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.

欠陥の同時修正支援における関数クローン検出ツールの有効性調査

沼田　聖也† 　吉田　則裕†† 　崔　　恩瀞††† 　井上　克郎†

† 大阪大学大学院情報科学研究科　〒 565–0871 大阪府吹田市山田丘 1–5
†† 名古屋大学大学院情報科学研究科　〒 464–8601 名古屋市千種区不老町
††† 奈良先端技術大学院大学情報科学研究科　〒 630–0192 奈良県生駒市高山町 8916-5
E-mail: †{s-numata,inoue}@ist.osaka-u.ac.jp, ††yoshida@ertl.jp, †††choi@is.naist.jp

あらまし　コードクローン (ソースコード中に存在する同一または類似した部分を持つコード片) の存在は，ソフトウェア保守を困難にする大きな要因となる．開発者はコードクローンに欠陥が見つかった場合，同一クローンセット (互いにコードクローンとなっているコード片の集合) 内に含まれる全てのコードクローンに対して同一の修正をするか検討する必要がある．本研究では，情報検索技術に基づいて関数単位のコードクローンを検出するツールに対して，欠陥を含むコード片のコードクローンを検出できるか評価し有効性を調査した．その結果，関数クローン検出ツールは高い適合率を表した．

キーワード　コードクローン，ソフトウェア保守，関数クローン検出ツール，情報検索技術

Investigating the effectiveness of function clone detection tool for simultaneous fixing of defects

Seiya NUMATA†, Norihiro YOSHIDA††, Eunjong CHOI†††, and Katsuro INOUE†

† Graduate School of Information Science and Technology, Osaka University　1–5, Yamadaoka, Suita, Osaka, 565–0871, Japan
†† Graduate School of Information Science, Nagoya University Furo-cho, Chikusa-ku, Nagoya, 464-8601, Japan
††† Graduate School of Information Science, Nara Institute of Science and Technology　8916–5, Takayama, Ikoma, Nara, 630–0192,Japan
E-mail: †{s-numata,inoue}@ist.osaka-u.ac.jp, ††yoshida@ertl.jp, †††choi@is.naist.jp

Abstract　Code clone (i.e., code fragment that has identical or similar fragments in source code) is one of the factors that makes software maintenance more difficult. Once a developer finds a defect in a code fragment, he/she has to inspect the all of the code clones of the code fragment. In this study, we investigated the effectiveness of an IR-based function clone detection tool for detecting code clones that include defects, and then confirmed high precision of the tool.

Key words　Code clone, Software Maintenance, Function clone detection tool, Information retrieval technique

1. ま え が き

コードクローンとは，ソースコード中に存在する同一，あるいは類似した部分を持つコード片を意味し，主に既存のコード片のコピーアンドペーストによって生成される [3] [4] [5] [10] [13]．コードクローンの存在は，ソフトウェア保守を困難にしている大きな要因の 1 つとして挙げられている．

コードクローンに対する保守作業の 1 つとして，同時修正が挙げられる [6] [11]．例えば，あるコード片に欠陥が見つかった場合，そのコード片のコードクローンにも同様の欠陥が含まれる可能性が高い [17]．そのため，開発者はコードクローンに欠陥が見つかった場合，同一クローンセット（互いにコードクローンとなっているコード片の集合）内に含まれる全てのコードクローン対して同様の修正をするか検討する必要があるが，すべてのコードクローンを開発者が手作業で見つけて管理するのは困難である．そこで，開発者を支援するためにコードクローン検出ツールが利用される [9]．

山中らは，情報検索技術に基づいて関数単位のコードクロー

ンを検出するツール（以降、関数クローン検出ツールと呼ぶ）を開発した [19]．しかし，関数クローン検出ツールは，関数単位のコードクローンの検出精度や検出時間に対しての評価はされているが，欠陥の同時修正支援における有効性については評価されていない．そこで，本論文では欠陥の同時修正対象のコードクローン検出における関数クローン検出ツールの有効性を調査した．同時修正対象のコードクローンを検出することにより，開発者のコードクローンの同時修正作業の支援ができる．また，検出されたコードクローンが同時修正対象を多く含む場合，開発者は検出したコードクローンから，同時修正対象を見つける手間を減らすことができる．本研究では，同時修正対象のコードクローン検出における関数クローン検出ツールの有効性を評価する際に，字句解析ベースのコードクローン検出ツール CCFinder との比較を行った [12]．評価実験では，Li らが用意した評価セットを用いた [14] [15]．この評価セットでは，欠陥が含まれているコード片とそのコード片と同じ欠陥を含むコードクローンの事例が含まれている．これらの評価セットに関数クローン検出ツールと CCFinder を適用し，false positive と false negative や再現率，適合率と F 値という評価指標を用いて評価した．その結果，関数クローン検出ツールは適合率と F 値について高い値を表した．

2. 背　　景

2.1 コードクローン

コードクローンとは，ソースコード中に存在する同一，あるいは類似した部分を持つコード片のことを意味する．一般的にコードクローンの存在は，ソフトウェアの保守を困難にさせる要因の一つと言われている [5]．コードクローンの発生の主たる要因は，既存のソースコードのコピーアンドペーストによる再利用である．一般的に，互いにコードクローンになるコード片の対のことをクローンペアと呼び，クローンペアにおいて推移関係が成り立つコードクローンの集合のことをクローンセットと呼ぶ．

コードクローンに対する保守作業の 1 つに，ソースコードの同時修正が挙げられる [6] [11]．例えば，あるコード片に欠陥が見つかった際に，そのコード片のコードクローンにも同様の欠陥が含まれている可能性が高い．そのため，開発者はコード片に欠陥が見つかった場合，同一クローンセット内に含まれる全てのコードクローンに対して同一の修正をするか検討する必要がある．しかし，すべてのコードクローンを開発者が手作業で見つけて管理するのは困難である．そこで，開発者を支援するためにコードクローン検出ツールが利用される [9]．

2.1.1 コードクローンの定義

コードクローンには，普遍的定義は存在しない．本論文では，コードクローンの定義として以下の 4 つのタイプの分類を用いる [16] [19]．

タイプ 1　空白の有無，レイアウト，コメントの有無などの違いを除き完全に一致する．

タイプ 2　タイプ 1 の違いに加えて，変数名などのユーザ定義名，関数の型などが異なる．

タイプ 3　タイプ 2 の違いに加えて，文の挿入や削除，変更などが行われている．

タイプ 4　類似した処理を実行するが，構文上の実装が異なる．

タイプ 4 のコードクローンとしては，以下のものが挙げられる．

- 条件分岐処理や繰返し処理などの制御構造の実装が異なる．
- 中間媒介変数の利用の有無が存在している．
- 文の並び替えが発生している．

2.1.2 コードクローンの検出

様々な粒度に基づいてコードクローンを検出するツールが開発されてきた．[9]．以下に本論文で利用したツールの検出粒度の例を挙げる．関数単位の検出では，関数を 1 つの要素とし，等価な要素対を見つけることでコードクローンを検出する [19]．この手法は，関数の一部のみがコードクローンであるものを検出できない．コードクローン検出のために，プログラムの性質を計測した特徴ベクトルを用いる．

字句単位の検出では，プログラムの字句解析を行った後，その字句を要素とした系列に対してコードクローンを検出する．字句解析を行うことにより，空白やコメントを無視することができる．また，識別子や定数等の特定の種類の字句を特殊な 1 つの字句に固定することで，変数名や関数名の変更されたコード片もコードクローンとして検出することができる．

CCFinder は字句単位のコードクローンを検出するツールである [12]．このツールはまず，入力されたソースコードを字句単位に分解し，変数名や関数名等のユーザー定義名を同一字句に変更する．この変更によって，変数名や関数名等が異なるタイプ 2 までのコードクローンを検出可能である．最後に，一致する字句列をコードクローンとして見つける．CCFinder はコードクローンを高速で検出可能であり，多くの企業や研究で使用されている．

2.2 関数クローン検出ツール

この説では山中らが開発した関数クローン検出ツール [19] の説明を行う．関数クローン検出ツールでは情報検索技術を利用し，タイプ 1 からタイプ 4 のすべてのタイプの関数単位のコードクローンを検出することができる．

関数クローン検出ツールは，入力されたソースコード中のワードに基づいて各関数を特徴ベクトルに変換する．ここでワードは以下の 2 つを示す．

- 変数や関数などにつけられた識別子名を構成する単語
- 条件文や繰返し文などの構文に利用される予約語

そして，その特徴ベクトル間の類似度を計算して，クローンペアの集合をリストとして出力する．また，検出の高速化のために，類似度の計算の直前に LSH（Locality-Sensitive Hashing）アルゴリズム [8] を用いて特徴ベクトルのクラスタリングを行っている．関数クローン検出ツールのコードクローン検出手法は大きく 4 つのステップに分けられる．その関数クローン検出ツールのコードクローン検出プロセスを図 1 に示す．

STEP1：ワードの抽出

このステップでは，ソースコード中の各関数からワードの抽出

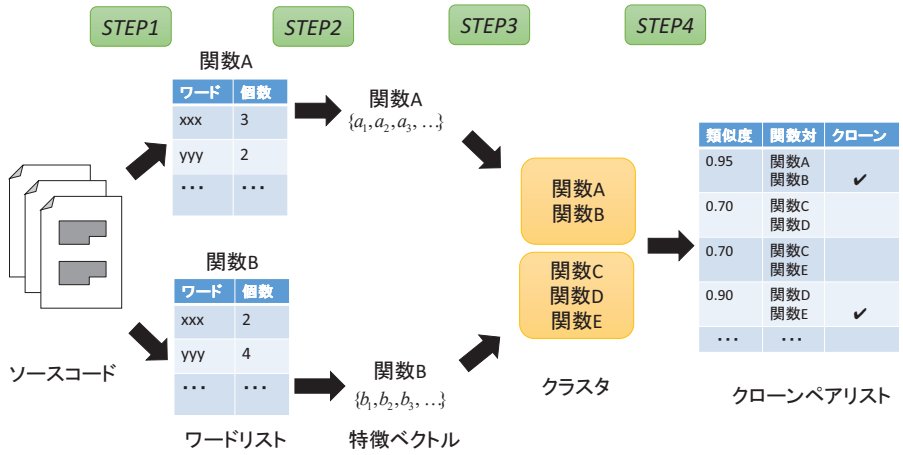

図 1　関数クローン検出ツールのコードクローン検出プロセス

を行う．識別子名が複数の単語で構成されている場合，分割を行い新たに複数のワードとする．その分割方法には，ハイフン等の区切り記号（デリミタ）による分割と，識別子名中の大文字になっているアルファベットによる分割がある．繰返し文等でよく用いられる“ i ”や“ j ”等，意味情報が込められていない変数も同一のものとして扱うために，2 文字以下の識別子名はすべて同一のワードとして認識する．

STEP2 : 特徴ベクトルの計算

STEP1 で抽出したワードに対して TF-IDF 法 [2] を利用して各ワードの重みを計算し，その値を特徴量として利用して，各関数を特徴ベクトルに変換する．TF-IDF 法による値は tf 値（関数中のワードの出現頻度）と idf 値（ソースコード全体のワードの希少さ）の積で与えられる．この手法では，全関数中の各ワードに対して重みを計算し，それらを特徴量として用いることによって特徴ベクトルを求めている．そのため，各関数の特徴ベクトルの次元はソースコード中に存在する全ワードの数となる．

STEP3 : 特徴ベクトルのクラスタリング

STEP2 で計算した各関数の特徴ベクトルに対してクラスタリングを行うことによって，クローンペアに成り得る候補を絞る．ここでは，LSH アルゴリズム [8] を用いて特徴ベクトルのクラスタリングを行う．LSH アルゴリズムを利用することによって，クエリとして 1 つの特徴ベクトルを与えると，特徴ベクトル集合からそのクエリと近似した特徴ベクトル集合のクラスタを取得することができる．本ステップであらかじめクラスタリングを行うことでクローンペアと成り得る候補を絞ることによって，検出時間にかかる計算コストを削減している．なお，LSH アルゴリズムが実装されている E2LSH [1] をこの手法では利用している．

STEP4 : 特徴ベクトルの類似度の計算

最後に，STEP3 で求めた各クラスタ中の関数のついに対してコサイン類似度を用いてクローンペアであるか否かの判定を行う．特徴量は常に正の値をとるため，コサイン類似度は 0 から 1 の範囲となる．もし，コサイン類似度が閾値以上であれば，その関数の対はクローンペアであると判定する．

こうして STEP1 から STEP4 を経て検出したクローンペアリストを結果として出力する．

山中らの関数クローン検出ツールは評価実験として，コーパス [18] を用いた検出精度の評価と，既存の関数単位のクローン検出ツールとの検出精度と検出時間の比較を行っている．しかし，彼らは関数クローン検出ツールの，欠陥の同時修正におけるツールの有効性の評価は行っていない．そこで本論文では，関数クローン検出ツールが欠陥を含むコードクローンを検出できるかを調査し，関数クローン検出ツールの欠陥の同時修正支援における有効性を調査した．

3. 調 査 手 法

3.1 調査対象の評価セット

Li らは Git, Linux kernel, PostgreSQL の 3 つのオープンソースソフトウェアから，欠陥を含むコード片とそのコードクローンを見つけた [14] [15]．Li らが，これらのオープンソースソフトウェアを利用した理由は，(1) 主に C/C++を使って書かれており，(2) バージョン履歴から欠陥を含むコードとそのコードクローンを見つけることが可能であり，(3)Git は 10 万行以上，PostgreSQL は 30 万行以上，Linux kernel は 100 万行以上ものコードであり，評価において十分なスケーラビリティを有しているからである．Li らは，欠陥を含むコード片と，そのコード片と同じ欠陥を持つコードクローンの事例（以降，評価セットと呼ぶ）をテクニカルレポートで公開している [14]．このテクニカルレポートは欠陥を含むコード片の Git リポジトリのコミット ID と，同じ欠陥を持つコードクローンの Git リポジトリのコミット ID を含んでいる．

本研究ではこの評価セットを用いて，関数クローン検出ツールの欠陥を含むコードクローン検出における有効性を調査し，CCFinder と比較した．しかし，その事例の中には欠陥を含むコード片とそのコードクローンが同じ関数内に存在するものも存在する．関数クローン検出ツールは関数単位のコードクローンを検出するため，そのようなコードクローンは検出することができない．よって，そのようなコードクローンは調査対象から取り除いて，全 38 種類のクローンセットに含まれる 58 個の

コードクローンの事例に対して調査を行った．

3.2 調 査 手 順

関数クローン検出ツールの性能を比べるために，Li らが用意した評価セットを用いて以下の手順で調査を行った．CCFinder を評価の指標として用いたのは，CCFinder は多くの実験で利用されており，その有効性が確認されているからである．本研究の調査手順の概要は以下の通りである．

（1） 欠陥を含むコード片と，それと同じ欠陥を持つコードクローンの例として挙げられているコード片が含まれている各コミットのスナップショットを Git リポジトリから取得する．

（2） 関数クローン検出ツール，CCFinder でコードクローン検出を行う．

（3） 検出結果の評価を行う．

まず，Git, Linux kernal, PostgresSQL の Git リポジトリから評価セットの欠陥を含むコード片と，そのコード片と同じ欠陥を持つコードクローンがあるコミットをチェックアウトする．次に，関数クローン検出ツールと CCFinder で評価セットのコード片やコードクローンの検出を行う．その際，Git と PostgreSQL はプロジェクト全体のソースコードに対してクローン検出を行った．しかし，Linux kernel はプロジェクトが大きすぎて，プロジェクト全体に対してコードクローン検出を行うことができなかったため，コードクローンが存在するディレクトリの周辺に範囲を絞ってコードクローン検出を行った．CCFinder と関数クローン検出ツール間で，適用範囲は統一している．

関数クローン検出ツールで欠陥を含むコードクローンを検出するためには，まず，buggy.c というファイルを作り，その中に欠陥を含む関数のコード片を抜き取り保存する．次に，buggy.c を欠陥を含むコード片と同じ欠陥を持つコードクローンが含まれているバージョンの中に保存をする．そして，そのバージョンに対して，関数クローン検出ツールをかけて buggy.c に保存した関数と，コードクローンの例が検出されるかどうかを調べた．buggy.c を用意したのは，欠陥のあるコード片を含む関数のある場所をわかりやすくして，検出結果を確認するときに，誤りが生じないようにするためである．また，関数クローン検出ツールの閾値はデフォルトの 0.9 と 0.5 の 2 種類で調査を行った．閾値の設定による関数クローン検出ツールの欠陥を含むコードクローンの検出精度の違いを確認するために，閾値をデフォルトだけでなく 0.5 でも検出を行った

CCFinder は Libra を用いて評価した [7]．Libra は特定のコード片を引数として与えると，そのコードクローンを見つけるツールである．Libra に，引数として欠陥を含むコード片を与えて，欠陥を含むコード片と同じ欠陥を持つコードクローンの例があるバージョンに対してコードクローン検出を行い，コードクローンの例を検出することができるかどうかを調べた．CCFinder の最小一致トークン数は，Li らの実験に合わせて 10 として用いた．

最後に，38 個の欠陥を含むクローンセットの例すべてに対して上で述べた手順を繰り返し行い，次に説明する N1〜N4 の評価指標と，再現率と適合率の評価指標を用いて評価を行った．

3.3 評 価 指 標

2 種類の評価指標を用いた．具体的には，Li らの研究と同様に false positive と false neative に基づく指標や再現率と適合率と F 値いう評価指標を用いた

3.3.1 false positive と false negative

false positive とは，評価セットで欠陥を含む事例として挙げられていないが，評価対象のツールではコードクローンとして検出したものを指す．falise negative とは，評価セットで欠陥を含む事例として挙げられているが，評価対象のツールではコードクローンとして検出できなかったものを指す．この 2 つの指標を用いて評価対象のツールによる検出結果を以下の 4 つに分類する．

N1：no false positives, no false negatives
評価セットに含まれている全てのコードクローンが検出され，かつ含まれてないコードクローンは 1 つも検出されてないクローンセットが分類される．

N2：no false positives, some false negatives
評価セットに含まれているコードクローンに検出漏れがあり，かつ含まれてないコードクローンは 1 つも検出されてないクローンセットが分類される．

N3：some false positives, no false negatives
評価セットに含まれている全てのコードクローンが検出され，かつ含まれてないコードクローンも検出されたクローンセットが分類される．

N4：some false positives, some false negatives
評価セットに含まれているコードクローンに検出漏れがあり，かつ含まれてないコードクローンも検出されたクローンセットが分類される．

3.3.2 再現率と適合率と F 値

false positive と false negative による評価指標では，false positive と false negative の数量の大小を考慮していない．そこで，false positive と false negative の数量をはっきりとさせるために再現率と適合率と F 値いう評価指標を用いた．再現率と適合率と F 値の説明を以下に示す．

再現率とは，正解のうち検出されたものを指すものであり，網羅性に関する指標として用いられる．本研究での再現率は，評価セットで欠陥を含む事例として挙げられているコードクローンのうち検出ツールが検出したものの割合を表す．なお，再現率が高いほどそのツールは性能が良いと判断できる．

適合率とは，結果において本当に正しかったものの割合を指すものであり，正確性に関する指標として用いられる．本研究での適合率は，検出ツールが実際に検出したコードクローンのうち，評価セットで欠陥を含む事例として挙げられているコードクローンの割合を表す．なお，適合率が高いほどそのツールは性能が良いと判断できる．

F 値とは，再現率と適合率という網羅性と正確性の総合的な評価の際に利用される尺度として用いられるものである．再現率と適合率は互いにトレードオフの関係であるため，F 値を高くできれば総合的に良い評価となる．F 値は，再現率と適合率の調和平均によって求められる．

4. 調査結果

表 1 は，評価セットの全 38 種類のクローンセットに含まれる 58 個のコードクローンに対して関数クローン検出ツールと CCFinder を適用して，3.3.1 節で説明した N1〜N4 の数をまとめたものを示す．

表 1　N1〜N4 の分類結果

	関数クローン検出ツール		CCFinder
分類	閾値 0.9	閾値 0.5	
N1	11	10	10
N2	22	13	11
N3	4	11	16
N4	1	4	1

N1〜N4 の特性をふまえてこの結果からわかることは，まず，N1 と N3 に分類されたものは，漏れなく検出できたクローンセットである．よって，評価セットの 38 種類のクローンセットの例の内，関数クローン検出ツールの閾値 0.9 の場合は 15 個，閾値 0.5 の場合は 21 個，CCFinder は 26 個のクローンセットを漏れなく検出できたことがわかった．また，N2 と N4 に分類されたものは，評価ツールが検出できなかったコードクローンを含んでいるクローンセットである．よって，関数クローン検出ツールの閾値 0.9 の場合は 23 個，閾値 0.5 の場合は 17 個，CCFinder は 12 個のクローンセットのコードクローンを少なくとも検出できなかったことがわかった．そして，N1 と N2 に分類されたものは，評価セットに含まれているコードクローンのみ検出されたクローンセットであり，N3 と N4 に分類されたものは，評価セットに含まれているコードクローン以外にも検出されたクローンセットである．よって，関数クローン検出ツールの閾値 0.9 の場合は 5 個，閾値 0.5 の場合は 15 個，CCFinder は 17 個のクローンセットで評価セットに含まれているコードクローン以外にも検出されたことがわかった．この結果をまとめると，関数クローン検出ツールに関しては，閾値 0.9 より閾値 0.5 の場合のほうが，欠陥事例を持つコードクローンを多く検出することができ，かつ検出漏れは少なくなるが，欠陥事例を持たないコードクローンも多く検出することがわかった．また，CCFinder と比べると，関数クローン検出ツールの閾値に関係なく CCFinder のほうが欠陥事例を持つコードクローンを多く検出することができるが，欠陥事例を持たないコードクローンも多く検出したことがわかった．

N1〜N4 の評価では，評価セットのコードクローン対して false positive と false negative の数量的な評価ができない．同一クローンセットに検出されてないコードクローンと検出されたコードクローンが混在している可能性もある．この問題を解消するために，それぞれのツールで再現率，適合率，F 値を計算した．その結果を表 2 に示す．この表からわかるように，関数クローン検出ツールは閾値を 0.9 から 0.5 に下げると再現率が上がり，適合率が大幅に下がっている．また，CCFinder と比べると，関数クローン検出ツールの閾値が 0.9 の場合は，再現率が低くなり適合率は大幅に高くなっており，閾値が 0.5 の場合は似たような結果となっている．F 値を比べると，関数クローン検出ツールの閾値 0.9 の場合が最も良い値になり，閾値 0.5 の場合と CCFinder は低い値となった．以上のことから，関数クローン検出ツールは，閾値 0.9 の場合に適合率と F 値において良い結果を示したため，バグの同時修正において十分な性能を発揮できることがわかった．

表 2　関数クローン検出ツールと CCFinder の比較

	関数クローン検出ツール		CCFinder
	閾値 0.9	閾値 0.5	
検出数	41	293	2274
正解検出数	24	31	31
再現率	0.41	0.53	0.53
適合率	0.59	0.11	0.01
F 値	0.48	0.18	0.02

5. 考　　察

関数クローン検出ツールの検出結果は，閾値によって大きく変わった．閾値を下げると，再現率は上がり適合率は下がる．これら 2 つはトレードオフの関係にある．よって，適切な閾値を定める必要がある．閾値を 0.6，0.7，0.8 の場合も調べると，よりよい閾値が見つかるかもしれない．

本研究では，CCFinder の最小一致トークン数を Li らの調査に合わせて 10 に設定した．開発者がバグ事例ごとに，最小一致トークン数を適切に設定することができれば，CCFinder の適合率と F 値は上昇することが考えられる．そこで，最小一致トークン数を 20，30 に設定したときも調べてみた．その結果を表 3 に示す．正解クローン数が同じでないため，厳密には言えないが，最小一致トークン数を上げれば適合率と F 値も上がることが分かった．

表 3　CCFinder の閾値を変えた際の評価結果

	閾値 10	閾値 20	閾値 30
正解クローン数	58	7	4
検出数	2274	370	10
正解検出数	31	6	3
再現率	0.53	0.86	0.75
適合率	0.01	0.16	0.3
F 値	0.02	0.27	0.43

また，関数クローン検出ツールと CCFinder の検出結果から，両ツールに長所，短所があることが分かった．そのためこれら 2 つのツールを上手く使い分ければ，欠陥の同時修正を有効的に行えることがわかる．CCFinder はタイプ 1，2 のコードクローンしか検出できないが，関数クローン検出ツールはタイプ 3，4 のコードクローンまで検出することができる．しかし，関数クローン検出ツールでは，関数単位で類似しておらず，関数のごく一部が類似しているようなコードクローンを検出することができない．そこで，両ツールを組み合わせて利用することを考えてみた．両ツールの和集合を取った場合の，再現率と適合率と F 値を表 4 にまとめた．

表 4　関数クローン検出ツールと CCFinder の和集合の性能

	CCFinder と 関数クローン検出ツール (閾値 0.9)	CCFinder と 関数クローン検出ツール (閾値 0.5)
再現率	0.71	0.76
適合率	0.02	0.02
F 値	0.04	0.05

CCFinder と関数クローン検出ツールを組み合わせて利用する場合，表 2 にまとめている両ツールを単体で利用したときに比べて再現率が大幅に上がる．適合率と F 値に関しては，関数クローン検出ツール単体の閾値 0.9 の時に比べると大幅に下がってしまうが，閾値 0.5 の時と CCFinder 単体のときとはほとんど変わらない．よって，検出する際のコストが許されるのであれば，関数クローン検出ツールと CCFinder の両者を併用することにより，欠陥を含むコード片に対して同じ欠陥を含むコードクローンをより多く見つけることができると考える．

6. まとめと今後の課題

本研究では，関数クローン検出ツールについて，欠陥を含むコードクローン検出における有効性の評価を行った．具体的には，関数クローン検出ツールと CCFinder を使って，Li らが用意した評価セット [14] に対してコードクローン検出を行った．その結果，関数クローン検出ツールは，再現率でやや CCFinder に劣ったが，適合率と F 値について高い値を表したことがわかった．

今後の課題としては，まず，関数クローン検出ツールの適切な閾値を求めることが考えられる．本研究で利用した評価セットに対して，関数クローン検出ツールの閾値を 0.6，0.7，0.8 のそれぞれに設定した場合についても検出を行って，その結果の再現率と適合率と F 値を求めて比べる予定である．そして，CCFinder の最小一致トークン数についても適切な値を考える必要があると考えられる．欠陥事例ごとに適切な最小一致トークン数を設定することができれば，適合率と F 値は上昇すると考えられるため，開発者に最小一致トークン数を欠陥事例ごとに設定してもらいながら CCFinder を利用してもらう評価も必要であると考えられる．

謝辞　本研究を実施するにあたり，貴重なご意見をくださった日本電気株式会社 鈴木 明彦 氏，前田 直人 氏に感謝いたします．本研究は JSPS 科研費 25220003，26730036，15H06344 の助成を受けたものです．

文　献

[1] Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *Proc. of FOCS*, pp. 459–468, 2006.

[2] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. *Modern information retrieval: The concepts and technology behind search.* Addison-Wesley, 2011.

[3] Brenda S. Baker. Finding clones with dup: Analysis of an experiment. *IEEE Trans. Softw. Eng.*, Vol. 33, No. 9, pp. 608–621, 2007.

[4] Ira D. Baxter, Andrew Yahin, Leonardo Moura, Marcelo Sant Anna, and Lorraine Bier. Clone detection using abstract syntax trees. In *Proc. of ICSM*, pp. 368–377, 1998.

[5] 肥後芳樹, 楠本真二, 井上克郎. コードクローン検出とその関連技術. 電子情報通信学会論文誌, Vol. J91-D, No. 6, pp. 1465–1481, 2008.

[6] 肥後芳樹, 吉田則裕. コードクローンを対象としたリファクタリング. コンピュータソフトウェア, Vol. 28, No. 4, pp. 43–56, 2011.

[7] Yoshiki Higo, Yasushi Ueda, Shinji Kusumoto, and Katsuro Inoue. Simultaneous modification support based on code clone analysis. In *Proc. of APSEC*, pp. 262–269, 2007.

[8] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proc. of STOC*, pp. 604–613, 1998.

[9] 井上克郎, 神谷年洋, 楠本真二. コードクローン検出法. コンピュータソフトウェア, Vol. 18, No. 5, pp. 47–54, 2001.

[10] Lingxiao Jiang, Ghassan Misherghi, Zhendong Su, and Stephane Glondu. Deckard: Scalable and accurate tree-based detection of code clones. In *Proc. of ICSE*, pp. 96–105, 2007.

[11] 神谷年洋, 肥後芳樹, 吉田則裕. コードクローン検出技術の展開. コンピュータソフトウェア, Vol. 28, No. 3, pp. 29–42, 2011.

[12] Toshihiro Kamiya, Shinji Kusumoto, and Katsuro Inoue. CCFinder: a multilinguistic token-based code clone detection system for large scale source code. *IEEE Trans. Softw. Eng.*, Vol. 28, No. 7, pp. 654–670, 2002.

[13] Bruno Lague, Daniel Proulx, Jean Mayrand, Ettobre M. Merlo, and John Hudepohl. Assessing the benefits of incorporating function clone detection in a development process. In *Proc. of ICSM*, pp. 314–321, 1997.

[14] Jingyue Li and Michael D. Ernst. CBCD:Cloned buggy code detector. Technical report, Computer Science & Engineering, University of Washington, UW-CSE-11-05-02, 2011.

[15] Jingyue Li and Michael D. Ernst. CBCD: Cloned buggy code detector. In *Proc. of ICSE*, pp. 310–320, 2012.

[16] Chanchal K. Roy, James R. Cordy, and Rainer Koschke. Comparison and evaluation of code clone detection techniques and tools: A qualitative approach. *Science of Computer Programming*, Vol. 74, No. 7, pp. 470–495, 2009.

[17] Chanchal K. Roy, Minhaz F. Zibran, and Rainer Koschke. The vision of software clone management: Past, present, and future (keynote paper). In *Proc. of CSMR-WCRE*, pp. 18–33, 2014.

[18] Ewan Tempero. Towards a curated collection of code clones. In *Proc. of IWSC*, pp. 53–59, 2013.

[19] 山中裕樹, 崔恩瀞, 吉田則裕, 井上克郎. 情報検索技術に基づく高速な関数クローン検出. 情報処理学会論文誌, Vol. 55, No. 10, pp. 2245–2255, 2014.