networks (STNs), a tool designed for interpreting and analyzing optimization algorithm behavior. This new scheme, implemented in Python 3.11, was integrated into the open-source STNWeb tool, which served as our experimental platform. Our analysis, encompassing both discrete and continuous optimization problems, consistently

demonstrated the superiority of the new partitioning method across all cases (except for a unimodal smooth function), significantly enhancing STNs' interpretive capacity.

Several avenues exist for future research. Extending STN visualizations from 2D to 3D could further improve their utility for algorithm comparison. Additionally, integrating auto-generated natural language explanations could enhance user-friendliness by providing detailed insights into graph elements, thereby reducing the need for prior domain knowledge.

Note

This was the final piece of work I developed related to STNWeb during my thesis, and it was accepted at GECCO 2024 (as was the work presented in Chapter 11). It was also the last opportunity I had during my PhD to collaborate with Gabriela Ochoa, which was a truly rewarding experience.

However, as discussed in Chapter 9, although this line of research on visual tools has been a rewarding part of my doctoral journey, it also feels incomplete. I had planned to develop a fully reimplemented version of STNWeb that included a 3D visualization mode, enhancing many existing features and redesigning others from scratch. One of the key improvements was the integration of real-time, in-browser rendering using D3.js^a, which would have significantly increased plotting speed and allowed users to interactively edit the visualizations. Furthermore, moving from 2D to 3D would have enabled richer, more expressive representations by revealing additional structural insights in the algorithmic trajectories.

Unfortunately, due to the approaching end of my PhD fellowship, I ran out of time to complete it, although I managed to implement approximately 30% of the planned improvements. Moreover, my focus had increasingly shifted toward research involving Large Language Models (see Chapters 5 to 7), which I found intellectually more stimulating at that stage of my doctoral work.

I hope to find time in the future to substantially improve STNWeb!

^a An open-source JavaScript library for visualizing data. <https://d3js.org/>

13

A Benchmark Generator for Assessing Variability in Graph Analysis Using Large Vision-Language Models

Foundational Work for This Chapter

This chapter is based on the following publication:

- **Title:** *VisGraphVar: A Benchmark Generator for Assessing Variability in Graph Analysis Using Large Vision-Language Models*
- **Published in:** IEEE Access
- **Type:** Journal Paper
- **Year:** 2025
- **Main contribution:** A benchmark generator to evaluate Large Vision-Language Models (LVLMs) on graph analysis tasks
- **Problem addressed:** Are LVLMs capable of analyzing graph images?
- **Type of contribution:** Tool/software & Experimental & Methodological
- **DOI:** <https://doi.org/10.1109/ACCESS.2025.3535837>

13.1 Introduction

This research emerged after completing the work presented in Chapter 5, and upon the suggestion of my supervisors, after discussing the need to advance in the automatic interpretation of STNWeb plots (as explored in Chapter 11). This time, however, the idea was to use generative multimodal models to analyze the STNWeb graphical directly. Therefore, before applying fine-tuning to a multimodal model to analyze STNWeb graphs (which are directed graphs), we decided to first examine how multimodal models behave when dealing with different types of graphs. This led to the

idea of creating VisGraphVar, a benchmark generator designed to test Large Vision-Language Models (LVLMs), which we will explain in this chapter.

It is worth mentioning that I did not manage to perform the fine-tuning needed to automatically analyze STNWeb plots, which might have led to a new journal article. As a result, this work remains somewhat “disconnected” within the context of my thesis. Nevertheless, it allowed me to discover that it is indeed possible to use LVLMs within STNWeb!

One of the most challenging domains for Large Vision-Language Models (LVLMs) is the analysis of geometric structures, particularly graphs. The complexity of graph-based data arises from its structural flexibility—nodes and edges can be arranged in countless configurations, resulting in highly diverse and potentially intricate patterns. An LVLM’s ability to interpret such visual graph representations could be transformative, enabling solutions to complex problems in graph theory with real-world relevance. Potential applications span the analysis of information propagation in social networks, communication flow in distributed systems, metabolic pathways in biology, routing for robotics and autonomous vehicles, circuit design, and the modeling of dynamic processes. This broad applicability highlights the fundamental role of graphs across a wide range of scientific and engineering domains [83].

This research aims to address two central questions:

1. **Primary question:** How robust are LVLMs in interpreting visual representations of graphs?
2. **Secondary question:** To what extent do visual stylistic choices—such as node coloring, labeling, or layout—affect LVLM performance?

To explore these questions, we introduce **VisGraphVar** (**V**isual **G**raph **V**ariability), a benchmark generator designed to create graph images with controlled variation in both structure and style. We developed VisGraphVar under the principle that a benchmark is only as effective as the lessons it incorporates from prior work. While existing benchmarks typically focus on narrow tasks (e.g., visual reasoning [221]) or ignore stylistic variability altogether [225, 118], VisGraphVar adopts a multidimensional evaluation framework. Drawing inspiration from [37], we define seven distinct dimensions, each corresponding to a specific task in visual graph interpretation:

1. **Node and Edge Detection** — *How many nodes and edges are present in the graph?*
2. **Graph Type Classification** — *What type of graph is depicted?*
3. **Segmentation** — *Which edges constitute the graph’s cut-edges?*
4. **Pattern Recognition** — *Which structural patterns can be identified, and how many instances of each exist?*
5. **Link Prediction** — *Which pairs of nodes are likely to form future connections?*
6. **Reasoning** — *What is the shortest path between two given nodes?*

7. **Matching** — Are two displayed graphs structurally identical, disregarding differences in color or layout?

Through this structured and comprehensive approach, *VisGraphVar* offers a rigorous evaluation framework that captures the strengths and limitations of current LVLMs in handling visual graph tasks. By addressing multiple levels of interpretation, it provides nuanced insights into how these models perform across different aspects of graph understanding.

13.1.1 Contributions

Our main contributions—and what differentiates *VisGraphVar* from existing benchmarks and benchmark generators—can be summarized as follows:

- *VisGraphVar* is highly configurable and user-friendly, allowing researchers to generate datasets that uncover specific weaknesses in LVLMs when applied to graph-based visual tasks.
- Unlike most prior work that focuses on a single task, *VisGraphVar* spans seven distinct tasks, enabling more comprehensive evaluations of model capabilities across varied graph-related challenges.
- *VisGraphVar* emphasizes realism by introducing imperfections commonly found in practical visualizations, such as overlapping nodes or occluded edges. This design choice ensures that evaluations better reflect the real-world conditions under which models are expected to perform—conditions that humans typically navigate with ease.

In short, *VisGraphVar* extends prior efforts by integrating spatial layout variability (e.g., node positioning and edge routing) and stylistic diversity (e.g., color schemes, label formats). We demonstrate that such visual factors significantly influence model performance across the seven tasks described above, revealing important limitations and promising directions for future LVLM research in graph analysis.

13.2 *VisGraphVar*: A benchmark generator

We present *VisGraphVar*, a customizable benchmark generator for evaluating LVLMs in multimodal graph analysis. It targets a key challenge: *representational variability*—how changes in visual style and structure affect model performance. By spanning seven distinct tasks, *VisGraphVar* enables a detailed, task-specific assessment of LVLM robustness in visual graph understanding.

13.2.1 A Custom Synthetic Dataset

VisGraphVar was developed in Python 3.11 using NetworkX for graph generation. Customizability and extensibility—core principles of effective benchmark design [105]—

are supported through a modular architecture. The visualization pipeline offers the following key configuration options:

1. **Layout Selection.** Unlike prior benchmarks that rely on default or aesthetic layouts, *VisGraphVar* emphasizes layout variability, as spatial arrangements can affect LVLM inferences. For example:
 - *Spring layouts* highlight centrality,
 - *Circular layouts* clarify disconnected components,
 - *Spectral layouts* reveal community structures.

Layouts accentuate different graph properties, and LVLMs may yield inconsistent results across them—raising concerns about layout sensitivity in practical scenarios [18].

2. **Graph Complexity.** Users can adjust node counts and edge densities to evaluate how increasing visual complexity affects LVLM performance.
3. **Stylistic Variation.** Color schemes and text labels are configurable, enabling assessment of whether LVLMs can interpret subtle visual cues—an aspect often overlooked in other benchmarks.

While GITA [221] includes some visual graph attributes, its evaluation is limited to reasoning tasks. In contrast, *VisGraphVar* supports a broader range of styles and tasks, providing a more comprehensive framework for testing both current and future LVLMs.

13.2.2 Tasks

Limited benchmarks risk overlooking critical technology shortcomings. To address this, *VisGraphVar* implements seven comprehensive evaluation tasks (Figure 13.1), each assessing distinct aspects of Large Vision-Language Model (LVLM) graph comprehension while maintaining customization compatibility. Below, we explain each task and its rationale.

Task 1: Node and Edge Detection

Proficient LVLMs for graph analysis must accurately detect and count nodes and edges as a prerequisite for complex reasoning. To rigorously test this, we incorporated diverse stylistic variations (Figure 13.2), including layouts, arrows, labels, and node colors. An effective LVLM should strictly adhere to prompt instructions, ensuring attributes like labels, edge directionality, or colors do not influence counts. However, as shown later, not all LVLMs meet this criterion.

Visual benchmarks often assume flawless representations [221, 118]. In reality, visual imperfections, common in large networks, can be intentional or unintentional. Since humans adapt to these defects, *VisGraphVar* deliberately incorporates them. Node/edge overlap exemplifies these challenges. We expect LVLMs to closely approx-

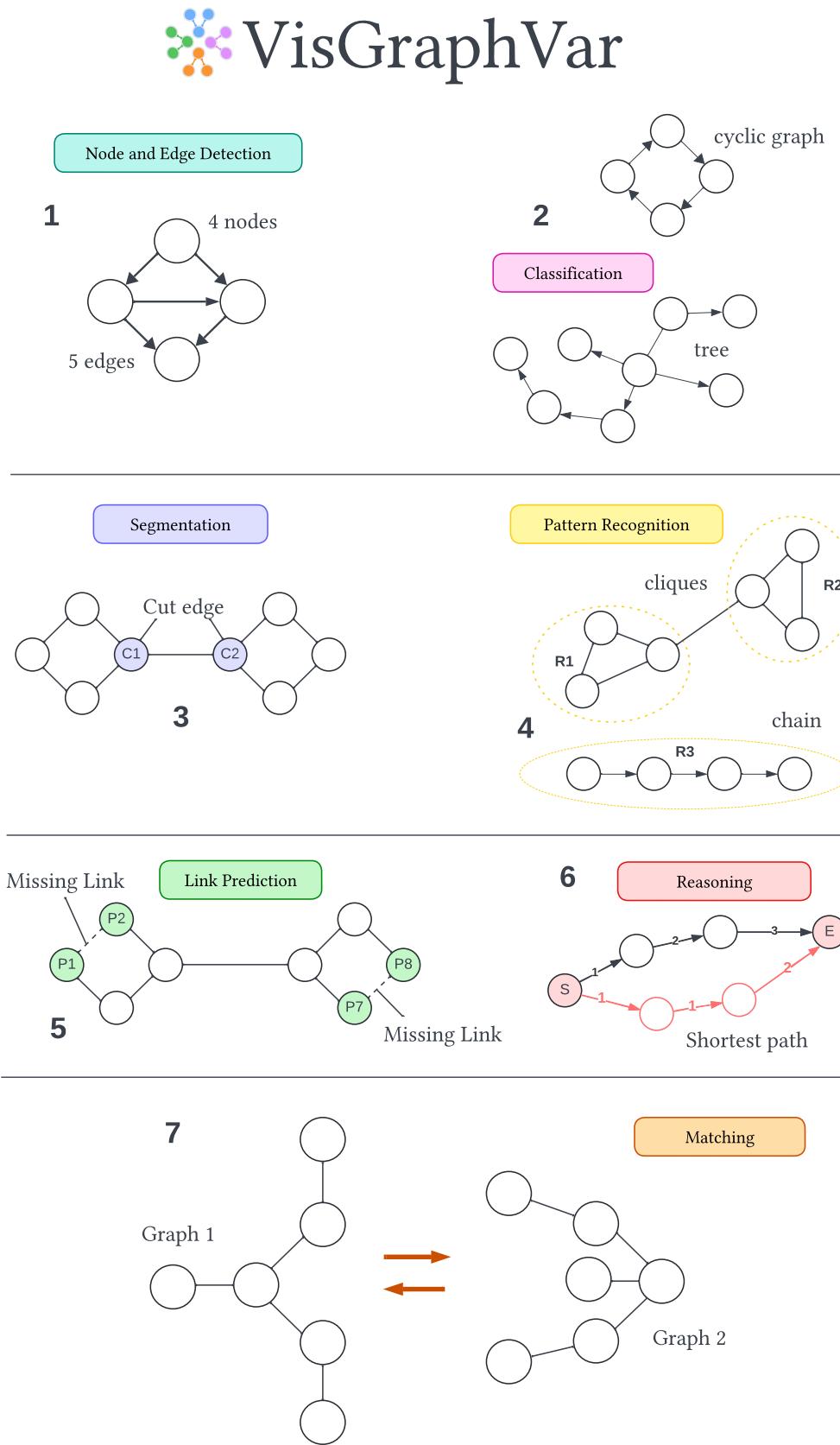


Figure 13.1: A general overview of the seven tasks covered by VisGraphVar (1-7), each representing a different challenge for LVLMs, enabling us to conduct a more detailed performance comparison and evaluation.

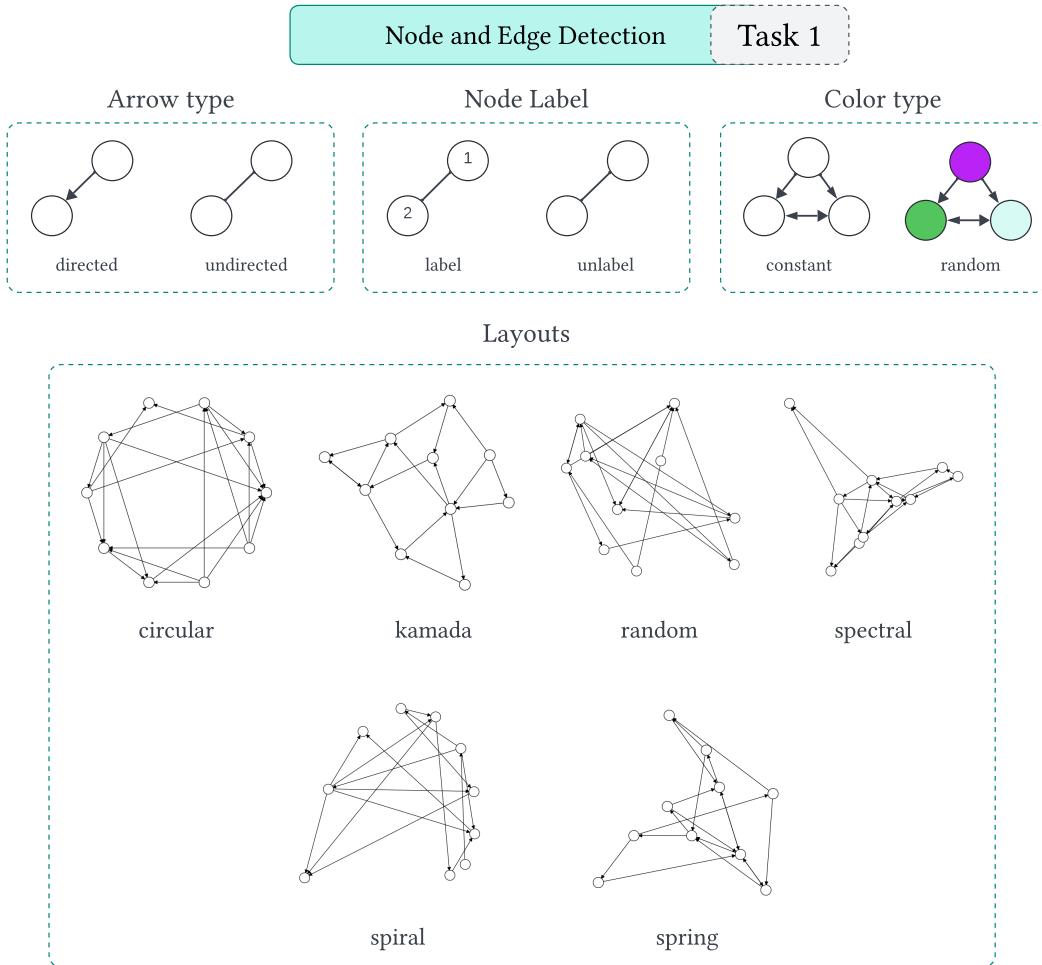


Figure 13.2: Available configurations for generating graph images to evaluate node and edge detection capabilities.

imate actual element counts even with overlap. For instance, detecting 9 nodes from 10 with three overlapping is better than identifying 8. This mirrors human visual processing, where we estimate overlapping elements. LVLMs should match or exceed this, especially with slight overlap (Figure 13.3), as these imperfections critically test a visual model's robustness.

Task 2: Classification

Beyond element detection, LVLMs must classify graph types. This complex task requires global image analysis, considering edge positions and directions to recognize broader patterns. Such capabilities are crucial for comprehensive graph analysis. This is essential, as demonstrated by LVLMs interpreting various chart types [140, 91] and in real-world applications like medical image classification [86]. Classification is fundamental for any robust vision model.

To evaluate LVLM proficiency in understanding higher-order node/edge relation-

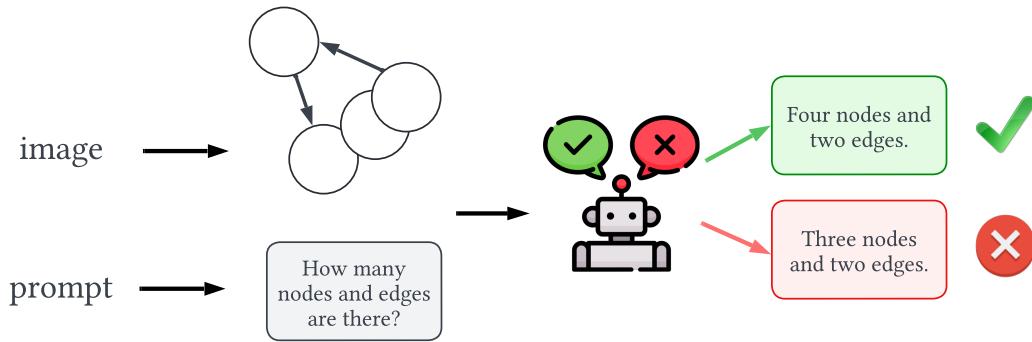


Figure 13.3: LVLM execution of Task 1 with overlapping nodes and prompt input.

ships, we assess its ability to classify seven fundamental graph types (Figure 13.4):

1. **Acyclic graphs:** Graphs without loops.
2. **Cyclic graphs:** Graphs containing loops.
3. **Bipartite graphs:** Nodes partitioned into two sets, with edges only between sets.
4. **Complete graphs:** Every node connected to all others.
5. **Meshes:** Regular, grid-like structures.
6. **Planar graphs:** Graphs drawable without edge crossings.
7. **Trees:** Hierarchical graphs branching from a root, without cycles.

Each graph type presents unique analytical challenges. Our selection includes clearly distinct and subtly different graphs to enhance analysis thoroughness. Identifying cycles requires tracing paths; recognizing bipartite structures demands understanding node groupings; tree classification requires comprehending hierarchical relationships. LVLMs must possess these analytical capabilities to accurately classify graph structures and understand their inherent properties, forming a crucial foundation for complex visual reasoning.

Unlike VisionGraph's focus on cycle detection within reasoning tasks [118], our benchmark generator offers six additional graph types for classification. This difference arises because VisGraphVar reserves reasoning tasks primarily for Task 7 (Section 13.2.2).

Task 3: Segmentation

Beyond classification, LVLMs must identify critical graph regions, such as cut-edges (bridges). A cut-edge, whose removal increases connected components [76, 17], effectively segments the graph. As Camilus et al. [28] stated, "Image segmentation can simply result in an image partition composed of relevant regions." Dividing a graph into relevant regions has numerous applications, including analyzing connectivity [212], optimizing routing [146], and identifying network failure points [106].

A main challenge is accurately detecting cut-edges. As nodes increase, identify-

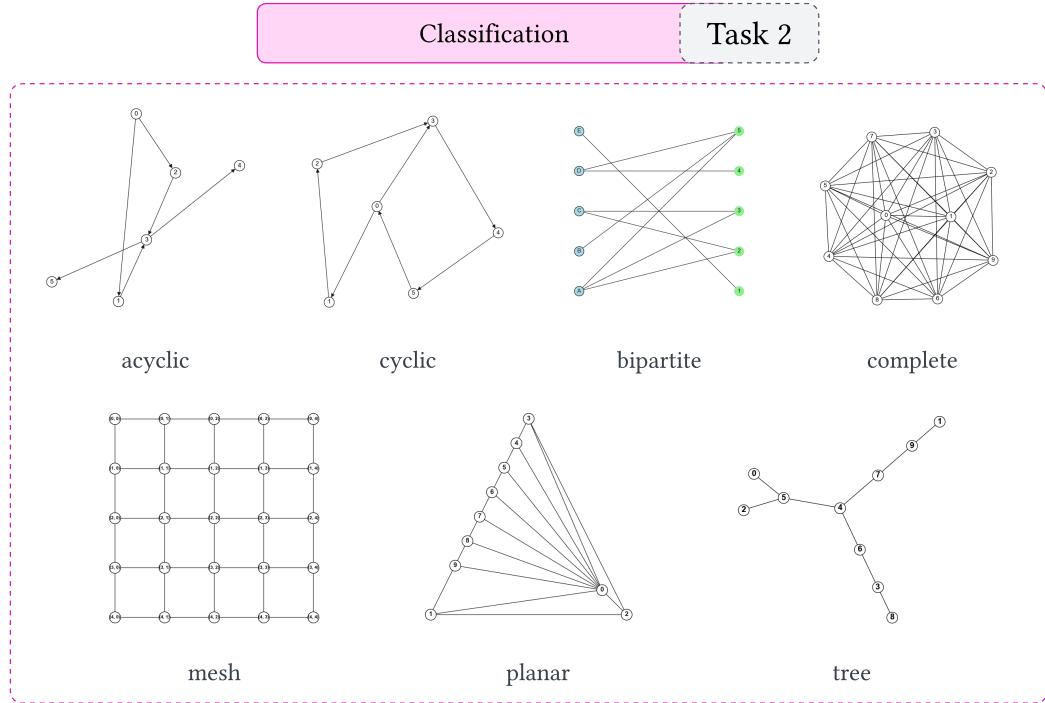


Figure 13.4: Seven different types of graphs.

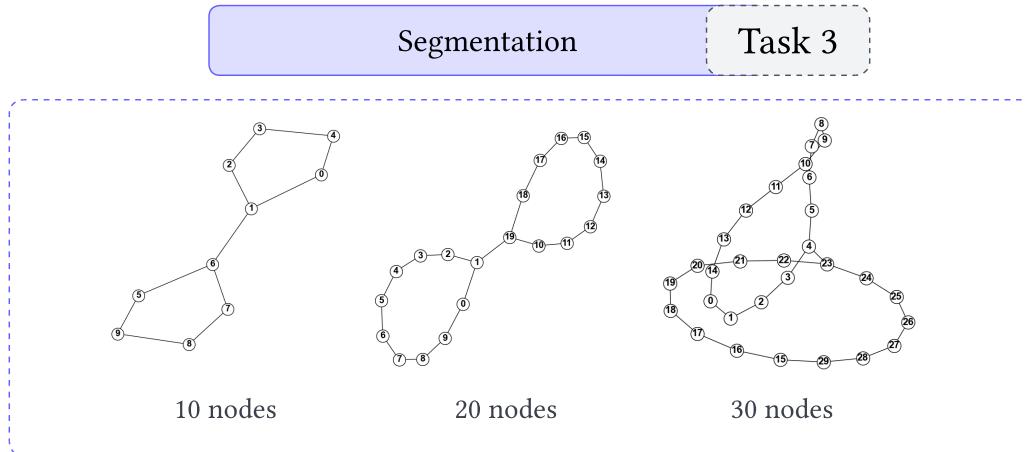


Figure 13.5: Networks with an increasing number of nodes and a single cut-edge: the graph on the left has cut-edge (6, 7); the one in the center has (1, 19); and the most complex one to detect, on the right, has (4, 23).

ing cut-edges becomes harder due to overlaps (Figure 13.5), challenging even humans. Thus, a rapid visual method to pinpoint bridge nodes would greatly aid analysis. An LVLM correctly detecting a bridge demonstrates its ability to analyze the graph globally and identify specific nodes whose removal increases connected components. This signifies a nuanced understanding of both overall structure and critical connectivity points.

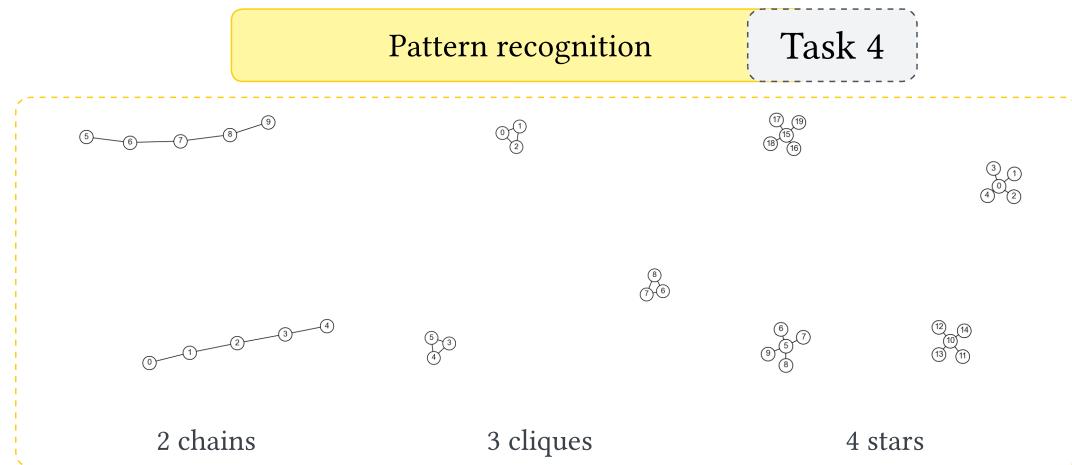


Figure 13.6: Three graphs with different types of patterns.

Task 4: Pattern Recognition

While cut-edge identification requires precise global analysis, recognizing and counting specific graph patterns introduces higher complexity. Task 4 demands more than a surface scan; the LVLM must accurately recognize, classify, and count each unique node/edge pattern (Figure 13.6). This builds on Task 2 (classification, Section 13.2.2) with added depth, particularly useful for disconnected graphs where understanding isolated cluster structures is crucial.

Recognizing patterns in mathematical graphs is computationally demanding [189]. LVLMs could reduce this if patterns are efficiently identified via image analysis. This task also tests LVLM memory retention [244]. Accurately counting patterns requires storing detected structures and maintaining a tally. A less powerful LVLM might only identify a single pattern or fail to distinguish types. Thus, this task tests initial pattern recognition and subsequent accurate tracking.

Task 5: Link Prediction

Beyond memory and pattern recognition, link prediction is a crucial reasoning task in graph analysis. Graph structure and node/edge positioning often suggest missing connections. By analyzing topology, an LVLM can identify nodes with similar structural patterns that could logically be connected [138].

Interestingly, link prediction complexity doesn't always increase with node count (Figure 13.7). Smaller graphs can be harder due to fewer examples of existing connections, making replicable pattern identification difficult. Link prediction fundamentally differs from other graph analysis tasks, requiring deeper LVLM reasoning. Instead of just recognizing or memorizing, the model must:

- Analyze global graph structure.
- Identify similar topological patterns across node pairs.
- Make logical inferences about potential connections.

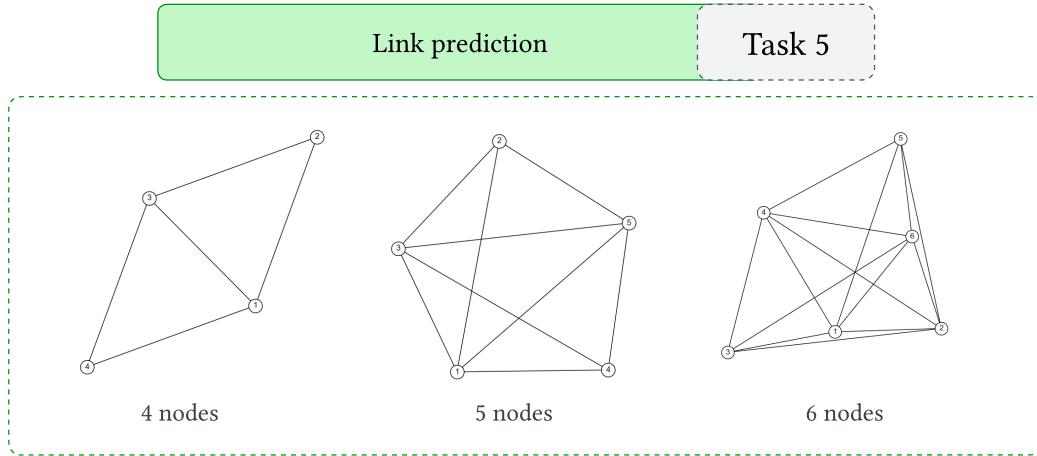


Figure 13.7: Three types of graphs with different numbers of nodes for which the LVLM is expected to predict a missing link/edge. The missing link on the left is (4,2), the one in the center is (2,4), and the most complex case is (3,5), on the right.

- Consider local and global network characteristics.
- Apply structural similarity principles to predict missing links.

Current graph link prediction research typically uses abstract mathematical objects, not visual representations [242, 116]. The advanced reasoning required makes visual link prediction a valuable benchmark for evaluating an LVLM’s ability to understand and reason about complex network structures from their visual depictions.

Task 6: Reasoning

While link prediction demands reasoning, asking an LVLM to apply an algorithm to an image-displayed graph significantly elevates difficulty. This requires genuine algorithmic reasoning, not just visual pattern recognition. For example, shortest path finding involves identifying nodes, detecting edge directions, accurately storing weights, and determining the shortest path. This task rigorously tests LVLM reasoning and memorization.

This task also necessitates numerical analysis of visual information. LLMs generally struggle with complex reasoning due to statistical inferences from training data [87]. This is more pronounced for LVLMs, which must simultaneously process and reason about numerous visual elements, understanding their interrelationships and integrating this with textual input. New prompting techniques like self-reflection [178] improve reasoning results [134] and have been applied to LVLMs [46].

Figure 13.8 shows examples of shortest path finding using VisGraphVar-generated graphs with weighted edges and labeled nodes. Given a source-target pair, the LVLM determines the shortest node sequence. Cases (a) and (b) are easier for humans, though (b) is slightly harder. Case (c) presents significant visual complexity with overlapping elements, likely leading to LVLM errors (and human errors). This case is

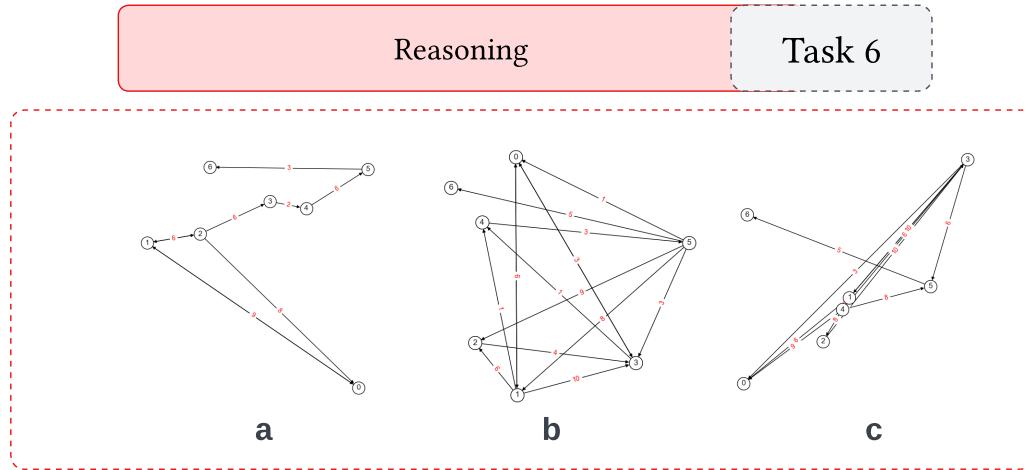


Figure 13.8: Three graphs with varying levels of interpretive difficulty in identifying shortest paths. (a) and (b) are simpler due to the lack of overlap between nodes and edges, whereas (c) makes it very hard to locate each node along a shortest path due to element overlap.

invaluable for assessing LVLM behavior in extreme conditions, illustrating how visual complexity impacts both human and LVLM performance in graph analysis.

VisionGraph [118] includes more reasoning tasks than VisGraphVar. However, our results (Section 13.3) show even top LVLMs struggle with simpler tasks like shortest path finding. Thus, adding more complex reasoning tasks may not be justified until foundational tasks are robustly handled.

Task 7: Matching

Determining if two graphs match is fundamental in graph theory [29]. The LVLM must identify both graphs' topologies and accurately align elements to map relationships. This has broad applications in neuroscience [143], computational biology [56], computer vision [198], and machine learning [117].

Unlike previous tasks, matching involves simultaneously analyzing two graphs for structural similarity. In this research, graphs match if their structure (including node labels) is identical, regardless of display style. They do not match if structures differ, even if isomorphic (Figure 13.9, right). Core LVLM verification steps for graph matching include:

- Analyzing overall graph structure.
- Comparing node and edge counts.
- Verifying equivalent connected nodes by labels.
- Accounting for edge directionality.
- Omitting stylistic details (colors, label size, layout) irrelevant to topology.

As Figure 13.9 shows, an LVLM should ideally filter out visual details irrelevant to topology (colors, sizes, layout). When processing the image, the LVLM must focus on

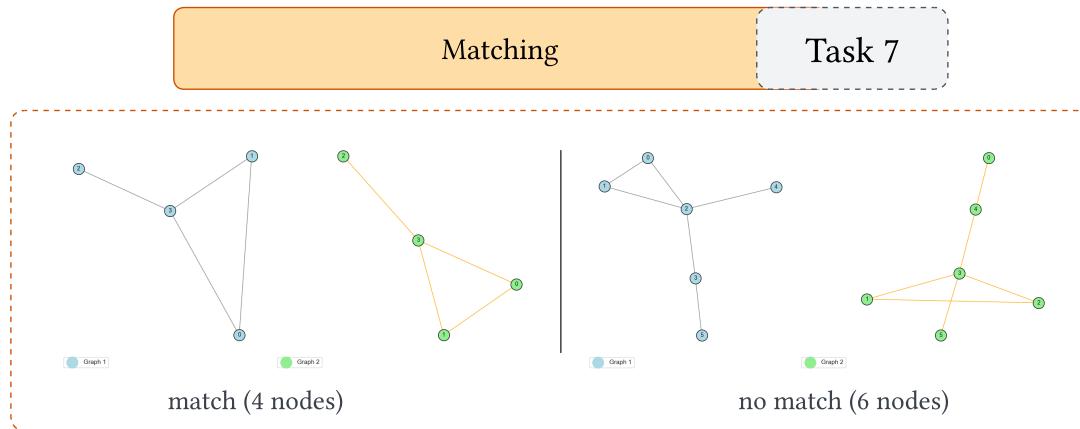


Figure 13.9: Graph pairs are shown with the goal for the LVLM to identify matches on the left and distinctions on the right. Note that, in this work, two graphs in the same image are said to match if their structure (including node labels) is equal; that is, only their display style might differ. For this reason, the two graphs on the right do not match, even though they are isomorphic.

the prompt, which clarifies the matching task, guiding it to prioritize relevant structural information over superficial visual cues.

13.2.3 Dataset Configuration and Statistics

Using `VisGraphVar`, we generated a dataset of 990 graph images (600×600 px) across seven tasks. Table 13.1 summarizes the image distribution per task. The number of images varies depending on task complexity and visual configuration settings in `VisGraphVar`.

Table 13.1: Dataset distribution by task.

Task No.	Task	Images
1	Detection	560 (56.57%)
2	Classification	70 (7.07%)
3	Segmentation	30 (3.03%)
4	Pattern Recognition	210 (21.21%)
5	Link Prediction	30 (3.03%)
6	Reasoning	30 (3.03%)
7	Matching	60 (6.06%)

Below is a brief overview of each task's configuration:

- **Task 1 (Detection):** 560 images with 10-node graphs and varied styles (colors, arrows, layouts, labels). Edges are added with 2% probability between node pairs.
- **Task 2 (Classification):** 70 images covering 7 graph types. Node count ≤ 10 ; edges depend on the graph type.

- **Task 3 (Segmentation):** 30 images with 10, 20, or 30 nodes. Two subgraphs connected by a cut-edge.
- **Task 4 (Pattern Recognition):** 210 images featuring chains, cliques, or stars (2–4 per image).
- **Task 5 (Link Prediction):** 30 images of complete graphs (4–6 nodes) with one missing edge.
- **Task 6 (Reasoning):** 30 graphs (5–7 nodes), random layout, edges added with 3% probability. Edge weights between 1 and 10.
- **Task 7 (Matching):** 60 paired-graph images (4–6 nodes), with or without structural equivalence. Edges added with 4% probability.

For each variation in visual style, 10 images were generated per task setting, enabling metric averaging (see Section 13.2.4). Given its foundational role, Task 1 was extensively diversified to test the sensitivity of LVLMs to stylistic variation.

13.2.4 Metrics

As described in the previous section, each visual combination is evaluated using a set of 10 images. We employ three different metrics, which differ by task, so not all tasks share the same evaluation criteria. Each metric is normalized on a scale from 0 to 1, where 1 indicates optimal performance (complete alignment with the ground truth), and 0 indicates the lowest performance (no alignment with the ground truth). Further details are provided below.

Mean absolute error (MEA). In Task 1, we used the MAE because we are interested in knowing the degree of error in predicting the number of nodes and edges that the model infers from the image.

$$\begin{aligned}
 \text{MAE} &= \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i| \\
 \text{NMAE} &= 1 - \min \left(\frac{\text{MAE}}{\text{Range}}, 1 \right) \quad \text{, where} \\
 \text{Range} &= \begin{cases} \max(y) - \min(y) & \text{if } \max(y_i) \neq \min(y_i) \\ 1 & \text{if } \max(y_i) = \min(y_i) \end{cases}
 \end{aligned} \tag{13.1}$$

Hereby, y_i is the actual number of elements (either nodes or edges) that appear in the image, while \hat{y}_i is the value predicted by the model. Moreover, NMAE is the normalization of MAE—using Range—to ensure that predictions closer to the true values approach a value of 1, while those further away approach 0.

Accuracy. Following Zou et al. [250], we assess model responses in tasks 2–5, and 7 using an accuracy metric, but we extended it to account for both exact and partial

matches. The accuracy is calculated as:

$$\text{Accuracy} = \frac{1}{n} \sum_{i=1}^n m_i \quad , \text{where}$$

$$m_i = \begin{cases} 1 & \text{if } y_i = \hat{y}_i \\ 0.5 & \text{if } y_i \text{ partially matches } \hat{y}_i \\ 0 & \text{otherwise} \end{cases} \quad (13.2)$$

This formulation allows us to account for partial correctness in tasks where a prediction may be partially correct. For instance, in task 5 (link prediction), an accuracy score of 0.5 is assigned if only one of the two nodes of the missing edge is correctly predicted by the model. The same principle applies to other tasks where partial matches are possible.

Jaccard Index. Task 6 (reasoning) requires the model to find a shortest path regarding a sequence of nodes. Therefore, we apply the Jaccard Index to provide a similarity value between the actual nodes that form a true path with the path predicted by the model.

$$\text{Jaccard Index} = \frac{|P_{\text{true}} \cap P_{\text{pred}}|}{|P_{\text{true}} \cup P_{\text{pred}}|} \quad , \quad (13.3)$$

where P_{true} represents the set of nodes in the true path and P_{pred} represents the set of nodes in the predicted path. That way, even if a model only correctly predicted four nodes of a five-node path, it would have a higher score than a model that only correctly predicted two nodes out of a five-node path.

13.2.5 Prompt design

Like all multimodal models, LVLMs require both an image and a text prompt to operate. We will be testing two different prompting approaches: (1) zero-shot, where the model makes direct predictions without examples) [107]; and (2) chain-of-thought, where the model explains its reasoning step by step [220]. Vatsal et al. [210] identified up to 39 prompting strategies available for natural language processing tasks; however, many of these are derivatives of the zero-shot and chain-of-thought approaches, which justifies our selection. Thus, this approach requires two distinct prompts for each task, resulting in 14 prompts in total. To streamline this process, we developed a single basic prompt per task and then used an LLM to create two versions of each—one using the zero-shot format and another using the chain-of-thought format. We use these two prompt versions alongside each image to test all considered LVLMs. Moreover, we request all model responses in JSON format (as specified in the prompts) to facilitate a later analysis.

13.3 Experiments and Evaluation

In this section, we evaluate six state-of-the-art LVLMs across seven tasks using the dataset described before and generated by `VisGraphVar`. More specifically, Section 13.3.1 details our experimental setup, execution environment, and the rationale behind our LVLM selection. Finally, we present our quantitative analysis in Section 13.3.2, followed by qualitative observations in Section 13.3.3.

13.3.1 Environment Setup and LVLM Configuration

We evaluated six top-ranked LVLMs using the 990 graph images from our dataset (see Section 13.2.3), categorized into seven tasks. Models were accessed via the OpenRouter API¹, which enables streamlined multi-model querying.

Model selection was based on the October 2024 Chatbot Arena Vision Leaderboard², and includes: GPT-4o [228], Gemini-Pro-1.5 [204], Claude-3.5-Sonnet [202], Llama-3.2-90B-Vision-Instruct [60], Qwen-2-VL-72B-Instruct [234], and Pixtral-12B [1]. The leaderboard, built by UC Berkeley’s LMSYS and SkyLab, aggregates over 130,000 human votes across 38 LVLMs.

Each model was tested using two prompting strategies—zero-shot and chain-of-thought—yielding 1980 evaluations per model (990 images per 2 prompts), and 11,880 evaluations in total. All runs used OpenRouter’s default parameters.

13.3.2 Results

In this section, we present a detailed comparative analysis of the obtained results. These results are shown regarding the utilized metrics (see Section 13.2.4). However, note that, for the sake of a better understanding, metric scores are shown in terms of percentages. Figures 13.12 and 13.13 provide summarized results to easily identify the leading models across all tasks. Additionally, Figure 13.14 shows the impact of prompt strategies on the results.

Task-Specific Performance Analysis

In Figure 13.10, we observe that Claude-3.5-Sonnet, Gemini-Pro-1.5, and GPT-4o perform similarly across most tasks (70%-80% overall accuracy rate). However, significant differences are observed in Task 7 and Task 2, where Claude-3.5-Sonnet and GPT-4o outperform Gemini-Pro-1.5. In contrast, Gemini-Pro-1.5 and GPT-4o exhibit a slightly lower performance in Task 1 and Task 3 when compared to Claude-3.5-Sonnet. Notably, Gemini-Pro-1.5 clearly outperforms all other models in Task 3 and shows a slight advantage in Task 1. These results highlight the general advantage of proprietary models over open-weight alternatives.

¹ <https://openrouter.ai>

² <https://lmarena.ai>

An interesting observation is that all models show a rather high performance on Task 4 when compared to their performance on other tasks. Moreover, another eye-catching finding is Qwen-2-vl-72B's performance in Task 6, where it nearly matches the top three models. Another significant observation concerns Llama3.2-90B, which, despite having substantially more parameters than Qwen-2-VL-72B and especially Pixtral-12B, exhibits markedly lower performance than both models and ranks below all other tested models. This outcome confirms the suspicion that simply increasing the number of model parameters does not necessarily lead to improved performance in visual analysis tasks; in fact, as noted by [McKenzie et al. \[141\]](#), inverse scaling may occur due to flaws in the training objective and data.

Performance Distribution Analysis by Task

Figure 13.11 employs violin plots to visualize the performance distribution patterns across our evaluated models for all seven tasks. The visualization combines two key elements: individual points representing each model's average performance score, and varying plot widths that indicate the density of scores at each performance level. This dual representation enables comprehensive analysis of both individual model performance and the overall distribution patterns across different tasks.

The green violin plots for Tasks 1, 5, 6, and 7 exhibit a narrow and condensed shape, with an approximate score distribution height of $\sim 20\%$ on the y-axis. This indicates that the six models performed consistently and with a lower variation on these tasks. The concentrated distribution suggests that the models' responses were more homogeneous and closely aligned for these particular tasks. In contrast, purple violin plots for Tasks 2, 3, and 4 are wider and more dispersed, with an approximate score distribution height ranging from $\sim 40\%$ to $\sim 60\%$ on the y-axis. The increased width and height of these violin plots signify greater performance variability among the six models for these tasks. The heterogeneous distribution implies that the models generated diverse responses and exhibited varying performance levels on Tasks 2, 3, and 4.

Task 4 reveals a notably asymmetric distribution pattern characterized by concentrated performance scores in the upper range, with five LVLMs demonstrating robust performance ($\geq 80\%$). However, the presence of a single outlier at $\sim 50\%$ introduces significant dispersion into the distribution, creating a clear performance dichotomy among the evaluated models.

These variations demonstrate VisGraphVar's capability to capture diverse model behaviors, attributable to its comprehensive task design that spans multiple aspects of visual analysis.

Aggregate Performance Evaluation

In Figure 13.12, it is shown that multimodal models like Claude-3.5-Sonnet, Gemini-Pro-1.5, and GPT-4o exhibit similar performance across all tasks, but with a slight advantage of Claude-3.5-Sonnet. Moreover, there is a significant performance gap of

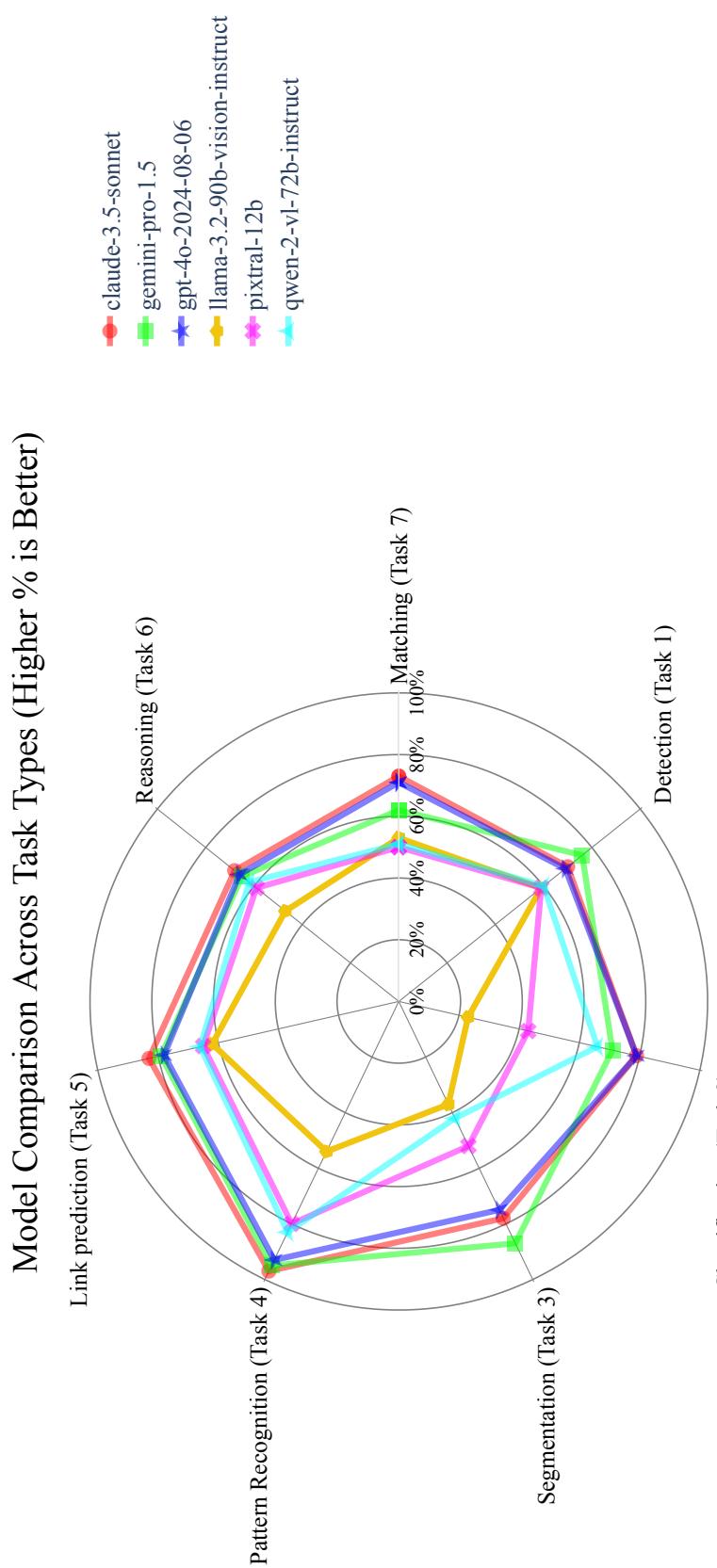


Figure 13.10: An overview of LLM performance across the seven tasks (complete dataset).

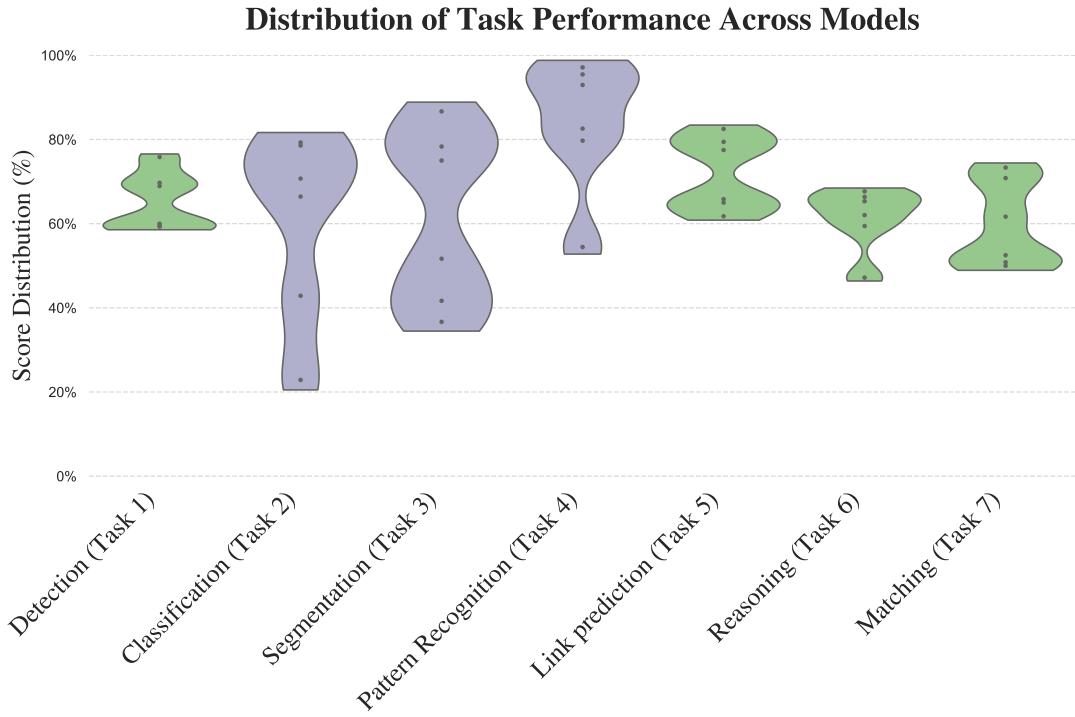


Figure 13.11: The distribution of average scores across the six LVLMs for each task. The violin plot is configured with `bw_adjust = 0.5` (which adjusts the bandwidth of the kernel density estimation, making the plot more detailed) and `cut = 0` (which ensures the plot is limited to the range of the data without extending beyond it) using the Seaborn library in Python.

30.47% between the top model, Claude-3.5-Sonnet, and the model with the weakest performance, Llama3.2-90B. This trend aligns with the fact that all three top models are proprietary models, further confirming that, at present, these models outperform open-weight models in visual tasks. This confirms the findings of [78], which demonstrated that there is currently no way for an open-weight model to match the performance of a proprietary model without improvements to its underlying base language model.

Figure 13.13 shows that Claude-3.5-Sonnet exhibits a mixed performance across tasks. It excels in Task 4 but shows a relatively lower performance in Tasks 1 and 6. Only for Tasks 4 and 5 an average accuracy of over 80% performance is obtained. These results indicate that, except for Task 4, future LVLMs have significant room for improvement, in particular for what concerns detection (Task 1), Reasoning (Task 6), and Matching (Task 7).

Prompting Strategy Impact Analysis

In Figure 13.14, we observe that different prompting strategies, in general, only slightly affect model performance. Llama3.2-90B benefits from CoT prompting across most tasks, with two notable exceptions: in Task 2, where 0-shot prompting performs better, and in Task 3, where both strategies yield comparable results.

Claude-3.5-Sonnet, Gemini-Pro-1.5, and GPT-4o demonstrate consistent perfor-

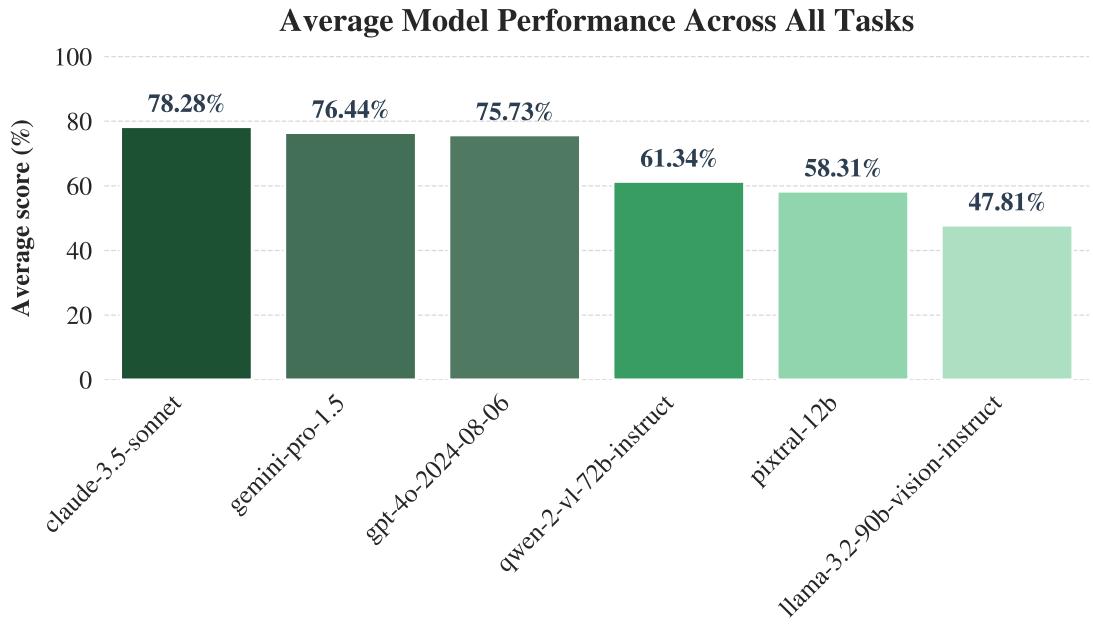


Figure 13.12: Average LVLM performance (best to worst from left to right) regarding the *VisGraphVar* dataset.

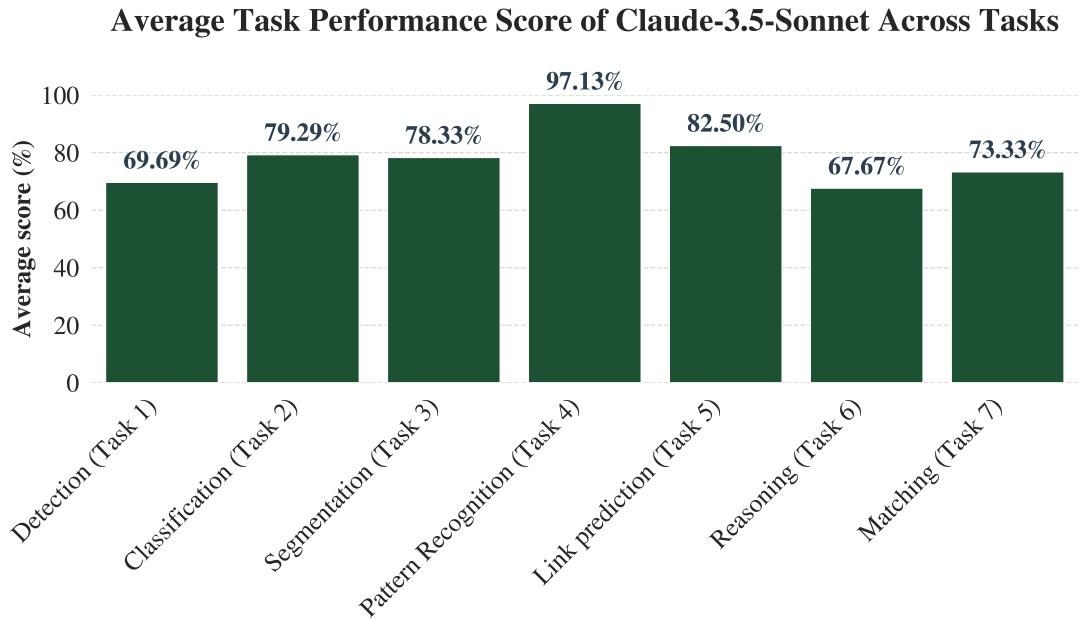


Figure 13.13: Average performance of Claude-3.5-Sonnet for each task from the *VisGraphVar* dataset.

mance across all tasks, showing minimal variation between different prompting strategies. Similarly, Pixtral-12B and Qwen-2-VL-72B generally exhibit little variation between prompting methods. However, notable exceptions emerge in the analysis: Pixtral-12B demonstrates enhanced performance with CoT prompting compared to 0-shot approaches, achieving a significant 16% improvement in Task 5. Conversely, in Task 3, we observe a 10% performance advantage when using 0-shot over CoT prompting.



Figure 13.14: Average scores for each task by prompt strategy, Chain-of-Thought (**top**) and 0-shot (**bottom**). Green indicates strong results, while red denotes poor results.

Overall, and somewhat surprisingly, we do not observe any single prompt strategy consistently outperforming the other. Since 0-shot prompts are easier to create than CoT prompts, we recommend beginning with 0-shot testing; if the results are unsatisfactory, then CoT prompts can be attempted. Other benchmarks in the field support these findings regarding prompting strategies. VisionGraph [118] similarly fails to demonstrate clear superiority of either prompting approach. GITA [221] takes a different approach, focusing solely on 0-shot prompting and fine-tuned LLMs, while omitting CoT evaluation entirely.

Our analysis provides a more detailed and comprehensive examination of this phenomenon, emphasizing that there is no clear superior approach among the most popular prompting strategies, namely 0-shot and CoT. This finding is important to the ongoing discussion about optimal prompting strategies in visual-language tasks.

13.3.3 Observations

This section analyzes three cases that clarify key aspects of LVLM behavior on our dataset created with VisGraphVar. We first examine Figures 13.15 and 13.16, which reveals crucial insights about spectral layout interpretation in Task 1 (Detection). We then explore Figure 13.17, highlighting distinctive characteristics in Pixtral-12B’s performance on Task 7 (Matching). Finally, Figure 13.18 demonstrates the influence of node labeling on model performance and accuracy.

The Striking Case of Spectral Layout

The bar plot in Figure 13.15 shows the average score of the best-performing LVLM (Claude-3.5-Sonnet) on the images of Task 1 (Detection). The scores are separately shown for graphs displayed in different layouts. Interestingly, we observe a substantial decline in performance for the spectral layout (48.82%) compared to other layouts.

Figure 13.16 showcases such a graph with a spectral layout, consisting of 10 nodes and 20 edges. While GPT-4o accurately identifies these elements, Claude-3.5-Sonnet misidentifies the composition, reporting 9 nodes and 14 edges; see below. Interestingly, our prompts not only requested a count of elements but also encouraged the models to provide an analysis justifying their responses.

```
Output: Claude-3.5-Sonnet
1  {
2    "total_nodes": 9,
3    "total_edges": 14,
4    "analysis": "The graph consists of 9 nodes numbered 0-8 in various colors
5      ↳ including pink, red, purple, grey, blue, and teal. The nodes are
6      ↳ connected by 14 black directed edges forming an asymmetric network
7      ↳ structure. The graph has a denser cluster of connections on the left side
8      ↳ and sparser connections extending to nodes 6 and 8 on the right."
9 }
```

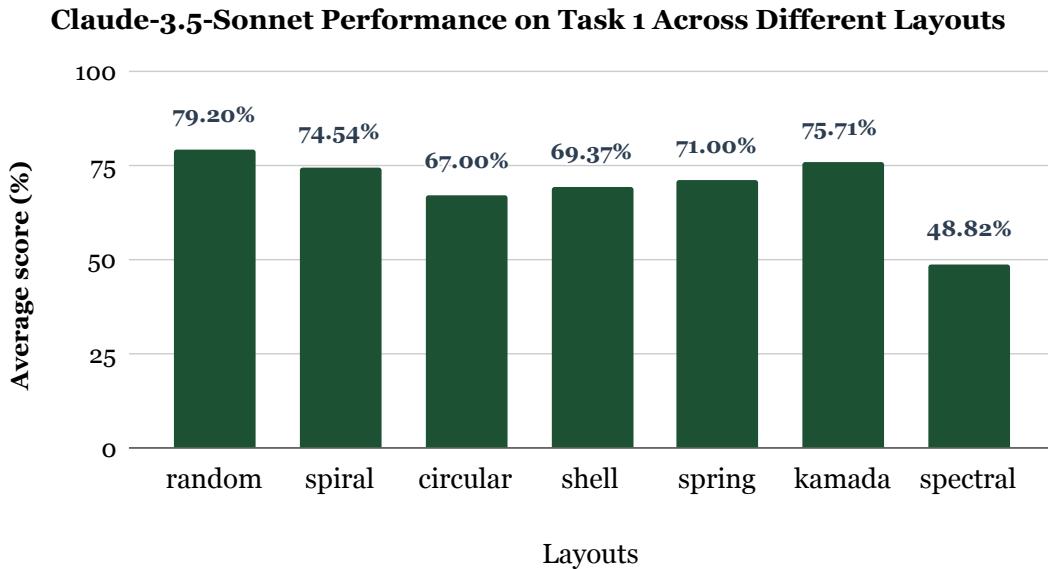


Figure 13.15: Average performance of Claude-3.5-Sonnet on Task 1 for each considered graph layout.

Output: GPT-4o

```

1  ```json
2  {
3      "total_nodes": 10,
4      "total_edges": 20,
5      "analysis": "The graph contains nodes with various colors including shades of
6          ↪ purple, red, orange, and blue. Edges are directed and have no colors."
7  }
8  ```


```

As can be seen, Claude-3.5-Sonnet demonstrates a more comprehensive analysis than GPT-4o. It successfully identifies the overall node positioning, whereas GPT-4o offers a brief and irrelevant explanation. This observation suggests that when analyzing images with spectral layouts, Claude-3.5-Sonnet tends to be captivated by the layout's structural form (focusing on the analysis) rather than performing the exact count requested. This phenomenon could be attributed to the model's remarkably high performance (97.13%) in pattern recognition (Task 4; see Section 13.3.2), which potentially biases its analytical approach towards structural interpretation over precise enumeration. Conversely, GPT-4o appears to concentrate specifically on the task at hand.

This behavior was only observed with the spectral layout, where all models worsen their predictions (see Table 13.2).

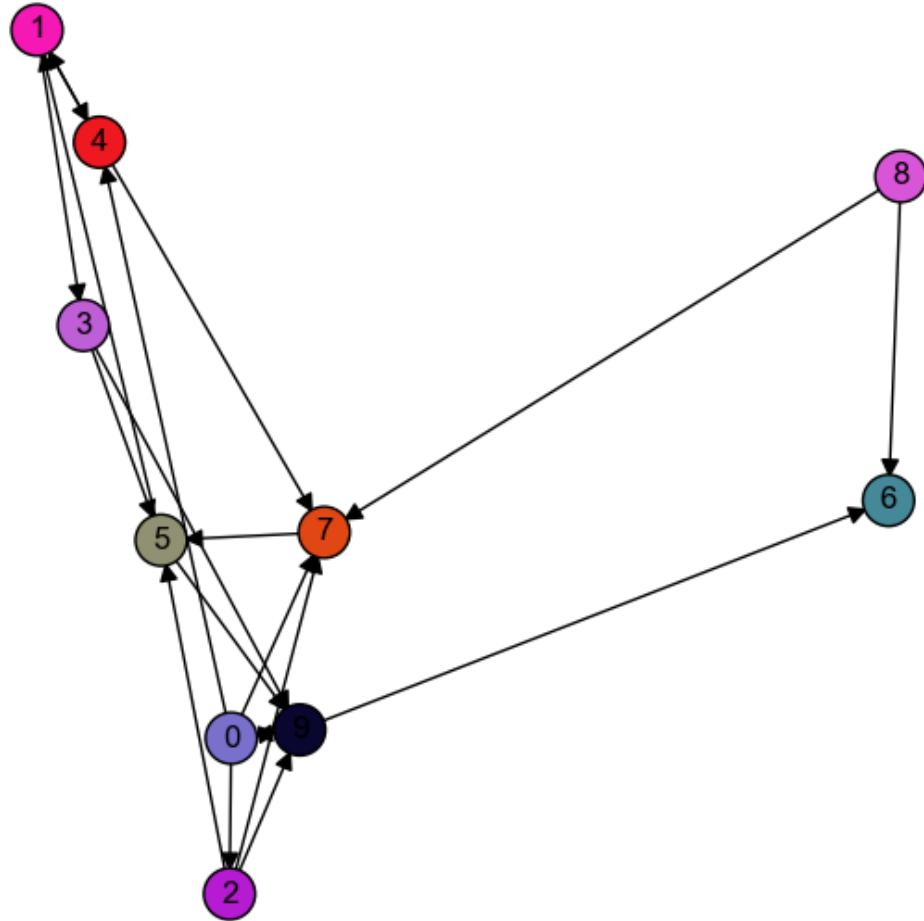


Figure 13.16: Image from our dataset (Task 1), showcasing a spectral layout with randomly colored nodes, directed edges, and 10 nodes with 20 edges.

Table 13.2: Performance percentage for each LLM with the spectral layout.

LLM	spectral layout	
	CoT	0-shot
claude-3.5-sonnet	47.11	48.82
gemini-pro-1.5	55.47	55.11
gpt-4o-2024-08-06	64.07	66.04
llama-3.2-90b-vision-instruct	39.99	44.34
pixtral-12b	44.71	50.92
qwen-2-vl-72b-instruct	55.78	54.56

Important

For detailed results for each layout, please visit our supplementary materials at <https://camilochs.github.io/visgraphvar-website>.

Pixtral-12B and the Complex Task of Matching

Pixtral-12B failed in all instances of Task 7 (Matching). Remember that this dataset contains images showing graphs with 4, 5, and 6 nodes. Moreover, it was the only model to fail for all images. Figure 13.17 shows an example of an image where it failed; see below for Pixtral-12B's response.

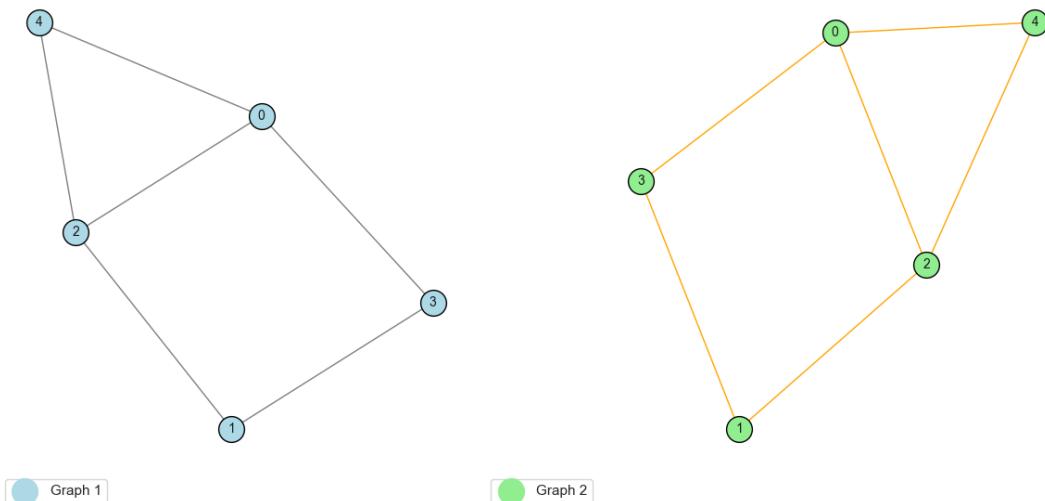


Figure 13.17: Image from the dataset concerning Task 7 (Matching), containing two structurally equal graphs.

Output: Pixtral-12B

```

1  {
2    "match": "no",
3    "analysis": "Upon examining both graphs, it is clear that they are different.
4      ↳ The structure, the arrangement of nodes, and the connections between the
5      ↳ nodes do not match between Graph 1 and Graph 2. Specifically, the filling
6      ↳ color of the nodes and the type of lines (gray vs. orange) differ in both
7      ↳ graphs."
8  }

```

Pixtral-12B consistently makes errors due to a misunderstanding of the prompt (both in 0-shot and CoT scenarios). The prompts specifically ask for comparing the graphs based on their identical structure and node connections, not their visual styles. VisGraphVar presents a significant challenge with this task because it requires the model to infer that it is about graph matching, rather than comparing their stylistic

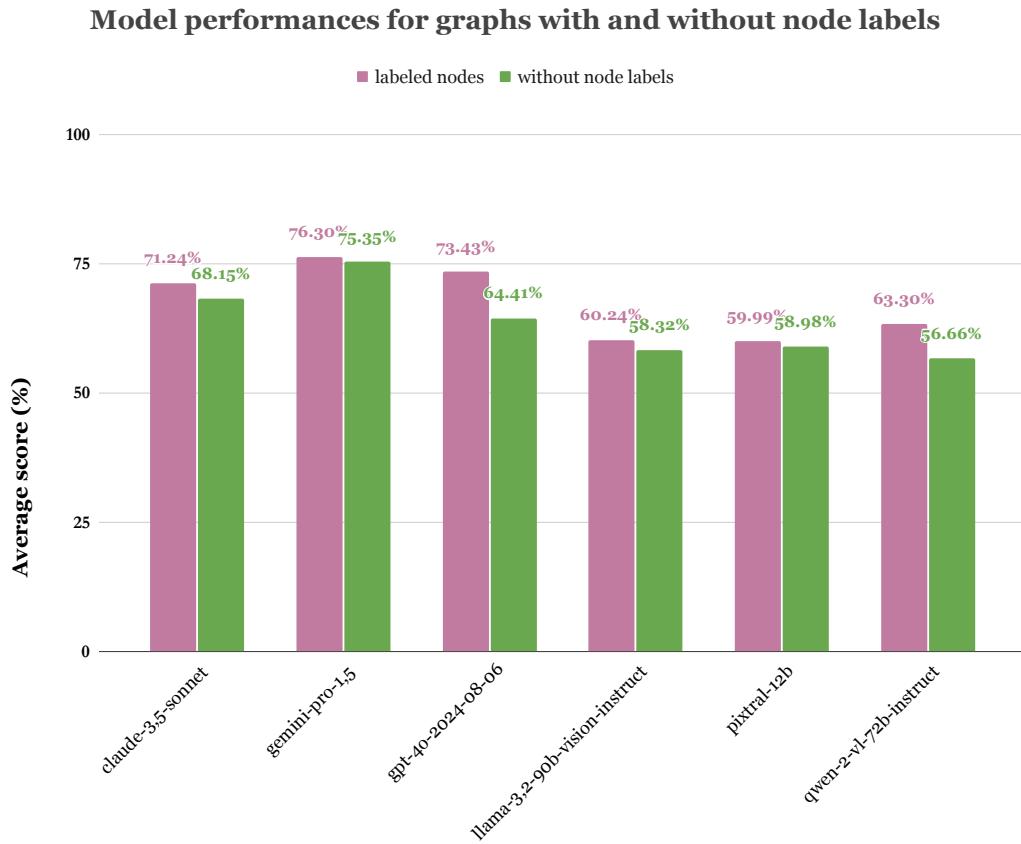


Figure 13.18: Comparison of the average model performance for graphs with labeled nodes (pink) and graphs with unlabeled nodes (green) in Task 1.

similarities. Consequently, Pixtral-12B, likely due to its training set, cannot detect subtle nuances in the prompts, such as distinguishing structural equivalence from stylistic similarity in graphs.

This complexity in detecting a matching task is further highlighted by the task of detecting that graphs do not match. Paradoxically, Pixtral-12B is the only model that succeeds in all 30 cases, but for the wrong reasons. Rather than focusing on graph structure, it simply identifies a stylistic difference between Graph 1 and Graph 2 and concludes they do not match.

The Impact of Node Labels on Model Performance

In the following, we present an intriguing observation that affects all models. In particular, this observation concerns possible performance differences the models show for images with node-labeled graphs and images with unlabeled graphs. Figure 13.18 shows that in Task 1 (Detection), images of graphs with labeled nodes generally result in higher model performance than images with unlabeled graphs.

To explain this, consider the case of GPT-4o, which demonstrates the largest performance gap of $\sim 9\%$. In Figure 13.19, the graphs in (a) and (b) both have 10 nodes and

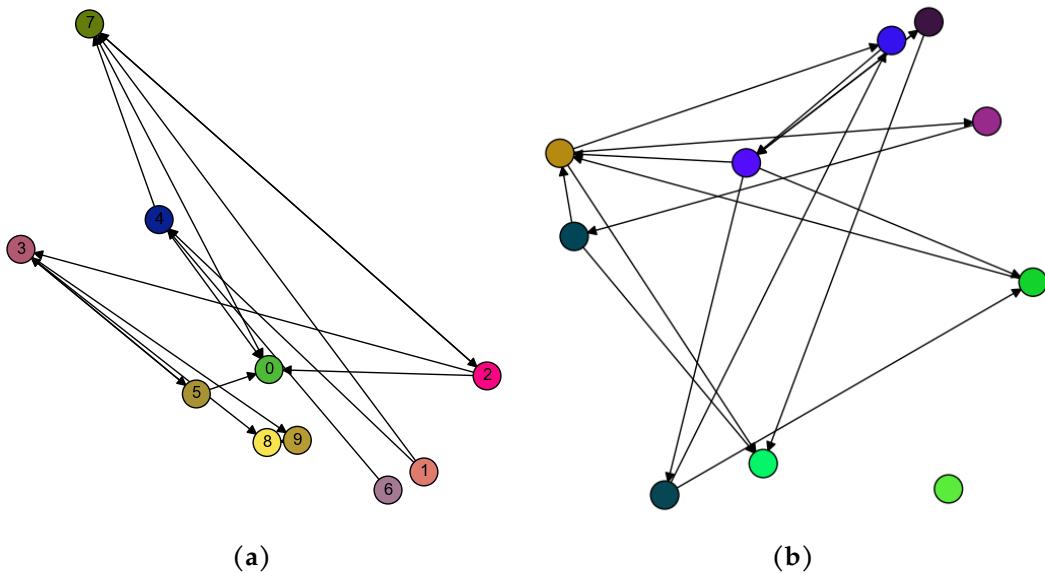


Figure 13.19: (a) A node-labeled graph with 10 nodes and 16 edges and random node-colors. (b) A similar, unlabeled graph. Both belong to the dataset of Task 1.

16 edges. However, GPT-4o's responses produce the following discrepancies:

Output: GPT-4o (for the graph in Figure 13.19a)

```

1 {
2   "total_nodes": 10,
3   "total_edges": 18,
4   "analysis": "The graph contains nodes of various colors, including green,
5   ↳ pink, blue, yellow, and brown. Edges are directed and represented by
   ↳ black arrows connecting the nodes."
}
```

Output: GPT-4o (for the graph in Figure 13.19b)

```

1 {
2   "total_nodes": 9,
3   "total_edges": 23,
4   "analysis": "The graph contains nodes of various colors, including green,
5   ↳ blue, and purple, connected by directed edges. One node is isolated and
   ↳ not connected by any edges."
}
```

When confronted with images of node-labeled graphs, GPT-4o accurately identifies the total number of nodes (10) and closely estimates the number of edges (18 vs. the actual 16). However, the total edges are drastically overestimated for images with unlabeled graphs, reporting 23 instead of 16 (resulting in a difference of 7). This is not an isolated case, which is shown in Figure 13.18. Although the difference is not substantial, images of graphs with labeled nodes consistently yield better performance across

all models.

We hypothesize that models with poorer performance on unlabeled graphs likely had insufficient exposure to such graphs during training, particularly in analysis tasks. This suggests an opportunity to fine-tune a model using more unlabeled graph images to evaluate whether performance improves in these cases.

13.4 Discussion and Open Questions

The expanded potential of LLMs through multimodal capabilities is exemplified by LVLMs, which now offer detailed, low-cost image analysis via API, from object detection to comprehensive captioning. However, tasks requiring both vision and inference remain inherently challenging, with a long history in computer vision. To properly evaluate emerging LVLMs, testing against visually complex tasks is essential. Flexible geometric structures like graphs are crucial here, as they can introduce significant complexity through simple alterations in spatial arrangement or visual representation.

Our study introduced `VisGraphVar`, a benchmark generator designed to challenge current LVLMs and serve as a testing platform for future models focused on visual graph inference. Below, we present open questions from our study that warrant further exploration:

- 1. Which visual style changes truly impair a model’s prediction?** Our `VisGraphVar` dataset provided evidence that visual changes, such as adding node labels or modifying layout, affect inference performance. However, precisely identifying which changes harm model performance and quantifying their impact remains an open question. We believe `VisGraphVar` can inspire further research into these questions—beyond just visual graph inference—through comprehensive analyses to clearly pinpoint which visual styles distort predictions.
- 2. Can an LVLM achieve over 90% accuracy across all seven `VisGraphVar` tasks?** Testing LVLMs against these seven tasks proved challenging; while Claude-3.5-Sonnet performed best, it is still far from achieving high scores across all tasks. Potential strategies to improve results on our dataset, or future `VisGraphVar`-generated datasets, include:
 - Training fine-tuned models capable of handling all seven tasks and variations in visual styles [240].
 - Researching advanced prompt strategies [210, 178] and hypothesizing how prompt modifications could improve results.
 - Moving beyond single prompts to a series of prompts that enable self-correction within an image context [159], fostering interaction through LLM-based agents [173].
- 3. How can LVLMs be incorporated into real-world graph theory applications?** Benchmarks like `VisionGraph` [118], `GITA` [221], and our `VisGraphVar` focus on

small graphs, as current LVLMs are not yet mature enough for larger scales. Furthermore, for larger or denser graphs, visualization challenges are not unique to LVLMs but are common limitations for any automated interpretation system. However, we anticipate that once LVLMs achieve greater precision in various graph analysis tasks, they will be invaluable for quickly analyzing complex graph images in domains like social networks, communication networks, and educational applications. Given the current state of multimodal model technology, it is already feasible to integrate them into applications for general analytical questions rather than precise algorithmic operations on an image. The first step for integration would be to define the specific tasks an LVLM should perform (using this study as a guide) and then identify the model best suited for those tasks.

13.5 Conclusions

This research evaluated Large Vision-Language Models (LVLMs) using graphs, chosen for their rich geometric and structural complexity. We introduced `VisGraphVar`, a customizable benchmark generator exploring visual variability across stylistic elements and graph structures. The framework generates datasets for seven distinct tasks (detection, classification, segmentation, pattern recognition, link prediction, reasoning, and matching), comprehensively covering graph interpretation and analysis. Our analysis reveals two key findings: visual variability significantly impacts LVLM performance, and current LVLM architectures require fundamental enhancements for complex visual graph interpretation.

Concrete steps to advance graph analysis with LVLMs include:

- **Testing LVLMs in real-world graph analysis applications.** LVLMs could summarize graph images and provide general insights (e.g., structure, clusters, density) in fields like social or biological networks. However, extracting specific data (e.g., solving algorithms, counting elements in large graphs) remains challenging, requiring further advancements.
- **Fine-tuning.** Fine-tuning models with input-output examples could address visual variability issues, training LVLMs to handle specific scenarios like overlapping shapes.
- **Creating more complex datasets using `VisGraphVar`.** As a benchmark generator, `VisGraphVar` can design more challenging scenarios with greater visual variability to push new models' limits. An LVLM achieving high average scores across `VisGraphVar`'s seven tasks would be ready for real-world graph image analysis, providing general insights. However, detailed quantitative analysis (e.g., counting nodes in large graphs) remains a significant challenge requiring careful validation.

Note

Although, as I mentioned at the beginning of this chapter, it remains somewhat “disconnected” from my thesis, this work proved valuable in demonstrating the potential of using LVLMs within STNWeb. Given that STNWeb is more complex than the graph images used in this chapter—employing its own unique symbology—there are promising possibilities to achieve good results through fine-tuning smaller models (such as Small Language Models like LLaVA^a).

I suspect that the textual extraction of graph features generated by STNWeb (Chapter 11) cannot be fully replaced by LVLMs, but rather should be integrated with them. Even with fine-tuning, LVLMs may mistake on simple tasks such as counting the total number of nodes. However, they could prove useful for detecting regions of a graph where overlaps occur; for example, “In the upper-right corner, there is noticeable overlap in the trajectory of ALGORITHM_A.” This kind of spatial-visual insight is impossible with the purely textual LLM approach presented in Chapter 11, as it lacks access to visual-spatial information.

I hope to have the opportunity to explore this further in the future!

^a <https://llava-v1.github.io/>

14

Conclusion

This thesis explored various strategies aimed at mitigating the intrinsic weaknesses of metaheuristics—namely, their lack of instance-specific information and the difficulty in interpreting their behavior on particular problems.

To address these challenges, two complementary lines of research were developed. The first focuses on enhancing the algorithmic performance of metaheuristics by integrating modern machine learning techniques. The second involves the development of a web-based tool that enables visual analysis of metaheuristic behavior. These two approaches are mutually reinforcing: improved interpretability helps justify algorithmic enhancements, while algorithmic improvements can reveal previously unnoticed visual patterns worthy of further investigation.

14.1 Discussion of Main Contributions

14.1.1 Part I – Algorithmic Enhancements

My initial exploration into the use of machine learning within metaheuristics focused on integrating Graph Neural Networks (GNNs). GNNs are particularly well-suited for this purpose, as they are designed to capture the topological structure of graphs—a common representation in combinatorial optimization problems. This approach proved effective in two distinct scenarios: first, by combining GNNs with a Biased Random-Key Genetic Algorithm (BRKGA) to tackle the Multi-Hop Influence Maximization problem in social networks (see Chapter 3); and second, by incorporating GNNs into an Ant Colony Optimization (ACO) framework for the Target Set Selection problem, also in the context of social networks (see Chapter 4).

A significant advantage of the GNN-based approach is its scalability: it can evaluate graphs with millions of nodes without a substantial increase in evaluation time. However, as with any deep learning model, training requires large amounts of data and computational resources. These trade-offs must be carefully considered in order to fully leverage the benefits of this method.

Motivated by these limitations, and driven by a curiosity to explore emerging technologies, I decided to move beyond GNNs and investigate the potential of Large Language Models (LLMs) to enhance the quality of solutions produced by metaheuristics. This shift opened a new line of experimentation, where the generative and reasoning capabilities of LLMs were used to guide the search process and influence algorithmic decisions dynamically.

In Chapter 5, we demonstrate that LLMs can be valuable in metaheuristics not only through code generation but also as pattern recognition engines applied to tabular data containing graph instance metrics. In this context, the LLM acts as a numeric pattern detector, identifying which regions of the graph are likely to be more promising for exploration. To test this hypothesis, we applied a BRKGA to the same problem addressed in Chapter 3, and the LLM-guided approach outperformed the previous GNN-based method.

After, we also explored the potential of LLMs to enhance metaheuristics from a different perspective—code improvement. Rather than prompting LLMs to generate algorithms from scratch, we focused on leveraging expert-written code and using LLMs as collaborators in the design process. This was done in two ways: (1) identifying regions in the code that could benefit from heuristic modifications (see Chapter 6); and (2) automatically updating entire algorithm implementations using modern techniques, drawing on the LLMs’ broad algorithmic knowledge (see Chapter 7).

The main result of this research line (Part I) is the demonstration that metaheuristics can be effectively integrated with LLMs. We also present a method for incorporating GNNs into metaheuristic frameworks, further developing a direction previously explored in the literature. Each approach has its own strengths and limitations, and the choice of hybridization strategy should be guided by the specific characteristics of the problem at hand. Both approaches can contribute to strengthening metaheuristics by addressing their weaknesses through the use of modern techniques.

14.1.2 Part II – Enhanced Interpretability

The performance of a metaheuristic is typically evaluated through statistical analysis and numerical comparisons with other methods, aiming to determine whether it produces higher-quality solutions for a given problem. However, due to the stochastic nature of these algorithms, such evaluation alone is often insufficient. Additional layers of analysis are needed to better understand and justify newly proposed heuristic strategies.

In this part of the thesis, we focus on research surrounding Search Trajectory Networks (STNs). To this end, we developed a tool called STNWeb, which encapsulates the STN methodology within a user-friendly web interface. Beyond simply making STNs more accessible, STNWeb introduces new functionalities designed to enrich analysis and promote broader adoption by the research community.

The STNWeb tool and its purpose are detailed in Chapters 9 and 10. We then intro-

duce a novel feature designed to lower the entry barrier for new researchers interested in deepening metaheuristic analysis: the integration of LLMs to generate textual reports that facilitate the interpretation of the visualizations produced by STNWeb (see Chapter 11).

Additionally, we enhanced the quality of STNWeb’s visual outputs by improving the concept of search space partitioning. This allows STN to detect finer nuances in the trajectories of each algorithm run—particularly useful in cases where algorithms behave similarly or when the solution space is vast (with hundreds of variables). To achieve this, a clustering algorithm was integrated into STNWeb (see Chapter 12).

This section closes with preliminary research focused on integrating direct visual-spatial analysis of STNWeb’s graph visualizations. In Chapter 13, we introduce a benchmark generator aimed at assessing the capabilities of Large Vision-Language Models (LVLMs) in analyzing images of various types of graphs. These include both directed and undirected graphs, with or without labels, featuring different node and edge styles, and exhibiting diverse topologies. This investigation remains preliminary because STNWeb generates directed graphs with a specialized visual language, presenting unique challenges for LVLM interpretation.

14.2 Limitations and Challenges

14.2.1 Algorithmic Improvement

Based on my experience working with both GNNs and LLMs for algorithmic improvement, the primary limitations and challenges detected with these technologies are as follows:

- **Graph Neural Networks**

1. **Data Quality and Representativeness:** A significant challenge lies in acquiring or constructing a sufficiently high-quality dataset, both in terms of instance quality and their representativeness of the underlying optimization problem [20]. In our work, we adopted a strategy of generating synthetic random graphs of various sizes, specifically Erdős–Rényi graphs, which are commonly used to capture the topology of social network graphs [19]. However, “similarity” does not necessarily guarantee that these synthetic graphs accurately capture the complex topologies and characteristics of real-world graphs. Furthermore, determining the optimal number of synthetic graphs required for effective training remains an open question. These limitations are intrinsically linked to a well-known problem in Deep Learning: the inherent need for vast quantities of high-quality data to achieve adequate predictive performance, a stark contrast to some more traditional machine learning approaches that can perform well with less data.
2. **Sensitivity to Graph Topology:** GNNs’ performance can be highly sensitive to the input graph’s topology. If a graph contains significant noise—such

as spurious edges, irrelevant or mislabeled nodes, or structural inconsistencies that distort the underlying relational patterns—or if it is highly sparse or possesses a complex, irregular structure, GNNs may struggle to detect meaningful patterns in node relationships [99, 53]. This limitation directly impacts their ability to perform accurate inference on nodes, edges, or subgraphs, which is fundamental for guiding optimization algorithms.

- **Large Language Models**

1. **Limited Context Window:** A primary constraint for LLMs in optimization is their limited context window [88]. This restricts the amount of problem-specific information (e.g., large graph structures, extensive tabular data) that can be fed into the model in a single prompt. While context windows are continually expanding, it is important to note that a larger context does not necessarily guarantee sustained LLM inference quality [85]. Furthermore, current context windows are not yet large enough to effectively handle very large-scale optimization instances with millions of nodes, which are common in real-world applications.
2. **High Economic Cost per Token:** Despite a general trend of decreasing prices, the computational cost associated with processing large numbers of tokens can still be substantial, especially for iterative optimization processes or when dealing with complex prompts for large instances. This can make extensive experimentation or deployment economically prohibitive for some researchers or applications.

In the specific context of LLMs' application in optimization, the scenario presents a nuanced trade-off. While LLM costs have generally decreased over time, and they do not necessitate the laborious construction of problem-specific datasets like GNNs, a significant hurdle remains: the current context window limitations prevent LLMs from effectively evaluating and generating probabilities for individual nodes in graphs containing millions of elements. This scale is currently feasible with GNNs.

Furthermore, our experiments demonstrate that certain metaheuristics are more susceptible to LLM-driven enhancements than others. The BRKGA, for instance, proved highly receptive to subtle biases provided by LLMs, enabling it to explore more promising search spaces effectively. This was not the case with Ant Colony Optimization (ACO); attempts to replicate the LLM-biasing approach from Chapter 5 for ACO did not yield comparable positive results, unlike when GNNs and Q-learning were incorporated into ACO (Chapter 4). This suggests that for population-based metaheuristics like ACO, a static, offline bias value from an LLM might be insufficient. Such algorithms may require more structural changes to their operation or dynamic, iterative feedback from the LLM, which points towards the first line of future research discussed in the next section.

14.2.2 Interpretability Enhanced

Visualization is always open to improvement, particularly because it is, to some extent, a *subjective and context-dependent task*. The clarity and usefulness of a visualization often evolve based on feedback from domain experts and users interested in specific functionalities. In this sense, STNWeb still offers significant room for enhancement. Below, we outline some of its current limitations:

- **Limited to 2D visualization.** One of the main directions for future research (see next section) is the development of a 3D version of STNWeb. A 3D interface would allow for a better perception of trajectory intersections across algorithms, offering richer angles and perspectives compared to the current static 2D layout. This improvement could help reveal patterns and convergence behaviors that remain hidden in two dimensions.
- **No temporal representation.** Both STN and STNWeb rely on static visualizations of fitness values throughout the execution. However, they currently lack any encoding of *temporal dynamics*, such as the time required to move from one solution to another or the convergence speed. Incorporating temporal information would offer a more complete picture of algorithmic behavior, especially in time-sensitive or dynamic problem domains.
- **Limited support for complex solution structures.** Some algorithms—such as those used in Genetic Programming—evolve highly complex solution structures, including expression trees, graphs, or full programs. These are difficult to represent through linear (string-based) encodings and require more sophisticated visualization models. Extending STNWeb to support such cases would significantly broaden its applicability across different classes of metaheuristics and optimization problems.

14.3 Future Research Directions

The following are research directions that, unfortunately, I was unable to fully pursue by the time of submitting this thesis (July 2025). However, some of them already show promising results and partial progress, which makes it likely that I will have more to say about them by the time of the defense—possibly even in the form of a publication.

Below, I briefly outline each of them:

14.3.1 STNWeb 3D

The most natural next step for the work presented in Part II of this thesis, which focuses on STNWeb, is the development of a fully redesigned 3D version. The aim is not only to add a third dimension to improve the spatial perception of trajectory intersections, but also to rebuild the entire web application architecture. In particular, the render-

ing engine would be moved entirely to the frontend, leveraging `D3.js`¹, which would significantly reduce the latency currently experienced in the 2D version.

At the time of writing, the project is approximately 30% complete. I am optimistic about being able to finish and release this new version in the near future.

14.3.2 Path-Dependent Runtime Heuristic Steering (PathSteer)

I plan to investigate a novel paradigm, tentatively named PathSteer, focused on the runtime self-modification of metaheuristic components. Unlike traditional static optimization approaches (Chapters 5 to 7), this system would leverage a LLM to dynamically generate new heuristic functions during execution. For example, in the case of Ant Colony Optimization, the LLM would synthesize new variants of pheromone update or exploration strategies based on detailed runtime metrics—such as solution quality trends, stagnation indicators, or constraint violations.

A key innovation of this approach is the idea of navigating a dynamic graph of heuristics, where the effectiveness of a given heuristic may depend not only on the current problem state, but also on the *sequence* of previous modifications—its *modification path*. Our hypothesis is that this path-dependent adaptation will yield more effective and problem-specific behaviors than any fixed heuristic configuration.

Initial results are promising: our current implementation has already outperformed state-of-the-art methods for the *Target Set Selection* problem, as discussed in Chapter 4.

This is arguably the most ambitious idea to emerge from this thesis, as it synthesizes the insights and results from Chapters 5 to 7 and pushes them beyond static algorithm design into a radically different paradigm.

Although research on metaheuristics that adapt at runtime already exists [201]—including coevolutionary algorithms, where different components evolve in response to each other [169]—this new approach differs fundamentally: it modifies the actual structure of the implementation code, not merely its parameter values. In this way, *runtime adaptation*, powered by LLMs, transforms heuristics from fixed components into dynamic, evolving strategies—continuously shaped by live feedback and contextual performance, and governed by a temporally dependent process, not by an evolutionary one as in AlphaEvolve [152].

¹ An open-source JavaScript library designed for fast and dynamic data visualization. <https://d3js.org/>

Appendix

A

A Brief Guide to Optimization

Optimization problems, understood from a mathematical perspective, focus on finding the best possible solution to a problem given a set of constraints and objectives. The simplest case involves maximizing or minimizing a function by selecting input values (that satisfy the constraints) and computing the function's value.

Although optimization problems can be purely mathematical in nature, they have a wide range of real-world applications. In virtually any business, for example, it is necessary to maximize profits and minimize costs, a classic optimization problem. The same applies to logistics challenges, such as reducing delivery times within a city that offers dozens of possible routes from one point to another. Or to airline route planning, where hundreds of takeoffs and landings must be coordinated without overlap, while also optimizing fuel consumption by selecting the most efficient path. Even in social networks with millions of users, it is crucial to identify the subset of individuals capable of exerting the greatest influence over others. These are just a few examples of optimization problems that shape our everyday lives—and there are many more.

A.1 Types of Optimization Problems

There are many kinds of optimization problems. One way to classify them is based on the structure of the solution space.

A.1.1 Continuous Optimization

In continuous optimization, the decision variables can take any real value within specified bounds or domains. This contrasts with combinatorial or discrete optimization, where variables are restricted to a finite or countable set of options. The solution space in continuous problems is typically a subset of \mathbb{R}^n , which allows the use of calculus-based techniques such as gradient descent, Newton's method, and other analytical approaches that rely on differentiability and continuity of the objective function.

Example

Consider the following optimization problem:

$$\text{Minimize: } f(x) = x^2$$

$$\text{Subject to: } x \geq 1$$

Solution: The goal is to find the smallest value of $f(x) = x^2$ for $x \geq 1$. The minimum occurs at $x = 1$, and the minimum value is $f(1) = 1$.

A.1.2 Discrete or Combinatorial Optimization

In discrete (or combinatorial) optimization, the solution space consists of discrete elements, typically drawn from a finite or countably infinite set. Unlike continuous optimization, where variables can take any value within an interval, here the decision variables are often integers, binary values, or categorical choices. The goal is to find the optimal configuration or selection that satisfies a given set of constraints while optimizing an objective function.

Example

Consider the following combinatorial optimization problem:

Problem: Given the graph $G = (V, E)$ where $V = \{A, B, C, D\}$ and $E = \{(A, B), (B, C), (C, D)\}$, find the largest set of nodes such that no two nodes in the set are connected.

Solution: An example of an optimal solution is $\{A, C\}$. No edge connects these two nodes, and no larger independent set exists in this graph.

In this thesis, combinatorial optimization problems are primarily used, and continuous problems in specific cases. However, there are also other types of optimization problems, such as:

- **Mixed-Integer Optimization**, which involves both continuous and discrete variables and is commonly used in problems like scheduling and resource allocation.
- **Multi-objective Optimization**, where more than one objective function must be optimized simultaneously, often leading to trade-offs between competing goals.
- **Stochastic Optimization**, which deals with uncertainty in the data or model, requiring solutions that perform well on average or under varying conditions.
- **Dynamic Optimization**, in which decisions are made over time and each choice influences future states and outcomes.

B

A Brief Introduction and Defense of Metaheuristics

A metaheuristic is a type of optimization algorithm that does not guarantee finding the global optimum, but aims to quickly provide a sufficiently good approximate solution by employing heuristic operators. The prefix “meta” highlights its emphasis on generality: metaheuristics are designed as high-level frameworks applicable to a broad range of optimization problems, rather than being tailored to a specific one.

Metaheuristics typically operate by searching for approximate solutions through non-deterministic/stochastic processes. They take advantage of the pseudo-random capabilities of modern computation to strike a balance between exploitation (intensifying the search around promising areas) and exploration (diversifying the search across the solution space). Notable examples include Genetic Algorithms, Simulated Annealing, Tabu Search, and Ant Colony Optimization.

The proliferation of metaheuristics over the years has led to two notable outcomes. First, a wide range of taxonomies has been proposed to classify them—such as discontinuous vs. trajectory-based, population-based vs. single-solution, constructive vs. local search-based, and memory-based vs. memory-less approaches [193]. Second, and more problematically, there has been a growing misuse of metaphors. Many recent metaheuristics contribute little beyond a catchy name and an associated metaphor, offering no substantive methodological innovation—a trend increasingly criticized in the academic literature [8].

A Simple and Formal Example of Metaheuristics

A simple example of a metaheuristic using pseudo-random operators is a variant of the **local search** strategy. Consider again the problem of finding the highest point in a landscape (the global maximum), where the altitude is given by a function $f(x)$. Instead of always moving in the steepest uphill direction, the algorithm uses pseudo-randomness to probabilistically explore the neighborhood. Let the process be defined as follows:

1. Choose an initial solution x_0 randomly.
2. At each iteration t , generate a set of candidate solutions $\mathcal{N}(x_t)$ in the neighborhood of x_t using a pseudo-random sampling mechanism (e.g., Gaussian perturbation, uniform sampling within a radius).
3. Select a candidate $x_{t+1} \in \mathcal{N}(x_t)$ such that $f(x_{t+1}) > f(x_t)$, or with some small probability ϵ , accept a worse solution (to escape local optima).
4. Repeat until a stopping criterion is met (e.g., max iterations or no improvement).

This stochastic variation allows the search to avoid getting trapped in local optima and introduces a balance between *exploration* and *exploitation*. The use of pseudo-randomness makes the process non-deterministic, which is a defining trait of many metaheuristics such as Simulated Annealing or Evolutionary Algorithms.

However, the field of metaheuristics has not been exempt from criticism. I outline the main objections commonly raised against their use and argue that, despite these concerns, metaheuristics continue to be a valuable asset in computational optimization.

1. **Opacity.** Perhaps the most frequent criticism is one shared with many stochastic and approximation methods: the lack of transparency in how results are obtained.

Response: While metaheuristics often lack a step-by-step theoretical explanation of how they reach specific solutions—primarily due to their use of pseudo-random operators—their value lies not in interpretability but in practical effectiveness. Full process transparency, while desirable, is not always necessary to exploit their results. Nevertheless, this dissertation addresses this challenge in Part II by incorporating a visual tool aimed at improving the understanding of metaheuristic result comparisons.

2. **Lack of Exactness.** As approximation methods, metaheuristics do not guarantee reaching the global optimum, unlike exact algorithms. This limitation, compounded by their stochastic nature, stems from their reliance on randomness to diversify the search process.

Response: As demonstrated in Part I of this thesis—there have been significant efforts to overcome this shortcoming by hybridizing metaheuristics with other techniques. For instance, the integration of machine learning methods allows metaheuristics to learn from the structure of specific problems and instances, transforming that knowledge into guiding heuristics that improve performance. Similarly, earlier works have explored hybridization with exact methods (see [23]).

3. **Lack of rigor.** This criticism focuses on the claim that, due to their metaphor-driven design and lack of exactness, metaheuristics are not considered *serious* methods.

Response: Although they may lack rigorous theoretical foundations, metaheuristics have proven effective in practice, as demonstrated by numerous experiments. In other words, it is a primarily empirical field whose foundation rests on experimental validation. (A similar criticism has been made regarding generative AI, albeit for different reasons.)

However, this defense has also led to misunderstandings and misuse—namely, the creation of metaheuristics that offer nothing new beyond a metaphor. Therefore, a way to strengthen the field is to discourage such poor practices.

This thesis contributes to addressing both criticisms. However, these challenges are far from settled, and further research is necessary to continue strengthening the field.

B.1 An Invitation to Metaheuristics

If someone asked me why they should study metaheuristics in the context of research, I would offer the following reasons:

- **A laboratory for algorithm designers.** If you enjoy designing and implementing algorithms, metaheuristics offer a highly flexible and fertile ground for experimentation. Their adaptability makes them ideal for incorporating novel techniques and tools from other domains—as shown in this thesis by integrating Large Language Models into metaheuristic.
- **A preference for simplicity over complexity.** There is significant room for incorporating tools that enhance the interpretability and transparency of metaheuristic outputs. Such contributions would not only strengthen the field but also provide valuable support tools for researchers, as exemplified by the STNWeb tool.
- **A balance between theory and practice.** Few areas offer such a well-calibrated balance. Many of the most impactful contributions in metaheuristics arise from the simple question: “Can it be implemented?” It is a field that rewards those who enjoy programming, even if the problems tackled are sometimes abstract rather than directly practical.
- **A role in modern technologies.** Google DeepMind’s AlphaEvolve uses evolutionary techniques (metaheuristic) to guide its generative AI in the discovery of new algorithms [152]. The growing number of papers applying principles from

evolutionary computation—along with the increasing use of swarm-based behaviors in robots and drones for coordination—opens up fascinating opportunities for new students: integrating metaheuristics and cutting-edge technologies in both directions.

Bibliography¹

- [1] Pravesh Agrawal et al. *Pixtral 12B*. 2024. arXiv: 2410.07073 [cs.CV]. URL: <https://arxiv.org/abs/2410.07073>.
- [2] Janice Ahn et al. *Large Language Models for Mathematical Reasoning: Progresses and Challenges*. 2024. arXiv: 2402.00157 [cs.CL].
- [3] Mehmet Anil Akbay and Christian Blum. “Two examples for the usefulness of STNWeb for analyzing optimization algorithm behavior”. In: *Metaheuristics International Conference*. Springer. 2024, pp. 341–346.
- [4] Mehmet Anil Akbay et al. “Variable Neighborhood Search for the Two-Echelon Electric Vehicle Routing Problem with Time Windows”. In: *Applied Sciences* 12.3 (2022). ISSN: 2076-3417. DOI: 10.3390/app12031014. URL: <https://www.mdpi.com/2076-3417/12/3/1014>.
- [5] Mirko Alicastro et al. “A reinforcement learning iterated local search for makespan minimization in additive manufacturing machine scheduling problems”. In: *Computers & Operations Research* 131 (2021), p. 105272.
- [6] Guilherme F. C. F. Almeida et al. “Exploring the psychology of LLMs’ moral and legal reasoning”. In: *Artificial Intelligence* 333 (2024), p. 104145. ISSN: 0004-3702. DOI: <https://doi.org/10.1016/j.artint.2024.104145>.
- [7] Anthropic. *The Claude 3 Model Family: Opus, Sonnet, Haiku*. 2024. eprint: <https://paperswithcode.com/paper/the-claude-3-model-family-opus-sonnet-haiku>.
- [8] Claus Aranha et al. “Metaphor-based metaheuristics, a call for action: the elephant in the room”. In: *Swarm Intelligence* 16.1 (Mar. 2022), pp. 1–6. ISSN: 1935-3820. DOI: 10.1007/s11721-021-00202-9. URL: <https://doi.org/10.1007/s11721-021-00202-9>.
- [9] Kavosh Asadi and Michael L. Littman. *An Alternative Softmax Operator for Reinforcement Learning*. 2017. arXiv: 1612.05628 [cs.AI]. URL: <https://arxiv.org/abs/1612.05628>.
- [10] Kehinde Babaagba, Ritwik Murali, and Sarah Thomson. “Exploring the use of fitness landscape analysis for understanding malware evolution”. In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. 2024, pp. 77–78.
- [11] Liam Barkley and Brink van der Merwe. *Investigating the Role of Prompting and External Tools in Hallucination Rates of Large Language Models*. 2024. arXiv: 2410.19385 [cs.CL]. URL: <https://arxiv.org/abs/2410.19385>.

¹ References with more than three authors include *et al.*

- [12] Solon Barocas, Moritz Hardt, and Arvind Narayanan. *Fairness and machine learning: Limitations and opportunities*. MIT Press, 2023.
- [13] Partha Basuchowdhuri and Subhashis Majumder. "Finding Influential Nodes in Social Networks Using Minimum k-Hop Dominating Set". In: *Applied Algorithms*. Ed. by Prosenjit Gupta and Christos Zaroliagis. Cham: Springer International Publishing, 2014, pp. 137–151. ISBN: 978-3-319-04126-1. URL: https://link.springer.com/chapter/10.1007/978-3-319-04126-1_12.
- [14] Emily M. Bender et al. "On the Dangers of Stochastic Parrots: Can Language Models Be Too Big?" In: *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*. FAccT '21. Virtual Event, Canada: Association for Computing Machinery, 2021, pp. 610–623. ISBN: 9781450383097. doi: [10.1145/3442188.3445922](https://doi.org/10.1145/3442188.3445922). URL: <https://doi.org/10.1145/3442188.3445922>.
- [15] Jacob Benesty et al. "Pearson Correlation Coefficient". In: *Noise Reduction in Speech Processing*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 1–4. ISBN: 978-3-642-00296-0. doi: [10.1007/978-3-642-00296-0_5](https://doi.org/10.1007/978-3-642-00296-0_5). URL: https://doi.org/10.1007/978-3-642-00296-0_5.
- [16] Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. "Machine learning for combinatorial optimization: a methodological tour d'horizon". In: *European Journal of Operational Research* 290.2 (2021), pp. 405–421.
- [17] Arthur Benjamin, Gary Chartrand, and Ping Zhang. *The fascinating world of graph theory*. en. Princeton, NJ: Princeton University Press, Jan. 2015.
- [18] Chris Bennett et al. "The aesthetics of graph visualization". In: *Proceedings of the Third Eurographics Conference on Computational Aesthetics in Graphics, Visualization and Imaging*. Computational Aesthetics'07. Alberta, Canada: Eurographics Association, 2007, pp. 57–64. ISBN: 9783905673432.
- [19] Lee Bernick. "Modeling human networks using random graphs". In: (2018).
- [20] Christopher M. Bishop and Hugh Bishop. *Deep Learning - Foundations and Concepts*. Springer, 2024. ISBN: 978-3-031-45467-7. doi: [10.1007/978-3-031-45468-4](https://doi.org/10.1007/978-3-031-45468-4). URL: <https://doi.org/10.1007/978-3-031-45468-4>.
- [21] Christian Blum. *Construct, Merge, Solve & Adapt: A Hybrid Metaheuristic for Combinatorial Optimization*. Computational Intelligence Methods and Applications. Springer Nature Switzerland, 2024. ISBN: 9783031601026. URL: <https://books.google.es/books?id=ENCt0AEACAAJ>.
- [22] Christian Blum and Andrea Roli. "Metaheuristics in combinatorial optimization: Overview and conceptual comparison". In: *ACM Computing Surveys* 35.3 (2003), pp. 268–308.
- [23] Christian Blum et al. "Hybrid metaheuristics in combinatorial optimization: A survey". In: *Applied Soft Computing* 11.6 (2011), pp. 4135–4151. ISSN: 1568-4946. doi: <https://doi.org/10.1016/j.asoc.2011.02.032>. URL: <https://www.sciencedirect.com/science/article/pii/S1568494611000962>.
- [24] Christian Blum et al. "Construct, Merge, Solve & Adapt A new general algorithm for combinatorial optimization". In: *Computers & Operations Research* 68 (2016), pp. 75–88.

ISSN: 0305-0548. doi: <https://doi.org/10.1016/j.cor.2015.10.014>. URL: <https://www.sciencedirect.com/science/article/pii/S0305054815002452>.

- [25] Tom B. Brown et al. *Language Models are Few-Shot Learners*. 2020. arXiv: 2005.14165 [cs.CL].
- [26] Chen Cai and Yusu Wang. “A Note on Over-Smoothing for Graph Neural Networks”. In: (2020). arXiv: 2006.13318. URL: <http://arxiv.org/abs/2006.13318>.
- [27] Laura Calvet et al. “Learnheuristics: hybridizing metaheuristics with machine learning for optimization with dynamic inputs”. In: *Open Mathematics* 15.1 (2017), pp. 261–280. doi: [doi:10.1515/math-2017-0029](https://doi.org/10.1515/math-2017-0029). URL: <https://doi.org/10.1515/math-2017-0029>.
- [28] Santle Camilus and Valappil Kunnumal Govindan. “A Review on Graph Based Segmentation”. In: *International Journal of Image, Graphics and Signal Processing* 4 (June 2012). doi: [10.5815/ijigsp.2012.05.01](https://doi.org/10.5815/ijigsp.2012.05.01).
- [29] Raphaël Candelier. “Graph matching based on similarities in structure and attributes”. In: *arXiv* [cs.DS] (Sept. 2024).
- [30] Camilo Chacon Sartori, Christian Blum, and Gabriela Ochoa. “Search Trajectory Networks Meet the Web: A Web Application for the Visual Comparison of Optimization Algorithms”. In: *Proceedings of the 2023 12th International Conference on Software and Computer Applications*. ICSCA ’23. Kuantan, Malaysia: Association for Computing Machinery, 2023, pp. 89–96. ISBN: 9781450398589. doi: [10.1145/3587828.3587843](https://doi.org/10.1145/3587828.3587843). URL: <https://doi.org/10.1145/3587828.3587843>.
- [31] Camilo Chacón Sartori and Christian Blum. “Boosting a Genetic Algorithm with Graph Neural Networks for Multi-Hop Influence Maximization in Social Networks”. In: *Proceedings of FedCSIS 2022 – 17th Conference on Computer Science and Intelligence Systems*. IEEE, 2022, pp. 363–371.
- [32] Camilo Chacón Sartori and Christian Blum. “Boosting a Genetic Algorithm with Graph Neural Networks for Multi-Hop Influence Maximization in Social Networks”. In: *2022 17th Conference on Computer Science and Intelligence Systems (FedCSIS)*. 2022, pp. 363–371. doi: [10.15439/2022F78](https://doi.org/10.15439/2022F78).
- [33] Camilo Chacón Sartori, Christian Blum, and Gabriela Ochoa. “STNWeb: A new visualization tool for analyzing optimization algorithms”. In: *Software Impacts* 17 (2023), p. 100558. ISSN: 2665-9638. doi: <https://doi.org/10.1016/j.simpa.2023.100558>. URL: <https://www.sciencedirect.com/science/article/pii/S2665963823000957>.
- [34] Camilo Chacón Sartori, Christian Blum, and Gabriela Ochoa. “An Extension of STNWeb Functionality: On the Use of Hierarchical Agglomerative Clustering as an Advanced Search Space Partitioning Strategy”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO ’24. Melbourne, VIC, Australia: Association for Computing Machinery, 2024, pp. 151–159. ISBN: 9798400704949. doi: [10.1145/3638529.3654084](https://doi.org/10.1145/3638529.3654084). URL: <https://doi.org/10.1145/3638529.3654084>.
- [35] Camilo Chacón Sartori, Christian Blum, and Gabriela Ochoa. “Large Language Models for the Automated Analysis of Optimization Algorithms”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO ’24. Melbourne, VIC, Australia: As-

sociation for Computing Machinery, 2024, pp. 160–168. ISBN: 9798400704949. doi: [10.1145/3638529.3654086](https://doi.org/10.1145/3638529.3654086). URL: <https://doi.org/10.1145/3638529.3654086>.

- [36] Moses Charikar, Yonatan Naamad, and Anthony Wirth. “On Approximating Target Set Selection”. In: *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2016)*. Ed. by Klaus Jansen et al. Vol. 60. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2016, 4:1–4:16. ISBN: 978-3-95977-018-7. doi: [10.4230/LIPIcs.APPROX-RANDOM.2016.4](https://doi.org/10.4230/LIPIcs.APPROX-RANDOM.2016.4). URL: <http://drops.dagstuhl.de/opus/volltexte/2016/6627>.
- [37] Gary Chartrand. *A First Course in Graph Theory*. Dover Books on Mathematics. Mineola, NY: Dover Publications, Feb. 2012.
- [38] Antônio Augusto Chaves and Luiz Henrique Nogueira Lorena. “An adaptive and near parameter-free BRKGA using Q-learning method”. In: *2021 IEEE Congress on Evolutionary Computation (CEC)*. IEEE. 2021, pp. 2331–2338.
- [39] Angelica Chen et al. “Improving code generation by training with natural Language Feedback”. In: *arXiv [cs.SE]* (Mar. 2023).
- [40] Changqi Chen. *An Empirical Investigation of Correlation between Code Complexity and Bugs*. 2019. arXiv: [1912.01142 \[cs.SE\]](https://arxiv.org/abs/1912.01142). URL: <https://arxiv.org/abs/1912.01142>.
- [41] Deli Chen et al. “Measuring and relieving the over-smoothing problem for graph neural networks from the topological view”. In: *AAAI 2020 - 34th AAAI Conference on Artificial Intelligence*. 2020, pp. 3438–3445. ISBN: 9781577358350. doi: [10.1609/aaai.v34i04.5747](https://doi.org/10.1609/aaai.v34i04.5747). arXiv: [1909.03211](https://arxiv.org/abs/1909.03211). URL: <https://kddcup2016.azurewebsites.net>.
- [42] Hailin Chen et al. *ChatGPT’s One-year Anniversary: Are Open-Source Large Language Models Catching up?* 2024. arXiv: [2311.16989 \[cs.CL\]](https://arxiv.org/abs/2311.16989).
- [43] Liguo Chen et al. *A Survey on Evaluating Large Language Models in Code Generation Tasks*. 2024. arXiv: [2408.16498 \[cs.SE\]](https://arxiv.org/abs/2408.16498). URL: <https://arxiv.org/abs/2408.16498>.
- [44] Mark Chen and et al. *Evaluating Large Language Models Trained on Code*. 2021. arXiv: [2107.03374 \[cs.LG\]](https://arxiv.org/abs/2107.03374). URL: <https://arxiv.org/abs/2107.03374>.
- [45] Wenhui Chen. *Large Language Models are few(1)-shot Table Reasoners*. 2023. arXiv: [2210.06710 \[cs.CL\]](https://arxiv.org/abs/2210.06710).
- [46] Kanzhi Cheng et al. *Vision-Language Models Can Self-Improve Reasoning via Reflection*. 2024. arXiv: [2411.00855 \[cs.LG\]](https://arxiv.org/abs/2411.00855). URL: <https://arxiv.org/abs/2411.00855>.
- [47] Wei-Lin Chiang et al. “Chatbot arena: an open platform for evaluating LLMs by human preference”. In: *Proceedings of the 41st International Conference on Machine Learning. ICML’24*. Vienna, Austria: JMLR.org, 2024.
- [48] Nicos Christofides. “Worst-Case Analysis of a New Heuristic for the Travelling Salesman Problem”. In: *Operations Research Forum 3.1* (Mar. 2022), p. 20. ISSN: 2662-2556. doi: [10.1007/s43069-021-00101-z](https://doi.org/10.1007/s43069-021-00101-z). URL: <https://doi.org/10.1007/s43069-021-00101-z>.
- [49] Trevor D. Collins. “Applying software visualization technology to support the use of evolutionary algorithms”. In: *Journal of Visual Languages & Computing* 14.2 (Apr. 2003),

pp. 123–150. doi: [10.1016/s1045-926x\(02\)00060-5](https://doi.org/10.1016/s1045-926x(02)00060-5). URL: [https://doi.org/10.1016/s1045-926x\(02\)00060-5](https://doi.org/10.1016/s1045-926x(02)00060-5).

[50] Gennaro Cordasco, Luisa Gargano, and Adele Anna Rescigno. “Influence Propagation over Large Scale Social Networks”. In: *Proceedings of the 2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2015*. ASONAM ’15. Paris, France: Association for Computing Machinery, 2015, pp. 1531–1538. ISBN: 9781450338547. doi: [10.1145/2808797.2808888](https://doi.org/10.1145/2808797.2808888). URL: <https://doi.org/10.1145/2808797.2808888>.

[51] Gábor Csardi and Tamás Nepusz. “The igraph software package for complex network research”. In: *InterJournal Complex Systems* (2006), p. 1695.

[52] Chris Cummins et al. “Large Language Models for compiler optimization”. In: *arXiv [cs.PL]* (Sept. 2023).

[53] Enyan Dai et al. *Towards Robust Graph Neural Networks for Noisy Graphs with Sparse Labels*. 2022. arXiv: [2201.00232 \[cs.LG\]](https://arxiv.org/abs/2201.00232). URL: <https://arxiv.org/abs/2201.00232>.

[54] Omid E. David and Iddo Greenbaum. “Genetic Algorithms for Evolving Deep Neural Networks”. In: *Proceedings of the Companion Publication of the 2014 Annual Conference on Genetic and Evolutionary Computation*. GECCO Comp ’14. Vancouver, BC, Canada: Association for Computing Machinery, 2014, pp. 1451–1452. ISBN: 9781450328814. doi: [10.1145/2598394.2602287](https://doi.org/10.1145/2598394.2602287). URL: <https://doi.org/10.1145/2598394.2602287>.

[55] Christopher Davis et al. *Prompting open-source and commercial language models for grammatical error correction of English learner text*. 2024. arXiv: [2401.07702 \[cs.CL\]](https://arxiv.org/abs/2401.07702).

[56] Kapil Devkota et al. “Fast Approximate IsoRank for Scalable Global Alignment of Biological Networks”. In: *bioRxiv* (2023). doi: [10.1101/2023.03.13.532445](https://doi.org/10.1101/2023.03.13.532445). eprint: <https://www.biorxiv.org/content/early/2023/03/15/2023.03.13.532445.full.pdf>. URL: <https://www.biorxiv.org/content/early/2023/03/15/2023.03.13.532445>.

[57] Shifei Ding, Chunyang Su, and Junzhao Yu. “An optimizing BP neural network algorithm based on genetic algorithm”. In: *Artificial Intelligence Review* 36.2 (Feb. 2011), pp. 153–162. doi: [10.1007/s10462-011-9208-z](https://doi.org/10.1007/s10462-011-9208-z).

[58] Qingxiu Dong et al. *A Survey on In-context Learning*. 2024. arXiv: [2301.00234 \[cs.CL\]](https://arxiv.org/abs/2301.00234). URL: <https://arxiv.org/abs/2301.00234>.

[59] M. Dorigo and L. M. Gambardella. “Ant colony system: a cooperative learning approach to the traveling salesman problem”. In: *IEEE Transactions on Evolutionary Computation* 1.1 (1997), pp. 53–66. doi: [10.1109/4235.585892](https://doi.org/10.1109/4235.585892).

[60] Abhimanyu Dubey and et al. *The Llama 3 Herd of Models*. 2024. arXiv: [2407.21783 \[cs.AI\]](https://arxiv.org/abs/2407.21783). URL: <https://arxiv.org/abs/2407.21783>.

[61] Samuel Eilon et al. “Distribution Management-Mathematical Modelling and Practical Analysis”. In: *IEEE Transactions on Systems, Man, and Cybernetics SMC-4.6* (1974), pp. 589–589. doi: [10.1109/TSMC.1974.4309370](https://doi.org/10.1109/TSMC.1974.4309370).

[62] Paul Erdos and Alfred Renyi. “On the evolution of random graphs”. In: *Publ. Math. Inst. Hungary. Acad. Sci.* 5 (1960), pp. 17–61. URL: <http://citesearc.ist.psu.edu/viewdoc/summary?doi=10.1.1.153.5943>.

[63] Philip Feldman, James R. Foulds, and Shimei Pan. *Trapping LLM Hallucinations Using Tagged Context Prompts*. 2023. arXiv: 2306.06085 [cs.CL].

[64] Adrià Fenoy, Filippo Bistaffa, and Alessandro Farinelli. “An attention model for the formation of collectives in real-world domains”. In: *Artificial Intelligence* 328 (2024), p. 104064. ISSN: 0004-3702. doi: <https://doi.org/10.1016/j.artint.2023.104064>. URL: <https://www.sciencedirect.com/science/article/pii/S0004370223002102>.

[65] Matthias Fey and Jan E. Lenssen. “Fast Graph Representation Learning with PyTorch Geometric”. In: *ICLR Workshop on Representation Learning on Graphs and Manifolds*. 2019.

[66] Paola Flocchini et al. “On time versus size for monotone dynamic monopolies in regular topologies”. In: *Journal of Discrete Algorithms* 1.2 (2003), pp. 129–150.

[67] Luciano Floridi. “AI as Agency Without Intelligence: On Chatgpt, Large Language Models, and Other Generative Models”. In: *Philosophy and Technology* 36.1 (2023), pp. 1–7. doi: [10.1007/s13347-023-00621-y](https://doi.org/10.1007/s13347-023-00621-y).

[68] Luciano Floridi and Anna C. Nobre. “Anthropomorphising machines and computerising minds: the crosswiring of languages between Artificial Intelligence and Brain & Cognitive Sciences”. In: *SSRN Electron. J.* (2024).

[69] Thomas M. J. Fruchterman and Edward M. Reingold. “Graph Drawing by Force-directed Placement”. In: *Softw. Pract. Exper.* 21.11 (Nov. 1991), pp. 1129–1164.

[70] Yunfan Gao et al. *Retrieval-Augmented Generation for Large Language Models: A Survey*. 2024. arXiv: 2312.10997 [cs.CL].

[71] Michel Gendreau and Jean-Yves Potvin. “Metaheuristics in combinatorial optimization”. In: *Annals of Operations Research* 140.1 (2005), pp. 189–213.

[72] Michel Gendreau and Jean-Yves Potvin, eds. *Handbook of Metaheuristics*. 3rd. Springer Publishing Company, Incorporated, 2019.

[73] Fred W. Glover. “Tabu Search - Part I”. In: *INFORMS J. Comput.* 1 (1989), pp. 190–206. URL: <https://api.semanticscholar.org/CorpusID:5617719>.

[74] José Fernando Gonçalves and Mauricio G. C. Resende. “Biased random-key genetic algorithms for combinatorial optimization”. In: *Journal of Heuristics* 17.5 (Oct. 2011), pp. 487–525. ISSN: 1572-9397. doi: [10.1007/s10732-010-9143-1](https://doi.org/10.1007/s10732-010-9143-1). URL: <https://doi.org/10.1007/s10732-010-9143-1>.

[75] Mark Granovetter. “Threshold models of collective behavior”. In: *American journal of sociology* 83.6 (1978), pp. 1420–1443.

[76] Darij Grinberg. “An introduction to graph theory”. In: *arXiv [math.HO]* (Aug. 2023).

[77] Miguel Grinberg. *Flask Web Development: Developing Web Applications with Python*. 1st. O'Reilly Media, Inc., 2014. ISBN: 1449372627.

[78] Arnav Gudibande et al. *The False Promise of Imitating Proprietary LLMs*. 2023. arXiv: 2305.15717 [cs.CL]. URL: <https://arxiv.org/abs/2305.15717>.

[79] Taicheng Guo et al. *Large Language Model based Multi-Agents: A Survey of Progress and Challenges*. 2024. arXiv: 2402.01680 [cs.CL].

- [80] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. "Exploring Network Structure, Dynamics, and Function using NetworkX". In: *Proceedings of the 7th Python in Science Conference*. Ed. by Gaël Varoquaux, Travis Vaught, and Jarrod Millman. Pasadena, CA USA, 2008, pp. 11–15.
- [81] Will Hamilton, Zhitao Ying, and Jure Leskovec. "Inductive Representation Learning on Large Graphs". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017.
- [82] Bahareh Harandizadeh, Abel Salinas, and Fred Morstatter. *Risk and Response in Large Language Models: Evaluating Key Threat Categories*. 2024. arXiv: [2403.14988](https://arxiv.org/abs/2403.14988) [cs.CL].
- [83] Tim Hegeman and Alexandru Iosup. *Survey of Graph Analysis Applications*. 2018. arXiv: [1807.00382](https://arxiv.org/abs/1807.00382) [cs.SI]. URL: <https://arxiv.org/abs/1807.00382>.
- [84] Erik Hemberg, Stephen Moskal, and Una-May O'Reilly. *Evolving Code with A Large Language Model*. 2024. arXiv: [2401.07102](https://arxiv.org/abs/2401.07102) [cs.NE]. URL: <https://arxiv.org/abs/2401.07102>.
- [85] Peyman Hosseini et al. *Efficient Solutions For An Intriguing Failure of LLMs: Long Context Window Does Not Mean LLMs Can Analyze Long Sequences Flawlessly*. 2024. arXiv: [2408.01866](https://arxiv.org/abs/2408.01866) [cs.CL]. URL: <https://arxiv.org/abs/2408.01866>.
- [86] Yutao Hu et al. *OmniMedVQA: A New Large-Scale Comprehensive Evaluation Benchmark for Medical LVLM*. 2024. arXiv: [2402.09181](https://arxiv.org/abs/2402.09181) [eess.IV]. URL: <https://arxiv.org/abs/2402.09181>.
- [87] Jie Huang and Kevin Chen-Chuan Chang. *Towards Reasoning in Large Language Models: A Survey*. 2023. arXiv: [2212.10403](https://arxiv.org/abs/2212.10403) [cs.CL]. URL: <https://arxiv.org/abs/2212.10403>.
- [88] Lei Huang et al. *A Survey on Hallucination in Large Language Models: Principles, Taxonomy, Challenges, and Open Questions*. 2023. arXiv: [2311.05232](https://arxiv.org/abs/2311.05232) [cs.CL].
- [89] Sen Huang et al. *When Large Language Model Meets Optimization*. 2024. arXiv: [2405.10098](https://arxiv.org/abs/2405.10098) [cs.NE]. URL: <https://arxiv.org/abs/2405.10098>.
- [90] Marc Huber and Günther R Raidl. "Learning beam search: Utilizing machine learning to guide beam search for solving combinatorial optimization problems". In: *International Conference on Machine Learning, Optimization, and Data Science*. Springer. 2021, pp. 283–298.
- [91] Mohammed Saidul Islam et al. *Are Large Vision Language Models up to the Challenge of Chart Comprehension and Reasoning? An Extensive Investigation into the Capabilities and Limitations of LVLMs*. 2024. arXiv: [2406.00257](https://arxiv.org/abs/2406.00257) [cs.CL]. URL: <https://arxiv.org/abs/2406.00257>.
- [92] Hamish Ivison et al. *Camels in a Changing Climate: Enhancing LM Adaptation with Tulu 2*. 2023. arXiv: [2311.10702](https://arxiv.org/abs/2311.10702) [cs.CL].
- [93] Albert Q. Jiang et al. *Mistral 7B*. 2023. arXiv: [2310.06825](https://arxiv.org/abs/2310.06825) [cs.CL].
- [94] Huiqiang Jiang et al. *LLMLingua: Compressing Prompts for Accelerated Inference of Large Language Models*. 2023. arXiv: [2310.05736](https://arxiv.org/abs/2310.05736) [cs.CL].

[95] Juyong Jiang et al. "A survey on large Language Models for code generation". In: *arXiv [cs.CL]* (June 2024).

[96] Xue Jiang et al. "Self-planning code generation with large language models". en. In: *ACM Trans. Softw. Eng. Methodol.* 33.7 (Sept. 2024), pp. 1–30.

[97] Sathvik Joel, Jie JW Wu, and Fatemeh H Fard. "A survey on LLM-based code generation for Low-resource and Domain-specific programming languages". In: *arXiv [cs.SE]* (Oct. 2024).

[98] Sathvik Joel, Jie JW Wu, and Fatemeh H. Fard. *A Survey on LLM-based Code Generation for Low-Resource and Domain-Specific Programming Languages*. 2024. arXiv: 2410.03981 [cs.SE]. URL: <https://arxiv.org/abs/2410.03981>.

[99] Wei Ju et al. *A Survey of Graph Neural Networks in Real world: Imbalance, Noise, Privacy and OOD Challenges*. 2024. arXiv: 2403.04468 [cs.LG]. URL: <https://arxiv.org/abs/2403.04468>.

[100] Tomihisa Kamada and Satoru Kawai. "An algorithm for drawing general undirected graphs". In: *Information Processing Letters* 31.1 (1989), pp. 7–15. issn: 0020-0190.

[101] U. Kamath et al. *Large Language Models: A Deep Dive: Bridging Theory and Practice*. Springer Nature Switzerland, 2024. isbn: 9783031656477. URL: <https://books.google.es/books?id=kDobEQAAQBAJ>.

[102] Maryam Karimi-Mamaghan et al. "Machine learning at the service of meta-heuristics for solving combinatorial optimization problems: A state-of-the-art". In: *European Journal of Operational Research* 296.2 (2022), pp. 393–422. issn: 0377-2217. doi: <https://doi.org/10.1016/j.ejor.2021.04.032>. URL: <https://www.sciencedirect.com/science/article/pii/S0377221721003623>.

[103] Muhammad Khalifa et al. *Source-Aware Training Enables Knowledge Attribution in Language Models*. 2024. arXiv: 2404.01019 [cs.CL]. URL: <https://arxiv.org/abs/2404.01019>.

[104] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. "Optimization by Simulated Annealing". In: *Science* 220.4598 (1983), pp. 671–680. doi: [10.1126/science.220.4598.671](https://doi.org/10.1126/science.220.4598.671). eprint: <https://www.science.org/doi/pdf/10.1126/science.220.4598.671>. URL: <https://www.science.org/doi/abs/10.1126/science.220.4598.671>.

[105] Jóakim v. Kistowski et al. "How to Build a Benchmark". In: *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering*. ICPE '15. Austin, Texas, USA: Association for Computing Machinery, 2015, pp. 333–336. isbn: 9781450332484. doi: [10.1145/2668930.2688819](https://doi.org/10.1145/2668930.2688819). URL: <https://doi.org/10.1145/2668930.2688819>.

[106] Jon Kleinberg, Mark Sandler, and Aleksandrs Slivkins. "Network failure detection and graph connectivity". In: *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms*. SODA '04. New Orleans, Louisiana: Society for Industrial and Applied Mathematics, 2004, pp. 76–85. isbn: 089871558X.

[107] Takeshi Kojima et al. *Large Language Models are Zero-Shot Reasoners*. 2023. arXiv: 2205.11916 [cs.CL]. URL: <https://arxiv.org/abs/2205.11916>.

- [108] Aobo Kong et al. *Better Zero-Shot Reasoning with Role-Play Prompting*. 2024. arXiv: [2308.07702](https://arxiv.org/abs/2308.07702) [cs.CL]. URL: <https://arxiv.org/abs/2308.07702>.
- [109] Marie-Anne Lachaux et al. “Unsupervised translation of programming languages”. In: *arXiv* [cs.CL] (June 2020).
- [110] Jure Leskovec and Andrej Krevl. *SNAP Datasets: Stanford Large Network Dataset Collection*. <http://snap.stanford.edu/data>. June 2014.
- [111] Jure Leskovec and Rok Sosic. *SNAP: A General Purpose Network Analysis and Graph Mining Library*. 2016. doi: [10.48550/ARXIV.1606.07550](https://doi.org/10.48550/ARXIV.1606.07550). URL: <https://arxiv.org/abs/1606.07550>.
- [112] Brian Lester, Rami Al-Rfou, and Noah Constant. “The Power of Scale for Parameter-Efficient Prompt Tuning”. In: *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. Ed. by Marie-Francine Moens et al. Online and Punta Cana, Dominican Republic: Association for Computational Linguistics, Nov. 2021, pp. 3045–3059. doi: [10.18653/v1/2021.emnlp-main.243](https://doi.org/10.18653/v1/2021.emnlp-main.243). URL: <https://aclanthology.org/2021.emnlp-main.243>.
- [113] Patrick Lewis et al. *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*. 2021. arXiv: [2005.11401](https://arxiv.org/abs/2005.11401) [cs.CL].
- [114] Aitor Lewkowycz et al. *Solving Quantitative Reasoning Problems with Language Models*. 2022. arXiv: [2206.14858](https://arxiv.org/abs/2206.14858) [cs.CL].
- [115] Jia Li et al. “Large language model-Aware In-context learning for code generation”. In: *arXiv* [cs.SE] (Oct. 2023).
- [116] Juanhui Li et al. *Evaluating Graph Neural Networks for Link Prediction: Current Pitfalls and New Benchmarking*. 2023. arXiv: [2306.10453](https://arxiv.org/abs/2306.10453) [cs.LG]. URL: <https://arxiv.org/abs/2306.10453>.
- [117] Yujia Li et al. *Graph Matching Networks for Learning the Similarity of Graph Structured Objects*. 2019. arXiv: [1904.12787](https://arxiv.org/abs/1904.12787) [cs.LG]. URL: <https://arxiv.org/abs/1904.12787>.
- [118] Yunxin Li et al. *VisionGraph: Leveraging Large Multimodal Models for Graph Theory Problems in Visual Context*. 2024. arXiv: [2405.04950](https://arxiv.org/abs/2405.04950) [cs.CV]. URL: <https://arxiv.org/abs/2405.04950>.
- [119] Zekun Li et al. *Guiding Large Language Models via Directional Stimulus Prompting*. 2023. arXiv: [2302.11520](https://arxiv.org/abs/2302.11520) [cs.CL].
- [120] Yannis Liliis and Anthony Savidis. “A Survey of Metaprogramming Languages”. In: *ACM Comput. Surv.* 52.6 (Oct. 2019). ISSN: 0360-0300. doi: [10.1145/3354584](https://doi.org/10.1145/3354584). URL: <https://doi.org/10.1145/3354584>.
- [121] Yen-Ting Lin and Yun-Nung Chen. *LLM-Eval: Unified Multi-Dimensional Automatic Evaluation for Open-Domain Conversations with Large Language Models*. 2023. arXiv: [2305.13711](https://arxiv.org/abs/2305.13711) [cs.CL].
- [122] Defeng Liu et al. “A machine learning framework for neighbor generation in metaheuristic search”. In: *Frontiers in Applied Mathematics and Statistics* 9 (2023), p. 1128181.

- [123] Fei Liu et al. *Evolution of Heuristics: Towards Efficient Automatic Algorithm Design Using Large Language Model*. 2024. arXiv: 2401.02051 [cs.NE]. URL: <https://arxiv.org/abs/2401.02051>.
- [124] Fei Liu et al. *LLM4AD: A Platform for Algorithm Design with Large Language Model*. 2024. arXiv: 2412.17287 [cs.AI]. URL: <https://arxiv.org/abs/2412.17287>.
- [125] Pengfei Liu et al. *Pre-train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing*. 2021. arXiv: 2107.13586 [cs.CL].
- [126] Shengcai Liu et al. *Large Language Models as Evolutionary Optimizers*. 2024. arXiv: 2310.19046 [cs.NE]. URL: <https://arxiv.org/abs/2310.19046>.
- [127] Zhengzhong Liu et al. *LLM360: Towards Fully Transparent Open-Source LLMs*. 2023. arXiv: 2312.06550 [cs.CL].
- [128] Cheng Long and Raymond Chi-Wing Wong. “Minimizing Seed Set for Viral Marketing”. In: *2011 IEEE 11th International Conference on Data Mining*. 2011, pp. 427–436. doi: 10.1109/ICDM.2011.99.
- [129] Manuel López-Ibáñez et al. “The irace package: Iterated Racing for Automatic Algorithm Configuration”. In: *Operations Research Perspectives* 3 (2016), pp. 43–58. doi: 10.1016/j.orp.2016.09.002.
- [130] Andrea De Lorenzo et al. “An analysis of dimensionality reduction techniques for visualizing evolution”. In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. ACM, July 2019. doi: 10.1145/3319619.3326868. URL: <https://doi.org/10.1145/3319619.3326868>.
- [131] Margaret Lundy and Alistair Iain Mees. “Convergence of an annealing algorithm”. In: *Mathematical Programming* 34.1 (Jan. 1986), pp. 111–124. issn: 1436-4646. doi: 10.1007/BF01582166. URL: <https://doi.org/10.1007/BF01582166>.
- [132] Yingwei Ma et al. *At Which Training Stage Does Code Data Help LLMs Reasoning?* 2023. arXiv: 2309.16298 [cs.CL]. URL: <https://arxiv.org/abs/2309.16298>.
- [133] Zeyuan Ma et al. *LLaMoCo: Instruction Tuning of Large Language Models for Optimization Code Generation*. 2024. arXiv: 2403.01131 [math.OC].
- [134] Aman Madaan et al. *Self-Refine: Iterative Refinement with Self-Feedback*. 2023. arXiv: 2303.17651 [cs.CL]. URL: <https://arxiv.org/abs/2303.17651>.
- [135] Paula Maddigan, Andrew Lensen, and Bing Xue. *Explaining Genetic Programming Trees using Large Language Models*. 2024. arXiv: 2403.03397 [cs.NE]. URL: <https://arxiv.org/abs/2403.03397>.
- [136] Paula Maddigan and Teo Susnjak. *Chat2VIS: Fine-Tuning Data Visualisations using Multilingual Natural Language Text and Pre-Trained Large Language Models*. 2023. arXiv: 2303.14292 [cs.HC].
- [137] Paula Maddigan and Teo Susnjak. “Chat2VIS: Generating Data Visualizations via Natural Language Using ChatGPT, Codex and GPT-3 Large Language Models”. In: *IEEE Access* 11 (2023), pp. 45181–45193. doi: 10.1109/ACCESS.2023.3274199.

- [138] Haitao Mao et al. *Revisiting Link Prediction: A Data Perspective*. 2024. arXiv: [2310.00793](https://arxiv.org/abs/2310.00793) [cs.SI]. URL: <https://arxiv.org/abs/2310.00793>.
- [139] Rafael Martí and Gerhard Reinelt. *The Linear Ordering Problem: Exact and Heuristic Methods in Combinatorial Optimization*. Applied Mathematical Sciences. Springer Berlin Heidelberg, 2011.
- [140] Ahmed Masry et al. “UniChart: A Universal Vision-language Pretrained Model for Chart Comprehension and Reasoning”. In: *The 2023 Conference on Empirical Methods in Natural Language Processing*. 2023. URL: <https://openreview.net/forum?id=4MjZNeTCqZ>.
- [141] Ian R. McKenzie et al. *Inverse Scaling: When Bigger Isn’t Better*. 2024. arXiv: [2306.09479](https://arxiv.org/abs/2306.09479) [cs.CL]. URL: <https://arxiv.org/abs/2306.09479>.
- [142] Jesús-Adolfo Mejía-de-Dios and Efrén Mezura-Montes. “Metaheuristics: A Julia Package for Single- and Multi-Objective Optimization”. In: *Journal of Open Source Software* 7.78 (2022), p. 4723. doi: [10.21105/joss.04723](https://doi.org/10.21105/joss.04723). URL: <https://doi.org/10.21105/joss.04723>.
- [143] Ahmad Mheich, Fabrice Wendling, and Mahmoud Hassan. “Brain network similarity: methods and applications”. In: *Network Neuroscience* 4.3 (July 2020), pp. 507–527. issn: 2472-1751. doi: [10.1162/netn_a_00133](https://doi.org/10.1162/netn_a_00133). eprint: https://direct.mit.edu/netn/article-pdf/4/3/507/1867291/netn_a_00133.pdf. URL: https://doi.org/10.1162/netn_a_00133.
- [144] Krzysztof Michalak. “Low-Dimensional Euclidean Embedding for Visualization of Search Spaces in Combinatorial Optimization”. In: *IEEE Transactions on Evolutionary Computation* 23.2 (Apr. 2019), pp. 232–246. doi: [10.1109/tevc.2018.2846636](https://doi.org/10.1109/tevc.2018.2846636). URL: <https://doi.org/10.1109/tevc.2018.2846636>.
- [145] Shoma Miki, Daisuke Yamamoto, and Hiroyuki Ebara. “Applying Deep Learning and Reinforcement Learning to Traveling Salesman Problem”. In: *2018 International Conference on Computing, Electronics & Communications Engineering (iCCECE)*. 2018, pp. 65–70. doi: [10.1109/iccecome.2018.8659266](https://doi.org/10.1109/iccecome.2018.8659266).
- [146] Benjamin A. Miller et al. “Attacking shortest paths by cutting edges”. In: *arXiv* [cs.SI] (Nov. 2022).
- [147] Suvir Mirchandani et al. *Large Language Models as General Pattern Machines*. 2023. arXiv: [2307.04721](https://arxiv.org/abs/2307.04721) [cs.AI]. URL: <https://arxiv.org/abs/2307.04721>.
- [148] David R. Morrison et al. “Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning”. In: *Discrete Optimization* 19 (2016), pp. 79–102. issn: 1572-5286. doi: <https://doi.org/10.1016/j.disopt.2016.01.005>. URL: <https://www.sciencedirect.com/science/article/pii/S1572528616000062>.
- [149] Daniel Müllner. *Modern hierarchical, agglomerative clustering algorithms*. 2011. arXiv: [1109.2378](https://arxiv.org/abs/1109.2378) [stat.ML].
- [150] David Nettleton. “A Synthetic Data Generator for Online Social Network Graphs”. In: *Social Network Analysis and Mining* December 2016, (July 2016). doi: [10.1007/s13278-016-0352-y](https://doi.org/10.1007/s13278-016-0352-y).

- [151] Runbo Ni et al. *FastCover: An Unsupervised Learning Framework for Multi-Hop Influence Maximization in Social Networks*. 2021. doi: [10.48550/ARXIV.2111.00463](https://doi.org/10.48550/ARXIV.2111.00463). url: <https://arxiv.org/abs/2111.00463>.
- [152] Alexander Novikov et al. “AlphaEvolve : A coding agent for scientific and algorithmic discovery”. In: (May 2025).
- [153] Teddy Nurcahyadi and Christian Blum. “Adding Negative Learning to Ant Colony Optimization: A Comprehensive Study”. In: *Mathematics* 9.4 (2021). issn: 2227-7390. doi: [10.3390/math9040361](https://doi.org/10.3390/math9040361). url: <https://www.mdpi.com/2227-7390/9/4/361>.
- [154] Gabriela Ochoa, Katherine M. Malan, and Christian Blum. “Search trajectory networks of population-based algorithms in continuous spaces”. In: *Proceedings of EvoApps 2020 – International Conference on the Applications of Evolutionary Computation*. Springer. 2020, pp. 70–85.
- [155] Gabriela Ochoa, Katherine M. Malan, and Christian Blum. “Search trajectory networks: A tool for analysing and visualising the behaviour of metaheuristics”. In: *Applied Soft Computing* 109 (2021), p. 107492. issn: 1568-4946. doi: <https://doi.org/10.1016/j.asoc.2021.107492>. url: <https://www.sciencedirect.com/science/article/pii/S1568494621004154>.
- [156] OpenAI et al. *GPT-4 Technical Report*. 2024. arXiv: [2303.08774 \[cs.CL\]](https://arxiv.org/abs/2303.08774). url: <https://arxiv.org/abs/2303.08774>.
- [157] OpenAI et al. *GPT-4 Technical Report*. 2023. arXiv: [2303.08774 \[cs.CL\]](https://arxiv.org/abs/2303.08774).
- [158] Long Ouyang et al. *Training language models to follow instructions with human feedback*. 2022. arXiv: [2203.02155 \[cs.CL\]](https://arxiv.org/abs/2203.02155).
- [159] Liangming Pan et al. “Automatically Correcting Large Language Models: Surveying the landscape of diverse self-correction strategies”. In: *arXiv [cs.CL]* (Aug. 2023).
- [160] Rangeet Pan et al. “Lost in translation: A study of bugs introduced by large language models while translating code”. In: *arXiv [cs.SE]* (Aug. 2023).
- [161] Erdős Paul. “On random graphs I”. In: *Publicationes Mathematicae* 6 (1959), p. 290.
- [162] V. Pereira. *pyCombinatorial - A library to solve TSP (Travelling Salesman Problem) using Exact Algorithms, Heuristics and Metaheuristics*. <https://github.com/Valdecy/pyCombinatorial>. Accessed: 2025-03-10. 2022.
- [163] Fernando Pérez, Brian E. Granger, and John D. Hunter. “Python: An Ecosystem for Scientific Computing”. In: *Computing in Science & Engineering* 13.2 (2011), pp. 13–21. doi: [10.1109/MCSE.2010.119](https://doi.org/10.1109/MCSE.2010.119).
- [164] Vagelis Plevris and German Solorzano. “A Collection of 30 Multidimensional Functions for Global Optimization Benchmarking”. In: *Data* 7.4 (2022). issn: 2306-5729. doi: [10.3390/data7040046](https://doi.org/10.3390/data7040046). url: <https://www.mdpi.com/2306-5729/7/4/46>.
- [165] Michal Pluhacek et al. “Leveraging Large Language Models for the Generation of Novel Metaheuristic Optimization Algorithms”. In: *Proceedings of the Companion Conference on Genetic and Evolutionary Computation*. GECCO '23 Companion. Lisbon, Portugal: Association for Computing Machinery, 2023, pp. 1812–1820. isbn: 9798400701207. doi: [10.1145/3583133.3596401](https://doi.org/10.1145/3583133.3596401). url: <https://doi.org/10.1145/3583133.3596401>.

[166] Hartmut Pohlheim. "Multidimensional scaling for evolutionary algorithms – Visualization of the path through search space and solution space using Sammon mapping". In: *Artificial Life* 12 (2 2006), pp. 203–209.

[167] Jean-Yves Potvin. "Genetic algorithms for the traveling salesman problem". In: *Annals of Operations Research* 63.3 (June 1996), pp. 337–370. issn: 1572-9338. doi: 10.1007/BF02125403. url: <https://doi.org/10.1007/BF02125403>.

[168] XiPeng Qiu et al. "Pre-trained models for natural language processing: A survey". In: *Science China Technological Sciences* 63.10 (Sept. 2020), pp. 1872–1897. issn: 1869-1900. doi: 10.1007/s11431-020-1647-3. url: <http://dx.doi.org/10.1007/s11431-020-1647-3>.

[169] Kanchan Rajwar, Kusum Deep, and Swagatam Das. "An exhaustive review of the metaheuristic algorithms for search and optimization: taxonomy, applications, and open challenges". In: *Artificial Intelligence Review* 56.11 (Nov. 2023), pp. 13187–13257. issn: 1573-7462. doi: 10.1007/s10462-023-10470-y. url: <https://doi.org/10.1007/s10462-023-10470-y>.

[170] Aditya Ramesh et al. *Zero-Shot Text-to-Image Generation*. 2021. arXiv: 2102.12092 [cs.CV].

[171] Jairo Enrique Ramírez Sánchez, Camilo Chacón Sartori, and Christian Blum. "Q-Learning Ant Colony Optimization supported by Deep Learning for Target Set Selection". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO '23. Lisbon, Portugal: Association for Computing Machinery, 2023, pp. 357–366. isbn: 9798400701191. doi: 10.1145/3583131.3590396. url: <https://doi.org/10.1145/3583131.3590396>.

[172] Gerhard Reinelt. "TSPLIB - A Traveling Salesman Problem Library." In: *INFORMS J. Comput.* 3.4 (1991), pp. 376–384. url: <http://dblp.uni-trier.de/db/journals/informs/informs3.html#Reinelt91>.

[173] Matthew Renze and Erhan Guven. *Self-Reflection in LLM Agents: Effects on Problem-Solving Performance*. 2024. arXiv: 2405.06682 [cs.CL]. url: <https://arxiv.org/abs/2405.06682>.

[174] Laria Reynolds and Kyle McDonell. "Prompt programming for large language models: Beyond the few-shot paradigm". In: *arXiv* [cs.CL] (Feb. 2021).

[175] Bernardino Romera-Paredes et al. "Mathematical discoveries from program search with large language models". In: *Nature* 625.7995 (Jan. 2024), pp. 468–475. issn: 1476-4687. doi: 10.1038/s41586-023-06924-6. url: <https://doi.org/10.1038/s41586-023-06924-6>.

[176] Stefan Ropke and David Pisinger. "An Adaptive Large Neighborhood Search Heuristic for the Pickup and Delivery Problem with Time Windows". In: *Transportation Science* 40.4 (2025/02/28/ 2006). Full publication date: November 2006, pp. 455–472. url: <http://www.jstor.org/stable/25769321>.

[177] Baptiste Rozière et al. "Code Llama: Open foundation models for code". In: *arXiv* [cs.CL] (Aug. 2023).

[178] Pranab Sahoo et al. *A Systematic Survey of Prompt Engineering in Large Language Models: Techniques and Applications*. 2024. arXiv: 2402.07927 [cs.AI]. URL: <https://arxiv.org/abs/2402.07927>.

[179] Victor Sanh et al. *Multitask Prompted Training Enables Zero-Shot Task Generalization*. 2022. arXiv: 2110.08207 [cs.LG].

[180] Camilo Chacón Sartori. *Architectures of Error: A Philosophical Inquiry into AI-Generated and Human-Generated Code*. <https://ssrn.com/abstract=5265751>. Available at SSRN: <https://ssrn.com/abstract=5265751> or <http://dx.doi.org/10.2139/ssrn.5265751>. May 2025.

[181] Camilo Chacón Sartori and Christian Blum. “STNWeb for the Analysis of Optimization Algorithms: A Short Introduction”. In: *Metaheuristics: 15th International Conference, MIC 2024, Lorient, France, June 4–7, 2024, Proceedings, Part II*. Lorient, France: Springer-Verlag, 2024, pp. 367–372. ISBN: 978-3-031-62921-1. doi: 10.1007/978-3-031-62922-8_29. URL: https://doi.org/10.1007/978-3-031-62922-8_29.

[182] Camilo Chacón Sartori and Christian Blum. *Combinatorial Optimization for All: Using LLMs to Aid Non-Experts in Improving Optimization Algorithms*. 2025. arXiv: 2503.10968 [cs.AI]. URL: <https://arxiv.org/abs/2503.10968>.

[183] Camilo Chacón Sartori and Christian Blum. *Improving Existing Optimization Algorithms with LLMs*. 2025. arXiv: 2502.08298 [cs.AI]. URL: <https://arxiv.org/abs/2502.08298>.

[184] Camilo Chacón Sartori, Christian Blum, and Filippo Bistaffa. “VisGraphVar: A benchmark generator for Assessing Variability in Graph Analysis Using Large Vision-Language Models”. In: *IEEE Access* 13 (2025), pp. 21788–21810. doi: 10.1109/ACCESS.2025.3535837.

[185] Camilo Chacón Sartori et al. “Metaheuristics and Large Language Models Join Forces: Toward an Integrated Optimization Approach”. In: *IEEE Access* 13 (2025), pp. 2058–2079. doi: 10.1109/ACCESS.2024.3524176.

[186] Franco Scarselli et al. “The graph neural network model”. In: *IEEE transactions on neural networks* 20.1 (2008), pp. 61–80.

[187] Sander Schulhoff, Michael Ilie, and et al. *The Prompt Report: A Systematic Survey of Prompt Engineering Techniques*. 2025. arXiv: 2406.06608 [cs.CL]. URL: <https://arxiv.org/abs/2406.06608>.

[188] Albert López Serrano and Christian Blum. “A biased random key genetic algorithm applied to target set selection in viral marketing”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO ’22. Boston, Massachusetts: Association for Computing Machinery, 2022, pp. 241–250. ISBN: 9781450392372. doi: 10.1145/3512290.3528785. URL: <https://doi.org/10.1145/3512290.3528785>.

[189] Tianhui Shi et al. “GraphPi: High performance graph pattern matching through effective redundancy elimination”. In: *arXiv* [cs.DC] (Sept. 2020).

[190] Kevin Sim, Quentin Renau, and Emma Hart. *Beyond the Hype: Benchmarking LLM-Evolved Heuristics for Bin Packing*. 2025. arXiv: 2501.11411 [cs.NE]. URL: <https://arxiv.org/abs/2501.11411>.

- [191] Chandan Singh et al. *Rethinking Interpretability in the Era of Large Language Models*. 2024. arXiv: [2402.01761](https://arxiv.org/abs/2402.01761) [cs.CL].
- [192] Charlie Snell et al. *Scaling LLM Test-Time Compute Optimally can be More Effective than Scaling Model Parameters*. 2024. arXiv: [2408.03314](https://arxiv.org/abs/2408.03314) [cs.LG]. URL: <https://arxiv.org/abs/2408.03314>.
- [193] Helena Stegherr, Michael Heider, and Jörg Hähner. “Classifying Metaheuristics: Towards a unified multi-level classification system”. In: *Natural Computing* 21.2 (June 2022), pp. 155–171. ISSN: 1572-9796. doi: [10.1007/s11047-020-09824-0](https://doi.org/10.1007/s11047-020-09824-0). URL: <https://doi.org/10.1007/s11047-020-09824-0>.
- [194] Niki van Stein and Thomas Bäck. “LLaMEA: A Large Language Model Evolutionary Algorithm for Automatically Generating Metaheuristics”. In: *IEEE Transactions on Evolutionary Computation* (2024), pp. 1–1. doi: [10.1109/TEVC.2024.3497793](https://doi.org/10.1109/TEVC.2024.3497793).
- [195] Niki van Stein and Thomas Bäck. *LLaMEA: A Large Language Model Evolutionary Algorithm for Automatically Generating Metaheuristics*. 2024. arXiv: [2405.20132](https://arxiv.org/abs/2405.20132) [cs.NE]. URL: <https://arxiv.org/abs/2405.20132>.
- [196] Niki van Stein et al. *Code Evolution Graphs: Understanding Large Language Model Driven Design of Algorithms*. 2025. arXiv: [2503.16668](https://arxiv.org/abs/2503.16668) [cs.NE]. URL: <https://arxiv.org/abs/2503.16668>.
- [197] Hao Sun, Alihan Hüyük, and Mihaela van der Schaar. *Query-Dependent Prompt Evaluation and Optimization with Offline Inverse RL*. 2023. arXiv: [2309.06553](https://arxiv.org/abs/2309.06553) [cs.CL].
- [198] Hui Sun, Wenju Zhou, and Minrui Fei. “A Survey On Graph Matching In Computer Vision”. In: *2020 13th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI)*. 2020, pp. 225–230. doi: [10.1109/CISP-BMEI51763.2020.9263681](https://doi.org/10.1109/CISP-BMEI51763.2020.9263681).
- [199] Jiao Sun et al. “Investigating explainability of generative AI for code through scenario-based design”. In: *27th International Conference on Intelligent User Interfaces*. New York, NY, USA: ACM, Mar. 2022.
- [200] Xiaofei Sun et al. *Text Classification via Large Language Models*. 2023. arXiv: [2305.08377](https://arxiv.org/abs/2305.08377) [cs.CL].
- [201] Vasileios A. Tatsis and Konstantinos E. Parsopoulos. “Dynamic parameter adaptation in metaheuristics using gradient approximation and line search”. In: *Applied Soft Computing* 74 (2019), pp. 368–384. ISSN: 1568-4946. doi: <https://doi.org/10.1016/j.asoc.2018.09.034>. URL: <https://www.sciencedirect.com/science/article/pii/S1568494618305519>.
- [202] Anthropic Team. *Introducing Claude 3.5 Sonnet — anthropic.com*. <https://www.anthropic.com/news/clause-3-5-sonnet>. [Accessed 02-11-2024]. 2024.
- [203] DeepSeek-AI Team. *DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning*. 2025. arXiv: [2501.12948](https://arxiv.org/abs/2501.12948) [cs.CL]. URL: <https://arxiv.org/abs/2501.12948>.

[204] Gemini Team and et al. *Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context*. 2024. arXiv: 2403.05530 [cs.CL]. URL: <https://arxiv.org/abs/2403.05530>.

[205] Gemini Team et al. *Gemini: A Family of Highly Capable Multimodal Models*. 2023. doi: <https://doi.org/10.48550/arXiv.2312.11805>. arXiv: 2312.11805 [cs.CL].

[206] Meta Team. *The Llama 3 Herd of Models*. 2024. doi: <https://doi.org/10.48550/arXiv.2407.21783>. arXiv: 2407.21783 [cs.AI]. URL: <https://arxiv.org/abs/2407.21783>.

[207] Hugo Touvron et al. *Llama 2: Open Foundation and Fine-Tuned Chat Models*. 2023. doi: <https://doi.org/10.48550/arXiv.2307.09288>. arXiv: 2307.09288 [cs.CL].

[208] Priyan Vaithilingam, Tianyi Zhang, and Elena L. Glassman. “Expectation vs. Experience: Evaluating the Usability of Code Generation Tools Powered by Large Language Models”. In: *Extended Abstracts of the 2022 CHI Conference on Human Factors in Computing Systems*. CHI EA ’22. New Orleans, LA, USA: Association for Computing Machinery, 2022. ISBN: 9781450391566. doi: 10.1145/3491101.3519665. URL: <https://doi.org/10.1145/3491101.3519665>.

[209] Ashish Vaswani et al. *Attention Is All You Need*. 2023. arXiv: 1706.03762 [cs.CL].

[210] Shubham Vatsal and Harsh Dubey. *A Survey of Prompt Engineering Methods in Large Language Models for Different NLP Tasks*. 2024. arXiv: 2407.12994 [cs.CL]. URL: <https://arxiv.org/abs/2407.12994>.

[211] Petar Veličković et al. *Graph Attention Networks*. 2017. doi: 10.48550/ARXIV.1710.10903. URL: <https://arxiv.org/abs/1710.10903>.

[212] Sara Vicente, Vladimir Kolmogorov, and Carsten Rother. “Graph cut based image segmentation with connectivity priors”. In: *2008 IEEE Conference on Computer Vision and Pattern Recognition*. 2008, pp. 1–8. doi: 10.1109/CVPR.2008.4587440.

[213] Irena Petrijevcanin Vuksanovic and Bojan Sudarevic. “Use of web application frameworks in the development of small applications”. In: *2011 Proceedings of the 34th International Convention MIPRO*. 2011, pp. 458–462.

[214] Zhongwei Wan et al. *Efficient Large Language Models: A Survey*. 2024. arXiv: 2312.03863 [cs.CL].

[215] Hanchen Wang et al. “Scientific discovery in the age of artificial intelligence”. In: *Nature* 620.7972 (Aug. 2023), pp. 47–60. doi: 10.1038/s41586-023-06221-. URL: https://ideas.repec.org/a/nat/nature/v620y2023i7972d10.1038_s41586-023-06221-2.html.

[216] Jiaying Wang et al. “Reinforcement learning for the traveling salesman problem: Performance comparison of three algorithms”. In: *The Journal of Engineering* (2023). URL: <https://api.semanticscholar.org/CorpusID:261504600>.

[217] Christopher J. C. H. Watkins and Peter Dayan. “Q-learning”. In: *Machine Learning* 8.3 (May 1992), pp. 279–292. issn: 1573-0565. doi: 10.1007/BF00992698. URL: <https://doi.org/10.1007/BF00992698>.

[218] Albert Webson and Ellie Pavlick. "Do Prompt-Based Models Really Understand the Meaning of their Prompts?" In: *CoRR* abs/2109.01247 (2021). arXiv: 2109.01247. URL: <https://arxiv.org/abs/2109.01247>.

[219] Jason Wei et al. *Emergent Abilities of Large Language Models*. 2022. arXiv: 2206.07682 [cs.CL].

[220] Jason Wei et al. *Chain-of-Thought Prompting Elicits Reasoning in Large Language Models*. 2023. arXiv: 2201.11903 [cs.CL]. URL: <https://arxiv.org/abs/2201.11903>.

[221] Yanbin Wei et al. *GITA: Graph to Visual and Textual Integration for Vision-Language Graph Reasoning*. 2024. arXiv: 2402.02130 [cs.CL]. URL: <https://arxiv.org/abs/2402.02130>.

[222] Yeming Wen et al. *Grounding Data Science Code Generation with Input-Output Specifications*. 2024. arXiv: 2402.08073 [cs.LG]. URL: <https://arxiv.org/abs/2402.08073>.

[223] Colin White et al. *LiveBench: A Challenging, Contamination-Free LLM Benchmark*. 2024. arXiv: 2406.19314 [cs.CL]. URL: <https://arxiv.org/abs/2406.19314>.

[224] Marco Wiering and Martijn van Otterlo. *Reinforcement Learning: State-of-the-Art*. Springer Publishing Company, Incorporated, 2014.

[225] Evan M. Williams and Kathleen M. Carley. "Multimodal LLMs struggle with basic Visual Network Analysis: A VNA benchmark". In: *arXiv* [cs.CV] (May 2024).

[226] David Hilton Wolpert and William G. Macready. "No free lunch theorems for optimization". In: *Trans. Evol. Comp* 1.1 (Apr. 1997), pp. 67–82. ISSN: 1089-778X. doi: 10.1109/4235.585893. URL: <https://doi.org/10.1109/4235.585893>.

[227] Lingfei Wu et al., eds. *Graph Neural Networks: Foundations, Frontiers, and Applications*. Springer, Singapore, 2022. URL: <https://link.springer.com/book/10.1007/978-981-16-6054-2>.

[228] Yiqi Wu et al. *GPT-4o: Visual perception performance of multimodal large language models in piglet activity understanding*. 2024. arXiv: 2406.09781 [cs.CV]. URL: <https://arxiv.org/abs/2406.09781>.

[229] Zonghan Wu et al. "A Comprehensive Survey on Graph Neural Networks". In: *IEEE Transactions on Neural Networks and Learning Systems* 32.1 (Jan. 2021), pp. 4–24. doi: 10.1109/tnnls.2020.2978386. URL: <https://doi.org/10.1109%2Ftnnls.2020.2978386>.

[230] Zhiheng Xi et al. *The Rise and Potential of Large Language Model Based Agents: A Survey*. 2023. arXiv: 2309.07864 [cs.AI].

[231] Shuhong Xiao et al. *Prototype2Code: End-to-end Front-end Code Generation from UI Design Prototypes*. 2024. arXiv: 2405.04975 [cs.SE]. URL: <https://arxiv.org/abs/2405.04975>.

[232] Yingjie Xing et al. "An Improved Adaptive Genetic Algorithm for Job-Shop Scheduling Problem". In: *Third International Conference on Natural Computation (ICNC 2007)*. Vol. 4. 2007, pp. 287–291. doi: 10.1109/ICNC.2007.202.

[233] Keyulu Xu et al. *How Powerful are Graph Neural Networks?* 2018. doi: 10.48550/ARXIV.1810.00826. URL: <https://arxiv.org/abs/1810.00826>.

- [234] An Yang et al. *Qwen2 Technical Report*. 2024. arXiv: 2407.10671 [cs.CL]. URL: <https://arxiv.org/abs/2407.10671>.
- [235] Chengrun Yang et al. "Large Language Models as Optimizers". In: *The Twelfth International Conference on Learning Representations*. 2024. URL: <https://openreview.net/forum?id=Bb4VGOWELI>.
- [236] Haoran Ye et al. "ReEvo: Large Language Models as Hyper-Heuristics with Reflective Evolution". In: *Advances in Neural Information Processing Systems*. Ed. by A. Globerson et al. Vol. 37. Curran Associates, Inc., 2024, pp. 43571–43608. URL: https://proceedings.neurips.cc/paper_files/paper/2024/file/4ced59d480e07d290b6f29fc8798f195-Paper-Conference.pdf.
- [237] Shukang Yin et al. *A Survey on Multimodal Large Language Models*. 2023. arXiv: 2306.13549 [cs.CV].
- [238] Daoguang Zan et al. *Large Language Models Meet NL2Code: A Survey*. 2023. arXiv: 2212.09420 [cs.SE]. URL: <https://arxiv.org/abs/2212.09420>.
- [239] Mohamed Aymen Zermani et al. "FPGA-based hardware implementation of chaotic opposition-based arithmetic optimization algorithm". In: *Applied Soft Computing* 154 (2024), p. 111352.
- [240] Yuexiang Zhai et al. *Fine-Tuning Large Vision-Language Models as Decision-Making Agents via Reinforcement Learning*. 2024. arXiv: 2405.10292 [cs.AI]. URL: <https://arxiv.org/abs/2405.10292>.
- [241] Biao Zhang, Barry Haddow, and Alexandra Birch. *Prompting Large Language Model for Machine Translation: A Case Study*. 2023. arXiv: 2301.07069 [cs.CL].
- [242] Muhan Zhang and Yixin Chen. *Link Prediction Based on Graph Neural Networks*. 2018. arXiv: 1802.09691 [cs.LG]. URL: <https://arxiv.org/abs/1802.09691>.
- [243] Wenxuan Zhang et al. *Sentiment Analysis in the Era of Large Language Models: A Reality Check*. 2023. arXiv: 2305.15005 [cs.CL].
- [244] Zeyu Zhang et al. *A Survey on the Memory Mechanism of Large Language Model based Agents*. 2024. arXiv: 2404.13501 [cs.AI]. URL: <https://arxiv.org/abs/2404.13501>.
- [245] Haiyan Zhao et al. *Explainability for Large Language Models: A Survey*. 2023. arXiv: 2309.01029 [cs.CL].
- [246] Lianmin Zheng et al. *Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena*. 2023. arXiv: 2306.05685 [cs.CL].
- [247] Zibin Zheng et al. *A Survey of Large Language Models for Code: Evolution, Benchmarking, and Future Trends*. 2024. arXiv: 2311.10372 [cs.SE].
- [248] Yongchao Zhou et al. *Large Language Models Are Human-Level Prompt Engineers*. 2023. arXiv: 2211.01910 [cs.LG]. URL: <https://arxiv.org/abs/2211.01910>.
- [249] Kaijie Zhu et al. *PromptBench: Towards Evaluating the Robustness of Large Language Models on Adversarial Prompts*. 2023. arXiv: 2306.04528 [cs.CL].

[250] Chengke Zou et al. *DynaMath: A Dynamic Visual Benchmark for Evaluating Mathematical Reasoning Robustness of Vision Language Models*. 2025. arXiv: 2411.00836 [cs.CV]. URL: <https://arxiv.org/abs/2411.00836>.

