SCIS 2017 2017 Symposium on
Cryptography and Information Security
Naha, Japan, Jan. 24 - 27, 2017
The Institute of Electronics,
Information and Communication Engineers

協調フィルタリングを利用した名前難読化の逆変換
Reverse Conversion Method for Name Obfuscation Methods by Collaborative Filtering

磯部 陽介 * 玉田 春昭 *
Yosuke Isobe Haruaki Tamada

あらまし 今日では，ソフトウェア保護手法として難読化がよく用いられてる．中でも実装が容易であることから名前難読化は広く使用されている．名前難読化は，ソースコード内のメソッドや変数名を意味を持たない文字列に変換する難読化方法である．しかしながら，名前難読化の有効性は検証されていない．もし名前難読化されたソフトウェアを名前難読化される前のソフトウェアに戻すことができれば，名前難読化によるソフトウェアの保護は無効化される．本稿では，名前難読化によって変換されたメソッド名を逆変換し，意味のある文字列に変換することを復元と呼ぶ．名前難読化の有効性を検証するために，まず，名前難読化されたメソッド名の復元を行う．復元を行うために，ソフトウェア開発支援の一つである名前推薦法を使用する．名前推薦法は，クラスやメソッド内部の特徴から，メソッドが示す働きに相応しいメソッド名を開発者に推薦する方法である．本稿では協調フィルタリングを用いてメソッド名を推薦する．大量の OSS を取得し，そのメソッド情報を教師データとして協調フィルタリングを行う．用いるメソッド情報は，名前難読化では難読化されない情報とする．また，情報の出現頻度によって教師データをフィルタリングし，精度の高い復元を試みる．本手法によって，77.5%の確率でメソッド名を復元できることがわかった．一方で，18.0%の確率で誤ったメソッド名を推薦した．しかし，フィルタリングした教師データを用いた結果，メソッド名の復元率は 39.9%と落ちるものの，誤推薦の割合は 5.2%に下げることが確認できた．

キーワード 難読化，名前推薦法，協調フィルタリング

1 はじめに

インターネットの発展により，ソフトウェアが入手しやすくなっている．しかし，その一方で，ソフトウェアの不正利用は年々深刻化している．ソフトウェアの不正利用を防ぐためにソフトウェア保護技術が提案され，また，用いられており，その重要性が高まっている．ソフトウェア保護技術とは，ソフトウェア内部の情報を不正な利用から保護する技術の総称でる．

ソフトウェア保護技術の一つに難読化手法がある．難読化とはプログラムの外部的な振る舞いは変更せず，プログラムの理解が困難になるように変換する技術である．中でも，プログラム中に現れる名前を，意味のない名前に変換する名前難読化は，実装が容易なことから広く使用されている．名前を意味のない名前に変換することでプログラムの理解を困難にし，ソフトウェアの情報を保護する [1].

名前難読化を実現したツールは，ProGuard[1] や DashO[2]など，数多く存在する．しかし，それらツールの有効性は検証されていない．例えば，名前難読化が適用されたとしても，元の名前，もしくはそれに近い名前に復元できたとする．すると，名前難読化はもはや保護手法とは呼べない．しかし，現状ではそのような議論はほとんど行われていないため，従来の方法が慣習的に使用されている．

一方で，ソフトウェア開発支援の一つに名前推薦法がある [2]．クラスやメソッド内部，そして他のメソッドの情報から，開発者に対して適切なクラス名，メソッド名を推薦する手法である．この手法を名前が難読化されたプログラムに対して適用し，元の名前に復元できれば名前難読化が無効化されることになる．しかし，名前が難読化されたプログラムに，名前推薦法をそのまま適用できない．名前推薦法は，他のメソッドなど，名前難読化で変更される可能性のある名前を手掛かりに名前を推薦

* 京都産業大学コンピュータ理工学部, 京都市北区上賀茂本山，Motoyama, Kamigamo, Kita-ku, Kyoto-shi, Kyoto

[1] http://proguard.sourceforge.net
[2] http://www.agtech.co.jp/products/preemptive/dasho

```
 1: import javax.swing.*;
 2:
 3: public class ImageViewer extends JFrame{
 4:     private Icon icon;
 5:     public ImageViewer(String file){
 6:         setTitle(file);
 7:         Icon myIcon = new ImageIcon(file);
 8:         icon = myIcon;
 9:         add(new JLabel(icon));
10:         pack();
11:     }
12:     public static void main(String[] args){
13:         for(String arg: args){
14:             ImageViewer viewer = new ImageViewer(arg);
15:             viewer.setVisible(true);
16:         }
17:     }
18: }
```

図 1: サンプルプログラム（画像ビューワ）

```
 1: import javax.swing.*;
 2:
 3: public class A extends JFrame{
 4:     private Icon a;
 5:     public A(String a){
 6:         setTitle(a);
 7:         Icon b = new ImageIcon(a);
 8:         this.a = b;
 9:         add(new JLabel(this.a));
10:         pack();
11:     }
12:     public static void main(String[] a){
13:         for(String b: a){
14:             A c = new A(b);
15:             c.setVisible(true);
16:         }
17:     }
18: }
```

図 2: 名前難読化後の図 1

するためである．しかし，名前難読化では一般的に，ライブラリのクラス名，メソッド名は変更できない．そこで，本稿では，名前難読化では変更できないライブラリ名，メソッド名を手掛かりに名前の復元を試みる．もちろん，得られる情報が少ないため，完全な復元は困難であるため，メソッド名に限定する．

メソッド名復元のために，協調フィルタリング（Collaborative Filtering）を利用する．協調フィルタリングは，多くの項目の情報を蓄積しておき，対象となる項目と類似する項目の情報を用いて自動的に推論を行う手法である．買い物サイト，例えば Amazon.co.jp，のおすすめ商品の推薦などに用いられている．本稿では，メソッド中の名前難読化されない名前を利用してメソッド名の推薦を行うことで，メソッド名の復元を試みる．

以降，第 2 節で実験に必要な事前知識を説明し，第 3 節では提案方法について記述する．第 4 節は実験内容について，第 5 節では関連研究を述べる．最後に，第 6 節で本稿をまとめ，課題を述べる．

2 準備

2.1 名前難読化

難読化とは，ソフトウェア保護手法の一つであり，プログラムを意図的に理解が困難なプログラムに変換する手法である．一般的に，プログラムの情報の一部をより複雑な形に変換することや，その情報量を減らすことで実現される．例えば，opaque predicate は意味のない条件分岐を追加することでコントロールフローを複雑にして，理解困難さを上げている [3]．一方，名前難読化は，変数名やクラス名などが持つ意味という情報を削除することで困難さを向上させる．つまり，プログラムの理解には，フィールド名やメソッド名，クラス名などの名前が重要な手掛かりとなる．

図 1 に示す Java プログラムを，名前難読化手法を用いて難読化した例を図 2 に示す．一般的に名前難読化によって難読化される情報は，ユーザが定義したフィールドやメソッド，クラスの名前である．一方，API ライブラリが持つ名前は難読化できず，名前難読化後のプログラムにも元の名前のまま残されている．例えば，図 2 における `A`，`a`，`b`，`c` が難読化された名前であり，`JFrame` や `JLabel`，`Icon`，`ImageIcon` などが元のまま残された API ライブラリに含まれる名前である．

2.2 協調フィルタリング

協調フィルタリングは，ある項目の特徴と似た特徴を持つ項目を推薦する技術である．一般的に，類似度は Jaccard 係数や Dice 係数などで測定される．協調フィルタリングではこの類似度に対して閾値を決め，閾値を超えたものを推薦する．

3 提案手法

3.1 全体の流れ

ここでは，名前が難読化された Java プログラムに含まれるメソッド名の復元手法を述べる．協調フィルタリングを行うには，大量のデータの蓄積が必要である．近年のソフトウェア開発では，ライブラリを必要に応じてインターネット上のリポジトリからダウンロードし，実行することが行われるようになっている．例えば，Java プログラムでは，Maven Central Repository[4]，Ruby は Ruby Gems[3]などがある．これらのようなリポジトリ

[3] `https://rubygems.org`

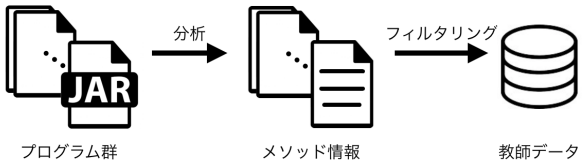


図 3: 教師データの準備

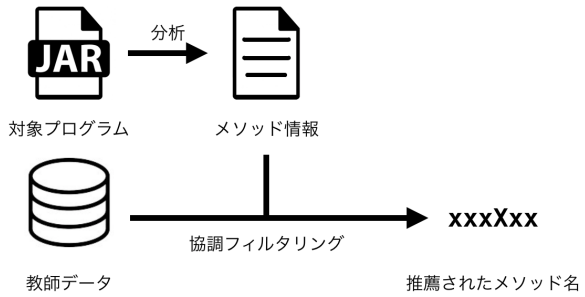


図 4: メソッドの推薦

に置かれているプログラム群を，復元対象のプログラムの分析の教師データとする．

復元手法は次の 2 つのステップから構成される．

1. 教師データの準備．
2. 教師データと対象プログラムの比較．

ステップ 1 では，名前が難読化されたプログラムとの類似性を測定するための教師データを構築する．教師データは，膨大である方が望ましく，そのためには，大量の元データが必要である．なお，このステップは，提案手法の実行前に一度だけ実施すれば良い．ただし，ステップ 1 実行後にリポジトリに追加されたプログラムを利用したい場合，追加されたプログラムを分析し，教師データへの追加が必要である．

ステップ 2 は，名前を復元したいプログラムを入力とし，名前の復元を行うステップである．入力されたプログラムからメソッドに分け，メソッドに含まれる難読化されていない API 名（クラス名，メソッド名，フィールド名）を抽出する．取得した API 名を元に，教師データから類似しているメソッドを協調フィルタリングにより推薦する．教師データのメソッド名は難読化されていないため，推薦されたメソッドの名前が復元の候補となる．

以下，2 つのステップの詳細を述べる．

3.2 教師データの準備

図 3 にステップ 1 の概要を示す．ステップ 1 では，協調フィルタリングに用いる教師データのデータベースを準備する．教師データの準備には，膨大な数のプログラムを用意することが望ましい．そこで，用意したリポジトリ $R = \{p_1, p_2, ..., p_n\}$ からプログラム p_i を取り出す．そして，プログラム p_i が持つ各メソッドからメソッド情報 M_j を抽出する（$\mathrm{mi}(p_i) = \{M_1, M_2, ..., M_m\}$）．ただし，$\mathrm{mi}(p)$ をプログラム p からメソッド情報の集合を抽出する方法とする．

メソッド情報 M_j はメソッド名 n_j，そのメソッドで呼び出しているメソッドのシーケンス $F_j = \{f_1, f_2, ..., f_l\}$ から構成される（$M_j = \{n_j, F_j\}$）．ただし，メソッドのシーケンス F_j では，名前難読化の影響を受けないものに限るものとする．すなわち，ユーザ定義のクラス，メソッドは名前難読化の対象となるため，教師データには含めない．対象とするメソッド集合は適宜与えられるものとする．最後に，取得した全てのメソッド情報の和集合を取り，教師データ $\mathcal{M}$ とする（$\mathcal{M} = \mathrm{mi}(p_1) \cup \mathrm{mi}(p_2) \cup ... \cup \mathrm{mi}(p_n)$）．

3.3 教師データと対象プログラムの比較

ステップ 2 では，ステップ 1 で準備した教師データを用いて対象プログラム中のメソッド名の復元を行う．ステップ 2 の概要を図 4 に示す．

入力されたプログラム p' からメソッド情報を抽出する（$\mathrm{mi}(p') = \{M'_1, M'_2, ..., M'_{m'}\}, M'_j = \{n'_j, F'_j\}$）．このプログラム p' は難読化されているため，$n_j (1 \leq j \leq m')$ は難読化された名前である．そして，n'_j の復元候補を教師データから見つけ出す．入力プログラムの各メソッド情報を対象に，教師データ $\mathcal{M}$ 中の各メソッド情報 M_i と比較する．比較には，M'_j に含まれる F'_j と，M_i に含まれる F_i とで行い，類似度 s_i を算出する（$s_i = \mathrm{sim}(F_i, F'_j)$）．算出された類似度 s_i が閾値 ϵ を超えた場合，難読化されたメソッドの復元候補として，見つかったメソッド情報 M_i に含まれる名前 n_i を推薦する．結果として，難読化されたメソッド情報 M'_j の名前復元候補 $N_j = \{n_1, n_2, ..., n_r\}$ が得られる．

4 実験

4.1 教師データの準備

本研究では，教師データの構築のために Maven Central Repository から 17,637 個の jar ファイルを取得した．これらのプログラムを解析し，メソッド名及び呼び出しているメソッドを取り出した．なお呼び出しメソッドは Java SE 8 に含まれるもののみを抽出する[4]．また，呼び出しメソッドが無いメソッドは教師データには含めないようにした．結果として，17,637 個の jar ファイルから計 5,110,506 個のメソッド情報を含む教師データを構築した．

[4] パッケージ名が `java` もしくは `javax` から始まるクラスとした．もちろん，`org.omg.CORBA` や `org.w3c.dom`，`org.xml.sax` なども Java SE 8 に含まれるが，今回は簡単化のため省略した．

教師データにおける，各メソッドの呼び出しメソッドの数（$|F|$）の頻度を図 5 に示す．横軸が呼び出しメソッド数ごとのメソッドであり，縦軸の第 1 軸が頻度，第 2 軸がその時の呼び出しメソッドの数を表している．また，横軸は，呼び出しメソッドの数の昇順でソートしている．図 5 から呼び出しメソッドが多くなるほど，教師データに現れる頻度が少なくなることがわかる．名前の復元を考えた時，頻度が十分でない場合は対象となるメソッドが見つからない場合がある．さらに，呼び出し回数が非常に多い場合，類似することすら稀であるため，今回は対象としない．したがって，本実験では，1 から 5 個の呼び出しメソッドを含むメソッドのみを復元の対象とする．

4.2 テストデータの準備

本稿で用いるテストデータは，Maven Central Repository 以外の Maven Repository である Sonatype Releases[5] から 9 つのプログラムを無作為に取得した．つまり，教師データに含まれないプログラムをテストデータとした．

本稿の本来の目的は，名前難読化され，意味のない名前に変更された名前の復元である．一方で，名前難読化は名前の変更以外の変更は行わない．そこで，本実験ではテストデータの名前難読化は行わず，推薦された名前と元のメソッド名を比較することで復元率を測定する．

取得したテストデータに対しても，教師データ構築の時と同じく分析し，メソッド名と呼び出しメソッドを取り出した．ただし，復元の対象となるメソッドは，$1 \leq |F| \leq 5$ を満たすメソッドであることとした．

4.3 教師データ，テストデータの加工

4.3.1 テストデータの絞り込み

本研究では，特定の名前を持つメソッドに対してのみ復元を行った．復元を行ったメソッドは，`accept`，`evaluate`，`execute`，`invoke`，`put`，`readObject`，`writeObject` の 7 つである．この 7 つを選んだ理由は，`java.lang.Object` クラスのメソッドではなく，かつ教師データ，テストデータともに十分な数が存在するメソッドであったためである．

7 つのメソッドに対して，呼び出しメソッドの数に分けて，それぞれ復元する．各メソッドの復元の際に使用する教師データは，元の教師データから，n_i が各メソッド名を持つものにフィルタリングした教師データである．すなわち，$\mathcal{M}_{\texttt{accept}} = \{M_i | M_i \in \mathcal{M} \wedge n_i = \texttt{accept}\}$ の部分集合を構築した．そして，$\mathcal{M}_{\texttt{accept}}$ を `accept` メソッドの実験に用いる教師データとしている．他のメソッドについても同様に，教師データ $\mathcal{M}$ から部分集合 $\mathcal{M}_{\text{メソッド名}}$ を構築して実験を行った．また，推薦を行う際の類似度の閾値 ϵ は 1.0（完全一致）とした．

[5] https://oss.sonatype.org/content/repositories/releases

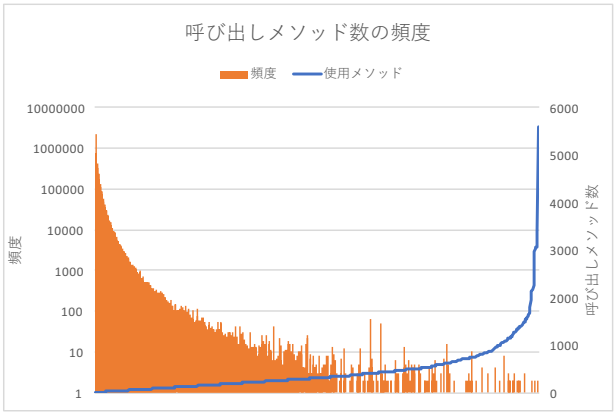


図 5: 呼び出しメソッド数の頻度

表 1: `readObject` の F の頻度上位 5 位

順位	割合	F
1	25.2%	`ObjectInputStream#defaultReadObject`
2	7.8%	`ObjectInputStream#defaultReadObject` `ObjectInputStream#readObject`

3	1.7%	`ObjectInputStream#defaultReadObject`
		`ObjectInputStream#readObject`
		`ObjectInputStream#readObject`

4	1.6%	`ObjectInputStream#readObject` `ObjectInputStream#readObject`

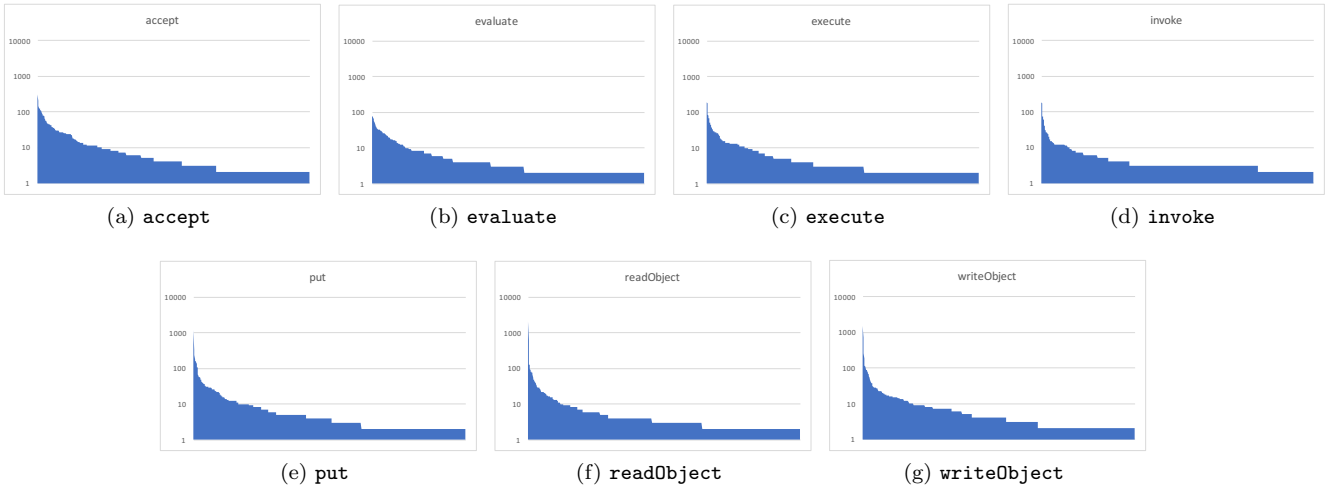
5	1.5%	`ObjectInputStream#defaultReadObject`
		`ObjectInputStream#readInt`
		`ObjectInputStream#readObject`

4.3.2 教師データのフィルタリング

$\mathcal{M}_n$ では，メソッドのシーケンス F が重複する場合がある．F が重複している場合，複数のメソッドで同じ呼び出しをしていることを表している．すなわち，重複数が多いほど F の出現頻度が高く，同じ名前のメソッドで一般的に使われていると考えられる．逆に出現頻度の低いものは，同じ名前のメソッドで呼び出しているメソッドとしては一般的ではなく，このようなデータを教師データに使うことは望ましくない．稀有な例を教師データとして使用すると，誤った推薦が行われるおそれがあるからである．

図 6 に各メソッド名における，F の出現頻度を示す．横軸が各 F であり，縦軸がその頻度を対数で示している．なお，F は出現頻度順にソートしており，出現回数が 1 回であったものは省略している．図 6a `accept`，図 6b `evaluate` は比較的なだらかなグラフであることがわかる．しかし，それ以外のメソッドである，図 6c〜6g は突出して出現頻度の高いシーケンスがあることがわかる．

例えば，$\mathcal{M}_{\texttt{readObject}}$ における F の頻度の上位 5 つを表 1 に示す．表の第 1 列は順位，第 2 列が $\mathcal{M}_{\texttt{readObject}}$



(a) `accept` (b) `evaluate` (c) `execute` (d) `invoke`

(e) `put` (f) `readObject` (g) `writeObject`

図 6: 各メソッドの教師データの出現頻度

表 2: `evaluate` の復元結果 ($t = 1$)

$\lvert F\rvert$	$\mathcal{T}$(`evaluate`) 推薦数	非推薦数	割合	$\overline{\mathcal{T}(\texttt{evaluate})}$ 推薦数	非推薦数	割合
1	14	0	100.0%	2408	4050	37.3%
2	11	0	100.0%	381	2004	16.0%
3	8	1	88.9%	131	1068	10.9%
4	1	2	33.3%	168	799	17.4%
5	3	2	60.0%	166	706	19.0%

表 3: `readObject` の復元結果 ($t = 1$)

$\lvert F\rvert$	$\mathcal{T}$(`readObject`) 推薦数	非推薦数	割合	$\overline{\mathcal{T}(\texttt{readObject})}$ 推薦数	非推薦数	割合
1	11	0	100.0%	976	5485	15.1%
2	17	1	94.4%	158	2220	6.6%
3	12	0	100.0%	0	1196	0.0%
4	4	1	80.0%	121	844	12.5%
5	7	4	63.6%	23	843	2.7%

に占める当該 F の割合，第 3 列が実際の F の内容を示している．なお，$|\mathcal{M}_{\texttt{readObject}}| = 8,145$ であった．

表 1 から，全体の 37.8%の処理が上位 5 位までで確認できる．これは，`readObject` が非常に典型的なメソッド呼び出しを持つことが伺える．表 1 中の F の処理が見つかれば，そのメソッドは `readObject` と名付けても問題はないと考えられる．一方，$\mathcal{M}_{\texttt{readObject}}$ 中の F の頻度が下位のものを見ると，$F = \{\texttt{Method\#invoke}\}$ が $\mathcal{M}_{\texttt{readObject}}$ 中に 2 件存在した．この F は全体の 0.02% でしかなく，一般的とは言えない．このように出現頻度の低い処理と類似しても，`readObject` である可能性は低い．

以上のことから，一般的でない教師データを除外することで，誤推薦の頻度を下げることを狙う．そのため，教師データ構築時に，F の出現頻度に閾値 t を設けた．フィルタリングを $\mathcal{F}(\mathcal{M}_n, t) = \{M_i | M_i \in \mathcal{M}_n \wedge freq(F_i, \mathcal{M}_n) \geq t\}$ と定義する．$freq(F_i, \mathcal{M}_n)$ は $\mathcal{M}_n$ 中に含まれる F_i の数を表す．なお，$\mathcal{F}(\mathcal{M}_n, 1) = \mathcal{M}_n$ となる．

4.4 名前の復元（閾値 $t = 1$）

本節では，F の出現頻度の閾値を $t = 1$，すなわち，教師データのフィルタリングなしで復元を試みた．また，メソッド名 n の評価のため，テストデータを以下のように $\mathcal{T}(n)$ と $\overline{\mathcal{T}(n)}$ に分割した．$\mathcal{T}(n) = \{M_i | M_i \in \text{mi}(p') \wedge n_i = n\}$，$\overline{\mathcal{T}(n)} = \{M_i | M_i \in \text{mi}(p') \wedge n_i \neq n\}$．

`evaluate` の復元結果を表 2 に示す．第 1 列が $|F|$ の長さを表しており，推薦数が推薦されたメソッドの数，非推薦数が推薦されなかったメソッドの数を示している．また割合が，推薦された割合，すなわち，推薦数/(推薦数+非推薦数) である．この場合，教師データは $\mathcal{M}_{\texttt{evaluate}}$ であり，$\mathcal{T}$(`evaluate`) に対して適用する場合，テストデータの情報から，名前が復元できるかを確認している．一方，$\overline{\mathcal{T}(\texttt{evaluate})}$ に対しての適用では，誤推薦がないことを確認したいため，推薦数が 0 に近づくことが望まれる．つまり，$\mathcal{T}$(`evaluate`) 列の推薦数が正しく推薦できたもの，$\overline{\mathcal{T}(\texttt{evaluate})}$ 列の推薦数が誤推薦のものを表している．

同じ条件で，$\mathcal{T}$(`readObject`) と $\overline{\mathcal{T}(\texttt{readObject})}$ に対して，$\mathcal{M}_{\texttt{readObject}}$ を適用した結果を表 3 に示す．

結果を見ると，$\mathcal{T}$(`evaluate`) に対して `evaluate` が推薦されている割合は最高で $|F| = 1, 2$ のとき 100%となっている．また，$|F| = 4$ の時が一番低く，33.3%であり，$|F| = 1, ..., 5$ の平均は，76.4%であった．同じく `readObject` の推薦された割合の平均が 87.6%と高い値となっている．また，第 4.3.1 節で述べた 7 つのメソッドの推

表 4: evaluate の復元結果（$t = 35$）

$\lvert F\rvert$	$\mathcal{T}$(evaluate) 推薦数	非推薦数	割合	$\overline{\mathcal{T}(\texttt{evaluate})}$ 推薦数	非推薦数	割合
1	4	10	28.6%	259	6199	4.0%
2	5	6	45.5%	36	2349	1.5%
3	5	4	55.6%	41	1158	3.4%
4	0	3	0.0%	25	942	2.6%
5	0	5	0.0%	0	872	0.0%

表 5: readObject の復元結果（$t = 25$）

$\lvert F\rvert$	$\mathcal{T}$(readObject) 推薦数	非推薦数	割合	$\overline{\mathcal{T}(\texttt{readObject})}$ 推薦数	非推薦数	割合
1	11	0	100.0%	3	6458	0.0%
2	14	4	77.8%	0	2378	0.0%
3	8	4	66.7%	0	1196	0.0%
4	1	4	20.0%	0	965	0.0%
5	4	7	36.4%	0	866	0.0%

薦された割合は，77.5%となり，半数以上で正答している．すなわち，$|F| = 1, ..., 5$ であれば，高い確率でメソッド名を戻すことができる．しかし，readObject の $|F| = 3$ の場合を除いて，$\overline{\mathcal{T}(n)}$ に対する誤推薦が多い．誤推薦の割合は，$\overline{\mathcal{T}(\texttt{evaluate})}$ で平均 20.1%，$\overline{\mathcal{T}(\texttt{readObject})}$ で平均 7.4%であった．また，実験を行った 7 つのメソッドの平均は，18.0%となり，$|F| = 1, ..., 5$ では 5 個に 1 個の割合で誤推薦が起こっていた．

このように，教師データをフィルタリングせずに復元を行うと，誤ったメソッド名を多く推薦することになる．誤推薦が多くなると，相応しいメソッド名が復元候補 N_n に含まれたとしても，$|N_n|$ が大きくなるため，相応しいメソッド名を見つけ出すのが困難になる．

4.5 名前の復元（閾値 $t \geq 2$）

本節では，$\mathcal{F}(\mathcal{M}_n, t)(t \geq 2)$ を用いて，第 4.4 節と同じテストデータに対して復元を行った．閾値 t の値は，F の頻度が突出して大きな値になるように，図 6 を元に著者の判断で定めた．invoke の閾値の例を図 7 に示す．図中の赤線が閾値 $t = 13$ である．同じように，各 $\mathcal{M}_n$ に対して，図 6 に示す頻度グラフから，著者の判断で閾値 t を設定した．

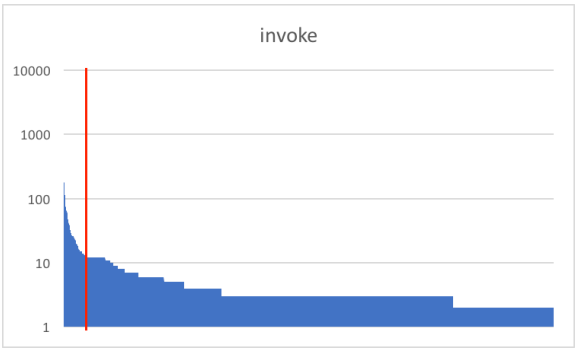


図 7: invoke における閾値設定の例

evaluate における，$\mathcal{F}(\mathcal{M}_{\texttt{evaluate}}, 35)$ での復元結果を表 4 に示す．同じく，readObject での $\mathcal{F}(\mathcal{M}_{\texttt{readObject}}, 25)$ を用いた復元結果を表 5 に示す．閾値を設定し，フィルタリングした教師データで復元をすると，誤推薦の割合の平均が $\overline{\mathcal{T}(\texttt{evaluate})}$ で 2.3%，$\overline{\mathcal{T}(\texttt{readObject})}$ では 0.0% と非常に低い値となった．特に，readObject の誤推薦の割合は，最も高い場合においても 0.05%（$|F| = 1$）という値になった．

$\overline{\mathcal{T}(\texttt{readObject})}$（$|F| = 1$）において，readObject を推薦したメソッドの元の名前を調べると，readResolove，readObjectNoData，readUTF であることがわかった．メソッドの名前から推測されるメソッドの振る舞いから考えると，readObject が推薦されても問題はないと考えられる．また，実験に用いた 7 つのメソッドにおける誤推薦の割合は平均 5.2%となり，$t = 1$ の時と比べて大幅に誤推薦を減らすことができた．上記の結果より，$\mathcal{F}(\mathcal{M}_n, t)(t \geq 2)$ で復元を行うと，誤推薦の数が減り，また，誤推薦をした場合においても似た意味を持つメソッド名を推薦できることがわかる．

一方で，元の名前を推薦できた割合は，$\mathcal{T}$(evaluate) で平均 25.9%，$\mathcal{T}$(readObject) の割合は平均で 60.2% と $t = 1$ に比べて低くなった．今回の実験で設けた閾値 t では，元のメソッド名を推薦できた割合は，本実験で復元した 7 つのメソッドで 39.9%となり，$t = 1$ の 77.5%に比べて大幅に落ちた．すなわち，正答率が半分程度まで落ち込んでいる．しかし，$t \geq 2$ において非推薦であったメソッドの詳細を見ると，F の一部が教師データの出現頻度上位の F と一致しており，類似度の閾値 ϵ を下げることで，正しい推薦が行われると考えられる．もちろん ϵ を下げることで，正しい推薦の割合も増えると考えられるが，先の結果から似た意味を持つメソッド名が推薦されると考えられる．よって正しく閾値を設定することで，正答率を上げることが期待出来る．

4.6 提案手法の限界

$\mathcal{M}_n$ に対して，$t = 1$ の場合と，$t \geq 2$ の場合で，結果に大きな差が出た．$t \geq 2$ では，誤推薦の割合を 18.0%から 5.2%と大きく下げられたものの，同時に正しい推薦も 77.5%から 39.9%と下がった．

ただし，第 4.4 節，第 4.5 節での誤推薦の結果を見ると，似た意味のメソッド名を推薦している場合もあった．例えば，invoke メソッドの誤推薦結果に，execute や perform といった名前が推薦されていた．この結果を見ると，一概に誤った結果とは言えない．しかし，一番相応しい名前を機械的に決める方法は今のところ存在しない．そのため，推薦結果を人が確認し，メソッド名を決

める必要がある．

本稿では，$\mathcal{M}_n$ に対して，出現頻度で閾値 t を設定した．しかし，その閾値の正当性に関しては評価できていない．さらに，閾値の決定は，図 6 の各メソッドの頻度グラフから著者の判断で設定した．現実的には，教師データの候補となるメソッドが大量にあるため，全てのメソッドに対して，同じように閾値を決定するのは現実的ではない．そのため，閾値を自動的に設定する方法が必要である．

本稿の取り組みでは，$\mathrm{sim}(F_i, F'_j)$ で求める類似度 s_i の閾値は $\epsilon = 1$ のみであった．復元の結果から，$|F|$ の値が大きくなると誤推薦の割合は下がるが，正しい推薦の割合も下がっていることがわかる．これは $|F|$ の値が大きくなると，$s_i = 1$ となること自体が稀になるからであると考えられる．ϵ を下げることで，部分一致した場合にもメソッド名を推薦できるが，誤推薦も当然増加すると予想できる．したがって，より良い ϵ を求めることも今後必要になる．

5　関連研究

難読化の逆変換に関連したいくつかの研究が行われている．コントロールフロー難読化の逆難読化 [5] や名前難読化の逆変換 [6] が提案されている．

Cimato らが行った名前難読化の逆変換では，変数名に着目した逆変換を行っている [6]．しかし，変数名に型情報を付与したのみであり，評価も行われていない．本提案手法の対象はメソッド名であり，Cimato らの手法と対象が異なる．

逆変換の前段階として，どのような難読化手法が適用されているのかを特定しようとする研究も行われている [7, 8]．この手法は，メソッドの命令列に着目し，難読化手法自体の特徴を見つけ出そうとしている．本提案手法では，名前難読化以外の難読化手法が適用されている場合，逆変換が困難になる場合もある．そのため，逆難読化を行う前に適用されている難読化手法を特定することは重要である．

鬼塚ら，柏原らは，一般的なプログラムでのメソッド名の推薦法を提案している [2, 9, 10]．あるメソッドから呼び出しているメソッドの情報などを使い，相関ルールを用いてそのメソッドの名前を推薦している．提案手法と目的が大きく異なるものの，手法自体は類似している．彼らの手法で得られた知見は，本研究にも応用可能である．

6　まとめと今後の課題

本稿では，名前難読化された Java プログラムのメソッドに対して，メソッド名の推薦を行う手法を提案した．提案手法は，教師データとして大量の Java プログラムを用意し，教師データ $\mathcal{M}$ と名前難読化されたプログラムを比較する．双方から呼び出しメソッドを抽出し，比較することで，類似した教師データからメソッド名を推薦する．また，教師データには復元に適さないデータが含まれているため，教師データに閾値 t を設定することでフィルタリングを提案した．

閾値を設定しなかった教師データを用いて復元を行うと，誤推薦された割合は 18.0%だった．しかし，閾値を設定した教師データで復元を行うと，誤推薦の割合は 5.2%となり大幅に削減できた．また，誤推薦された場合にも，推薦されたメソッド名は，元のメソッド名が持つ意味と似たものを推薦できる場合が多かった．一方で，正しいメソッド名を推薦できる割合がおおよそ半分になった．しかし，推薦できなかった場合においても，教師データに存在するデータの一部と一致する場合が多く，類似度の閾値を適切に変更することで改善が期待出来る．今後は教師データの質の向上と，類似度の閾値の適切な設定が課題といえる．

今回は簡略化のため，復元には，呼び出しメソッドという一つの情報のみを用いた．しかし，名前難読化に影響を受けない他の情報も存在する．例えば，メソッドの引数の型，数，返り値の型などである．それらを併用したより質の高い名前の復元に，今後も取り組んでいく．

参考文献

[1] Paul M. Tyma. Method for renaming identifiers of a computer program. United States Patent 6,102,966, August 2000. Filed: Mar.20, 1998.

[2] 鬼塚勇弥, 早瀬康裕, 石尾隆, 井上克郎. ソースコード中に出現する動詞-目的語関係を利用したメソッド名の命名支援手法. 信学技報, 第 SS2011-57 巻, pp. 1–6, 2012.

[3] 門田暁人, 高田義広, 鳥居宏次. ループを含むプログラムを難読化する方法の提案. 電子情報通信学会論文誌 (D-I), Vol. J80-D-I, No. 7, pp. 664–652, July 1997.

[4] Steven Raemaekers, Arie van Deursen, and Joost Visser. The maven repository dataset of metrics, changes and dependencies. In *Proc. of the 10th Working Conference on Mining Software Repositories (MSR 2013)*, pp. 221–224, 2013.

[5] Sharath K. Udupa, Saumya K. Debray, and Matias Madou. Deobfuscation: reverse engineering obfuscated code. In *12th Working Conference on Reverse Engineering*, November 2005.

[6] S. Cimato, A.De Santis, and U. Ferraro Petrillo. Overcoming the obfuscation of java programs by identifier renaming. In *Journal of Systems and Software*, Vol. 78, pp. 66–72, 2005.

[7] 匂坂勇仁, 玉田春昭. 適用保護手法特定の試み －不自然さ評価方法を用いて－. 信学技報 SS2015-21, pp. 63–68, July 2015.

[8] Hayato Sagisaka and Haruaki Tamada. Identifying the applied obfuscation method towards deobfuscation. In *Proc. 15th IEEE/ACIS International Conference on Computer and Information Science (ICIS 2016)*, pp. 873–878, July 2016.

[9] 柏原由紀, 石尾隆, 井上克郎. Java メソッドの動作を表現する動詞の自動推薦手法の評価. 情報処理学会論文誌, Vol. 56, No. 9, pp. 1900–1904, September 2015.

[10] 柏原由紀, 石尾隆, 井上克郎. 動詞に着目した相関ルールを利用するメソッド名の命名支援手法の評価. 情報処理学会研究報告, 第 2015-SE-187 巻, 2015.