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

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 プログラミングの授業において受講者に対して出す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．その原因は，採点作業は多くの工程からなり，これをすべての提出されたプログラムに対して繰り返し行う必要があるためである．また，他の受講者が作成したプログラムをコピーして提出する盗用が行われる可能性があるため，その確認作業も大きな負担となる．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受講者のプログラムを用いて評価した結果，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，本システムで用いた類似度の算出方法が盗用の発見にきわめて有効であることを確認した．

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

大学のプログラミングの授業において，学生の理解度を確認するために，指定された仕様を満たすプログラムを作成するプログラミング課題がよく用いられている．しかし，その採点作業の多くは手作業で行われており，受講者が多い授業では講師にとって大きな負担となっている．

その理由は，提出されたすべてのプログラムについて，コンパイル，実行，テストデータ入力，入力に応じた出力結果の確認，ソースコードの確認，そして採点という多くの工程が必要であるためである．各工程の作業内容は単純ではあるが注意力が必要であり，かつ相当の時間を要する．このことから，採点作業の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不正行為である．このプログラム盗用(program plagiarism)[1]を発見するためには，提出された全てのソースコードを相互に比較して，どの程度類似しているかを判断しなければならない．比較回数は答案数の 2 乗に比例するため，特に受講生の多い講義科目においては採点者の大きな負担となる．

本稿では，プログラミング課題の採点に関わる一連の作業を自動化するとともに，採点ルールを元に自動採点を行う機能と，類似度の算出により盗用の発見を助ける機能を有するシステムについて検討する．

2. 既存研究

熱田らの授業支援システム[2]は，ウェブアプリケーション形態のシステムであり，受講者がウェブブラウザによってアップロードした提出ファイルを自動実行する 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できる．

また，松浦によるプログラミングレポート採点支援ツール[3]は，ローカル環境で実行する形態のソフトウェアである．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約数を出力せよ」など実行時に入力操作が必要な課題の場合に,採点者が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問題がある．また，プログラム盗用への対策もなされていない．

一方，Prechelt らの JPlag[4]は，プログラムの盗用検出を目的としたシステムである．JPlag は，ソースコードを独自のトークン列に変換し，トークン単位で相互に比較を行うことにより盗用を発見する．しかし，初級レベルのプログラミング課題の場合，課題設定が同一である点，ソースコードが小規模である点から，盗用ではないプログラム間でもトークンが類似し，盗用の有無の判別が困難である場合が多い．

3. 提案手法

3.1. システム構成

本研究では，初級レベルのプログラミング授業で用いられる標準入出力によって処理対象の入力と処理結果の出力を行う形態のプログラムを対象とする．電子メールやファイルサーバによって課題の提出を行うよ

うな既存システムと連携させることが出来る点，システムの導入や保守という負担が採点者に掛からないという点から，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の形式とした．システムの概要を図 1 に示す．

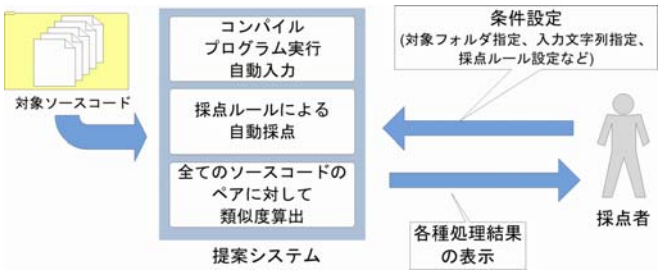

図 1　システム概要

次に，システムの操作方法を説明する．まず採点者は提出物が含まれているフォルダを指定する．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と採点ルールを設定し，コンパイル対象のファイル名を正規表現で指定する．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し，出力結果の記録と自動採点を行う．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．

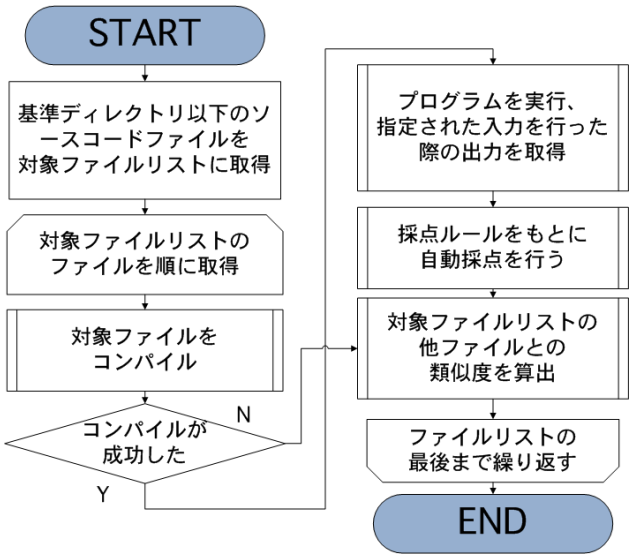

図 2　システムのフローチャート

本システムの実行画面を図 3 に示す．処理の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，出力結果，他のソースコードとの類似度が一覧できる．また，任意の項目を CSV 形式で外部のファイルへ出力できる．

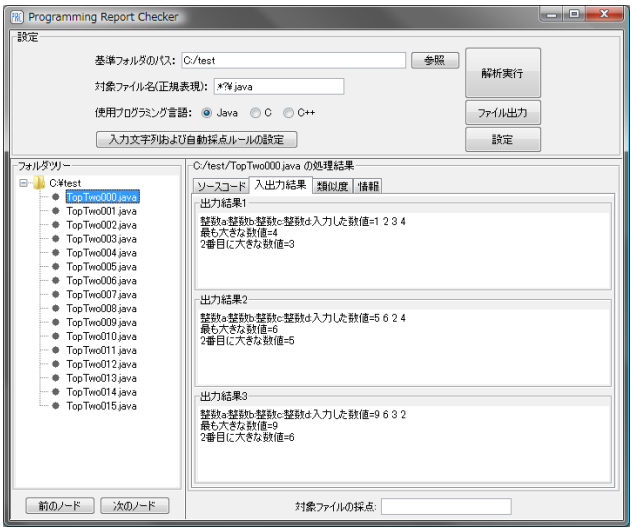

図 3　システム実行画面

3.2. 自動採点手法

自動採点は，採点者があらかじめ以下のルールと，その重み付けを定めておくことによって実現する．また，自動採点の結果に応じたグレード分けも行えるようにする(図 4).

(1) コンパイルが成功したか
(2) 入力に応じた出力が，指定した正規表現を満たしたか，あるいはエラーが出力されたか
(3) 指定した単語がソースコード中に含まれるか

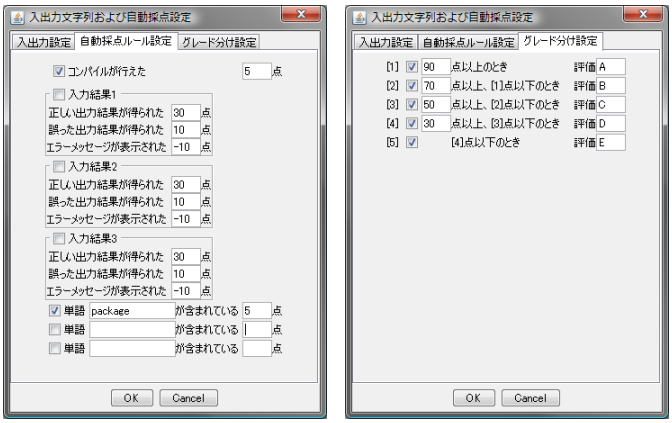

図 4 自動採点設定画面

3.3. 盗用発見手法

3.3.1. 盗用について

プログラムの盗用においては，入手した解答となるソースコードの一部表現を書き換えるという偽装が行われることが多い．しかし，一般に盗用を行う学生のプログラミング能力は低く，書き換えに多くの時間を要するような大規模な改変が行われることは少ない．このため，ソースコード間の文書としての類似度を求めることが，盗用の発見の大きな手助けとなると考え

られる．

3.3.2. 類似度算出手法

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，切り出した要素の共通性や，ベクトル化した距離によって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，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．

コサイン類似度 sim(*Va*, *Vb*)は，文書 a から求めたベクトルを *Va*，文書 b から求めたベクトルを *Vb* としたとき，次式で求めることができる．

$$sim(Va,Vb) = \cos\theta = \frac{Va \cdot Vb}{|Va||Vb|}$$

コサイン類似度は，0 から 1.0 の値をとり，値が大きいほどベクトル間の差異が少なく，類似していると言える．従って，この値に 100 を掛けた値をソースコード間の類似度[%]と定めることができる．

類似度算出手法の概要を図 5 に示す．

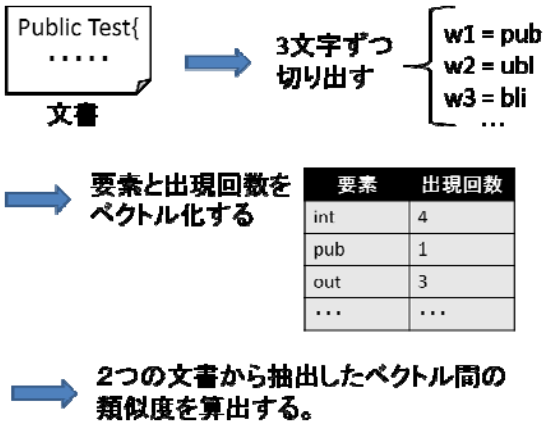

図 5　類似度算出の概要図

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．

3.3.3. 偽装対策手法

先に述べたように，盗用されたプログラムにおい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．従って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，様々な偽装への対策を前処理として行っておく必要がある．

プログラミング課題における代表的な偽装としては，以下のものがある．

(1) 空行の追加
(2) 注釈文の追加，変更
(3) 文字列リテラルの変更
(4) 変数名の変更
(5) 処理順序の入れ換え

このうち(1)，(2)，(3)に対しては，ソースコードから空行や注釈文，文字列リテラル部分を全て削除することによって対応する．(4)に対しては，予めソースコードから変数名を抽出しておき，登場順に固定の名称に置換していくことによって対応する．(5)に対しては，要素の出現順を問わない類似度算出手法を用いているため，新たな対応は必要ない．

4. 評価

4.1. プログラム演習課題への適用結果

本学の学部 2 年生を対象とした初等 Java プログラミング講義「インターネットプログラミング」の授業において，実際に学生が提出したソースコードのうち，コンパイルが成功しプログラムが動作したものを使用し，本システムの手法により類似度算出を行った．課題内容は以下に挙げる 3 つである．

課題1.「4 つの整数値を読み込み，値の大きい順に 2 つの整数を出力せよ」(99 ファイル，各 40 行程度)

課題2.「三角形の 3 頂点の座標をメンバとし，三角形の周囲の長さ，面積，重心を求めるメソッドを持つクラスを作成せよ」(82 ファイル，各 70 行程度)

課題3.「乱数によって発生させた 24 個の数字でビンゴカードを作成せよ」(114 ファイル，各 90 行程度)

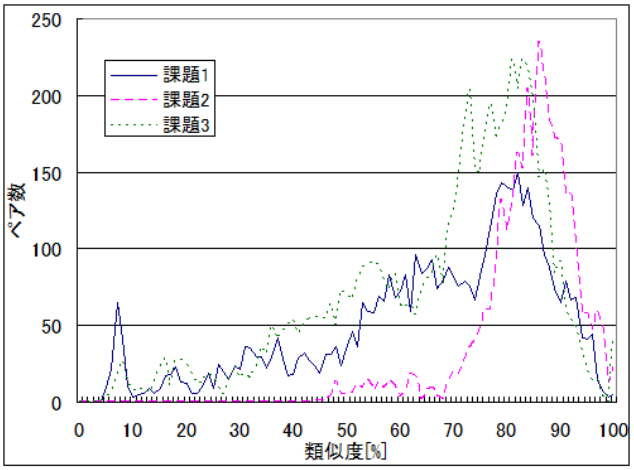

図 6　実際の演習課題からの類似度算出結果

このとき，それぞれの課題において目視による作業で確認したところ，以下の件数の盗用が認められた．

課題1.　23 ファイル，18 通りの組み合わせ
課題2.　16 ファイル，21 通りの組み合わせ
課題3.　32 ファイル，49 通りの組み合わせ

これら盗用が認められた組み合わせに対する類似度算出結果を図 7 に示す．

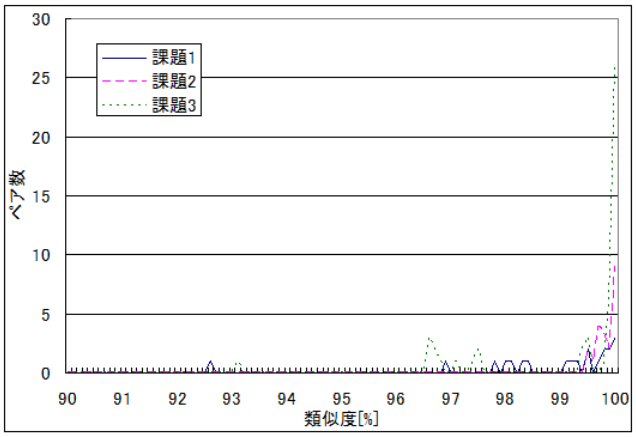

図 7　盗用が認められたペアの類似度分布

図 6，図 7 より，盗用が行われている場合の組み合わせは高い類似度を示している事がわかる．これは，盗用を行う学生はプログラミングが未熟であり，変数名の変更など小規模な偽装のみを行う例がほとんどであったためと考えられる．

また，類似度の低い組み合わせに関しては，特に盗用の疑いがある組み合わせは見受けられなかった．以上により，本システムの手法が盗用の発見に有効的に働く事が確認できた．

4.2. テストデータによる評価

4.2.1. テストデータの作成

偽装を施したテストデータを人工的に生成することにより多種類用意し，偽装元のデータとの類似度を算出することにより，偽装に対して本研究の手法がどれだけ有効的であるかを定量的に評価した．

まず，4.1 項の課題 1 で類似度算出に使用したソースコードのうち，偽装の認められなかった 20 ファイルに対して偽装を 10 回施したテストデータを，ソースコード 1 つにつき 5 組ずつ，計 100 組作成し，偽装前のソースコードとの類似度を算出した．

この時の偽装は，プログラムの実行結果に影響しない以下の 5 つの偽装をランダムに 10 回ずつ施した．

(1) 変数名の変更
(2) 冗長な変数宣言の追加
(3) 冗長な代入処理の追加
(4) 結果に影響しないような命令の順序入れ替え
(5) データ型のより広い型への変換(int→long など)

4.2.2. テストデータによる類似度算出結果

課題 1 で用いたソースコード群と，作成したテストデータに対してそれぞれソースコードから類似度算出を行った結果のうち，類似度 0.9 以上だった組み合わせのヒストグラムを図 8 に示す．なお，テストデータを用いた場合の類似度算出結果が 0.9 を下回る例はなかった．

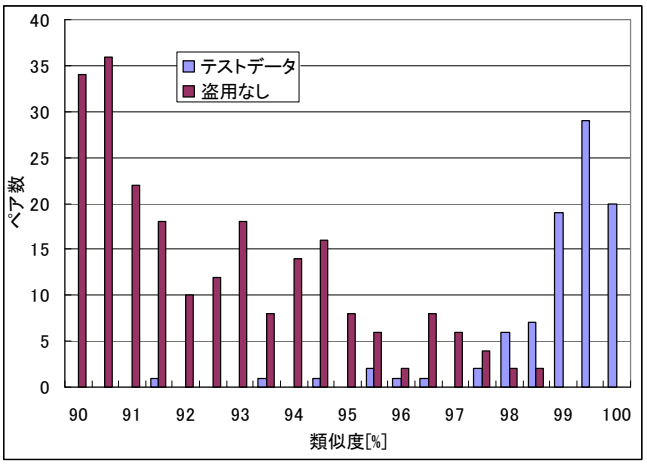

図 8　テストデータを用いた類似度算出結果

図 8 より，盗用されたプログラムは算出される類似度が高く，本システムが盗用の発見に有効に働いていることが分かる．また，この時，ある基準値以上の類似度を示したペアを盗用と定めた時の盗用の判断率，誤判断率を図 9 に示す．

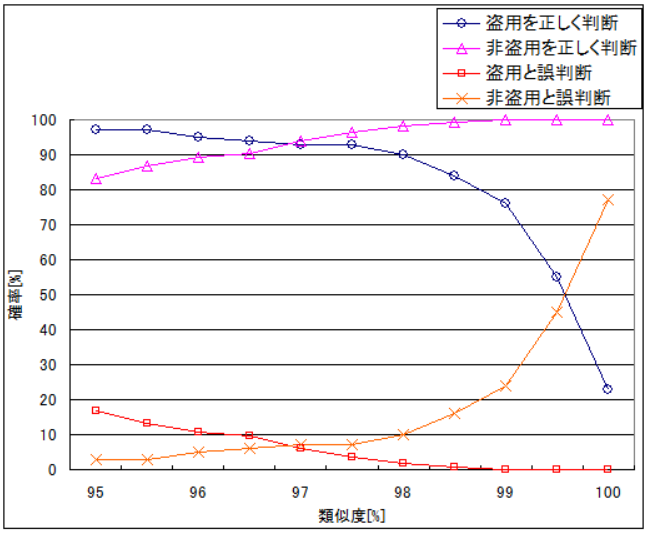

図 9　基準値による盗用の判断率，誤判断率

図 8，図 9 より，盗用と非盗用の両方を正しく判断可能である類似度 98 [%]付近が，盗用を判断する一つの基準となりうることが分かる．しかしながら，プログラミング演習課題の内容や規模，対象プログラミング言語などの条件によって算出される類似度は大きく異なるため，類似度算出結果を踏まえ，様々な課題設計に応じた採点者の判断が必要となる．

4.3. 既存研究との比較

2 章で述べた既存研究 JPlag と，本研究の手法で算出した類似度を比較，評価する．4.1 項の 3 つの課題のソースコードのうち盗用が認められなかった 30 件ずつを無作為に抽出，各課題に対して 435 通りのペア間の類似度を算出し，実際の授業で提出されたソースコード群に対してどのような類似度の算出結果の違いがあるか比較した結果を図 10 に示す．

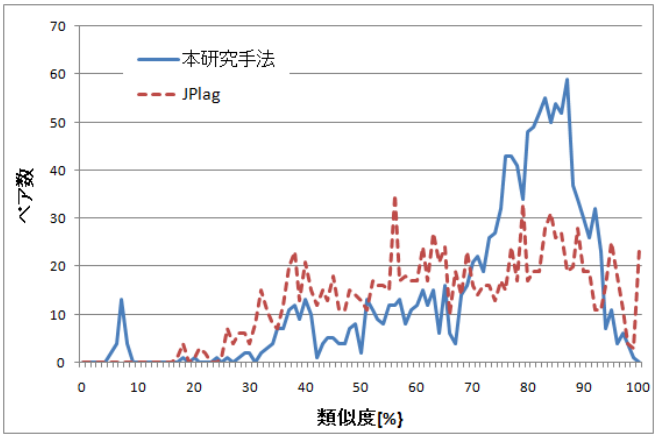

図 10　非盗用ペアに対する類似度算出結果比較

また，7.1 項の 3 つの課題の盗用が認められたペアに対する類似度算出結果のヒストグラムを図 11 に示す．

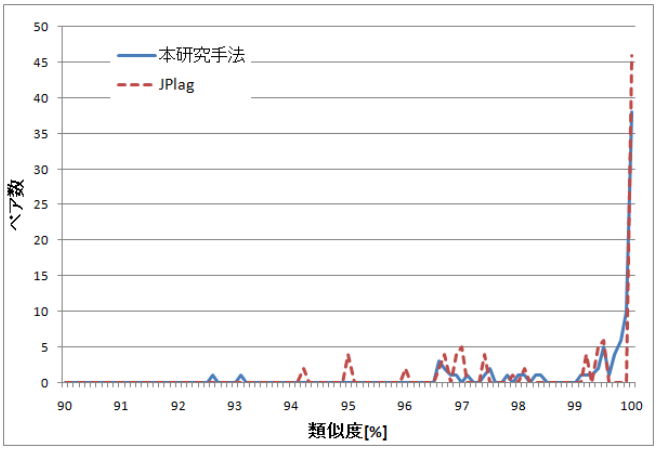

図 11　盗用ペアに対する類似度算出結果比較

図 10，図 11 より，JPlag による類似度算出結果は本研究による算出結果よりも算出された類似度が高くなる場合が多いことが分かる．JPlag の手法は偽装に対して有効的であるが，反面，盗用が存在しない場合に対しても類似度が一様に高くなってしまっている．このことから，JPlag の場合はより盗用を発見しやすくなる反面，盗用でないペアに対しても盗用であると誤判断してしまう場合が多いと言える．

課題 1,2,3 に対して類似度 98.0[%]を盗用の有無の閾値としたとき，本研究と JPlag を用いた場合の盗用の判断率，誤判断率は表 1 のようになる．

表 1　課題 1～3 の盗用の判断率

	本研究手法	JPlag
盗用を正しく判断	73 ペア (85.9%)	65 ペア (74.7%)
非盗用を正しく判断	1300 ペア (99.6%)	1271 ペア (97.4%)
盗用と誤判断 (実際は非盗用)	5 ペア (0.4%)	34 ペア (2.6%)
非盗用と誤判断 (実際は盗用)	14 ペア (14.1%)	22 ペア (25.3%)

本システムの手法のほうが盗用の誤判断率が低くなっており，本研究の手法が有効的であると言える．特に，盗用ではないのに盗用だと判断してしまう誤判断は，不正行為を行っていない学生に疑いを掛けてしまうことから一般に盗用を見逃してしまう誤判断よりも問題が大きい．よって盗用ではないのに盗用だと判断してしまうという誤判断の割合を減らすことがより重要だと考えられる．

4.4. 偽装対策手法の効果

4.1 項で使用した課題 1，課題 2，課題 3 のデータに対して，偽装対策について行った場合，行わなかった場合のそれぞれの結果を算出した結果を図 12 に示す．

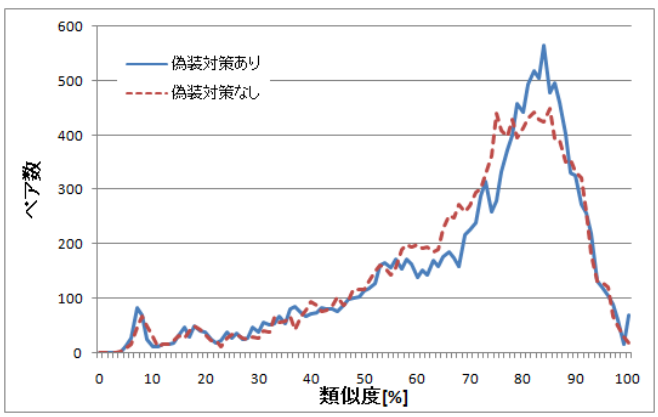

図 12　偽装対策の有無による算出結果の比較

また，課題 1，課題 2，課題 3 の盗用が認められたペアに対する，偽装対策の有無による類似度算出結果のヒストグラムを図 13 に示す．

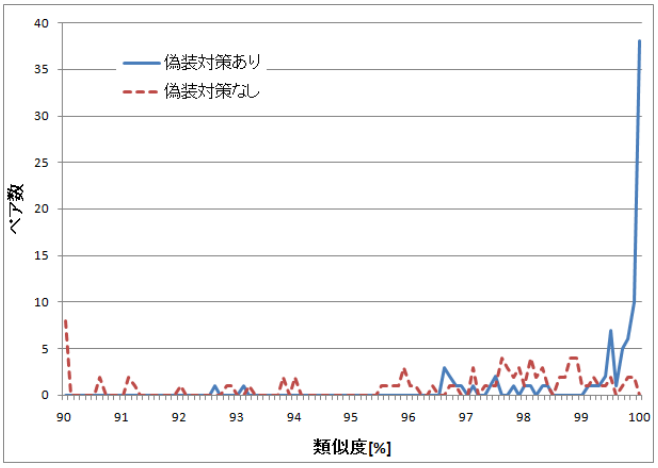

図 13　盗用ペアに対する偽装対策の有無の比較

図 12 より，偽装対策を行った場合，類似度算出の結果が盗用の有無を問わず高くなる傾向があることが分かる．これは，偽装対策を行うことによってソースコードの文字数が減り，N-gram 要素のベクトルが類似しやすくなっているのが原因と考えられる．

しかしながら，図 13 より，偽装が行われている場合においての類似度の上昇率が高く，盗用の有無が明確になり易いことから，本手法が盗用発見に有効的に働いていると言える．

4.5. システムの実行速度

本システムは，提出されたソースコードの数だけコンパイル，実行，テスト入力，自動採点を繰り返し行い，そして全てのソースコードの組み合わせに対して類似度算出を必要があるため，解析が完了するまでに相当の処理時間を要する．

4.1 項の課題 1 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 2 に示す．

表 2 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]
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

表 2 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，手作業によって同様の作業処理を行うと，1 ファイルにつき一連の作業に 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，本システムの使用によって大幅に採点作業の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比例しているが，類似度算出の処理時間はファイル数の 2 乗に比例して増加している．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．提案手法は 1 回あたりの計算量が比較的少ない手法を用いているため，ファイル数が 250 以下の場合は全体の処理時間に占める類似度算出の割合は 16%以下であり，大学の授業で使用する限り十分に実用的であるといえる．

5. 今後の課題

今後は，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検討し，盗用の発見と計算時間という観点から比較する必要がある．

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 文に書き換えるなどの類似命令への置き換えが行われる．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命令を定めるなどの手法を検討する必要がある．

また，今後は本システムを運用していく中で，機能の追加やインターフェースの改善を図る必要がある．

参　考　文　献

[1] X. Chen, M. Li, B. Mckinnon, and A. Seker. A theory of uncheatable program plagiarism detection and its pratical implementation. Manuscript, 2002.

[2] 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

[3] 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 I T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

[4] Prechelt, L.; Malpohl, G.; Philippsen, M. "JPlag: Finding plagiarisms among a set of programs," Technical Report, University of Karlsruhe, Department of Informatics, 2000. in computer program and other texts. In Proc. 27th SCGCSE, pages 130.134, 1996.