NAIST-IS-MT1051067

修士論文

複合圧縮度によるソースコード流用の検出

田中 智也

2012年 2月 2日

奈良先端科学技術大学院大学
情報科学研究科 情報システム学専攻

本論文は奈良先端科学技術大学院大学情報科学研究科に
修士 (工学) 授与の要件として提出した修士論文である。

田中 智也

審査委員：

松本 健一 教授　　（主指導教員）
藤川 和利 教授　　（副指導教員）
門田 暁人 准教授　（副指導教員）

複合圧縮度によるソースコード流用の検出*

田中 智也

内容梗概

本論文では，2つのソースコード間での流用の検出を目的として，ソースコード圧縮の度合いを評価する新しい尺度「複合圧縮度」を提案すると共に，提案尺度に基づく流用検出の実用性を実験的に評価する．一般に，重複する文字列を多く含むソースコードを圧縮すると，そのサイズは大きく減少する．この性質を利用して，提案尺度では，対象とする2つのソースコードそれぞれの圧縮後のサイズから，それら2つを連結した状態での圧縮後のサイズを差し引くことで，対象ソースコード間での流用の有無を評価する．なお，圧縮には，辞書式圧縮法であるLZMAアルゴリズムを用いる．CもしくはC++で記述されたオープンソースソフトウェア (Open Source Software: OSS) から作成した190組のソースコードペアに提案尺度を適用した結果，複合圧縮度を用いたロジスティック回帰モデルにより，適合率0.96，再現率0.79，F値0.87の判別精度が得られた．近年，商用ソフトウェアの開発では，開発コストの削減やソフトウェアの高信頼性確保を目的として， OSSの流用が増えている．流用においては，ソースコードの利用や改造に関する規則や制限（ライセンス）の遵守が求められるが，ソースコード管理が煩雑になると確認漏れが起こり，出荷後にライセンス違反が指摘される事例が増えている．提案尺度は，ソースコード流用の有無をソフトウェア出荷前に確認することを容易にし，意図しない，あるいは，故意のライセンス違反の防止に大きく貢献することが期待される．

* 奈良先端科学技術大学院大学 情報科学研究科 情報システム学専攻 修士論文, NAIST-IS-MT1051067, 2012年2月2日.

キーワード

プログラム圧縮，著作権保護，オープンソースソフトウェア，ソフトウェアライセンス

Detecting the Source Code Reuse Based on Compound Compression Metrics*

Tomoya Tanaka

Abstract

To detect the code reuse between two source code, this thesis proposes and evaluates "compound compression metrics," which evaluate the degree of souce code compression within two source code. Generally, when we compress source code containing duplicated code portions, its compression ratio becomes low. Therefore, we could identify whether or not source code A reused source code B by subtracting the compressed size of concatinated source code A + B from the sum of A's compressed size and B's compressed size. As a compression method, we used the LZMA algorithm, which is one of the commonly used dictionary compression algorithms. As a result of experinental evaluation with 190 pairs of Open Source Software (OSS) projects whose source code are written in C/C++, the prediction performance of a logistic regiression model using the proposed metrics was 0.96 in precision, 0.79 in recall and 0.87 in F value.

In recent software development, programmers often reuse source code of OSS as a part of their product to increase the productivity. To comply with OSS licenses, now it has become an important mission for software companies to inspect their software product to make sure not including unaware OSS source code in it. The proposed metrics can make it easy to inspect software products before releasing it so that OSS license violation is prevented.

* Master's Thesis, Department of Information Systems, Graduate School of Information Science, Nara Institute of Science and Technology, NAIST-IS-MT1051067, February 2, 2012.

Keywords:

Program Compression, Protection of Copyright, Software License, Open Source Software

目 次

1. はじめに **1**

1.1 背景 1

1.2 目的とアプローチ 3

1.3 本論文の構成 4

2. 準備 **5**

2.1 用語の定義 5

2.1.1 Open Source Software 5

2.1.2 ソースコード流用 6

2.1.3 プログラム圧縮 8

3. 複合圧縮度 **9**

3.1 ConSize 9

3.2 ConRate 11

4. 評価実験 **12**

4.1 実験環境 12

4.2 ロジスティック回帰モデル 14

4.3 評価指標 15

4.4 判別制度評価実験 16

4.4.1 実験手順 16

4.4.2 実験結果 18

4.4.3 考察 24

4.5 大規模流用を対象とした判別精度評価実験 26

4.5.1 実験手順 26

4.5.2 実験結果 27

4.5.3 考察 31

4.6 実行時間の計測 33

4.6.1 実験手順 34

4.6.2 実験手順 35
4.6.3 実験結果 36

5. 関連研究 **37**
5.1 プログラム圧縮を用いたソースコード流用検出に関する研究 . . . 37
5.2 ソフトウェア間におけるコードクローン含有率に基づく手法 . . . 38
5.3 ソフトウェアバースマーク 38

6. おわりに **41**

謝辞 **43**

参考文献 **45**

付録 **48**

A. 評価実験における実験対象ソフトウェアの一覧 **48**

図 目 次

1 過去 3 年間の IT ビジネスにおける OSS プロジェクト数の変化 [9] 1
2 オープンソースソフトウェアのライセンスの種類 7
3 複合圧縮度の概念図 10
4 実験対象プログラムのファイルサイズ分布 13
5 正解集合の作成手順 17
6 ConSize を用いたソースコード流用の判別精度 19
7 ConRate を用いたソースコード流用の判別精度 19
8 ロジスティック回帰モデルの判別値を用いたソースコード流用の判別精度 21
9 最長コードクローン長が 100 以上のものを対象とした，多重ロジスティックモデルの判別値を用いたソースコード流用ありの結果 . 28
10 最長コードクローン長が 150 以上のものを対象とした，多重ロジスティックモデルの判別値を用いたソースコード流用ありの結果 . 29
11 最長コードクローン長が 400 以上のものを対象とした，多重ロジスティックモデルの判別値を用いたソースコード流用ありの結果 . 29
12 最長コードクローン長が 600 以上のものを対象とした，多重ロジスティックモデルの判別値を用いたソースコード流用ありの結果 . 30

表 目 次

1 作成した正解集合 12
2 ConSize，ConRate を用いたソースコード流用判別における Precision，Recall の交点の値を閾値とした結果 20
3 ConSize の誤検出リスト 20
4 ConSize の未検出リスト 21
5 ConRate の誤検出リスト 22
6 ConRate の未検出リスト 23

7　ロジスティック回帰モデル (式 (8)) の判別値 P の閾値 0.5 における結果 23
8　ConSize，ConRate，およびロジスティック回帰モデルの判別値を閾値に用いた場合の検出数，未検出数，誤検出数 24
9　最長コードクローン長による区分を行った正解集合 27
10　ロジスティック回帰モデルの判別値 P の閾値 0.5 における結果 . . 30
11　最長コードクローン長が一定以上の要素を対象とした判別精度評価実験結果 31
12　最長クローン長 600 における誤検出・未検出リスト 32
13　実験対象ソフトウェアの一覧 48

1. はじめに

1.1 背景

近年，商用ソフトウェアの開発において，開発コストの削減やソフトウェアの高信頼性確保を目的として，オープンソースソフトウェア (Open Source Software: OSS) が流用される場合が増えている．例えば，IDC Japan の調査によれば，2011 年に IT ベンダーを対象に行った OSS の採用動向調査において，過去 3 年間で OSS を使用したプロジェクト数が増加したという回答は 37.3%に達している [9].

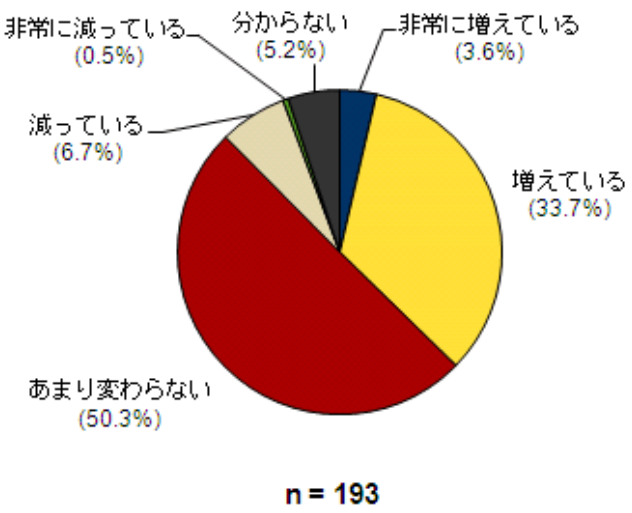

図 1 過去 3 年間の IT ビジネスにおける OSS プロジェクト数の変化 [9]

OSS は，文字通り，ソースコードが広く公開されているソフトウェアで，設計情報やテストデータなども合わせて公開されている場合もある．必要な機能や性能を有する OSS を入手することができれば，多少の修正が必要だとしても，全てを自前で開発するよりもコストを低く抑えることが出来る．また，運用実績が豊富な OSS であれば，不具合が含まれる可能性は低く，ソフトウェアの高信頼化にも貢献する．

ただし，OSS の流用には，ライセンス違反のリスクがある．一般に，OSS にはその利用や改造に一定の規則や制限が課されている．例えば，GNU General Public License(GNU GPL)[4] のソフトウェアは複製や再頒布・改変を行うことを認められているが，改変を行った場合そのソースコードを公開する必要がある．また，これを流用してソフトウェアを開発した場合，開発されたソフトウェアについてもそのソースコードを公開しなければならない．一方，Lesser General Public License(LGPL)[5] のソフトウェアを流用して開発を行っても，開発されたソフトウェアのソースコードを公開する必要はない．ただし，公開されているソースコードを改変して利用した場合は，その部分のソースコードを公開する必要があると定めている．このように，OSS を流用しようとする者は，そうしたライセンスに従わなければならない．しかし，ソフトウェア開発の一部が外注されるなどによりソースコード管理が煩雑になると，確認漏れが起こる場合がある．ましてや，開発者のモラルに問題がある場合には，ライセンス違反が頻発することになる．理由はともあれ，OSS 流用におけるライセンス違反は，ソフトウェア開発を行う組織にとって，法的責任を伴う大きなリスクとなりつつある．

ライセンス違反の例としては，SCE が発表した PS2 用のゲーム「ICO」の内部に GPL ライセンスのコードが流用されていた事例 [20] や，Microsoft が外部の業者に開発を委託した Windows7 への Upgrade 支援ツールの中に GPL ライセンスのコードが混入していた事例 [10] などが挙げられる．

ライセンス違反による法的リスクを回避する一つの方法として，ソフトウェア出荷前にソースコードにおける流用の有無を確認することが考えられる．最も単純な方法は，流用の可能性がある OSS と対象ソフトウェアのソースコードの文字列比較である．コードクローン検出技術 [8] を用いれば，流用時に多少の変更が加えられたコードであっても，流用コードとして特定することができる．Monden ら [16] の事例では，コードクローンメトリクスを用いた流用検出の精度は，平均の Precision が 95.5 %，Recall が 87.1 %であり，100 %の精度ではないものの，人手による流用検出を補う手段として有望である．ただし，コードクローンは現在盛んに研究が行われている段階であり，その検出方法についてコンセンサスが得られているわけではない．また，コードクローン検出ツールが利用できるプログ

ラミング言語や環境は限定されていることや，検出ツールに与えるパラメータが複雑であることから，現状では，誰もが使えるとはいえない．

ソースコード流用を確認する新しいアプローチとなり得るものとして最近注目されているのが，プログラム圧縮技術を用いる手法である．Hemel ら [7] は，ビット列中に重複を含むと圧縮率が低くなる（つまり，よりたくさん圧縮できる）ことに着目し，プログラム A がライブラリ B を含む場合に，A の圧縮後のサイズと，A と B を連結して圧縮した場合の圧縮後のサイズがほぼ等しくなるという予想を立てた．そして，代表的な OSS の一つである Subversion を対象とし，プログラム圧縮により Subversion が libsqlite3.a を含んでいることの判別に使えることを示した．ただし，Subversion には他にも数多くのライブラリが含まれており，それらの検出精度については評価していない．また，より一般的に，二つのプログラム間（ソースコード間）での流用の有無が確認できるかどうかについては議論されていない．

1.2 目的とアプローチ

本論文の目的は，ソースコード流用をプログラム圧縮技術で検出する具体的な方法とその実用性を明らかにすることで，OSS 流用におけるライセンス違反の回避を容易にすることである．そのため，新たに，プログラム圧縮度の評価尺度を提案すると共に，流用検出の精度を実験的に評価する．提案する評価尺度は，2 つのプログラム（ソースコード）を連結した状態も対象として圧縮度を評価するものである．従来，圧縮度を評価する尺度として「圧縮率」が用いられてきたが，これは圧縮アルゴリズムの性能評価のためのものであり，圧縮アルゴリズムが適用される個々のファイルが計測対象であった．これに対して，提案する評価尺度は，２つのプログラム（ソースコード）間での流用検出を目的とするものである．そこで，従来にはない,「連結した状態」も評価の対象としている．更に，サイズ（行数）が大きく異なるプログラム（ソースコード）間でも圧縮どの比較が出来るよう，サイズ当たりの圧縮度を評価する尺度も提案する．評価実験は，C もしくは C++で記述された OSS から作成した 190 組のプログラム（ソースコード）ペアを評価対象とする．圧縮には，辞書式圧縮法である LZMA アルゴリズムを用いる．

ペア間の流用の有無を表す正解集合は，コークローン検出ツール CCFinderX[1] によって検出された長さ 30 トークン以上のクローン全てについて目視で確認することで作成する．評価尺度の精度は，正解集合に対する Precision と Recall によって評価する．

1.3 本論文の構成

本論文の構成は次の通りである．2 章では，評価尺度提案の準備として，対象とするソフトウェアやソースコード，プログラム圧縮の概念について説明し，用語の定義などを行う．3 章では，プログラム圧縮度を評価する 2 つの新しい尺度を提案し，それらに基づくソースコード流用検出法について説明する．4 章では，提案した評価尺度とそれに基づく流用検出の精度を明らかにするための実験について述べると共に，実験結果に基づき，プログラム圧縮度を用いたソースコード流用検出法の判別精度について考察を行う．5 章では，関連研究の紹介することで，本研究の学術的意義や当該分野における位置づけを明確にする．最後に 6 章において，本論文のまとめと今後の課題について述べる．

2. 準備

本章では，本論文中において用いられる語句の定義について説明する．

2.1 用語の定義

2.1.1 Open Source Software

Open Source Software(OSS) とは，ソフトウェアの設計図にあたるソースコードを，インターネットなどを通じて無償で公開し，誰でもそのソフトウェアの改良，再配布を行うことのできるソフトウェアである [11]．Open Source Initiative(OSI) は，Open Source Definition(OSD) において，OSS に求められるライセンス条件を次のように定義している [18].

1. 再頒布の自由
    - 販売・無料頒布の制限禁止
    - ロイヤリティ要求の禁止
2. ソースコード
    - ソースコード入手の保障
3. 派生ソフトウェア
    - 改変の許可
    - 派生ソフトを同じライセンスで頒布することの許可
4. 作者のソースコードの同一性保持
5. 個人・グループに対する差別禁止
6. 利用分野に対する差別禁止
7. 再配布時におけるライセンス追加の禁止

- 機密保持契約の追加は禁止

8. 特定製品でのみ有効なライセンスの禁止
9. 他のソフトウェアを制限するライセンスの禁止
10. ライセンスは技術に対して中立でなければならない

これらの条件を満たす，OSI 認定ライセンスは 40 以上にも上り多種多様である．ここで，ライセンスのうち，OSS の定義の中で特に重要な 3 項目である (1) 再配布の自由，(2) ソースコード入手性の保証，(3) 改変・再配布の許可，を満たすものは，ソースコードの公開条件 (コピーレフト性) に着目すると以下の 3 つに類型化できる [13](図 2).

- GPL 型ライセンス
  フリーウェア財団が作成した，最も公開義務の制約が強いライセンスである．改変したソースコードだけでなく，一体となって動作するソフトウェアは必ず公開しなければならない.
- MPL 型ライセンス
  ＷｅｂブラウザＭｏｚｉｌｌａのライセンスである．ＧＰＬとは異なり，プラグイン等，ソースコードが独立したソフトウェアは公開の義務を負わない.
- BSDL 型ライセンス
  BSD 系ＵＮＩＸで使われているライセンスである．再配布時に著作権表示と再配布条件表示，無保証・免責宣言を行うことのみを条件とする，極めて制限の緩いライセンスである.

2.1.2 ソースコード流用

ソースコード流用とは，既存のプログラムのソースコードの一部または全体をコピー＆ペーストして開発中のプログラムのソースコードに組み込む行為のことである．また，次のいずれかの事象が起きた場合に，プログラム A とプログラム B が「ソースコード流用の関係にある」という.

1. プログラム A にプログラム B の一部または全体を組み込んだ.
2. プログラム B にプログラム A の一部または全体を組み込んだ.
3. プログラム A とプログラム B の両方において，既存のプログラム C を組み込んだ.

プログラム A とプログラム B がソースコード流用の関係にある場合，A と B の間には重複したソースコード（コードクローン）が存在することになる．そのため，従来，コードクローンによるソースコード流用の検出が行われてきた [16]. ただし，A と B の間にコードクローンが存在するからといって，A と B がソースコード流用の関係にあるとは限らない．コードクローンは，コピー＆ペースト以外によっても生じるためである．例えば，定型的な命令列，自動生成コード，偶然の一致などによってもコードクローンは生じる．そのため，ソースコード流用の有無を判定するには，コードクローンがコピー＆ペーストによって生じたものであるのか，そうでないのかについて，人間による判断が必要となる．本論文では，多数のプログラムの組について，人間による判断に基づいて流用の有無の正解集合を求め，それらを機械的に（プログラム圧縮によって）精度良く判別することを目指す.

類型	複製･再頒布可能	改変可能	改変部分のソースコード公開が必要	組み合わせた他ソフトウェアのソースコード公開も必要
GPL型	○	○	○	○
MPL型	○	○	○	×
BSDL型	○	○	×	×
フリーウェア	○	×	–	–
プロプラエタリソフト	×	×	–	–

図 2 オープンソースソフトウェアのライセンスの種類

2.1.3 プログラム圧縮

従来，プログラム圧縮は，組込み機器等における記憶容量の節約やデータ転送量の削減などを目的として，様々な方法が提案されてきたが [14]，本論文では，テキストファイルの圧縮と同様に，プログラム全体をビット列とみなして可逆圧縮することをプログラム圧縮と呼ぶ．

従来，圧縮の度合いを評価するための指標としては圧縮度が存在している．ここで，圧縮率とは，元の情報の大きさに対して，圧縮後の情報がどの程度のサイズになっているかを示す数値であり，式 (1) で与えられる．

$$\text{圧縮率} = \frac{\text{圧縮後の情報量}}{\text{圧縮前の情報量}} \tag{1}$$

圧縮前の情報量に対して圧縮後の情報量が小さい場合，圧縮前の情報をより小さく表現することができているといえる．そのため，圧縮率は，値が小さいほど高性能な圧縮であることを示す指標である．

3. 複合圧縮度

本論文では，ファイル圧縮を用いたソースコード流用の検出の実用性について検討を行う．検討を行うために，まず，ファイル圧縮を用いたソースコード流用の検出の精度を評価するための評価尺度である複合圧縮度を提案する．

複数のプログラムを圧縮する場合，それぞれ個別に圧縮する方法と，それらを連結して圧縮する方法が考えられる．ここで，前者を個別圧縮，後者を連結圧縮とする．複合圧縮度は，個別圧縮した場合のファイルサイズの合計と，連結圧縮したファイルサイズとの差である (図 3)．

プログラム間にソースコード流用が存在している場合，それらは重複している内容であるため，両者を連結して圧縮する場合において，圧縮量が大きくなると考えられる．これにより，個別圧縮後の合計ファイルサイズよりも，連結圧縮後のファイルサイズの方が小さくなるため，複合圧縮度の値は高くなる．このことから，複合圧縮度が高い値を示すと，ソースコード流用が含まれている可能性が高まるため，ソースコード流用検出に役立つことが期待できる．

次に，複合圧縮度の具体的な評価尺度である ConSize，ConRate を提案する．以降，各評価尺度の定義とソースコード流用における狙いを述べる．

3.1 ConSize

ConSize は，個別圧縮後の合計ファイルサイズから，連結圧縮後のファイルサイズを引いた圧縮度であり，式 (2) で表される．ここで，$Comp(X)$ は圧縮後のファイルサイズを，A, B はプログラムを，$A + B$ はプログラム A, B の連結を示す．

$$ConSize = Comp(A) + Comp(B) - Comp(A + B) \tag{2}$$

プログラム A, B 間に内容の重複があった場合，連結圧縮を行うことで $Comp(A)$, $Comp(B)$ よりも $Comp(A + B)$ のサイズが大幅に小さくなるため，ConSize の値が大きくなる．このため，ConSize の値が高いほど，そのプログラムペア間にはソースコード流用が含まれている可能性が高いと考えられる．

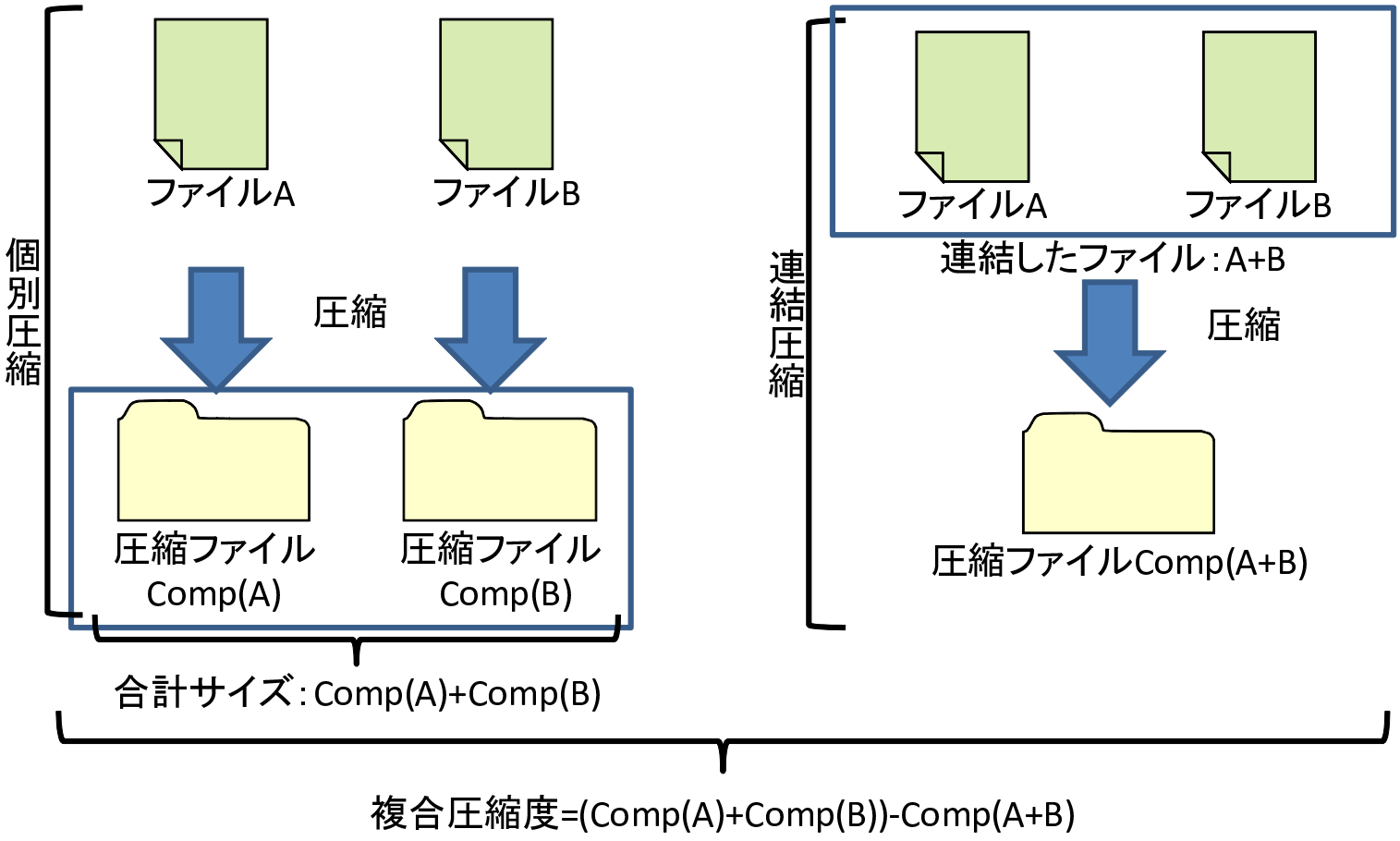

図 3　複合圧縮度の概念図

3.2 ConRate

ConRate は，ConSize を個別圧縮後の合計ファイルサイズで除算し，正規化したものであり，式 (3) で表される．

$$ConRate = \frac{Comp(A) + Comp(B) - Comp(A+B)}{Comp(A) + Comp(B)} \tag{3}$$

ConSize と同様に，プログラム A, B 間に内容の重複があった場合，連結圧縮を行うことで $Comp(A), Comp(B)$ よりも $Comp(A+B)$ のサイズが大幅に小さくなるため，ConRate の値が大きくなる．このため，ConRate の値が高いほど，そのプログラムペア間にはソースコード流用が含まれている可能性が高いと考えられる．また，ConSize を個別圧縮後の合計ファイルサイズで除算しているため，規模自体が大きく圧縮後もプログラムサイズが大きいままであり，ConSize がその影響を受けて高い値を示してしまうようなプログラム間に含まれる流用を判別する際にも，元々の規模の影響を受けにくく，安定して流用を判別できる可能性がある．

表 1 作成した正解集合

ソースコード流用の有無	組数
ソースコード流用あり	63
ソースコード流用なし	127

4. 評価実験

本章では，3 章で提案した複合圧縮度を評価尺度として用いることで，プログラム圧縮によるソースコード流用の判別実験を行う．まず，4.4 節では ConSize，ConRate，およびこの 2 値を独立変数としたロジスティック回帰モデルにより流用判別を行った場合の精度を確かめた．次に，4.5 節では大規模な流用 (流用のうち，最長コードクローン長が一定値以上のもの) を対象とし，ロジスティック回帰モデルにより流用判別を行った場合の精度を確かめた．そして，4.6 節では，従来のコードクローンメトリクスを用いた流用検出と比べて，どの程度コストを減らせるかを確認するため，プログラム圧縮の実行時間の計測を行った．

4.1 実験環境

3 章で提案した，複合圧縮度を用いたソースコード流用検出の精度を確かめるために，Free Software Foundation が提供する Free Software Directory[3] で公開されている GPL，LGPL の OSS を 20 件（190 組）使用した．これら OSS のリストを付録に示す．また，これら OSS のファイルサイズ分布を図 4 に示す．

また，これら 190 組について，ソースコード流用の有無の正解集合を人手により作成した．その結果を表 1 に示す．正解集合の作成手順は，4.4 節で後述するが，その際に，産業技術総合研究所の神谷年洋博士が開発した CCFinderX[1] を用いた．CCFinderX は，トークン単位によるコードクローン検出ツールである．

実験に用いた環境を以下にまとめる．

- 実験対象ソフトウェア

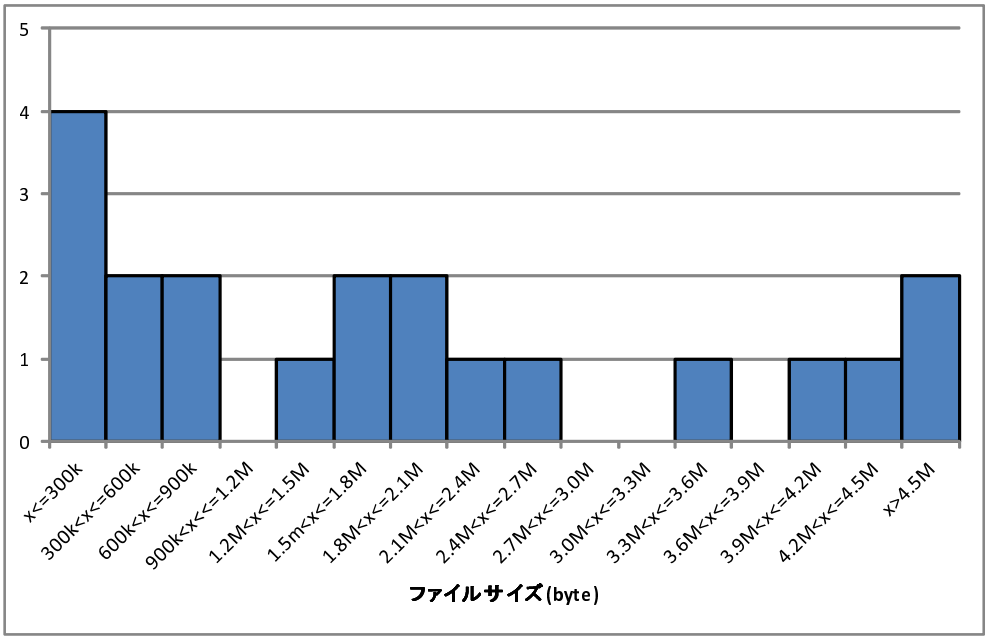

図 4 実験対象プログラムのファイルサイズ分布

- 件数：20
  - 最小ファイルサイズ：66,139 バイト (Avdbtools)
  - 最大ファイルサイズ：28,237,760 バイト (GIMP)
  - 平均ファイルサイズ：3,216,271 バイト
- ライセンス：GPL，LGPL
- ドメイン：Business，Audio，Game など
- 開発言語：C，C++

- クローン検出ツール
  - CCFinderX

- 圧縮ツール
  - Lzip

本実験で用いた圧縮ツール Lzip には，辞書式圧縮アルゴリズム LZMA が採用されており，ZIP，gzip，bzip2 など他の圧縮ツールよりも高い圧縮率が得られたため採用した．これにより，重複した内容を確実に圧縮することができ，圧縮度合いが不十分であることによる評価尺度への悪影響を防ぐことができる．

4.2 ロジスティック回帰モデル

ロジスティック回帰モデルとは，与えられた入力データから独立変数の変動によって，従属変数が 2 値のどちらを取る確率が高いかを出力する回帰式を求める手法である．この場合，独立変数は ConSize，ConRate の 2 値であり，従属変数はソースコード流用ありか，ソースコード流用なしかのどちらかを取ることになる．ロジスティック回帰モデルは次の (4) 式で表される．ここで，b_0 は定数値，$b_1, b_2, \ldots, b_p$ は独立変数 $X_1, X_2, \ldots, X_p$ にかかる係数である．また，$b_0, b_1, b_2, \ldots, b_p$ は，最尤法によって求めることができる．

$$
\begin{aligned}
P &= \frac{1}{1+exp(-\lambda)} \\
&= \frac{1}{1+exp\{-(b_0+b_1X_1+b_2X_2+\cdots+b_pX_p)\}}
\end{aligned} \tag{4}
$$

4.3 評価指標

ソースコード流用の判別精度を評価するために，本論文では Precision，Recall，F 値を用いる．本論文における Precision は，流用ありと判断したもののうち，実際に流用があったものの割合である．Precision は，流用ありの確率の高さを示すことができるため，流用のあり／なしが法廷で争われる場合などに有用である．

本論文における Recall は，実際に流用があるもののうち，流用ありと判別できたものの割合である．Recall は，流用のあるソースコードを漏れなく検出したい場合などに有用である．

Precision および Recall は，式 (5)，(6) で再現できる．

$$
Precision = \frac{\text{正検出数}}{\text{全検出数}} \tag{5}
$$

$$
Recall = \frac{\text{正検出数}}{\text{正解集合の要素数}} \tag{6}
$$

Precision と Recall は，トレードオフの関係にある．一般に，Precision を上ると Recall が低くなり，Recall を上げると Precision が低くなるため，状況に応じてどちらか一方を重視する場合がある．

判別性能を評価するために，Precision と Recall の調和平均である F 値が用いられる．F 値は，式 (7) で表される．

$$
F\text{ 値} = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{7}
$$

4.4 判別制度評価実験

4.4.1 実験手順

ソースコード流用の判別精度評価実験の手順を以下に示す.

1. 各 OSS ペアに対し，人手による分析を行い，正解集合を作成する.
2. 各 OSS ペアについて，個々に圧縮した場合のサイズと結合して圧縮した場合のサイズを記録する.
3. 記録したサイズに基づき，ConSize，ConRate を算出する．また，これら 2 値を独立変数に用いてロジスティック回帰モデルを作成する.
4. ConSize，ConRate，およびロジスティック回帰モデルの判別値に対して閾値を設定し，全てのプログラムペアに対して閾値以上のものを流用とみなす．その後，判別結果と正解集合を照らし合わせることで Precision，Recall を算出し，流用判別精度を評価する.

ここで，1. における正解集合は以下の手順で作成した (図 5)．作成した正解集合を表 1 に示す.

1. CCFinderX を用いてプログラム間のコードクローンを全て検出する．検出時の最小クローン長を 30 トークンとした.
2. コードクローンが全く検出されなかった場合，流用なしと判断する.
3. 最大コードクローンに対し，目視で流用であるかどうかを確認する.
4. 流用でないと判断した場合，次に長いクローンに対し同様の作業を行う.
5. 全てのクローンが流用でなかった場合，流用無しと判断する.

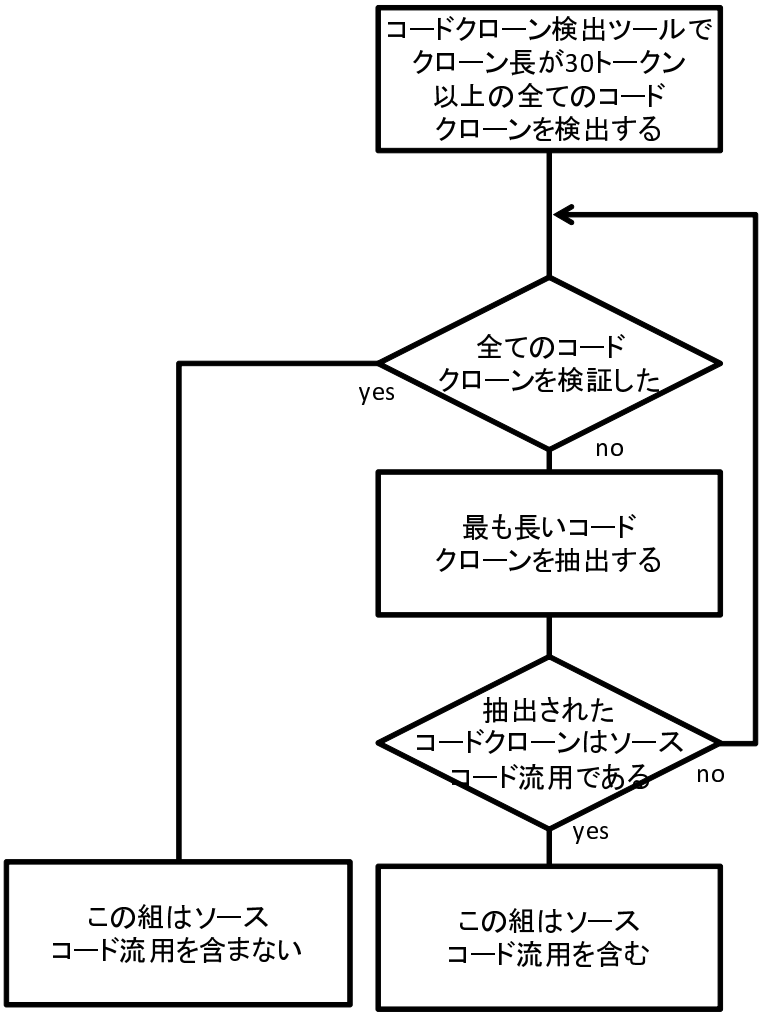

図 5 正解集合の作成手順

4.4.2 実験結果

ConSize を閾値として変化させた場合のソースコード流用の判別精度を図 6 に示す．図より，Precision と Recall が交差するのは，ConSize=9000 の点であり，このときの Precision と Recall は 0.82 であった（表 2）．その際，正解集合上では流用なしであるにも関わらず流用ありと判別してしまった誤検出は 12 件，正解集合上では流用ありであるにも関わらず流用なしと判別してしまった未検出は 11 件あった．誤検出であったプログラムペアを表 3 に，未検出であったプログラムペアを表 4 にそれぞれ示す．

ConRate を閾値として変化させた場合のソースコード流用の判別精度を図 7 に示す．図より，Precision と Recall が交差するのは，ConRate=0.013 の点であり，このときの Precision と Recall は 0.76 であり（表 2），ConSize の場合よりも低い値となった．また，このとき誤検出は 16 件 (表 5)，未検出は 14 件 (表 6) であった．

ConSize，ConRate を独立変数としたロジスティック回帰モデルを式 (8) に示し，判別結果を表 7 に示す．表より，Precision=0.96 と非常に高い値となった．その一方で，Recall=0.79 であり，ConRate，ConSize をそれぞれ用いた場合と遜色ない値となった．また，Precision と Recall の調和平均である F 値は 0.87 であった．

ロジスティック回帰モデルの判別値 P に対し，流用とみなす閾値を変化させた場合のソースコード流用の判別精度を図 8 に示す．図より，Precision と Recall が交差した点における Precision と Recall の値は約 0.85 であり，ConRate，ConSize を個別に用いた場合よりも高い精度が得られた．このことから，ConRate，ConSize のどちらか一方だけを用いるよりも，両方を用いて判別を行うことが有効であることが分かった．

$$P = \frac{1}{1 + exp\{-(-5.68 + 3.25e^{-4} \cdot consize + 197.2 \cdot conrate)\}} \tag{8}$$

表 8 に，Recall = 1.0 または Precision = 1.0 となるよう閾値を設定した場合における，(流用ありの）総検出数，正検出数，誤検出数，未検出数，Precision，Recall，F 値を示す．表より，ConSize，ConRate，ロジスティック回帰モデルのいずれの場合においても，Recall=1.0 では誤検出数がきわめて多くなることが分かった．一方，Precision=1.0 では，特に ConRate とロジスティック回帰モデルに

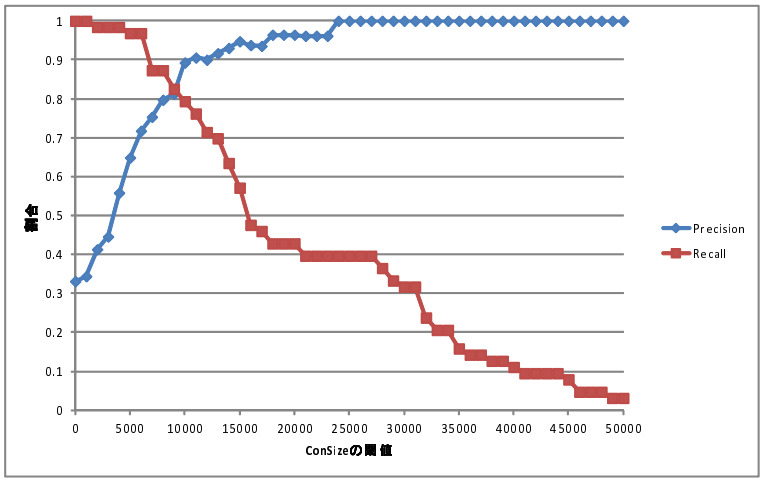

図 6 ConSize を用いたソースコード流用の判別精度

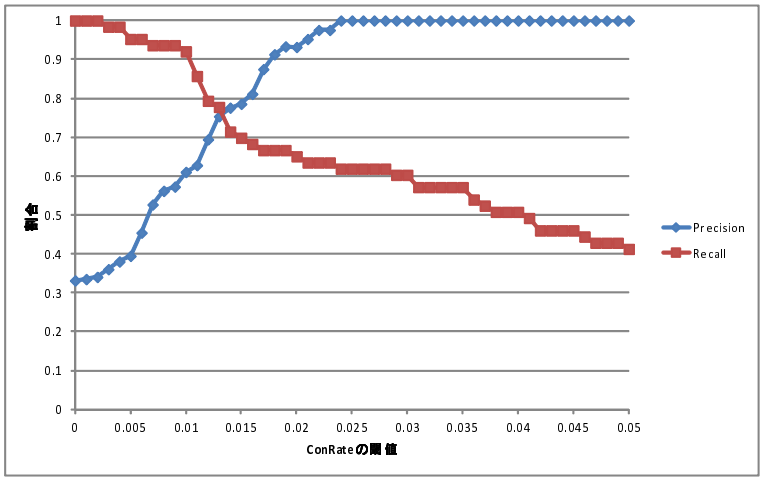

図 7 ConRate を用いたソースコード流用の判別精度

表 2 ConSize，ConRate を用いたソースコード流用判別における Precision，Recall の交点の値を閾値とした結果

	閾値	総検出数	正検出数	誤検出数	未検出数	Precision，Recall
ConSize	9000	64	52	12	11	0.82
ConRate	0.013	65	49	16	14	0.76

表 3 ConSize の誤検出リスト

プログラム A	プログラム B	サイズ A	サイズ B	サイズ差 A-B	サイズ比 B/A
gimp	cdrtool	28,237,760	4,273,311	23,964,449	6.60793469
gimp	gnubg	28,237,760	4,067,016	24,170,744	6.94311505
gimp	geomview	28,237,760	3,329,805	24,907,955	8.480304402
gimp	gnugo	28,237,760	2,413,223	25,824,537	11.70126424
gimp	glame	28,237,760	2,003,059	26,234,701	14.09731815
gimp	chemtool	28,237,760	1,986,249	26,251,511	14.21662641
gimp	easytag	28,237,760	1,454,918	26,782,842	19.408489
cinepaint	cdrtool	8,046,065	4,273,311	3,772,754	1.882864364
cinepaint	gnubg	8,046,065	4,067,016	3,979,049	1.978370628
cinepaint	geomview	8,046,065	3,329,805	4,716,260	2.416377235
cinepaint	glame	8,046,065	2,003,059	6,043,006	4.016888669
cdrtool	geomview	4,273,311	3,329,805	943,506	1.283351728

表 4　ConSize の未検出リスト

プログラム A	プログラム B	サイズ A	サイズ B	サイズ差 A-B	サイズ比 B/A
gimp	gbonds	28,237,760	514,290	27,723,470	54.906298
cinepaint	glabels	8,046,065	2,100,470	5,945,595	3.830602199
geomview	chemtool	3,329,805	1,986,249	1,343,556	1.676428786
geomview	easytag	3,329,805	1,454,918	1,874,887	2.288654756
geomview	danpei	3,329,805	625,874	2,703,931	5.320248165
geomview	battstat	3,329,805	261,989	3,067,816	12.709713
geomview	gmmusic	3,329,805	241,646	3,088,159	13.77968185
gnugo	glabels	2,413,223	2,100,470	312,753	1.148896676
glabels	glame	2,100,470	2,003,059	97,411	1.048631119
glame	cvsps	2,003,059	60,082	1,942,977	33.3387537
glame	gbonds	2,003,059	514,290	1,488,769	3.894804488

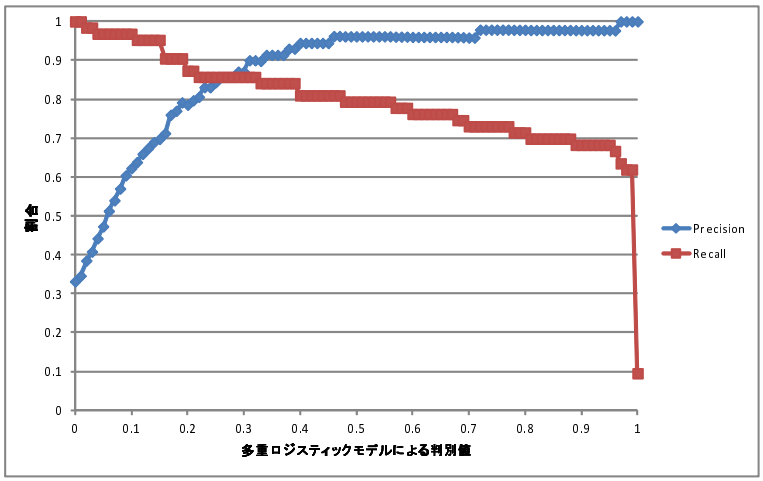

図 8　ロジスティック回帰モデルの判別値を用いたソースコード流用の判別精度

表 5 ConRate の誤検出リスト

プログラム A	プログラム B	サイズ A	サイズ B	サイズ差 A-B	サイズ比 B/A
glabels	gnofin	2,100,470	763,825	1,336,645	2.749936176
chemtool	aide	1,986,249	500,614	1,485,635	3.967625756
easytag	aide	1,454,918	500,614	954,304	2.906267104
gnofin	gbonds	763,825	514,290	249,535	1.485202901
gnofin	aide	763,825	500,614	263,211	1.525776347
danpei	gbonds	625,874	514,290	111,584	1.216967081
danpei	aide	625,874	500,614	125,260	1.250212739
gbonds	aide	514,290	500,614	13,676	1.027318453
gbonds	battstat	514,290	261,989	252,301	1.963021348
aide	battstat	500,614	261,989	238,625	1.910820683
aide	gmmusic	500,614	241,646	258,968	2.071683372
battstat	avdbtool	261,989	66,139	195,850	3.961187801
battstat	cvsps	261,989	60,082	201,907	4.360523951
gmmusic	avdbtool	241,646	66,139	175,507	3.65360831
gmmusic	cvsps	241,646	60,082	181,564	4.021936687
avdbtool	cvsps	66,139	60,082	6,057	1.100812223

表 6 ConRate の未検出リスト

プログラム A	プログラム B	サイズ A	サイズ B	サイズ差 A-B	サイズ比 B/A
gimp	glabels	28,237,760	2,100,470	26,137,290	13.44354359
gimp	gbonds	28,237,760	514,290	27,723,470	54.906298
cinepaint	glabels	8,046,065	2,100,470	5,945,595	3.830602199
cdrtool	glabels	4,273,311	2,100,470	2,172,841	2.03445467
cdrtool	gnugo	4,273,311	2,413,223	1,860,088	1.770789935
geomview	glame	3,329,805	2,003,059	1,326,746	1.662359921
geomview	chemtool	3,329,805	1,986,249	1,343,556	1.676428786
geomview	easytag	3,329,805	1,454,918	1,874,887	2.288654756
geomview	gmmusic	3,329,805	241,646	3,088,159	13.77968185
geomview	battstat	3,329,805	261,989	3,067,816	12.709713
geomview	danpei	3,329,805	625,874	2,703,931	5.320248165
glabels	glame	2,100,470	2,003,059	97,411	1.048631119
glame	gbonds	2,003,059	514,290	1,488,769	3.894804488
glame	cvsps	2,003,059	60,082	1,942,977	33.3387537

表 7 ロジスティック回帰モデル (式 (8)) の判別値 P の閾値 0.5 における結果

総検出数	正検出数	誤検出数	未検出数	Precision	Recall	F 値
52	50	2	13	0.96	0.79	0.87

表 8 ConSize，ConRate，およびロジスティック回帰モデルの判別値を閾値に用いた場合の検出数，未検出数，誤検出数

	ConSize		ConRate		ロジスティック回帰モデルの判別値	
	precision=1	recall=1	precision=1	recall=1	precision=1	recall=1
閾値	24000	1000	0.024	0.002	0.97	0.01
総検出数	25	183	39	185	40	182
正検出数	25	63	39	63	40	63
誤検出数	0	120	0	122	0	119
未検出数	38	0	24	0	23	0
Precision	1	0.34	1	0.34	1	0.35
Recall	0.40	1	0.62	1	0.63	1
F 値	0.57	0.51	0.76	0.51	0.78	0.50

おいては，未検出数よりも正検出数の方が多く，検出された流用について，法廷において流用ありを主張したい場合などに有用と考えられる．

4.4.3 考察

表 2 から，ConSize，ConRate それぞれについて，Precision，Recall の交点の値を閾値とした場合，ConSize の方が ConRate よりも Precision，Recall ともに高い値を示している．これは，式 (3) より，ConRate は割合によって表現される指標であるため，分母が小さいと値がぶれやすいことが原因であると考えられる．表 5 において，誤検出されたプログラムはいずれも実験対象プログラムの平均ファイルサイズを下回っていることから，本実験においても，ファイルサイズが小さなプログラムペアを，本来流用なしであるにも関わらず流用ありと判別してしまったと考えられる．

一方，表 3 より，ConSize を用いた流用判別では 12 件中 11 件が，実験対象プログラムにおいてファイルサイズが最大である GIMP と，2 番目に大きい Cinepaint を含んでいる．式 (2) より，ファイルサイズが大きいと ConSize の値も大きくなりやすいことに加えて，規模が大きいプログラムは多様なコードを含有しているため，流用でなくても類似したコードを圧縮できてしまうと考えられる．このことから，本実験においても，規模の大きなプログラムを含むプログラムペアを，本来流用なしであるにも関わらず流用ありと判別してしまったと考えられる．

また，本来流用があるにも関わらず流用なしと判別してしまった未検出について，表 4 より ConSize では 11 件中 5 件が，表 6 より ConRate では 14 件中 6 件が，Geomview を含むプログラムペアであった．この時，Geomview において正解集合作成時に流用ありと判断されたソースコード parse_tab.c には bison による自動生成コードである bison parser が含まれており，これが流用と判断された箇所であった．これは 2.1.2 項において定義した 3. のソースコード流用に該当する，特殊な形式での流用であった．更に，ソースコード中の他の箇所に流用は全く存在せず，加えて類似性も低かったため，ConSize，ConRate 共にあまり大きな値とはならず，流用ありであるにも関わらず流用なしと判別してしまったと考えられる．

表 7 では，ロジスティック回帰モデルの判別値として一般的に用いられる値 0.5 を閾値として流用の判別を行った．その結果，Precision=0.96，Recall=0.79，F 値=0.87 と，ConSize，ConRate をそれぞれ個別に用いて流用の判別を行うよりも高い精度で流用を判別できることが確認された．1.1 節で述べたとおり，コードクローンメトリクスを用いた流用検出の精度において，平均の Precision が 95.5%，Recall が 87.1%であったことから，Recall に関してはやや劣るが，Precision に関してはほぼ同等の値を示している．加えて，プログラム圧縮を用いる本手法は，複雑なパラメータを必要としないため，検出ツールを用いる場合よりもコストを低減できると考えられる．

表 8 から，Recall=1 となるような ConSize，ConRate，ロジスティック回帰モデルの判別値に基づく閾値を設定しても，正解集合においてソースコード流用なしと定義されている要素の大部分を流用ありと判断してしまうことが示されている．加えて，ロジスティック回帰モデルの判別値を表 8 のような値に設定するこ

とは一般的ではなく，汎用性に疑問が残る．

以上より，複合圧縮度は，ソースコード流用検出に役立つと考えられる．また，ConSize，ConRate を独立変数としたロジスティック回帰モデルを作成し，判別値 0.5 において評価することが望ましいが，どちらか一方のみを評価尺度として用いる場合，ConSize が望ましいといえる．

4.5 大規模流用を対象とした判別精度評価実験

4.4 節では，規模の大小を問わず，全てのソースコード流用を検出することを目的とした判別精度評価実験を行った．ここでは，大規模な流用のみを対象として，判別精度の評価実験を行う．小規模な流用よりも大規模な流用のほうが著作権侵害の度合いがより高いため，まず大規模流用を検出する必要があると考えられるためである．そこで，人手により流用ありと判断したソースコードの組のうち，ソースコード間の最長コードクローン長が一定以上であるもの（すなわち規模の大きい流用）を対象として同様の実験を行った．本実験では，正解集合である 190 組のプログラムペアの中で，最長コードクローン長に条件を設定しなかった場合に流用ありと判断された 63 組に対して，最長コードクローン長が一定以上のもののみを流用ありとするよう条件を設定し，流用ありと判断されたが，最長コードクローン長が小さい要素を除外した正解集合を作成し，判別精度を評価した．

また，本実験では，ソースコード流用の成否を判別するためのモデルを作成し，モデルの判別値を閾値とする．モデルの構築に多重ロジスティックモデルを使用し，説明変数には，ConSize と ConRate を用い，評価指標に Precision と Recall を使用した．

4.5.1 実験手順

1. 最長コードクローン長に対して閾値を設定し，4.4 節で作成した正解集合のうち，流用ありと分類したものの中から，最長コードクローン長が閾値以下の組を除外し，新しい正解集合を作成する．このとき，閾値は 0 から 200

表 9 最長コードクローン長による区分を行った正解集合

最長コードクローン長	流用ありの組数
100	62
150	60
400	54
600	46

までは 50 刻みで，200 から 600 までは 200 刻みで，それぞれ設定した．ここで，閾値 50 は閾値 0 のものと，閾値 200 は閾値 150 のものと同一の結果であったため除外した．作成した正解集合を表 9 に示す．

2. 新しい正解集合について，ConSize と ConRate を独立変数とし，流用有無の判別結果を従属変数としたロジスティック回帰モデルを作成する．

3. ロジスティック回帰モデルの判別値を閾値とし，閾値が一定以上の組を流用ありとみなした場合の Precision，Recall がどのような値を取るか算出することで，流用判別精度を評価する．

4.5.2 実験結果

最長コードクローン長が 100 以上のプログラムペアを正解集合における流用ありと設定し作成したロジスティック回帰モデルを式 (9) に，式 (9) の判別値 P を閾値としてソースコード流用の判別を行った結果を図 9 に示す．最長コードクローン長が 150 以上のプログラムペアを正解集合における流用ありと設定し作成したロジスティック回帰モデルを式 (10) に，式 (10) の判別値 P を閾値としてソースコード流用の判別を行った結果を図 10 に示す．最長コードクローン長が 400 以上のプログラムペアを正解集合における流用ありと設定し作成したロジスティック回帰モデルを式 (11) に，式 (11) の判別値 P を閾値としてソースコード流用の判別を行った結果を図 11 に示す．最長コードクローン長が 600 以上のプログラム

ペアを正解集合における流用ありと設定し作成したロジスティック回帰モデルを式 (12) に，式 (12) の判別値 P を閾値としてソースコード流用の判別を行った結果を図 12 に示す.

$$P = \frac{1}{1 + exp\{-(-6.33 + 3.63e^{-4} \cdot consize + 226.4 \cdot conrate)\}} \tag{9}$$

$$P = \frac{1}{1 + exp\{-(-6.65 + 3.81e^{-4} \cdot consize + 228.7 \cdot conrate)\}} \tag{10}$$

$$P = \frac{1}{1 + exp\{-(-7.39 + 3.80e^{-4} \cdot consize + 263.5 \cdot conrate)\}} \tag{11}$$

$$P = \frac{1}{1 + exp\{-(-7.41 + 3.51e^{-4} \cdot consize + 256.5 \cdot conrate)\}} \tag{12}$$

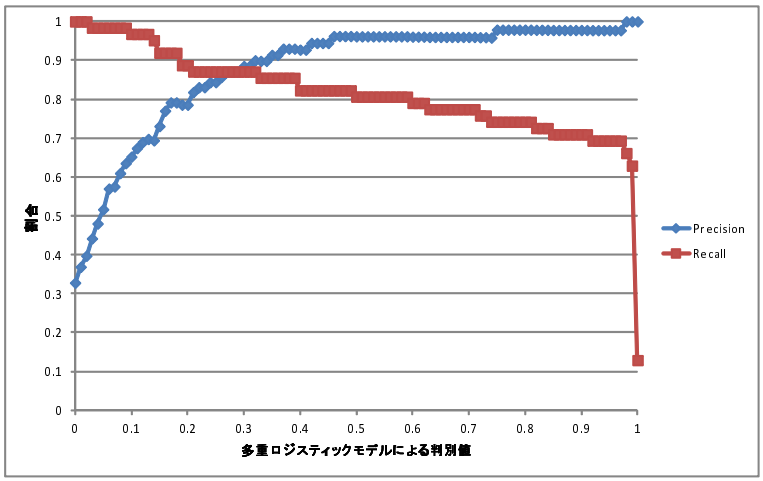

図 9 最長コードクローン長が 100 以上のものを対象とした，多重ロジスティックモデルの判別値を用いたソースコード流用ありの結果

ロジスティック回帰モデルでは，判別値 P=0.5 を判別境界値として設定する方法が用いられることがある．そこで，式 (9)，式 (10)，式 (11)，式 (12) で表されるロジスティック回帰モデルをそれぞれ用いたソースコード流用判別における，判別値 P の閾値 0.5 での正解集合の要素数 (正解集合)，判別値 P が閾値以上であった要素数 (閾値以上)，正解集合において流用ありとされている要素数 (正解)，判

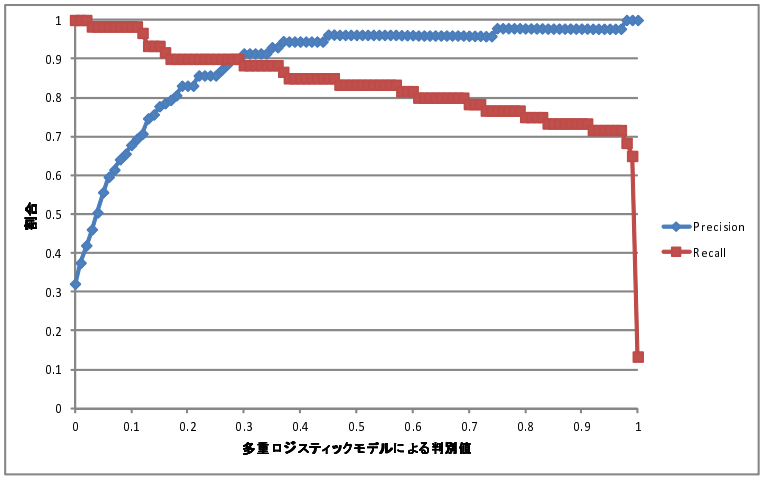

図 10　最長コードクローン長が 150 以上のものを対象とした，多重ロジスティックモデルの判別値を用いたソースコード流用ありの結果

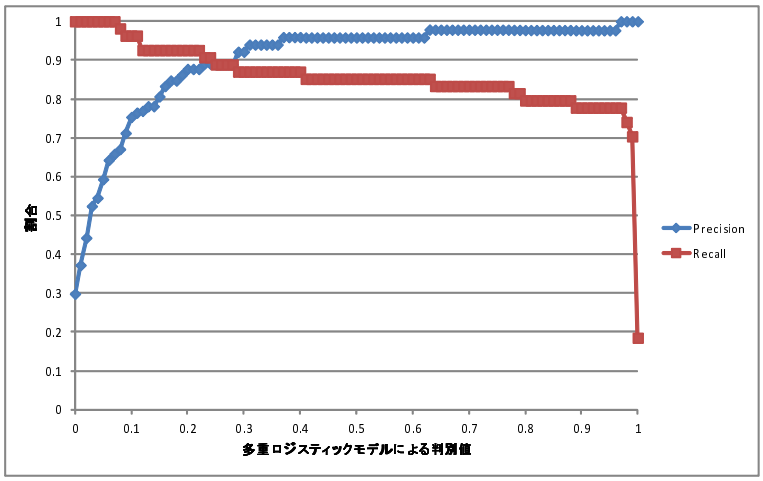

図 11　最長コードクローン長が 400 以上のものを対象とした，多重ロジスティックモデルの判別値を用いたソースコード流用ありの結果

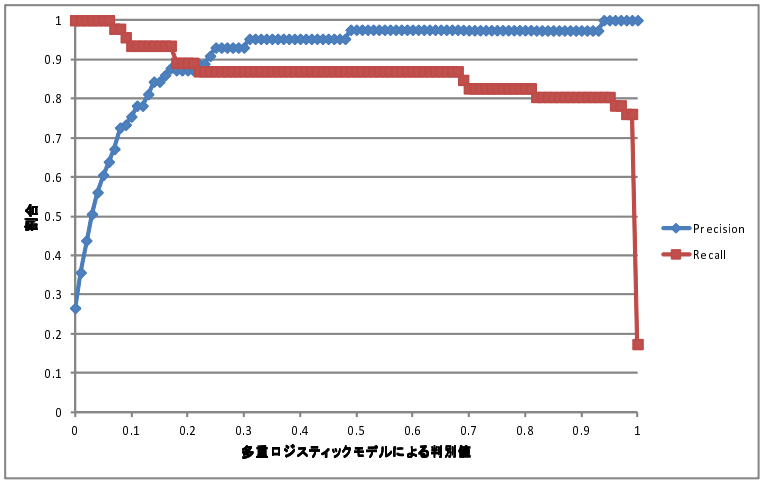

図 12　最長コードクローン長が 600 以上のものを対象とした，多重ロジスティックモデルの判別値を用いたソースコード流用ありの結果

別値 P が閾値以上であり，かつ，正解集合において流用ありとみなされている要素数 (判別)，Precison，Recall，および F 値を表 10 に示す．

図 9 に示すグラフから，最長コードクローン長が 100 以上のものを流用ありとみなした正解集合では，Precision=1 となる判別値 0.97 では Recall=0.63，Recall=1 となる判別値 0.02 では Precision=0.40 と導出されたことが分かる．また，図 10 に示すグラフから，最長コードクローン長が 150 以上のものを流用ありとみなした正解集合では，Precision=1 となる判別値 0.98 では Recall=0.68，Recall=1 と

表 10　ロジスティック回帰モデルの判別値 P の閾値 0.5 における結果

クローン長	正解集合	閾値以上	正解	判別	Precision	Recall	F 値
100	189	52	62	50	0.96	0.81	0.88
150	187	52	60	50	0.96	0.83	0.89
400	181	48	54	46	0.96	0.85	0.90
600	173	41	46	40	0.98	0.87	0.92

表 11　最長コードクローン長が一定以上の要素を対象とした判別精度評価実験結果

	Precision=1		Recall=1	
最長コードクローン長	判別値	Recall	判別値	Precision
100	0.97	0.63	0.02	0.40
150	0.98	0.68	0.02	0.42
400	0.97	0.78	0.07	0.66
600	0.94	0.80	0.06	0.64

なる判別値 0.02 では Precision=0.42 と導出されたことが分かる．また，図 11 に示すグラフから，最長コードクローン長が 400 以上のものを流用ありとみなした正解集合では，Precision=1 となる判別値 0.97 では Recall=0.78，Recall=1 となる判別値 0.07 では Precision=0.66 と導出されたことが分かる．また，図 12 に示すグラフから，最長コードクローン長が 600 以上のものを流用ありとみなした正解集合では，Precision=1 となる判別値 0.94 では Recall=0.80，Recall=1 となる判別値 0.06 では Precision=0.64 と導出されたことが分かる．これらの結果を表 11 に示す.

4.5.3 考察

表 10 から，最長クローン長 100 以上の場合，閾値 0.5 では 62 件の真に流用あり要素を含む 189 件の正解集合の中から 52 件が抽出され，うち 50 件が流用ありと判断され，かつ真に流用ありの要素であった．結果，約 81%の流用を検出できたことになる．同様に，最長クローン長 150 以上では約 83%，最長クローン長 400 以上では約 85%，最長クローン長 600 以上では約 87%のソースコード流用を検出できることが確認できる．また，Precision も同様に，最長クローン長の条件を増加させていくことで高い値を示していくことから，ロジスティック回帰モデルの判別値に基づくソースコード流用の判別は有効であるといえる.

表 12 最長クローン長 600 における誤検出・未検出リスト

	プログラム A	プログラム B	サイズ A	サイズ B	サイズ差 A-B	サイズ比 B/A
誤	gimp	gnubg	28,237,760	4,067,016	24,170,744	6.94311505
未検出	gimp	glabels	28,237,760	2,100,470	26,137,290	13.44354359
	geomview	glame	3,329,805	2,003,059	1,326,746	1.662359921
	geomview	easytag	3,329,805	1,454,918	1,874,887	2.288654756
	geomview	danpei	3,329,805	625,874	2,703,931	5.320248165
	geomview	battstat	3,329,805	261,989	3,067,816	12.709713
	geomview	gmmusic	3,329,805	241,646	3,088,159	13.77968185

しかし，流用ありとみなす最長コードクローン長の下限を大きくしていくことで，Precision はほぼ 100%となったが，Recall は，最大の 600 に設定した場合でも 87%に留まっていた．ここで，表 12 より，本来流用ありであるにも関わらず流用なしと判断されたプログラムペア 6 組のうち 5 組が，Geomview を含む組み合わせであった．4.4.3 項で述べた通り，Geomview に含まれるソースコード parse_tab.c には bison によって生成された自動生成コードが含まれており，正解集合作成時にはこれを流用であると判断した．この流用が特殊な流用であり，かつ，ソースコードの他の箇所に流用が全く存在せず，さらに類似性も低かったため，ConSize, ConRate 共にあまり大きな値とはならず，流用ありであるにも関わらず流用なしと判別してしまったと考えられる．このことから，自動生成コードの流用検出については，既存の parser ツールを中心に，個別に調査する必要がある．

次に，判別値 P の閾値を，Recall=1.0 となるような値に設定することを考える．表 11 より，最長コードクローン長を増加させていくことで Precision も高い値を示していき，最長コードクローン長が 600 以上のソースコード流用を対象とした場合，判別値 P の閾値 0.06 において 0.64，つまり，6 割程度の精度でソースコード流用を検出できることになる．この場合，4 割程度をソースコード流用ではないがソースコード流用であると誤検出してしまうことになるが，ソースコード流

用であるがソースコード流用ではないと誤認し，見落とすことはない．加えて，6 割程度がソースコード流用を含む可能性が濃厚である集合まで絞り込むことができるといえるため，全ての集合から人手によってソースコード流用の有無を判別するよりも，コストを低減できると考えられる．このような閾値の設定の仕方は，確実にソースコード流用の見落としを防ぐことを目的とする，出荷前の確認などにおいて有効であると考えられる．

次に，判別値 P の閾値を，Precision=1.0 となるような値に設定することを考える．表 11 より，最長コードクローン長を増加させていくことで Recall も高い値を示していき，最長コードクローン長が 600 以上のソースコード流用を対象とした場合，判別値 P の閾値 0.94 において 0.80，つまりソースコード流用の 80%を検出できることになる．この場合，全てのソースコード流用のうち，残りの 20%を見落としてしまうことになるが，検出された 80%は確実に流用であるといえるため，人手によってソースコード流用の有無を判別するよりも，コストを低減できると考えられる．このような閾値の設定の仕方は，確実にソースコード流用であることを主張することを目的とする，裁判における証拠提出などにおいて有効であると考えられる．

4.4 節および本節の実験結果から，ソースコード流用の有無を判断する際，プログラム圧縮を行い，ConSize，ConRate を算出し，それらを独立変数としたロジスティック回帰モデルを作成し，その判別値を用いることで，高い精度でのソースコード流用の判別が実現できると考えられる．また，この方法は，最長コードクローン長が 600 トークン以上であるような，規模の大きいソースコード流用に対して有効であるといえる．

4.6 実行時間の計測

従来のコードクローンメトリクスを用いた流用検出と比べて，どの程度コストを減らせるかを確認するため，プログラム圧縮の実行時間の計測を行う．

4.6.1 実験手順

実験に用いた環境を以下にまとめる.

- 計算機
  - CPU：Intel(R) Core(TM)2 CPU 6600 2.40GHz
  - メモリ：2.00GB
  - オペレーティングシステム：Windows 7 Professional (64bit)
- 実験対象ソフトウェア
  - GIMP
    - バージョン 2.3.19
    - 28,237,760 バイト
  - Cinepaint
    - バージョン 0.19-0
    - 8,046,065 バイト
- アーカイブツール
  - tar
- 圧縮ツール
  - Lzip
    - 最高圧縮のため-9 オプションを使用
- クローン検出ツール
  - CCFinderX
    - 比較対象として使用
    - Minimum Clone Length：30

* Minimum TKS：12
* Shaper Level：2 - Soft shaper
* P-match Application：Use P-match
* Prescreening Application：Use Prescreening

4.6.2 実験手順

プログラム圧縮の実行時間計測の手順を以下に示す.

1. 実行時間の計測を開始する.
2. 実験対象ソフトウェアを同じディレクトリに格納し，tar を用いてアーカイブする.
3. アーカイブしたファイルを，Lzip を用いて圧縮する.
4. 圧縮が完了し，入力を受け付けるようになった段階で実行時間の計測を終了する.

また，今回の実験では実行時間の比較対象としてCCFinderX を用いた．CCFinderX によるコードクローン検出の実行時間計測の手順を以下に示す.

1. CCFinderX に実験対象ソフトウェアを指定し，各種オプションを指定する.
2. クローン検出を開始すると同時に，実行時間の計測を開始する.
3. クローン検出が完了し，結果が表示されるようになった段階で実行時間の計測を終了する.

   なお，どちらの実行時間計測についても，計測は人手により行った.

4.6.3 実験結果

プログラム圧縮では，tar を用いたアーカイブの完了までに 8.6 秒，Lzip を用いた圧縮の完了までに 50.7 秒，合計で 59.3 秒かかった．一方，CCFinderX によるコードクローン検出では，検出完了までに 23 分 50 秒かかった．このことから，従来のコードクローンメトリクスを用いた流用検出と比べて，プログラム圧縮による流用検出はより短時間で行えると考えられる．

5. 関連研究

5.1 プログラム圧縮を用いたソースコード流用検出に関する研究

Cilibrasi ら [2] は圧縮を用いてファイル間の類似度を評価するための指標として，Normalised Compression Distance(ncd) を提案している．ncd は次式 (13) で算出される．ここで，C は圧縮を，x，y はプログラムを，xy はプログラム x，y の連結を，$|s|$ はファイルサイズにおける文字列長を示す．

$$ncd(x,y) = \frac{|C(xy)| - min(|C(x)|, |C(y)|)}{max(|C(x)|, |C(y)|)} \tag{13}$$

また，Hemel ら [7] は，ファイル間の類似度ではなく，一方のファイルが他方のファイルをどれだけ含んでいるかを示す指標として，$reuse_c$ を提案している．$reuse_c$ は次式 (14) で算出される．

$$reuse(x,y) = \frac{|C(x)| + |C(y)| - |C(xy)|}{|C(y)|} \tag{14}$$

Hemel らが式 (14) を提案した理由として，x が y を全て含んでおり，かつファイルサイズがより大きい場合，ncd は非常に低い値を示してしまうという点を挙げている．しかし，ncd に代わる指標として提案された $reuse_c$ についても，プログラムとライブラリのような，一方が他方を含んでいることが明らかであるような組み合わせを対象としているため，ファイル圧縮を用いたソースコード流用の有無を判別するための評価尺度としては一般的ではないと考えられる．加えて，現時点では流用があった場合には圧縮率が高かったと述べるに留めており，$reuse_c$ がどの程度の数値になれば流用ありと判別できるかも明らかにされておらず，また，判別精度も不明である．本論文では，包含関係にない，部分的なソースコード流用にも対応することができ，また，ソースコード流用ありと判別するための閾値やその判別精度にも言及しているという点が異なる．

5.2 ソフトウェア間におけるコードクローン含有率に基づく手法

JPlag は，Prechelt らによって開発された，与えられたソースコード集合から類似したソースコード対を検出する手法である [19]．JPlag はソースコードを構文解析し，トークン列に変換する．トークン列をペアで比較し，ペアごとに類似度を求めることによりソースコード流用の検出を行う．類似度は次式（15）,（16）で算出される．

$$Sim(A, B) = \frac{2coverage(tiles)}{|A| + |B|} \tag{15}$$

$$coverage(tiles) = \sum_{match(a,b,length) \in tiles} length \tag{16}$$

類似度比較アルゴリズムには，Wise の “Greedy String Tiling” を用いている [22]．“Greedy String Tiling” は，1 組のテキストをトークン列化し，片方のトークン列中の部分列で，他方をどの程度カバー出来るかにより類似度を導出する手法である．なお，JPlag はプログラミング言語の Java，C，C++，Scheme に対応している．

JPlag は，ソースコード全体が類似するソースコード流用検出に役立つ．しかし，ソースコードの一部分だけを流用した，部分的なソースコード流用は検出できないことがあると考えられる．また，ソースコード流用ありと判断する閾値について議論されていない為，ソースコード流用の判断が人に委ねられてしまうという問題がある．本論文では，一定以上の規模であれば部分的なソースコード流用にも対応でき，また，ソースコード流用が存在するプログラムペアの特定という点においては，全て人手による判断を行うよりコストを低減できるという点が異なる．

5.3 ソフトウェアバースマーク

ソフトウェアバースマークとは，対象となるソフトウェアが持つプログラムの特徴量を抽象化して表現したものである．2 つのプログラムから，それぞれソフ

トウェアバースマークを抽出し，ソフトウェアバースマークが類似している場合にソースコード流用が存在すると判断する．ソフトウェアバースマークは次のように定義される．

バースマークの定義

p, q を与えられたプログラムとする．$f(p)$ を p からある方法 f により抽出された特徴の集合とする．このとき，以下の条件を満たすとき，$f(p)$ を p のバースマークであるという．

- $f(p)$ はプログラム p のみから得られる．
- q が p のコピーならば，$f(p) = f(q)$ である．

また，ソフトウェアバースマークは以下の性質を満たすことが望まれる．

保存性

p から任意の等価変換により得られた p に対して $f(p) = f(p')$ を満たす．

弁別性

同じ仕様を持つ p と q に対し，それらが全く独立に実装された場合，$f(p)! = f(q)$ となる．

ソフトウェアバースマークは，静的バースマークと動的バースマークの 2 つに分類される．

静的バースマークとは，プログラムを実行せずに取得可能な特徴を使用したソフトウェアバースマークである．Tamada らは，初期値代入，メソッドの呼び出し系列，そして継承関係をソフトウェアバースマークとして提案している [21]．動的バースマークとは，プログラムの実行時に得られる特徴を使用したソフトウェアバースマークである．岡本らは，WindowsAPI の実行順序，実行頻度を動的バースマークとして提案している [17]．林らは，Java バイナリ中の実行系列中の処理間隔を動的バースマークとして提案している [6]．処理間隔を動的バースマークとして用いることで，攻撃態勢を向上させた．楓らは，部分的なソースコード流用を検出するために，ソースコード流用部分で定義されているメソッド毎に実行系

列を抽出し，メソッドの部分系列毎に抽象化を行うことで部分的なソースコード流用への有効性を示した [12].

ソフトウェアバースマークの性質上，多数のソフトウェアバースマークを複合的に検証する必要がある．そのため，ソースコード流用ありと判断する際に，ソースコード流用の判断が人手に委ねられてしまうという問題がある．本論文では，複合圧縮度の値という単一のパラメータを検証すればソースコード流用の存在するプログラムペアを特定できるという点において，人手による検証に比べてコストを低減できるという点が異なる．

6. おわりに

本論文では，オープンソースソフトウェアの流用におけるライセンス違反の回避を容易にすることを目的として，ソースコード圧縮の度合いを評価する新しい尺度「複合圧縮度」を提案すると共に，提案尺度に基づく流用検出の実用性を実験的に評価した.「複合圧縮度」とは，対象とする 2 つのソースコードそれぞれの圧縮後のサイズから，それら 2 つを連結した状態での圧縮後のサイズを差し引いた値である．重複する文字列を多く含むソースコードを圧縮すると，そのサイズが大きく減少することから，2 つのソースコード間で流用が行われていれば，連結した状態での圧縮後のサイズは小さくなり，複合圧縮度の値は大きくなる.「複合圧縮度」の具体的な評価尺度として，個別圧縮後の合計ファイルサイズから，複合圧縮後のファイルサイズを引いた値である「ConSize」と,「ConSize」を個別圧縮後の合計ファイルサイズで除算し，正規化した値である「ConRate」を提案した．C もしくは C++で記述されたオープンソースソフトウェアから作成した 190 組のソースコードペアに提案尺度を適用した結果，得られた主な知見は次の通りである.

- ConSize について，Precision，Recall の交点の値を閾値とした場合，閾値 9000 で Precision=Recall=0.82 の判別精度が得られた.
- ConRate について，Precision，Recall の交点の値を閾値とした場合，閾値 0.013 で Precision=Recall=0.76 の判別精度が得られた.
- ConSize，ConRate を独立変数としたロジスティック回帰モデルにより判別を行った場合，Precision=0.96，Recall=0.79，F 値=0.87 の判別精度が得られた.
- 大規模なソースコード流用の検出について，最長コードクローン長が 600 トークン以上のソースコード流用を対象とした場合，ConSize，ConRate を独立変数としたロジスティック回帰モデルにより，Precision=0.98，Recall=0.87，F 値=0.92 の判別精度が得られた.

- プログラム圧縮に要する時間を，コードクローン検出に要する時間と比較した結果，36 メガバイト程度のプログラムペアに対し，プログラム圧縮は 59.3 秒，コードクローン検出は 23 分 50 秒であった．このことから，プログラム圧縮による流用検出は，従来のコードクローンによる検出よりも短時間で行えると考えられる．

これら得られた知見を活用することで，全て人手によりソースコード流用を判別する場合に比べて，検出に要するコストを低減できると考えられる．

ただし，提案方法では，parser ツールである bison によって生成された自動生成コードを検出できなかった．ライセンス違反を引き起こす可能性のある自動生成コードの流用検出については，既存の parser ツールを中心に，個別に調査する必要があると考えられる．

なお，提案尺度 ConSize と ConRate は，バイナリプログラムに対しても適合が可能である．したがって，今後の課題としては，バイナリプログラムに対して同様の実験を行うことが考えられる．バイナリプログラムに対しても有効性が示された場合，ソースコードが入手困難な状況下にあってもソースコード流用の識別が可能となり，適用範囲がさらに広がると期待される．

謝辞

本研究を進めるにあたり，多くの方々にご指導を賜りました．この場を借りてお世話になった方々に感謝の意を表させて頂きたいと思います．

奈良先端科学技術大学院大学 情報科学研究科 松本 健一 教授には，本研究の主指導教員を担当して頂くとともに，本論文の審査委員も務めて頂きました．大学院入学以来，様々な視点からご指導して頂き，研究者としての在り方や，研究活動に取り組む上での心構えなど数多くのご指導を賜りました．また，学内での研究活動のみならず，研究会発表や就職活動など，様々な面で相談に乗って頂いたことは大きな心の支えとなりました．心より感謝します．

奈良先端科学技術大学院大学 情報科学研究科 藤川 和利 教授には，本研究の副指導教員を担当して頂くとともに，本論文の審査委員も務めて頂きました．学内での発表において，本研究を進めるに当たり，非常に有益な，的確かつ多数の御指摘および御指導を賜りました．深く感謝致します．

奈良先端科学技術大学院大学 情報科学研究科 門田 暁人 准教授には，本研究の副指導教員を担当して頂くとともに，本論文の審査委員も務めて頂きました．大学院入学以来二年間，研究活動や就職活動など，多岐に渡って相談に乗って頂きました．特に，研究活動については，本研究をまとめるに至るまで，何度もくじけそうになった私に，熱心に，粘り強く，親身に，数々の御助言をしてくださいました．本研究は，門田准教授の的確な御指導，御助言がなければなしえなかったものだと認識しています．また，現在の私はいなかったことだと感じております．心より厚くお礼申し上げます．

奈良先端科学技術大学院大学 情報科学研究科 大平 雅雄 助教には，研究者としての在り方，研究活動に取り組む上での心構え，論文執筆に対するアドバイス，プレゼンテーションの作法など，数々の適切かつ建設的なアドバイスを頂きました．大平助教の研究に熱心に取り組む姿勢や，研究が思うように進まず行き詰っていたときに声をかけていただいたことが，本研究を進めるに当たり大きな励みとなりました．深く感謝致します。

奈良先端科学技術大学院大学 ソフトウェア工学講座 伊原 彰紀 氏には，本論文について有益な御指摘，御意見を頂きました．氏の研究に向ける熱意は大きな

励みとなり，また，日常生活においても様々な場面で気にかけて頂きました．更に，平時の研究のみならず，学内発表においても的確な御指摘，御指導を賜りました．心より感謝します．

奈良先端科学技術大学院大学 ソフトウェア工学講座の皆様には，日頃より多大な御協力と御助言を頂きました．大学院において，厳しくも充実した二年間を過ごすことができたのは，皆様の多岐に渡る御支援や激励の賜物です．研究のみならず，進路や時には日常生活でも，皆様との交流は私にとって大きな励みとなり，また，大きな支えとなりました．深く感謝致します．

最後に，大学院への進学という私の意志を尊重してくださり，日々の生活の中で常に私を励まし，経済的にも精神的にも支えてくれた家族に，心より厚くお礼申し上げます．

参考文献

[1] CCFinder ホームページ. CCFinderX, 2005.
http://www.ccfinder.net/ccfinderx-j.html.

[2] R. Cilibrasi and P. M. B. Vitanyi, “Clustering by compression,” IEEE Transactions on Information Theory, 51(4), Apr. 2005.

[3] Free Software Directory. Free Software Foundation,
http://directory.fsf.org/.

[4] The GNU General Public License v3.0. Free Software Foundation, 2007.
http://www.gnu.org/licenses/gpl.html.

[5] GNU Lesser General Public License v3.0. Free Software Foundation, 2007.
http://www.gnu.org/licenses/lgpl.html.

[6] 林 晃一郎, 楓 基靖, 真野 芳久, “特徴抽出と抽象化による動的バースマークの構成とその検証”, 情報処理学会研究報告, Vol. 2005, No. 122, pp. 31-36, 2005.

[7] Armijn Hemel, Karl Trygve Kalleberg, Rob Vermaas, Eelco Dolstra, “Finding Software License Violations Through Binary Code Clone Detection,” In Proceedings of MSR, pp. 63-72, 2011.

[8] 肥後 芳樹, 楠本 真二, 井上 克郎, “コードクローン検出とその関連技術”, 電子情報通信学会論文誌, Vol. J91-D, No. 6, pp. 1465-1481, 2008.

[9] 国内オープンソースソフトウェア利用実態調査を発表. IDC Japan, 2012.
http://www.idcjapan.co.jp/Press/Current/20120111Apr.html.

[10] 「USB版Windows 7」作成ツールにGPLコード　Microsoftが謝罪. ITmedia News, 2009.
http://www.itmedia.co.jp/news/articles/0911/16/news026.html.

[11] オープンソースソフトウェアとは【オープンソース】. IT 用語辞典, 2004. http://e-words.jp/w/E382AAE383BCE38397E383B3E382BDE383BCE382B9.html.

[12] 楓 基靖, 真野 芳久, “Java プログラムの部分盗用に対する動的バースマーク”, 電子情報通信学会総合大会後援論文集, Vol. 2007, pp. 219, 2007.

[13] オープンソースソフトウェアに関する現状と課題. 経済産業省北海道経済産業局, 2004. http://www.hkd.meti.go.jp/hokim/open_houkoku/houkoku_01.pdf.

[14] Darko Kirovski, Johnson Kin and William H. Mangione-Smith, ”Procedure Based Program Compression,” Proceedings of the 30th annual ACM/IEEE international symposium on Microarchitecture (MICRO 30), pp.204-213, 1997.

[15] Simone Livieri, Yoshiki Higo, Makoto Matsushita and Katsuro Inoue, “Very-Large Scale Code Clone Analysis and Visualization of Open Source Programs Using Distributed CCFinder: D-CCFinder,” Proc. 29th International Conference on Software Engineering (ICSE 2007), pp.106-115, Minneapolis, MN, May 2007.

[16] Akito Monden, Satoshi Okahara, Yuki Manabe, and Ken-ichi Matsumoto, “Guilty or Not Guilty: Using Clone Metrics to Determine Open Source Licensing Violations,” IEEE Software, Special Issue on Software Protection (March/April 2011), Vol.28, No.2, pp.42-47, March 2011.

[17] 岡本 圭司, 玉田 春昭, 中村 匡秀, 門田 暁人, 松本 健一, “APL 呼び出しを用いた動的バースマーク”, 電子情報通信学会論文誌, Vol. J89-D, No. 8, pp.1751-1763, 2006.

[18] The Open Source Definition. Open Source Initiative, 1998. http://opensource.org/docs/osd.

[19] Luts Prechelt, Guido Malpohl, and Michael Philippsen, "Finding plagiarisms among a set of programs with JPlag," Journal of Universal Computer Science, Vol. 8, No. 11, pp. 1016-1038, 2001.

[20] PlayStation 2 Game ICO Violates the GPL. Slashdot, 2007. http://news.slashdot.org/story/07/11/28/0328215.

[21] Haruaki Tamada, Masahide Nakamura, Akito monden, and Ken-ichi Matsumoto, "Java Birthmarks : Detecting the Software Theft," IEICE transactions on information and systems, Vol. 88, No. 9, pp.2148-2158, 2005.

[22] Michael J. Wise, "String Similarity via Greedy String Tiling and Running Karp Rabin Matching," http://vernix.org/marcel/share/RKR_GST.ps.

付録

A. 評価実験における実験対象ソフトウェアの一覧

表 13 実験対象ソフトウェアの一覧

ソフトウェア名	バージョン	ドメイン	URL
Avdbtools	0.3	Hobbies	http://directory.fsf.org/wiki/Abdbtools
AIDE	0.9	Software-development	http://directory.fsf.org/wiki/AIDE
Battstat	2.0.11	System-administration	http://directory.fsf.org/wiki/Battstat
Cdrtools	2.01	Audio	http://directory.fsf.org/wiki/Cdrtools
Chemtool	1.6.11	Science	http://directory.fsf.org/wiki/Chemtool
Cinepaint	0.19-0	Video	http://directory.fsf.org/wiki/Cinepaint
CVSps	1.3.3	Software-development	http://directory.fsf.org/wiki/CVSps
Danpei	2.9.6	Images	http://directory.fsf.org/wiki/Danpei
Easytag	1.1	Audio	http://directory.fsf.org/wiki/Easytag
Gaby	2.0.3	Business	http://directory.fsf.org/wiki/Gaby
Gbonds	2.0.2	Business	http://directory.fsf.org/wiki/Gbonds
Geomview	1.8.1	Graphics	http://directory.fsf.org/wiki/Geomview
GIMP	2.3.19	Graphics	http://directory.fsf.org/wiki/GIMP
Glabels	2.0.3	Business	http://directory.fsf.org/wiki/Glabels
Glame	2.0.1	Software-development	http://directory.fsf.org/wiki/Glame
Glom	0.9.8	Database	http://directory.fsf.org/wiki/Glom
Gmmusic	1.1.91	Music	http://directory.fsf.org/wiki/Gmmusic
Gnofin	0.8.4	Business	http://directory.fsf.org/wiki/Gnofin
Gnubg	0.14.3	Game	http://directory.fsf.org/wiki/Gnubg
Gnugo	3.6	Game	http://directory.fsf.org/wiki/Gnugo