論　文

より短い編集スクリプトを目指して
— 行単位の差分情報に基づく GumTree の拡張 —

松本淳之介[†a)]　肥後　芳樹[†]　楠本　真二[†]

Towards Generating Shorter Edit Scripts
— An Extension of GumTree Based on Line Differences —

Junnosuke MATSUMOTO[†a)], Yoshiki HIGO[†], and Shinji KUSUMOTO[†]

あらまし　ソフトウェア開発においてコードレビューは頻繁に行われ，レビューをする開発者はソースコードの差分を理解する必要がある．編集前のソースコードを編集後のソースコードに変換する操作の列を編集スクリプトと呼び，開発者はその編集スクリプトを理解することでソースコードの差分を理解する．編集スクリプトを出力するツールとして GumTree がある．GumTree は二つのバージョンのソースコードを入力として受けとり，抽象構文木のノード単位の挿入・削除・更新・移動といった 4 種類の操作で構成された編集スクリプトを出力する．しかし，GumTree が更新と移動を検知する精度は低く，差分の理解が困難であるという問題がある．GumTree の精度が低い原因の一つとして，抽象構文木の情報のみで編集スクリプトを出力しようとすることが挙げられる．そこで提案手法では抽象構文木の情報のみでなく，行単位の差分情報を用いて編集スクリプトを出力する．そしてオープンソースソフトウェアに対して提案手法が有効か実験を行い，提案手法が差分の理解に有効であることを確認した．

キーワード　差分理解，編集スクリプト，GumTree

1. まえがき

ソフトウェア開発においてコードレビューは頻繁に行われており，開発者はソースコードの差分を理解する必要がある．そのような場面において，開発者を支援するためにソースコードの差分を分かりやすく表現する研究が数多く行われている．

編集前のソースコードを編集後のソースコードに変換する操作の列を編集スクリプトと呼び，開発者はその編集スクリプトを理解することでソースコードの差分を理解する．編集スクリプトを出力する方法として，コマンドラインツールである Unix の `diff` を用いることが多い．`diff` は Myers のアルゴリズム [1] を基に，行単位でソースコードが挿入・削除されたという編集スクリプトを出力する．また，`diff` から派生したツールも開発されており，例えば `wdiff` は行単位ではなくトークン単位の差分を出力する

しかし，`diff` や `wdiff` をソースコードの差分表示に利用する場合には，二つの問題があると著者らは考えている．一つ目の問題は差分を出力する粒度が不適切ということである．開発者が理解したい対象の多くはソースコードである．`diff` ではソースコードの行の一部しか編集されていないにもかかわらず，その行全てが編集されたと出力され，編集されていないプログラム要素まで編集されたと出力される．また，例えば波括弧 (`{}`) では，始め括弧 (`{`)・終わり括弧 (`}`) といった対応関係があり，二つの括弧で一つのプログラム要素を表している．しかし，`diff` は片方だけが編集された (若しくはされていない)，という編集スクリプトを出力することが多々ある．これらは `diff` がソースコードを単なるテキストファイルとして扱っていることが原因である．このことから，構造化されているソースコードの差分を出力するのに `diff` は適していないことがわかる．

二つ目の問題は操作の種類が少ないということである．`diff` の出力には挿入と削除の 2 種類の操作しか

† 大阪大学，吹田市
Osaka University, Suita-shi, 565–0871 Japan
a) E-mail: j-matumt@ist.osaka-u.ac.jp
DOI:10.14923/transinfj.2019JDP7100

ない．オプションを指定することで，更新という操作を表現することは可能であるが，あくまで削除と挿入の組み合わせでしかない．また，トークンの位置が近いという理由で更新という操作を出力しているが，それがプログラム要素として更新されているかは考慮していない．このようにソースコードの編集において，開発者が挿入や削除以外の操作をする場合，開発者の意図した編集を表現することができない．

このような問題を解決する手法として GumTree [2] がある．GumTree は抽象構文木のノード単位でソースコードの差分を出力する．更にノードの挿入や削除といった操作に加えて，ノードの更新や移動といった計 4 種類の操作を出力することができる．GumTree 及び GumTree を改良した手法が出力した編集スクリプトは多くの研究で利用されている．例えば API のコードのサジェスト [3] や Maven のビルドファイルの解析 [4]，自動プログラム修正 [5], [6] であったり，JavaScript のバグのパターンの発見 [7] などに用いられている．ただし，GumTree には移動と更新の検知精度が低いという課題がある [8]．実際に著者らが目視でオープンソースソフトウェア (以下 OSS と呼ぶ) の差分を確認したところ，更新であるべき変更が削除して挿入と出力されたり，プログラム要素が移動していないのに移動されたと出力されており，編集スクリプトが必要以上に長くなった．編集スクリプトが長ければ長いほど，開発者はその編集スクリプトを理解することが難しくなると言われている [8]〜[10]．

そこで本研究では，より短く理解のしやすい編集スクリプトの生成を目標とし，GumTree の改良を行う．GumTree の課題である更新や移動の精度をあげるために，編集前のノードと編集後のノードのマッチング処理をより適切に行う必要がある．マッチング処理とは編集の前後でどのノードが同じノードか対応づけることであり，その結果から編集スクリプトを計算する．例えば，編集前のソースコードのノードが対応づけられなかった場合，GumTree はそのノードが編集前のソースコードには存在しているが編集後のソースコードには存在していないと認識し，このノードに対して削除という編集操作を計算する．どのノードとどのノードが対応づけられるかは抽象構文木の構造の情報を基に計算される．

編集スクリプトが長くなる原因として，抽象構文木の構造の情報のみを用いたマッチング処理にあると著者らは考えた．そこで本研究では抽象構文木の情報に加えて，行単位の差分の情報を用いたマッチング処理を提案する(注1)．`diff` によって変更されたと出力された行，変更されていないと出力された行の二つにソースコードを分け，それぞれに対して異なるマッチング手法を用いてノードのマッチング処理を行う．

GumTree よりも短い編集スクリプトを提案手法が出力していることを確認するため，7 個の OSS 差分に対して提案手法を実行し，提案手法が有効であることを確認した．また，提案手法が理解しやすい編集スクリプトを出力していることを確認するため，14 人の被験者に対して実験を行い，GumTree より理解しやすい編集スクリプトを出力していることを確認した．

2. 抽象構文木の差分検知

2.1 抽象構文木

抽象構文木 (以降 AST と呼ぶ) とはソースコードを表現した木構造である．一つのソースファイルに対して一つの AST が構築され，AST の各ノードは以下の五つで構成されている．

ID　各 AST 内で固有の識別子

親ノードへの参照　各ノードは木構造上の親ノードへの参照をもつ．ただし根ノードには親が存在しないので，何も保持しない．

子ノードへの参照　各ノードは木構造上の子ノードへの参照をもつ．ただし葉ノードには子が存在しないので，何も保持しない．

ラベル　if 文や変数宣言など文法上の型を表す．

値　メソッド名や変数名など各ノードがもつラベル以外の情報である．もつ情報がない場合は null になる．

2.2 GumTree の差分検知

AST の差分を検知する手法として，GumTree [2] がある．GumTree は入力として編集前のソースコードと編集後のソースコードを受けとる．それぞれのソースコードから AST を構築し，それらの木構造の違いを編集スクリプトとして出力する．

GumTree の出力する編集スクリプトは以下の種類の操作から構成される．

挿入　$insert(t, t_p, i, l, v)$

ノードの追加を表す．挿入するノードの ID として t，ラベルとして l，値として v，挿入後に親ノードとするノードの ID として t_p，何番目の子にするかを表す数値として i を引数にもつ．

(注1)：本論文は文献 [11] に対して詳細な評価を行った論文である．

削除　$delete(t)$
ノードの削除を表す．削除するノードの ID として t を引数にもつ．

更新　$update(t, v)$
ノードの値の更新を表す．更新の対象となるノードの ID として t，新しい値として v を引数にもつ．

移動　$move(t, t_p, i)$
部分木の移動を表す．移動する部分木の根ノードの ID として t，移動先の親ノードの ID として t_p，何番目の子にするかを表す数値として i を引数にもつ．

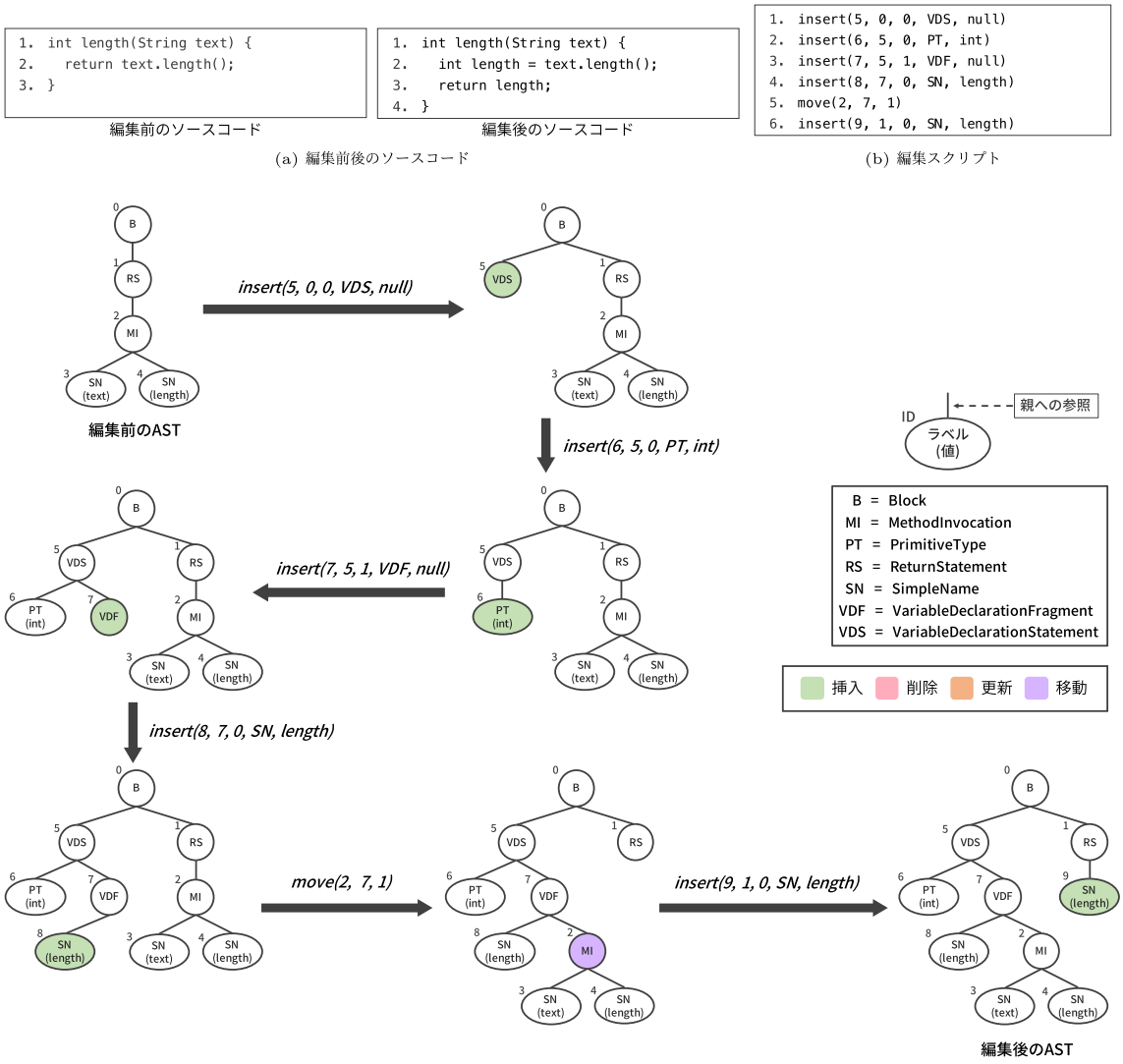
一つの編集スクリプトに含まれる操作の数を本論文では編集スクリプトの長さと呼ぶ．図 1 に編集スクリプトの例と AST を操作する例を示す．この例の場合，編集スクリプトの長さは 6 である．

GumTree の処理は以下の二つで構成されている．

（1） ノードのマッチング処理

（2） マッチング処理の結果を基にした編集スクリプトの計算処理

GumTree が行うノードのマッチング処理はトップダウンフェーズとボトムアップフェーズの 2 段階で構



(a) 編集前後のソースコード　(b) 編集スクリプト

(c) 編集スクリプトによって AST が変換される例

図 1　編集スクリプトの例と，その編集スクリプトに基づいて AST を操作する例
Fig. 1　An example of an edit script, and an exmple that it converts AST.

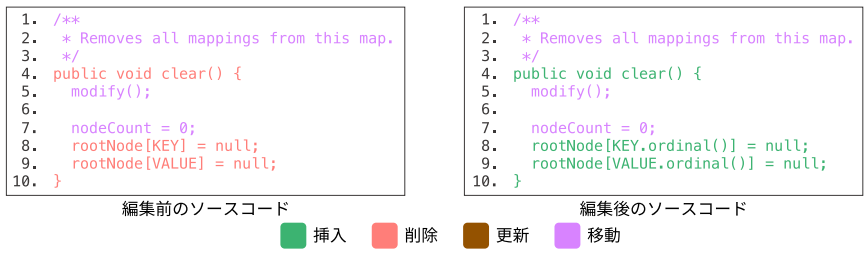


図 2　GumTree の実行例
Fig. 2　The results of GumTree.

成されている．まずトップダウンフェーズでは，入力された二つの AST の根から辿っていき，二つの AST 間で完全一致する部分木を対応づける．ボトムアップフェーズでは，トップダウンフェーズで対応づけられた部分木の根ノードから，編集前の親を辿り，その編集前の親ノードと対応づける候補のノードを編集後の AST から探す．ボトムアップフェーズでは候補のノードを根とした部分木の類似度を計算し，それがしきい値を超えていれば対応づける．

その後，マッチング処理の結果を基に GumTree は編集スクリプトを計算する．マッチング処理の結果から，各ノードは以下の三つに分類される．

- 編集前の AST にのみ存在するノード
- 編集後の AST にのみ存在するノード
- 編集前後の AST に存在するノード

編集前にのみ存在するノードは，言い換えれば編集後には消えているので編集によって削除されたことがわかる．同様に編集後にのみ存在するノードは編集によって挿入されたことがわかる．編集前後の AST に存在するノードの値が変わっていればそのノードが更新されたことがわかる．また，編集前後の AST に存在し，挿入や削除を考慮した上で親ノードに対する並び順が編集前後で変わっていたり，編集前後で親ノードが変わっていれば，そのノードの位置が移動したことがわかる．同じ親ノード，親ノードに対して同じ位置，かつ同じ値であればそのノードは編集されていないことがわかる．

このように，マッチング処理の結果から GumTree は編集スクリプトを計算することができる．二つの木とマッチング処理の結果からその木構造の違いを計算する際の計算に GumTree は Chawathe らのアルゴリズム [12] を採用している．このアルゴリズムの計算量は最適化されており，編集スクリプトを計算する上での改善すべき瑕疵が知られていない．

3. 研究動機

GumTree はしばしば不適切な対応づけを行い，その結果必要以上に長い編集スクリプトを出力する問題がある．図 2 にその例を示す．これは Commons-Collections(注2) 内のある二つのコミット(注3) 間の TreeBidiMap.java に対して GumTree を実行し，出力された編集スクリプトの一部である．左右のソースコードを見比べると，8 行目と 9 行目しか編集されていないにもかかわらず，GumTree は全ての行が挿入，削除，若しくは移動されたと出力している．例えば 4 行目は削除と挿入になっている．これは編集前後の `public` や `void` などのノードが対応づけられていないことが原因である．もし編集前後で `public` を対応づけることができれば，GumTree は編集前に存在した `public` が編集後にも存在していると認識することができ，`public` が削除・挿入されたとは出力しない．

この編集前後のノードのマッチング処理の結果を図 3 に示す．GumTree はトップダウンフェーズで完全一致する高さが 2 以上の部分木を対応づけ，その結果図 3 のようになる．GumTree はトップダウンフェーズの後にボトムアップフェーズのマッチング処理を行うが，ボトムアップフェーズのマッチング処理はそのノードの部分木の類似度が高くなければ対応づけない．そのため，この例のようにトップダウンフェーズで対応づけられたノードの数が少ない場合，そのノードは対応づけられない．例えば MethodDeclaration のノードの場合，そのノードの部分木の類似度は低く，ボトムアップフェーズで MethodDeclaration のノードは対応づけられない．

このように GumTree は AST の構造の情報のみを利用してマッチング処理を行い，結果として必要以上

(注2)：https://github.com/apache/commons-collections
(注3)：コミットID: d16bc8509fc423540a131184552649de1bbcaf98

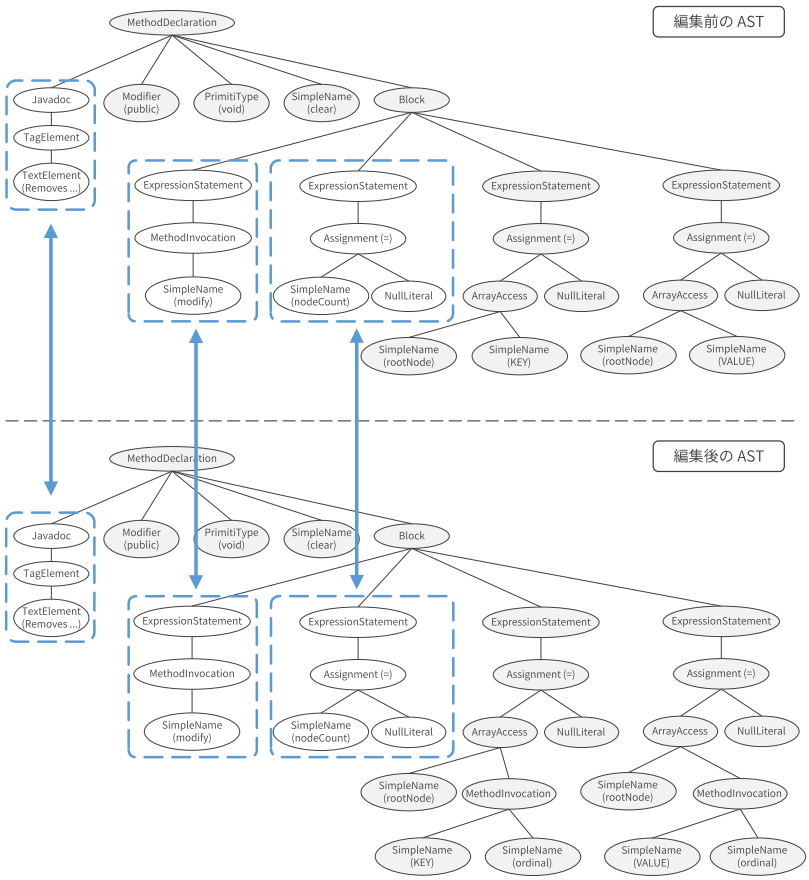


図 3　マッチング処理の結果．白色のノードは対応づけられたノードを，灰色のノードは対応づけられなかったノードを表している．図を簡潔にするため，この図では部分木単位での対応づけのみ示しているが，実際は各ノードが 1 対 1 で対応づけれられている．

Fig. 3　The example which presents matching of AST nodes. White nodes are matched node, and gray nodes are non-matched nodes. In this figure, we showed the matching results per subtree in order to make the figure simple.

に長い編集スクリプトを出力する．

4. 提 案 手 法

4.1 概　　要

提案手法は GumTree を基にしている．GumTree では編集スクリプトの計算は十分に最適化されているので，提案手法ではノードのマッチング処理を改善する．提案手法の概要を図 4 に示す．マッチング処理の精度を上げるために，提案手法では AST に関する情報だけでなく，`diff` を用いた行単位の差分情報を用いる．GumTree によるマッチング処理を行う前に `diff` から得られた情報を基にマッチング処理を行う．

`diff` で出力される行単位の差分から，どの行が変更されてどの行が変更されていないかがわかる．それらの情報を基に提案手法ではソースコードを，変更がない行と変更がある行の二つに分け，それぞれに対して異なるアプローチで探索範囲を狭めたマッチング処理を行う．探索範囲を狭めたマッチング処理の後，対応づけられていないノードに対して GumTree のマッチング処理を行い，それらの結果から編集スクリプトの計算をする．図 5 は図 2 のソースコードに対して提案手法を実行した結果である．この編集スクリプトは図 2 の編集スクリプトより簡潔に差分を示せている．

4.2 変更がない行

`diff` で変更がないと出力された行は，開発者は実際に編集していないと提案手法では扱う．変更がないと

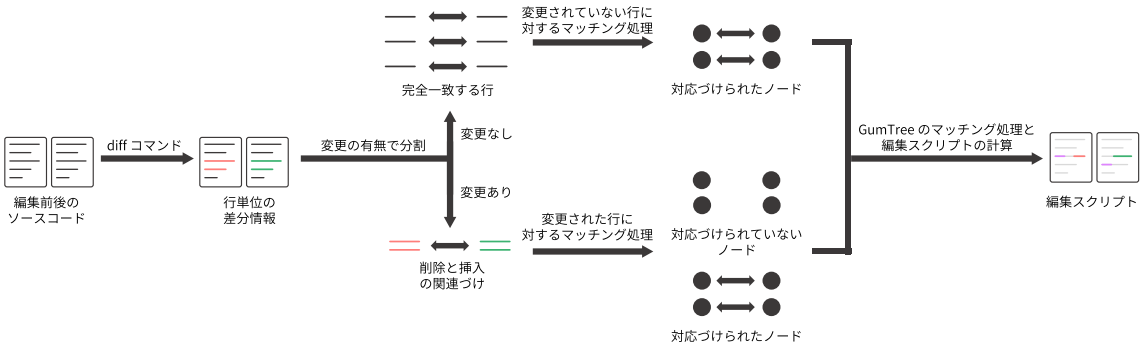


図 4　提案手法の概要
Fig. 4　The overview of our methodology.

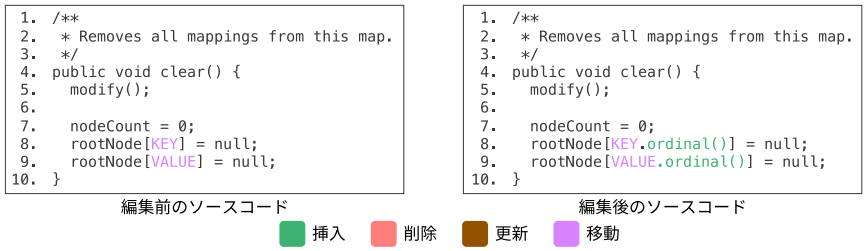


図 5　提案手法の実行例
Fig. 5　An edit script generated by our methodology.

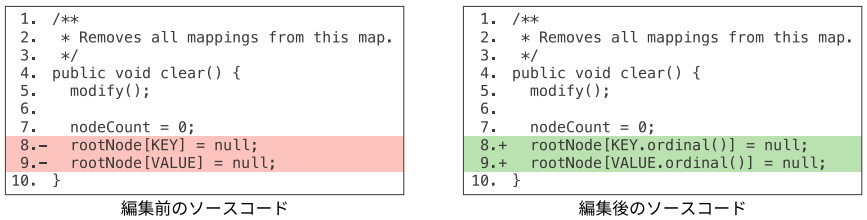


図 6　図 2 のソースコードに対して `diff` を実行した結果
Fig. 6　The results that we executed diff command for the source code difference in Figure 2.

出力された行は，編集前後で完全一致する行が存在し，その行の中の全てのノードが対応づけられる．ノードは必ずしも 1 行に収まっているとは限らない．複数行にまたがるノードは，ノードの先頭の行とノードの最後の行が変更のない行のときに対応づける．

図 2 で示したソースコードに対して `diff` で行単位の差分を計算した結果を図 6 に示す．例えば，編集前の 4 行目と編集後の 4 行目は変更のない行であり，完全一致する行である．そこで編集前後の 4 行目に含まれる各ノードを対応づける．このようなマッチング処理を行うことで **3.** で説明したような AST の構造のみでは対応づけられなかった `public` などを対応づけることが可能となる．

4.3　変更がある行

変更がある行については，`diff` で出力された削除と挿入を関連づけし，その関連づけられた中でノードの対応づけを行う．変更がある行の前後には，変更がない行が存在し，それらが編集の前後で共通しているとき，その削除と挿入を関連づける．連続した複数の行がまとめて削除・挿入されている場合はそれらの行を一つの削除と挿入としてまとめて関連づける．

図 7 を例に考える．図 7 の編集前のソースコードの 2 行目と 3 行目には削除が出力されており，その前後の 1 行目と 4 行目は変更がない行である．変更がない行は編集の前後で対応関係があるので，編集前の 1 行目に対応するのは編集後の 1 行目である．同様に編

表 1　実験 1 の結果
Table 1　The results of the experiment 1.

OSS	コミット数	対象の差分の数	短い編集スクリプトを出力した割合 GumTree	提案手法
dnsjava	1,749	5,539	3.97%	17.73%
Eclipse JDT Core	24,054	364,872	3.66%	22.06%
JHotDraw	2,186	29,463	2.98%	8.39%
JUnit4	2,397	14,864	3.20%	20.04%
Log4j	3,275	27,460	3.23%	41.11%
Struts	5666	52,444	1.05%	7.58%
Tomcat	21,097	52,876	3.32%	18.10%

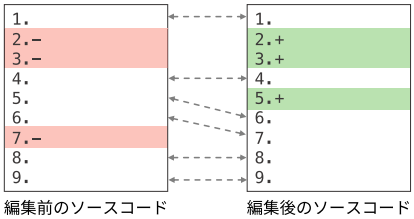


図 7　行単位の差分の例．破線は変更のない行の対応関係を表している．
Fig. 7　The example of diff command's results. Dashed lines present matching lines.

集前の 4 行目に対応するのは編集後の 4 行目である．これらの情報から，編集前の 1 行目と 4 行目に挟まれた削除と，編集後の 1 行目と 4 行目に挟まれた挿入を関連づける．一方，7 行目の削除の前後は 6 行目と 8 行目であり，それらに対応するのは編集後の 7 行目と 8 行目である．しかし，7 行目と 8 行目の間に挿入された行が存在しないため，7 行目の削除はどの挿入とも関連づけられない．5 行目の挿入についても同様である．

削除と挿入を関連づけられたら，その中のノードに対してマッチング処理を行う．まず同じ値，同じラベルをもつ葉ノードを探し，対応づける．その後対応づけられた葉ノードを起点に GumTree のボトムアップアルゴリズムを適用し，他のノードを対応づける．

5. 実　　験

提案手法を評価するため，二つの評価実験を行った．

実験 1　7 個の OSS を対象に，編集スクリプトが短くなるか確認する実験

実験 2　開発者にとって理解しやすい編集スクリプトが出力されることを確認するための被験者実験

5.1　実験 1 について

OSS を対象に，提案手法が GumTree よりも短い編集スクリプトを出力できるか実験を行った．既存研究 [2] で用いられているデータセット [13] を実験に用いた．これは CVS で管理されている 14 個の Java の OSS を集めたデータセットである．実験ではその 14 個の OSS のうち，Git に移行された 7 個の OSS を対象とした．実験対象の OSS を表 1 に示す．これらの OSS に対して，master ブランチ (master ブランチが存在しない場合はメインで開発されているブランチ) に含まれる各コミットの Java ファイルの差分を実験対象とした．実験対象の差分ごとに GumTree と提案手法を実行し，出力される編集スクリプトの長さを比較する．

実験の結果を表 1 に示す．7% ～ 40%の差分に対して，提案手法の方が短い編集スクリプトを出力した．一方，1% ～ 4%の差分に対しては，提案手法の方が GumTree より長い編集スクリプトを出力した．

また，どのような差分に対して提案手法が有効か確認するため，GumTree が出力した編集スクリプトの長さで分類した結果を表 2 に示す．各範囲の編集スクリプトの長さは正規分布に従っていないため，Wilcoxon の符号順位検定を用いて統計的に有意であるか検定を行った．その結果全ての範囲において，p 値が 0.05 を下回り，GumTree と提案手法の間には統計的に有意な差があることが確認できた．また，Cliff のデルタ [14] を用いて効果量を求めたところ，多くの範囲で無視できない結果となっており，5,000 ～ 10,000 の範囲で提案手法が特に有効であることがわかる．

5.2　実験 2 について

提案手法で短くなった編集スクリプトが開発者にとって理解しやすいか確認する被験者実験を行った．

被験者は Java を用いた開発の経験が 1 年以上ある 4 人の大学生・9 人の大学院生・1 人の大学教員の合計 14 人である．

実験には Apache Commons Math を用いた．Apache Commons Math を用いたのは，扱うドメインが数値計算であり，ネットワークや DB を扱う

表 2　実験 1 の結果を GumTree が出力した編集スクリプトの長さに基づいて分類した結果
Table 2　The results that the edit scripts in experiment 1 were classified based on the size of GumTree's edit script.

GumTree が出力した編集スクリプトの長さの範囲	対象の差分の数	実行時間の中央値 (秒)		長さの中央値		長さについての効果量
		GumTree	提案手法	GumTree	提案手法	
0 ～ 50	340,039	0.04	0.04	8	8	negligible
50 ～ 100	51,348	0.10	0.11	71	68	negligible
100 ～ 150	26,594	0.13	0.15	121	117	small
150 ～ 200	14,683	0.13	0.16	173	167	small
200 ～ 250	10,531	0.15	0.19	223	214	small
250 ～ 300	6,484	0.14	0.2	277	266	small
300 ～ 350	6,073	0.18	0.25	327	313	medium
350 ～ 400	4,866	0.19	0.28	375	358	medium
400 ～ 450	3,916	0.17	0.26	423	409	medium
450 ～ 500	2,967	0.18	0.24	467	456	medium
500 ～ 550	2,685	0.29	0.40	530	514	medium
550 ～ 600	2,326	0.33	0.52	577	554	medium
600 ～ 650	1,841	0.20	0.32	627	606	large
650 ～ 700	1,060	0.29	0.40	673	655	medium
700 ～ 750	1,241	0.52	0.79	722	703	large
750 ～ 800	828	0.37	0.56	774	756.5	medium
800 ～ 850	695	1.02	1.74	824	802	large
850 ～ 900	586	0.40	0.87	874	845	large
900 ～ 950	662	0.78	1.41	928	916	medium
950 ～ 1,000	1,199	0.47	0.84	982	953	large
1,000 ～ 2,000	8,109	0.79	1.44	1,334	1,289	negligible
2,000 ～ 3,000	1,440	0.95	2.48	2,451	2,332	small
3,000 ～ 4,000	741	2.52	3.82	3,315	3,254	small
4,000 ～ 5,000	261	2.82	25.81	4,537	4,209	medium
5,000 ～ 6,000	404	0.65	1.08	5,198	5,170	large
6,000 ～ 7,000	290	2.65	27.64	6,237	6,026	large
7,000 ～ 8,000	228	4.11	9.77	7,404	7,079	large
8,000 ～ 9,000	97	2.94	8.56	8,093	8,013	large
9,000 ～ 10,000	50	0.64	10.11	9,767	9,233	large
10,000 ～	38	1.59	13.82	11,207	10,120.5	medium

OSS などと異なり，差分を理解する際にプログラミング以外の知識を必要としないからである．

Apache Commons Math の差分の中から，GumTree が出力する編集スクリプトの長さが 50 ～ 200 で，どちらが短いかに関係なく GumTree の編集スクリプトと提案手法の編集スクリプトの差が大きい上位 15 個を対象にした．編集スクリプトの長さを 50 ～ 200 に限定したのは被験者にとって過負荷とならないように調整するためである．Apache Commons Math の差分の数は 24,989 個であり，その中で GumTree が出力する編集スクリプトの長さが 50 ～ 200 であった差分の数は 2,941 個であった．また，その 2,941 個の差分に対して，二つの手法が出力した編集スクリプトの差の分布を図 8 に示す．

また，二つの手法間での編集スクリプトの長さの差が大きい 15 個を用いたのは，明確に異なる編集スクリプトを用いることで実験結果に差を出しやすくすることが目的である．被験者にとって過負荷になることを避けるため，GumTree が出力する編集スクリプトの長さが 50 ～ 200 の差分を用いた結果，図 8 から分かるとおり二つの手法間での長さの差が −5 ～ 0 の範囲に多くの差分が集中した．これは長さが 50 ～ 200 の編集スクリプトは既に最短な編集スクリプトである場合が多く，二つの手法が出力する編集スクリプトに差がないからである．二つの手法が同様の長さの編集スクリプトを出力するこのような差分を用いても，提案手法が GumTree よりも理解しやすい編集スクリプトを出力できているか確認することができないため，今回の実験では理解に差が出やすい差分を用いた．

提案手法が出力する編集スクリプト，及び GumTree が出力する編集スクリプトの確認には，図 2 や図 5 のように Web ブラウザで視覚的に編集スクリプトを確認することができる GumTree の機能を用いた．この機能は編集スクリプト内の編集操作の順序によって表示が変わることはない．

まず被験者は，実験対象の 15 個の差分に取り組む

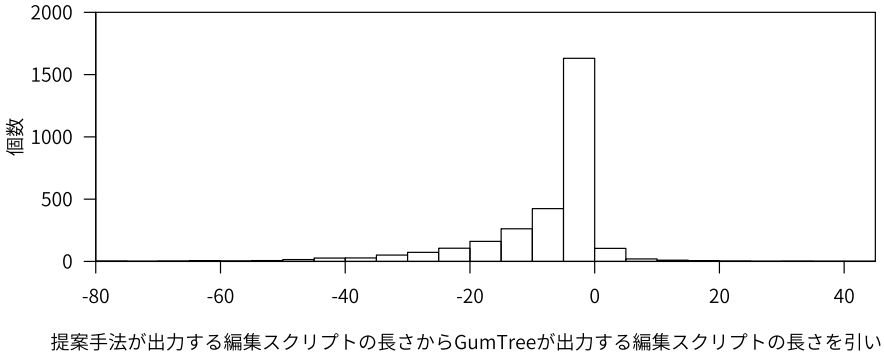


図 8　各手法が出力した編集スクリプトの長さに対する差の分布
Fig. 8　The distribution of differences between the lengths of an edit script generated by each methodology.

前に別の三つの差分を確認し，ツールの使い方を学ぶ．その後，実験対象の 15 個の差分を一つずつ確認し，どのような差分だったかを答え，差分を確認するのにかかった時間を測る．

被験者を X と Y の二つのグループに分ける．このとき，グループ内の大学生，大学院生，教員の人数が均等になるように割り振った．X のグループは，奇数番目の差分に対して GumTree が出力する編集スクリプトを確認し，偶数番目の差分に対して提案手法が出力する編集スクリプトを確認する．Y のグループは，奇数番目の差分に対して提案手法が出力する編集スクリプトを確認し，偶数番目の差分に対して GumTree が出力する編集スクリプトを確認する．なお，被験者にはどちらの手法の差分を確認しているか知らせずに実験を行った．

15 個の差分の順番はランダムに決めた．奇数番目の差分と偶数番目の差分に対して，提案手法と GumTree の編集スクリプトの差の平均を求めたところ，長さは 6 しか変わらず，奇数番目の差分と偶数番目の差分に対して，編集スクリプトの長さがどちらかに偏ってはいない．

被験者は時間を計りながり差分を確認する．被験者はその差分でどのような変更が行われてたのかを理解し，その理解に費やした時間とその変更の内容をテキストファイルに記述する．

著者らがそのテキストファイルを読み，各差分の変更内容に対して明らかに間違った内容を記述している，若しくは各差分の変更内容を表現するのに明らかに説明が足りていない場合，その差分を十分に理解できていないと扱い，取り除いた．制限時間を設けていないため，被験者は理解するまで差分を確認することができるので，理解が不十分な差分は 210 個（差分 15 種類 × 被験者 14 人）のうち 3 個のみであり，それらは全て GumTree が出力した差分であった．

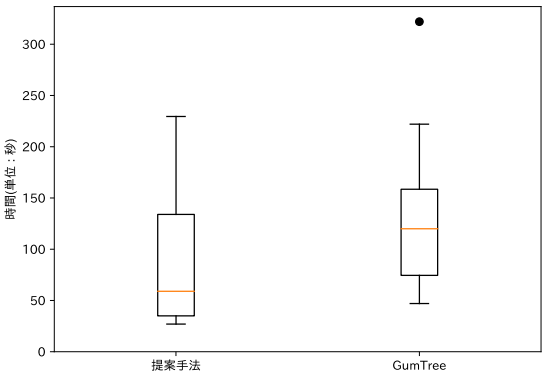


図 9　各差分を理解するのに費やした時間の中央値の箱ひげ図
Fig. 9　The results of experiment 2: the medians of time for each difference.

図 9 は各差分の理解に費やした時間の中央値を箱ひげ図で表している．全体の中央値に関して，提案手法が GumTree の半分の時間になっている．

得られた時間の分布は正規分布に従っていないので，Mann Whitney の U 検定を用いて，これらが統計的に有意であるか確認した．サンプル数が少ないため有意な差が得られない差分もあったが，全体に関する有意な差は得られ，提案手法の方が短い時間で理解することができたといえる．

6. 考　　察

実験 1 では提案手法によって多くの差分で短い編集スクリプトを出力することに成功しており，そのこと

からも提案手法が有効であることは分かる．表 2 の結果から，GumTree の出力する編集スクリプトの長さが 0 ～ 300 である差分に対しては提案手法も同等の長さの編集スクリプトを出力し，GumTree の出力する編集スクリプトの長さが 1,000 を超える差分に対して，提案手法は中央値が 50 以上も短い編集スクリプトを出力している．これらのことから，今回の実験では GumTree が出力する編集スクリプトの長さが 1,000 を超える差分に対して提案手法は特に有効的であり，GumTree の出力する編集スクリプトが長いほど提案手法が有効であると考えられる．実際，GumTree の出力した編集スクリプトの長さが 31,484 と長い差分も実験対象の中には存在したが，提案手法ではその差分の長さを 10,195 にまで減らすことができ，およそ 20,000 以上も短い編集スクリプトを出力できていた．また効果量が 4,000 ～ 10,000 の範囲で large になっていることからも，GumTree が長い編集スクリプトを出力した差分に対して提案手法が特に有効であることがわかる．以上のことから，提案手法は全ての範囲において GumTree より短い，若しくは同等の編集スクリプトを出力できていることがわかる．

短い編集スクリプトを出力できた最大の要因は完全一致する行に含まれるノードに対して，適切に対応づけることができたからだと考えられる．再度，図 2 と図 5 について考えてみる．図 2 では 4 行目のメソッド宣言について適切に対応づけることができておらず，`clear()` を削除して挿入，と出力されている．更に，メソッド宣言のノードが対応づけられていないことから，5 行目の `modify();` の親ノードが変わったと扱われ，5 行目が移動していないのに移動したと出力される．一方，図 5 は図 2 に比べて理解のしやすい編集スクリプトになったといえる．例えば，4 行目には行単位の差分が発生しておらず，提案手法ではメソッド宣言のノードを対応づけることができる．その結果，5 行目は編集の前後で同一の親をもつと扱うことができ，移動していないと出力することができる．このように，一つのノードを対応づけることで，他のノードの扱いも変わり，結果として短い編集スクリプトを出力できたと考えられる．

一方，1% ～ 4%の差分に対して提案手法は GumTree よりも長い編集スクリプトを出力した．これは，メソッド抽出リファクタリングなど，編集の前後でメソッドに抽出されたノードの位置が大きく変わる変更を行った場合に，削除と挿入の関連付けを適切に行うことができないためだと考えられる．

また，表 2 の結果からわかるように多くの範囲で実行時間が長くなった．多くの範囲では高々1 秒の増加ではあるものの，一部の範囲では 10 秒以上も長くなった．実行時間が 10 秒も以上長くなった範囲は，GumTree の出力する編集スクリプトの長さが 1,000 以上の範囲であることから，提案手法は変更の量が多いほど長い実行時間を必要とすると考えられる．これは特に変更された行数が多いときに顕著であり，関連づけられた削除と挿入のそれぞれの行数が多いと関連づけるノードの候補の数が増えるため，関連づけられた削除と挿入の中からノードの対応づけを行う処理が実行速度を低下させていると考えられる．

実験 2 では提案手法の方が，全体を通して短い時間で差分を理解できていた．理解に要した時間が短く，統計的に有意な差がみられた差分は，`@Override` の追加といった比較的単純な差分が多かった．一方で統計的な有意な差が出なかった差分は，クラス名の変更やメソッドの API の変更といった複数の変更が一つの差分の中に存在し，差分として複雑であった．また，どちらの差分についても提案手法の方が短い編集スクリプトを出力していた．これらのことから，差分の理解という点で，提案手法は複雑な差分よりも単純な差分に対してより有効であることが分かる．

7. 関連研究

Guillermo らは，GumTree が出力する編集スクリプトが開発者の意図を十分に表現できていないと指摘している [15]．彼らは GumTree を C#のプロジェクトに適用し，86 個の差分のうち 27%の差分が開発者の意図を表現できていなったと述べている．GumTree のマッチング処理に問題があることを指摘し，更に GumTree は AST を単なる木構造と扱っていることに問題があると述べている．彼らは GumTree がマッチング処理をする際に，プログラミング言語の特徴を考慮する必要があると提言している．

また，Veit らはプログラミング言語の特徴を考慮した GumTree の拡張として IJM (Iterative Java Matcher) を提案している [8]．IJM は Java の特徴を利用し，(1) 部分的な対応づけ (2) 名前による対応づけ (3) ノードのマージ，の三つのアプローチで構成されている．部分的な対応づけでは同一のメソッドから該当するノードを探す．名前による対応づけでは，変数名などの名前を考慮する．ノードのマージでは差分

を理解する際に不必要なノードを親ノードとマージした AST を構築する．これら三つの手法により，短く正確な編集スクリプトの出力を行う．

一方，IJM は Java に特化した手法であるため，Java 以外のプログミング言語に適用できない．更に AST の構造を変更しているため，差分の理解には適しているが，自動プログラム修正などの GumTree を応用した研究で用いることができないという問題がある．

なお，IJM は AST の構造を変更し，ノードの数が変わってしまうため，編集スクリプトの長さの違いを平等に評価することが難しいので，今回の実験では比較対象にしていない．ただし，提案手法と IJM は排他的な手法ではなく，組み合わせて使うことができると著者らは考えている．

また，著者らは異なる手法で GumTree の拡張を行っている [10]．この手法は提案手法と同様に編集スクリプトの長さを短くし，より理解しやすい編集スクリプトを生成することを目的としている．GumTree を拡張し，挿入・削除・移動・更新の他にコピーアンドペーストという編集操作を追加している．編集操作を追加するため，提案手法と全く異なる編集スクリプトを出力するため，純粋な比較を行うことができないため，今回の実験では比較していない．なお，この手法と提案手法も IJM と同様排に他的な手法ではなく，提案手法と組み合わせて使うことができると著者らは考えている．

8. 妥当性の脅威

実験 1 では恣意的にデータセットを選ぶことを避けるために，既存研究 [2] で用いられているデータセット [13] を用いた．このデータセットは Java を対象にしているため，実験対象は Java で記述された OSS のみとなった．Java 以外のプログラム言語でも同様の結果が得られるとは限らない．

実験 2 で用いた Web ブラウザで編集スクリプトを確認するツールは編集操作の順序によって表示が変わることはない．そのため編集操作の順序が理解に影響を与えるかどうかは評価していない．

実験 2 では被験者が大学生，大学院生，教員であるかに基づいて被験者を二つのグループに分けて実験を行った．プログラミングの経験年数など他の指標に基づいてグループ分けを行った場合，実験の結果が変わる可能性がある．

実験 2 の差分の理解が十分であったかどうかの判断は著者らが手作業により行ったため，その判断には著者らの主観が含まれている．そのため，第三者が判断を行った場合，同様の結果になるとは限らない．

9. むすび

本論文では，行単位の差分情報を用いることで，AST ノードを対応づける精度を上げ，より短く，理解のしやすい編集スクリプトを出力する手法を提案した．提案手法を評価するために，7 個の OSS で実験を行い，全てのプロジェクトで編集スクリプトの長さを短くすることに成功した．また，提案手法が出力する編集スクリプトが理解の助けになることを 14 人の被験者に対して実験を行い，差分理解に費やす時間が全体として減ることを確認できた．

今後の課題としては，Java 以外の言語での実験を行い，提案手法の方が有用かどうかの確認をしていきたい．特に Python のようにインデントが意味をもつようなプログラミング言語の場合，行単位の差分がより発生しにくいと考えられ，Java の差分と比べて提案手法がより有効であると考えられる．

謝辞　本研究は日本学術振興会科学研究費補助金基盤研究 (B) (課題番号:17H01725) の助成を得て行われた．

文　献

[1] E.W. Myers, “Ano(nd) difference algorithm and its variations,” Algorithmica, vol.1, no.1, pp.251–266, Nov. 1986.

[2] J. Falleri, F. Morandat, X. Blanc, M. Martinez, and M. Monperrus, “Fine-grained and accurate source code differencing,” ACM/IEEE International Conference on Automated Software Engineering, ASE ’14, Vasteras, Sweden - Sept. 15 - 19, 2014, pp.313–324, 2014.

[3] A.T. Nguyen, M. Hilton, M. Codoban, H.A. Nguyen, L. Mast, E. Rademacher, T.N. Nguyen, and D. Dig, “Api code recommendation using statistical learning from fine-grained changes,” Proc. 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2016, pp.511–522, 2016.

[4] C. Macho, S. Mcintosh, and M. Pinzger, “Extracting build changes with builddiff,” Proc. 14th International Conference on Mining Software Repositories, MSR ’17, pp.368–378, 2017.

[5] J. Yi, U.Z. Ahmed, A. Karkare, S.H. Tan, and A. Roychoudhury, “A feasibility study of using automated program repair for introductory programming assignments,” Proc. 2017 11th Joint Meeting on Foundations of Software Engineering, ESEC/FSE

2017, pp.740–751, 2017.
[6] M. Tufano, J. Pantiuchina, C. Watson, G. Bavota, and D. Poshyvanyk, “On learning meaningful code changes via neural machine translation,” Proc. 41st International Conference on Software Engineering, ICSE ’19, pp.25–36, 2019.
[7] Q. Hanam, F.S.d.M. Brito, and A. Mesbah, “Discovering bug patterns in javascript,” Proc. 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2016, pp.144–156, 2016.
[8] V. Frick, T. Grassauer, F. Beck, and M. Pinzger, “Generating accurate and compact edit scripts using tree differencing,” 2018 IEEE International Conference on Software Maintenance and Evolution (ICSME), pp.264–274, 2018.
[9] G. Dotzler and M. Philippsen, “Move-optimized source code tree differencing,” Proc. 31st IEEE/ACM International Conference on Automated Software Engineering, ASE 2016, pp.660–671, 2016.
[10] Y. Higo, A. Ohtani, and S. Kusumoto, “Generating simpler ast edit scripts by considering copy-and-paste,” The 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE2017), pp.532–542, Oct. 2017.
[11] J. Matsumoto, Y. Higo, and S. Kusumoto, “Beyond gumtree: A hybrid approach to generate edit scripts,” Proc. 16th International Conference on Mining Software Repositories, MSR ’19, pp.550–554, 2019.
[12] S.S. Chawathe, A. Rajaraman, H. Garcia-Molina, and J. Widom, “Change detection in hierarchically structured information,” Proc. 1996 ACM SIGMOD International Conference on Management of Data, SIGMOD ’96, pp.493–504, 1996.
[13] M. Monperrus and M. Martinez, Cvs-vintage: A dataset of 14 cvs repositories of java software, 2012.
[14] N. Cliff, “Dominance statistics: Ordinal analyses to answer ordinal questions,” The Psychological Bulletin, vol.114, pp.494–509, 1993.
[15] G. de laTorre, R. Robbes, and A. Bergel, “Imprecisions diagnostic in source code deltas,” Proc. 15th International Conference on Mining Software Repositories, MSR ’18, pp.492–502, 2018.

（2019 年 12 月 25 日受付，2020 年 3 月 13 日再受付，
4 月 24 日早期公開）

松本淳之介

平成 30 大阪大学基礎工学部情報科学科卒業．現在，大阪大学大学院情報科学研究科博士前期課程在学中．リポジトリマイニングに関する研究に従事．

肥後　芳樹　（正員）

2002 大阪大学基礎工学部情報科学科中退．2006 同大学大学院博士後期課程了．2007 同大学大学院情報科学研究科コンピュータサイエンス専攻助教．2015 同准教授．博士（情報科学）．ソースコード分析，特にコードクローン分析，リファクタリング支援，ソフトウェアリポジトリマイニング及び自動プログラム修正に関する研究に従事．情報処理学会，日本ソフトウェア科学会，IEEE 各会員．

楠本　真二　（正員）

1988 大阪大学基礎工学部卒業．1991 同大学大学院博士課程中退．同年同大学基礎工学部助手．1996 同講師．1999 同助教授．2002 同大学大学院情報科学研究科助教授．2005 同教授．博士（工学）．ソフトウェアの生産性や品質の定量的評価に関する研究に従事．情報処理学会，IEEE，IFPUG 各会員．