Efficiently Optimizing Hyperparameters for the Gumtree Hybrid Code Differencing Algorithm within HyperAST

Alexander Nitters[1]

Supervisor(s): Caroline Brandt[1]**, Quentin Le Dilavrec**[1]

[1]**EEMCS, Delft University of Technology, The Netherlands**

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 25, 2025

Name of the student: Alexander Nitters
Final project course: CSE3000 Research Project
Thesis committee: Caroline Brandt, Quentin Le Dilavrec, Jesper Cockx

Abstract

Code differencing allows understanding changes between different versions of software, especially when using Abstract Syntax Trees (ASTs) to structurally represent code. The Gumtree algorithm is the current state-of-the-art algorithm for AST differencing, however its main drawback is long runtimes, especially for large ASTs. The Simple and Hybrid variants aims to solve this issue with a heuristic-based recovery phase, but their evaluation has been limited to small ASTs. Benchmarks show that the Simple variant results in significant improvements in output quality and performance with very large ASTs, while the Hybrid variant has mixed results. Furthermore, increasing the `max_size` hyperparameter with the Hybrid variant has an unpredictable effect on output quality and a negative effect on performance, but `min_priority` can generally be set to 1. The Diff-Auto-Tuning optimization approach is deemed often unusable for large ASTs, though it can be used to find cases where the Hybrid variant outperforms the Simple variant on a single data point.

1 Introduction

As software evolves, tracking structural changes in code is essential for understanding its development over time. A common way to analyze these changes is computing the difference between two Abstract Syntax Trees (ASTs), which represent the structure of the code. A code differencing algorithm that works with ASTs generally outputs a sequence of edit actions to transform one tree into the other, called an edit script [7]. These edit actions can be, for example, the addition of a new variable declaration node or moving a method node to a different class node.

A widely-used algorithm for doing this is Gumtree [7]. The original Gumtree approach includes an optimal *recovery* phase with a cyclomatic complexity of $O(n^3)$, resulting in large computation times even for relatively small files. To limit this, the optimal recovery phase is only run on subtrees smaller than a certain `max_size` threshold, by default 1000. The `max_size` can be increased to offer better edit scripts, or decreased to make the algorithm run faster.

With the recovery phase being the most expensive, designing a better one would be one of the ways to significantly improve the performance of Gumtree. A *simple* recovery phase based on heuristics was designed [6] with a cyclomatic complexity of $O(n^2)$, resulting in significantly better runtimes and edit scripts of similar or close quality to the optimal version. However, in some cases the optimal version does produce a better edit script. In order to combine these the *hybrid* approach uses the optimal algorithm for subtrees smaller than the `max_size` threshold, but switches to the simple algorithm for larger subtrees.

The Diff-Auto-Tuning (DAT) approach was proposed in order to optimize the hyperparameters, including `max_height` but also some others [11]. However, Martinez et al. mostly focus on improving the quality of edit scripts. There is an evaluation of the performance for some aspects, but it only uses small file pairs for the hyperparameter optimization. ASTs with multiple millions of nodes, such as those that might be generated for a diff between two full commits in a large repository, are not considered.

More recently, a new variation of the AST, called HyperAST, was developed to address scalability issues with Gumtree using de-duplication [9]. The original Gumtree Greedy algorithm was re-implemented to work with HyperAST [10].

In this paper, we will look at how the Gumtree Hybrid algorithm can enable efficient differencing on large ASTs, using HyperAST. The Simple and Hybrid recovery phases have currently only been evaluated on smaller ASTs, generated from a single file. The effect of this new recovery phase on very large ASTs has not been studied yet to our knowledge, and HyperDiff has not implemented the Hybrid recovery. Implementing Simple recovery within HyperDiff allows us to analyse its scalability and the effect of the hyperparameters. This gives insights into default good values for these hyperparameters and how they affect both output quality and performance. Finally, we will evaluate data-driven hyperparameter optimization approaches, similar to DAT, for feasibility and performance.

The rest of this paper is structured as follows. Section 2 presents the terminology, Gumtree algorithm, HyperAST and Diff-Auto-Tuning in detail, as well as other related work and the research gap. Section 3 explains the contributions of this paper, including the implementation of Hybrid recovery and hyperparameter optimization in HyperAST. Section 4 presents the experimental setup for the benchmarks and the results of the evaluations, and Section 5 discusses these results. Section 6 presents Responsible Research considerations, and finally Section 7 concludes the paper.

2 Background

This section introduces key terminology, explains Gumtree Hybrid, HyperAST and DAT hyperparameter optimization, and introduces related work and the research gap.

2.1 Terminology

An **Abstract Syntax Tree (AST)** represents code as a tree structure, such as in Figure 1. Each node has a label that represents the grammar element, such as `MethodDeclarations` or `Identifier`, and an optional value, which can for example be the name of a variable. Tools such as the Eclipse JDT Java parser or the more recent tree-sitter can generate an AST from a piece of code. We will use the latter as it is both included in later versions of the Java Gumtree implementation and used by HyperAST.

Differencing algorithms (diff algorithms) compute the differences between two versions of a program, the *source* and *destination*. Most common diff algorithms, such as the one used by git, work on the level of lines in a file. However, there are also many diff algorithms that work with ASTs, including in chronological order X-Diff [14], Change Distiller [8], and Gumtree [7]. AST diff algorithms generally

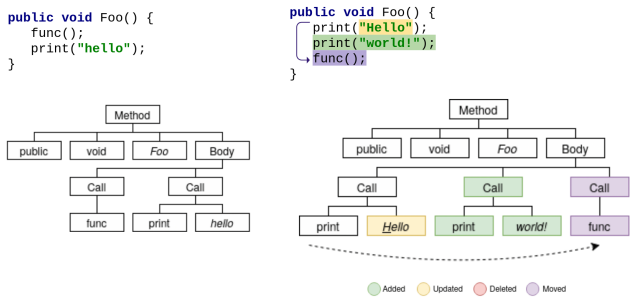

Figure 1: Two example code excerpts and their corresponding Abstract Syntax Trees, with the changes highlighted. Nodes with the identifier label have the value in italic.

output *mappings*, which *match* one node in the source tree with one node in the destination tree.

An **edit script** describes a sequence of operations that should be applied to an AST in order to transform it into another AST, and can be generated from a set of mappings. The most common set of operations includes inserting, deleting or updating the label of a node (resp. `insert-node`, `delete-node`, `update-node`), and moving an entire subtree (`move-subtree`). More types of operations have been proposed, such as inserting or deleting an entire subtree [6], or copy-and-pasting a subtree [12]. The length of an edit script is a widely-accepted proxy for its quality [7].

2.2 The Gumtree Hybrid algorithm

Gumtree is a state-of-the-art algorithm for differencing ASTs [7]. This algorithm is divided into three phases, each of which adds new mappings visible on the example in Figure 2.

The first phase, called **top-down matching**, matches the largest isomorphic subtrees in the two ASTs being compared, but only if they are strictly larger than a predefined size. In the example, the only identical subtree of size strictly greater than 1 is the *func* function call. In this case the function has been moved to a different place, which the first phase is able to detect since it does not look at the location of the subtrees.

The second phase, **bottom-up matching**, propagates the previous matches up to their parents if the parents have the same label. In the example, in both the source and destination trees the parent of the previously matched *Call* node has the label *Body*, so this node will also be matched. This match is again propagated up, matching the *Method* nodes.

In the original version of Gumtree, the third phase, called **recovery**, is called on each of the nodes mapped during bottom-up matching in order to find more mappings in the descendants. To do so, it uses an optimal Tree Edit Distance algorithm to determine the mappings that minimize the number of `insert-node`, `update-node` and `delete-node` actions. This phase has a worst-case time complexity of $O(n^3)$ where n is the number of nodes in both trees, so it is often not practical to use on large ASTs.

A simpler, heuristic-based recovery phase was developed, that does not always give the optimal result but runs significantly faster [6]. This **Simple recovery** is divided into three steps. The first finds the largest structurally isomorphic sub-

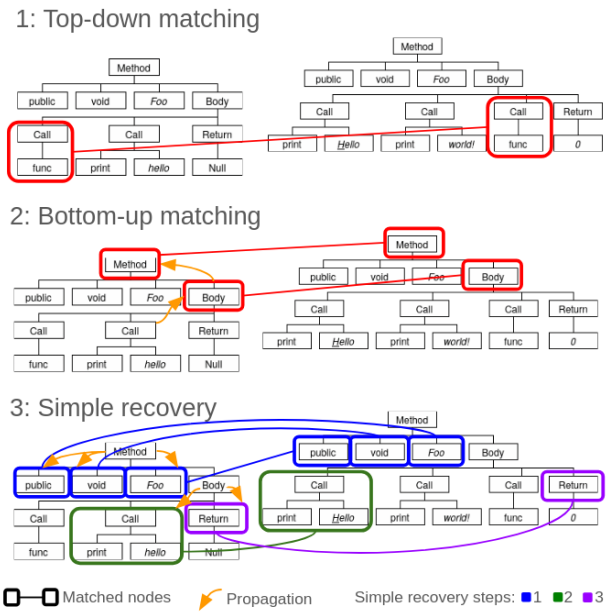

Figure 2: The mappings created in each phase of the Gumtree algorithm with the Simple recovery phase

trees, just like in the top-down phase, but without a minimum size cut-off. In the example from Figure 2, the *public*, *void* and *Foo* nodes are matched during this step. The second step is identical to the first but ignores the values of the nodes, looking only at the labels. Here the subtree with the *Call*, *print* and *Hello* nodes is matched, since *Hello* and *hello* have different values but both have the identifier label. The last step, called histogram matching, searches for node types that appear only once in the children of the nodes matched during the bottom-up phase, matching the *Return* node in the example. The same paper proposes a **Hybrid recovery** where the optimal recovery is used for trees smaller than the `max_size` threshold, and the Simple variant for larger trees.

2.3 HyperAST and HyperDiff

A HyperAST structure stores the AST as a compressed Directed Acyclic Graph, where shared subtrees are deduplicated [9]. This sometimes allows comparing subtrees using referential instead of structural equality, skipping an expensive check. However, it also means that some structural information is not stored. When this information is needed, the nodes are *decompressed*. HyperDiff, the adaptation of Gumtree to work with HyperAST, does so in a lazy way, which leads to some trees remaining compressed and therefore improves the performance [10].

2.4 Diff-Auto-Tuning

Martinez et al. specify the hyperparameter space of GumTree, which is described in Table 1 [11]. They include 5 hyperparameters, but there are dependencies: `sim_threshold` is only applicable for the Greedy variant, and `max_size` is only applicable for the Greedy or Simple variants. This leaves 2210 possible configurations.

Table 1: Hyperparameter space for the GumTree AST Differencing Algorithm [11] (edited)

Hyperparameter	Description	Default	Range	Cardinality
`bottom-up_matcher`	Bottom-up matching algorithm	Greedy	{Greedy, Simple, Hybrid}	3
`priority_metric`	Priority calculator used during top-down matching	Height	{Size, Height}	2
`min_priority`	Minimum priority threshold (using `priority_metric`)	1	[1;5]	5
`sim_threshold`	Minimum similarity between AST nodes during bottom-up matching	0.5	[0.1; 1]	10
`max_size`	Maximum AST node size to match during bottom-up matching	1000	[100;2000]	20

The Diff-Auto-Tuning (DAT) approach uses multiple hyperparameter optimization frameworks in order to obtain optimized values for these hyperparameters [11]. The first is a grid search, which is an exhaustive but inefficient search amongst all combinations. Martinez et al. also use two similar python optimization frameworks, Hyperopt [4] and Optuna [1], in order to run a smarter search.

The cost function used by DAT is the length of the edit script. When DAT is used for *local optimization*, a single pair of files is used in the cost function, meaning that the hyperparameters are fitted to a single file. For *global optimization*, the average length of the edit script for all pairs of files in a dataset is used, to find hyperparameter values that work well in general. Martinez et al. recommend at least 10 or 25 evaluations of the cost function with HyperOpt and Optuna, with 50 or 100 evaluations leading to improvements in more cases.

2.5 Other related work

While the paper by Martinez et al. is the most relevant to hyperparameter optimization for Gumtree, there are also many others that have looked into both the quality of the edit scripts produced by Gumtree and its performance.

Alikhanifard and Tsantalis look in depth at the edit script quality of the Greedy and Simple variants among others [2]. While the most common proxy for edit script quality is the number of actions, they also look at more advanced metrics: the accuracy in multi-mapping, the number semantically incompatible mappings, the accuracy in matching program elements or the percentage of perfect diffs. They show that Gumtree Greedy generally produces more problematic mappings than Gumtree Simple. They also present a new tool, RefactoringMiner, which outperforms Gumtree in edit script quality, but which is out of scope for this paper. However, in their evaluation they always use the default hyperparameter values, so we cannot gain insights into the effects of the hyperparameters on edit script quality from this paper, except for `bottom-up_matcher`.

Hyperparameter optimization in general is a common problem in many fields of computer science, such as machine learning, but the approaches are often similar for different contexts. Bergstra et al. compare various optimization approaches for training neural networks and deep belief networks: Gaussian Process (GP), Tree-structured Parzen Estimator (TPE), and Random Search [3]. They found that TPE outperforms GP, random search, and manual search for finding good hyperparameters. They also mention that their "methods are quite general, and extend naturally to any hyper-parameter optimization problem in which the hyperparameters are drawn from a measurable set", showing these approaches can be applied diff algorithms. For example, the TPE approach is used by DAT since it is implemented by the HyperOpt and Optuna libraries [4] [1].

More efficient hyperparameter optimization approaches have been proposed, in order to allow optimization even with a computationally expensive cost function. TPE already allows for a lower amount of evaluations to achieve similar results to grid search, with Bergstra et al. showing that the Boston Housing regression task is feasible in 24 hours on a machine with five GPUs [3]. However, for optimizing complex machine learning models, such as Deep Learning models, the number of executions of the cost function needs to be reduced even more. Brzek et al. show that using a genetic algorithm can allow converging towards good values after a low number of iterations [5], and Moya and Ventura propose the GAMF2O evolutionary algorithm as a solution to this problem [13].

2.6 Research gap

The scalability of DAT does not seem to have been extensively evaluated, especially on larger ASTs. Martinez et al. do analyse the overhead of DAT for local optimization, but show that performance is not a significant issue for small files: even when running 100 evaluations with HyperOpt, the maximum execution time is quite low (9.85s), and the median diff time is 0.065s. In comparison, running HyperDiff on a single pair of commits took up to 1min 24s on the Hadoop and Flink repositories in the evaluation by Le Dilavrec et al. [10], which is 1292 times higher. With 2210 possible algorithm configurations, running a grid search on this pair of commits in order to do local hyperparameter optimization would take over 50 hours assuming the cost function always runs in around 1min 30s. Even more optimized approaches, such as TPE, would likely take multiple hours.

Global hyperparameter optimization with grid search or TPE is likely unfeasible in practice for large ASTs. The execution time when running global hyperparameter optimization is not evaluated by Martinez et al. The cost function is significantly more expensive than for local optimization since

the diff has to be run over all pairs of files (or commits) in a dataset. A single evaluation of the cost function over only 100 commits would take around 2.3h on the Hadoop repository using the runtimes from the HyperDiff evaluation. As such, another approach for finding optimized hyperparameter values is needed for large ASTs.

More efficient data-driven approaches, such as using genetic algorithms or GAMF20, could also be evaluated. However, due to time constraints and the complexity of re-implementing these to work with HyperAST, they will not be evaluated within this paper. Furthermore, these approaches are designed for machine learning problems with a large number of hyperparameters, so it is possible that they might not result in large efficiency gains for an algorithm with only 5 hyperparameters.

A possible solution is analysing the effect of each of the hyperparameters individually. Since it is not a fully data-driven approach, it might not result in an optimal configuration and is subject to different interpretations. However, having an insight into the effect of these parameters can help determine good default values for different use cases. To our knowledge, this has not been done before with large ASTs with up to multiple millions of nodes.

Therefore, in this paper we look into the effects of the hyperparameters on both the edit script quality and the runtime of the algorithm, in order to gain insights and suggest default values for different use cases. Furthermore, we study the use and feasibility of local hyperparameter optimization on large ASTs.

3 Contribution

This section describes in detail the contributions of this paper. The source code for these contributions available in a Github repository[1].

3.1 Porting Gumtree Hybrid to work with HyperDiff

The first contribution is a port of the Gumtree Hybrid algorithm to work with HyperDiff, using reference implementation of Gumtree, in Java, as a base[2]. We reused the top-down phase from the HyperDiff, leaving only the bottom-up matching and recovery phase to be implemented. To minimize the risk of differences in the implementation, the program structure of the original version was followed as closely as possible.

The correctness of the new implementation was checked by comparing the edit scripts produced by the Java implementation and the re-implementation when given an identical tree. These tests include small example Java files specifically written to find edge cases and files from the Defects4j dataset.

3.2 Applying HyperDiff optimizations

We implemented a variant of the Hybrid bottom-up matcher using lazy decompression, based on the lazy Greedy bottom-up matcher from HyperDiff [10]. During bottom-up matching, each node must be decompressed using the `decompress_to`. For optimal recovery, the full subtree must be decompressed using `decompress_descendants` method, but for Simple recovery only `decompress_to` is necessary. For the first two steps of the recovery, we can reuse the lazy algorithm from the top-down phase with a minimum size of 0. Histogram matching only looks at the children and does not need a fully decompressed subtree. In this way, the lazy implementation might skip decompressing some subtrees resulting in improved performance.

3.3 Hyperparameter tuning approach

We developed a hyperparameter tuning approach based on DAT. Since the libraries used by Falleri et al. are not available in Rust, the `rust-tpe` library was used instead, which also implements TPE. Grid search was also implemented.

A first version was implemented where `bottom-up_matcher`, `sim_threshold` and `max_size` were included in the optimization. `priority_metric` and `min_priority` were excluded, as values other than 1 showed no improvements. This results in 221 possible algorithm configurations, a significant decrease from the 2210 configurations from DAT.

A second version was implemented optimizing only the `max_size` hyperparameter, due to results showing this has the most potential to result in shorter edit scripts. In our implementation we made the `max_size` values continuous, instead of using steps of 100, since this should not impact the number of evaluations needed but might result in more optimized values.

4 Experimental Setup and Results

This section first presents the research questions, then explains the methodology, the datasets used for the benchmarks, and the results.

4.1 Research questions

We first study the effect of the hyperparameters on both the edit script quality and performance: `bottom-up_matcher`, `max_size` and `min_priority` (resp. RQ1, RQ2, RQ3). `sim_threshold` is excluded as it is not relevant for the Simple or Hybrid variants. We then evaluate if the lazy variant of the Hybrid algorithm improves the performance without changing the output (RQ4). Finally, we see if the DAT approach for local and global hyperparameter optimization is applicable to large ASTs (RQ5).

RQ1 To what extent is the performance and output quality of the Gumtree Hybrid algorithm affected by the size of the AST, compared to Gumtree Greedy?

RQ2 What is the effect of the `max_size` hyperparameter with ASTs of varying sizes?

RQ3 What is the effect of the `min_priority` hyperparameter with ASTs of varying sizes?

RQ4 To what extent do the optimizations included in HyperDiff (mainly lazy decompression) affect the performance when applied to the Gumtree Hybrid algorithm?

[1] https://github.com/alexn5/hyperast

[2] https://github.com/GumTreeDiff/gumtree

RQ5 To what extent do the local and global hyperparameter optimization approaches from DAT also apply to large ASTs?

4.2 Methodology

This paper focuses on two aspects of the Gumtree algorithm: performance and output quality.

We measure performance using execution time of the bottom-up matching phase. There are alternatives, such as the number of CPU cycles or of memory accesses, but runtime is a common metric that reflects scalability well. Its main drawback is that it varies between devices, so all benchmarks were run on the following system: *AMD Ryzen 7 8845HS @ 3.8GHz, 32Gb RAM, 1 Tb SSD*, running *Fedora Linux 42*. All runs used identical power settings, with the laptop plugged in and, to the best of our ability, no other intensive processes running. We also measured the memory footprint using Jemalloc, but excluded it from results since it did not allow for any additional insights and was measured for the entire algorithm, instead of just the bottom-up matcher.

Small runtime benchmarks, on a single pair of files or commits, were done using the Criterion statistics-driver Rust benchmarking library. We used the `iter_custom` method to only measure the runtime for the bottom-up matcher.

For benchmarks over the entire dataset we measured the runtime a single time for each pair of commits in order to limit computation time. We expect the execution times over the dataset to follow a log-normal distribution, instead of a normal distribution, due to some parts of the algorithm having a time complexity of $O(n^2)$ or $O(n^3)$. Therefore, for statistical confidence we always use the Mann-Whitney U-test which does not assume normality.

We use the edit script length as a metric for output quality. We did not have the time to do a study with a manual comparison of the edit script quality as done by Falleri et al. [7] [7]. Their qualitative analysis validates the length of the edit script as a metric for the quality, with the only relevant exception being the rare *aggressive recovery* case where a shorter edit script contains confusing update actions. Alikhanifard and Tsantalis also show that in limited cases shorter edit scripts include actions that are semantically confusing [2], but this problem cannot easily be solved by language-agnostic algorithms.

Appendix A includes instructions to run the benchmarks for reproducibility.

4.3 Datasets

Datasets were selected based on use in other papers and compatibility with HyperAST. HyperAST currently supports the Tree-Sitter parser, Java language and Maven build system. The same dataset was used as for evaluating HyperAST [9], which includes a selection of large Java repositories. We excluded *Hadoop*, *Flink* and *Dubbo* since their size meant the benchmarks could not be run within a reasonable time on the available hardware. We used 50 pairs of commits for the smaller repositories (slf4j, gson, arthas, jackson-core, skywalking, spark, maven), and 30 pairs for the larger ones (aws-toolkit-eclipse, spoon, javaparser, jenkins, logging-log4j2, fastjson, netty, guava, quarkus).

Furthermore, the Defects4j dataset was also used in order to check the output of the implementation, and to get an idea of the behaviour of the algorithm. This is one of the dataset used to evaluate the Gumtree Hybrid algorithm [6].

4.4 Results

RQ1 To what extent is the performance and output quality of the Gumtree Hybrid algorithm affected by the size of the AST, compared to Gumtree Greedy?

The Simple variant tends to be faster than the Greedy variant, also for diffs on large AST, though the difference tends to decrease both with an increasing AST size and a decreasing number of changes. Figure 3 shows relative gains of -77.7% ($p < 0.019$) on some small ASTs and -56.9% ($p < 0.002$) on some larger ones, for 1 to 10 changes. ASTs over 18M nodes even result in increase in the mean runtime, but the p-value is too high to be able to say this is a statistically significant difference. Furthermore, a higher number of changes also leads to a larger relative gain, and this is visible across all AST sizes.

The extra cost incurred in the Hybrid variant by always running Simple recovery is often between 20% and 50%, even on large ASTs. We can always expect Gumtree Hybrid to be slower than Gumtree Greedy for the same `max_size` hyperparameter value since the Hybrid variant will always run the same optimal recovery as the Greedy variant, but also run the Simple recovery. However, as can be seen in Figure 3, the relative increase in runtime is limited to 70.7% ($p < 0.001$) on ASTs with 1.00M to 2.83M nodes and 10 to 1000 changes, while being lower than that for all other facets.

On the other hand, Hybrid often produces shorter edit scripts than Greedy. With a `max_size` of 50, Hybrid results in shorter edit scripts in 67.0% of cases and no difference in 33% of cases, as can be seen in Figure 4. With a `max_size` of 1000, 65.1% of cases result in a decrease, 34.5% in no change, and two cases (0.4%) in an increase. Extra actions decrease with a larger `max_size`, with a median increase of 728.5 for 50 and 373.0 for 1000 (nonzero cases only).

RQ2 What is the effect of the max_size hyperparameter on the Gumtree Hybrid algorithm with ASTs of varying sizes?

A `max_size` of 200 does not result in a significantly higher execution time than Simple (equivalent to a `max_size` of 0), but 500 and 1000 do show increases. Figure 5 shows that for `max_size` of 50 the loss in the mean runtime is always below 10%, with p-values often above 0.5 indicating no statistically significant difference. For a `max_size` of 500 we see losses up to 471.8% ($p < 0.001$), and for 1000 up to 2064.6% ($p < 0.001$). However, some other facets with high losses also show high p-values, likely due to insufficient data points, which limits reliability.

The increase in runtime when using a larger `max_size` is more pronounced for smaller ASTs and ASTs with many changes. With a `max_size` of 1000, Figure 5 shows losses of up to 2155.0% ($p < 0.001$) for ASTs with over 1000 changes, but only in a 320.1% ($p < 0.001$) with 1 to 10 changes. This effect is visible across all AST sizes and values of `max_size`. Furthermore, with larger ASTs the loss

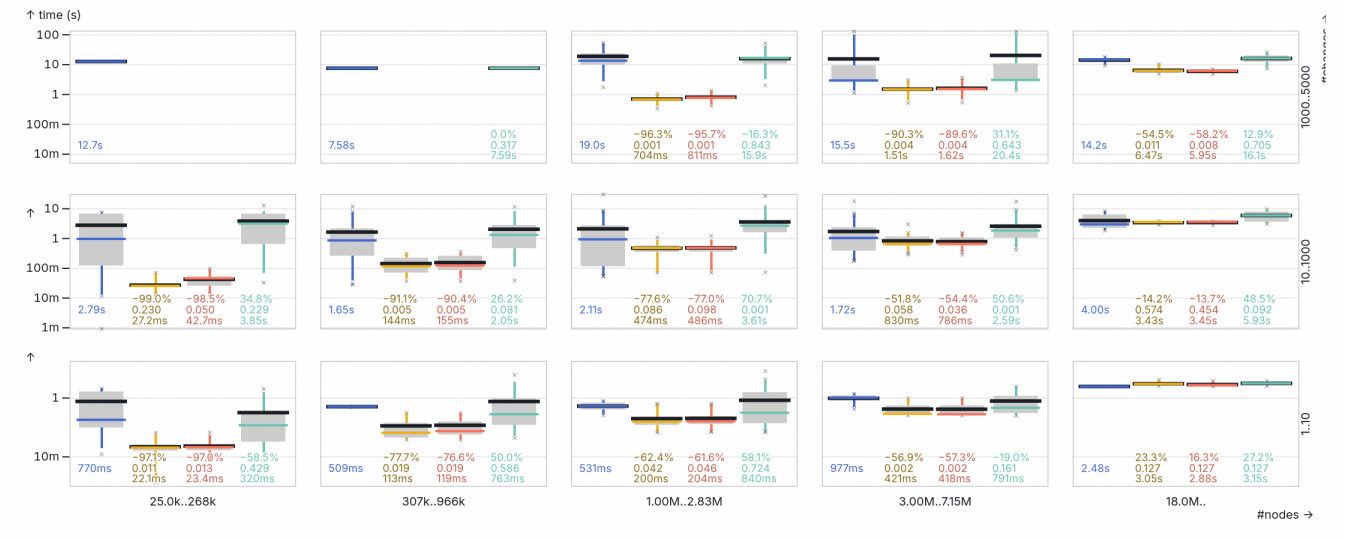

Figure 3: Bottom-up phase execution times for Greedy-1000, Simple, Hybrid-100 and Hybrid-1000
Facet Legend: relative gain in % compared to Greedy-1000, two-tailed p-value (Mann-Whitney), avg. time

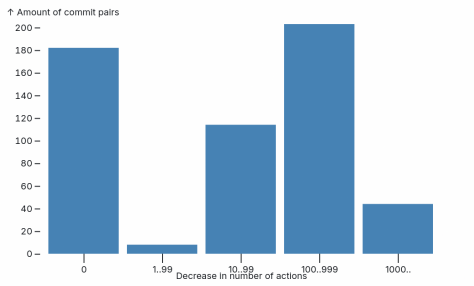

Figure 4: Number of commit pairs by decrease in number of actions between the Greedy and Hybrid variants, with max_size = 50

also tends to be smaller. In fact, for ASTs with more than 18M nodes and 1 to 10 changes, we observe no statistically significant difference in runtime for any value of `max_size`.

At the same time, the effect of `max_size` on the output is unpredictable. Higher vThough there does seem to be a slight tendency towards shorter edit scripts with higher values, as seen in Table 2, a Mann-Whitney U-test on the number of actions between a `max_size` of 0 and 1000 gives a p-value of 0.51, indicating no statistically significant difference. Many commit pairs result in the same edit script length for all values of `max_size` we tested.

This contrasts with Greedy, where the output quality improves as `max_size` increases. We find a mean increase of 224 actions from a `max_size` of 50 to 500 ($p < 0.096$) and 294 actions from 50 to 1000 ($p < 0.008$) on our dataset. We almost always see a decrease or no change in edit script length when increasing `max_size`: between 50 and 1000, we observe 214 decreases, 341 unchanged cases and 2 increases.

No clear pattern emerges between the edit script length related and `max_size`. Figure 6 shows the number of actions obtained when running with different `max_size` values, compared to Simple, on various commit pairs. In many cases, the edit script produced with a `max_size` of 50 is already significantly longer or shorter. Further increases to `max_size` may reduce, increase, or leave the number of actions unchanged, and this variability is similar across all AST sizes.

`max_size`	**50**	**100**	**200**	**500**	**1000**
Decrease	39	56	67	90	91
No change	452	433	420	401	392
Increase	61	62	62	56	63

Table 2: Distribution of changes in edit script length for different values of `max_size`, compared to `max_size` = 0 (Simple).

RQ3 What is the effect of the min_priority hyperparameter on the Gumtree Hybrid algorithm with ASTs of varying sizes?

Increasing the `min_priority` hyperparameter negatively affects output quality without any significant effect on performance. The median number of actions on a subset of the dataset increases from 104 with `priority_metric` = `height` and `min_priority` = 1, to 202 with `min_priority` = 5, while the median runtime stays the same at 0.40s. 60.5% of cases where the edit scripts are not equal show an increase in length with a `min_priority` of 2 and 80.9% with 3.

RQ4 To what extent do the optimizations included in HyperDiff (mainly lazy decompression) affect the performance when applied to the Gumtree Hybrid algorithm?

As mentioned in Section 3.2, the output of the Hybrid variant and Hybrid Lazy variant is always equal. This has been tested by checking the output of both algorithms for all pairs of commits in the repository dataset, with 4 pairs of commits out of 560 resulting in different outputs.

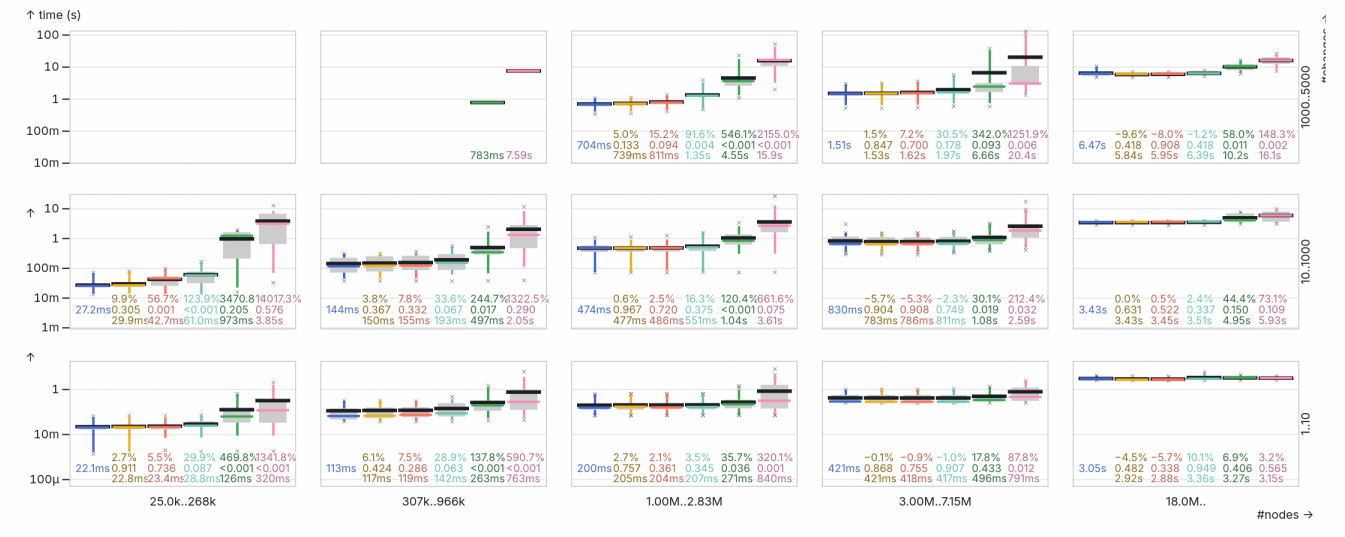

Figure 5: Bottom-up phase execution times for Gumtree Hybrid with max_size 0 (simple), 50, 100, 200, 500, 1000
Facet Legend: relative loss in % compared to Simple, two-tailed p-value (Mann-Whitney), avg. time

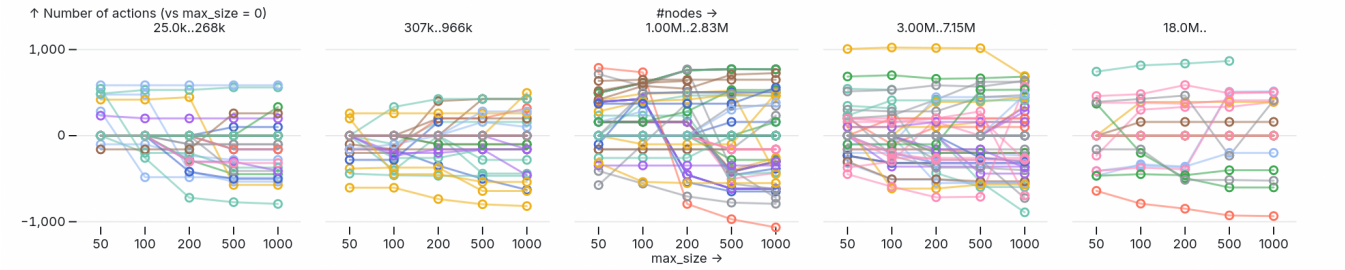

Figure 6: Change in number of actions for different max_size values (symlog scale), compared to Simple. Each line is a commit pair.

However, the lazy implementation almost always results in an increase in execution time. On the lazy implementation, we can consistently observe an increase in the runtime compared to the non-lazy implementation. The biggest increase is with a `max_size` of 50, for ASTs with 25k to 268k nodes and 1 to 10 changes, with the original implementation being 355.1% faster than the lazyfied version ($p < 0.001$). On the other side, for ASTs above 18M nodes with more than 1000 changes and for a `max_size` of 1000, the original implementation is still 69.5% faster than the lazyfied one ($p < 0.008$).

RQ5 To what extent does the DAT approach also apply to large ASTs?

Because of the results from the previous research questions, our latest implementation of hyperparameter optimization only included the `max_size` hyperparameter.

Local hyper optimization is possible but sometimes impractical with small or medium-sized repositories, but it does often result good hyperparameters. Running TPE with 100 evaluations on a single pair of commits from the maven repository took 2.1 hours and resulted in a `max_size` of 619, giving an edit script length of 4224 compared to 4302 when running Gumtree Simple on the same commit. Running TPE with 10 evaluations on the slf4j dataset took 11.6 minutes, finding that a `max_size` of 984 produced an edit script of length 993, compared to 1051 with Gumtree simple.

On the smallest repositories in the dataset, local hyperparameter optimization is feasible. For example, on the gson respository, it found a better configuration than Simple in 2 out of 17 cases with a median runtime of 283.1s and a maximum runtime of 952.0s, with 10 evaluations.

Global hyperparameter optimization over the entire dataset is likely unfeasible even on more powerful hardware. On the hardware we used, a single evaluation of the cost function was not finished after over 30 hours and therefore the optimization was stopped early. As such, the hardware we have does not permit running global optimization. Since the dataset also excludes the largest repositories (Hadoop, Flink and Dubbo), including these would increase the runtime even more, likely making optimization unfeasible even with significantly better hardware.

5 Responsible Research

There are multiple factors that can affect the reproducibility of this paper. We have listed some of them below as well Reproducibility is particularly important in this field of research, however there are three factors that can have a negative impact on it.

The first is the variability of benchmarks on different systems. All types of benchmarks can produce very different results depending on the operating system, architecture, and other parameters. There is little that can be done about this, but the threat to the validity of the results is limited if all benchmarks are run on an identical system. In this paper, all benchmarks were run on the system indicated in section 4, in order to allow all results to be compared.

The second factor is the inherent variability in benchmarks, even on the same system. Benchmarks that measure runtime can be very strongly affected by other processes running on the system, if the system is in a power saving state, and many other factors. In order to mitigate this, all benchmarks were run in as similar of a state as possible. An effort was made to use sample sizes as large as possible, but due to the high computational costs when working with the larger repositories this was not always possible.

The third factor is the accessibility of the results. All of the results in this paper will be supported with instructions to run the benchmarks in Appendix A. Both the code for the implementation of the Hybrid algorithm and benchmarks are available in a Github repository, and the notebooks used to analyze the data are also accessible. All datasets used are openly available and based on software projects with an open-source license, legally permitting their use in this research project.

6 Discussion

This section discusses the results, starting with the effect of the hyperparameters (RQ1, RQ2, RQ3), the optimizations from HyperDiff (RQ4) and the applicability of DAT (RQ5).

6.1 Effect of hyperparameters

Our results suggest that the Simple variant is generally the best algorithm to use on large ASTs. The Greedy variant has a both worse performance and output quality. Alikhanifard and Tsantalis also show the Simple variant produces better edit scripts than Greedy using more detailed metrics than edit script length [2]. The Hybrid variant shows mixed results for output quality and a similar performance to Greedy. If using the Hybrid variant we recommend a `max_size` of at least 1000, since values below this seem to lead to increases in edit script length more often than decreases, though this trend falls just short of statistical significance on our dataset.

Furthermore, the `min_priority` hyperparameter results in the shortest edit scripts at its default value of 1, and any other value does not result in a significant improvement in runtime. Therefore, it is likely not worth including this hyperparameter when doing hyperparameter optimization. This is confirmed by the results from Martinez et al., where the global configurations found by DAT always include `min_priority` = 1 [11]. In this case, `priority_metric` is also not important, since the size and height of a single node is always 1.

This leaves `max_size` as the only hyperparameter that benefits meaningfully from optimization, as well as `bottom-up_matcher` using only the Simple and Hybrid variants. We recommend keeping the default value of 1 for `min_priority` and the `sim_threshold` hyperparameter is only relevant to the Greedy variant.

6.2 Effect of HyperDiff optimizations

Our lazy decompression implementation results in an increase in runtime, which might be caused by an implementation bug. In the evaluation of the lazyfied bottom-up phase of Gumtree Greedy by Le Dilavrec et al., we only see an improvement for large ASTs, where a significant part of the tree is not decompressed due to it being above the `max_height` threshold [10]. Since the Hybrid recovery also should leave some parts of the tree decompressed, we expect to see a similar result. As such, this increase could be caused by an implementation bug. Another possible explanation for the increase is that decompressing the entire subtree all at once is faster than calling the decompression multiple times, but we do not expect to see such a large increase if this is the issue.

6.3 Applicability of DAT on large ASTs

Local hyperparameter optimization is able to find the cases where the Hybrid variant does produce a better result than the Simple variant, however on larger repositories it is often impractical or even infeasible to run due to large computation times.

If quick runtimes are needed, for example in developer tools which need to show a diff to the user in a reasonable time, we recommend using the Gumtree Simple algorithm which has no hyperparameters to optimize. However, if a shorter edit script is more important, we recommend using local hyperparameter optimization on the Gumtree Hybrid algorithm while optimizing only the `max_height` hyperparameter, as long as the ASTs are not too large. This might be the helpful in research into software evolution for example. This mirrors the recommendation from Martinez et al. [11].

With only the `max_size` hyperparameter to optimize, TPE efficiently determines good hyperparameter values. As such, we do not think it is necessary to look at more advanced hyperparameter optimization approaches such genetic algorithms, as they will likely not result in significant improvements.

6.4 Threats to validity

Internal validity. Benchmarks results vary depending on the hardware they are run on, but also for example on power saving settings or another process taking up resources. We have made efforts to limit this variability (see Section 5) and used statistical tests on the results. There might be bugs in our implementations, resulting in incorrect outputs or slowdowns. For the first case, we have written tests to validate the outputs of our implementation, but the second case is more difficult to mitigate. We suspect there is a bug in the lazy implementation since we are unable to explain the decrease in performance.

Finally, we used a relatively small dataset due to the limitations of available hardware.

External validity. In a lot of places, we did not benchmark certain items or values due to the hardware at our availability not being powerful enough to run the benchmarks in a reasonable time. For example, values of `max_size` above 1000 were not included in the evaluation for RQ2, and the three largest repositories were excluded from the dataset. While we did observe patterns in the data without these larger values and extrapolated these in our conclusions, Furthermore, this paper only uses a single language (Java), build system (Maven) and AST meta-model (tree-sitter). Further experimentation is necessary to determine if our findings are generalizable to other situations.

7 Conclusions

The Gumtree Simple algorithm can enable efficient code differencing on large ASTs, which can be shown using HyperAST in order to run diffs on very large ASTs generated from a commit in a repository. It produces shorter edit scripts compared to the Greedy variant, and is generally faster, though the gain in performance compared to the Greedy variant decreases when the size of the ASTs increase.

If quick runtimes are needed, we recommend using the Gumtree Simple algorithm, which has no hyperparameters. If a shorter edit script is more important, we recommend using local hyperparameter optimization on the Gumtree Hybrid algorithm while optimizing only `max_height`, as long as the repository or ASTs are not too large.

A Reproducibility

This appendix describes how to reproduce the results from this paper. All of the following instructions are for our fork of the HyperAST repository at https://github.com/alexn5/HyperAST. Commit `1b48a6dd9e18f8af58b268e87fd67a352bb557e7` includes all of the benchmarks that were included in this paper.

The algorithm implementations are in `crates/hyper_diff/src/matchers/heuristic/gt`. In this paper we mainly compare the `greedy_bottom_up_matcher.rs` (modified), `hybrid_bottom_up_matcher.rs` (new), `lazy_hybrid_bottom_up_matcher.rs` (new).

The tests to check the output of these algorithms are in `crates/hyper_diff/tests`.

The benchmarking scripts are in `benchmark_diffs/benches`, and can be run with:

```
cargo bench -p hyperast_benchmark_diffs
--bench <name_of_benchmark>
```

The benchmarks that measure runtime and memory usage for all variants for the large AST dataset are in `benchmark_diffs/cross_repo_hybrid.rs`. To run them, use:

```
cargo run --release --package
hyperast_benchmark_diffs --bin
cross_repo_hybrid <output_file_path>.csv
<number_of_commits> <name_of_repo>;
```

ObservableHQ was used to generate the plots and analyze the data.

- https://observablehq.com/d/100b329c70c51752
- https://observablehq.com/d/c029bfeb2e9597ba
- https://observablehq.com/d/45a0bd91ce2d73e9

B Use of LLMs

Large Language Models were used used as a productivity tool while writing this paper. This includes ChatGPT and Gemma.

No output of the LLM was included verbatim in the text of the paper, instead the output LLM was used as a starting point for additions or modifications. LLMs were used only for the following cases (each case has a representative sample of prompts):

Creating plots in ObservableHQ. LLMs were used to understand existing plots in ObservableHQ and create or modify new ones.

- How can I facet this plot based on src_s? I would like to specify the upper and lower bound for each facet. [existing code for the plot]
- Can you explain the following code from an ObservableHQ notebook? [existing code from a notebook]
- I have a csv with the following columns: input, kind, max_size, src_s, dst_s, mappings, actions, prepare_topdown_t, topdown_t. Could you generate a scatterplot for ObservableHQ that shows the max_size on the x-axis, and on the y-axis the ratio between the number of mappings for the specified max_size and the number of mappings with max_size 0?

Formatting of the report. LLMs were used to create tables and other elements in this report. For these prompts some output of the LLM was used verbatim in the report, but it was always checked for correctness and did not contain any changes to the content itself.

- Can you format this list nicely in latex? [list of RQs]
- How can I make a table full-width in a two-column latex document
- Could you transform this into a list of the repositories (eg slf4j, gson) that have the value 50, and a list that have the value 30: cargo run –release –package hyperast_benchmark_diffs –bin cross_repo_hybrid ./perfs_new/perfs_slf4j.csv 50 slf4j [rest of commands]
- Could you transform this data into a latex table? increase_table = Object { 50: Object { decrease: 39, equal: 452, increase: 61 }, 100: Object { decrease: 56, equal: 433, increase: 62 }, 200: Object { decrease: 67, equal: 420, increase: 62 }, 500: Object { decrease: 90, equal: 401, increase: 56 }, 1000: Object { decrease: 91, equal: 392, increase: 63 } }

Shortening the text. LLMs were used to determine which parts of the text could be removed or shortened, since the original paper was too long.

- Could you determine if there is anything that can be removed in this section, such as unnecessary or duplicate information: [contents of section]
- How could I make this sentence more concise? [sentence]

Brainstorming. LLMs were used to come up with ways to visualize data and phrase sentences.

- Is this first sentence of a research paper a good hook? [sentence]
- What type of plot could I use to represent this data? I want to show the increase in the number of mappings, depending on src_s and max_size: [data]

References

[1] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.

[2] Pouria Alikhanifard and Nikolaos Tsantalis. A Novel Refactoring and Semantic Aware Abstract Syntax Tree Differencing Tool and a Benchmark for Evaluating the Accuracy of Diff Tools. *ACM Transactions on Software Engineering and Methodology*, 34(2):1–63, February 2025.

[3] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Proceedings of the 25th International Conference on Neural Information Processing Systems*, NIPS'11, page 2546–2554, Red Hook, NY, USA, 2011. Curran Associates Inc.

[4] James Bergstra, Brent Komer, Chris Eliasmith, Dan Yamins, and David D Cox. Hyperopt: a python library for model selection and hyperparameter optimization. *Computational Science Discovery*, 8(1):014008, jul 2015.

[5] Bartłomiej Brzek, Barbara Probierz, and Jan Kozak. Exploration-driven genetic algorithms for hyperparameter optimisation in deep reinforcement learning. *Applied Sciences*, 15(4), 2025.

[6] Jean-Remy Falleri and Matias Martinez. Fine-grained, accurate and scalable source differencing. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, pages 1–12, Lisbon Portugal, April 2024. ACM.

[7] Jean-Rémy Falleri, Floréal Morandat, Xavier Blanc, Matias Martinez, and Martin Monperrus. Fine-grained and accurate source code differencing. In *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering*, pages 313–324, Vasteras Sweden, September 2014. ACM.

[8] Beat Fluri, Michael Wursch, Martin PInzger, and Harald Gall. Change distilling:tree differencing for fine-grained source code change extraction. *IEEE Transactions on Software Engineering*, 33(11):725–743, 2007.

[9] Quentin Le Dilavrec, Djamel Eddine Khelladi, Arnaud Blouin, and Jean-Marc Jézéquel. HyperAST: Enabling Efficient Analysis of Software Histories at Scale. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, pages 1–12, Rochester MI USA, October 2022. ACM.

[10] Quentin Le Dilavrec, Djamel Eddine Khelladi, Arnaud Blouin, and Jean-Marc 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*, pages 288–299, San Francisco CA USA, November 2023. ACM.

[11] Matias Martinez, Jean-Rémy Falleri, and Martin Monperrus. Hyperparameter Optimization for AST Differencing. *IEEE Transactions on Software Engineering*, 49(10):4814–4828, October 2023.

[12] Junnosuke Matsumoto, Yoshiki Higo, and Shinji Kusumoto. Beyond GumTree: A Hybrid Approach to Generate Edit Scripts. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, pages 550–554, Montreal, QC, Canada, May 2019. IEEE.

[13] Antonio R. Moya and Sebastian Ventura. A multi-fidelity genetic algorithm for hyperparameter optimization of deep neural networks. *IEEE Transactions on Evolutionary Computation*, 2025.

[14] Y. Wang, D.J. DeWitt, and J.-Y. Cai. X-diff: an effective change detection algorithm for xml documents. In *Proceedings 19th International Conference on Data Engineering (Cat. No.03CH37405)*, pages 519–530, 2003.