7H-05

C言語プログラムを対象とした効率の良い盗用検知手法の提案及び実装

東 拓磨† 篠埜 功‡

芝浦工業大学大学院 理工学研究科 電気電子情報工学専攻†‡

1.研究背景と目的

プログラミング教育において、学生が課題に取り組む際に、プログラムの盗用を行う可能性がある。

盗用の発見は採点者にとって時間がかかる作業であり、分担してプログラムを採点することもある。このため盗用の発見を支援する環境が望まれる。プログラムの盗用検知には様々な手法があるが、コードクローンおよび類似度に基づいた手法に分けられる。これまでプログラムの類似度は、字句および構文の情報を用いた方法で算出されていた。

コードクローンで盗用検知を行う補助として主にレーベンシュタイン法が用いられたり[1]、N-gram法が用いられたりしている[2]。またスタンフォード大学のAlex Aiken氏が制作したMoss (a Measure Of Software Similarity)[3]も盗用検知システムの一つとして挙げられる。

本研究では、従来文章の類似度判定に使用されていたN-gram法とレーベンシュタイン法を用い、盗用検知を行うプログラムを実装する。実装したシステムを用いた実験で、盗用と疑わしいプログラムを短い時間で発見できることを示す。

2.提案手法

本研究ではN-gram法とレーベンシュタイン法を用いることで盗用検知を行う。

2.1 システム構成

提出されたC言語プログラムに対して前処理を行い、N-gram法とレーベンシュタイン法を実行する。システムの概要を図1に示す。

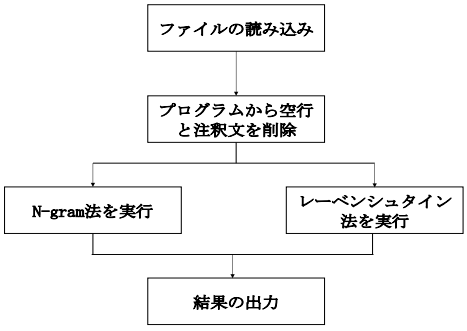

図1. システムの概要

An approach to efficiently detecting plagiarism in C language programs

† Azuma Takuma, Shibaura Institute of Technology

‡ Sasano Isao, Shibaura Institute of Technology

2.2 盗用発見手法

2.2.1 盗用について

プログラムの盗用は、入手した他の人のソースコードの一部を変更する偽装が行われることが多い。たとえば、空白文の挿入やコメント文の変更・削除、変数名や関数名、定数の変更、条件文の書き換えなどが挙げられる。

2.2.2 類似度計算手法

ソースコード間の類似度算出にはN-gram法を用いる。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、ベクトルの角度のコサインで類似度を算出する手法である。

本システムでは、N-gram法を用いて、2文字単位(2-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。

N-gram法を用いる利点は、利用する文書の形式を問わない点である。この特長により、様々なプログラミング言語のソースコードに対して本手法がそのまま適用できる。また、N-gram法、コサイン類似度はともに計算コストが比較的小さいため、十分に実時間処理が行える点も利点である。

またもうひとつの方法としてレーベンシュタイン法を用いる。レーベンシュタイン法は、文字列の編集における文字の削除、および追加を距離1としたとき、文字列全体を目的の文字列に置き換えるために必要となる最小距離のことである。レーベンシュタイン法は、2つの文字列の長さを n と m とすると、動的計画法により $O(nm)$ で求める。

3.評価実験とまとめ

3.1 実験概要

芝浦工業大学工学部情報工学科1年生を対象に実験を行った。学生から提出された課題すべてに対して総当りで本提案手法を実行した。出力されたN-gram法とレーベンシュタイン法の数値を対応、目視で比較した。その中で閾値を超えた盗用と疑わしいプログラムを発見できたかを分析した。

今回の閾値において、N-gram法ではコサイン類似度が0.9以上かつ、レーベンシュタイン法では距離50以下を盗用の疑いのある可能性の閾値とした。

3.2 実験結果

評価実験の結果を表1に示す。縦軸横軸共に受講者に割り振った番号であり、要素はレーベンシュタイン法による距離を表している。レーベンシュタイン法では距離50以下を赤色とした。つまりレーベンシュタイン法での数値が低いほど盗用の疑いがあるということである。

たとえば表1の赤い項目の6番と75番において、N-gram法ではコサイン類似度が0.9以上かつレーベンシュタイン法では距離9という結果が出ている。該当のプロ

グラムを目視で比較したところ、2.2.1節で述べた空白の挿入や変数式の違いが見られた。他の場合でも2.2.1節で述べたような傾向が見られ、疑わしいプログラムを検出できたと考えられる。

表1. 学生間での評価実験結果

	66	75	76	77	93	94
6	92	9	18	93	36	79
26	157	105	107	38	103	151
39	250	236	248	296	214	282
41	128	58	67	81	81	125
⋮	⋮					

しかし、盗用されたものが偶然に一致してしまう誤検知や、盗用したにも関わらず判定されない可能性が存在する。そこでそれを調べるために追加で実験を行った。故意の盗用を検知できるか調べるため表1とは異なるプログラムを用意した。表2では、本学大学生の実験協力者2人が受講生の作成したプログラムの39番と76番をそれぞれ選び、故意に盗用を行った場合の実験の結果を図3に示す。

表2. 学生と協力者のプログラムの比較

	6	39	41	60
協力者1	190	40	186	161
協力者2	133	112	129	82
	66	76	93	105
協力者1	212	117	202	175
協力者2	215	2	127	93

今回の実験において、簡単なプログラムであったが、偶然の一致は存在せず、盗用と正しく判定することができた。

3.3 比較実験

また本研究とMOSSとの比較実験も行った。前述とは別のソースコードを用意し、故意に盗用を行った。盗用を行うにあたり、printfとbooleanの関数呼び出し式を追加する方法を使用した。図2では例の一部を記す。

盗用元のプログラム	盗用したプログラム
printf(“・・・”); scanf(“%d”,&i); int a[i]; ⋮	boolean a; printf(“・・・”); printf(“”); scanf(“%d”,&i); printf(“”); int a[i]; ⋮

図2. 比較実験で使用した例

3.4 比較実験の結果

MOSSの実行結果は以下の図3と図4の通りである。図3は盗用元のプログラム、図4は盗用を行ったプログラムである。

図3と図4の赤い部分は、MOSSがこの二つを比較して類似していると判定した箇所である。3.3節で述べた実験と同じ方法の場合、MOSSはこの二つのプログラムを盗用と判定しなかった。理由として盗用と疑わしい箇所が連続していないため、盗用と判定しなかったからであると推測できる。

本提案の実行結果として、N-gram法ではコサイン類似度が0.964、レーベンシュタイン法では距離が42であった。これはN-gram法では、printfの関数呼び出し式における文字の出現頻度が高くなったことが理由として挙げられる。またレーベンシュタイン法では、宣言における文字数が比較的低いことから設定した閾値よりも低い数値となり、盗用の判定を出したと考えられる。

```
int main(void){
  int i,j;
  printf("配列の長さをしていしてください:");
  scanf("%d",&i);
  int a[i];
  for(j=0;j<i;j++){
    printf("a[%d] = ",j);
    scanf("%d",&a[j]);
  }
  printf("平均値は%fです. \n",average(a,i));
  return 0;
}
```

図3. MOSSによる盗用元のプログラムの実行結果

```
int main(void){
  int i,j;
boolean a;
  printf("配列の長さをしていしてください:");
printf("");
  scanf("%d",&i);
printf("");
  int a[i];
  for(j=0;j<i;j++){
    printf("a[%d] = ",j);
    scanf("%d",&a[j]);
printf("");
  }
  printf("平均値は%fです. \n",average(a,i));
  return 0;
}
```

図4. MOSSによる盗用したプログラムの実行結果

また時間効率について、表3に示す。縦軸は使用したシステム、横軸はプログラム群の名前、要素は時間(s)である。

表3. 本研究とMOSSの実行時間の比較

	P1	P2	P3
本研究	1.222	1.214	1.062
MOSS	12.862	10.586	13.115

今回の実験において、図2の追加方法については盗用の検知に成功した。

4. まとめと今後の課題

本研究ではN-gram法とレーベンシュタイン法を組み合わせて用いる盗用判定手法を提案した。少ない変更で故意に盗用した場合に、盗用と判定できた。また比較実験により本研究の優位性を示すことが出来た。しかし、今回実験したプログラムは初等クラスのもので、100行未満のプログラムが多い。今回は閾値を固定して判定を行っているために、わずかでも閾値に達しない場合に検出できないことがある。

今後の課題として100行を超えるプログラムについても検討する必要がある。また故意か偶然の判断基準を調査する必要がある。そして閾値を自動的に設定する方法を検討する必要がある。

参考文献

[1] 岩本舞, 小島俊輔, 中嶋卓雄, 学生のプログラミング演習におけるトークンベースのコードクローン検出手法, 情報処理学会研究報告 DPS-152, 2012

[2] 和田修平, 井上潮, 盗用発見と自動採点によるプログラミング演習課題の評価支援システム, DEIM フォーラム, 2011

[3] Saul Schleimer, DanielS. Wilkerson, and Alex Aiken, Winnowing: Local Algorithms for Document Fingerprinting, *ACM SIGMOD*, pp.76-85, 2003