Vol.2010-CE-107 No.9
2010/11/21

類似性に基づくレポート剽窃の検出ツールのプログラミング課題への適用

上田 和志† 富永 浩之†

類似性に基づくレポート剽窃の検出ツールを開発し，各種の教育支援システムにおいて，オンライン提出モジュールのプラグインとして提供する．類似性の判定には，編集距離および情報距離に基づく 2 つの手法を併用する．特に，プログラミング課題への応用を目指し，効率性と精密性の両立を目指して，適切な事前処理を導入する．ソースコードだけでなくアセンブルコードの組に対しても類似度を算出する．実際の学生の答案として，ポーカーの戦略プログラムの課題に対して適用した結果を報告する．また，軽微なコード変更による隠蔽行為についても，対策を検討する．

Plagiarism Detection Methods based on Similarity by Distance for Programming Reports

Kazushi UETA† Hiroyuki TOMINAGA†

Recently, the network environment in classroom has been much improved. However, the facilities may promote the imprudent plagiarism of other's answer in online report submission. We developed the detection tool for report plagiarism based on similarity. We adopt two kinds of approach for similarity judgment. The one is text base by edit distance. The another is binary base by file compression ratio. We prepare some preprocess for adequate judgment precision and cost performance. We apply it for source codes of C language. We calculate both similarity for source codes and assembled codes. It uses not only for exposure of the fraudulent activity by a teacher but also for the warnings at the time of students' uploading. We carried out an experiment for answer programs of Poker game strategy. We discuss concealment action and the counter plan for changing codes slightly.

1. はじめに

本研究室では，学内サイトの授業支援サービスと連携し，授業関連の個人情報を集約して提示する学生ポータルサイト PASPort を提案している[1]．そのドキュメント管理サービスとして，レポート作業を支援する WebBinder を開発中である(図 1)[2].

WebBinder は，オンラインストレイジの拡張であり，教員サイトおよびファイル共有サーバとユーザの間で橋渡しの役割をする．学生は，学内外を問わずレポート課題にアクセスできる. 文書作成などのアプリケーション依存の作業はローカル側で行い，関連ファイルの保管はサーバ側で行う．取得・作成・提出の 3 段階で支援機能を提供する．特に，問題選択の条件が煩雑で，複数の成果ファイルを要求されるプログラミング課題の提出を支援する．プロトタイプとして，フォルダーツリー表示による GUI を実装している．MS Windows のエクスプローラと同様に，マウスのクリックやドラッグ&ドロップでファイル操作が行える．

本論では，補助的なツールとして，オンラインレポートの剽窃を検出するモジュールの開発について議論する[3]．この機能は，教員側での不正行為の摘発や，提出前の学生への警告として利用する．また，コンテスト形式の C プログラミングの演習支援サーバ tProgrEss[4]など，関連研究のシステムにもプラグインとして組み込む.

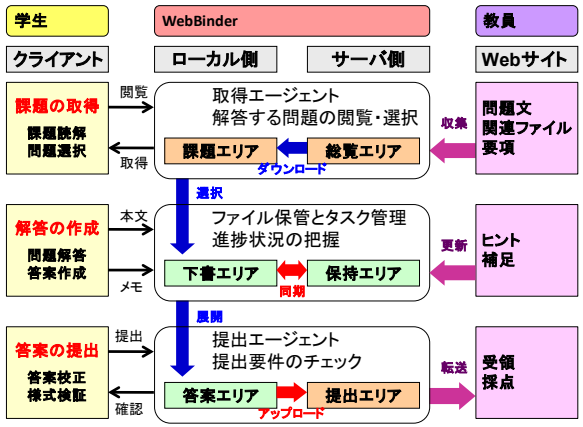

図 1 レポート作業支援システム WebBinder

†香川大学工学部
Faculty of Engineering, Kagawa University

2. プログラミング課題とレポート剽窃

2.1. レポート剽窃

大学キャンパスのネットワーク環境が整備され，学生 1 人に 1 台の PC を前提とした授業が可能となってきている．さらに，自宅の PC から大学ネットワークにアクセスしたり，ノート PC や携帯端末を持参することが日常的になっている．授業資料の配布，レポートの提出なども，オンラインで行うことが普通になってきた．オンラインでのレポート提出は，答案ファイルの管理や採点業務のシステム支援など，教員側にとって利便性が高い．反面，学生側には，安易なコピーが可能で，不正行為を誘発する原因にもなる．他人の著作物を引用として明示せず，自分のものとして掲載する剽窃は，決して見逃してはならない問題である[5][6][7].

オンラインでのコピーは，PC 上の操作として，コピー・アンド・ペーストで簡単に行えるため，「コピペ」という俗称がある．そのくらい，多くの学生にとって，日常的に可能な行為となっている．学生によるレポートの剽窃には，Wikipedia など，Web 上の文献からのコピーと，学生同士の答案のコピーとがある．前者については，検索エンジンの発展が剽窃することを助長しており，グーグルコピーペーストシンドロームなる言葉まで登場している[8]．後者については，インスタント・メッセンジャーの普及により，個人間でのファイル転送が気軽にインタラクティブに行えるようになった影響が大きい．

2.2. レポート剽窃の検出

前者の検出については，コピー元の文章は Web 上にあるため，その全てと比較することは困難である．そのため，システムが剽窃されていそうな文章を予め抽出しておかなければならない．したがって，Web 検索サイトのクローラーとの連携が必要となる. すなわち, 答案の文章の一部を検索エンジンにかけて, コピー元を突き止める[9]. 特に，金沢工業大学知的財産科学研究センター長の杉光一成教授が考案し，株式会社アンクが開発した，コピペ判定支援ソフト「コピペルナー」が販売され，話題となった[10]．判定結果にコピペ割合やコピペ元も表示し，直感的な GUI となっている．米国では, 分かち書きで分析しやすいこともあり, 多くの研究がある[11][12][13][14][15]. 有名なシステムでは，PAIRwise[16]や TurnItIn[17][18]が挙げられる．

後者については，全ての答案の組合せに対し，類似性を調べることになる．この方法は，答案数が多いと，非常に時間のかかる処理になる．「コピペルナー」には，クロスチェッカーとして，こちらの機能も存在する．ただし，製品の説明からは，日本語文書への適用を想定したものと思われる．いずれの方法でも，システム的な処理だけで，剽窃かどうかを完全に見分けることは困難である．最終的には，教員による確認が必要である．また，語句の説明，自由な論評，数学の解答，プログラムなど，対象領域によっても，その精度は大きく影響される．文書の類似度を求める手法については，n-gram を用いる方法[19]や，単語の出現数を比較する方法が提案されている[20].

我々は，主に情報系の講義とプログラミング演習を教えている．当初のターゲットは，初級 C 言語演習での C ソースコードである．プログラミング問題においては，剽窃は学生相互で行われることが多いと考えられる．同じような動作をするコードは，Web を探してもなかなかないからである．そこで，相互コピーに焦点を当てる．

2.3. プログラミング課題

学生が扱う学業関連のドキュメントは，学務との事務書類，教員との配布教材や課題レポートである．学生にとってのドキュメント管理は，教員や学務からの指示に従い，所定の形式で文書を作成し，期日までに提出し，それに対する評価を受けることである．指定された締切や様式に従わないと不利益を被る．本研究では，これらの文書を Web 上で管理する上で，特にプログラミング課題に着目し，取得と提出の代行，および形式的な作業支援を検討する．

プログラミング課題には， 以下のような特徴がある．受講者によって，理解度が大きく異なり，解答 時間にも開きがある．問題の正否が明確で，出来不出来がはっきりする．そのため，難易度の異なる複数の問題を提示し，各自が選択できるようにすることが多い．また，課題提示として，サンプルコード，ライブラリ，処理対象データなど，付随する配布物が多い(図 2)．答案提出として，ソースコード，実行バイナリ，実行結果など，必要なファイルが多様で，選択した問題によって異なることもある．教員側でファイルの有無や実行確認などの自動処理を行う場合，指定されたフォルダ構成でファイル群を整理して提出することが要求される．

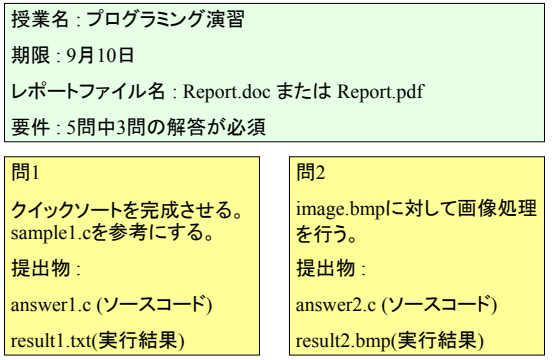

図 2 プログラミング課題の例

Vol.2010-CE-107 No.9
2010/11/21

3. 類似度に基づくレポート剽窃の検出手法

3.1. レーベンシュタイン距離に基づく類似度

レーベンシュタイン距離は，文字列同士の編集距離のうち，最も簡単なものである[21][22][23]．文字単位の挿入・変更・削除という基本操作の最低回数で，距離を求める．例えば，文字列 bcabc と abdd は，以下の 2 通りの一致のさせ方が考えられるが，基本操作の回数がより少ない方を採用して距離 4 となる．

```
a  b  d  d          1 文字一致　距離 5
   b  c  a  b  c
         a  b  d  d  2 文字一致　距離 4
```

レーベンシュタイン距離は，漸化式を用いた再帰的定義で与えられる．ただし，文字列長を|X|，先頭文字を削除した文字列を X'で表す．

$$LEV(S,T)=\begin{cases}|S| & ;\ T=\phi\\ |T| & ;\ S=\phi\\ LEV(S',T') & ;\ \text{先頭が一致}\\ \min\begin{Bmatrix}LEV(S',T)\\ LEV(S,T')\\ LEV(S',T')\end{Bmatrix}+1 & ;\ \text{不一致}\end{cases}$$

実際の計算には，動的計画法による効率的な算法が知られている[21][22][23]．図 3 のような作業表を作り，左上から右下へ，最も操作回数が少なくなるような経路を選んでいく．初期値として，文字列の先頭に仮想的な空文字を入れておき，最上行と最右列に，0 から順に数値を入れておく．左上隅の 0 から始め，右・下・右下への移動が，横の文字列からみて，削除・挿入・変更に相当し，値を 1 加算する．ただし，文字が一致するときは，右下の移動において加算しない．これらの 3 通りの値のうち，最小となるものを作業表に書き込んでいく．右下隅に達したときの値が，求める距離である．

	ϕ	b	c	a	b	c
ϕ	0	1	2	3	4	5
a	1	1	2	2	3	4
b	2	1	2	3	2	3
d	3	2	2	3	3	3
d	4	3	3	3	4	4

図 3 レーベンシュタイン距離の計算

レーベンシュタイン距離は，文字列同士がどれくらい異なるかの絶対量を表しており，文字列長の影響を受ける．一般に，|S|≧|T| ならば |S|－|T|≦LEV(S,T)≦|S| となる．特に，T＝φなら LEV(S,T)＝|S| となる．そこで，編集距離を割合として正規化し，区間[0,1]に収まる類似度 Sim_L を以下のように定義する．この値は，完全に一致しているとき，1.0 となり，全ての文字が異なるとき，0.0 となる．

$$Sim_L(S,T)=1-\frac{LEV(S,T)}{\max\{|S|,|T|\}}$$

3.2. NCD による類似度

正規化圧縮距離 NCD(Normalized Compression Distance)は，コルモゴロフ記述量を具体的な圧縮算法で近似し，距離として定義したものである[24]．コルモゴロフ記述量は，文字列の複雑さを，その文字列を出力するための算法の記述量の最小値として定義される．すなわち，データ圧縮の下限である．これは特定のプログラム言語に依存しない抽象的な定義であるが，通常は 1 つの言語を仮定して議論を進める．例えば，文字列 0123012301230123 は，Ruby 言語では print "0123" *4 と記述できる．ここで，文字列 T を 0123 とすれば，print T *4 とさらに短くできる．コルモゴロフ記述量は K(S)と書く．また，文字列 T に含まれる情報を用いて，文字列 S を圧縮したものを相対コルモゴロフ記述量と呼び，K(S|T)と書く．一般に，K(S)≧K(S|T) となるので，K(S)－K(S|T) は，T を用いたデータ圧縮の改善度，また S に含まれる T の情報量である．これを両者の類似度とみなす．

実用的には，一般的に用いられる圧縮ツールで圧縮したサイズ C(S)で十分である．相対記述量については，文字列 T,S を連結した文字列 TS を使い，C(S|T)＝C(TS)－C(S)で代用する．この値は，文字列同士がどれくらい異なるかを表しているが，文字列長の影響を受ける．そこで，割合として正規化し，NCD を以下のように定義する．

$$NCD(S,T)=\frac{C(TS)-C(T)}{C(S)}\quad ;\quad C(S)\geq C(T)$$

類似度としては，1.0 が完全一致となるように，1 から引いて，以下のように NCD を定義する．ただし，実際の圧縮ツールでは，ヘッダ情報などが付加されるため，正確に 0.0 から 1.0 の間に収まるわけではないが，実用的に問題はない．

$$Sim_N(S,T)=1-\frac{C(TS)-C(T)}{C(S)}=\frac{C(S)+C(T)-C(TS)}{\max\{C(S),C(T)\}}$$

4. レポート剽窃の検出方法と事前処理

4.1. ソースコードに対する事前処理

レーベンシュタイン距離の計算は，動的計画法を用いても，文字数の 2 乗に比例する処理時間が必要である．レポートの文章が長くなると膨大な計算時間がかかる．しかし，各文字の比較は，一致/不一致の単純な判定である．したがって，比較対象の文字種を増やし，文字数を削減することが考えられる．

そこで，ソースコードの事前処理としては，字句解析によってトークンに分け，予約語やユーザ定義の識別子のみを抽出する．括弧やコンマなどの区切り文字は，トークン数としてはかなりの量になる．しかし，構文的にほぼ正しいコードの場合，類似性に大きな影響を与えないと考えられる．また，コードの一部を補完するような問題では，共通のコード部分を除外しておく．

コード自体を簡易パーサにかける以外に，アセンブラ言語の中間コードに変換し，そのコード上でのトークン列とする方法が考えられる．この方法では，変数名の置換や while 構文を for 構文に書き換えただけといった，軽微な変更の影響を受けない．データ量は余り減らないが，より精密に類似度を算出でき，少し細工された剽窃も見つけられる．コード自体のトークン列で剽窃の候補を絞り，中間コードのトークン列でさらに確認するという使い方が考えられる．

4.2. 検出ツールの実装

事前処理の主要部分および剽窃検出の本体処理は，Ruby 言語で実装した．また，Web アプリケーションからのプラグイン利用を想定して,インタフェースを整備する．ただし，編集距離は計算時間がかかるため，その部分だけを C 言語で実装し，Ruby スクリプトから呼び出している．アセンブラには，GCC を用いる．

圧縮ツールとして，GZIP と BZIP2 を検討する．GZIP は圧縮率が低いが速い，BZIP2 は圧縮率が高いが遅い, という特徴がある. 圧縮率が高いアルゴリズムを利用すれば，類似度の精度を上げることができる．しかし，テキストデータに対しては，GZIP で十分な圧縮率を得ることができるため，テキストデータには GZIP，バイナリデータには BZIP2 を使用する．

4.3. 検出ツールのモジュール構成

レポート剽窃の検出ツールは，編集距離と情報距離のそれぞれで，図 4 のようなフィルタ群として設計する．対象領域の特性に応じて，両者の手法を組み合わせ，パラメタ設定が柔軟に行えるようにする．また，ツールだけで，剽窃を検出するわけではないので, 判定結果や疑わしい組を, 教師に分かりやすく表示する GUI も必要となる．単独で用いる場合には，対象データを指定し，全ての組合せの類似度を求め，その順位表を表示する GUI を用意する．

4.4. WebBinder へのプラグイン

本研究室では，学内サイトの授業支援サービスと連携し，授業関連の個人情報を集約して提示する学生ポータルサイト PASPort を提案している．そのドキュメント管理サービスとして，レポート作業を支援する WebBinder を開発中である．

WebBinder は，オンラインストレイジの拡張であり，教員サイトおよびファイル共有サーバとユーザの間で橋渡しの役割をする．学生は，学内外を問わずレポート課題にアクセスできる. 文書作成などのアプリケーション依存の作業はローカル側で行い，関連ファイルの保管はサーバ側で行う．取得・作成・提出の 3 段階で支援機能を提供する．特に，問題選択の条件が煩雑で，複数の成果ファイルを要求されるプログラミング課題の提出を支援する．

レポート剽窃の検出ツールは，教員側での不正行為の摘発のため，WebBinder の教員サイトへのサービスとして，API を提供する(図 5)．また，提出フェーズにおいて，提出前の学生への警告としても利用する．

4.5. tProgrEss へのプラグイン

本研究室では，小コンテスト形式の C プログラミングの演習支援サーバ tProgrEss を開発している．プログラミング問題の解答となるソースコードをアップロードし，サーバ側でコンパイルして，入力サンプルを与えて実行する．実行結果と出力サンプルを照合し，プログラムの正誤判定を行う．出力サンプルは，正解となる模範プログラムの実行結果である．正誤判定には，6 段階あり，最初の段階で，不正提出の判定を行っている．空ファイル，巨大ファイル，バイナリなどを排除する．また，プロセス制御文など悪意のある危険コードのうち，字面で判断できるものも排除する．

剽窃検出は，上記の不正提出の判定部分に含める．既に提出された他の学生のコードと明らかに類似性の高い場合を排除する．疑わしいが明らかと言えない場合は，警告のみを出して，以降の正誤判定を進める．コードにフラグを立てておき，後で教員が目視で確認する. なお, 同じ学生の修正されたコードとの比較は行わない. tProgrEss は，Ruby 言語で開発されており，Ruby 言語で実装された剽窃検出ツールとの親和性が高い．即時的な判定が求められるため，効率性を高める事前処理が重要となる．

Vol.2010-CE-107 No.9
2010/11/21

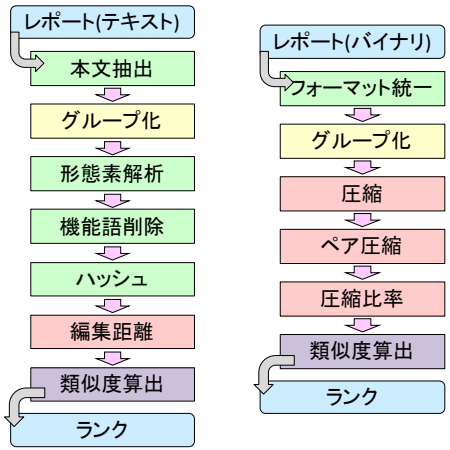

図 4 剽窃検出ツールのモジュール構成

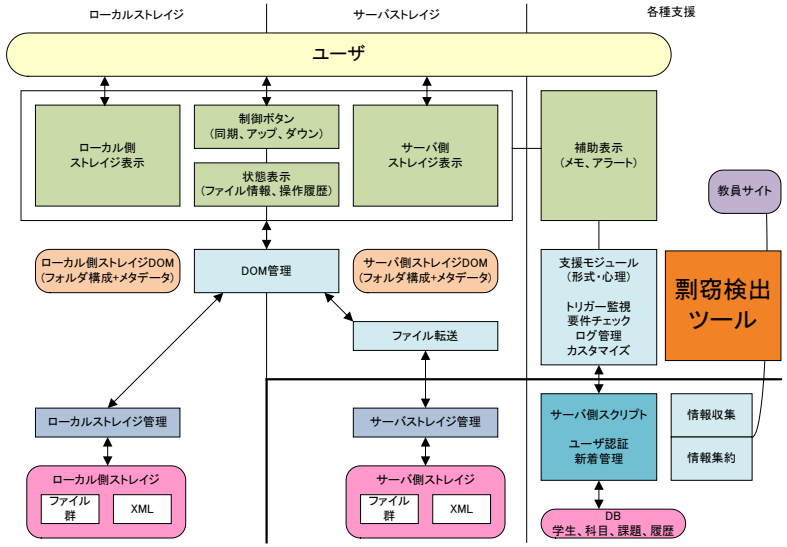

図 5 WebBinder への剽窃検出ツールの組込み

5. ソースコードに対する剽窃検出の実験

5.1. 実験対象のソースコード

C 言語のソースコードに対して，剽窃検出の予備実験を行った．対象データは，2004 年度から 2008 年度までの「ソフトウェア開発演習」(3 年次選択科目)におけるプログラミング課題のソースコードである．内容は，知識情報処理の分野の応用であり，ポーカーにおいて捨てる札を選ぶ戦略を関数 strategy() として実装するものである．この関数を記述するファイルは，ゲーム進行の共通処理とは別ファイルになっており，学生に提示したコードは，関数原型と僅かなコードの例示だけである．そのため，純粋に類似度を判定するのに適している．データ数は 170 件で，類似度を計算する組としては，14450 件となる．平均で，ファイルサイズは 9546 バイト，コメントを除外して，行数は 225 行，文字数は 4865 字(バイト)である．予約語と識別子のトークン数の平均は，元のコードで 617，中間コードで 3800 であり，前者は後者の約 1/6 である．また，GZIP による圧縮では，平均 928 バイトとなっており，約 1/5 に圧縮されている．

5.2. ソースコードに対する処理時間

編集距離による類似度の計算では，事前処理として字句解析に数秒以下，本体処理に約 8 分かかった．1 件のデータを他の 169 件の対象データと比較する時間は，2 秒程度である．1 組だけの比較は，0.02 秒程度である．中間コードを用いた場合は，事前処理のアセンブルに数秒，本体処理に約 3 時間かかった．ただし，構文エラーなどでアセンブルできないコードもあったため，正しく中間コードが生成できたのは，157 件である．1 件につき，比較の総時間は，2 分程度である．情報距離による類似度の計算では，事前処理として GZIP での圧縮に 2 秒，本体処理に 23 秒かかった．1 件につき，比較の総時間は，0.14 秒程度である．実験に用いたマシンの性能は，CPU Intel Core2Duo 3.0GHz，RAM DDR2 4GB である．OS は，Linux の 64bit 版 CentOS 5 上である．開発言語は，Ruby 1.9.1 である．以前よりもメモリ管理が改良され，性能が向上した．

5.3. ソースコードに対する実験結果

ソースコードについては，元コードおよび中間コードについて，編集距離による類似度の相関が高かった．両者の値が高い組を目視で確認した結果，上位から 5 件が明白な剽窃であると判断できた．このうち，2 組は学年を超えた学生のものであり，両者の所属ゼミが一致していることから，剽窃の疑いが極めて濃厚であるといえる．

一方，情報距離による類似度は，編集距離による類似度との相関が，必ずしも高いと言えなかった．類似度が高い組には，明らかな剽窃が認められないものが含まれて

いた．その理由として，例示のコードから，ほとんど修正していないコードや，if 構文を延々と繰り返す冗長なコードの存在が挙げられる．前者は，剽窃以前の状態である．後者は，圧縮比が高く，自己コピペ状態になっていた．これらのコードは，剽窃検出としては除外するが，別の観点から教育的指導を行うべきである．

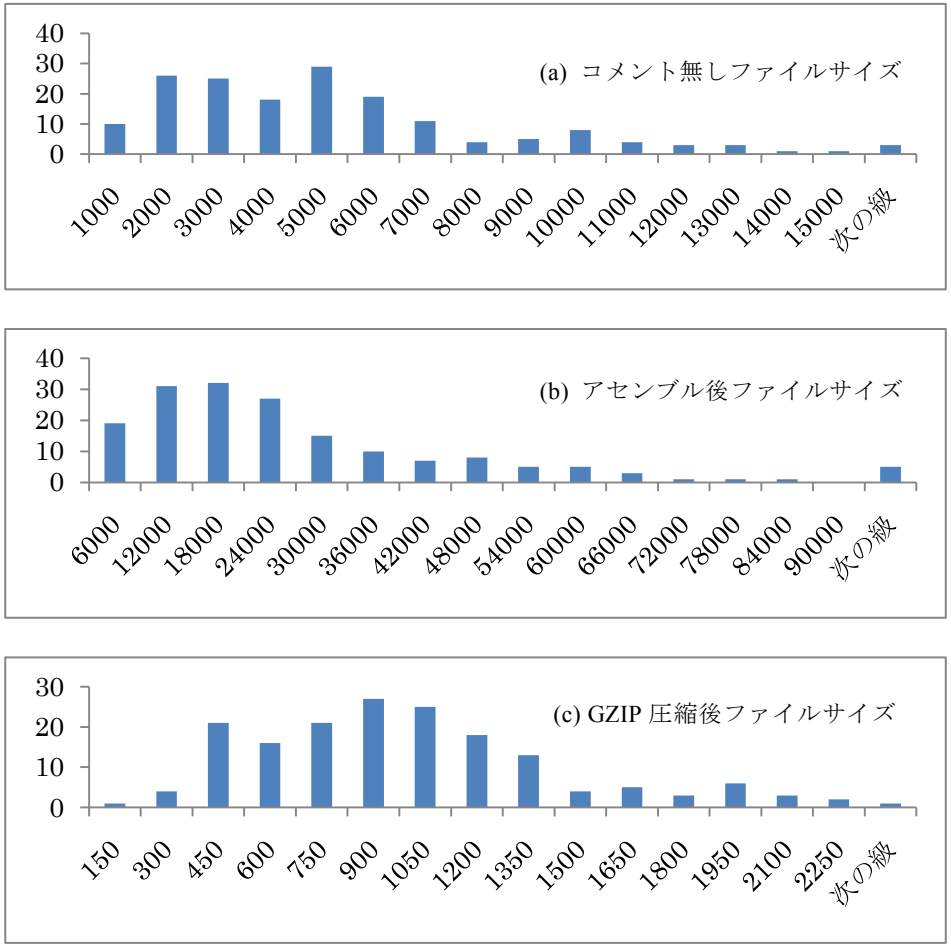

図 6 戦略プログラムのサイズ分布

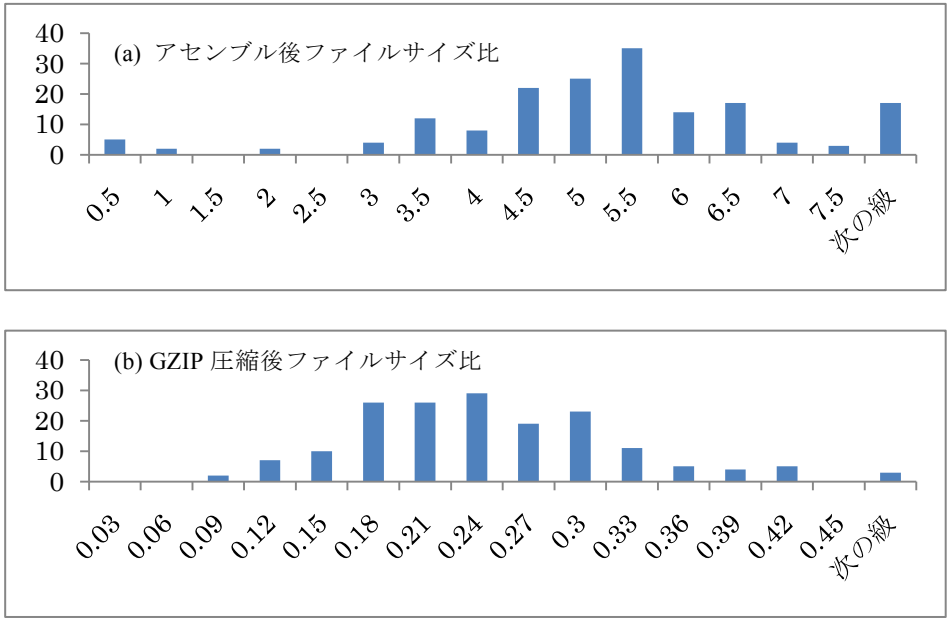

図 7 戦略プログラムの事前処理によるサイズ比

6. 改良とその設計

6.1. 事前処理におけるスクリーニング

剽窃検出の前段階として，正しくコンパイルができないコードや C 言語ソースコードとして体をなしていないものの検出を行う．図 6 は，コメントを抜いたコードのファイルサイズ，アセンブル後，GZIP 圧縮後のサイズ分布である．図 7 は，コメント抜きのファイルサイズと事前処理を行った後のサイズとの比率である．

まず，アセンブルコードとオリジナルコードの比率から異常なコードを検出する．比率が小さすぎるもの，つまりアセンブル後のコード量が少ないものはコンパイルに失敗している．また，大きすぎるものは冗長なコードである．比率が 2.0 よりも少ない 9 件は，コンパイル時に何らかのエラーが出ている．また 9.0 を超えるようなコードでは，冗長なコードになっていることが多い．例としては，

```
if (hd[0]%13==hd[1]%13 && hd[2]%13==hd[3]%13) return 4;
```

のようなコードが，何行も続くプログラムがあった．

また，オリジナルコードの圧縮前と圧縮後のサイズを比較する．比率が小さすぎるもの，つまり小さく圧縮されたコードは，自己相関性が高く，冗長なコードとして判定できる．特に 0.1 以下のコードは，for 文を効果的に使用していないコードが目立った．比率が大きなものは，課題のレベルと比較してコード量が少なく単純すぎるコードであった．今回は 0.4 以上が該当した．

前者と後者どちらも冗長なコードを発見できるが，傾向が異なり，2 つの手法を組合せることによってより正確に冗長なコードを発見することができた．これらのコードとの組では，類似度が不適切な値になりがちであるため，除外した上で，処理を行う．

6.2. レポート剽窃における軽微なコード変更のパターン

理解度の低い学生は，他の学生のレポートをコピーしたうえで，丸々コピーだとわからないように様々な改変を加える．学生がプログラミングレポートにおいてどのような剽窃行為を行うかいくつかまとめられている[25]．それぞれの行為に対し，本研究でどのような対策をとるのか述べる．

(1) コメント文の書換え

コメントを消したり，書き加えたりすることによって見た目を変える行為である．どのようなコメントを記述しても，プログラムの動作には影響がないため，コードを評価する上でコメントは対象外とし，事前処理にて削除する．

(2) 括弧の位置，インデントの変更

ブロックを表す中括弧の位置を行頭，行末に移動したり，インデントを Tab にしたりするなどのプログラムの動作には全く影響を及ぼさない変更である．これらの変更は，パージングすることで無効となる．コード整形ツールを用いて，標準的なスタイルに変換してから，処理を進めることも考えられる．

(3) 変数名，関数名の変更

変数名や関数名を変更する行為である．置換するだけの処理であるため，全くプログラミング技術を必要としない．中間言語に置き換えることによって，変数名に関係のないレジスタ名に置き換えることができる．

(4) 式の書換え

a++; を a+=1; にしたり，for 文を while 文にするといったプログラムの意味の変更を伴わない変更を指す．中間言語へ変換することによって，コンパイラの最適化機能が働き，このような変更を排除することができる．それでも a=b*c+d; を a=b*c; a+=d; と式を分割するなどの方法によってごまかそうとする場合は，そのままのコードの編集距離を算出するなど，複数の手法を組み合わせて対策を行う．また puts 関数を printf 関数で置き換えるといった行為に関しては，前処理においてどちらかに統一する処理を行う．

(5) 順序に依存しない式の入替え

a++; b++; を b++; a++; にするなど，プログラムとしての意味が変わらない程度に式の順序を入れ替えてしまう行為が考えられる．レーベンシュタイン距離を改良した Smith-Waterman アルゴリズム[26]を使用することによって，このような入替えの影響を排除することができる．

(6) 定数の変更

定数で配列の要素数が定義されている場合など，その数字を変えることによって実行バイナリはやや異なるものになる．しかし，大きく様変わりすることはないと考える．対策が必要な場合は，事前に定数を展開した上で置換処理を行うことが考えられる．

(7) コード分割

コードの一部を関数としてまとめる行為が考えられる．Smith-Waterman アルゴリズムで検出することは可能である．ただ，プログラミング教育という観点では，きちんと構造化されたコードを書くことは評価される行為であり，当システムでは，特に剽窃とは扱わない．

6.3. 検出方法の精密化

検出手法の精密化について，検討している手法を述べる．まず，オリジナル部分の多少を測る指標として，非対称な依存度を導入する．X の Y に対する依存度は，以下で定義する．類似度の性質から依存度も区間[0,1]に収まる．

$$Dep(X,Y)=\begin{cases} Sim(X,Y) & ; \ |X|\geq|Y| \\ \frac{|Y|}{|X|}Sim(X,Y) & ; \ |X|\leq|Y| \end{cases}$$

例えば，S=abcdefgh，T=abx のとき，類似度は Sim(S,T)=6 で，両者の依存度は Dep(S,T)=0.25, Dep(T,S)=0.25×8/3＝0.67 となる．S は T と似ている部分 ab があるものの，cdefgh とオリジナルな部分の方が多い．T は S にも含まれる ab を除けば，オリジナルな部分は x しかない．

この依存度の対象データ全てに亘る総和として，剽窃度 Plag(S)＝ΣDep(S,X) を定義する．剽窃度が大きいと，他と似ている部分が多く，オリジナルな部分が少ない．すなわち，コピペの疑いが高い．特に，サイズが小さく，他からつまみ食いしたようなデータを検出し易い．

7. おわりに

レポート剽窃の効率的な検出方法を検討し，実用的なツールを開発中である．類似性の判定には，編集距離および情報距離に基づく 2 つの手法を併用する．適切な事前処理を行って，実用的な判定精度と効率化を目指す．特に，C 言語のソースコードに対しては，字句解析によるトークン列への帰着，共通コードや区切記号の削除などを行う．アセンブルした中間コードも利用する．Ruby 言語で実装し，ユーザインタフェースとシステムインタフェースを整備する．レポート作業支援 WebBinder やコード正誤判定 tProgrEss にプラグインとして組み込む．実際の適用に向けて，ソースコードの剽窃の典型的なパターンを列挙し，軽微な変更による抜け道への対策を議論した．また，一致する部分列の出現を調べるアルゴリズムも検討している．新村[27]では，時系列情報を基に，コピーしあう学生グループを検出している．このようなグループに警告を与えるような機能も検討する．これらの手法を組合せ，検出の精度と処理の効率の実用的なバランスを目指す．

文　献

1) 三嶋宏資, 福島直文, 富永浩之: 授業支援サービスと連携した個人適応の学生ポータルサイト -授業関連情報を時間割ベースで共有する予定表-, 信学技報, Vol.107, No.536, pp.101-106, (2008).
2) 上田和志, 富永浩之: プログラミング課題のレポート提出を支援するオンラインストレイジ WebBinder の開発, JSiSE 研究報告, Vol.23, No.6, pp.112-119, (2009).
3) 上田和志, 富永浩之: 類似性に基づくレポート剽窃の検出ツールの開発, 信学技報, Vol.109, No.335, pp.61-66, (2009).
4) 川崎慎一郎, 富永浩之: 競争型学習を取り入れた入門的 C プログラミング演習 - 演習支援サーバ tProgrEss の出題解答と採点結果のページ表示の改良 -, 信学技報, Vol.109, No.335, pp.187-192, (2009).
5) H. Maurer, B. Zaka: Plagiarism - A Problem And How To Fight It, Proceedings of ED-MEDIA 2007, pp.4451-4458, (2007).
6) Y. Wang: University Student Online Plagiarism, International Journal on E-Learning, Vol.7, No.4, pp.743-757, (2008).
7) J. Suarez, A. Martin: Internet Plagiarism: A Teacher's Combat Guide, Contemporary Issues in Technology and Teacher Education, Vol.1, No.4, pp.546-549, (2001).
8) H. Maurer, N.Kulathuramaiyer: Coping With the Copy-Paste-Syndrome, Proceedings of E-Learn 2007, pp.1071-1079, (2007).
9) 高橋勇, 他: Web サイトからの剽窃レポート発見支援システム, 信学論, Vol.J90-D, No.11, pp.2989-2999, (2007).
10) アンク社, コピペ判定支援ソフト コピペルナー, http://www.ank.co.jp/works/products/copypelna/.
11) R. Patton, D. Johnson, B.Bimber, K. Almeroth, G. Michaels: Technology and Plagiarism in the University: Brief Report of a Trial in Detecting Cheating, AACE Journal, Vol.12, No.3, pp.281-299, (2004).
12) V. Brown, N. Robin, R. Jordan: A Faculty's Perspective and Use of Plagiarism Detection Software, Proceedings of SITE 2008, pp.1946-1948, (2008).
13) M. Thomas: Plagiarism Detection Software, Proceedings of E-Learn 2008, pp.2390-2397, (2008).
14) A. Knight, K. Almeroth, B. Bimber: An Automated System for Plagiarism Detection Using the Internet, Proceedings of ED-MEDIA 2004, pp.3619-3625, (2004).
15) M. Tang, R. Byrne, M. Tang: University anti-plagiarism efforts versus commercial anti-plagiarism software and services and do online students cheat more?, Proceedings of E-Learn 2007, pp.6595-6601, (2007).
16) A. Knight, K. Almeroth, B. Bimber: Design, Implementation and Deployment of PAIRwise, Journal of Inteactive Learning Research, Vol.19, No.3, pp.489-508, (2008).
17) iParadigms, TurnItIn, http://turnitin.com/.
18) M. E. Stetter: Plagiarism and the use of Blackboard's TurnItIn, Proceedings of ED-MEDIA 2008, pp.5083-5085, (2008).
19) 小高, 他: n-gram を用いた学生レポート評価手法の提案, 信学論, Vol.J86-D-I, No.9, pp702-705, (2003)
20) 深谷亮, 他: 単語の頻度統計を用いた文章の類似性の定量化 - 部分的類似性の考慮 -, 信学論, Vol.J87-D-II, No.2, pp.661-672, (2004).
21) G. Pighizzini: How Hard Is Computing the Edit Distance, Information and Computation, Vol.165, No.1, pp.1-13, (1995).
22) E. Ukkonen: Finding approximate patterns in strings, Information and Control, Vol.64, pp.100-118, (1985).
23) H. Hyyroe: A Bit-Vector Algorithm for Computing Levenshtein and Damerau Edit Distance, Nordic Journal of Computing, Vol.10, pp.1-11, (2003).
24) Sven Helmer: Measuring the Structural Similarity of Semistructured Documents Using Entropy, Proceedings of the 33rd international Conference on Very Large Data Bases, pp.1022-1032, (2007).
25) 布目淳, 福澤理行, 平田博章: プログラミング実習におけるコード評価のための e ラーニングバックエンドシステムの開発, 信学技報, Vol.108, No.24, pp.59-64, (2008).
26) 太田貴久, 増山繁: 学生レポート採点支援のためのレポート類似部分発見手法, 信学技報, Vol.105, No.594, pp.37-42, (2006).
27) 新村, 他: レポート類似度の時系列解析に基づく学習者間の依存関係発見システムの開発, JSiSE 学会誌, Vol.26, No.1, pp.59-67, (2009)