Project Proposal: A Framework for Algorithm Selection

Artem Kiselev

March 3, 2025

Abstract

This proposal outlines the design and implementation of a flexible framework for algorithm selection, with an initial focus on sorting algorithms. The framework dynamically selects the optimal algorithm based on runtime data characteristics, thereby maximizing performance and efficiency. Ultimately, the goal is to establish a generalized approach to algorithm selection that can be adapted to a wide range of computational problems.

Problem Description

Many problems in computer science can be solved by multiple algorithms, each excelling under different conditions. An algorithm that performs well on one type of data might underperform on another, leading to significant performance degradation—especially for computationally intensive tasks such as NP-hard problems. This project addresses these challenges by employing a strategy known as algorithm selection, where the best-suited algorithm is chosen at runtime based on the specific characteristics of the input data.

The primary objective of this project is to develop an open source C++ library that provides a suite of problem-specific algorithm selection implementations along with tools to enable users to systematically implement and evaluate their own selection strategies. Initially, the framework will be validated through a case study on sorting algorithms. If time permits, the project will also explore the application of these techniques to other domains, such as SAT solving.

Literature Review

Rice's seminal work [1] laid the foundation for algorithm selection by proposing a general framework that maps problem characteristics to algorithm performance.

Mention Satzilla and other NP problem solvers using algorithm selection.

Proposed Approach

The proposed framework is a modular C++ library that implements algorithm selection strategies across various problem domains. The design emphasizes flexibility, enabling users to integrate custom implementations and evaluate performance improvements dynamically. The approach is structured in the following key steps:

1. **Design and Architecture:** The library will be built with a clear separation of concerns. Key modules shall include:
   - **Data Characterization Module:** Extracts and processes input data features (e.g., data size, distribution, and pre-sortedness) that inform the selection process.
   - **Algorithm Profiling Module:** Collects runtime performance data of various algorithms for specified datasets.
   - **Selection Engine:** Uses the characterized data and profiling results to dynamically choose the optimal algorithm—leveraging either rule-based logic or machine learning techniques.
2. **Implementation for Sorting Algorithms:** As an initial test case, the library will include multiple sorting algorithms (e.g., quicksort, mergesort, heapsort). The system will:
   - Profile these algorithms under a range of input conditions.
   - Dynamically select the best algorithm through the use of a model that is tuned based on the results of the profiling.
3. **Extension to Other Domains:** If time permits, the framework will be extended to implement known algorithm selection strategies for other computationally intensive problems, such as SAT solving.
4. **Theoretical Verification:** If time permits, attempt to theoretically verify the improvement of using the developed algorithm selection-based method for sorting in comparison with static sorting algorithms.

Evaluation

The evaluation of the proposed framework will focus on both performance improvements and ease of integration. Key evaluation metrics include:

1. **Algorithmic Efficiency:** Benchmark the performance of the developed selection algorithm for sorting in comparison to static sorting algorithms, on standard sorting datasets with varying sizes and data distributions.

2. **Accuracy of Selection:** Determine the accuracy of the algorithm selection model by computing how often the model selects the optimal algorithm, for a variety of datasets.

3. **Scalability and Flexibility:** Assess how well the library scales with increasing data sizes and a growing pool of algorithms. Determine how difficult it is to implement an already existing algorithm selection model into the library.

4. **Case Studies:** Conduct practical case studies starting with sorting problems and, if possible, extend evaluations to other domains like SAT solving. These studies will provide real-world insights into the library's applicability and performance.

References

[1] J. R. Rice, "The Algorithm Selection Problem," *Advances in Computers*, vol. 15, pp. 65–118, 1976.