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

信学技報
IEICE Technical Report
SS2007-47(2007-12)

識別子の共起関係に基づく類似コード検索法の提案と欠陥検出への適用

服部　剛之† 　吉田　則裕† 　早瀬　康裕† 　肥後　芳樹†
松下　誠† 　楠本　真二† 　井上　克郎†

† 大阪大学大学院情報科学研究科　〒560–8531 大阪府豊中市待兼山町 1–3
E-mail: †{thattori,n-yosida,y-hayase,higo,matusita,kusumoto,inoue}@ist.osaka-u.ac.jp

あらまし　ソフトウェアの保守を行う際に，複数箇所に同様の修正を加える必要が生じることがある．例えば，ソースコード中にある欠陥が見つかった際は，他の箇所に存在する同様の欠陥を修正しなければならない．このような場合，grep 等を使ったキーワード検索やコードクローン検出法を用いて，修正を検討すべき箇所を検索する方法が考えられる．しかし，これらの方法は，キーワードをクエリ（検索質問）として考案する必要があるという問題点や，字句解析後のトークン列が連続して一致もしくは類似する箇所でなければ，同様の修正を検討するべき箇所として提示できないという問題点がある．本稿では，これらの問題点を解決するために，クエリとしてコード片（ソースコードの一部）を与えると識別子の共起関係に基づいて類似コードを検索する手法を提案する．提案手法は，まず修正を検討すべきコード片を１つ選択しクエリとして与えると，そのコード片から特徴語（頻出する語）を自動的に抽出する．次に，抽出した特徴語の含むコード片をソフトウェア中から検出する．提案手法は２つのソフトウェアに含まれていた欠陥の多くを提示できることを確認した．
キーワード　ソフトウェア保守，類似コード，情報検索，欠陥検出

Retrieving Similar Code based on Co-occurrence of Identifiers and Its Application to Defect Detection

Takeshi HATTORI†, Norihiro YOSHIDA†, Yasuhiro HAYASE†, Yoshiki HIGO†,
Makoto MATSUSHITA†, Shinji KUSUMOTO†, and Katsuro INOUE†

† Graduate School of Information Science and Technology, Osaka University
1–3, Machikaneyama-cho, Toyonaka-shi, 560–8531, Japan
E-mail: †{thattori,n-yosida,y-hayase,higo,matusita,kusumoto,inoue}@ist.osaka-u.ac.jp

Abstract　During software maintenance, if developers modify a part of a system, they often have to apply the same modifications to other parts of the system. For detecting code that is needed to apply simultaneous modifications, developers can use keyword-based search (e.g. grep) and code clone detection tool (e.g. CCFinder). However, it is difficult to determine appropriate search keywords, and code clone detection tool has a problem with recall. In this paper, we propose a code retrieval method based on co-occurrence of identifiers. In addition, we also present some case studies on defect detection.
Key words　Software Maintenance, Similar Code, Information Retrieval, Defect Detection

1. はじめに

近年，ソフトウェアの大規模化・複雑化に伴い，ソフトウェアの保守作業を効率化することが重要な課題となってきている．ソフトウェアの保守とは，“納入後，ソフトウェア・プロダクトに対して加えられる，フォールト修正，性能または他の性質改善，変更された環境に対するプロダクトの適応のための改訂”であると定義されている [4].

ソフトウェアの保守を行う際に，複数箇所に同様の修正を加える必要が生じることがある．例えば，ソースコード中にある欠陥が見つかった際は，他の箇所に存在する同様の欠陥を修正する必要が生じる．また，ある機能を追加するために複数の箇所を修正する必要が生じることがある．このような場合，一般の開発者は grep [3] 等を使ったキーワード検索を用いて，修正

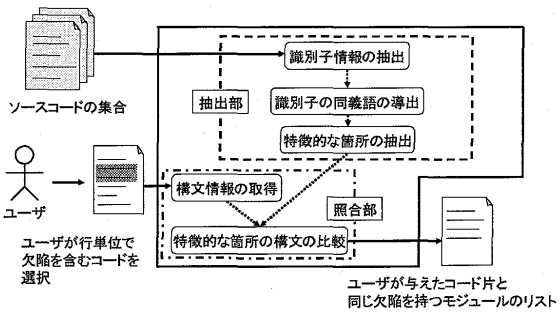


図 1　手法の概略図

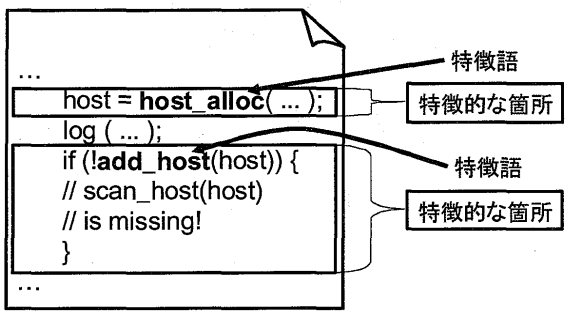


図 2　特徴語と特徴的な箇所の関係

を検討すべき箇所を検索すると考えられる．例えば，修正を検討すべき箇所に含まれると予想されるキーワードを考え，検索を行う．しかし，適切なキーワードを発見することが難しいという問題点がある．また，我々の研究グループでは，以前にコードクローン検索ツール Libra [9] を提案している．Libra は，コードクローン検出ツール CCFinder [6] が検出するクローン関係に基づく検索を行うツールである．しかし，クローンの関係でなければ検索対象にならないという問題点がある．

本研究では，コード片をクエリとし，識別子の共起関係に基づいて類似コードを検索する手法を提案する．この手法は，まず修正を検討すべきコード片を 1 つ選択しクエリとして与えると，そのコード片から頻繁に出現している識別子（特徴語）を自動的に抽出する．次に，抽出した特徴語の含むコード片をソフトウェア中から検出する．適用実験として，提案手法を用いて 2 つのソフトウェアに含まれていた欠陥の検索を行った．その結果，提案手法を用いた検索は，ソフトウェアに含まれていた欠陥の多くを提示できることがわかった．

2. 類似コード検索

2.1 定　　義

本稿では，類似コード検索を “ソースコードの集合から，クエリ（検索質問）として与えられたコード片（ソースコードの一部）と一致もしくは類似したコード片を探すこと” と定義する．なお，コード片は 5 項組 (ファイルID, 開始行, 開始桁, 終了行, 終了桁) で定義する．以降，類似コード検索に用いることができる手法について述べる．

2.2 grep を用いた方法

多くのソフトウェア開発者は，類似コード検索を行う際に grep [3] を利用すると考えられる．以下に，本稿が想定する grep を用いた類似コード検索手順を示す．

（1） あるコード片から重要と思われるキーワード（識別子，式など）を発見する．

（2） そのキーワードを grep の引数として与える．

（3） grep の出力結果を基に，キーワードを含むコード片を特定する．

この手順にしたがう開発者は，同様の修正を検討すべきコード片間で，識別子や式などのキーワードが共起しているであろうという予想に基づいて類似コード検索を行っていると考えられる．しかし，識別子の同義語をはじめ，キーワードには様々な変化形が存在するため，単一のキーワードで同様の修正を検討すべきコードを漏れなく検索結果に含めることは難しい．

また，保守対象のソフトウェアを十分に理解している開発者でなければ，適切なキーワードを発見することは難しい．grep は正規表現を用いたパターン検索を行うことができるが，grep の正規表現を十分に理解した開発者でなければ，正規表現を用いたパターンを適切に指定することは難しい．

2.3 コードクローン検索ツール Libra を用いた方法

我々の研究グループでは，以前にコードクローン検索ツール Libra [9] を提案している．Libra は，ソースコードの集合から，クエリとして与えられたコード片のコードクローン [1], [2], [6] を検索するツールである．Libra はコードクローンの検出に CCFinder [6] を用いているため，Libra は CCFinder と同じく，連続して一致するトークン列をコードクローンとする．

そのため，クエリとして与えられたコード片と比べてトークン列上の差異があるコード片は，検索結果に含まれない．例えば，クエリとして与えられたコード片に対してログ出力文や例外処理を追加したコード片は，検索結果に含まれない．

Libra はコード片をクエリとして与えられるため，開発者がキーワードやパターンを考えなくて良いという利点がある．しかし，前述のとおり，トークンが連続して一致するコード片でなければ，検索結果に含めることができないという欠点がある．また，識別子の情報（変数名など）を用いていないため，同様の修正をすべきコード片間で識別子が共起している場合は，grep を用いた方が有利である．

3. 提 案 手 法

3.1 手法の概要

本研究が提案する手法では，まず，欠陥を含むコード片とソースコードの集合から識別子の情報を抽出し，頻繁に出現している識別子（特徴語）を求める．次に，欠陥を含むコード片と同じ重要語を含むコード片を検索することで同じ欠陥を含むモジュール（C 言語における関数，Java 言語におけるメソッドなど）の検出を行う．提案手法の概略図を図 1 に示す．この手法により，自動的に複数のキーワードを発見し，類似コード片の検索を行うことができる．

提案手法は大きく分けてソースコード集合から識別子等の情報を抽出する抽出部と入力コード片，ソースコード集合の抽出された情報を比較する照合部の 2 つで構成されている．抽出

部では，ソースコードの集合から各ソースコードをモジュール単位で分割して識別子の抽出を行い，抽出された識別子から同義語を求める．そして，同義語の情報を基にモジュールの特徴的な箇所を抽出する．特徴的な箇所というのは，特徴語が含まれていて構文上意味のあるコード片とした．特徴語と特徴的な箇所の関係を図2に示す．照合部では，始めに欠陥を含むコード片を与え，特徴語と特徴的な箇所を抽出する．この特徴語と特徴的な箇所の構文情報の組を検出対象として同じ組を持つモジュールを検出することで，同じ欠陥を含むモジュールを提示する．

3.2 抽 出 部

3.2.1 識別子情報の抽出

まず，ソースコードの集合からすべてのモジュールを抽出する．次に，モジュール中に存在する識別子を抽出し，各識別子の出現回数を計算する．このとき予約語，型名は除外している．また，抽出した識別子が複数の語から成り立つ場合は，分解し個々の語を識別子として扱う．

3.2.2 識別子の同義語の導出

識別子の同義語を求めるために，識別子間の距離を求める．識別子間の距離は，自然言語処理において文書中の語の類似性を計測するために用いられている Jensen-Shannon divergence [7] [12] で求める．

ある分布を基準として別の分布との違いを計測する尺度として Kullback-Leibler divergence がある [5] [7]．Jensen-Shannon divergence とは，非対称である Kullback-Leibler divergence を対称にした値である．識別子の集合を I，識別子 a，b 間の Kullback-Leibler divergence を $D(a\|b)$，識別子 a，b 間の Jensen-Shannon divergence を $J(a,b)$ とすると，

$$D(a\|b) = \sum_{I'} P(i \mid a) \log \frac{P(i \mid a)}{P(i \mid b)} \tag{1}$$

$$I' = \{i \in I \mid (i \neq a) \cap (i \neq b)\} \tag{2}$$

$$J(a,b) = \frac{1}{2}\left[D\left(a \,\middle\|\, \frac{a+b}{2}\right) + D\left(b \,\middle\|\, \frac{a+b}{2}\right)\right] \tag{3}$$

と表せる．ただし，$P(i \mid a)$ については次のようにしている．

$$P(i \mid a) = \frac{\text{識別子 } i, a \text{ が共起するモジュールの数}}{\text{識別子 } a \text{ が出現するモジュールの数}} \tag{4}$$

識別子間の距離を求めた後，クラスタリングを行うことで共起の分布が類似した識別子をグループ化する．グループ化された識別子を同義語として定義する．クラスタリングには，全ての識別子がそれぞれ1つのクラスタという状態から始めて，クラスタ間の距離が最小となるクラスタの組から順次結合していく形式を採用した．なお，クラスタ間距離については，2つのクラスタに属する要素間の距離の平均をクラスタ間距離とする群平均法 [13, p. 136] を用いた．アルゴリズムは以下で表される．C はクラスタの集合，$distance(c_\alpha, c_\beta)$ はクラスタ間距離を表している．

（1） 初期状態 $C = \{c_\alpha \mid 1 \leqq \alpha \leqq |I|\}, c_\alpha = \{i_\alpha\}$

（2） $\forall\alpha\forall\beta\ distance(c_p, c_q) \leqq distance(c_\alpha, c_\beta)$ となる p, q を探索

（3） $c_p = c_p \cup c_q$，C から c_q を取り除く

（4） これを $|C| = 1$ となる，もしくは終了条件 $\forall\alpha\forall\beta\ s \geqq distance(c_\alpha, c_\beta)$，($s$ は与えられた閾値) を満たすまで繰り返す．

3.2.3 特徴的な箇所の抽出

モジュールごとに特徴語を求め，特徴語を含む文を特徴的な箇所とする．具体的には，まず対象のソースコード集合固有の値として基準値を求め，モジュールごとに基準値の正規化を行い閾値を設定する．閾値を超える出現回数を持つ識別子をそのモジュールの特徴語と定義する．重要語を含む文は，ソースコード中で特徴語を含む行とした．基準値，閾値の定義は以下のように設定した．

$$\text{基準値} = \frac{\text{識別子の出現回数の総和}}{\text{各モジュールで出現した識別子の種類の総和}} \tag{5}$$

$$\text{平均出現回数} = \frac{\text{識別子の出現回数の総和}}{\text{ソースコード集合中のモジュールの数}} \tag{6}$$

$$\text{閾値} = \text{基準値} \times \frac{\text{対象モジュールの識別子の総出現回数}}{\text{平均出現回数}} \tag{7}$$

3.3 照 合 部

3.3.1 構文情報の取得

ユーザが入力として与えたコード片から特徴的な箇所を検出し，構文情報を取得する．特徴的な箇所の検出は，入力コード片を1つのモジュールと見なして抽出部で述べた方法で行う．入力コード片の特徴的な箇所において，出現している特徴語の集合と文の種類の組を構文情報として定義する．文の種類は以下のように分類した．

- 変数宣言などを表す宣言文
- 条件文，繰り返し文などの条件節
- break 文，return 文を表す補助制御文
- 上記の3つのどれにも当てはまらない文

3.3.2 特徴的な箇所の構文の比較

入力コード片が各モジュールの特徴的な箇所と構文情報が対応するか比較を行う．構文情報の比較の条件については次のように設定した．

手順1 入力コード片側の構文情報を { 特徴語の集合 A, 文の種類 A}，比較対象の構文情報を { 特徴語の集合 B, 文の種類 B} とする

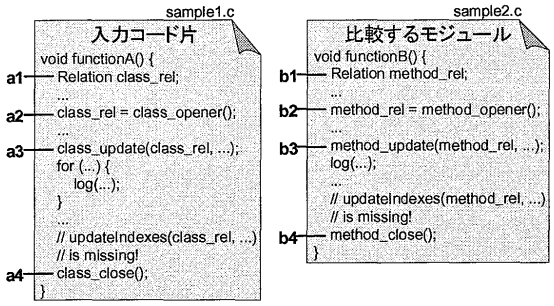
手順2 特徴語の判定を行う

（1） 特徴語の集合 A と 特徴語の集合 B について，特徴語を同義語のクラスタに変換する．

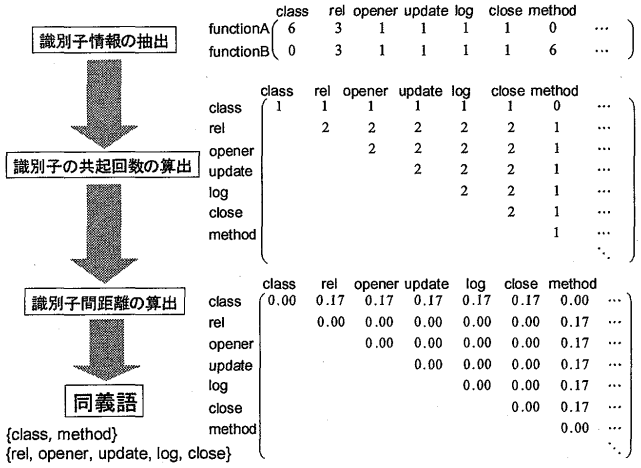
（2） 特徴語の集合 A 中の同義語クラスタすべてが，特徴語の集合 B に存在する場合，特徴語の集合 A は 特徴語の集合 B と対応した判定する

手順3 特徴語が対応し，文の種類が一致した場合，入力コード片側の構文情報が比較対象の構文情報と対応したと判定する

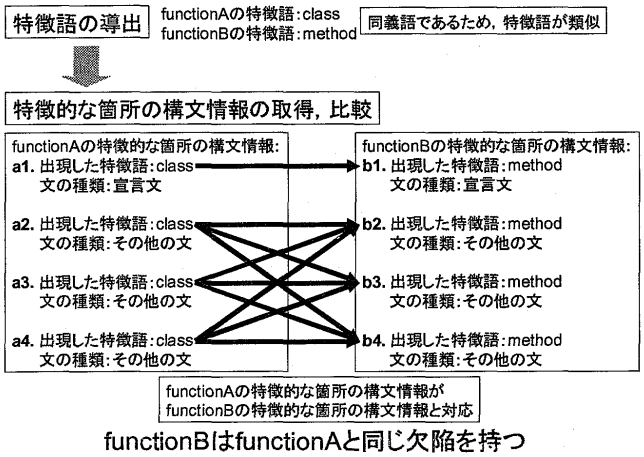
入力コード片の特徴的な箇所の構文情報すべてが，モジュール中の特徴的な箇所のいずれかの構文情報と対応する場合，そのモジュールを入力コード片と同じ欠陥を持つモジュールとする．比較の計算時間コストを削減するため，入力コード片の特



(a) 入力コード片と比較対象のモジュール



(b) 同義語の導出



(c) 構文情報の比較

図 3　提案手法の流れ

徴語，または同義語が，特徴語となっているモジュールのみを比較対象とした．

3.4 例　　題

提案手法の流れを例題を用いて説明する．図 3 は，入力コード片である関数 functionA と比較対象のモジュールである関数 functionB について，特徴的な箇所の構文を比較する流れを示している．

図 3(a) では，関数 functionA，関数 functionB のコードの一部を表している．これらの関数では，opener，update，close という関数を順に呼び出しているが，update の後に updateIndexes という関数呼び出しが欠落しているという欠陥が存在している．

図 3(b) では，同義語を求めるまでの流れを表している．識別子情報の抽出では，関数ごとに識別子が何回出現しているかを計算し，行列を作成する．なお，図中の行列は図 3(a) で表示されている部分に対応する箇所を示している．次に，識別子間の距離を求める際に識別子の共起回数が必要となるので，行列の各要素が識別子 2 つの共起回数を表す共起行列を作成する．共起回数は，2 つの識別子が同じ関数中に共に出現している関数の数とした．なお，行と列が表す識別子が同じ要素の場合は，その識別子が出現している関数の数となる．識別子の共起行列を求めた後，Jensen-Shannon divergence を用いて，識別子間の距離を求める．そして，クラスタリングを行うことで同義語を求める．例題の場合では，クラスタリングの閾値を 0.00 としたときの同義語を表している．

図 3(c) では，特徴的な箇所の構文情報を比較する流れを表している．まず，関数ごとに特徴語を抽出する．例題では，functionA の特徴語が class，functionB の特徴語が method であったとしている．このとき，class と method は同義語となっているため，functionA の特徴語が functionB の特徴語と対応していると判定される．次に特徴的な箇所の構文情報を抽出する．functionA の特徴的な箇所は a1，a2，a3，a4，functionB の特徴的な箇所は b1，b2，b3，b4 となっている．構文情報の比較は，特徴的な箇所それぞれについて行う．例題の場合では，a1 が b1 と構文情報が対応し，a2 が b2，b3，b4 と構文情報が対応する．同様に a3，a4 も b2，b3，b4 と構文情報が対応している．比較の結果，入力コード片である functionA の重要な箇所 a1，a2，a3，a4 すべてが，構文情報が対応する組を持っているので functionB は入力コード片と同じ欠陥があると判定される．

4. 適 用 実 験

ツールの評価を行うため，C 言語のソフトウェアについてツールを作成し，実験を行った．実験対象は，オープンソースソフトウェアの日本語入力システム “かんな” [11] と我々の研究グループで開発しているソフトウェア部品検索システム SPARS-J [8] を用いた．なお，実験対象が C 言語のソフトウェアであるため，検出を行う単位は関数単位とした．実験対象のデータは表 1 となっている．欠陥の数とは，修正が行われた箇所の数を表している．以降，それぞれのソフトウェアについて行った実験について説明する．

4.1 実験の概要

かんなはクライアント・サーバ方式の日本語入力システムであり，かな漢字変換をサーバと接続して実行している．かんなでは，バージョン 3.6 とバージョン 3.6p1 間でバッファのオーバーフローを検知するコードが追加された [9]．修正された箇

```
static
ProcWideReq2(buf)
BYTE *buf ;
{
   ir_debug( Dmsg(10, "ProcWideReq2 start!!¥n") );
   if (Request.type2.datalen != SIZEOFSHORT)
            return( -1 );                          追加されたコード
   buf += HEADER_SIZE; Request.type2.context = S2TOS(buf);
   ir_debug( Dmsg(10, "req->context =%d¥n", Request.type2.context) );

   return( 0 ) ;
}
```

図 4　かんなの修正事例

```
key.data = package;                    型キャストの追加
key.size = (u_int32_t)(size + 1);

/* Acquire a cursor for the database. */
dbcp = get_cursor(&DBlist[PACKAGEDB]);

/* Curosr set */
if ((ret = dbcp->c_get(dbcp, &key, &data, DB_GET_BOTH)) != 0) {
   if (ret == DB_NOTFOUND) {
      cursor_close(dbcp);
      spars_free(package);
      return;
   }
   .
   .
   .
```

図 5　SPARS-J の修正事例

所の例は図 4 である．バッファのオーバーフローを検知するコードが追加された箇所は，サーバの処理を行っているコードであった．サーバ関連の欠陥に対する修正と考え，サーバ関連の処理を行っているディレクトリ下のファイルを実験対象とした．図 4 で示した修正は 19 個の関数で行われており，これらの関数を正解集合とする．修正前バージョンから修正が行われる箇所と前後の 2 行を入力コード片とした．修正は 21 箇所存在していたので，21 個の入力コード片に対して実験を行った．

SPARS-J では，複数の関数において型キャストの追加が同時に行われる修正があった．修正された箇所の例は図 5 である．修正が行われた関数は様々なディレクトリに存在していたため，ソフトウェア全体を実験対象とした．実験に用いたソースコード集合は，修正が行われる前の最新の状態を用いた．図 5 で示した修正は 48 個の関数で行われており，これらの関数を正解集合とする．かんなの場合と同様に，修正前のソースコードから修正が行われる箇所と前後の 2 行を入力コード片とした．修正は 75 箇所存在していたので，75 個の入力コード片に対して実験を行った．

なお，かんな，SPARS-J で実験を行うに当たって同義語を求める際のクラスタリングの閾値を以下のように設定した．

（1） 初期状態のクラスタ間距離の最大値を求める
（2） 求めた値の x% をクラスタリングの閾値とする
（3） x の値を 0 から 100 まで 10 刻みで実験を行う

4.2 評価基準

実験結果の評価は，情報検索の分野でよく用いられている適合率，再現率 [10, pp. 17–19] で評価する．適合率とは，検索された集合の中で適合するものの割合であり，検索ノイズの少なさを示す尺度である．再現率とは，検索対象中の適合する集合の中で実際に検索されたものの割合であり，検索漏れの少なさを示す尺度である．適合率，再現率はトレードオフの関係となっている．それぞれの定義は以下のように定義した．

$$\text{適合率} = \frac{\text{ツールが検出した正解集合の数}}{\text{ツールが検出した関数の数}} \times 100(\%) \quad (8)$$

表 1　実験対象のデータ

ソフトウェア	総行数	ファイル数	欠陥の数	欠陥を含む関数の数
かんな	7,613 行	7 ファイル	21 個	19 個
SPARS-J	35,744 行	171 ファイル	75 個	48 個

$$\text{再現率} = \frac{\text{ツールが検出した正解集合の数}}{\text{正解集合の総数}} \times 100(\%) \quad (9)$$

4.3 実験結果

かんなの実験結果は表 2，SPARS-J の実験結果は表 3 となった．表は，クラスタリングの閾値として，初期状態のクラスタ間距離の最大値に対する割合を 0% から 100% まで 10% 刻みで変化させた結果を表している．クラスタリングの閾値が高いほど，より多くの識別子が 1 つのクラスタ中に存在するようになる．つまり，ソースコードを比較する際に識別子の違いが小さくなる．かんな，SPARS-J 共に，複数の入力コード片を用意して実験を行ったが，表では適合率，再現率の平均値を求めた．かんな，SPARS-J 共に，全体的に適合率は低い値となっているが，再現率は設定した割合が 20% 以降で高い値を取る結果となった．また，閾値が高くなってくると適合率，再現率に変化があまり見られない結果となっている．

4.4 考　　察

(1) 実験結果からわかること　かんな，SPARS-J の実験結果は，共にクラスタリングの閾値がある閾値以降高い再現率となっていた．手法が利用される状況を考えると，欠陥を漏れなく検出することが要求されるため，高い再現率は重要であり，意義ある結果と言える．また，設定した割合が 10% から 20% の間で再現率が大きく変化していることから，検査を行う際に目安となる閾値を設定できると考えられる．また，クラスタリングの閾値が高い場合，適合率，再現率にあまり変化が見られなかった．検査する閾値の範囲を絞り込むことで，検査の効率を上げることが期待できる．しかし，適合率については全体的に低い値となっていた．これは欠陥が含まれていない場合でも，構文情報が同じであれば出力と判定されてしまうためである．この問題を解決するには，ツールの出力結果を分類する必要がある．分類の種類は以下が考えられる．

- 入力コード片と同じ欠陥を含むモジュール
- 入力コード片と同じ処理内容であるが欠陥を含まないモジュール
- 入力コード片と異なる処理内容のモジュール

これらの分類のうち，入力コード片と同じ処理内容であるが欠陥を含まないモジュールについては，入力コード片と特徴的な箇所が同じモジュールを検出できたと言える．入力コード片と同じ特徴的な箇所を検出するという意味では，正しく検出できていると見なすことができる．また，特徴語の閾値について評

価が行われていないため，特徴語の閾値を変化させて実験結果を調べる必要がある．C 言語のソフトウェアを対象としたが，他言語のソフトウェアでも同じような結果となるか調べる必要がある．

(2) Libra との比較　Libra でもかんなの同じ修正事例に対して適用実験を行っている [9]．Libra の適用実験では，入力として与えたコード片は修正が行われる箇所のうち 1 つとなっていた．提案手法の実験結果は，修正が行われる箇所すべてをそれぞれ入力コード片として実験した結果の平均値を表しており，Libra と比較すると適合率は劣るが，再現率は平均して高いと言える．

(3) grep との比較　Libra の適用実験で grep との比較が行われている．提案手法と Libra の適用実験における grep の結果を比較すると，クラスタリングの閾値が高くなると提案手法がの再現率が若干上回る結果となった．grep による検索では，検索キーワードの設定が結果に大きく左右されるという問題点がある．提案手法では，コード片を入力と与えることができるため検出結果が grep に比べ安定すると考えられる．

5. むすび

本論文では，識別子間の共起関係に着目して類似したコードを検索する手法を提案した．この手法は，ソースコードから重要な箇所を抽出し，構文の情報を比較することで類似したコードを検出する．また，欠陥の検出に提案手法を適用し，実験を行った．C 言語のソフトウェアを対象として実験を行ったところ，再現率はある閾値を境に高い値となった．欠陥を漏れなく検出することが目的であるため，高い再現率は意義のある結果と言える．一方で，適合率については全体的には低い結果となった．欠陥が含まれていなくても，構文の情報が同じであれば欠陥が含まれていると判定してしまうため，手法の出力結果を分類することが必要であると考えられる．また，他言語のソフトウェアに対しても実験を行い，手法の有効性を確かめたいと考えている．

表 2　かんなの実験結果

設定した割合	適合率の平均値	再現率の平均値
0%	17.03%	5.51%
10%	89.86%	18.55%
20%	39.45%	81.95%
30%	21.09%	100.00%
40%	9.64%	100.00%
50%	9.64%	100.00%
60%	9.55%	100.00%
70%	9.55%	100.00%
80%	9.55%	100.00%
90%	9.55%	100.00%
100%	9.55%	100.00%

表 3　SPARS-J の実験結果

設定した割合	適合率の平均値	再現率の平均値
0%	8.05%	0.36%
10%	22.67%	18.28%
20%	7.01%	87.19%
30%	7.33%	88.31%
40%	7.33%	88.31%
50%	7.33%	88.31%
60%	7.23%	89.53%
70%	7.23%	89.53%
80%	7.23%	89.53%
90%	7.23%	89.53%
100%	7.23%	89.53%

謝辞　本研究を進めるにあたり，貴重なコメントをくださいました株式会社富士通研究所 松尾昭彦氏，小林健一氏，前田芳晴氏，ならびに株式会社富士通東北システムズ 須藤茂雄氏に深く感謝いたします．

文　献

[1] I. D. Baxter, A. Yahin, L. Moura, M. S. Anna, and L.Bier. Clone detection using abstract syntax trees. In *Proc. of ICSM '98*, pages pp.368–377, 1998.

[2] B. S. Baker. Finding clones with Dup: Analysis of an experiment. *IEEE Transactions on Software Engineering*, Vol.33(No.9):pp.608–621, 2007.

[3] Gnu grep. `http://www.gnu.org/software/grep/`.

[4] IEEE Std 1219: Standard for software maintenance, 1997.

[5] S. Kullback. *Information theory and statistics*. John Wiley and Sons, 1959.

[6] T. Kamiya, S. Kusumoto, and K. Inoue. CCFinder: A multi-linguistic token-based code clone detection system for large scale source code. *IEEE Transactions on Software Engineering*, Vol.28(No.7):pp.654–670, 2002.

[7] I. Dagan, L. Lee, and F. C. N. Pereira. Similarity-based models of word cooccurrence probabilities. *Machine Learning*, Vol.34(No.1-3):pp.43–69, 1999.

[8] 横森 励士, 梅森 文彰, 西 秀雄, 山本 哲男, 松下 誠, 楠本 真二, 井上 克郎. Java ソフトウェア部品検索システム SPARS-J. 電子情報通信学会論文誌 **D-I**, Vol.J-87-D-I(No.12):pp.1060–1068, 2004.

[9] 泉田 聡介, 植田 泰士, 神谷 年洋, 楠本 真二, 井上 克郎. ソフトウェア保守のための類似コード検査ツール. 電子情報通信学会論文誌, Vol.J-86-D-I(No.12):pp.906–908, 2003.

[10] 北 研二, 津田 和彦, 獅々堀 正幹. 情報検索アルゴリズム. 共立出版, 2002.

[11] 日本語入力システム "かんな". `http://canna.sourceforge.jp`.

[12] 松尾 豊, 石塚 満. 語の共起の統計情報に基づく文書からのキーワード抽出アルゴリズム. 人工知能学会論文誌, Vol.17(No.3):pp.217–223, 2002.

[13] 齋藤 堯幸, 宿久 洋. 関連性データの解析法–多次元尺度構成法とクラスター分析法. 共立出版, 2006.