Vol.2014-SE-183 No.14
2014/3/20

関心度に基づいたソースコード推薦システム

村上 直也[1,a)]　増原 英彦[2,b)]　青谷 知幸[2,c)]

概要：コード推薦システムの役割は，開発者のコード記述の補助をする用例を推薦することである．中でも大規模コードリポジトリを利用したコード推薦システムは、開発者が編集中であるコードの周辺部分に類似したプログラムをリポジトリから検索し，類似部分の後続行を開発者が次に書くと予測されるコードとして提示する．検索の際に編集中であるコードの周辺部分だけでなくその背景情報—周辺部分が果たす役割に関連がある他箇所のコード—も用いれば，より精度の高い予測が出来ると我々は予想したが，背景情報をソースコードのみから特定することは容易でない．そこで我々は開発者の統合開発環境の編集操作から，現在関心を持っているメソッド集合を特定しそれを背景情報とした．そして Eclipse 上で稼働するコード推薦システム Selene を拡張し，直近に閲覧・編集したコード中の単語を抽出・重み付けして検索・コード推薦を行うシステムを開発した．

1. はじめに

コード推薦システムの役割は，開発者が次に書くであろうコード断片を提示することである．例として図 1 のような状況を仮定する．その状況下でもしコード推薦システムが以下のコードを提示すれば，開発者はその推薦コードから `BufferedReader` クラスの `readLine` メソッドを使用すること，返値が `null` であることによって終了を検知することを知ることが出来る．

```
BufferedReader br =
  new BufferedReader(
    new InputStreamReader(is));
  while((tmp=br.readLine()) != null){
```

本研究は，リポジトリに対する検索のクエリの作成方法を改良することで，コード推薦システムの推薦結果の質の向上を図る．我々が開発した Selene[10] を基にコード推薦システムを作成する．Selene は統合開発環境 Eclipse のプラグインとして動作し，オープンソースのプログラムが格納されたデータベース（以降では単にリポジトリと呼ぶ）から開発者が編集中のプログラムに類似したコード断片を検索・提示する．本研究ではクエリの作成の際に，これから記述しようとしている内容やそれが使われている場所も用いることでより良いクエリを作成する．

開発者はテキストエディタを作ろうとしている．そのためにテキストエディタのメニューバーを表す `OpenFileMB` クラス (Listing 1) を作成した．そしてその次に，選択したファイルの文字列をエディタペインにセットする `actionPerformed` メソッドを，`OpenFileMB` クラス内に定義した．そして `actionPerformed` メソッドでファイルから文字列を読み込むために，`FileUtils` クラス (Listing 2) の `readFile` メソッドを作成し始めた．しかし，開発者は行単位で文字列を読み込むために使用するクラス・メソッドの名前や使い方が思い出せないでいる．

図 1　開発者のコード記述状況の例
Fig. 1　An example situation

第 2 節では既存のコード推薦システムとその問題点を述べる．第 3 節では提案手法の概要を説明する．第 4 節ではシステムで用いるモデルの調整方法を，第 5 節ではシステムの評価を行う．第 6 節では関連研究と本研究との比較を行う．第 7 節ではまとめを行う．

2. 既存のコード推薦システムとその問題点

2.1 コード推薦システム Selene

本研究のシステムの基となる Selene の仕組みを説明する．Selene の内部は図 2 に示す様に以下の 5 段階の処理からなる．

編集中ソースファイルの取得

フロントエンドは Eclipse のプラグインとなっていて，

[1] 東京大学大学院総合文化研究科広域科学専攻
Dept. of General Systems Studies, Graduate School of Arts and Sciences, the University of Tokyo
[2] 東京工業大学大学院情報理工学研究科 数理・計算科学専攻
Dept. of Mathematical and Computing Sciences, Graduate School of Information Science and Engineering, Tokyo Institute of Technology
[a)] murakami@graco.c.u-tokyo.ac.jp
[b)] masuhara@acm.org
[c)] aotani@is.titech.ac.jp

Listing 1 開発者のコード記述状況：最初に書いたクラス

```
class OpenFileMB extends JMenuBar {
  Application app;
  JFileChooser chser;
  void actionPerformed(ActionEvent eve) {
    edPane.setContentType("text/html");
    File file = chser.getSelectedFile();
    String t = FileUtils.readFile(file);
    edPane.setText(t.trim());
  }
}
```

Listing 2 開発者のコード記述状況：現在編集中のクラス

```
class FileUtils {
  Object readObject(File file){
    FileInputStream inFile=new
        FileInputStream(file);
    ObjectInputStream in=
      new ObjectInputStream(
        new BufferedInputStream(inFile));
    Object obj=in.readObject();
    inFile.close();
    in.close();
    return obj;
  }
  String readFile(File file){
    InputStream is =
      new FileInputStream(file);(カーソル)
```

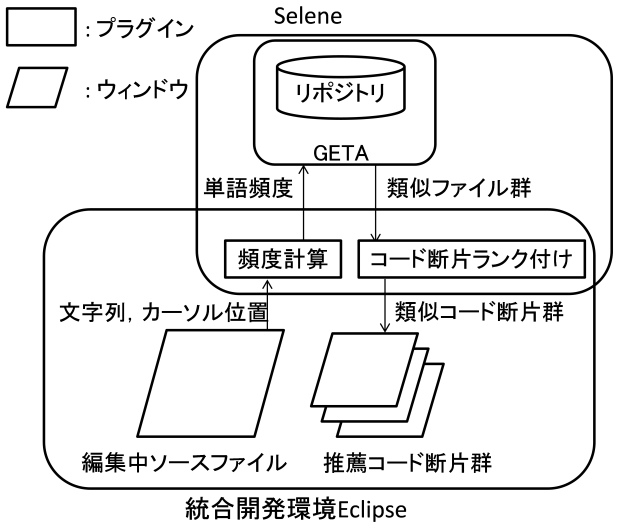

図 2 Selene の機構
Fig. 2 Internal Structure of Selene

エディタ上でカーソルが存在するソースファイル中文字列とカーソル行の位置を取得する．取得は一定文字数の入力，カーソルの行移動のたびに行われる．

編集中ソースファイルの単語頻度情報算出

編集中ソースファイル中の単語 [*1] の重み付き頻度を算出する．単語の重みつき頻度は，各出現位置についてカーソル行からの近さを求めその和をとったものである．これによりカーソル行の周辺を重視している．

類似ソースファイルの検索

単語頻度情報をクエリとして，類似した単語頻度情報を持つソースファイル群をリポジトリから一定数検索する．検索には GETA[9], [11] を利用している．リポジトリは約 200 万ファイルのオープンソースのプログラムで構成されている UCI Source Code Data Sets 18k[4] を使用している．

類似コード断片のランク付け

検索結果上位の類似ソースファイルについて，編集中ソースファイルのカーソル行周辺に類似した位置を特定する．各類似ソースファイルを約 10 行ずつのコード断片に分割し，編集中ソースコードのカーソル行周辺と最も類似している断片を選択する．類似度の判定は，単語並びの最長共通部分列の長さで行っている．

推薦結果の表示

類似箇所を特定した検索結果のソースファイルを推薦結果として Eclipse のウィンドウ上に表示する．表示の際に，類似コード断片とその後の部分を中心に表示する．

2.2 既存のコード推薦システムの問題点と解決策

Selene を含む既存のコード推薦システムは，開発者が現在記述している一連の処理が複数のファイルに跨がる場合，その一連の処理とは無関係推薦をすることがある．そのことを図 1 の状況を仮定して，Selene を例にして説明する．図 1 は `FileUtils` クラスの `readFile` というメソッドの作成中に「ファイルの内容を文字列として読み込む方法を知りたくなった」状況である．このときカーソルは `FileUtils.java` 上に置かれているので, Selene はカーソル近傍の `readObject` メソッドの単語 (`ObjectInputStream` や `readObject` 等) をリポジトリへの検索に用いる．よってファイルからの文字列読込とは無関係な「ファイルからオブジェクトをデシリアライズする」コード (Listing 3) を Selene は推薦する可能性がある．

開発者が編集中 n のメソッドの背景にある情報をシステムが把握していれば，開発者が記述する一連の処理が複数のモジュールに跨がっている場合でも対応出来る．`readFile` メソッドの背景にある情報とは，`readFile` メソッドの呼出元でかつ `readFile` メソッドの直前に編集していた，`actionPerformed` メソッドが相当する．Selene は

[*1] 現在の Selene では連結した英字を単語としていて，大文字小文字を区別しない．

Listing 3 オブジェクトを読み込むコード [*1]

```
String readFile(File filename){
  inputStream =
    new ObjectInputStream(
  new FileInputStream(filename));
  Object obj = null;
  while ((obj = inputStream.readObject())
                                    != null){
    if (obj instanceof Property){
      list.add(
        ((Property)obj).getSettings());
    }
  }
}
```

Listing 4 行ごとに文字列を読み込むコード [*2]

```
InputStream fis =
  new FileInputStream(selectedFile);
InputStreamReader in =
  new InputStreamReader(fis);
BufferedReader br =
  new BufferedReader(in);
res.setContentType(
  "text/html;charset=UTF-8");
while((tmp=br.readLine()) != null){
  ...tmp.trim()...
      略
}
```

actionPerformed メソッドの setContentType や trim といった単語をリポジトリへの検索に用いるため，それらを含む「ファイルから行単位で文字列を読み込む」コード (Listing 4) を推薦することが可能である．

3. 関心度に基づくコード推薦

我々は，上述の問題は**背景情報**の不足に起因すると考え，これを解決するために統合開発環境 Eclipse のタスク管理システム Mylyn[3] で導入された *Degree Of Interest*(以降では**関心度**と呼ぶ) を用いた推薦手法を提案する．ここでコードの背景情報とは，そのコードが実現すべき機能 (数行から 1 メソッド程度で実現できる処理のかたまり) と，他の機能との関係を意味する．その機能がエディタの「Open File」機能，つまりファイル選択ダイアログによって選ばれたファイルの内容を編集可能領域に読み込むメニュー項目から呼び出されることなどに相当する．

このような背景情報を得る方法としプログラム解析なども考えられるが，我々は，統合開発環境の操作履歴を基にした関心度を用いることを提案する．その理由には以下が挙げられる．

- 編集中コードのように不完全なプログラムでも関心度は計測可能である．一方プログラム解析で扱うのは困難であったり，精度が大きく低下したりする．
- 1 つのコードにたくさんの要素が関係しているような場合でも関心度を用いれば，開発者が興味を持っている要素を重視すること出来る．例えば現在編集中のメソッドに多くの呼び出し元があり，その 1 つに関連する処理を記述している場合などである．
- 例えば編集中のコードと開発者が興味を持っている要素の関係が直接的でない場合でも，関心度を用いればその要素を把握出来る．一方プログラム解析ではそのような関係を辿って適切な要素を重視することは難しい．

3.1 関心度の計測と利用

提案する推薦手法は，Selene を拡張して (1) 統合開発環境操作の監視，(2) 背景情報の抽出，(3) 関心度による単語の重み付けを行わせることで実現した．以下でそれぞれについて説明する．

3.1.1 統合開発環境操作の監視

最近閲覧・編集したコードがどれかを捕捉するために，統合開発環境の操作を監視し，後述する関心度の算出部分に操作情報を逐次に渡す．操作には文字の入力と削除，メソッド内のカーソル位置の移動などが含まれ，それらは Eclipse による操作感知の仕様に基づき一定間隔で 1 つのイベントにまとめられる．イベントが保持する情報は，

- 種類
- 対象要素 (クラス，フィールド，メソッド)
- ファイル名

である．イベントの種類は，編集と閲覧による 2 種類を定義する．編集によるイベント (以降では *eve_edit* とする) は，**エディタでのメソッド上のカーソル移動を伴う操作のこと**である．閲覧によるイベント (以降では *eve_view* とする) は，*Eclipse* **のビューやエディターでメソッドをマウスで選択すること**である．

3.1.2 背景情報の抽出

最近閲覧・編集したコードから背景情報を抽出するために，関心度を変更し用いる．本研究における関心度は，**開発中のプロジェクトに含まれるメソッド名と実数の組で表される値で，どれだけ最近に，どれだけ多くの閲覧・編集をメソッドが受けたか**を示している．最近に・多くの閲覧・編集を受けるほど実数値は大きくなる．

[*2] http://www.herongyang.com/Swing/JEditorPane-File-Chooser-Dialog-Box.html に掲載されたソースコードの一部を改変
[*3] http://www.javadb.com/reading-objects-from-file-using-objectinputstream に掲載されたソースコードの一部を改変

3.1.3 関心度による単語の重み付け

背景情報を推薦システムに反映させるために，類似ファイル検索で用いる単語頻度情報を推薦システム使用時の関心度で重み付けする．開発中プロジェクトの全ファイル中の単語について，メソッド内で現れている単語はそのメソッドの関心度で重みを付ける．それによって，最近閲覧・編集したメソッド中の単語ほど検索への関与を大きくする．

3.2 関心度モデル

Mylyn の関心度モデルを本研究のシステムのために変更し用いる．

メソッド m の時刻 t[*4] における関心度 $DOI(m,\ t,\ eve_t)$ を，m，t，時刻 t で起きたイベント eve_t，イベントの得点を求める関数 S，イベント eve_t の対象である要素 $type(eve_t)$，単位時間あたりの関心度の減衰度 c を用いて以下のように定義する．関数 S はイベント eve_t の種類を表す関数 $k(eve_t)$ と eve_edit のスコア α，eve_view のスコア β によって表される．

$$DOI(m,t,eve_t) = \begin{cases} S(m, eve_t) & (t=0) \\ DOI(m,t-1)\times c + S(k_t) & (t\neq 0) \end{cases}$$

$$S(m, eve_t) = \begin{cases} 0 & (type(eve) \neq m) \\ \alpha & (k(eve_t) = eve\_edit) \\ \beta & (k(eve_t) = eve\_view) \end{cases}$$

まず，単位時間経過するごとに関心度は c 倍に減少する．そして発生したイベントに応じてスコアが加算される．イベントのスコアは関数 S で求められ，メソッド m が対象のイベントならばイベントの種類に応じたスコアが加算される..

3.3 単語の重み付け

正規化前の単語重みを関数 $wWgt$ で定義する．$wWgt$ は，単語 w，時刻 t，編集中ソースファイル ef，編集中ソースファイルの関心度 $efDOI$，要素 ele 中の w の出現回数を返す関数 $cnt(w,\ ele)$，開発中プロジェクト中のメソッド定義の集合 $allmtds$，w を含むメソッドの集合を返す関数 $mtds$，時刻 t における関心度の最大値を返す関数 $maxDOI(t)$ から求められる．関数 $mtds$ は，単語 w と開発中プロジェクト中のメソッド定義の集合 $allmtds$，メソッド m に含まれる単語を返す関数 $tokens(m)$ で表される．

[*4] 時刻は他のメソッドに対するイベントも含めて，イベントが発生するごとに進む．ただしイベントの種類で時刻の進み幅は異なる．

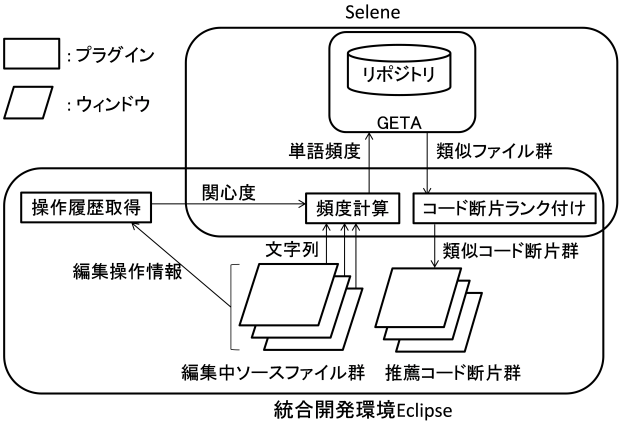

図 3 本研究のシステムの構成
Fig. 3 The structure of our system

$$wWgt(w,\ t,\ ef) = \left(\sum_{m\in mtds(w)} DOI(m,\ t) \times cnt(w,\ m) \right) + efDOI \times cnt(w,\ ef)$$

$$mtds(w) = \{m \mid m \in allmtds,\ w \in tokens(m)\}$$

関数 $wWgt$ では単語 w が含まれる各メソッドと編集中ソースファイルについて，関心度に単語 w の出現回数を掛けることで重みを算出している．編集中ソースファイルは開発者による一定の関心が存在すると考えられるため，関心度を定数 $efDOI$ で与えている．

類似ファイル検索で用いる単語の重みは関数 $stdWgt$ で定義する．関数 stdWgt は単語 w，時刻 t，編集中ソースファイル ef，編集中ソースファイルの関心度 $efDOI$，時刻 t における関心度の最大値を返す関数 $maxDOI(t)$，関数 $wWgt$ から求められる．関数 $maxDOI(t)$ は時刻 t と単語 w と $allmethods$ で表される．

$$stdWgt(w,\ t,\ ef) = \frac{1}{\max(maxDOI(t),\ efDOI)} \times wWgt(w,\ t,\ ef)$$

$$maxDOI(t) = \max_{f\in allmtds} DOI(f,\ t)$$

$wWgt$ が返す単語重みをその時点における関心度の最大値で割ることで，値の範囲を [0, 1] に正規化している．これは Selene におけるカーソル行からの距離による係数の範囲が [0, 1] であることを考慮したためである．

3.4 推薦システム構築

推薦システムの構築方法について説明する．図 3 の様に，推薦システムのコード検索・推薦部分は Selene を，関心度の算出に関する部分は Mylyn を元に作成した．コード検索・推薦部分とは，単語頻度から類似コードを検索し，推薦している部分である．関心度の算出に関する部分では，統合開発環境操作の監視と操作履歴に基づく関心度の算出を行う．

4. 開発履歴を用いたパラメータ調整法

開発履歴を用いて調整する手法を提案し，それを用いて関心度モデルに関するパラメータを調整する．我々が開発した履歴収集用 Eclipse プラグインを複数の開発者に導入して開発をしてもらうことで，開発者の開発履歴を収集する．収集した開発履歴を入力とし関心度を用いたコード推薦を行わせ，推薦結果中の単語に対する再現率 [6] を計算する．そして再現率が極大化するようにパラメータを変化させながら調整する．

調整するパラメータは 3.2 節で紹介した，関心度の減衰度 c，eve_edit のスコア α，編集中ファイルの関心度 $efDOI$ の 3 つである．それらのパラメータの最適値が不明であるため調整が必要となる．なお，eve_view のスコア β は 1-α として eve_edit のスコアとの相対値にする．

4.1 調整法の概要

以下の手順でパラメータを調整する．

(1) 開発者の開発履歴を収集する
(2) 履歴からサンプリングした開発途中の状態をデータセットとし，交差検定のために学習セットとテストセットに分割する
(3) 学習セットから学習データを作成する
(4) 学習データの各学習事例について再現率を測定し，その平均を算出する
(5) 学習データにおける再現率の平均が極大となるよう，パラメータを変化させて (4) を繰り返し行う

ある時点のソースファイル群と関心度の状態，その直後に入力された内容を再現したものを 1 学習事例とする．再現率は各学習事例のソースファイル群と関心度の状態を基にコード推薦を行い，直後の入力内容中の単語が推薦コード中に出現した割合とする

4.2 学習セット作成

学習セットの作成に用いる式や変数を定義する．収集した開発履歴は $H = \langle h_1,\ h_2,\ \ldots,\ h_n \rangle$ と定義する．H は時刻 $t \in T$ に起きたイベント h_t の列である．ただしイベント発生時刻は $T = \langle 1,\ 2,\ \ldots,\ n \rangle$ のように発生順に連続する整数で表す．イベント h_t が保持する情報は，3.1 節で定義したイベントが保持する情報に，h_{t-1} からの差分を追加したものである．

時刻 t におけるプロジェクトの状態 F_t を，ファイル名 n_i とその内容の文字列 c_i の組からなる写像 $F_t = \{n_1 \rightarrow c_1,\ n_2 \rightarrow c_2, \ldots\}$ で表す．

時刻 t_1 から時刻 t_2 までの増分行数 $delta(t_1,\ t_2)$ を各ファイルに追加された行数の和，つまり

$$delta(t_1,\ t_2) = \sum_{f \in dom(F_{t_1} \cap F_{t_2})} |\ inc(F_{t_1}(f),\ F_{t_2}(f))| + \sum_{f \in dom(F_{t_2}) \setminus dom(F_{t_1})} |F_{t_2}(f)|$$

とする．ただし $inc(c_1,\ c_2)$ はテキスト c_1 から c_2 を得る差分のうちの追加テキスト [*5] を，$|c|$ はテキスト c の行数を表わす．

学習セットの作成方法について述べる．学習セット P は，検索開始時刻 q_i と編集経過時刻 a_i の組から成る列 $P =< (q_1,\ a_1),\ (q_2,\ a_2),\ \ldots,\ (q_m,\ a_m) >$ であり，

$$q_i,\ a_i \in T,\ q_i < q_{i+1},\ q_i < a_i$$
$$delta(q_i,\ q_{i+1}) \geq 5,\ delta(q_i,\ a_i) \geq 10$$

を満たす範囲で作成する．つまり，検索開始時刻 q_i と q_{i+1} の間隔は，増分行数の累計が最低 5 行となるイベント間隔となる．そして検索開始時刻 q_i と編集経過時刻 a_i の間隔は，増分行数の累計が最低 10 行となるイベント間隔となる．ただし q_1 は $delta(t_i,\ t_{i+1}) > 0$ を満たす最小の i による t_i とする．初めてテキストが増加するイベントから学習セットを作成し始めると言うことである．

4.3 再現率測定

4.3.1 再現率

テキスト推薦における一般的な再現率の定義は以下の式となる．推薦結果として表示するテキスト (単数または複数) 中の単語集合 S，解答セット中の単語集合 A，単語の重みを表す関数 wgt で表される．

$$recall(S, A) = \frac{\sum_{w \in S \cap A} wgt(w)}{\sum_{w \in A} wgt(w)}$$

解答セット中の単語 (つまり開発者が入力した単語) のうち，推薦コード断片に含まれていたものの割合である．ただし単語の種類によって重み付けを行っている．一般的には wgt が返す重みは，ストップワードや予約語に対しては 0 となる．我々の先行研究では，推薦結果として表示する複数コード断片中の単語集合を S とし，wgt が返す値を予約語は 0，他の単語は定数としている．

本研究では，既出語の重みを忘却度によって補正した再現率を提案し，パラメータ調整に用いる．忘却度は、より最近に編集閲覧したコードに含まれている語は 0 に近く、長い期間 (あるいは一度も) 編集閲覧を受けていない語は 1 に近くする．単語 w の忘却度 $wgt(w)$ を，w を含むメソッド定義 $mtds(w)$ を用いて定義する．

[*5] 差分には追加テキスト削除テキストがある。

Listing 5 再現率の計算アルゴリズム

```
evaluateSystem(H,P){
  for((q_i, a_i): P){
    //q_i 時の DOI を算出
    dois = calcDOI(H, q_i);
    //q_i 時の開発中プロジェクトを復元
    proj = reProj(H, q_i);
    //推薦コード断片を取得
    recomSnips = gRecom(dois, proj);
    //a_i 時の開発中プロジェクトを復元
    neProj = reProj(H, a_i);
    ansFrag= proj と neProj の増分コード断片
    //a_i 時の再現率計算用の DOI を算出
    doi4Rs = calcDOI4R(H, q_i);
    //再現率を計算
    calRecall(ansFrag,recomSnips,doi4Rs);
  }
}
```

$$wgt(w) = \begin{cases} \dfrac{1}{1 + \max\limits_{m \in mtds(w)} \dfrac{DOI'(m,\ t)}{maxDOI'(t)}} & (w \text{ が既出}) \\ \max\limits_{t \in A} wgt(t) & (w \text{ が未出}) \\ 0 & (w \text{ が予約語}) \end{cases}$$

推薦システム使用時に既出の単語は，関心度の逆数を取ることで長い期間編集閲覧を受けていない語の重みが大きくなっている．また，忘却度に用いる関心度のパラメータは固定し，パラメータ調整の影響を受けないようにしている．関心度は単語重みと同様に正規化している．推薦システム使用時に未出の単語は，解答セット中の単語の忘却度の最大値と同一にする．そして予約語は重みを 0 とする．

4.3.2 測定アルゴリズム

再現率の測定アルゴリズムを，Listing 5 を用いて述べる．アルゴリズムの入力は，H と P である．アルゴリズムは q_i と a_i の組に対する m 回の繰り返しとなっているので，繰り返しの内容を 2 段階に分けて以下で述べる．

推薦システム使用状況の復元と推薦結果取得

まず，イベント発生履歴 H から q_i 時の開発中プロジェクトのメソッドの DOI を算出する (`dois`)．次に H から，q_i 時までのイベントの差分を順に当てることで，開発中プロジェクトを復元する (`proj`)．プロジェクト中のソースファイルの名前とそのテキストを復元可能である．そして `dois` と `proj` があれば本研究の推薦システムを使用することが出来るので，推薦されたコード断片群を取得する (`recomSnips`)．

解答セット作成と再現率計算

まず，q_i 時の開発中プロジェクトの復元と同様に，a_i 時の開発中プロジェクトのファイルを復元する (`neProj`)．次に q_i 時と a_i 時の編集中ソースコードの増分を取ることで，解答セット `ansFrag` を作成する．また，イベント発生履歴 H から a_i 時の開発中プロジェクトのメソッドの DOI を算出する (`doi4Rs`)．ただしこの算出に用いる関心度モデルのパラメータは，著者の経験則から定めた固定値とする．そして推薦されたコード断片群 `recomSnips` と解答セット `ansFrag` を関心度で重み付けして比較することで再現率を算出する (`calRecall`)．

4.4 パラメータ探索

パラメータ探索は以下のように再現率が極大となるよう行う．

(1) パラメータを 1 つ選び，残りのパラメータは固定する
  (a) 選んだパラメータの値の候補を等間隔に 11 箇所決定
  (b) 各候補にパラメータを定め，それぞれの学習セットに対する再現率を測定
  (c) 11 箇所の中で最も再現率が高くなったものにパラメータを固定
(2) 次のパラメータを選び，パラメータが 1 周するまで同様に行う
(3) 間隔を半分にし次の周を行う
(4) 前の周との再現率の差が 0.1% になるまで繰り返す

減衰度 c の範囲は [0.8, 1.0]，*eve_edit* のスコア α と編集中ファイルの関心度 *efDOI* の範囲は [0, 1] とする．減衰度が 0.8 以上であるのは，イベントの発生頻度を考慮して高めに設定したためである．各パラメータを順に繰り返し調整することで，パラメータ間の相互作用を考慮した調整方法となっている．調整の結果再現率が収束するということは，パラメータが互いに適したものになっていると考えられるからである．そして前の周との再現率の差が閾値である 0.1% 以下まで小さくなれば，パラメータの設定が再現率がほぼ極大になる場合に収束したということである．

5. 評価

5.1 方法

従来の推薦システムとの比較評価のため，Selene との比較を行う．そのために 4 節でのパラメータ調整の際に，問題セットと解答セットの組の集合を 10 分割し，10 分割交差検定を行う．時系列が隣り合う学習事例は類似性が高いため，それらが学習データとテストデータに分かれると評価の信頼性を損なう．そこで開発履歴をリストにし，それぞれについて学習データを作成し時系列順に並べる．そして先頭から順に 10 分割し，交差検定を行う．Selene もテストデータセットに対して用い再現率の平均値を求め比較する．なお，Selene にも多数の設定項目が存在するが，

表 1　学生の情報
Table 1　The information of the students

所属	プログラミング歴	Java 使用歴
学部生	5 年	5 年
学部生	3 年	3 年
大学院生 (修士課程)	5 年	5 年

Selene の設定は我々の以前の研究 [6] での調整結果を使用する．

調整・評価に用いるデータセットについて述べる．筆者と情報系の学生の Eclipse 使用履歴 (図 2) を用いる．学生は 3 名で表 1 の様な経歴を持っている．約 2 週間 4 節で述べたプラグインを使用し履歴を収集した．ただし 1 名のデータセットからは学習データ作成に十分なコード編集量が無かったため除外する．評価に使用した履歴の内容は表 2 のようになる．記述行数は，後に削除した行の記述も含んでいる．イベント総数は 3.1 節で定義したイベントのことである．

5.2　結果

5.2.1　評価結果

10 分割交差検定の結果が表 3 である．行は各学習の結果とテストデータによるテストの結果を表している．2 項目から 4 項目がパラメータの調整結果である．再現率の項はテストデータにおける本研究のシステムの再現率の平均を表している．同様に Selene について求めたものが Selene の再現率の項である．本研究の再現率のテストデータ 10 個における平均は 27.1%，Selene の再現率の平均は 27.3% であった．個別に見ると 4 つのデータで本研究のシステムが Selene を上回ったが，残りは Selene の方が高かった．調整後のパラメータについては，減衰率はどのデータでもほぼ同様の調整結果となったが，*eve_edit* のスコアと編集中ファイルの関心度はデータによって調整結果が大きく異なった．

5.3　議論

本研究のシステムの再現率が一部のテストデータでしか Selene を上回らなかった原因として評価の方法の影響が考えられるまず評価方法で順位を考慮していないことが考えられる．今回の評価では表示する推薦結果 6 つを等価に扱っている．本研究と Selene で推薦結果上位 6 つ内の順位変動があったとしても再現率は変化しない．よって推薦結果の変化が評価結果として顕在化しなかった恐れがある．実際推薦結果第 1 位のみを対象に再現率を測定したところ，10 テストデータのうち 6 つで本研究が Selene を上回り，再現率の平均も本研究が 10.7%，Selene が 10.2% と Selene を上回った．

次に，推薦結果の表示数を含む Selene 自体のパラメータを調整してないことが挙げられる．Selene のパラメータは以前の研究での調整値を用いたが，本研究で Selene に変更を加えたので Selene のパラメータへ影響があると考えられる．1 学習データでの調整に現状では何時間も必要なため関心度のパラメータのみ調整したが，Selene のパラメータも同時に調整することで再現率が向上する可能性がある．

Selene との再現率の差が大きいテスト事例を抽出して考察した結果を述べる．Selene から改善した事例は，関連するメソッド群を順次・同時に編集した場合であった．反対に Selene から改悪した事例は，別の関心事のメソッドの編集に移行した場合であった．

6.　関連研究

編集履歴を使っている推薦システムには，Robbes らの研究 [7] と Reverb[8] がある．Robbes らは，統合開発環境のコード補完候補を統合開発環境の編集履歴から抽出，ランク付けしている．コード補完の候補を最近編集されたメソッドやその中で呼び出されているメソッドにしていて，最近の編集ほどコード補完の上位にランク付けされる．Reverb は，開発者が過去に閲覧した Web ページに掲載されているコード断片の中から，現在作成中のコードに関係するものを推薦する．Web ページの閲覧履歴を推薦に使用していて，本研究と同様に最近の閲覧や頻繁な閲覧ほど重みが大きくなるように検索の重み付けを行っている．

背景情報を何らかの方法で利用しているコード推薦システムには，Strathcona，Code Conjurer，Mishne らの研究がある．Strathcona[1] は，クラス，メソッド，又はフィールドのどれか 1 つを開発者が指定すると，それに関連する情報をもとに推薦を行う．例えばメソッドを指定した場合は，そのメソッド定義内のメソッド呼出の型情報，メソッドが属するクラス名，その親クラスインターフェース名を基に推薦を行う．指定した要素自体とその親要素の情報が背景情報に当たるが，1 つのファイルから得られる情報しか用いていない．よって本研究で提案した手法が活用可能といえる．Code Conjurer[2] はクラス名とメソッドのシグネチャを開発者が指定し，それを検索のクエリとして用いて推薦を行う．クラスが背景情報に相当する．Mishne ら [5] は開発者が API の呼出の並びを記したクエリを記述し，それを元に推薦を行う．API の呼出の並びが背景情報に相当する．どちらのシステムも本研究とは異なり開発者がクエリを作成する手間が存在する．

7.　まとめ

本研究の貢献は以下の点である．

(1) 統合開発環境の編集操作履歴を用いた，背景情報のクエリ作成に対する活用方法を提案した
(2) 複数の開発者のコード編集履歴を用いた，本研究のシステムのパラメータ調整方法を提案した

表 2　開発履歴の内容
Table 2　The contents of development histories

履歴のプロジェクト内容	クラス数	総行数	記述行数	イベント総数
Apache Wicket を用いた Web アプリケーション	17	933	737	2934
Android のバックアップアプリケーション	5	413	31	469
Android のライブ壁紙アプリケーション	3	658	561	1929
外部コマンドを用いたファイル比較プログラム	4	174	127	907
本研究の学習データを作成する Eclipse プラグイン	11	1255	281	2670
ビューを作成する Eclipse プラグイン	5	400	96	706
Swing を用いたブラウザ	4	316	176	1582
数値計算アルゴリズムの実装	5	340	421	1590
合計	54	4490	2430	12787

表 3　10 分割交差検定の結果
Table 3　The result of 10-fold cross-validation

データ番号	減衰度 c	*eve_edit* のスコア α	編集中ファイルの関心度 *efDOI*	再現率	Selene の再現率
1	0.964	0.3625	0.55	21.6%	21.2%
2	0.964	0.3	0.55	27.5%	30.5%
3	0.964	0.525	0.5	25.6%	30.2%
4	0.964	0.55	0.55	18.0%	24.2%
5	0.964	0.525	0.5	25.6%	27.3%
6	0.964	0.35	0.525	28.3%	30.8%
7	0.9675	0.7375	0.8725	30.4%	27.4%
8	0.964	0.7	0.81	26.0%	29.1%
9	0.960	0.5375	0.85	30.0%	26.4%
10	0.959	0.35	0.575	37.8%	26.2%

(1) では，統合開発環境の編集操作履歴から背景情報抽出のための指標として関心度を定めた．そしてリポジトリへの検索のクエリである単語頻度ベクトルに，関心度で重みを付けた．(2) では，複数の開発者のコード編集履歴から学習セットを作成した．解答セットと推薦コード断片の単語種類の一致率を再現率として定義し，学習セットに対する再現率の平均値を推薦システムの性能評価指標とし，それに基づいてパラメータの調整を行った．現状では Selene を上回る結果は得られなかったが，評価方法に検討の余地があるため本研究の有効性はまだ判明していない．また 2.1 節で述べた類似コード断片のランク付けにも履歴を用いることが改良点として考えられる．

参考文献

[1] Holmes, R., Walker, R. J. and Murphy, G. C.: Approximate structural context matching: An approach to recommend relevant examples, *Software Engineering, IEEE Transactions on*, Vol. 32, No. 12, pp. 952–970 (2006).
[2] Hummel, O., Janjic, W. and Atkinson, C.: Code conjurer: Pulling reusable software out of thin air, *Software, IEEE*, Vol. 25, No. 5, pp. 45–52 (2008).
[3] Kersten, M. and Murphy, G. C.: Using task context to improve programmer productivity, *Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering*, ACM, pp. 1–11 (2006).
[4] Lopes, C., Bajracharya, S., Ossher, J. and Baldi, P.: UCI Source Code Data Sets (2010).
[5] Mishne, A., Shoham, S. and Yahav, E.: Typestate-based semantic code search over partial programs, *ACM SIGPLAN Notices*, Vol. 47, No. 10, pp. 997–1016 (2012).
[6] Murakami, N. and Masuhara, H.: Optimizing a search-based code recommendation system, *Recommendation Systems for Software Engineering (RSSE), 2012 Third International Workshop on*, IEEE, pp. 68–72 (2012).
[7] Robbes, R. and Lanza, M.: How program history can improve code completion, *Automated Software Engineering, 2008. ASE 2008. 23rd IEEE/ACM International Conference on*, IEEE, pp. 317–326 (2008).
[8] Sawadsky, N., Murphy, G. C. and Jiresal, R.: Reverb: recommending code-related web pages, *Proceedings of the 2013 International Conference on Software Engineering*, IEEE Press, pp. 812–821 (2013).
[9] Takano, A.: Association computation for information access, *Discovery Science*, Springer, pp. 33–44 (2003).
[10] Watanabe, T. and Masuhara, H.: A spontaneous code recommendation tool based on associative search, *Proceedings of the 3rd International Workshop on Search-Driven Development: Users, Infrastructure, Tools, and Evaluation*, ACM, pp. 17–20 (2011).
[11] 高野明彦，西岡真吾，丹羽芳樹：連想に基づく情報アクセス技術: 汎用連想計算エンジン GETA を用いて，情報の科学と技術， Vol. 54, No. 12, pp. 634–639 (2004).

正誤表

p.4, 3.2

(誤)

$$DOI(m,t,eve_t) = \begin{cases} S(m,eve_t) & (t=0) \\ DOI(m,t-1) \times c + S(k_t) & (t \neq 0) \end{cases}$$

$$S(m,eve_t) = \begin{cases} 0 & (type(eve_t) \neq m) \\ \alpha & (k(eve_t) = eve\_edit) \\ \beta & (k(eve_t) = eve\_view) \end{cases}$$

(正)

$$DOI(m,t,eve_t) = \begin{cases} 0 & (t=0) \\ DOI(m,t-1) \times c + S(m,eve_t) & (t \neq 0) \end{cases}$$

$$S(m,eve_t) = \begin{cases} 0 & (type(eve_t) \neq m) \\ \alpha & (k(eve_t) = eve\_edit) \\ \beta & (k(eve_t) = eve\_view) \end{cases}$$

p.5, 4.2

(誤)

$$\begin{aligned} delta(t_1,\ t_2) &= \sum_{f \in dom(F_{t_1} \cap F_{t_2})} |\ inc(F_{t_1}(f),\ F_{t_2}(f))| \\ &+ \sum_{f \in dom(F_{t_2}) \setminus dom(F_{t_1})} |F_{t_2}(f)| \end{aligned}$$

(正)

$$\begin{aligned} delta(t_1,\ t_2) &= \sum_{f \in dom(F_{t_1}) \cap dom(F_{t_2})} |\ inc(F_{t_1}(f),\ F_{t_2}(f))| \\ &+ \sum_{f \in dom(F_{t_2}) \setminus dom(F_{t_1})} |F_{t_2}(f)| \end{aligned}$$