Universitat Oberta de Catalunya

MSc in Computational and Mathematical Engineering

PEC2 - Work Development - Phase 1

Author:
Manuel Canedo Tabares

Reviewer:
Dr. Antonio Burguera Burguera

An assignment submitted for the subject:

TFM - Artificial Intelligence

May 3, 2023

Contents

1 Identification of the Work

- **Title:** optimisation Heuristics Performance Analysis and Implementation Guidelines
- **Context:** The goal of this project is to democratize artificial intelligence (AI) research by providing a comprehensive analysis of optimisation heuristics and establishing implementation best practices for them. By addressing the challenges of efficient implementation and optimisation, this project aims to enable researchers without an extensive background in low-level optimisation to utilize these techniques, thereby advancing the AI field effectively.

2 Project Progress and Updates

2.1 Progress on Objectives and Expected Results

Significant progress has been made towards achieving the project's objectives and expected results. A comprehensive review of existing heuristics was conducted, identifying those with computationally intensive tasks where the bottleneck might not be the randomization component.

A literature review on high-performance computing principles revealed that standard algorithm implementation best practices apply to these heuristics. Consequently, Julia's documentation and low-level high-performance principles were studied.

A profiling plan was developed using Julia's built-in tools, such as @time and @profile, to profile the heuristics and identify optimisation candidates. Profiling provided insights into performance characteristics and potential improvement areas. Profiling results will be further reported in Phase 2.

2.2 Changes to the Project Plan and Justifications

Two significant changes were made to the project plan:

1. **Permutation Flow Shop Problem (PFSP) Heuristic Translation Challenges:** The PFSP heuristic translation from Python to Julia required a more significant time investment than initially planned due to its complexity. A complete set of unit tests was developed to ensure correct functionality, and the focus shifted to this single most promising heuristic, discarding the rest.
2. **Profiling Tool Change:** Initially, Flamegraphs were planned for visualizing profiling data. However, after receiving feedback from an ICSO Meta researcher, the decision was made to use Julia's profiling package ProfileView, offering similar functionality and better integration with the Julia programming language.

3 Literature Review

This section outlines the relevance of the identified literature to this study on optimisation heuristics performance analysis and implementation guidelines.

3.1 Optimisation and Artificial Intelligence

These sources provide a comprehensive overview of optimisation heuristics, cover various optimisation techniques, discuss the importance of optimisation in AI research, and contribute to Objective 3 of the thesis ("benchmark heuristics and conclude"). The methodology to select these sources involved searching for foundational texts and articles on metaheuristic optimisation algorithms and their relevance to AI research.

- "An Introduction to optimisation" by Chong, E. K. P., and Zak, S. H. (2013) [1]: Chapter 3 covers metaheuristic optimisation algorithms relevant to the thesis topic, identifying specific algorithms such as Particle Swarm optimisation and Genetic Algorithms as potential candidates for implementation in Julia and performance analysis.
- "Artificial Intelligence: A Modern Approach" by Russell, S. J., and Norvig, P. (2021) [2]: Chapter 4 discusses local search and optimisation, providing insights into optimisation techniques and their importance in AI research. It emphasizes the role of efficient algorithms in solving complex AI problems and offers a theoretical foundation for the optimisation techniques employed in this project.

3.2 Julia Programming Language

These sources introduce the Julia programming language, explain its potential for high-performance computing, offer specific performance tips for Julia, and are essential for Objective 2 of the thesis ("optimise heuristics using high-performance computing principles"). The selection criteria focused on performance optimisation and best practices in the Julia programming language.

- "The Julia Programming Language: Fast Dynamic Dispatch for High-Performance Technical Computing" by Bezanson, J., Karpinski, S., Shah, V., and Edelman, A. (2012) [3]: This paper introduces Julia and its features for high-performance computing, highlighting its Just-In-Time (JIT) compilation for efficient code execution, making it suitable for optimising heuristics in this project.
- "Performance Tips for Julia" by Julia Documentation (latest version) [4]: This official documentation provides practical tips and best practices for optimising code in Julia relevant to the optimisation aspect of the thesis. Specific information, such as using type annotations, avoiding global variables, and leveraging multiple dispatches, has been considered during Julia's implementation and optimisation of heuristics.

3.3 Comparison of Programming Languages

The following paper compares various programming languages, including Python and Julia, highlights their relative strengths and weaknesses, and offers insights into language choice for implementing optimisation heuristics. The paper was selected for its comprehensive comparison of programming languages, including Python and Julia.

- "A Comparative Study of Programming Languages in Rosetta Code" by Nanz, S., and Furia, C. A. (2015) [5]: This study compares the performance, expressiveness, and conciseness of various programming languages, including Python and Julia. The findings indicate that Julia offers improved performance over Python while retaining similar expressiveness and conciseness, influencing the choice of Julia for implementing and optimising heuristics in this project.

3.4 High-Performance Computing

This book explains high-performance computing principles, covers optimisation techniques for various programming languages, and is relevant to Objective 4 of the thesis ("develop generalisable guidelines"). The book was chosen for its comprehensive coverage of high-performance computing principles and techniques, which can be applied to the heuristics implemented in the Julia programming language.

- "High-Performance Computing: Modern Systems and Practices" by Connolly, T., and Begg, C. (2017) [6]: This book covers various techniques for optimising code performance across different programming languages, including Julia. The text presents parallelism, vectorisation, and other optimisation strategies that can be applied to the heuristics implemented in the Julia programming language. These techniques have been considered in the optimisation phase of the project to achieve maximum performance gains.

These resources provide a strong foundation for understanding and implementing optimisation heuristics in the Julia programming language, as well as for analyzing their performance and creating generalisable guidelines for their use in AI research. By incorporating the insights gained from these sources, the thesis can contribute to the democratization of AI research and help researchers without a low-level optimisation background to implement and utilize optimisation heuristics effectively.

4 List of Tasks Carried Out

4.1 Planned Activities

The following planned activities were carried out during the project:

- Review of existing Python heuristics developed by the ICSO META research group and identification of heuristics to translate to Julia for this study.

- Conduct of literature review on high-performance computing principles to determine relevant techniques for optimising heuristics in Julia.
- Translation of the selected Python heuristic to Julia and verification of the same results as the original Python version.
- Development of a profiling plan to identify performance bottlenecks in the Julia implementation, including multiple iterations of profiling, result analysis, and code refinement.

4.2 Unplanned Activities

The following unplanned activities were carried out during the project:

- Implementation of a set of unit tests for the selected heuristic to ensure that optimisation efforts do not compromise functionality.
- Revision of the profiling and optimisation plan based on feedback from an ICSO Meta researcher, focusing on Julia's built-in profiling tools.

5 Deviations in Timing and Mitigation Actions

5.1 Deviations

During the project, a deviation in timing occurred while translating the PFSP heuristic, which took twice the initially allocated time due to its complexity. Consequently, the overall project timeline had to be adjusted. No other deviations were encountered during the project.

5.2 Mitigation Actions and Schedule Update

To mitigate the impact of the delay in translating the PFSP heuristic, the time allocation for other tasks was carefully redistributed, ensuring that critical tasks received sufficient attention. Moreover, the number of heuristics targeted for optimisation was reduced to one, allowing for a more focused and in-depth analysis. As a result, the project schedule was updated accordingly:

- Translate existing Python heuristics to Julia (5 weeks)
  - Understand the existing Python code (1 week)
  - Implement and benchmark the code in Julia (4 weeks)
- Profile the heuristics (2 weeks)
  - Learn to use Julia built-in profiling tools (1 week)
  - Profile and identify performance bottlenecks in the code (1 week)

- Optimise the heuristics (4 weeks)
  - Analyze and address performance bottlenecks (3 weeks)
  - Test and validate optimised heuristics (1 week)
- Benchmark heuristics and conclude (2 weeks)
  - Compare the performance of the optimised heuristics against the original implementations (1 week)
  - Analyze and discuss the results (1 week)
- Develop generalisable guidelines and apply feedback (1 week)
- Finalize report write-up and prepare a presentation (2 weeks)

By redistributing the time and adjusting the project's scope, the overall timeline was maintained, and the project's objectives were met without compromising the quality of the work.

6 Partial Results and Attached Deliverables

The following partial results and deliverables are attached to this report:

1. **Profiling plan:** The methodology used to carry out the profiling efforts is detailed in Annex 1 of this report.
2. **Translated heuristic:** The translated heuristic code is in Annex 2 of this report.

7 Comments from the Director

On top of the suggestions from the ICSO Meta researcher mentioned in section 2.1, I received additional formal feedback from the thesis director:

1. Existence of orthography mistakes and typos.
2. Inconsistent and excessive use of enumerations.
3. Necessity of a unified format for bibliography citations.
4. Necessity of an "accessed" field for online references.
5. Alternative ordering for the report sections.

Annexe 1: Profiling Plan

Introduction

This annexe presents the profiling plan for the heuristics implemented in the Julia language. Profiling is a crucial step in the optimisation process, as it enables the analysis of the heuristics' performance and the identification of potential bottlenecks.

The primary objectives of the profiling plan are to:

1. Identify performance bottlenecks in the heuristic
2. Determine potential areas for improvement
3. Evaluate the impact of optimisation efforts

Profiling Methodology

The profiling process consists of the following steps:

1. **Preparing the code and test cases:** Select appropriate test cases that comprehensively evaluate the heuristics.
2. **Running the profiling tools:** Modify the code to use Julia's built-in profiling functions on the selected test cases.
3. **Collecting and analysing the profiling data:** Gather the data and perform a thorough analysis to identify trends and patterns.
4. **Identifying performance bottlenecks:** Pinpoint areas in the heuristics where performance can be improved, focusing on critical bottlenecks.

To evaluate the performance of the heuristics, the following metrics are used:

- Execution time
- Memory allocations

Profiling Tools

The profiling plan utilises Julia's built-in functions for profiling, offering several key features and benefits, including ease of use, integration with the language, and detailed performance data.

Useful Julia Built-in Functions for Profiling

Below are some of the most useful Julia built-in functions for profiling, according to the Julia documentation:

- **@time:** This macro measures the execution time of an expression, along with memory allocations and garbage collection time. It provides a quick overview of a particular code segment's performance.
- **@allocated:** This macro returns the number of bytes allocated during the execution of an expression. It helps identify memory allocation hotspots in the code.
- **@profile:** This macro collects statistical profiling data while running an expression. The gathered data can be analysed using various tools, such as Profile.print() and ProfileView.jl, to visualise and understand the code's performance characteristics.
- **Profile.print():** This function displays the collected profiling data in a human-readable format, enabling the identification of bottlenecks and performance issues.
- **Profile.clear():** This function clears previously collected profiling data, allowing for a clean start when profiling new code sections.

Annex 2: Translated Heuristic

This annexe presents the translated Julia code for the heuristics used in the project. The translated Julia code retains the original functionality of the Python implementation while providing the benefits of the Julia programming language, such as enhanced performance and more efficient memory management.

https://github.com/ManuCanedo/icso-neh

References

[1] E. K. P. Chong and S. H. Zak, *An Introduction to Optimization.* John Wiley & Sons, 2013.

[2] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 4th ed. Pearson, 2021.

[3] J. Bezanson, S. Karpinski, V. B. Shah, and A. Edelman, "The julia programming language: Fast dynamic dispatch for high-performance technical computing," *arXiv preprint arXiv:1209.5145*, 2012.

[4] "Performance tips for julia," Julia Documentation (latest version), accessed: 2-May-2023. [Online]. Available: https://docs.julialang.org/en/v1/manual/performance-tips/

[5] S. Nanz and C. A. Furia, "A comparative study of programming languages in rosetta code," in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, vol. 1. IEEE, 2015, pp. 778–788.

[6] T. Connolly and C. Begg, *High-Performance Computing: Modern Systems and Practices*. Morgan Kaufmann, 2017.