社団法人 電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

テキスト行の類似度に基づいた設計文書向け差分検出手法

早瀬 康裕† 加藤 狩夢†† 伊藤 弘貴†† 坂田 祐司††† 谷野 英昭†††

† 筑波大学システム情報系
†† 筑波大学システム情報工学研究科
††† 株式会社 NTT データ

E-mail: †hayase@cs.tsukuba.ac.jp, ††karimu-katou@aist.go.jp, †††{sakatayu,taninoh}@nttdata.co.jp

あらまし コードや設計文書の差分理解に広く用いられる diff ツールは，行の内容が変更された場合には，行の追加と削除が起こったと判定してしまう．本研究では，完全一致しない行についてもその類似度に基づいて差分検出を行う手法を提案し，リバースエンジニアリングによって生成した設計文書に適用した結果を報告する．
キーワード 差分検出，Longest Common Subsequence

Difference Detection between Design Documents Based on Line Similarity

Yasuhiro HAYASE†, Karimu KATO††, Hiroki ITOU††, Yuji SAKATA†††, and Hideaki TANINO†††

† Faculty of Engineering, Information and Systems, University of Tsukuba
†† Graduate School of Systems and Information Engineering, University of Tsukuba
††† NTT DATA Corporation

E-mail: †hayase@cs.tsukuba.ac.jp, ††karimu-katou@aist.go.jp, †††{sakatayu,taninoh}@nttdata.co.jp

Abstract Diff utility is employed in difference comprehension on source code or design documents. When part of line is edited, diff detects line deletion and line insertion since diff is based on identicalness of text lines. This paper proposes difference detection algorithm that can detects partial changes in a text line and duplication and copying of lines. The algorithm is implemented and applied for a design document generated by a reverse engineering tool.
Key words Difference Detection, Longest Common Subsequence

1. はじめに

行ベースの差分 (diff) は，プログラムの差分理解の目的で広く利用されている．一例として，オープンソースソフトウェア開発では，開発者からの変更を受け付ける際には，その変更差分 (patch) をプロジェクトの核となるメンバーがレビューするのが一般的な手順となっている．[1], [2] また，多くのバージョン管理システムはで，ソースコード等の記録されたファイルに対して，任意の 2 つの時点を指定してその差分を表示することで，プログラムにどのような変更が行われたのかを提示する機能を持っている．

差分検出の応用として，NTT データで開発したリバースエンジアリングツールには， ソフトウェアプロダクトの 2 つの時点のソースコードからリバースエンジニアリングによって設計文書を生成し，その差分を視覚的に表示する機能が含まれている [3]．この機能の特徴として，一般的な diff ツールが検出する行の追加削除だけではなく，インデントの深さが変化したことを検出することができる．しかし，このインデント変化の検知は，開発者が実際に行なった変更と必ずしも一致しないという問題があった．

そこで，本論文では，開発者が実際に行なった変更により近い形で差分を検出することを目的として，新しい差分検出手法を提案する．一般的な diff ツールが，行の内容が一致することを基準に差分を検出するのに対し，提案手法は行の内容とインデントの深さを基準とした類似度に基づいて，差分検出を行う．さらに，提案手法は，コード行の複製や移動を検出することができる．

以下，2 節で関連研究を，3 節で提案手法の詳細を，4 節で提案手法を適用した結果について，5 節でまとめと今後の課題について述べる．

2. 背 景

2.1 テキストの差分検出コマンド diff

まず，列構造のデータの比較に用いられる概念に， Longest

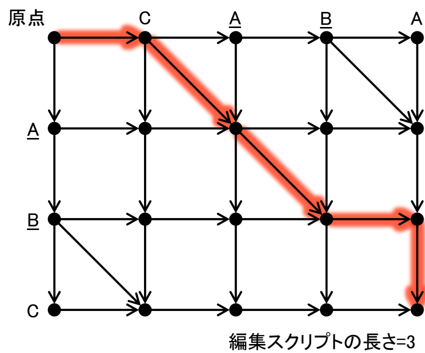

図 1　Edit Graph の例

Common Subsequence (LCS) [4] と Shortest Edit Script (SES) [5] がある．LCS は，複数の 列に共通する部分列のうち，最長のものとして定義される．一方，SES は，与えられた 2 つの列に対して，一方の列からもう一方の列へ変換する最も短い一連の操作 (編集スクリプト) と定義される．編集スクリプトに用いることができる操作として最もシンプルなものは，「列中への 1 つの要素の追加」と「列中の 1 つの要素の削除」の 2 つである．2 つの列について考える場合，LCS と SES は相補的であり，一方を得られれば，もう一方を比較的少ない計算時間で求めることができる．

2 つの列の LCS を求める方法はいくつか知られているが，Wu らによって提案された計算量が O(NP) アルゴリズム [6] は，広く利用されており，効率もよいと考えられている．O(NP) アルゴリズムは，Myers によって提案された O(ND) アルゴリズム [5] を改善したものである．

O(ND) アルゴリズムと O(NP) アルゴリズムでは， Edit Graph というデータ構造を用いて LCS を求める．Edit Graph の例として，列「A B C」と列「C A B A」を比較した場合を図 1 に示す．Edit Graph は二次元のグリッドに頂点をした有向グラフである．グリッドの縦横軸は，比較する 2 つの列の各要素に対応する．ただし，最も左と最も右のグリッドには対応する要素はない．全ての頂点には，左または上に隣接する頂点から有向辺が引かれる．また，グリッドに対応する記号が縦横で同一の頂点には，左上の頂点から有向辺が引かれる．辺にはコストが定義されており，水平もしくは垂直の辺のコストは 1，斜辺のコストは 0 である．

このグラフを用いて LCS と SES を同時に求めるには，左上の原点を起点とし，右下の頂点を終点とするコスト最小の経路を求めればよい．経路のうち，斜辺は LCS に，水平または垂直の辺が SES に対応する． O(ND) と O(NP) の両アルゴリズムは，一般のグラフに対する最小コスト経路探索アルゴリズムであるダイクストラ法 [7] に変更を加えることで，より小さな計算量で最小コストの経路を発見することができる． 図 1 の場合，縁取りで強調した経路が，コスト最小の経路として発見される．この場合， LCS は AB となる．

diff コマンド [8] は，テキストファイルを，各行を要素とした列とみなして LCS と SES の差分検出を行う． diff コマンドには，いくつかの条件でテキストの違いを無視するオプション

```
2       if      ソート{CTL-SORT} == 入力_在庫情報{CTL-I-ST}
3       then    ソート{CTL-SORT} == 入力_在庫情報{CTL-I-ST}
4       if      入力_在庫情報_引当数量{I-ST-ALLOCATE-QTY} > 0
5       then    入力_在庫情報_引当数量{I-ST-ALLOCATE-QTY} > 0
6       assign  出力_明細書{O-SP-REC}    ソート{SORT-REC}
6       if      入力_在庫情報_在庫区分{I-ST-STOCK-KBN} == '01'
7       then    入力_在庫情報_在庫区分{I-ST-STOCK-KBN} == '01'
8       assign  出力_明細書更新{O-SP-NEW}         '21'
7       else    入力_在庫情報_在庫区分{I-ST-STOCK-KBN} == '01'
8       assign  出力_明細書更新{O-SP-NEW}         '22'
```

図 2　処理記述ファイルの例

```
<<処理記述>>

 1．  【SORT-IN-PROC-01】
   1)   PROC-I-SP-READを実行する。

   2)   以下の処理を繰り返し行う。
        終了条件：入力_明細書{SW-END-I-SP} ＝ '1'
     (1)   入力_明細書{CNT-I-SP} ＋ 1        ⇒   入力_明細書{CNT-I-SP}
     (2)   入力_明細書{I-SP-REC}             ⇒   ソート{SORT-REC}

     (3)   PROC-READ-100を実行する。

     (4)   条件（受注区分{SORT-ORDERED-KBN} ＝ SPACE）に対して
        a.    条件を満たす場合
          a)    条件（セントラル倉庫{SORT-CENTRAL} ＝ '0001' または
                     CENTRAL-100 ＝ '0002' または
                     CENTRAL-100 ＝ '0003'）に対して
            (a)   条件を満たす場合
              ｱ.    入力_明細書{I-SP-REC}             ⇒   出力_明細書{O-SP-REC}

              ｲ.    出力_明細書{O-SP-REC}を書き込む。
```

図 3　処理記述ファイルの視覚表示の例

があり，一定ルールの範囲内で行なわれた行内の変更を吸収して差分検出を行うことができる．具体的には，大文字小文字の違いを無視するオプションや，空白文字に対する数種類の変更を無視するオプションがある．

2.2　大規模ソフトウェアシステムの変更の把握

大規模 COBOL システムの仕様を把握することを目的として，NTT データでは，システムのソースコードから仕様書を生成するツール (リエンジニアリングツール) を開発した [3]．本論文では，リエンジアリングツールの持つ機能のうち，COBOL 文書からリバースエンジニアリングにより生成する処理記述ファイルと呼ばれる設計文書と，その設計文書の差分表示機能に着目する．

以下で，処理記述ファイルと，その差分検出機能について詳しく説明する．

2.2.1　処理記述ファイルと視覚化

処理記述ファイルは，リエンジアリングツールがソースコードから生成するドキュメントであり，処理記述ファイルの 1 行がソースコードの 1 つの文に対応している．図 2 に処理記述ファイルのサンプルを示す．

行の先頭に書かれた数値は，条件文や繰返し文などによる入れ子の深さを表している．この入れ子の深さが，次に述べるの視覚化システムによりインデントとして表示される．数値の右に書かれている単語は，文の種類を表すラベルである．文の種類には，条件文や代入文などがある．さらにその右にある文字列が，文の内容である．文の内容の例としては，条件文の場合には条件式が，代入文の場合は代入先の変数や代入される値が挙げられる．

生成された処理記述ファイルは，視覚化機能により，より分

```
追 (4)  条件 (入力_明細書{CNT-I-SP} = '99') に対して
追   a.  条件を満たす場合
+2     a)  条件 (受注区分{SORT-ORDERED-KBN} = SPACE) に対して
+2       (a)  条件を満たす場合
+2         ア.  条件 (セントラル倉庫{SORT-CENTRAL} = '0001' または
                    CENTRAL-100 = '0002' または
                    CENTRAL-100 = '0003') に対して
+2           ア)  条件を満たす場合
追             (ア)  入力_明細書{I-SP-REC}          ⇒  出力_明細書{O-SP-REC}

追             (イ)  出力_明細書{O-SP-REC}を書き込む。

追             (ウ)  スキップ1{CNT-SKIP1} + 1        ⇒  スキップ1{CNT-SKIP1}
追             (エ)  出力_明細書{CNT-O-SP} + 1       ⇒  出力_明細書{CNT-O-SP}

追           イ)  条件を満たさない場合
追             (ア)  ソート{SORT-REC}にレコードの値を引き渡す。

追       (b)  条件を満たさない場合
         ア.  入力_明細書{I-SP-REC}          ⇒  出力_明細書{O-SP-REC}

         イ.  出力_明細書{O-SP-REC}を書き込む。
```

図 4　処理記述ファイルの差分表示の例

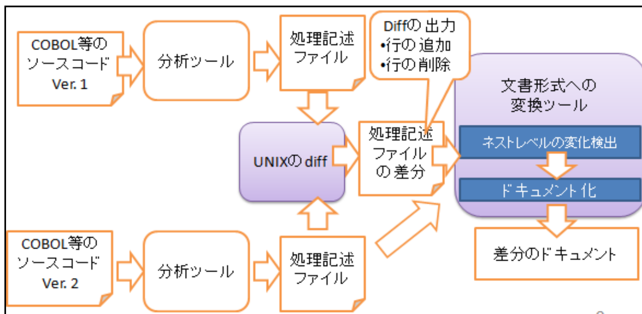

図 5　既存手法で差分を計算する手順

かりやすい形式に変換されて表示される．視覚化機能による表示例を図 3 に示す．

2.2.2　処理記述ファイルの差分検出 (既存手法)

リエンジアリングツールは，開発者がプログラムの変更内容を理解する作業を支援することを目的とした，2 バージョンの処理記述ファイルの差分を視覚的に表示する機能 (差分表示機能) を持つ．差分表示に必要な情報を得るため，内部で diff ツールが利用されている．

まず，差分表示機能の動作例を 図 4 に示す．差分の表示は，新しいコードに対する注釈として表示され，1) 追加された行，(2) インデントレベルが変化した行，の 2 種類が分かりやすく表示される．追加された行では，行の先頭に「追」の文字が表示され，行全体がハイライト表示される．また，インデントが変化した行では，行の先頭に変化量を表す正または負の数値が表示される．

次に，差分表示に必要な情報を得るための手順 (図 5) を説明する．差分を求める 2 つのソースコードが，ユーザから入力として与えられる．この 2 つのソースコードから処理記述ファイルを生成し， diff コマンドに与えることで，2 つの処理記述ファイルの間で，どこにどのような行が追加され，削除されたかが得られる．最後に，連続する場所に追加された行と削除された行に対して，インデントレベルのみが異なり，その他の内容が同一である行を探し出し，これをインデントレベルの変化として検出する．

しかしこの手法は，差分検出を適切に行えない場合があった．

不適切な検出 (1) 行の移動やコピーが行なわれると，行の追加と削除として扱われる:行の移動は，処理の順序を変更したり，コードの可読性を改善する目的で行なわれる．しかし，既存の差分計算手順では，移動について特別な扱いはしないため，移動前の行が削除され，移動後の行が追加されたとして検出されてしまう．

不適切な検出 (2) 行の中身が一部でも変更されていると，行の追加削除と計算されてしまう: 行中の変更は，手続呼び出しのパラメータに使われている変数名が変更されたり，パラメータを一部変更する場合などに行なわれる．既存の差分計算手順は，行中の一部に変更があると，その行の内容は異なると扱うため，行全体が追加されて削除されたと判定してしまう．

不適切な検出 (3) インデントの変化と他の変更が同時に起こった場合，インデントの変化を検出できない:　前述のように，インデントレベルの変化の検出は，連続する場所で検出された追加行と削除行の間で行なわれる．例えば，開発者によって行の移動とインデントレベルの変化が同時に行われた場合，削除と追加が検出される場所が異なるため，インデントレベルが変化された行があることは検出できず，行の追加削除として検出されることになる．

3. 提 案 手 法

既存の差分検出手順である diff に後処理を加えた手法よりも，開発者の変更を正確に反映した処理記述ファイルの差分計算を行うことを目的として，行の移動とコピー，および行中の変更を検出する手法を提案する．目的の達成のためのキーアイデアは下の 2 点である: 1) 行中の変更の大きさを定量化するために，内容に基いた行の類似度を定義する．2) 行中の変更を検出するために，行の内容の一致ではなく，行の内容の類似度に基づいて，類似した部分列を検出する． 3) 移動とコピーの検出のために，類似部分列を再帰的に検出する．なお，行の類似の定義は処理記述ファイルに特化しているが，類似部分列の検出方法および移動とコピーの検出方法は一般の列に適用可能な手法となっている．

以下，行の類似度の定義，類似部分列の計算方法，移動とコピーの検出方法についてそれぞれ説明する．

3.1　処理記述ファイルにおける行の類似度

処理記述ファイルの行にどれくらいの変更があるかを定量化するために，2 つの行の類似度を定義する．類似度は，0 以上 1 以下の実数値であり，行の内容が似通っているほど 1 に近い値となる．行の内容が完全に一致している場合は，類似度は 1 となる．

類似度の正確な定義は下の通りである．

文の種類が異なる場合　処理記述ファイルの 2 カラム目で示される文の種類が異なる場合には，類似度は 0 とする．

文の種類が同一である場合　2 つの行から文の内容 s_1 と s_2 と，そのレーベンシュタイン距離 [9] $d(s_1, s_2)$ に基づいて $1 - (d(s_1, s_2)/(\mathrm{len}(s_1) + \mathrm{len}(s_2)))$ と定義する．

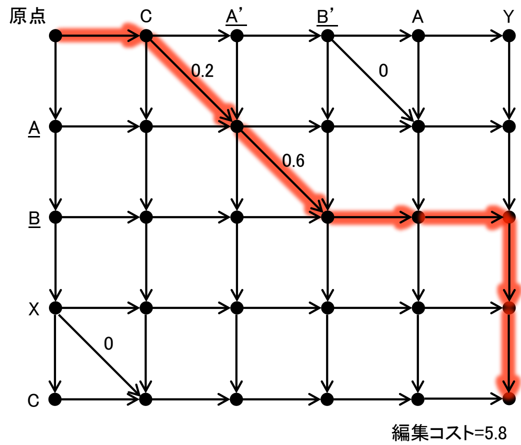

図 6　類似度について拡張した Edit Graph

3.2　類似部分列の計算手順

3.2.1　Edit Graph の類似度拡張

Edit Graph 上で類似度を表現するために， Edit Graph の斜辺に重みの実数値を与え，このグラフ上でコスト最小の経路を発見する．斜辺のコストは，要素同士の類似度 s を用いて $2 \times (1 - s)$ と定義する．この式の値は， 0 以上 2 以下の実数値であり，類似度が 1 (すなわち，行の内容が同一) であるときに最小値 0 をとる．

Edit Graph の拡張を，例を用いて説明する．図 6 は，2 つの列「A B X C」と「C A' B' A Y」を比較する場合の Edit Graph である．ここで，A と A'，B と B' は同一ではないが類似した行であり，類似度がそれぞれ 0.9，0.7 であるものとする．拡張前の Edit Graph と同一であるのは，頂点をグリッドに配置することと，頂点と列の関係，水平および垂直の辺とそのコストが 1 であることである．一方，斜辺については，要素の内容が同一でなくとも，類似度が閾値よりも高い場合には斜辺が存在する．例では， A と A' に対応する斜辺のコストは $2 \times (1 - 0.9) = 0.2$，B と B' に対応する斜辺のコストは $2 \times (1 - 0.7) = 0.6$ となっている．

次に，拡張 Edit Graph 上での最小コスト経路を発見するアルゴリズムについて，詳細を説明する．このアルゴリズムは O(ND) アルゴリズムと O(NP) のアイデアに基いている．O(NP) アルゴリズムはコストが離散値であることを利用し，インクリメントしながら計算を進めるため，そのままでは斜辺が実数値となる拡張 Edit Graph の探索には使用できない．そこで，O(NP) アルゴリズムの核となるアイデアである P 値と交換可能な意味を持つ P' 値を定義し，この P' 値に基づいてダイクストラ法を実施する． P'-value の定義は下の通りである．

P' = 起点からの既知の経路のうち最小のコスト ＋ 終点のある斜辺までの移動コスト

図 6 の例では，縁取りで強調した経路が最小コストの経路として発見される．このとき，取り出される部分列は，列「A B X C」からは「A B」が，列「C A' B' A Y」からは「A' B'」となる．

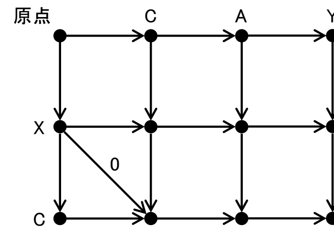

図 7　移動の検出 (再帰 1 回目) の Edit Graph

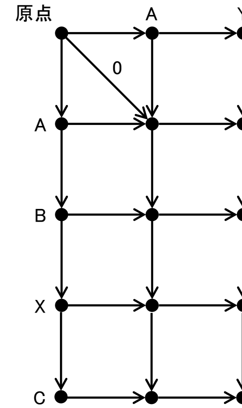

図 8　コピーの検出 (再帰 1 回目) の Edit Graph

3.3　行の移動とコピーの検出

行の移動とコピーを検出するために，類似部分列の検出を再帰的に適用する．

先に行うのは移動の検出であり，これは一度目の類似部分列検出において，類似部分列に含まれなかった要素による部分列をそれぞれ作成し，これに対して類似部分列検出を実施する．2 回目の検出で検出された類似部分列は，2 つの列の中で移動したものと考えることができる．類似部分列の検出が十分に少なくなるまで，この処理を再帰的に実施する．

図 6 の例では，類似部分列に含まれなかった部分列として，列「A B C」からは「X C」が，列「C A' B' A Y」からは「C A Y」が取り出される．そこで，この 2 つの部分列に対して，類似部分列検出を再度実施する (図 7) これにより，それぞれの列から「C」と「C」が類似部分列として検出される．残る要素は，「X」と「A Y」となるが，ここからは類似部分列は検出されないため，移動の検出を終了する．

次にコピーの検出を行う．移動の検出と同様に再帰的な適用を行うが，縦横軸に置かれる列が異なる．縦軸には常に，最初に与えられた入力の前者を置く．横軸には，移動の検出で最後に残った部分列を置く．ここで検出される類似部分列は，コピーされたものと考えることができる．再帰的実行では，縦軸の列は変化させず，横軸の列を直前の実行で残った要素の列とする．再帰的実行は，類似部分列の検出が十分に少なくなるまで繰り返す．

コピー検出を 図 7 の実行後に行なった場合の例を， 図 8 に示す．縦軸には最初に与えられた列である「A B X C」が対応する．横軸には，移動の検出で横軸で最後に残った列である「A Y」が対応する．類似部分列はそれぞれ「A」と「A」であ

表 1　評価と変更内容，検出結果の詳細

提案手法の評価値	既存手法の評価値	変更内容	件数	提案手法の検出結果	既存手法の検出結果
5	5	新しい行の追加と削除	5	正しく検出	正しく検出
		行の削除とインデント変化	2	正しく検出	正しく検出
	4	IF 文の分岐条件の追加	7	条件の追加部分を正しく検出	IF 文の削除と追加
	3	行のコピーと移動，行中の変更の複合	6	正しく検出	行の追加と削除
4	5	強い特徴のない呼出し文の追加	1	コピーと行中の変更	行の追加
	3	行のコピーと，行中の変更の複合	2	10/12，6/8 を正しく検出し，残りを行の追加として検出	行の追加

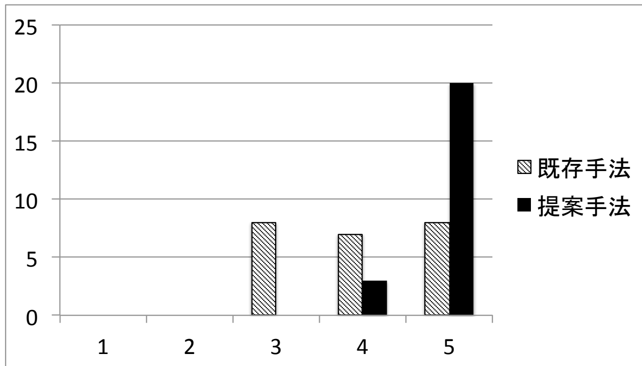

図 9　既存手法と提案手法の評価値の分布

り，それぞれがコピー元とコピー先となる．横軸で残る要素の列は「Y」となるため，次の再帰的実行は「A B X C」と「Y」の間で行なわれるが，ここからは類似部分列は検出されないため，コピーの検出を終了する

4. 適用事例

提案手法の精度を評価するために，実際のソフトウェアプロダクトを用いた評価実験を行なった．同一プロダクトの 2 つのバージョンに対して，提案手法と既存手法を用いて差分検出を行い，その結果を人間が目視によって評価した．

評価対象は，実際に運用されているソフトウェアシステムの 1 つのサブシステムであり，記述言語は COBOL である．対象サブシステムの規模は 41K ステップ，74 ソースファイルである．比較を行なったリビジョンは 2 つで，このリビジョン間で変更されたのは，9 つのファイル中の 23 箇所であった．

評価の方法としては，検出された差分が開発者による変更内容を分かりやすく表現していたかを基準とし，システムに詳しい人間が 5 段階で評価した．評価値 5 が最も良い評価を表しており，開発者による変更内容を正確に表現できたことを意味する．一方，評価値 1 は，開発者による変更内容とは全く異なる検出結果が得られたことを表す．

評価結果の概要を， 図 9 に評価値の分布として示す．全 23 件のうち 20 件で，提案手法は行のコピーや行中の変更といった変更内容を正確に検出することができていた．

次に，表 1 に，変更内容と検出結果の詳細を，評価値と並べて示す．23 件の変更のうち，7 件はどちらの手法でも変更内容を正確に検出することができた．15 件では提案手法の評価が既存手法を上回った一方，1 件で，提案手法の評価値が 4，既存手法の評価値が 5 となり，既存手法が上回った．

提案手法が既存手法を下回った 1 件について，変更内容と検出結果をさらに具体的に説明する．古いソースコードの該当する箇所では呼び出し文 (PERFORM 文) が連続して 10 行存在しており，行ごとに呼び出し先が異なっていた．一方で，新しいソースコードでは，連続した呼び出しの末尾に，類似した呼び出し文がさらに 5 行追加され，呼び出しの連続が 15 行になっていた．この変更に対して提案手法が出力した結果は,「10 行の後半 5 行をコピーして，行中を変更した」というものであった．しかし，コピーされたと判断された 5 行は，個々の文が類似しているといった具体的な対応は無いと考えられたため，変更内容を不正確に表現していると判断された．一方で，既存手法は，追加された 5 行の呼び出し文を行の追加と判断したため，開発者による変更を正確に表現できたと判断された．

5. まとめと今後の課題

本研究では，ソースコードから生成された設計文書の 2 つのバージョンに対して，開発者による変更内容をより正確に検出することを目的とした，差分検出アルゴリズムを提案し実装した．提案手法による変更の検出の粒度は diff やリエンジアリングツールと同じくテキストの行であるが，行の移動とコピー，行中の変更を検出することができる．

提案したアルゴリズムでは，それぞれのテキストを行の列とみなし，行の内容に基づいて定義した行の類似度を最大化するように，部分列を抽出するアルゴリズムを提案した．さらに，この類似した部分列の検出を再帰的に適用することで，行の移動とコピーを検出することができる．

実際のソースコードを用いて提案手法を評価した結果，多くの場合において，既存手法と同等かそれ以上に，変更内容を正確に検出できることがわかった．

今後の課題としては，より大規模な評価実験を行うことと，構文木に対する差分検出手法との比較実験を行うことが考えられる．また，提案手法を拡張し，他の種類のテキストデータに対しても適用可能にすることを計画している．

文　献

[1] J. Asundi and R. Jayant, “Patch review processes in open source software development communities: A comparative case study,” Proceedings of the 40th Annual Hawaii International Conference on System Sciences, pp.166c–, HICSS ’07, IEEE Computer Society, Washington, DC, USA, 2007.

[2] A. Mockus, R.T. Fielding, and J.D. Herbsleb, “Two case studies of open source software development: Apache and mozilla,” ACM Trans. Softw. Eng. Methodol., vol.11, no.3, pp.309–346, July 2002.

[3] 平岡正寿，谷野英昭，坂田祐司，“確実なシステム更改を実現するマイグレーション／リエンジニアリング技術,” ビジネスコミュニケーション，vol.48，no.1，pp.74–75，2011.

[4] D. Maier, “The complexity of some problems on subsequences and supersequences,” J. ACM, vol.25, no.2, pp.322–336, April 1978.

[5] E.W. Myers, “An O(ND) difference algorithm and its variations,” Algorithmica, vol.1, pp.251–266, 1986.

[6] S. Wu, U. Manber, G. Myers, and W. Miller, “An O(NP) sequence comparison algorithm,” Inf. Process. Lett., vol.35, no.6, pp.317–323, 1990.

[7] E.W. Dijkstra, “A note on two problems in connexion with graphs,” NUMERISCHE MATHEMATIK, vol.1, no.1, pp.269–271, 1959.

[8] Free Software Foundation, Inc., “Gnu diffutils - comparing and merging files,” diffutils 2.8.7 edition, April 2004.

[9] R.A. Wagner and M.J. Fischer, “The string-to-string correction problem,” J. ACM, vol.21, no.1, pp.168–173, Jan. 1974.