プログラミング演習課題の評価支援システム

和田 修平† 井上 潮‡

†‡東京電機大学大学院 工学研究科情報通信工学専攻 〒101-8457 東京都千代田区神田錦町 2-2

E-mail: †09kmc41@ms.dendai.ac.jp, ‡inoue@c.dendai.ac.jp

あらまし 近年，プログラミング教育科目が多く取り入れられており，プログラミング演習課題の採点作業が講師にとって大きな負担となっている．採点に際しては解答の盗用に関する問題も存在するため，その確認作業が大きな負担となっている．そこで，採点作業を効率化するとともに，プログラム間の類似度を算出して盗用の発見を助ける評価支援システムを実現した．実際の授業で行った演習課題で提出された学生 126 名のプログラムを用いて評価した結果，本システムで用いた類似度の算出方法が盗用の発見に有効であること，人手による採点に比べて採点に要する時間を大幅に短縮できることを確認できた．

キーワード e-ラーニング，採点支援，類似度算出

1. はじめに

近年，情報技術教育への需要の高まりから，大学の教育課程においてプログラミング教育科目が多く取り入れられている．そうした講義において，学生の理解度の確認のために，指定した動作を満たすプログラムを作成させるという課題がよく用いられているが，その採点作業が講師にとって大きな負担となっている．

プログラミング課題の採点においては，提出されたソースコードの数だけ個別にコンパイル，プログラム実行，複数のテストデータ入力，出力結果の確認，ソースコードの確認，採点という工程が必要であり，多くの時間を要することから自動化や効率化が望まれている．

また，一般的なレポート課題の場合と同様に，解答の盗用に関する問題も存在する．盗用とは，インターネットや友人を通して解答となるソースコードを入手し，少々の偽装を加えて提出するような不正行為である．盗用を発見するためには，提出された全てのソースコードについて比較確認作業を行い，どの程度類似しているかを判断しなければならず，特に受講生の多い課題においては採点者の大きな負担となっている．

そこで，本研究では，提出されたソースコードのコンパイル，プログラムの実行，テストデータ入力の自動化によってこれら採点作業の負担を軽減するとともに，ソースコード間の全ての組み合わせについて類似度を算出し，採点者に提示することによって盗用の発見を助けるシステムの実現を目指す．

2. 既存研究

2.1. プログラミング演習課題の採点システム

熱田らの授業支援システム[1]は，受講者がウェブブラウザの操作によって提出ファイルのアップロードを行う e ラーニングシステムである．採点者はウェブブラウザ上でコンパイル後のプログラム出力結果を確認し，コメントや採点を行うことが可能となっている．

また，松浦による採点ツール[2]は，ローカル環境で実行するソフトウェア形式である．特に採点作業の効率化に重点が置かれており，一つの画面で学生ごとに採点を行うことができるよう工夫されている．

いずれのシステムにおいても，「入力された整数値の累乗を求める」など使用者に入力操作を強いる課題の場合に，手動で数パターンの入力を行い，出力結果を目視で確認しなくてはならないという採点者への負担が存在するという問題がある．

また，前述した課題の盗用の存在が考慮されていない点も改良の余地が残されていると言える．

2.2. 類似度算出手法

二つの文書間の類似度を算出する手法としては，N-gram 法がよく用いられている．N-gram 法は，対象文書を N 文字もしくは N 単語ずつ切り出していき，そうして抽出できた要素の和集合をとる，ベクトル化して距離を求めるなど比較することで類似度を算出する手法である．

N-gram 法は自然言語で書かれた文章を対象として考案された手法ではあるが，英語，日本語など利用する言語を問わないのが特徴であり，プログラミングのソースコードにおいても適用可能であると考えられる．

また，プログラミングのソースコード間の類似度を算出する手法としては他にも，ソースコードから抽出される，行数やオペレータ数，反復処理の回数などを表すメトリクスを用いて比較を行う手法[3]が提案されている．

しかし，この手法はある程度規模の大きなシステム間の類似度を求めることを目的としており，プログラミング演習課題のソースコードには向いていない．これは，ソースコードの行数が少なく，また課題設定が

同じなので必然的にプログラムの構造が似通ったものとなり，メトリクスが有効に働かないと考えられるためである．

3. 研究手法

3.1. システム設計

本研究では，GUI によってプログラミング課題の採点支援をするシステムの開発を行う．対象とするプログラミング言語は，Java ，C，C++言語などで，対象とする課題は標準出力によって出力を得る形態のプログラムである．

システムの形態はウェブアプリケーションではなく，独立したアプリケーションとした．その理由は，類似度の算出は計算量が多く，サーバ全体への負担となってしまう点と，共有フォルダ内への提出，指定アップローダを用いた提出，電子メールでの提出など，既存のシステムをそのまま利用できるようにするためである．

操作方法は，まず提出物が含まれている基準となるフォルダの指定を行う．次に，対象となるプログラミング言語を指定し，入力として与えたい文字列のセットを行い，コンパイル対象のファイル名を正規表現で指定をする．

その後，解析を実行すると指定フォルダ以下の正規表現で与えられた全てのファイルに対してコンパイルを行い，コンパイルエラーが発生しなかった場合はプログラムを実行，標準入力操作を行い，入力に応じた出力結果を記録していく．また同時に，ソースコードに対して類似度の算出を行う．

本システムの概要を図１に示す．

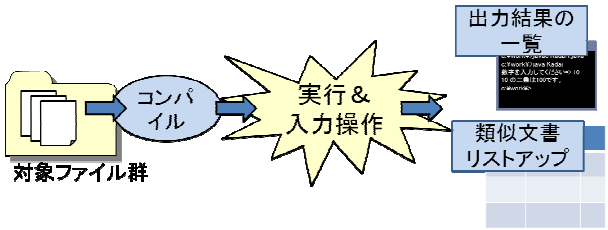


図１　システム概要図

本システムの解析処理部のフローチャートを図２に示す．

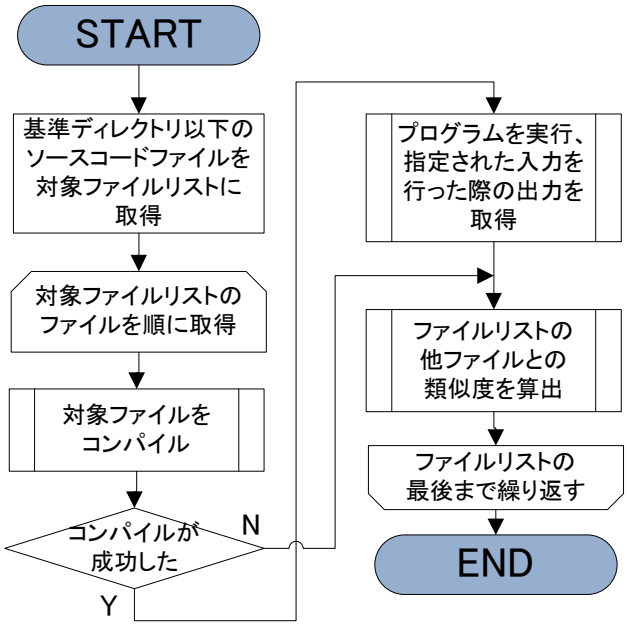


図２　システムのフローチャート

本システムの実行画面を図３に示す．

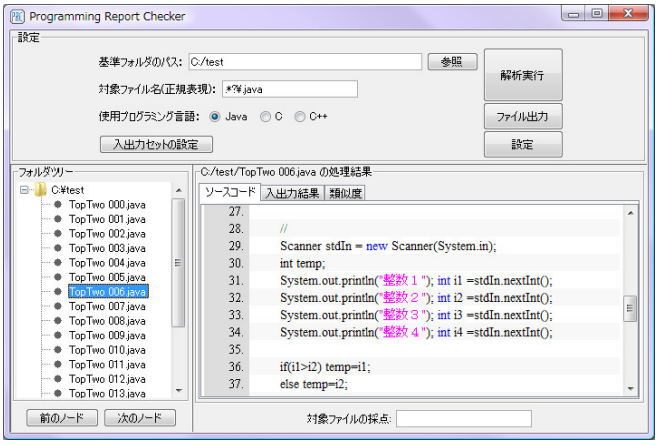


図３　実行画面

標準入力は３パターンまで指定することができ，それぞれの場合の出力結果を取得できる（図４）．

解析結果は，フォルダツリーからファイルを選択することで表示でき，タブの切り替えによって，ソースコード，入力に応じた出力結果，他のソースコードとの類似度が一覧できるようになっている（図５）．類似度は昇順，降順でソートすることができる．また，採点は画面下の採点欄に入力する．

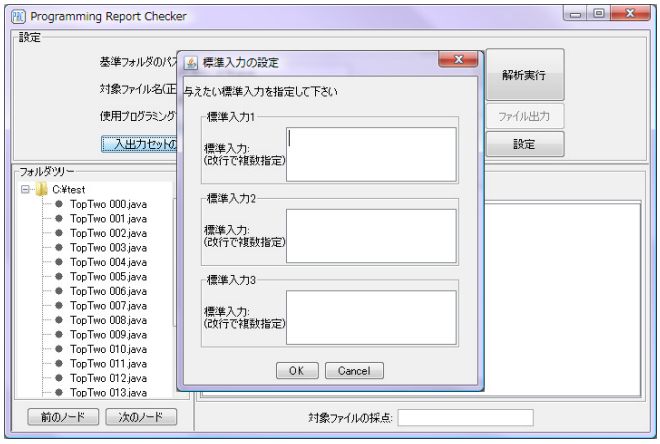


図 4　標準入力の指定

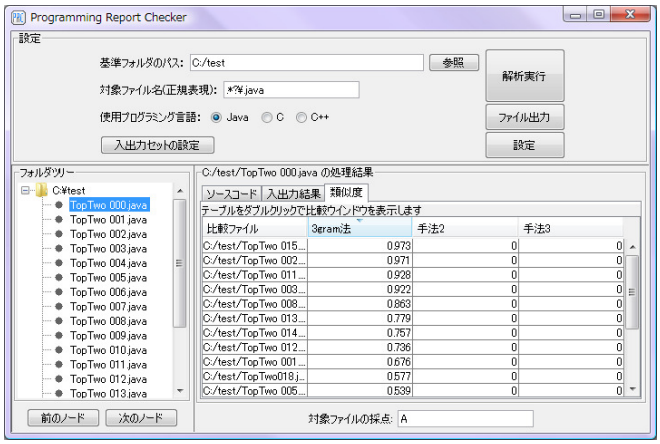


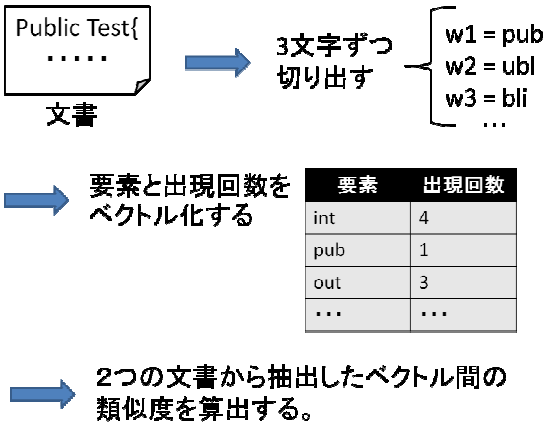
図 5　類似度表示画面

なお，ソースコードのコンパイルならびにプログラムの実行結果の取得は，外部プロセスにコンパイルコマンドや実行コマンドを引き渡し，その標準出力もしくは標準エラー出力を取得することによって行う．また,プログラムへの入力はリダイレクトによって行う．これは，コンパイルコマンドや実行コマンドを変更することにより Java や C など,標準出力によって出力結果を得ることができる様々なプログラミング言語に対応できるようにするためである．

また，出力内容や採点結果は，任意の項目を csv 形式でファイル出力を行うことができる．

3.2. 類似度の算出手法

類似度は，2.2 項で挙げた N-gram 法を用いて，3 文字単位(3-gram)で文書から切り出した要素とその出現回数をベクトル化し，ベクトル間のコサイン類似度を算出することによって求めた．算出手法の概要を図 6 に示す．



要素	出現回数
int	4
pub	1
out	3
・・・	・・・

図 6　類似度算出の概要図

コサイン類似度は，二つの文書から抽出したベクトルをそれぞれ A,B としたとき，次式で求めることができる．

$$\text{コサイン類似度} = \frac{A \cdot B}{|A| \times |B|}$$

コサイン類似度は，0～1.0 の値をとり，値が 1.0 に近いほど類似度が高い．

3.3. 盗用の発見手法

課題の盗用に際しては，変数名の変更をはじめとして，軽微な偽装が行われることが多い．ゆえに盗用の発見のためには，単純に文書間の類似度を求めるだけでは不十分であり，あらかじめ，ソースコードに対して様々な偽装への対策を施しておく必要がある．

プログラミング課題における代表的な偽装として以下が挙げられる．

(1) 冗長な行や空白の追加
(2) 変数名などの書き換え
(3) 処理順序の入れ換え

冗長な行や空白の追加に対しては，ソースコードから空行などの冗長な行や空白を全て削除することによって対応する．

また，変数名などの書き換えに対しては，予めソースコードから変数名などを抽出しておき，登場順に英字一文字に置換していくことによって対応する．

処理順序の入れ替えに対しては，本システムの場合は要素の出現順を問わないという N-gram 法の特長により対応できている．

4. 結果

4.1. 類似度算出結果

「4 つの整数値を読み込み，値の大きい順に 2 つの整数を出力する」という Java 言語のプログラミング課題で提出された 50 行程度のソースコード全 122 ファイル，14762 通りの組み合わせに対して，3.2 項の手法によって類似度を算出した結果の分布の結果を，3.3 項に示した偽装の対策を行った場合，行わなかった場合に分けて図 7 に示す．

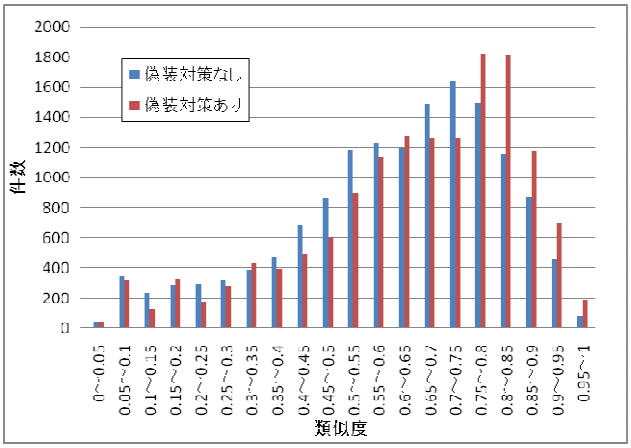


図 7　類似度算出結果の分布

図 7 より，提出されたソースコード群の類似度は，広い範囲に分布している．

また，偽装対策に関しては，偽装対策を施すことにより，偽装の有無に関わらず文字数が減るために，全体的に類似度が増加しており，偽装対策を施したことによって算出される類似度が低くなる組み合わせは存在しなかった．

次に，盗用であると考えられるソースコードを手作業で確認したところ，盗用が疑われるソースコード間の類似度はおおむね 0.95 以上であり，類似度の算出が盗用の発見に有効的であることが確認できた．

なお，どの程度のソースコードの類似を盗用とするかは，課題のケースや採点者の裁量によって大きく異なるため，閾値を定めて盗用を自動判定するといった手法の実現は難しいと考えられる．

4.2. 盗用の抽出結果

4.1 項で用いたファイル群に対して，手作業による盗用の有無の判断を行ったとき，24 ファイル，60 通りの組み合わせに対して盗用の可能性が高いと認められた．このときの盗用の判断基準は，以下に挙げる作業を行った際，同一のソースコードになる場合を盗用と定めた．

- 変数名の変更
- 空行，タブ，空白の除去
- 注釈文の無視
- 出力文字列の無視

これら盗用の組み合わせの類似度を，3.3 項に示した偽装の対策を行った場合，行わなかった場合に分けて図 8 に示す．

また、参考のために偽装の対策を行うことにより類似度が大幅に高くなったプログラムの例を付録に示す。

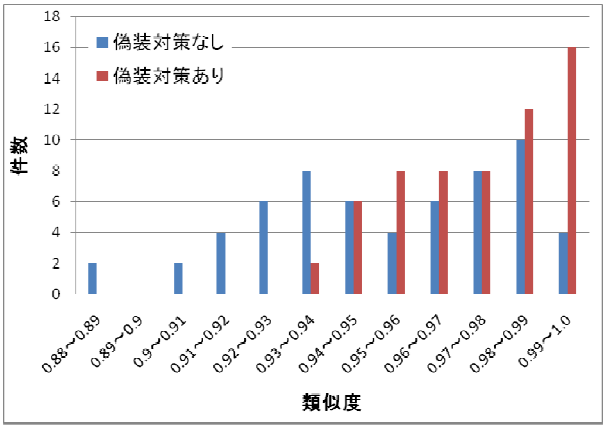


図 8　盗用が疑われる組み合わせの類似度

図 8 より，偽装対策を施すことによって盗用を行っている組み合わせの類似度がより高く算出されていることが確認できる．よって，ソースコードに対する偽装対策が，盗用の発見に有効的に働いていると言える．

4.3. システムの実行速度

本システムは，提出されたファイル数だけコンパイル，実行，テスト入力，そして類似度算出を繰り返し行う必要があるため，解析が完了するまでに相当の処理時間を要する．

4.1 項で用いたファイル群を対象に，解析実行のボタンを押してから解析完了のダイアログが表示されるまでの時間の，対象ファイル数による変化を表 1 に示す．

表 1　ファイル数による処理時間の変化

ファイル数	コンパイル，実行時間[秒]	類似度算出時間[秒]	合計時間[秒]
1	1.0	0.0	1.0
50	80.0	3.0	83.0
100	163.4	12.7	176.0
150	250.5	30.2	280.7
200	330.0	51.6	381.6
250	398.6	72.4	471.0

また，表１の処理時間を縦軸に，対象ファイル数を横軸にとったグラフを図９に示す．

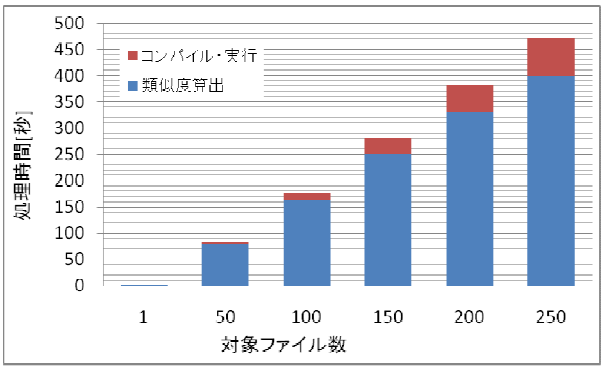


図９　ファイル数による処理時間の変化

表１，図９より，対象が 100 ファイルのときに処理時間はおよそ 3 分程度となっている，手作業によって同じ処理を行うと，１ファイルにつき一連の操作に２分程度要するとすると，合計約 200 分以上の時間がかかる．これを考慮すると，十分効率化が出来ていると言える．

また，コンパイル，実行の処理時間はファイル数に正比例しているが，類似度算出の処理時間はファイル数に対して指数的に増加している．これは類似度の算出は総当りで行わなければならず，対象ファイル数を n としたとき，{n×(n-1)÷2}回の計算が必要であるのが原因である．今回は N-gram 要素に対するコサイン類似度の算出という比較的計算量の少ない手法を用いているため，全体の処理時間に対する類似度算出の占める割合は少ないが，２つの文書間の差分をとるなど，より計算量の大きな類似度算出手法を用いる場合，類似度の算出がボトルネックとなる可能性が高く，注意が必要となる．

5. 今後の課題

今後は，ソースコード中の単語の連続性に着目した手法[4]など，対象となるプログラミング言語の構造を利用した類似度算出手法についても実装し，盗用の発見と計算時間という観点から比較検討したい．

また，盗用における偽装に関しては，本稿で挙げた以外にも，for 文による反復命令を do 文や while 文に書き換える，int 型の整数型変数の宣言を long 型に書き換えるなどの類似命令への置き換えが考えられる．これら偽装に対応するために，対象となるプログラミング言語ごとに類似する命令を定めるなどの手法を検討したい．

他には，入力文字列に対して正解となる出力文字列を正規表現等の利用によって定め，正しい出力が得られているかを判定する，インデントや文法規則の正確さを評価するなど，さまざまな指標をもとに利用者が採点ルールを定義することによって，自動的に採点を行うことができる機能の実装を検討したい．

また，今後は本システムの GUI などの使用感の評価や機能の改善のために，複数人に採点作業を行わせて評価を行っていきたい．

文　献

[1] 熱田智士,松浦佐江子, "Java プログラミング演習向け課題レポート提出・管理機能を付加した授業支援システム", 情報処理学会 FIT2004 情報科学技術レターズ ,Vol.3,pp.359-362,2004

[2] 松浦佐江子, "プログラミングレポート採点支援ツールと課題設計による評価方法の改善", 論文誌ＩＴ活用教育方法研究,Vol.9,No.1,pp.36-40,(社)私立大学情報教育協会,2006

[3] 小堀 一雄,山本 哲男,松下 誠,井上 克郎,"類似度メトリクスを用いた Java ソースコード間類似度測定ツールの試作", 信学技報 ,Vol.103,No.102,pp.7-12, 電気情報通信学会,May,2003.

[4] M. Wise. YAP3: improved detection of similarities in computer program and other texts. In Proc. 27th SCGCSE, pages 130.134, 1996.

付録

偽装対策により類似度が大幅に高くなったプログラムの例

```
int med=0,max=a,sec=b;

if(b >= max){
        max=b;
        sec=a;
}
if(c >= sec){
        med=sec;
        sec=c;
        if(c>=max){
                sec=max;
                max=c;
        }}
if(d >= sec){
        med=sec;
        sec=d;
        if(d>=max){
                sec=max;
                max=d;
        }}
// 出力
System.out.println("最大値は" + max + "です。");
System.out.println("2番目の値は" + sec + "です。");

}
```

```
int med=0,max=a,max2=b;
if(b >= max){
        max=b; max2=a;
}

if(c>=max2){
        med=max2;
        max2=c;
        if(c>=max){
                max2=max; max=c;
        }
}

if(d>=max2){
        med=max2;
        max2=d;
        if(d>=max){
                max2=max; max=d;
        }
}


System.out.println("最も大きな数値 =" +
max );
System.out.println("二番目に大きな数値 ="
+ max2);
}
```

偽装対策後

```
int e=0,f=a,g=b;
if(b >= f){
f=b;
g=a;
}
if(c>=g){
e=g;
g=c;
if(c>=f){
g=f;
f=c;
}}
if(d >= g){
e=g;
g=d;
if(d>=f){
g=f;
f=d;
}}
System.out.println(""+f+"");
System.out.println(""+g+"");
}
```

```
int e=0,f=a,g=b;
if(b>= f){
f=b;g=a;
}
if(c>=g){
e=g;g=c;
if(c>=f){
g=f;f=c;
}
}
if(d>=g){
e=g;
g=d;
if(d>=f){
g=f;f=d;
}
}
System.out.println(""+f );
System.out.println(""+g);
}
```