Efficiently Optimizing Hyperparameters for the Gumtree Hybrid Code Differencing Algorithm

TU Delft

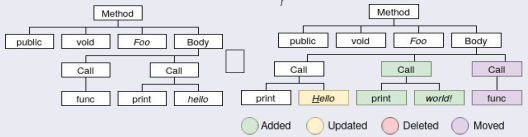
1 Introduction

Code

```
public void Foo() {
    func();
    print("hello");
}
```

```
public void Foo() {
    print("Hello");
    print("world!");
    func();
}
```

Abstract Syntax Tree



In order to analyze software evolution, it is useful to compare two Abstract Syntax Trees (ASTs), which is the purpose of *diff algorithms* such as Gumtree.
HyperAST focuses on improving scalability of Gumtree, which is a common issue with AST diff algorithms and can prevent the analysis of large codebases.

2 Background

Gumtree is a state-of-the-art algorithm for AST differencing. It consists of three phases, with the third called *recovery*. Three main algorithms exist for this part:
- **Optimal**: finds all mappings, but has a complexity of O(n³);
- **Simple**: uses heuristics to find good results faster;
- **Hybrid**: switches between Optimal and Simple depending on the size of the tree.

The Gumtree algorithm has 5 hyperparameters:

Hyperparameter	Description	Default	Range
`bottom-up_matcher`	Recovery phase variant	Greedy	{Greedy, Simple, Hybrid}
`priority_metric`	How to calculate priority for first phase	Height	{Size, Height}
`min_priority`	Minimum priority to match for first phase	1	[1;5]
`sim_threshold`	Minimum similarity between trees for second phase	0.5	[0.1; 1]
`max_size`	Maximum tree size to match for second phase	1000	[100;2000]

The Diff Auto Tuning approach optimizes these hyperparameters, either on a large dataset (global tuning) or for a single case (local tuning), using either grid search or TPE via the HyperOpt and Optuna libraries.

3 Research question

How can we optimize the hyperparameters of the Gumtree Hybrid algorithm to enable efficient and accurate differencing on very large ASTs?

4 Contributions

Porting Gumtree Hybrid to HyperAST
From the reference Gumtree implementation in Java to HyperAST in Rust. The output of both was compared to test the correctness of the implementation.

Adapting HyperDiff optimizations
Making the algorithm lazy in order to delay and possibly skip expensive decompression operations on certain subtrees.

Local hyperparameter optimization for HyperAST
Using the rust-tpe library, first following the DAT approach and then optimizing only the `max_size` hyperparameter.

5 Methodology

We mainly used benchmarking, for which we measure both performance and output quality.

Performance: We measured the execution time of the bottom-up matching phase only for each pair of commits, and used Mann-Whitney U-tests for statistical analysis since the benchmark results are not normally distributed.

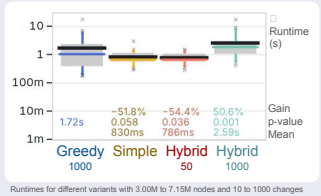
Output quality: We used the length of an edit script, which is a commonly accepted proxy for its quality.

We used the dataset from the HyperAST evaluation, with 18 popular large Java repositories. The 3 largest repositories were excluded to allow for reasonable computation times on available hardware.
We also used the Defects4j file pair dataset for output equality tests and smaller benchmarks.

6 Results

Effect of hyperparameters

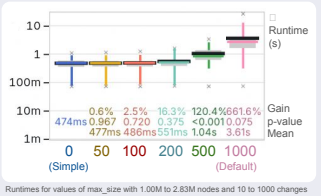
`bottom-up_matcher`



Runtimes for different variants with 3.00M to 7.15M nodes and 10 to 1000 changes

The Simple variant generally performs better, but the improvement is less marked with larger ASTs and few changes. Hybrid is 10 to 70% slower than Greedy.
Hybrid also often produces better outputs.

`max_size`



Runtimes for values of max_size with 1.00M to 2.83M nodes and 10 to 1000 changes

Computation time increases with `max_size`, but the difference is less marked with larger ASTs and few changes.
We found no pattern between `max_size` and edit script quality.

`min_priority`

We found worse output quality with values other than the default (1), with no major performance improvement.

HyperDiff optimizations

We observed a decrease in performance between 70% and 355%, likely due to an implementation bug.

Hyperparameter tuning approach

Global optimization is unfeasible on available hardware since runtime exceeds practical limits.
Local optimization is possible on smaller repositories, and can be used to find values of `max_size` for the Hybrid variant that give a better edit script than Simple on a single commit.
On the gson repository, TPE with 10 evaluations found improvements in 2 out of 17 cases, with a median runtime of 283.1s.

7 Conclusions

If **quick runtimes** are needed, for example for integration in developer tools, we recommend using the **Gumtree Simple** algorithm, which has no hyperparameters and generally produces good results.
If a **shorter edit script** is more important, for example for research into software evolution, we recommend using **local optimization** with the Hybrid bottom-up matcher. Only the `max_size` hyperparameter is worth optimizing, to improve efficiency.

Alexander Nitters | Supervisors: Quentin Le Dilavrec, Carolin Brandt

See paper for references