Scalable Structural Code Diffs

Comparing Gumtree Greedy and Gumtree Stable adapted for scaling

TU Delft

Introduction

Text-based differencing is fast but does not capture the semantics of code. AST[1]-based differencing does and allows for more fine-grained diffs. *Gumtree* is a well-known reference implementation of multiple structural diff heuristics. *Gumtree* Greedy [1] was the first heuristic but does not scale well with large trees. *Gumtree* Simple [2] is designed to scale better but makes stronger assumptions. The HyperAST [3] is a data structure that improves scaling by *"leveraging code redundancy through space and time"*.

```
1 public void Foo() {
2     func();
3     print("hello");
4 }
```

```
1 public void Foo() {
2     print("Hello");
3     print("world!");
4     func();
5 }
```

1. AST: Abstract Syntax Tree

Research Question

Compared to Gumtree Greedy, does Gumtree Simple enable additional adaptations helping with scalability?

Background

The Greedy algorithm consists of three phases: *Top-Down*, *Bottom-Up* and *Recovery*. The recovery phase uses an expensive TED[2] algorithm and runs each time after a Bottom-up mapping. Because of the cost there is a *Size threshold* which limits the subtree size on which recovery is run (we used 200 and 1000) in the Greedy heuristic.

The Simple algorithm differentiates itself from Greedy in the recovery phase. It uses a simpler strategy that can be broken up into three sub-phases. The first two sub-phases search for *Exact* and *Structural –Isomorphism* respectively, the final one searches for nodes that have unambiguous type matchings.

2.TED: Tree Edit Distance

Relative performance compared to Greedy-1000 (in %)

		GitHub Java	Defects4J
Total mappings	Greedy-200	-0.472	-0.183
	Simple	-0.317	0.772
	Lazy Greedy-200	-0.470	-0.183
	Lazy Simple	-0.517	0.726
CPU-cycles	Greedy-200	-95.633	-96.828
	Simple	-99.900	-99.732
	Lazy Greedy-1000	0.282	3.348
	Lazy Greedy-200	-95.526	-96.682
	Lazy Simple	-99.779	-99.510

Methods

We ported the Simple heuristic to the HyperAST framework and implemented a variant optimized to leverage HyperAST structure (*Lazy Simple* in the table). Our benchmarks evaluated three metrics: *number of mappings*, *CPU-cycles*, and *runtime*. The number of mappings is a proxy for the quality of the resulting diff. As a baseline, we used the original Greedy algorithm with a size threshold of 1000 (*Greedy-1000*). We then benchmarked both versions of the Simple heuristic. For all heuristics, we used the default similarity threshold of 0.5.

Results

The Simple heuristic performed better than Greedy, as expected based on the Results of Falleri et al. [2]. That the lazy variants often performed worse than their non-lazy counterpart is in stark contrast to the results found bij Le Dilavrec et al. [4]. This discrepancy suggests a bug in our code and/or benchmarking setup.

Conclusion

The well structed nature of the recovery phase of the Simple heuristic makes it easy to reason about, and is more modular than the complex recovery phase of the Greedy Heuristic. This enables Simple to be easier adapted for Scaling Compared to Greedy.

[1] J.-R. Falleri, F. Morandat, X. Blanc, M. Martinez, and M. Monperrus, "Fine-grained and Accurate Source Code Differencing," in *Proceedings of the International Conference on Automated Software Engineering*, Västeras, Sweden, 2014, pp. 313–324. doi: 10.1145/2642937.2642982.

[2] J.-R. Falleri and M. Martinez, "Fine-grained, accurate and scalable source differencing," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, in ICSE '24. New York, NY, USA: Association for Computing Machinery, Apr. 2024, pp. 1–12. doi: 10.1145/3597503.3639148.

[3] Q. Le Dilavrec, D. E. Khelladi, A. Blouin, and J.-M. Jézéquel, "HyperAST: Enabling Efficient Analysis of Software Histories at Scale," in *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, in ASE '22. New York, NY, USA: Association for Computing Machinery, Jan. 2023, pp. 1–12. doi: 10.1145/3551349.3560423.

[4] Q. Le Dilavrec, D. E. Khelladi, A. Blouin, and J.-M. Jézéquel, "HyperDiff: Computing Source Code Diffs at Scale," in *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, in ESEC/FSE 2023. New York, NY, USA: Association for Computing Machinery, Nov. 2023, pp. 288–299. doi: 10.1145/3611643.3616312.

Author: Ruben van Seventer

Supervised by: Carolin Brandt and Quentin Le Dilavrec

Course: CSE3000