Overview

- Modern optimising compilers offer their users many command-line parameters and options affecting performance.
- Difficult to identify the best parameter configurations for a given application scenario.
- Prefixed default configurations are likely not ideal for specific cases.
- We investigate the performance of general-purpose algorithm configuration tools for finding optimised JavaScript compiler parameter configurations.
- We find that there is great potential for these tools to produce configurations with performance significantly better than the defaults.

JavaScript Compilers

- JavaScript (ECMAScript): among the most popular languages.
- Increasing complexity of web applications
  - advanced compilers, highly-optimised code:
    - These compilers are complex, make many performance-affecting parameters available.
    - How code should be optimised depends on the application, problem data and execution environment.
    - Users and developers want to find configurations optimising performance:
      - runtime, memory, battery usage, and more.
    - Identical code can be run millions of times a day, small improvements translate to massive resource savings.
    - Automated tools to efficiently find these configurations are of great benefit.

Algorithm Configuration

- Many previous approaches to algorithm configuration:
  - random sampling, local search, racing methods.
- More recently, sequential model-based (Bayesian) optimisation, with excellent results.
  - Iteratively construct models of how parameters affect performance.
  - Use models to select new configurations to evaluate.
- Best known model-based configurator (current state of the art) is SMAC.
- SMAC uses random forest models:
  - supports extremely large parameter spaces.
  - mixed discrete and continuous spaces, conditionality and other constraints between parameters.
  - Has been successfully used to advance the state of the art in SAT, CSP, MIP, scheduling, AI planning, machine learning hyperparameter optimisation, and more.

Important Parameters

We used ablation analysis to obtain the parameters most responsible for improved performance.
These trends were identified using model-based optimisation.

JavaScriptCore:

- Objective: Octane splay
  - numberOfGCMarshals (38%)
  - collectionTimerMaxPercentCPU (6%)
  - forceDFGCodeBlockLiveness (16%)

Future Work

This is an ongoing research project, and we plan to further this work by:

- Applying our approach to LLVM optimisation passes, including C/C++ benchmarks and the Soot compiler.
- Including, and defining the parameter space for, Microsoft’s ChakraCore JavaScript engine.
- More thorough investigation of optimised configuration performance on single benchmark instances.
- Further classification and identification of trends for important parameters across scenarios.
- Optimisation for metrics other than runtime, including memory footprint and battery consumption.

Selected Publications


Acknowledgements

The experimental portion of this work has been supported by a Microsoft Azure grant of cloud computing resources.

Further Information

We’re sorry to be unable to present this poster in person. We’d much prefer to be in sunny Barcelona than rainy Vancouver! If you would like more information about this ongoing project, please visit our project site:

https://www.cs.ubc.ca/labs/beta/Projects/ComplexParameterOptimization