情報処理学会論文誌 Vol.61 No.2 454–473 (Feb. 2020)

大量のソフトウェアを対象にしたソフトウェアバースマークによる盗用検出——全文検索システムを用いた検査対象の絞り込み手法

中村 潤[1] 玉田 春昭[2,a)]

受付日 2019年1月21日, 採録日 2019年11月7日

概要：大量のプログラムからソフトウェアの盗用を発見するために，ソフトウェアバースマークが提案されている．バースマークは，プログラム中の特徴を抽出し比較を行い盗用を発見する技術である．従来のバースマークで想定している対象のプログラム数は数百から数千程度であり，それを超えて比較する場合，現実的な時間では比較できない場合もある．そこで本稿では，対象となるプログラムの大幅なスケールアップかつ，盗用検出に要する処理時間の短縮を目的とする．そのために，対象プログラムの比較の前に，精度は高くないが高速に比較できる手法を用いて，無関係なプログラムを除外することを考える．これを絞り込み段階として，従来の抽出段階，比較段階の間に導入する．提案手法に基づき，バースマーク絞り込みシステム *Mituba* を構築し，実験を行った．評価項目は所要時間，絞り込み率，誤検出，検出漏れ，精度，そして，保存性の 6 項目である．結果は，盗用か否かを判定するための閾値が 0.2 のとき，所要時間は従来の 40%以下に抑えられ，80%以上のプログラムが無関係と判定された．残ったプログラムのうち，誤検出は 90%程度と非常に高いものの，検出漏れは 0%であり，精度も 70%以上となっている．また保存性評価においても，一番強力な難読化が施された場合であっても 80%以上のプログラムを見つけ出せ，良好な結果を示している．これらの結果をもとに最適な閾値を議論した結果，標準的には 0.6 程度の閾値が最適であるが，ユーザの問題設定によっては，閾値が 0.2 でも本手法は有効であることを示した．

キーワード：ソフトウェアバースマーク，スケールアップ，全文検索システム

The Software Birthmarking Method for a Huge Number of Programs to Detect the Theft — Narrowing the Targets with the Fulltext Search Engine

Jun Nakamura[1] Haruaki Tamada[2,a)]

Received: January 21, 2019, Accepted: November 7, 2019

Abstract: The software birthmarks were proposed for detecting the software theft from a large number of suspected programs. The birthmark is a technique to compare the extracted characteristics as the software birthmarks, and compute similarities between two birthmarks. The conventional birthmarks require a much time by increasing the target programs for comparison. Therefore, it is the bottleneck of the conventional birthmarking technique. In this paper, we propose a method for reducing total processing time by introducing the narrowing phase between conventional extracting and comparison phases. The narrowing phase employs the fast but rough comparison algorithm in order to eliminate unrelated programs. We developed the narrowing system for the birthmark, named *Mituba*, then conducted the experimental evaluation with *Mituba*. The evaluation points are required time, narrowing rate, false negatives, false positives, accuracy, and preservation property of the birthmarks. The results are, in the threshold for deciding the theft is 0.2, the required time reduced to 20%, the system successfully narrowed 80% programs. The false negatives in the rest of the programs are about 80%, it was quite high. However, the false positives were 0%, and the accuracies were over 70%. Additionally, the proposed method satisfied the preservation property by 80% in the strongest obfuscation method in the experiments. The results were totally quite well. From the results of the experimental evaluations, we discussed the suitable threshold. In ordinal case, the suitable threshold is 0.6, however, the proposed method is acceptable in the threshold is 0.2 in the user's discretion.

Keywords: software birthmarks, scaling up, full-text search engine

1. はじめに

ソフトウェアの盗用はいつ，どこで行われるか予測できないため，発見すること自体が非常に困難である．盗用を発見するための手法として，Tamada らによってソフトウェアバースマークが提案されている [1], [2]．バースマークは，プログラムから変更が困難な特徴を抽出・比較することで類似度を算出する．その類似度を用いて盗用か否かを判定する技術であり，バイナリを対象とする，高速で動作するなどの特徴がある [1], [2], [3], [4]．しかし，従来のバースマーク手法には，次の 2 つの問題がある．まず，バースマークでの比較を行うために対象の収集やツールの導入など事前作業が必要となる点である．もう 1 つは，現実的な時間での検査という観点において，従来手法での検査対象数には限界がある点である．

一般的な開発者がバースマークによる検査を行う場合を考えたとき，実際に検査を行う前にいくつかのハードルがある．多くの場合，開発者自身が開発したプログラムが盗まれたことを想定し，自分のプログラムと似たものを見つけ出すためにバースマーク技術が用いられる．仮に盗用後のプログラムの目星がついているのであれば，より詳細な比較が可能なバースマーク以外の技術，たとえばコードクローン [5] などを利用する方が良いであろう．一方，目星がついていない場合，世の中のどこで盗用が行われているか分からないため，非常に膨大な数のプログラムを調査する必要がある．そのためには，まず検査対象となるプログラムを集める必要がある．次に，バースマーク検査を行うツールをインストールし，ようやくバースマークによる検査が行えるようになる．存在するかどうかが不明な盗用を見つけるために，これらの作業を要求する従来のバースマーク手法は，実用化には大きなハードルがあるといえる．つまり，盗用をより効果的に発見するためには，準備を必要とせず，よりカジュアルに比較できる環境が必要となる．

バースマークは本来，盗用の疑いのあるプログラムを見つけることを目的としている．そのため，前提となるシナリオは，大規模なプログラム群を対象に，自分の持つオリジナルなプログラムと似たものがないかを見つけるものである．世の中のすべてのプログラムを調べることはできないまでも，npm[*1]や Maven Central Repository[*2]（MCR）など，何らかの基準で集められたリポジトリ内のプログラムを調べることは重要であろう．しかし，これらのリポジトリに限ったとしても，現実的な時間で比較が終わらない可能性が高い．本来のバースマークが高速に動作するとはいえ，逐次的な比較では対象数の爆発的な増加への対応に限界があるためである．なお，modulecounts.com[*3]によると 2018 年 10 月 18 日時点で，npm で 711,676，MCR で 252,432 のプログラムが登録されている．加えて，1 日あたり平均で，npm で 539 プログラム，MCR で 120 プログラム増加している．

世の中に膨大な数のプログラムが存在するとはいえ，その大半は盗用ではない．そのため，ラフで高速な比較により無関係なプログラムを取り除いてから，従来のバースマークによる比較の実行を考える．このように対象を絞り込むことで，バースマークの処理に要する全体的な時間の短縮を狙う．我々はこの考えをもとに，全文検索システムを用いたバースマーク検査の Web システム *Mituba* を提案する．全文検索システムは，登録された膨大な量の文書を，少数のキーワードから絞り込むためのシステムである．また，全文検索システムで検索するためには，あらかじめバースマークを登録しておく必要がある．そのために，多数のプログラムを収集し，バースマーク抽出を経て *Mituba* に登録しておく．これにより，一般ユーザは事前準備が不要となり，カジュアルな検査が可能となる．

なお，本稿で対象とするソフトウェアは Java で作成されたものとしている．そのうえで，ソフトウェアは jar ファイル，プログラムは jar ファイルに含まれる 1 つのクラスファイルを指すものとする．ただし，本稿の対象を Java に限定しているのは，Java 向けのバースマークを利用していることからの制限である．そのため，他言語向けのバースマークであっても提案手法の適用は同様に可能である．

1.1 本稿の位置付け

我々の研究グループでは，バースマークの比較時間を短縮するため次のような試みを行った．

- バースマークを Fuzzy Hash 化し，単純化する方法を提案した [6], [7]．しかし，この方法も逐次的に比較するため，比較対象の爆発的な増加に対する問題には未対応である．
- そこで，全文検索システムを用いた比較時間の短縮法のコンセプトを提案し [8]，提案手法での評価を行った [9]．ただし，これらでは全文検索システムのスコア算出方法については考慮していない．全文検索システムのスコア算出には複数のアルゴリズムが用意されているものの，デフォルトのアルゴリズムである編集距離を用いている．
- 次に，全文検索システムでのスコア算出方法の比較を行い，BM25 が最適であることを求めた [10]．

本稿では文献 [10] までに得られた知見をふまえ，全文検索

[1] 京都産業大学大学院
Graduate School of Kyoto Sangyo University, Kyoto 603–8555, Japan
[2] 京都産業大学
Kyoto Sangyo University, Kyoto 603–8555, Japan
[a)] tamada@cc.kyoto-su.ac.jp

[*1] https://www.npmjs.com/
[*2] http://central.maven.org/maven2/
[*3] http://www.modulecounts.com/

システムのスコア算出方法を BM25 として，実験を再構成している．そのうえで提案手法を実際に使うときに必要になるであろう，最適な閾値を求めるための議論を行っている．

1.2 本稿の構成

以降，本稿では，2 章でバースマークなどの定義を行う．続く 3 章では提案手法について述べ，4 章で実装を説明する．5 章では，提案手法の評価について記述する．そして，提案手法の結果を受け，6 章で提案手法の閾値の議論を行い，7 章で関連研究について述べる．最後に，8 章で本稿をまとめる．

2. 準備

2.1 バースマークの定義

提案手法を説明する前に，ソフトウェアバースマークの定義を説明する．ソフトウェアバースマークは Tamada らによって次のように定義されている [2]．

定義 1（バースマーク） p，q を与えられたプログラムとし，$\mathcal{B}_f(p)$ をプログラム 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ならば，$\mathcal{B}_f(p)$ を p の**バースマーク**であるという．

条件 1 $\mathcal{B}_f(p)$ はプログラム p のみから得られる．

条件 2 q が p のコピーであれば，$\mathcal{B}_f(p) = \mathcal{B}_f(q)$．

条件 1 は，バースマークがプログラムの付加的な情報ではなく，p の実行に必要な情報であることを示す．すなわち，バースマークは電子透かしのように付加的な情報を必要としない．条件 2 はコピーされたプログラムからは同じバースマークが得られることを示す．対偶より，もしバースマーク $\mathcal{B}_f(p)$ と $\mathcal{B}_f(q)$ が異なっているならば，q は p のコピーではないことを意味する．

また，バースマークは以下の保存性，弁別性の 2 つの性質を満たすことが望まれる．

性質 1（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，$\mathcal{B}_f(p) = \mathcal{B}_f(p')$ を満たす．

性質 2（弁別性） 同じ処理を行うプログラム p と q がまったく独立に実装された場合，$\mathcal{B}_f(p) \neq \mathcal{B}_f(q)$ を満たす．

多くの場合，盗用者は盗用の事実を隠すために，元のプログラムを等価変換により変更すると考えられる．典型的な等価変換の手段は，プログラムの難読化や最適化など，入出力仕様を保ったままプログラムを自動変換する方法である．つまり保存性とは，何らかの方法によりプログラムを変更したとしても，バースマークに変化がないことを表す性質である．

一方で弁別性は，まったく独立に作成されたプログラムは，同じ仕様であっても区別できることを示す．世の中には似た機能を持つソフトウェアが存在する．それらは多くの場合，異なる開発者によって互いに参照せずに作成される．この場合，機能が非常に酷似していたとしても，盗用ではないためバースマークによって区別できることが望ましい．これを表す性質が弁別性である．なお，弁別性は条件 2 に示した命題の裏である．

もちろん，この 2 つの性質を完全に満たすバースマークを提案することは困難である．そのため，実用上はユーザの判断により適宜強度を設定する必要がある．また，条件 1 より，プログラムに特別な情報の追加なしにバースマーク情報が構築できる点も注目すべき箇所である．

なお，実行時情報に基づいた動的バースマークも提案されている [11], [12]．しかし，動的バースマークは抽出の自動化自体が困難であり，本稿のアプローチをそのまま適用することは難しい．そのため本稿では議論しない．

2.2 バースマークの種類

2.1 節に示したバースマークの定義に従い，異なる種類のバースマークが提案されている．これらは，プログラムの着目する箇所，もしくは，情報の構成方法が異なる．たとえば，制御フロー [13]，opcode シーケンス [3]，プログラムの構造 [14]，データフロー [15]，振舞い [4] などから構築されるバースマークが提案されている．

一方，抽出した情報も構成方法により，異なるバースマーク情報として定義されることもある．たとえば，opcode シーケンスをそのまま利用する方法，ベクトルとして利用する方法 [16]，また，k-gram として構成する手法 [3] などがある．バースマーク情報の構成は，大きく，順列（順序あり集合），集合（順序なし集合），ベクトル，グラフに分類できる．

2.3 バースマークの類似度

バースマークの種類ごとに類似度計算法が定義されている．いい換えれば，バースマーク抽出法 f で得られたバースマーク $\mathcal{B}_f(p)$ と $\mathcal{B}_f(q)$ の比較には定義された $\mathrm{sim}_f(\mathcal{B}_f(p), \mathcal{B}_f(q))$ を用いる必要がある．なお，$\mathrm{sim}_f(\mathcal{B}_f(p), \mathcal{B}_f(q))$ の値域は $[0, 1]$ である．

類似度が 0 であれば，両プログラムは互いにコピー関係ではない可能性が非常に高いことを意味し，類似度が 1 であれば，どちらかのプログラムがもう一方のコピーである疑いが強いことを意味する．ただし，どの程度 1 に近ければコピーであるのかを判定するため，多くの場合，閾値として ε が導入されている．ε のとりうる範囲も sim_f の範囲と同じく $[0, 1]$ である．もし 2 つのバースマークの類似度が閾値より大きいとき，p もしくは q のいずれかが他方から盗用されていることを意味する．そして，類似度の結果を以下の 3 つのグループに分類する [17]．

$$\mathrm{sim}_f(\mathcal{B}_f(p), \mathcal{B}_f(q)) = \begin{cases} \geq \varepsilon & \textit{copy relation} \\ \leq 1 - \varepsilon & \textit{no copy relation} \\ \text{otherwise} & \textit{inconclusive} \end{cases}$$

上記のように，$\mathrm{sim}_f(\mathcal{B}_f(p),\mathcal{B}_f(q)) \geq \varepsilon$ の場合は，プログラム p と q はコピー関係の疑いが強いことを表し，$\mathrm{sim}_f(\mathcal{B}_f(p),\mathcal{B}_f(q)) \leq 1-\varepsilon$ の場合は，コピー関係にない可能性が高いことを表す．そして，それ以外（$1-\varepsilon < \mathrm{sim}_f(\mathcal{B}_f(p),\mathcal{B}_f(q)) < \varepsilon$）の場合は，バースマークではコピー関係を判定できないことを示している．なお，過去の慣例では $\varepsilon = 0.75$ とすることが多い [18]．

2.4 バースマークによる盗用検出プロセス

ソフトウェアバースマークの目的は，盗用を証明することではなく，盗用の疑いがあるソフトウェアを発見することである．そのため，バースマーク手法は，コピーの疑いがあるソフトウェアを大量のプログラムから見つける必要がある．仮に，プログラムから抽出したバースマーク情報が似ていた場合，元のプログラムどうしを別の方法で詳細に調査し，盗用を証明する必要がある．バースマーク手法は，あらかじめ情報を埋め込まないため，偶然の一致がどうしても避けられないことから必要な作業となる．一方，非常に小さなプログラムの場合，比較に十分な情報が抽出できない場合もある．そのため，抽出したバースマークの要素数などでフィルタリングを行う場合もある．

バースマークによって盗用の疑いのあるプログラムを検出するプロセスは，暗黙的に以下の 4 つの工程が含まれる．

(1) 収集段階
(2) 抽出段階
(3) 比較段階
(4) 検査段階

これらの工程を**図 1** に図示する．

2.4.1 収集段階

まず収集段階では，調査対象となるプログラムを収集する．収集に先立ち，盗用された可能性のあるオリジナル（原告）プログラムが手元にあるものとする．そして，様々なサイトから盗用の疑いのあるプログラム（被告）が含まれるように収集する．なお，どこで盗用が行われているかを事前に検知することはできない．そのため，できるだけ多くの被告を収集することで発見の可能性を上げる．

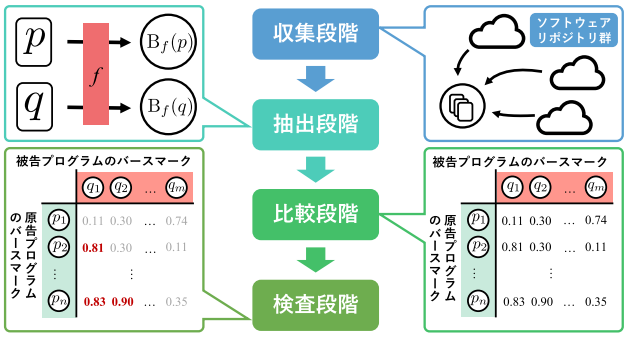


図 1 バースマーク手法の典型的な実行手順
Fig. 1 The procedure of the birthmarking method.

より多くのプログラムを用意するため，ソフトウェアリポジトリからプログラムを収集する場合もある．近年，各プログラミング言語には，複数のソフトウェアリポジトリが用意されている．たとえば，Java 言語向けには，Maven repository [*4] があり，Ruby には，rubygems [*5] が存在する．これらのソフトウェアリポジトリに含まれているプログラムを調査対象とすることで，多くのプログラムを用意できる．また，ソフトウェアリポジトリ以外にも，GitHub [*6] などのリポジトリサービスの利用も有用であろう．

2.4.2 抽出段階

第 2 の工程である抽出段階では，対象となる原告プログラム群 $P = \{p_1, p_2, \ldots, p_n\}$ の各プログラムと被告プログラム群 $Q = \{q_1, q_2, \ldots, q_m\}$ から，バースマーク抽出法 f によりバースマークを抽出し，バースマーク情報 $\mathcal{B}_f(p_i)$ $(1 \leq i \leq n)$ と $\mathcal{B}_f(q_j)$ $(1 \leq j \leq m)$ を得る．この工程は，大量のプログラムからバースマークを抽出する必要があるため，一般に非常に膨大な時間を要する．

2.4.3 比較段階

続く比較段階では，抽出段階で得られた $\mathcal{B}_f(p_i)$ $(1 \leq i \leq n)$ と $\mathcal{B}_f(q_j)$ $(1 \leq j \leq m)$ を相互比較し，類似度を算出する．この段階の結果として，類似度のリストが得られる．リストの長さ，すなわち比較回数は nm 回である．すなわち，n，m が大きくなるに従い，比較時間が大きくなっていく．

2.4.4 検査段階

最後の検査段階では，比較段階で求めた類似度のリストの中から，どの被告プログラムに盗用の疑いがあるのかの判断を行う．一般に類似度のリストから類似度が ε 以上となるペアを抜き出すことで，盗用の疑いのあるプログラムを見つける．しかし，異なるライブラリに同じ部品が含まれている場合もあり，高類似度のペアであることが，必ずしも盗用の疑いがあるわけではない．加えて，非常に小さいプログラムの場合は，偶然の一致を避けることが非常に難しい．これらのことをふまえてどのプログラムに盗用の疑いがあるのかの判断を行うのが，この検査段階である．

3. 提案手法

3.1 キーアイデア

従来のバースマーク手法を**図 2** に示す．図では，3 つのプログラム p と q_1，q_2 を対象にバースマーク手法を適用し，q_1，q_2 が p のコピーであるかを判定しようとしている．そのために，p，q_1，q_2 それぞれから $\mathcal{B}_f(p)$ と $\mathcal{B}_f(q_1)$，$\mathcal{B}_f(q_2)$ を抽出・比較して類似度を算出し，ε をもとに盗用か否かの判断を下している．しかし，ここでの問題は，大量のプログラムを対象に検査を行う必要がある点である．

[*4] https://mvnrepository.com
[*5] https://rubygems.org
[*6] https://github.com/

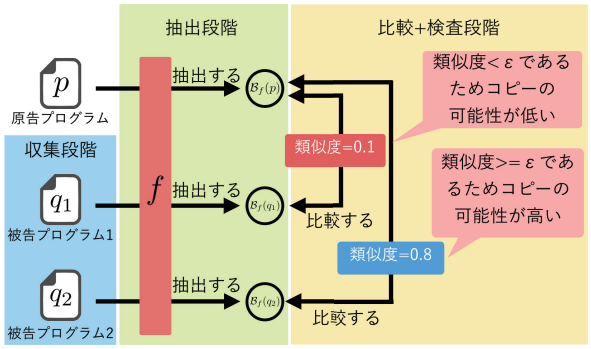


図 2 従来手法の実行手順
Fig. 2 The overview of the conventional birthmarking method.

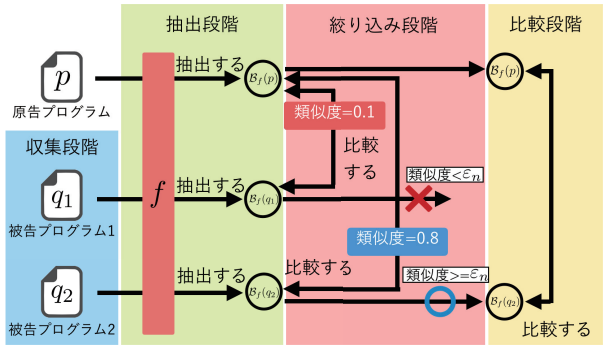


図 3 提案手法の実行手順
Fig. 3 The overview of the proposed birthmarking method.

この場合，抽出，比較それぞれに要する時間がごく短時間であったとしても，対象プログラムの数が増加するに従い，合計時間が増加することが問題となる．

従来手法では，少数のプログラムを検査するには十分であるものの，世の中のプログラムから盗用を見つけ出す用途には不十分である．一方で，世の中の多くのプログラムを検査することは重要である．どこに盗用されたプログラムが存在するか，事前に分からないためである．ただし，世の中のほとんどのプログラムは盗用ではない．

そこで，大量の被告プログラム群 $Q = \{q_1, q_2, \ldots, q_m\}$ から少数の原告プログラム群 $P = \{p_1, p_2, \ldots, p_n\}$ に似たプログラムを見つけ出すことを考える（$m \gg n$）．この場合でも，m が大きければ，Q の収集および，Q と P の相互比較に膨大な時間が必要である．そこで，Q の収集とバースマークの抽出を事前に行っておき，データベースに格納しておく．1 度リリースされたソフトウェアは，基本的に更新はない．更新がある場合はバージョンアップとして，別のバージョンのソフトウェアとして公開されるためである．つまり，いったんリリースされれば，バースマーク情報が変化しない．そのため，世の中の数多くのプログラムは，あらかじめ収集しバースマークを抽出しておくことで結果を保存できる．

加えて，抽出段階と比較段階の間に絞り込み段階を導入する．先ほど述べたように，世の中のほとんどのプログラムは盗用ではない，そのため，絞り込み段階で明らかに無関係のプログラムを取り除き，従来の比較段階で盗用の疑いのあるプログラムを見つけ出すことを目指す．このように行うことで，膨大な数のプログラムから盗用の疑いのあるプログラムをより高速に発見できるようになると期待できる．

提案手法の模式図を**図 3** に示す．絞り込み段階では，従来のバースマーク手法よりも簡易で高速な比較アルゴリズムでバースマークどうしの類似度を算出する．そして，この絞り込み段階には検査対象がより大規模になったとしても分散処理などを容易に導入できるような手法が求められる．これにより，比較段階での対象となる被告プログラムを減らすことを狙う．

このように収集・抽出段階の大半をあらかじめ済ませておき，比較段階の対象を絞り込み段階で絞り込むことで大量のプログラムを対象としたときの，バースマーク処理全体の処理時間の削減を狙う．

3.2 絞り込み段階での類似度計算法

絞り込み段階で導入する類似度計算手法は，従来の比較方法よりも簡易で高速な手法が求められる．そのため，本稿ではバースマークそれぞれに定義された類似度計算法について議論しない．個々のバースマークに特化した手法ではなく，あらゆるバースマークに対して共通して利用できる類似度計算法を導入し，$\mathrm{comp}(\mathcal{B}_f(p), \mathcal{B}_f(q))$ と記す．そして，提案手法の閾値として ε_n を導入し，次の式に従って盗用か否かを判定する．

$$\mathrm{comp}(\mathcal{B}_f(p), \mathcal{B}_f(q)) = \begin{cases} \geq \varepsilon_n & \textit{copy relation} \\ < \varepsilon_n & \textit{no copy relation} \end{cases}$$

3.3 絞り込み段階への全文検索システムの適用

絞り込み段階では，続く比較段階で対象となる被告を絞り込む．そのため，大量のバースマークを比較でき，被告数の爆発的な増加に対しても分散処理などの対応が容易に可能であるような手法が必要となる．そこで，クエリとして与えたキーワードに関連する文書を高速に絞り込む全文検索システムに着目する．

全文検索システムは，大量の文書集合から特定のキーワードに関連する文書を発見するためのツールである．一般的な全文検索システムは，大量の文書情報をデータベースに保持している．そして，検索には複数の単語をキーワードとして用いる．そのキーワードを含む文書を関連度（スコア）に応じてランク付けし，結果として示す．このとき，大量の文書をデータベースに保持していたとしても，結果は数秒程度で示される．これは，索引に対して検索を行い，関連しない文書のスコア算出を省略することで比較回数を減らしているためである [19]．索引は，ハッシュや木構造など検索しやすい形式であり，データベース上の文

図 4 提案手法の登録ステップ
Fig. 4 The procedure of the registration step.

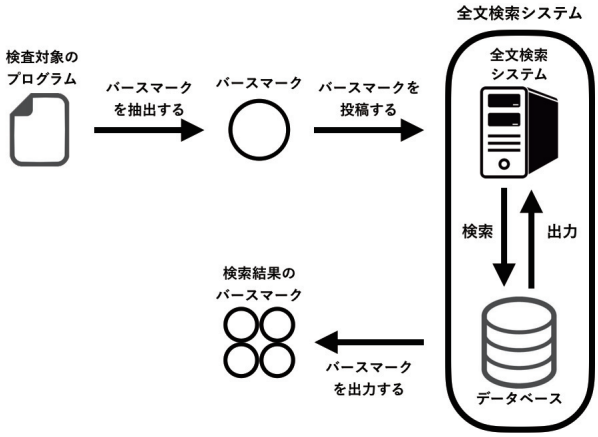


図 5 提案手法の検索ステップ
Fig. 5 The procedure of the search step.

書から事前に作成される．つまり，索引を作成するオーバヘッドがあるものの，与えられたキーワードに関連する文書を非常に高速に見つけ出せる．そして，保存されているデータ数が増え，1 つのサーバで扱えるデータ量が目安量を超えたとしても，今日提供されているほとんどの全文検索システムではクラスタ構成が可能である．つまり，文書数の爆発的な増加に対する対応策が用意されているといえる．

そのため，あらかじめ大量のプログラムからバースマークを抽出し，全文検索システムのデータベースに登録しておく．その後，検索のキーワードとして原告プログラムのバースマークを指定する．結果として，類似したバースマークが得られる．得られたバースマーク情報が絞り込み段階の結果となり，続く比較段階への入力となる．

なお，この絞り込み段階は，登録と検索の 2 つのステップからなる．最初のステップである登録ステップの内容を**図 4** に示す．このステップは，従来の収集段階，抽出段階を，絞り込み段階向けに実施するものである．このステップでは，まず，世の中の多くのプログラムを集め，そこから特定のバースマークを抽出する．そのバースマークを全文検索システムのデータベースに登録することで，このステップが完了する．

次の検索ステップの処理を**図 5** に示す．典型的なユースケースとして，原告プログラム p を提案システムに投稿することを考える．このとき，提案システムは，p から指定されたバースマーク x を抽出し，$\mathcal{B}_x(p)$ を得る．次に，システムは，類似バースマークを検索するために，$\mathcal{B}_x(p)$ を全文検索システムに投稿する．最終的に全文検索システムの結果として出力されたバースマークとそのスコアを提案システムの結果とする．そして，全文検索システムのスコアを提案手法の類似度として扱う．

なお，それぞれのバースマークに定義された比較手法はここでは適用しない．なぜなら，全文検索システムに個々のバースマークに適した比較手法を実装することは容易ではない．バースマークの定義により比較方法が異なるためである．そこで本稿では，スコア算出アルゴリズムとして BM25 を採用する．なお，BM25 は文書におけるクエリの単語の出現頻度に基づく，スコア算出アルゴリズムである [20]．

なお，検索ステップには，バースマークの検査段階のような複雑な判断は導入せず，ε_n のみで判断する．なぜなら，複雑な判断は実行時間の低下につながるためである．また，絞り込み段階で仮に誤検出したとしても，続く比較段階，検査段階でより正確性の高い検査が行われるためである．一方で，検出から漏れたプログラムは今後，検査の俎上に登ることはないため，厳しく避ける必要がある．そのため ε_n はできる限り小さな値にすることになるであろう．

3.4 提案手法のスコア算出アルゴリズム BM25

BM25 はクエリの単語の出現頻度に基づいて，文書集合を順位付けするスコア算出アルゴリズムである．単語 $q_1, q_2, \ldots, q_n$ を含むクエリを Q，単一の文書を D とすると BM25 の定義は以下のようになる [20]．

$$\mathrm{score}(Q, D) = \sum_{i=1}^{n} \mathrm{idf}(q_i) \cdot \frac{f(q_i, D) \cdot (k_1 + 1)}{f(q_i, D) + k_1 \cdot (1 - b + b \cdot \frac{|D|}{avgdl})}$$

このとき，$f(q_i, D)$ が文書 D における単語の出現頻度，$|D|$ を文書 D の単語数，$avgdl$ を文書集合の平均単語数とする．k_1 および b は任意のパラメータであり，本稿では $k_1 = 1.2$，$b = 0.75$ とする[*7]．また単語 q_i の idf は以下の式で求められる．

$$\mathrm{idf}(q_i) = \log \frac{1 + (N - n(q_i) + 0.5)}{n(q_i) + 0.5}$$

このとき，N を全文書数，$n(q_i)$ を q_i を含む文書数とする．以上の定義を用いて，全文検索システムは全文書に対してスコアを算出する．

3.5 提案手法の定式化

本節では，提案手法を定式化する．まず，提案手法と従来のバースマーク手法のシナリオを定義する．

3.5.1 対象集合

提案手法で扱う被告プログラムの集合を Q =

[*7] この値は，本稿で用いた全文検索システム Apache Solr のデフォルト値である [21]．

$\{q_1, q_2, \ldots, q_n\}$ とする．また，提案システムが対応するバースマークの種類を $X = \{x_1, x_2, \ldots, x_m\}$ とする．バースマークの抽出手法 x_i によって Q から取り出されたバースマークを $S_{x_i} = \{\mathcal{B}_{x_i}(q_1), \mathcal{B}_{x_i}(q_2), \ldots, \mathcal{B}_{x_i}(q_n)\}$ とする．抽出した各種バースマーク $S_{x_1}, S_{x_2}, \ldots, S_{x_m}$ を全文検索システムのデータベースに ${}^{\forall}S = \{S_{x_1}, S_{x_2}, \ldots, S_{x_m}\}$ として登録する．提案手法のシナリオでは，ユーザは全文検索システムに $\mathcal{B}_f(p)$（$f \in X$）をリクエストとして投稿する．このとき，全文検索システムはリクエストとして投げられたバースマーク $\mathcal{B}_f(p)$ と S_f にある各バースマークを比較する．

3.5.2 絞り込み段階の定式化

全文検索システムの結果の集合を $R_x(\mathcal{B}_x(p)) = \{r_{x,1}, r_{x,2}, \ldots, r_{x,n}\}$ とする．$r_{x,j} = \{c_j, \mathcal{B}_x(q_j), s_j\}$ は，プログラム名 c_j，q_j から x によって抽出したバースマーク $\mathcal{B}_x(q_j)$，そして，全文検索システムによって計算された $\mathcal{B}_x(p)$ と $\mathcal{B}_x(q_j)$ の間の類似度 $s_j = \mathrm{comp}(\mathcal{B}_x(p), \mathcal{B}_x(q_j))$ を含んでいる（$1 \le j \le n$）．なお，ここでは，プログラム名 c_j はクラス名やファイル名など，各プログラムを区別するための名前とし，プログラム q から $\mathrm{name}(q)$ で得られるとする．最後に，$s_j < \varepsilon_n$ となる $r_{x,j}$ を R_x から削除し，結果となる $\mathcal{R}_x = \{r_{x,i} | r_{x,i} \in R_x \wedge s_i \ge \varepsilon_n\}$ を得る．なお，ε_n はユーザにより与えられるものとする．

3.5.3 絞り込み段階後の比較段階の定式化

従来のバースマーク手法の結果の集合を，$V_x(\mathcal{B}_x(p)) = \{v_{x,1}, v_{x,2}, \ldots, v_{x,n}\}$ とする．$v_{x,j} = \{c_j, \mathcal{B}_x(q_j), t_j\}$ は，プログラム名 c_j（$c_j = \mathrm{name}(q_j)$），q_j から x によって抽出したバースマーク（$\mathcal{B}_x(q_j)$），そして，従来手法で定義されたバースマーク計算法により得られた $\mathcal{B}_x(p)$ と $\mathcal{B}_x(q_j)$ の類似度 t_j を含んでいる（$t_j = \mathrm{sim}_x(\mathcal{B}_x(p), \mathcal{B}_x(q_j))$，$1 \le j \le n$）．最後に，$t_j < \varepsilon$ となる $v_{x,j}$ を V_x から削除し，結果となる $\mathcal{V}_x = \{v_i | v_i \in V_x \wedge t_i \ge \varepsilon\}$ を得る．ε も ε_n と同じくユーザにより与えられるものとする．

4. 実装

4.1 バースマーク絞り込みシステム *Mituba*

我々は，3 章で述べた手法に従って Java 向けのバースマークを利用した絞り込み処理システム，*Mituba* を試作した．*Mituba* は Web システムとして動作し，多数の被告プログラム Q があらかじめ登録されているものとする．そして，*Mituba* は *MitubaWeb*，*MitubaSE*，*MitubaCrawler* の 3 つのコンポーネントから構成されている．*MitubaWeb* は，クライアントからのリクエスト処理を行い，*MitubaSE* は，実際に絞り込み処理を実施するコンポーネントである．一方，*MitubaCrawler* は，定期的にデータを収集する役割を担っており，被告プログラムを日々収集していく．

また，*Mituba* が行う処理は，検索と登録の 2 つに大別できる．以下にそれぞれの処理を説明する．

4.1.1 *Mituba* の検索処理

Mituba の検索プロセスは次のとおりである．

(S1) ユーザは，原告プログラム p とバースマーク抽出法 f をブラウザに入力する．
(S2) ブラウザは受け取った p と f をリクエストとして *MitubaWeb* に送信する．
(S3) *MitubaWeb* は受け取った p と f からバースマーク $\mathcal{B}_f(p)$ を抽出する．
(S4) *MitubaWeb* は *MitubaSE* に対して，f と $\mathcal{B}_f(p)$ を送信する．
(S5) *MitubaSE* は受け取った $\mathcal{B}_f(p)$ を内包する全文検索システムに送信し，結果を取得する．
(S6) *MitubaSE* は取得した結果を，*MitubaWeb* に返信する．
(S7) *MitubaWeb* は，受け取った検索結果をブラウザに返信し，ユーザに検索結果を示す．なお，検索結果は，被告プログラムの名前，原告プログラムのバースマーク $\mathcal{B}_f(p)$ との類似度で示される．

Mituba の検索プロセスを図 6 のステップ (S1) から (S7) に示す．このように，ユーザは原告プログラム p と抽出法 f をリクエストすることで，原告プログラムと *Mituba* に登

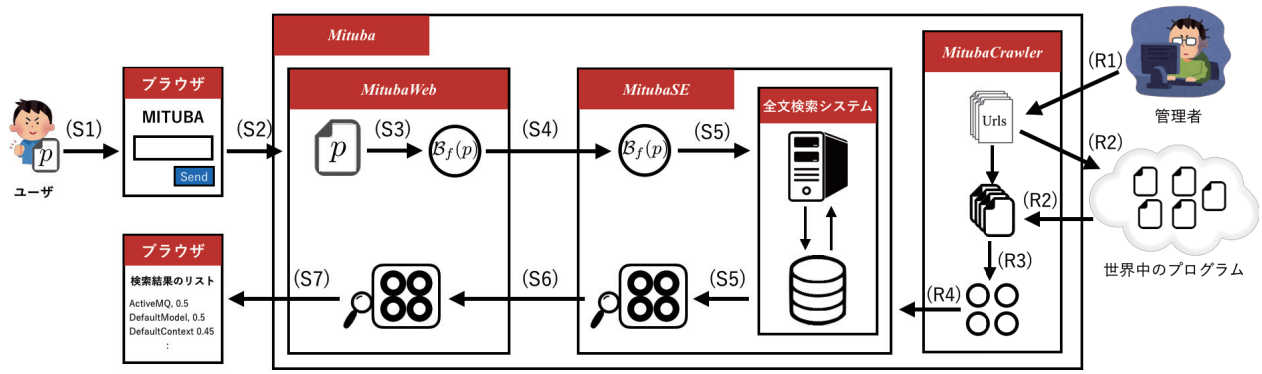


図 6 *Mituba* システム構成図
Fig. 6 The system overview of *Mituba*.

```
public class HelloWorld{
  public static void main(String[] args){
    System.out.println("Hello World");
  }
}
```

図 7 サンプルプログラム（HelloWorld.java）
Fig. 7 The sample program (HelloWorld.java).

録されているすべての被告プログラム Q とを比較できる．

なお，*MitubaSE* が内包する全文検索システムには，Apache Solr [*8] を利用した．Apache Solr はオープンソースの全文検索システムであり，登録したテキストファイルから検索キーに関連するファイルを検索できる [21]．

4.1.2 *Mituba* の登録プロセス

Mituba による検索をより効率化するためには，より多くのプログラムをあらかじめ登録しておくことである．しかし，プログラムは，世界中で日々リリースされ続けており，*Mituba* に登録されている情報も時間の経過とともに古い情報になってしまう．そのため，定期的に新たなプログラムの情報を全文検索システムに登録する必要がある．その役割を担うのが，*MitubaCrawler* である．*MitubaCrawler* による登録プロセスは次のとおりである．

(R1) 管理者は *MitubaCrawler* に対して，複数のプログラムの URL を送信する．
(R2) *MitubaCrawler* は与えられた URL に従って，インターネットにアクセスを行い，プログラムを取得する．
(R3) *MitubaCrawler* は取得したプログラムからバースマークを抽出する．
(R4) *MitubaCrawler* は抽出したバースマークを *MitubaSE* に送信し，登録処理を行う．

Mituba による登録プロセスを図 6 のステップ (R1) から (R4) に示す．管理者が与える情報は，たとえば，Maven Repository などのソフトウェアリポジトリや，GitHub などである．こうすることで，入力の更新の手間を大幅に減らしつつ，最新のプログラムを取り込むことが可能となる．もちろん，*MitubaCrawler* を用いて，盗用と疑わしいソフトウェアを手動で登録することも可能とする．

4.2 事前登録されたバースマーク情報

Apache Solr が扱うデータは基本的には文字列データであるため，バースマーク情報を文字列データとして扱う．バースマーク情報の構造は，順列，集合，ベクトル，グラフなど様々であるが，単純な構造として表現できる．

図 7 にサンプルプログラムとして `HelloWorld.java` を示す．このプログラムから UC バースマーク [1]，2-gram バースマーク [3] を抽出して得られたバースマーク情報を**表 1** に示す．なお，UC バースマークはあるクラスが利用

[*8] http://lucene.apache.org/solr/

表 1 図 7 から抽出したバースマーク
Table 1 The extracted birthmark from Fig. 7.

f	$\mathcal{B}_f(p)$
UC	`java.io.PrintStream`, `java.lang.Object`, `java.lang.String`, `java.lang.System`
2-gram	25 183,183 177,177 178,178 18,18 182,182 177

しているクラスのうち，標準 API に含まれるものの集合であり，k-gram バースマークはメソッドに含まれる命令列の k-gram の集合である．たとえば，UC（Used Classes）バースマークは，クラス名の集合であり，クラス名は文字列で表現される．また，2-gram バースマークは，命令列の 2-gram である．Java 仮想マシンの各命令は `iload` などの文字列として表現されるが，その命令には数値（opcode）が割り当てられている [22]．たとえば，25 は `aload`，183 は `invokespecial` という命令である．このように，単純なリストであり，要素を文字列で表現可能である．同様に，その他のバースマーク情報も文字列として扱える．

そこで，プログラム名とバースマーク抽出法をファイル名として扱い，ファイルの内容を当該クラスから指定された抽出法により抽出したバースマーク情報とした仮想のファイルを作成する．その仮想ファイルを Apache Solr にバースマーク情報として登録する．Apache Solr へのリクエストであるバースマーク情報も同様に文字列として扱う．こうすることで，Apache Solr の仕組みでバースマークの検索を可能とする．

5. 評価実験

5.1 実験準備

5.1.1 評価項目

本稿では，提案手法を評価するために大きく 3 つの観点から評価を行う．

(1) 絞り込みの効果に関する評価
  (1a) 絞り込み処理の所要時間評価
  (1b) 絞り込み率の評価
  (1c) 誤検出の評価
(2) 絞り込みの精度に関する評価
  (2a) 検出漏れの調査
  (2b) メトリクスによる提案手法の精度調査
(3) 保存性の評価

提案手法の目的は，バースマーク処理全体の時間，特に比較時間を削減することである．絞り込みの効果に関する評価では，単純に提案手法を実行した結果，比較段階に要する時間がどのように変化するかを計測する．加えて，絞り込み処理によって，対象がどの程度絞り込まれているかを評価する．そして，絞り込み後の対象にどのくらい盗用でないプログラムが含まれているか，誤検出の評価を行う．ただし，誤検出は少ない方が望ましいものの，3.1 節でも

述べたように，大きな問題ではない．続く比較段階で詳細に検査されるためである．

その一方で，検出漏れは大きな問題となる．そこで，続く実験では，絞り込み処理の精度を評価する．ここでは，まず，検出漏れがどの程度存在するかを評価する．さらに，精度，適合率，再現率，f 値のメトリクスを用いて，提案手法の精度を示す．

最後に，保存性の評価を行う．悪意のある人物が盗用するとき，盗用の事実を隠すため元のプログラムに最適化や難読化など，何らかの変換が行われると考えられる．そのような場合でも，提案手法は元のプログラムを検出できるかを評価する．なお，全文検索システムでの類似度算出アルゴリズムには，BM25 を採用した．

5.1.2 実験環境

実験の環境は次のとおりである．*Mituba* を Mac Book Pro（OS X El Capitan with 2.7 GHz Intel Core i5 CPU, 16 GB RAM）で動作させ，同 PC 上のブラウザからリクエストを送信した．また，比較のための既存のバースマーク手法のツールとして，pochi[*9]を使用する．使用するバースマークの種類 f は，Myles らによる k-gram バースマークと [3]，Tamada らによる UC バースマークとした [2]．なお，k は 2〜6 とした．これらのバースマークは非常に簡素であるが，3-gram では 240,109,765 回の比較に約 16 時間の時間を要する．そのようなバースマークが絞り込み処理でどのように変化し，いかに比較時間を削減できたかを調査する．

5.1.3 予備実験 プログラム数の増加にともなう比較時間の増加の割合

本項では，評価実験のための予備実験として従来手法，提案手法それぞれに対して，プログラム数の変化にともなって比較時間がどの程度増加していくかを調査する．ここで 10^x 個の被告プログラムを用意し，比較時間を調査した（$x = 2, \ldots, 6$）．比較は 1 個の原告プログラムと 10^x 個の被告プログラム間で行った．$\varepsilon_n = 0$ と設定したため，比較回数は従来手法，提案手法ともに 10^x 回である．そして比較に用いたバースマークは，5.1.2 項で示した k-gram と UC バースマークを用いた（$k = 2, \ldots, 6$）．

結果を**図 8** に示す．横軸は被告プログラム数，縦軸は各バースマークの比較時間（秒）の平均を対数で表している．2 つの折れ線はそれぞれ，従来手法，提案手法を各バースマークに適用したときの比較時間の平均である．図 8 より，従来手法，提案手法の比較時間はともに被告プログラム数が増えることで比較時間は増えるものの，提案手法の比較時間の増加は従来手法の比較時間の増加に比べてわずかであることが分かる．このことから提案手法の比較の方が，よりバースマーク手法のスケールアップに適している

[*9] https://github.com/tamada/pochi

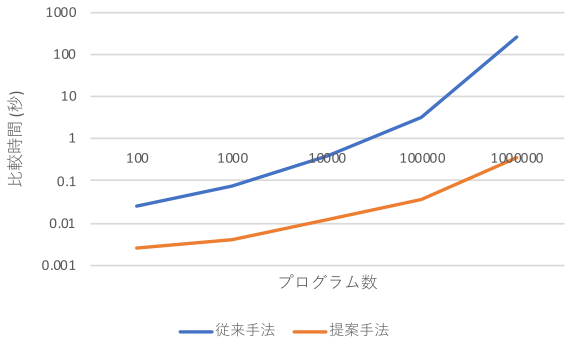


図 8 プログラム数の増加における比較時間

Fig. 8 The time of the previous and proposed birthmark process with increasing programs.

表 2 実験対象から除外するバースマークの要素数

Table 2 The element count of removal birthmarks.

$\mathcal{B}_f$	Minimum
2-gram	15
3-gram	24
4-gram	31
5-gram	37
6-gram	41
UC	2

といえる．

5.1.4 情報量の少ないバースマークの除去

バースマークは事前に情報を埋め込まないことから，偶然の一致を避ける手段が存在しない．たとえば，要素数の少ないバースマークの場合，偶然一致する可能性が高くなる．このような結果は，そもそもバースマークで判定不能であるため，実験のノイズとなる．そのため本実験では，短かすぎるバースマークを原告プログラム，被告プログラムから除去したうえで行うものとする．

除去の条件は，被告プログラムから取り出した各バースマークの長さの平均の 1/3 以下とした．その結果として，**表 2** に示すように，各バースマークの長さの最小値（Minimum）を決め，その長さ以下のバースマークは除去した．

5.1.5 評価のためのメトリクス

提案手法の評価のために，提案手法の精度，適合率，再現率を利用する．前提として，既存のバースマーク手法のツール（ここでは pochi）は，正しい結果を返すと仮定して，本システムの評価を行う．そして，評価尺度の定義の前に，TP（true positive），TN（true negative），FP（false positive），FN（false negative）を定式化する．

まず，TP を計算するために，プログラム名の集合 $\mathcal{N}_{\mathcal{R}_x} = \{c_i | c_i \in r_i \wedge r_i \in \mathcal{R}_x\}$ を得る．同様に，$\mathcal{N}_{\mathcal{V}_x}$ も $\mathcal{V}_x$ から取得する．なお，$\mathcal{R}_x$ は全文検索システムから得られた絞り込み結果の集合であり，$\mathcal{V}_x$ は従来手法で得られた盗用の疑いのあるプログラムの集合を表す（3.5.2 項，3.5.3 項参照）．また，$^{\forall}\mathcal{N}$ をすべてのプログラムのプロ

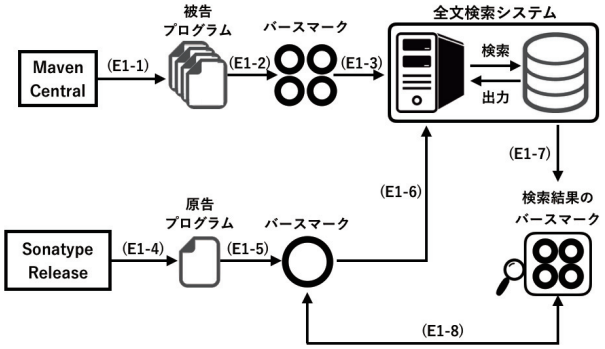


図 9 評価実験 1 の手順
Fig. 9 The procedure of the experiment 1.

グラム名とする（$^{\forall}\mathcal{N} = \{n_i | n_i \in q_i \wedge q_i \in Q\}$）．ただし，$n_i$ は q_i のプログラム名とする（$n_i = \mathrm{name}(q_i)$）．ここで，$\overline{\mathcal{N}_{\mathcal{R}_x}} = \{c_i | c_i \in \forall\mathcal{N} \wedge c_i \notin \mathcal{N}_{\mathcal{R}_x}\}$．同様に，$\overline{\mathcal{N}_{\mathcal{V}_x}} = \{c_i | c_i \in \forall\mathcal{N} \wedge c_i \notin \mathcal{N}_{\mathcal{V}_x}\}$ とする．最後に，TP，TN，FP，FN をそれぞれ次のように定義する．そして，それらを用いて，精度（M_a），適合率（M_p），再現率（M_r），f 値（M_f）を定義する．

$$\mathrm{TP} = |\mathcal{N}_{\mathcal{R}_x} \cap \mathcal{N}_{\mathcal{V}_x}| \qquad \mathrm{FP} = |\mathcal{N}_{\mathcal{R}_x} \cap \overline{\mathcal{N}_{\mathcal{V}_x}}|$$

$$\mathrm{TN} = |\overline{\mathcal{N}_{\mathcal{R}_x}} \cap \overline{\mathcal{N}_{\mathcal{V}_x}}| \qquad \mathrm{FN} = |\overline{\mathcal{N}_{\mathcal{R}_x}} \cap \mathcal{N}_{\mathcal{V}_x}|$$

$$M_a = \frac{\mathrm{TP}+\mathrm{TN}}{\mathrm{TP}+\mathrm{TN}+\mathrm{FP}+\mathrm{FN}} \qquad M_p = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FP}}$$

$$M_r = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}} \qquad M_f = \frac{2M_pM_r}{M_p+M_r}$$

5.2 絞り込みの効果に関する評価

5.2.1 実験手順

ここでは，大量の被告プログラムに対して絞り込み段階を適用し，実行時間がどの程度低減できるか，また，どの程度絞り込めるかを確認する．また，絞り込んだ結果にどの程度のノイズが含まれているかを確認する．

実験の手順は**図 9** と以下に示すとおりである．この手順をバースマークの種類ごとに行った．

(E1-1) 被告プログラムは Maven リポジトリ [23] に格納されている jar ファイルとする．
(E1-2) 各 jar ファイルのクラスからバースマークを抽出する．
(E1-3) 抽出されたバースマークを，全文検索システムのデータベースに登録する．
(E1-4) 検索のために Sonatype Releases Maven リポジトリ[*10]から 10 個の jar ファイルを原告プログラムとしてランダムに選ぶ．
(E1-5) 選ばれた原告プログラムに含まれる各クラスからバースマークを抽出する．
(E1-6) 得られた各バースマークをリクエストとして，全文検索システムに POST する．
(E1-7) 全文検索システムの検索結果を取得する．
(E1-8) 投稿されたバースマークと検索結果のバースマークを，従来手法で比較し，類似度を測定する．

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

表 3 各パターンで利用した jar ファイルのクラス数
Table 3 The search time of previous birthmarking and proposed birthmarking.

jar ファイル数	被告クラス数	原告クラス数
100	15,941	391
200	23,844	445
300	32,549	847
400	36,314	1,663
500	57,143	1,724
600	82,792	1,030
700	85,410	926
800	102,639	1,809
900	115,466	2,904
1,000	126,707	1,895

この実験では，文献 [23] で提供されている Maven リポジトリを被告プログラムとして検索を行う[*11]．Maven リポジトリには複数のソフトウェア（jar ファイル）が存在し，そのソフトウェア内にはプログラム（クラスファイル）が存在している．そして今回は，Maven リポジトリから 100, 200, ..., 1,000 個の 10 パターンの jar ファイルをランダムに取得し，それぞれ全文検索システムの別のデータベースに登録する．

また各パターンにおいて，原告プログラムも，Sonatype Releases Maven リポジトリから 10 個の jar ファイルをランダムに選択した．各パターンでの被告・原告のクラス数を**表 3** に示す．この原告ソフトウェアは被告ソフトウェアに含まれていない．そのため理論上は，同じプログラムが存在しない，つまり，バースマークにより弁別できる．もちろん，完全に弁別性を満たすことは困難であるため，類似するバースマークのペアが出てくる可能性がある．しかし，その数は非常に少なくなるはずである．

5.2.2 絞り込み処理の所要時間調査

この項では，提案手法の実施の有無による比較段階に要する時間を調査する．比較時間とは，図 9 でのステップ (E1-6)，(E1-7) に要する時間である．すなわち，原告プログラムをクエリとして投げて結果が得られ，さらにその結果を従来手法で比較するまでの時間を指す．そこで，従来手法で要する時間 t^f と，(E1-6)，(E1-7)，(E1-8) に要する時間 t^a を比較する．そして，どの程度高速化できたのかを速度比として，t^a/t^f で確認する．なおここでは，10 パターン分の結果の平均値を算出しそれを結果として示す．

実験の結果として，実際の比較に要した時間を**表 4** に，速度比を**図 10** に示す．表 4 の各列はバースマーク，そし

[*11] これは，2011 年 7 月 30 日時点の Maven Central リポジトリのスナップショットである．

表 4 従来手法と提案手法の時間比較（sec）
Table 4 The time of the previous and the proposed birthmark process (sec).

		2-gram	3-gram	4-gram	5-gram	6-gram	UC
	The conventional method	12,279.35	60,017.65	25,647.25	44,788.60	11,237.52	4,075.97
The proposed method	$\varepsilon_n = 0.0$	12,297.64	60,037.65	25,959.95	44,856.47	11,836.51	4,110.70
	$\varepsilon_n = 0.1$	1,499.05	6,550.66	2,929.62	2,696.14	1,989.74	206.61
	$\varepsilon_n = 0.2$	1,040.86	4,737.95	1,961.82	1,805.48	1,243.88	91.63
	$\varepsilon_n = 0.3$	849.79	3,421.08	1,462.06	1,282.19	857.16	78.31
	$\varepsilon_n = 0.4$	737.61	2,649.42	1,031.78	871.88	668.50	77.97
	$\varepsilon_n = 0.5$	566.05	2,104.72	926.27	645.46	642.15	73.75
	$\varepsilon_n = 0.6$	413.27	1,707.53	786.79	677.87	642.14	73.15
	$\varepsilon_n = 0.7$	374.65	1,565.16	759.55	663.57	595.92	68.14
	$\varepsilon_n = 0.8$	363.88	1,556.95	738.12	637.50	595.42	56.24
	$\varepsilon_n = 0.9$	359.66	1,538.63	672.35	565.25	495.32	54.11

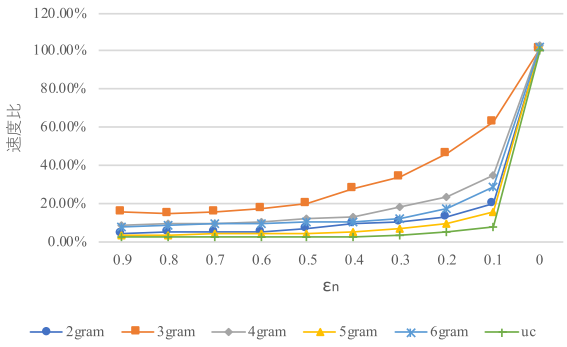


図 10 各閾値の速度比
Fig. 10 The speed rate of each threshold.

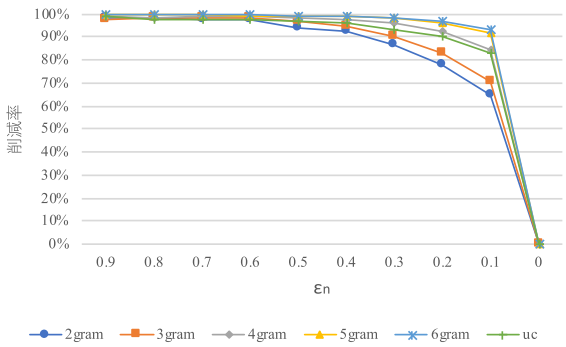


図 11 絞り込みによる対象プログラムの削減率
Fig. 11 The reduction rate of the target programs by the narrowing phase.

て 1 行目が従来手法の比較時間，それ以降の行が提案手法の各閾値での比較時間を示している．図 10 の横軸は提案手法における閾値（ε_n），縦軸は速度比を示す．図 10 から，$\varepsilon_n \geq 0.4$ での各バースマークの速度比は 30%以下と比較時間を大きく削減できていることが分かる．当然ながら，$\varepsilon_n = 0.0$ での各バースマークの速度比は 100%を超えている．従来手法での処理に加えて，提案手法の処理を実行しているためである．

表 4 の結果から従来手法は，多大な時間がかかっており，一番時間を要する 3-gram の従来手法の比較時間で約 16 時間（60,017.65 sec）を要している．一番短時間の UC であっても，1 時間程度の時間（4,110.70 sec）を要していることが分かる．ここで着目すべき点は，提案手法の $\varepsilon_n = 0.0$ での比較時間と従来手法の比較時間である．$\varepsilon_n = 0.0$ とは絞り込みを行わないことを表している．各バースマークにおいて，両時間はほぼ同じ時間である．これはつまり，(E1-6), (E1-7) に要する時間はごくわずかであり，所要時間の大半は (E1-8) に要する時間であることが分かる．つまり，絞り込みが効率的に行われることでよりバースマーク処理がより高速化できると期待できる．

5.2.3 絞り込み率の調査

次に絞り込み処理によって，対象プログラムがどれほど絞り込めたかを確認する．結果を**図 11** に示す．横軸は提案手法の閾値（ε_n），縦軸は対応する閾値における対象プログラムの削減率である．また，絞り込み後のクラス数を**図 12** に示す．横軸は図 11 と同じく提案手法の閾値（ε_n），縦軸は絞り込み後のクラス数を対数で表している．

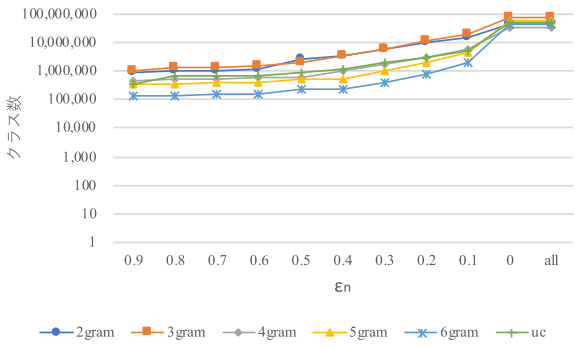


図 12 絞り込み処理後のクラス数
Fig. 12 The number of classes after the narrowing phase.

結果から，$\varepsilon_n \geq 0.4$ での削減率は，すべてのバースマークにおいて 90%以上と非常に高い．しかしながら，$\varepsilon_n \leq 0.1$ になると，削減率が急激に悪化する．

一方，図 12 を見ると，各バースマークのクラス数は $\varepsilon_n = 0.9$ で 100,000 と多い．しかし，図 11 のどのバース

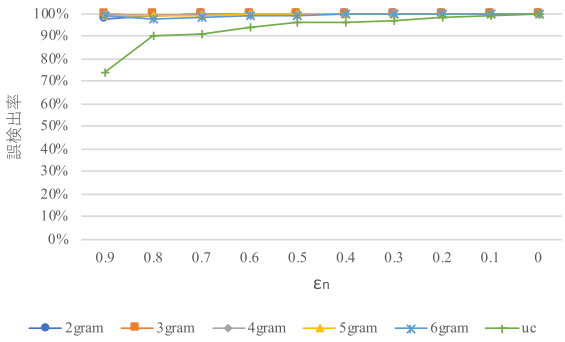


図 13 誤検出率
Fig. 13 The rate of the false positives.

マークを見ても，$\varepsilon_n \geq 0.4$ での削減率は 90%以上と高く，十分絞り込めていることが分かる．

5.2.4 誤検出の調査

次に，提案手法による検索結果の誤検出の割合を評価する．誤検出とは，盗用でないにもかかわらず，盗用であると検出されたプログラムのことである．本実験では，$\mathcal{B}(p)$ を投稿し，検索結果からプログラム名の集合 $N_{\mathcal{R}}$ を手に入れる．次に，p と n_i（$\in N_{\mathcal{R}}$）に対応するプログラムを従来手法により比較し，その結果から，次の誤検出率 C_{FP} を計算する．

$$C_{\mathrm{FP}} = \frac{\mathrm{FP}}{|N_{\mathcal{R}}|} = \frac{|N_{\mathcal{R}} \cap \overline{N_{\mathcal{V}}}|}{|N_{\mathcal{R}}|}$$

図 13 に実験の結果を示す．横軸は提案手法の閾値（ε_n），縦軸は誤検出率 C_{FP} を示す．また本実験も 5.2 節，5.2.2 項と同様，10 パターンの結果の平均値を示すものとする．図 13 から，$\varepsilon_n = 0.9$ での各バースマークの誤検出率は，70%以上と非常に高いことが分かる．しかし，図 11 から，$\varepsilon_n = 0.4$ でも 90%以上の大幅な絞り込みに成功しており，提案手法には一定の成果があり，この誤検出率であっても許容できるものと思われる．

5.3 絞り込みの精度に関する評価

5.3.1 実験手順

この実験では，従来のバースマーク手法と，提案手法の比較結果にどの程度違いがあるのかを確認する．前述のとおり，従来のバースマーク手法が正しい結果を返すと仮定している．実験の手順を**図 14** に示す．具体的な手順は次のとおりである．

(E2-1) 実験対象プログラムとして，文献 [23] で提供される Maven リポジトリから 50 個のプロダクトをランダムに選ぶ[*12]．ここで選ばれたプロダクトは被告，原告の両方として用いる．

(E2-2) すべてのプロダクトのクラスファイル（6,712 個）からバースマークを抽出する．

(E2-3) 抽出されたバースマークを，全文検索システムのデータベースに登録する．

(E2-4) 抽出したすべてのバースマークを従来手法のツールで比較する．比較した結果を $\varepsilon = 0.75$ でフィルタリングし，$\mathcal{V}$ を得る．

(E2-5) そして，$\mathcal{V}$ の各バースマークを全文検索システムへのリクエストとして送信し，$\mathcal{R}$ を得る．このときの結果は ε_n でフィルタリングされる．

(E2-6) 検索結果 $\mathcal{R}$ と $\mathcal{V}$ から，検出漏れ率 C_{FN} を計算する．

提案手法の閾値を $\varepsilon_n = 0.9 \sim 0.0$ の 0.1 刻みに設定し，バースマークごとに上記の手順を実行した．

なお，検出漏れ率 C_{FN} は次のとおりである．

$$C_{\mathrm{FN}} = \frac{\mathrm{FN}}{|N_{\mathcal{V}}|} = \frac{|\overline{N_{\mathcal{R}}} \cap N_{\mathcal{V}}|}{|N_{\mathcal{V}}|}$$

[*12] 従来手法の総当たりの比較が必要であるため，数を限定している．

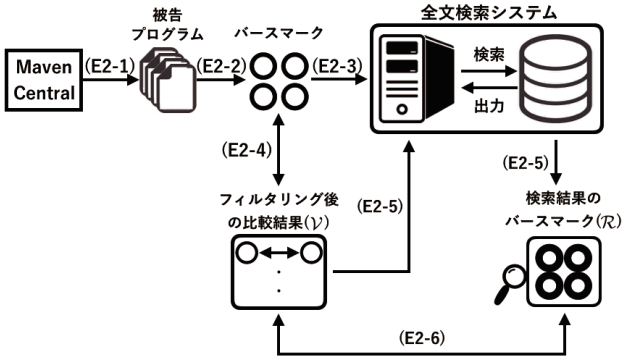


図 14 評価実験 2 の手順
Fig. 14 The procedure of the experiment 2.

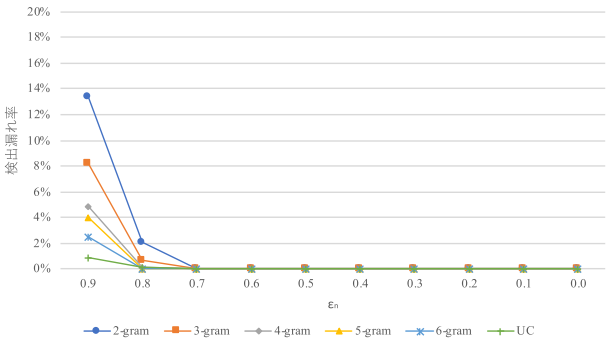


図 15 検出漏れ率
Fig. 15 The rate of the false negatives.

5.3.2 検出漏れの調査

図 15 に検出漏れ検査の結果を示す．横軸は提案手法の閾値（ε_n），縦軸は各閾値での検出漏れ率 C_{FN} を示す．図 15 を見ると，$\varepsilon_n = 0.9$ のときのすべてのバースマークの C_{FN} は 14%を下回っており，$\varepsilon_n = 0.6$ 以下は検出漏れがなかった．一方，$\varepsilon_n = 0.7$ の場合，C_{FN} は 0.3%を下回っており，低い検出漏れ率であることが分かる．ただし，低い検出漏れ率であっても 5.1.1 項で述べたように，検出漏れ自体が問題である．そのため，$\varepsilon_n = 0.7$ で検出漏れしたプログラムのペアを調査したところ，2-gram で 4 件，3-gram で 1 件，UC で 8 件であることが分かった．この詳

表 5 $\varepsilon_n = 0.7$ で検出漏れしたプログラムの詳細
Table 5 The pairs of the false nagatived programs in $\varepsilon_n = 0.7$.

$\mathcal{B}_f$	sim_f	comp	$\lvert\mathcal{B}_f(p)\rvert$	$\lvert\mathcal{B}_f(q)\rvert$	Class names		Jar files (Abbrev.)	
					p	q	p	q
2-gram	0.778	0.653	16	16	JstlFmtTLV	JstlSqlTLV	JONAS	JONAS
	0.778	0.653	16	16	JstlSqlTLV	JstlXmlTLV	JONAS	JONAS
	0.778	0.657	16	16	ProcessContext	BufferChangedEven	MMM	JONAS
	0.778	0.685	16	16	JstlFmtTLV	JstlXmlTLV	JONAS	JONAS
3-gram	0.758	0.654	29	29	Token$Attribute	Token$MapIdToValue	XSTR	XSTR
UC	0.778	0.685	8	8	SimpleCharStream	ASCII_CharStream	JONAS	JONAS
	0.778	0.661	8	8	ASCII_CharStream	SimpleCharStream	JONAS	JONAS
	0.778	0.697	8	8	AcegiAuthWrapper	ReceivePing	JUST	JUST
	0.778	0.697	8	8	SpringAuthWrapper	ReceivePing	JUST	JUST
	0.778	0.685	8	8	SimpleCharStream	SimpleCharStream	JONAS	JONAS
	0.778	0.661	8	8	ASCII_CharStream	ASCII_UCodeESC_CharStream	JONAS	JONAS
	0.778	0.685	8	8	SimpleCharStream	ASCII_UCodeESC_CharStream	JONAS	JONAS
	0.778	0.685	8	8	ASCII_UCodeESC_CharStream	SimpleCharStream	JONAS	JONAS

表 6 表 5 に現れる jar ファイル名
Table 6 The jar file names shown in Table 5.

Abbrev.	Full name of jar file
JONAS	jonas-web-container-jetty-6.1-5.2.0-M4.jar
MMM	mmm-util-core-2.0.1.jar
XSTR	org.apache.servicemix.bundles.xstream-1.2.2-1.0-m2.jar
JUST	just-ice-1.8.2.jar

表 7 sim_f，comp 間の相関係数
Table 7 The correlation coefficients between sim_f and comp.

$\mathcal{B}_f$	r	# of pairs
2-gram	0.78	10,118,257
3-gram	0.78	7,897,884
4-gram	0.83	6,182,618
5-gram	0.88	4,300,018
6-gram	0.92	2,714,488
UC	0.64	14,852,653

細を**表 5** に示す．表 5 の列は左から，バースマークの種類（$\mathcal{B}_f$），原告プログラム p と被告プログラム q から算出された従来手法の類似度（sim_f），提案手法の類似度（comp），各バースマークの長さ（$\lvert\mathcal{B}_f(p)\rvert$，$\lvert\mathcal{B}_f(q)\rvert$），$p$，$q$ のクラス名，p，q を取り出した元の jar ファイル名の省略形（Jar files (Abbrev.)）を示している．省略していない jar ファイル名は**表 6** に示している．5.1.4 項で述べたように，要素数が規定数以下のバースマークは情報量が少ないものとして，あらかじめ除去を行っている．すなわち，表 5 を見ると，2-gram，3-gram は 16，29 と表 2 に示す Minimum よりは大きいものの，全体の平均よりは短いバースマークとなっている．このため，2-gram，3-gram で検出漏れが起こったのは情報量が少なかったからである可能性がある．一方の UC は要素数の平均が 6 であるため，平均よりも大きなバースマークであるといえる．しかし，UC の sim_f は 0.778 であり，(E2-4) で $\varepsilon = 0.75$ と設定したことを考えるとそれほど高い類似度であるとはいえない．加えて，UC の comp も 0.661〜0.697 と $\varepsilon_n = 0.7$ を下回っているものの，大きく外れた値ではないことが分かる．

また，この実験で得られた sim_f と comp からピアソンの相関係数を求めた．結果を**表 7** に示す．各行にバースマークを示しており，該当するバースマークにおける相関係数を r 欄に，sim_f，comp を算出したペア数を# of pairs 欄に示している．なお，# of pairs 欄がバースマークごとに異なるのは，要素数の少ないバースマークを削除しているためである（5.1.4 項参照）．表 7 での r の値が一番低いのは UC であるが 0.64 と正の相関があり，UC 以外は 0.78 以上と強い正の相関があることが分かる（$p < 0.001$）．このことから，sim_f と comp は似通った値となるものの，$\varepsilon = 0.75$ であるため，$\varepsilon_n = 0.7$ あたりでは検出漏れの可能性があることが分かる．そのため，この検出漏れは $\varepsilon_n = 0.7$ がふさわしくなかったものと推測できる．以上のことから，より低い ε_n を適用することで検出漏れを防ぐことができると考えられる．

5.3.3 メトリクスによる提案手法の精度調査

ここでは 3.5 節で定義した指標を用いて，提案手法の精度を評価する．結果を**図 16**，**図 17**，**図 18**，**図 19** に示す．各図の横軸は提案手法における閾値（ε_n），縦軸は各精度項目の割合を表している．結果から，図 16 に示す精度（M_a）は $\varepsilon_n \geq 0.3$ では 90%以上であるものの，$\varepsilon_n < 0.3$ で急激に落ち込んでいることが分かる．また，再現率（M_r; 図 18）は，$\varepsilon_n = 0.9$ のとき，80%以上となっており，閾値を下げることにより 100%まで到達する．一方，適合率（M_p; 図 17）は，$\varepsilon_n = 0.9$ のとき，90%以上であるものの，閾値を下げることにより 0%まで到達する．このことから，ε_n が下がることにより検索結果に大量のノイズが含まれ

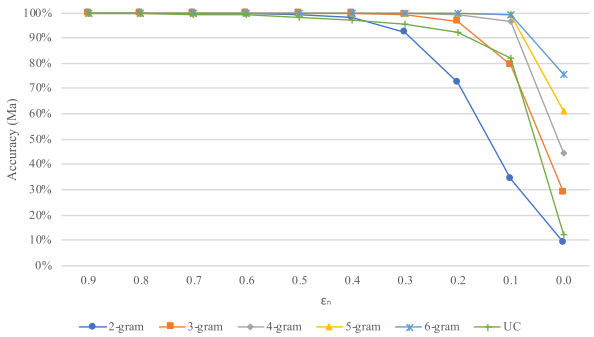


図 16 精度（M_a）
Fig. 16 Accuracy (M_a).

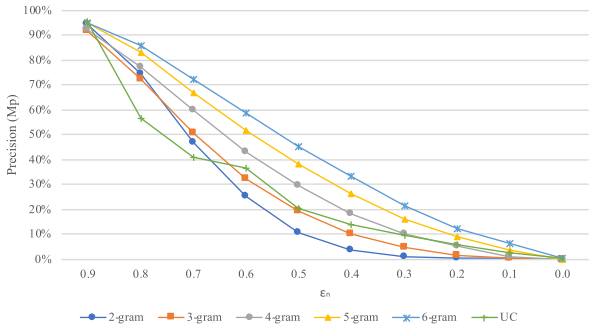


図 17 適合率（M_p）
Fig. 17 Precision (M_p).

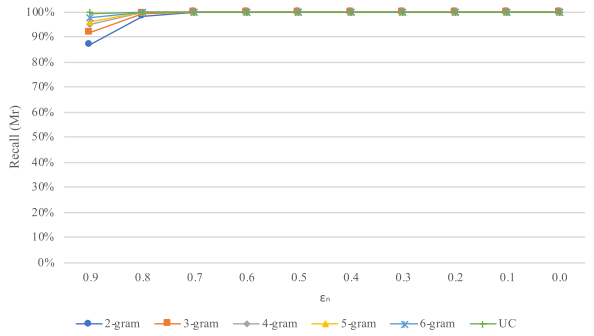


図 18 再現率（M_r）
Fig. 18 Recall (M_r).

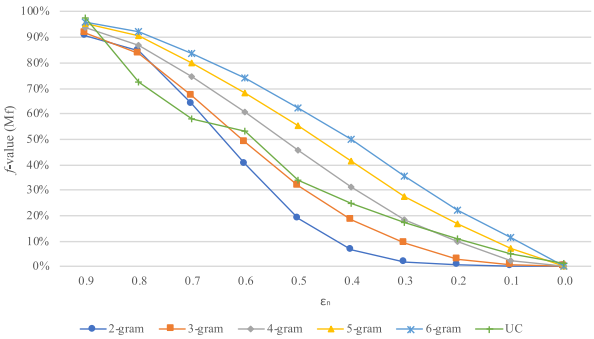


図 19 f 値（M_f）
Fig. 19 f-value (M_f).

るようになることが分かる．図 19 に示す f 値（M_f）も，M_p よりも多少高いものの，M_p と同様の推移をたどっていることが分かる．M_f は M_p と M_r の調和平均であり，ε_n

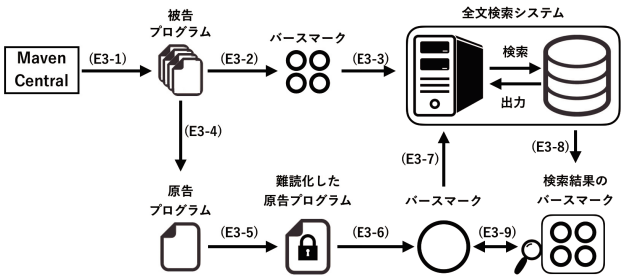


図 20 評価実験 3 の手順
Fig. 20 The procedure of the experiment 3.

の低下にともなう M_p の悪化のため，提案手法の性能も下がる．しかしながら，5.2.2 項から分かるようにプログラム数が増えたとしても，提案手法は従来手法の比較時間を削減できている．したがって検索結果のプログラム数が増えることは，速度的な観点から問題にはならないことが分かり，提案手法は良好な結果を示しているといえる．

5.4 保存性評価

5.4.1 実験手順

プログラムが盗用されるとき，盗用の事実を隠すため，何らかの変換が行われることが考えられる．そのような場合でも，提案手法が保存性を満たすかを評価する．

実験の手順は以下のとおりである．同手順を**図 20** にも示す．なお，適用するバースマーク手法ごとに以下の手順を実施する．

(E3-1) 被告プログラムとしては，Maven リポジトリ [23] から 50 個のプロダクトをランダムに選ぶ．クラスファイル数は 6,712 である．

(E3-2) 各 jar ファイルのクラスからバースマークを抽出する．

(E3-3) 抽出されたバースマークを，全文検索システムのデータベースに登録する．

(E3-4) 検索のために (E3-1) の被告プログラムから 1 つの jar ファイルを原告プログラムとしてランダムに選ぶ．この原告プログラムは，被告プログラムに含まれている．

(E3-5) 選ばれた原告プログラムに含まれる各クラスを難読化手法により，難読化する．

(E3-6) 難読化された各プログラムからバースマークを抽出する．

(E3-7) 難読化された原告プログラムのバースマークを，提案手法に入力として与える．

(E3-8) 検索結果を取得する．

(E3-9) 難読化前後のプログラムの提案手法における類似度を得る．

5.4.2 保存性能の評価メトリクス

保存性能を評価するためのメトリクスを次のように定義する．まず，プログラム t を被告プログラムの集合 Q か

表 8 各難読化手法の略称と概要
Table 8 Abbreviations and overviews of the obfuscation methods.

分類	略称 (難読化手法の名称) 手法の概要
Control flow	IOP (Insert Opaque Predicates [24]) すべての条件式に恒偽式を付加する.
	IR (Irreducibility [25]) 制御フローを単純化不可能なように変更する.
Data	DR (Duplicate Registers) 変数への代入を重複させる.
	MLI (Merge Local Integers) 2 つの int 型を 1 つの long 型に統合する.
Layout	DNR (Dynamic Name Resolution [26]) メソッド呼出を動的呼出に変更しメソッド名を暗号化する.

ら取り出す．次に，ある難読化手法により t を難読化して τ を得る．τ からバースマーク手法 x を用いて，$\mathcal{B}_x(\tau)$ を得る．得られた $\mathcal{B}_x(\tau)$ を提案手法に与え，検索結果として $\mathcal{R}_x(\tau)$ を得る．このとき，検索結果に t が含まれているとき，提案手法に保存性能があったとする．

$$\mathrm{preserve}(t,\tau)=\begin{cases}1 & \mathcal{R}_x(\tau)\text{ に }t\text{ が含まれていた場合}\\ 0 & \text{含まれていなかった場合}\end{cases}$$

ここで，あるプログラム集合 $T=\{t_1,\ldots,t_l\}$ が与えられたとする（$|T|\ll|Q|$）．T の各プログラムを難読化手法 n により難読化して得られたプログラムを $\mathcal{T}=\{\tau_1,\ldots,\tau_l\}$ とする．このとき，保存性能 $P_x(T)$ を $\dfrac{\sum_{i=1}^{l}\mathrm{preserve}(t_i,\tau_i)}{|T|}$ とする．なお，本実験においてプログラム集合は jar ファイル中のクラスファイルの集合を指す．

5.4.3 保存性の評価

ここでは，**表 8** に示す 5 つの難読化手法を利用して保存性を評価する．以降，難読化手法は表 8 の略称を用いる．難読化は大きくコントロールフロー難読化，データ難読化，レイアウト難読化に分類される [25], [27]．ここでは，コントロールフロー難読化から 2 種類（IOP，IR），データ難読化から 2 種類（DR，MLI），レイアウト難読化から 1 種類（DNR）を選択している．レイアウト難読化は名称やフィールドやメソッドの定義場所を変更するのみであり，一般的にバースマークに大きな影響を与えない．しかし，DNR はメソッドの利用部分を動的呼び出しに変更する難読化手法であり，命令列を大きく変更する．すなわち，バースマークを大きく変更する恐れがあるため採用している．このように，難読化手法の各分類から代表的な手法としてこれらを採用している．ただし，他の難読化手法でバースマークを大きく変更する手法については別途評価の必要がある．なお，これらの難読化手法は Sandmark [28] と DonQuixote [29] により提供されている．各難読化手法の具体例は付録 A.1 に示す．適宜参照されたい．また，原

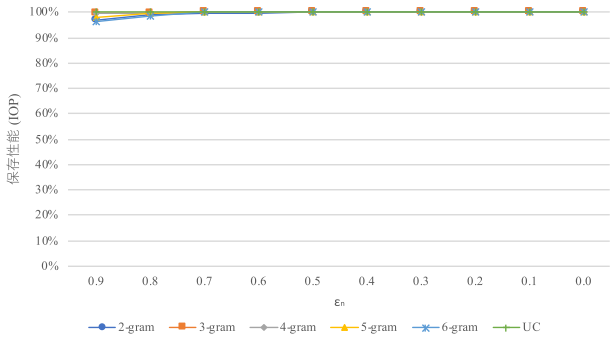


図 21 保存性評価（IOP）
Fig. 21 Preservation evaluation of IOP.

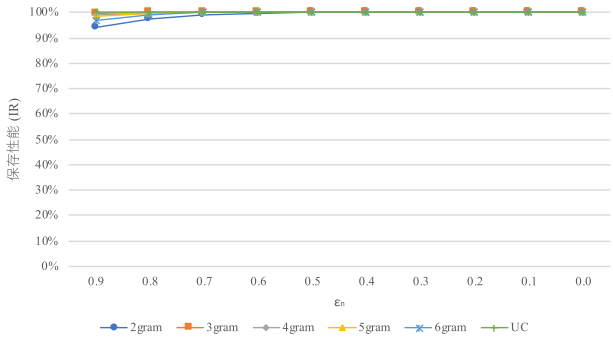


図 22 保存性評価（IR）
Fig. 22 Preservation evaluation of IR.

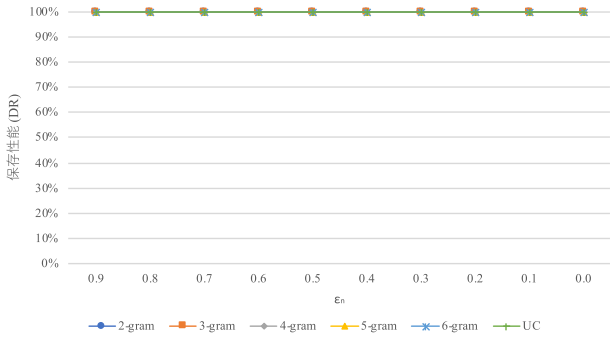


図 23 保存性評価（DR）
Fig. 23 Preservation evaluation of DR.

告プログラムには，ランダムに取り出した複数の jar ファイルを用いた．jar ファイル数は 20，クラスファイル数は 565 である．

保存性評価の結果を**図 21**，**図 22**，**図 23**，**図 24**，**図 25** に示す．横軸は提案手法の閾値（ε_n），縦軸は各閾値での保存性能を示している．結果より，DNR 以外の難読化手法に対する保存性能（図 21〜24）は，$\varepsilon_n=0.9$ でも 80%以上の割合を示している．すなわち，提案手法においても，難読化手法によってバースマークは大きく変化しないことが示されたといえる．一方，DNR に対する保存性能は $\varepsilon_n=0.9$ で，40%以下であり保存性を満たしていない（図 25）．しかし，ここでも閾値を $\varepsilon_n=0.3$ と下げることにより，80%以上の割合を示し高い保存性を示すようになる．

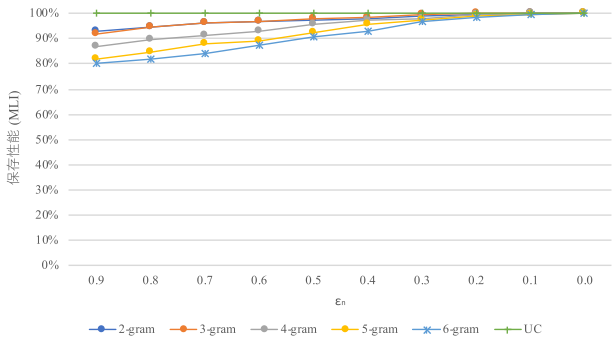


図 24 保存性評価（MLI）
Fig. 24 Preservation evaluation of MLI.

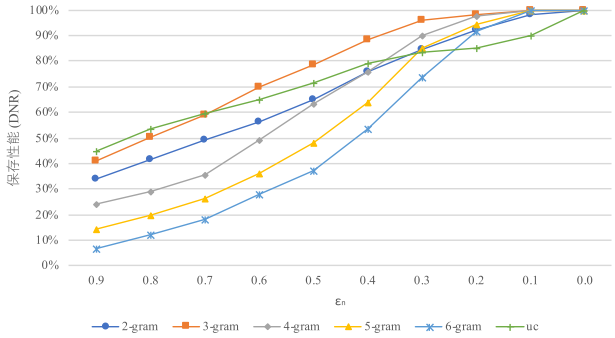


図 25 保存性評価（DNR）
Fig. 25 Preservation evaluation of DNR.

6. 議論

この章では，前述した実験結果をもとに全文検索システムを用いた提案手法の最適な閾値（ε_n）について議論する．最適な ε_n の決定には次の 6 項目を考慮する必要がある．各項目に対応する以下の実験をこれまでに行っている．以下には，実験結果となる図も併記している．

(1) 検索時間（図 10）
(2) クラス数の削減率（図 11）
(3) 誤検出（図 13）
(4) 検出漏れ（図 15）
(5) 検索精度
　(a) 検索精度（M_a）（図 16）
　(b) 適合率（M_p）（図 17）
　(c) 再現率（M_r）図 18）
　(d) f 値（M_f）（図 19）
(6) 保存性（図 21〜25）

これらの項目に関する評価の結果を，**図 26** に示す．横軸は，提案手法の閾値（ε_n），縦軸は各実験の結果の数値を表す．また，各折れ線は，各実験での結果の平均である．ただし，保存性評価では，DNR のみ他の結果と大きくかけ離れているため，別の折れ線としている．なお，速度比（図 10）と検出漏れ（図 15），誤検出（図 13）は結果となる数値が低い方が良いことを示している．そのため，結果を 1 から減算することで，他の評価結果と同じように，数値が高い方が良い結果を示すようにしている．このことを表すため，図 26 の凡例の後ろに印（*）を付けている．

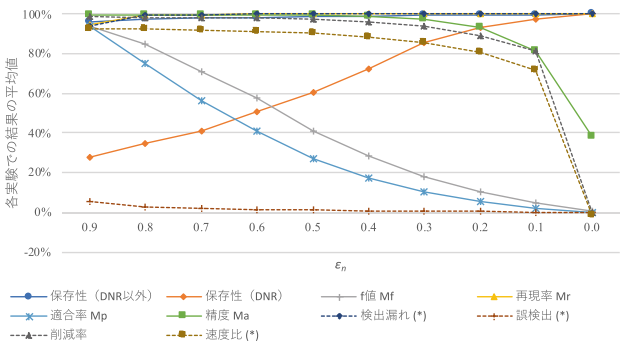


図 26 各実験の結果をもとにした最適な閾値（ε_n）
Fig. 26 The suitable threshold (ε_n) from each experiment.

結果を見ると，誤検出率が 0%のあたりにあるものの，削減率が 90%以上であり大きな問題にはならない（5.2.4 項参照）．また，保存性（DNR）と適合率（M_p）が $\varepsilon_n = 0.65$ あたりで交わっている．そして，誤検出以外の結果は，すべて交点より上であるため，最適な閾値は $\varepsilon_n = 0.6$ あたりであろう．ただしこの場合，本稿の実験においては $\varepsilon = 0.75$ のとき，$\varepsilon_n = 0.65$ で検出漏れはなかったものの，従来手法での ε の値によっては検出漏れの可能性がある（5.3.2 項ならびに，表 5 参照）．そのため，実運用時には検出漏れの可能性があることを考慮に入れておく必要がある．

しかし，DNR はすべてのメソッド呼び出しを動的呼び出しに変更する手法であり，利用していることの検知は容易である．そのため，DNR が利用されていれば，バースマークを適用する前に何らかの対策を講じることも可能である．一方，たとえ誤検出が多くても，続く比較段階で検査できる．そのため，誤検出，適合率（M_p）はユーザの問題設定によっては無視できる，同様に，f 値を表す M_f は適合率と再現率の調和平均であり，適合率と同じ理由で無視できる．そうなった場合，最適な閾値は，精度 M_a，速度比，削減率が急激に落ち込む前の値である $\varepsilon_n = 0.2$ となろう．すなわち，ユーザの問題設定によっては，$\varepsilon_n = 0.2$ 程度であっても提案手法は有効であるといえる．

7. 関連研究

ソフトウェアの類似度を測定するためにはコードクローンがよく用いられている．コードクローンはソースコード中のまったく同一の，もしくはよく似たコード断片のことである [5]．一般的にコードクローンはソースコードを対象にしており，盗用の疑いのあるプログラムを見つけ出すという目的には最適な方法ではない．一般に盗用されたプログラムのソースコードは公開されないため，適用できないためである．ただし，逆コンパイラの利用や [30]，一部バイナリを対象としたコードクローン検出も存在する [31]．しかしこの場合であっても，盗用の事実を隠すために難読

化などが適用された場合の対策が講じられていないため，適しているとはいい難い．

Android マーケットにおいて，再パッケージされた App を見つける研究が行われている [32], [33]．再パッケージとは，公開されているパッケージをもとに，新たなアプリケーションを第三者が開発，公開することである．Android マーケットはいわば巨大なソフトウェアリポジトリである．これらの中から再パッケージされた App を見つけるために，Kim はこの目的に特化したバースマークを提案している [32]．この手法は従来のバースマーク手法と同じアプローチである．一方，Oprişa らは本稿と似たアプローチをとっている [33]．あらかじめ Android マーケットから対象の App を取得し，ローカルの Mongo DB [*13] に必要情報を格納しておく．その後，ローカルの Mongo DB に対して検索する方法である．Oprişa らの手法は Android マーケットにおける再パッケージされた App を見つける方法に特化しており，我々の手法はバースマーク一般を対象にした高速化であるという点に違いがある．

8. まとめ

本稿では，バースマークの処理手順に新たな絞り込み段階を導入し，ラフな比較により，対象となる被告プログラムを絞り込む手法を提案した．そして，被告プログラム数の爆発的な増加に対しても，分散処理などによる対応が可能な全文検索システムを用いて絞り込み段階を実行する．提案手法に従い，Apache Solr を用いたバースマーク絞り込みシステム *Mituba* を作成し，評価実験を行った．

評価実験では，6 つの実験を行った．絞り込みによる比較時間の削減率，絞り込み段階でどの程度被告プログラムを絞り込めたかの絞り込み率，誤検出，検出漏れの調査，精度，適合率，再現率，f 値による調査，そして，保存性の評価である．いずれの実験においても，提案手法はバースマーク処理全体の時間を削減するという目的に対して，良好な結果を示した．

最後に，各実験の結果から，最適な類似度について議論を行った．若干の検出漏れが許容できるのであれば，$\varepsilon_n = 0.6$ が最適であろうことが分かった．しかし，ユーザの問題設定によっては，$\varepsilon_n = 0.2$ であっても提案手法は有効であることを示した．

ただし本稿では，全文検索システムの類似度算出アルゴリズムは BM25 固定であり，他の手法については考慮していない．また，より複雑なバースマークに対しても評価が必要となる．それらに対しても，本稿の結果が適用できるかを確認することが今後必要となろう．

加えて，保存性評価として選択した 5 つの難読化手法以外にも提案手法の保存性を脅かす手法がある可能性がある．そのため，他の難読化手法や難読化手法の組合せによる影響の調査は今後の課題とする．また，実際にクラスタを構成した場合の提案手法の性能評価についても，今後の課題とする．

謝辞 本研究の一部は JSPS 科研費 17K00196, 17K00500, 17H00731 の助成を受けた．

[*13] https://www.mongodb.com

参考文献

[1] Tamada, H., Nakamura, M., Monden, A. and Matsumoto, K.: Design and Evaluation of Birthmarks for Detecting Theft of Java Programs, *Proc. IASTED International Conference on Software Engineering* (*IASTED SE 2004*), pp.569–575 (2004).

[2] Tamada, H., Nakamura, M., Monden, A. and Matsumoto, K.: Java Birthmarks – Detecting the Software Theft, *IEICE Trans. Information and System*, Vol.E88-D, No.9, pp.2148–2158 (2005).

[3] Myles, G. and Collberg, C.: K-gram based software birthmarks, *Proc. 20th Annual ACM Symp. Applied Computing*, pp.314–318 (2005).

[4] il Lim, H., Park, H., Choi, S. and Han, T.: Detecting Theft of Java Applications via a Static Birthmark Based on Weighted Stack Patterns, *IEICE Trans. Information and Systems*, Vol.E91-D, No.9, pp.2323–2332 (2008).

[5] Kamiya, T., Kusumoto, S. and Inoue, K.: CCFinder: A Multilinguistic Token-based Code Clone Detection System for Large Scale Source Code, *IEEE Trans. Softw. Eng.*, Vol.28, No.7, pp.654–670 (2002).

[6] 山本照明，玉田春昭，門田暁人：大量のプログラムを対象としたファジーハッシュを用いたバースマーク手法，2015 年暗号と情報セキュリティシンポジウム予稿集（SCIS 2015），pp.3B4–4 (2015).

[7] Tsuzaki, T., Yamamoto, T., Tamada, H. and Monden, A.: Scaling Up Software Birthmarks Using Fuzzy Hashing, *International Journal of Software Innovation* (*IJSI*), Vol.5, pp.89–102 (2017).

[8] Nakamura, J. and Tamada, H.: Fast Comparison of Software Birthmarks for Detecting the Theft with the Search Engine, *Proc. 4th International Conference on Applied Computing and Information Technology* (*ACIT 2016*), pp.152–157 (2016). (Las Vegas, NV, USA).

[9] 中村 潤，玉田春昭：検索エンジンを用いたソフトウェアバースマークによる検査対象の絞り込み手法，第 24 回ソフトウェア工学の基礎ワークショップ（FOSE2017），福井，pp.99–104 (2017).

[10] Nakamura, J. and Tamada, H.: Mituba: Scaling Up Software Theft Detection with the Search Engine, *Proc. 2018 International Conference on Software Engineering and Information Management*, pp.6–10, ACM (online), DOI: 10.1145/3178461.3178475 (2018).

[11] Tamada, H., Okamoto, K., Nakamura, M., Monden, A. and Matsumoto, K.: Dynamic Software Birthmarks to Detect the Theft of Windows Applications, *Proc. International Symposium on Future Software Technology 2004* (*ISFST 2004*) (2004).

[12] Myles, G. and Collberg, C.: Detecting Software Theft via Whole Program Path Birthmarks, *Proc. Information Security the 7th International Conference, ISC 2004*, pp.404–415 (2004).

[13] il Lim, H., Park, H., Choi, S. and Han, T.: A Static Java Birthmark Based on Control Flow Edges, *Proc. 33rd Annual IEEE International Computer Software and Ap-*

plications Conference (*COMPSAC 2009*), pp.413–420 (2009).
[14] Chan, P.P.F., Hui, L.C.K. and Yiu, S.: Heap Graph Based Software Theft Detection, *IEEE Trans. Information Forensics and Security*, Vol.8, No.1, pp.101–110 (2013).
[15] Jhi, Y.-C., Wang, X., Jia, X., Zhu, S., Liu, P. and Wu, D.: Value-based Program Characterization and Its Application to Software Plagiarism Detection, *Proc. 33rd International Conference on Software Engineering*, pp.756–765 (2011).
[16] 岡本圭司，玉田春昭，中村匡秀，門田暁人，松本健一：API 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).
[17] Tian, Z., Zheng, Q., Liu, T. and Fan, M.: DKISB: Dynamic Key Instruction Sequence Birthmark for Software Plagiarism Detection, *Proc. 2013 IEEE 10th International Conference on High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing* (*HPCC_EUC*), pp.619–627 (2013).
[18] Schuler, D., Dallmeier, V. and Lindig, C.: A Dynamic Birthmark for Java, *Proc. 22nd IEEE/ACM International Conference on Automated Software Engineering* (*ASE 2007*), pp.274–283 (2007).
[19] 山田浩之，末永　匡：検索エンジン自作入門～手を動かしながら見渡す検索の舞台裏，技術評論社 (2014).
[20] Jones, K.S., Walker, S. and Robertson, S.E.: A Probabilistic Model of Information Retrieval Development and Comparative Experiments, *Information Process & Management*, Vol.36, No.6, pp.779–808 (online), available from ⟨http://dx.doi.org/10.1016/S0306-4573(00)00015-7⟩ (2000).
[21] Grainger, T. and Potter, T.: *Solr in Action*, Manning Publications (2014).
[22] Lindholm, T., Yellin, F., Bracha, G. and Buckley, A.: *The Java^{TM} Virtual Machine Specification Java SE 8 Edition*, Addison-Wesley Professional (2014). available from ⟨https://docs.oracle.com/javase/specs/jvms/se8/html/index.html⟩.
[23] Raemaekers, S., Deursen, A. and Visser, J.: The Maven repository dataset of metrics, changes, and dependencies, *Proc. 2013 10th IEEE Working Conference on Mining Software Repositories* (*MSR 2013*), pp.221–224 (2013).
[24] Collberg, C., Thomborson, C. and Low, D.: Manufacturing Cheap, Resilient, and Stealthy Opaque Constructs, *Proc. 25th ACM SIGPLAN-SIGACT Symp. Principles of Programming Languages* (*POPL '98*), pp.184–196 (1998).
[25] Collberg, C., Thomborson, C. and Low, D.: A Taxonomy of Obfuscating Transformations, Technical Report 148, Dept. of Computer Science, University of Auckland (1997).
[26] Tamada, H., Nakamura, M., Monden, A. and Matsumoto, K.: Introducing Dynamic Name Resolution Mechanism for Obfuscating System-defined Names in Programs, *Proc. IASTED International Conference on Software Engineering*, pp.125–130 (2008).
[27] Hosseinzadeh, S., Rauti, S., Laurén, S., Mäkelä, J.-M., Holvitie, J., Hyrynsalmi, S. and Leppänen, V.: Diversification and obfuscation techniques for software security: A systematic literature review, *Journal of Information and Software Technology*, Vol.104, pp.72–93 (2018).
[28] Collberg, C., Myles, G. and Huntwork, A.: Sandmark —A Tool for Software Protection Research, *IEEE Security & Privacy*, Vol.99, pp.40–49 (2003).
[29] 玉田春昭，中村匡秀，門田暁人，松本健一：Java クラスファイル難読化ツール DonQuixote，ソフトウェア工学の基礎 XIII，日本ソフトウェア科学会 FOSE2006，pp.113–118 (2006).
[30] Ragkhitwetsagul, C. and Krinke, J.: Using Compilation/Decompilation to Enhance Clone Detection, *IEEE 11th International Workshop on Software Clones* (*IWSC 2017*), pp.8–14 (2017).
[31] Hu, Y., Zhang, Y., Li, J. and Gu, D.: Binary Code Clone Detection across Architectures and Compiling Configurations, *IEEE 27th International Conference on Program Coprehension* (*ICPC 2017*), pp.88–98 (2017).
[32] Kim, G.: On computing similarity of android executables using text mining, *Proc. Symposium on Applied Computing* (*SAC 2017*), pp.1761–1762 (2017).
[33] Oprişa, C., Gavriluţ, D. and Cabău, G.: A scalable approach for detecting plagiarized mobile applications, *Knowledge and Information Systems*, Vol.49, No.1, pp.143–169 (2016).
[34] Aho, A.V., Sethi, R. and Ullman, J.D.: *Compilers: Principles, Techniques and Tools*, Addison-Wesley (1986).
[35] 齋藤鐵男，鈴木　貢，渡邊　坦：非可約な制御フローグラフのための簡潔で高速な支配木と支配辺境の検出算法，情報処理学会論文誌：プログラミング，Vol.43, No.SIG 8 (PRO 15), pp.72–86 (2002).

付　　録

A.1　5.4.3 項で用いた難読化の適用例

ここでは 5.4.3 項の表 8 で示した難読化を簡易な説明とともに，プログラムへの適用例を示す．なお，分かりやすさのため Java のソースコードを示すが，実際にはクラスファイルを対象に難読化が実行される．そのため実際の難読化結果と多少異なる場合があることに留意されたい．

再帰呼び出しを利用した Fibonacci 数例の n 項目を求めるプログラムを対象に表 8 で示した難読化手法を適用する．難読化前のプログラムを図 **A·1** に示す．

A.1.1　IOP（Insert Opaque Predicates）

条件分岐に Opaque Predicate（値によらず恒真，恒偽となる条件式）を追加する．たとえば，図 **A·2** では，4 行目と 7 行目に `int` 型の乱数値を計算した式を条件に追加して

```
 1: public class Fibonacci {
 2:   public int fibonacci(int n) {
 3:     if(n <= 0)
 4:       throw new IllegalArgumentException();
 5:     if(n == 1 || n == 2)
 6:       return 1;
 7:     int n1 = fibonacci(n - 1);
 8:     int n2 = fibonacci(n - 2);
 9:     return n1 + n2;
10:   }
11: }
```

図 **A·1**　難読化前のサンプルプログラム（`Fibonacci.java`）
Fig. A·1　The sample program (`Fibonacci.java`).

```
 1: public class Fibonacci {
 2:   public int fibonacci(int n) {
 3:     int c1 = (int)(Math.random() * 100);
 4:     if(n <= 0 || (c1 * (c1 + 1)) % 2 != 0)
 5:       throw new IllegalArgumentException();
 6:     int c2 = (int)(Math.random() * 100);
 7:     if(n == 1 || n == 2 || c2 * c2 < 0)
 8:       return 1;
 9:     int n1 = fibonacci(n - 1);
10:     int n2 = fibonacci(n - 2);
11:     return n1 + n2;
12:   }
13: }
```

図 **A·2** 図 A·1 を IOP で難読化した例
Fig. A·2 Obfuscated program of Fig. A·1 by IOP.

```
 1: public class Fibonacci {
 2:   public int fibonacci(int n1) {
 3: LABEL1:
 4:     if(n1 <= 0)
 5:       throw new IllegalArgumentException();
 6:     if(n1 != 1) {
 7:       do {
 8:         if(n1 != 2) break LABEL3;
 9:         if(!((n1 * (n1 - 1)) % 2) == (0 ^ 1))
10:           break LABEL2;
11:       } while(false);
12:     }
13:     goto LABEL1;
14: LABEL2:
15:     return 1;
16: LABEL3:
17:     int n2 = fibonacci(n1 - 1);
18:     int n3 = fibonacci(n1 - 2);
19:     return n2 + n3;
20:   }
21: }
```

図 **A·3** 図 A·1 を IR で難読化した例
Fig. A·3 Obfuscated program of Fig. A·1 by IR.

いる．しかしこの条件は，$x(x+1)$ は偶数であることや x^2 はつねに正数である数学的な性質を利用しているため，恒偽となる条件式である．そのため，両 if 文は難読化前後で条件分岐の結果が変わることはない．

A.1.2 IR（Irreducibility）

この難読化手法は逆コンパイラをクラッシュさせるために，コントロールフローグラフ（CFG）を既約なグラフ（irreducibile graph）[34] に変形する難読化手法である．**図 A·3** に難読化後のプログラムを示す．既約な CFG とは，ループへの入口が複数あることを指す [35]．図 A·3 をフローチャートで表したのが**図 A·4** である．なお，図 A·3 の 9 行目の if 文は恒真であるため，図 A·4 では *opaque* と表現している．図 A·4 で点線で表現されているパスは到達しないことを表す．ただし，このパスがあることによりこの CFG が既約なグラフとなる．また，13 行目に goto 文

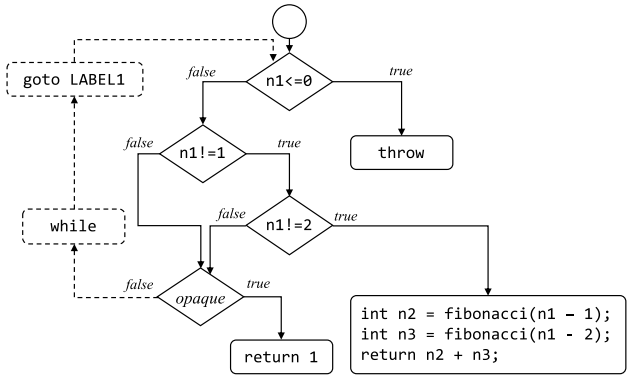


図 **A·4** 図 A·3 のフローチャート
Fig. A·4 The flowchart of Fibonacci shown in Fig. A·3.

```
 1: public class Fibonacci {
 2:   public int fibonacci(int n1) {
 3:     int v1 = n1;
 4:     if(v1 <= 0)
 5:       throw new IllegalArgumentException();
 6:     if(n1 == 1 || v1 == 2)
 7:       return 1;
 8:     int n2 = fibonacci(n1 - 1);
 9:     int v2 = n1;
10:     int n3 = fibonacci(v1 - 2);
12:     int v3 = n2;
13:     return n2 + v3;
14:   }
15: }
```

図 **A·5** 図 A·1 を DR で難読化した例
Fig. A·5 Obfuscated program of Fig. A·1 by DR.

があるが，Java 言語には goto 文は存在しない．しかし，Java 仮想マシンの命令には存在するため，Java バイトコードレベルでは成り立つプログラムである [22]．

A.1.3 DR（Duplicate Registers）

DR は変数への代入文を重複させる手法である．**図 A·5** の例では，変数 n1，n2，n3 を別の変数 v1，v2，v3 に代入する．その後，元の変数を参照していた箇所で，元の変数もしくは再代入後の変数のどちらかを参照するように変更する手法である．

A.1.4 MLI（Merge Local Integer）

この難読化手法は，2 つの int 型の値を 1 つの long 型にまとめる手法である．**図 A·6** では 3 行目で引数を long 型の変数 v1 の上位 4 バイトに格納し，n1 を参照していた箇所では v1 の上位 4 バイトを取り出して利用している．同様に再帰呼び出しの返り値も v1 の下位 4 バイト，別の long 型変数 v2 の上位 4 バイトに格納している．そして，必要な箇所では v1，v2 から値を取り出して利用している．

A.1.5 DNR（Dynamic Name Resolution）

DNR は通常のメソッドやクラス名を文字列として扱うこ

```
 1: public class Fibonacci {
 2:   public int fibonacci(int n1) {
 3:     long v1 = n1 << 32;
 4:     if((v1 >>> 32) <= 0)
 5:       throw new IllegalArgumentException();
 6:     if((v1 >>> 32) == 1 || (v1 >>> 32) == 2)
 7:       return 1;
 8:     v1 = v1 | fibonacci((v1 >>> 32) - 1);
 9:     long v2 = fibonacci((v1 >>> 32) - 2) << 32;
10:     return (v1 & 0xffffffff) + (v2 >>> 32);
11:   }
12: }
```

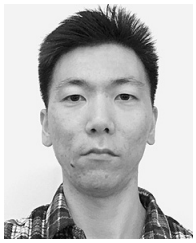
図 **A·6** 図 A·1 を MLI で難読化した例
Fig. A·6 Obfuscated program of Fig. A·1 by MLI.

```
 1: public class Fibonacci {
 2:   public int fibonacci(int n1) throws Throwable {
 3:     if(n1 <= 0)
 4:       throw (Throwable)Class.forName(decrypt(
 5:        "kbwb/mboh/JmmfhbmBshvnfouFydfqujpo"))
 6:          .getDeclaredConstructor()
 7:          .newInstance();
 8:     if(n1 == 1 || n1 == 2)
 9:       return 1;
10:     Object n2 =
11:       Class.forName(decrypt("Gjcpobddj"))
12:         .getMethod(decrypt("gjcpobddj"), int.class)
13:         .invoke(this, n1 - 1);
14:     Object n3 =
15:       Class.forName(decrypt("Gjcpobddj"))
16:         .getMethod(decrypt("gjcpobddj"), int.class)
17:         .invoke(this, n1 - 2);
18:     return ((Integer)n2) + ((Integer)n3);
19:   }
20:   private static String decrypt(String arg) {
21:     char[] c = arg.toCharArray();
22:     for (int i = 0; i < c.length; i++) {
23:       c[i] = (char)(c[i] - 1);
24:     }
25:     return new String(c);
26:   }
27: }
```

図 **A·7** 図 A·1 を DNR で難読化した例
Fig. A·7 Obfuscated program of Fig. A·1 by DNR.

とで, 名前を暗号化して扱う．そのために, 通常のメソッド呼び出しをリフレクション API を用いて動的呼び出しに変更している．**図 A·7** の例では, `IllegalArgumentException` のインスタンス生成や `fibonacci` メソッドの呼び出しが動的呼び出しに変換されている．そして，`fibonacci` メソッドの返り値は `Object` 型として受け取っており，`n2`，`n3` の加算時にキャストを行っている[*14]．また，クラス名，メソッド名などの名前はあらかじめ暗号化されており，実行時に `decrypt` メソッドにより復号される．図 A·7 の例では鍵 1 のカエサル暗号を用いている．

[*14] Auto unboxing により `int` 型に変換される．



中村 潤

2017 年京都産業大学コンピュータ理工学部卒業．2019 年同大学大学院先端情報学研究科博士前期課程修了．

玉田 春昭 （正会員）

2006 年奈良先端科学技術大学院大学情報科学研究科博士後期課程修了．同年同大学産学官連携研究員．2007 年同大学情報科学研究科特任助教．2008 年京都産業大学コンピュータ理工学部助教．2013 年同大学同学部准教授．2018 年同大学情報理工学部准教授．ソフトウェアセキュリティ, エンピリカルソフトウェア工学の研究の従事．IEEE, IEICE 各会員．