Proposal Title

A Comparative Study of Foundational Search Algorithms for RAG Hyperparameter Tuning

Author names

Taha El Mouatadir
Hesam Nasiri

Introduction

1. **Overall Idea:** This project will implement and compare the performance of some of the metaheuristics algorithms seen in class to solve the problem of hyperparameter tuning for a Retrieval-Augmented Generation (RAG) system. The specific parameters to be tuned will be chunk_size and top_k, as these are identified in the literature as highly sensitive.
2. **Significance:** As established in the literature review (e.g., Gao et al., 2024), the performance of RAG systems is critically dependent on their configuration. Manual tuning is time-consuming and often suboptimal. This project investigates whether simple, foundational Search-Based Software Engineering (SBSE) techniques can provide a cost-effective and efficient alternative to manual tuning or more complex optimization methods.
3. **Relevance to SBSE:** We are formulating a software engineering problem (configuring a RAG pipeline) as a search-based optimization problem. We will then apply and compare metaheuristic algorithms to navigate the search space of possible chunk_size and top_k values to find a configuration that maximizes a fitness function.

Related Work

The provided literature review ("A Systematic Review of Search-Based Optimization...") confirms that RAG configuration is a critical, high-cost optimization problem. Recent work, such as **Barker et al. (2025)** and **AutoRAG-HP (2024)**, focuses on highly sample-efficient but complex algorithms like Bayesian Optimization and Multi-Armed Bandits to manage the high evaluation cost. This project builds on the same problem definition but takes a different approach. We will build on the work of **Bulhakov et al. (2025)**, which applies a Genetic Algorithm (NSGA-II) to a related LLM tuning problem. Our project will compare a foundational Genetic Algorithm against even simpler course-based methods (Hill Climbing, Random Search) to establish a baseline and determine if these less complex algorithms, which are often overlooked in advanced literature, can provide a cost-effective "good enough" solution.

Research Questions

- **RQ1:** How do the search algorithms compare in their ability to find a high-performing RAG configuration (in terms of F1 score)?
- **RQ2:** What is the difference in computational cost (measured in number of evaluations) for each algorithm to converge on a "good enough" solution?

Data Set and Replication plan

- **System:** A basic RAG pipeline will be implemented in Python using a standard library (e.g., langchain or llama-index).
- **Data Set:** A small, well-defined question-answering dataset will be used, such as a subset of the SQuAD (Stanford Question Answering Dataset) or a custom set of questions based on a few source documents (e.g., the course syllabus itself or a few Wikipedia articles).
- **Fitness Function:** The core of the project will be a fitness function. This function will take a configuration (a chunk_size and top_k value) as input, run the RAG pipeline on all questions in the dataset, and return the F1 score (a metric combining precision and recall) of the generated answers against the ground-truth answers. This F1 score will be the value that the algorithms try to maximize.

Overview

The technical approach is to build a "harness" that connects the search algorithms to the RAG pipeline.

1. A RAG "evaluator" module will be built that takes a configuration, re-processes the documents (if chunk_size changes), and runs the QA evaluation, returning a fitness score.
2. some "solver" modules will be implemented: based on some metaheuristics algorithms that are introduced in the class.
3. Each solver will be run for a fixed computational budget (e.g., 20 total evaluations of the fitness function) to find the best configuration it can within that limit.

Analysis or implementation Procedure

1. **Step 1 (Setup):** Implement the RAG pipeline and the fitness function (evaluator) using the chosen dataset. Define the search space (e.g., chunk_size from 128 to 1024, top_k from 1 to 10).
2. **Step 2 (Algorithm 1):** Implement Random Search. This will simply evaluate random configurations and record the best one found.
3. **Step 3 (Algorithm 2):** Implement another metaheuristic algorithm. It will start at a random configuration, then iteratively "mutate" one parameter (e.g., chunk_size +/- 64 or top_k +/- 1) and accept the new configuration if its fitness is higher.
4. **Step 4 (Analysis):** Each of the algorithms will be run 10 times to account for randomness. The results (best fitness scores and convergence speed) will be compared using box plots to answer RQ1 and RQ2.

Teamwork Plan

- Rag Pipeline (Hesam) Fitness function (Hesam with help of Taha)
- Algorithm 1 (Taha)
- Algorithm 2 (Taha with help of Hesam)
- Analysis (Hesam & Taha)

References

Barker, M., Bell, A., Thomas, E., Carr, J., Andrews, T., & Bhatt, U. (2025). *Faster, cheaper, better: Multi-objective hyperparameter optimization for LLM and RAG systems*. arXiv:2502.18635. https://arxiv.org/abs/2502.18635

Bulhakov, V., d'Aloisio, G., Di Sipio, C., Di Marco, A., & Di Ruscio, D. (2025). *Investigating the role of LLMs hyperparameter tuning and prompt engineering to support domain modeling*. arXiv:2507.14735. https://arxiv.org/abs/2507.14735

Kim, J., Kim, J., Kim, G., Kang, I., & Myaeng, S. H. (2024). *AutoRAG-HP: Automatic online hyper-parameter tuning for retrieval-augmented generation*. In *Findings of the Association for Computational Linguistics: EMNLP 2024* (pp. 3338–3347). Association for Computational Linguistics. https://aclanthology.org/2024.findings-emnlp.223