Vol. 46 No. 8 情報処理学会論文誌 Aug. 2005

ベイズ学習アルゴリズムを用いた未知のコンピュータウイルス検出手法

小池竜一[†] 中谷直司[†] 萩原由香里[†]
厚井裕司[†] 高倉弘喜[††] 吉田等明[†††]

コンピュータウイルスを検出および防御するためには，シグネチャと呼ばれる各ウイルス固有のパターンとのパターンマッチングを行う必要がある．しかし，未知ウイルスに対応するためのシグネチャ更新には時間を要するため，その間に被害が拡大する傾向にあった．そこで本論文では Paul Graham ベイズ学習アルゴリズムを用いることにより，確率ベースに基づいてウイルスを検出する手法を提案する．提案手法は過去に発生したウイルスの特徴点を学習することで，未来に発生する未知ウイルスを検出可能としている．特徴点としては実行ファイル中の表示可能文字列である strings を利用し，これらを学習すれば，70%以上の Netsky の亜種ウイルスと，Bagle の亜種ウイルスを検出できることが明らかになった．

The Unknown Viruses Detection Method Using Bayes Learning Algorithm

Ryuiti Koike,[†] Naoshi Nakaya,[†] Yukari Hagihara,[†]
Yuuji Koui,[†] Hiroki Takakura[††] and Hitoaki Yoshida[†††]

Unique patterns called signature are needed to detect the computer viruses by the pattern matching method. However, it takes time to generate the signatures because it is necessary for the signatures to be updated by human hands. Therefore, unknown computer viruses can infect many computers in the world easily until generating signatures. In this paper, we propose the method which can detect future unknown viruses by learning known viruses features in Graham Bayes. The features are "strings" which are printable sequences in binary files. Once learning features, the proposal method detects 70% Netsky variants and Bagle variants.

1. はじめに

コンピュータウイルス（以下，ウイルス）による被害は年々深刻なものとなり2004年度は過去最高を更新するのが確実であるとされている．近年では，各ニュースの媒体でコンピュータウイルスの被害がとりあげられるなど，ネットワーク管理者にとどまらず，一般の人々にもウイルスの危険性が知られるようになってきた．しかしながら，情報処理推進機構（IPA）が発表している国内のウイルス被害届出状況[1]の統計によると，2004年の上半期の時点で，2003年の年間届出件数の17,452件を大きく上回る21,952件の届出があった．今後の本格的な電子社会をむかえるにあたって，これらのウイルス被害を未然に防ぐことはきわめて重要である．

被害が増加している原因としては，あるウイルスをベースとした亜種が大量に出現し，シグネチャの更新が間にあわないことがあげられる．ウイルスを検出するためには，ウイルス対策ソフト（以下，アンチウイルス）を利用する方法が原則である．アンチウイルスがウイルスを検出する際には，ウイルスの特徴点を収めたシグネチャを用いる必要がある．このデータベースに存在しないウイルスは，アンチウイルスにとって未知のものであり，検出することはできない．したがって，アンチウイルスメーカはつねにウイルス解析およびシグネチャ生成を行い，ユーザはつねに最新のシグネチャに更新し続ける必要がある．しかし，近年は未知ウイルスの発生速度に，シグネチャの生成やユーザのシグネチャの更新が間に合わなくなりつつある．そ

† 岩手大学工学部
Faculty of Engineering, Iwate University
†† 京都大学・学術情報メディアセンターネットワーク研究部門
Academic Center for Computing and Media Studies, Kyoto University
††† 岩手大学総合情報処理センター
Super Computing and Information Sciences Center, Iwate University

こで，シグネチャに過度に依存せずに未知のウイルスでも検出できる手法の開発が重要となっている．

未知ウイルスを検出する手法には，静的なコード検査を行うもの[2),3)]や，ウイルスを実際に動作させネットワークレベルで検出を行うもの[4),5)]などがある．また，本論文でも取り上げる学習アルゴリズムを利用したものとしては，Naive ベイズを用いたもの[6),7)]などがある．しかし近年はウイルスの約 90%に実行可能圧縮と呼ばれるものが施されているため，それらの特徴点を圧縮されたまま学習することはウイルスの検出率に悪影響を与えることが分かった．また，未知ウイルス発生速度の増大に対応するためには，可能な限り高速な学習アルゴリズムを用いる必要がある．

そこで本論文では，解凍したウイルスの特徴点を学習アルゴリズムで学習し，未来に発生する未知のウイルスを検出する手法を提案する．提案手法は，入力を 2 つのカテゴリへ分類することに特化している Paul Graham ベイズ学習アルゴリズムを選択している．そのため現実的な処理時間で既知のウイルスの学習を行い，未来に発生する未知のウイルスの検出を可能としている．また，学習データは特別な技巧を必要とせずに，実行ファイルから得ることができる strings を使用しているため実装が容易であるという特徴もある．提案手法の最終的な目標は，過去のウイルスと類似性を持った未知ウイルスを検出可能にすることである．それによって，シグネチャ更新がなされるまでの間は未知ウイルスが検出できないという既存のアンチウイルスの弱点を補完することができる．

以下，2 章では近年のウイルスの現状と特徴，それらのウイルスの引き起こす問題を述べる．3 章では提案手法ついて詳しく述べる．4 章では実験に用いるデータについて述べ，5 章では圧縮ウイルスを学習した場合の悪影響について述べ，6 章では提案手法の評価を行う．

2. 現在のウイルスの特徴と課題

ここでは，近年のウイルスの特徴と，それらのウイルスが引き起こしている問題について述べる．

2.1 亜種ウイルス

近年のウイルスの特徴として，短期間で未知のウイルスが大量発生したことがあげられる．その理由としは，亜種の存在がある．未知のウイルスは細かく見ていくと，以下の 2 種類に分類が可能である．

- 以前に類似性のあるウイルスが存在しない新種
- 以前に類似性のあるウイルスが存在する亜種

亜種は，オリジナルのウイルスを一部改変して作成されたものであり，一からウイルスを作り始めるのに比べて作成に時間がかからない．亜種は多くの場合，送信されるメールの内容がほぼ同じものであったり，動作や PC に与える被害が同一のものであったりする．

シマンテック社の報告[8)]によると，2004 年に最も猛威を振るったウイルスである Netsky シリーズの最初にあたる Netsky.B が発生したのが 2 月 18 日，亜種 AB が出現したのが 4 月の 27 日であった．亜種の名前は “ウイルス名．アルファベット” の規則で命名され，アルファベット部分は発見された順番に A，B，… Z，AA，AB，… となるため，Netsky の例だけを見ても 2 カ月あまりで 30 種類近くの亜種が発生していることになる．また，2004 年上半期の統計[1)]を見ると，検出数の 80%以上を Netsky シリーズが占め，そのほかにも Bagle，My doom などの亜種が蔓延している状況が分かる．

アンチウイルスメーカは，次々に発生する亜種を検出可能にするために，シグネチャを生成し続ける必要ある．しかし，Netsky.Q の発生時にはシグネチャの配布が遅れてしまったため[9)]，結果として Netsky.Q は国内で 2 番目に流行したウイルスになってしまった．つまり，シグネチャの配布の遅れがそのままウイルス流行につながっており，シグネチャ更新に過度の依存をしないような，新しいウイルス検出技術が必要とされている．

2.2 実行可能圧縮

近年のウイルスは実行可能圧縮とよばれる形式に変換されている．実行可能圧縮に変換された実行ファイルは，圧縮や難読化がほどこされる．圧縮によりサイズが小さくなったウイルスは，ネットワークを介して拡散しやすくなる．また，難読化によりアセンブラレベルでの解析が困難な状態になる．本来は商用ソフトウェアが内部の動作を解析されないようにする目的で実行可能圧縮を使用するが，ウイルスはこの技術を悪用している例といえる．代表的な実行可能圧縮の形式としては ASPack，UPX があげられる．

これらの形式に変換されると，静的なコード検査を行うウイルス判定手法の適用が難しくなる．Microsoft Windows OS 上で動作する実行ファイルは PE 形式[10)]とよばれ，本来は**図 1** のようにセクションに分かれている．各セクションはその実行ファイル固有の情報が含まれており，たとえば，code セクションは実行コード，idata セクションはシステムコールの情報などが含まれている．実行可能圧縮への変換時には，これらすべてのセクションは圧縮，難読化され，それを元に戻すためのデコード部分が追加される．この変換

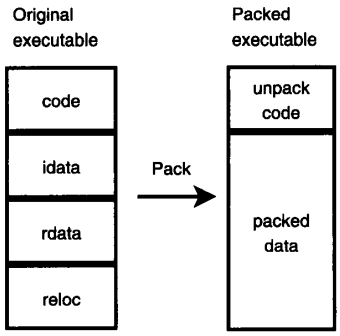


図 1 実行可能圧縮前後の状態
Fig. 1 The state before or after the executable compression.

表 1 ウイルスにかけられている実行可能圧縮の種類
Table 1 Kinds of pack types in viruses.

Pack Type	Breakdown(%)
ASPack	12.0
FSG	6.7
PE Pack	4.0
PEX	5.3
PEtite	4.0
UPX	48.0
tElock	8.0
Others	4.0
Unpacked	8.0

で圧縮されたセクションからは，元の実行ファイルの持っていた固有の情報が認識しにくくなる．この現象はファイルを ZIP 圧縮したときにも同様に見られるものである．

これらの実行可能圧縮形式とウイルスの実態を知るために以下の調査を行った．

(1) 2004 年の上半期に岩手大学総合情報処理センターに届いたウイルス付きメールを，FRISK 社の UNIX 用アンチウイルスである F-Prot [11] を用いてウイルスの種類ごとに分類する．

(2) 実行可能圧縮の形式判定ツールである PEiD [12] を用いてウイルスの圧縮形式を判定する．

F-Prot は UNIX 系 OS で動作するアンチウイルスとして広く用いられており，英国のコンピュータウイルス専門雑誌である Virus Bulletin が定期的に行っている検出テスト[13] において高い検出率が記録されているため，今回の分類に使用した．また，PEiD は実行ファイルの形式判定ツールのデファクトスタンダートであるため使用した．

上記のツールによって，F-Prot によって 75 種類のウイルスに分類され，PEiD によりそれらのウイルスは**表 1** のような実行可能圧縮の種類と内訳を持っていることが分かった．約 90%のウイルスは何らかの実行可能圧縮が施されており，ウイルス検出を行う際にはこれらの形式を考慮することが必須であるといえる．

3. 提案手法

3.1 概　　要

提案手法は，定期的に既知のウイルスを学習させておくことで，共通点を持った未来のウイルスを検出する手法である．提案手法と既存のアンチウイルスを組み合わせて用いることで，シグネチャ更新まで未知ウイルスが検出できない既存のアンチウイルスの欠点を補完することができる．

提案手法は学習アルゴリズムである Paul Graham ベイズ[14] に，解凍したウイルスから抽出できる表示可能な文字列（以下，strings）を学習させるものである．Paul Graham ベイズはウイルスにだけ頻繁に現れる strings を危険なものと認識し，入力されたファイルにも同様の strings が多く含まれる場合には，それをウイルスとして検出を行う．

Paul Graham ベイズは，本来スパム向けに開発されたものであるため，パラメータの値を一部変更することでウイルス向けへの改良を行う．またウイルスを解凍するのは，そのままの状態で学習を行うことが，ウイルス検出率に悪影響を与えるからである．悪影響の詳細については 5 章で明らかにする．

3.2 ウイルスとスパムの類似性

ウイルス検出に本来スパム向けの検出アルゴリズムである Paul Graham ベイズを選択した理由は，スパムとウイルスは「ある目的を達成するための単語，あるいは命令の集合体」であり，類似性があると考えたためである．

スパムは商品などの宣伝を目的として，望まない相手に対して強制的に送りつけられるメールである．そのため，どのスパムにも同様の単語が現れるという性質がある．オリジナルの Paul Graham ベイズではスパムに頻繁に現れる単語を学習することで，それらの単語を多く含んだメールをスパムとして検出している．このようなスパム検出フィルタは，ベイジアンフィルタと呼ばれ 90%以上の精度でスパムを検出している[15]．

一方，ウイルスも自分自身を拡散させ，多くの PC に感染していくという共通の目的を持っており，目的達成のために様々な行動をとる．ただし，亜種どうしにおいては似たような行動をとることが多い．これは，2.1 節で述べたように亜種は，オリジナルのウイルスに多少の修正を加えたものだからである．そこで，一定の手続きでウイルスから自身の動作に関する命令を取り出せば，亜種どうしにだけ頻繁に現れる命令が存

表 2 バイナリから抽出した strings
Table 2 strings extracted from binaries.

p$[8	RLENGTH	kernel32.DLL
X-ALMail-Status	VioGetCp	ole32.dll
jTgTTT.uuuuTg	Usage Error	rshift:F(0,19)
GetModuleFileNameA	gDgggggggg	checkbox
password	closesocket	cl_,

在すると考えられる．このように亜種どうしの命令が似ていると仮定すればスパムと同様に，既知の亜種を学習することで，その後に発生する未来の亜種を検出できる可能性がある．

3.3 学習データ：strings

提案手法では実行ファイルから自身の実行に関わるような命令を取り出し，それらを学習データとして用いる．ここでは学習データである strings について述べる．strings はバイナリに含まれる文字として表示可能なデータのことであり，UNIX 系 OS 上で動作する GNU strings コマンドによってバイナリから抽出することができる．文字として表示が可能なデータとは ASCII コードの 0x20 から 0x7E までであり，日本語などの 2 バイト文字は含まない．実行ファイルは PE 形式で構成されており各種情報が strings として格納されている．よって，GNU strings コマンドによって実行ファイル中から使用する共有ライブラリ名（以下，DLL）やシステムコール名（以下，API），動作時に参照する文字列，バイナリの一部などを抽出することが可能である．

PE 形式である実行ファイル中には使用する DLL と API が strings として記述されている．実行ファイルが Microsoft Windows OS 上で動作するためには，OS に対して様々な処理を依頼する必要があり，そのときに必要な API と DLL の情報が実行ファイル内部に含まれている．**表 2** は一般の実行ファイルから抽出した strings の一部である．表中には DLL や API が含まれることが確認できる．

実行ファイルには自らの動作時に参照する文字列が strings として記述されている．たとえば，メールを送信するようなウイルスは，メール本文や使用する単語のリストを内部に持っている．また改竄を行うファイルのパスや，追加を行うレジストリのキーなども strings として内部に持っている．

3.3.1 strings の長さ

GNU strings コマンドがバイナリから strings を抽出する際には，strings と見なす最短長を設定することができる．表 2 はデフォルトの 4 で抽出を行ったものであり，長さが 3 文字以下の strings は無効なものとして切り捨てられている．最短長を 1 のように小さく設定することで多くの情報を得ることができる．しかし，それに比例して意味を持たない余分な情報が混ざり，学習やカテゴリに分類するのに処理時間がかかるようになってしまう．逆に最短長を長くすることで，必要な情報を切り捨ててしまう恐れがある．最短長と処理時間およびウイルス検出率との関係は 6.2 節の実験で明らかにする．

3.4 学習アルゴリズム：Paul Graham ベイズ

Paul Graham ベイズはベイズ理論[16] による機械学習を行う分類器の一種であり，ある入力が与えられた場合にそれを適切なカテゴリへと分類する．本方式は入力を 2 つのカテゴリに分類することに特化しているため高速に動作する特徴がある．一方，テキスト分類などで広く用いられている Naive ベイズは入力を複数のカテゴリに分類するアルゴリズムであり，Paul Graham ベイズより多くの計算量やメモリを必要とする．ウイルスの流行を防ぐためには，いかに素早く対応をしていくのかが重要であるため，Paul Graham ベイズのような高速なアルゴリズムを用いる必要がある．

オリジナルの Paul Graham ベイズはベイズ定理をスパム向けに改良したもので，スパムと非スパムを学習し，新たな入力があった場合にそれをスパムカテゴリ，あるいは非スパムカテゴリに分類するものである．提案手法ではカテゴリをウイルスと非ウイルスとし，入力を実行ファイルの特徴点である strings とすることでスパム向けのフィルタをウイルスフィルタとして応用できることを示す．

提案手法は P_{base} の値以外は，オリジナルの Paul Graham ベイズの定義に従う．また，以下ではオリジナルの値をそのまま使う部分でもウイルス検出におけるその値の意味を交えながら説明を行っていく．

3.4.1 学習部分

処理は，大きく分けて学習部分とウイルス検出部分に分けることができる．学習部分の流れは以下のようになる．

(1) アンチウイルスを用いてウイルスと一般の実行ファイルを分類しておく．

(2) ウイルスと一般の実行ファイルから特徴点である strings を抽出する．

(3) 上記の strings からなるある特徴点 s のウイルス確率 $P(s)$ を求める．

このとき，ウイルスファイルの総数を n_{bad}，一般の実行ファイルの総数を n_{good}，ある特徴点 s がウイルスに現れた回数を b，一般の実行ファイルに現れた回

数を g とするとある特徴点のウイルス確率 $P(s)$ は以下のように表せる．

$$P(s) = \frac{b/n_{bad}}{2g/n_{good} + b/n_{bad}} \quad (1)$$

ただし，$P(s)$ の最大値を 0.99，最小値を 0.01 とする．これらの値は論文 14) において経験によって得られた最適な値とされている．また，学習時にほとんど表れないような特徴点は，ウイルスや一般の実行ファイルの一方の性質を強く反映しているというよりは，その特徴点を含む実行ファイル固有のものである．よって $2g + b < 5$ であるような出現回数の少ない特徴点は無視され，計算を行わない．

これにより，ウイルスにしか現れない特徴点には 0.99，一般の実行ファイルにしか現れない特徴には 0.01 という確率が割り当てられることになる．また両方に表れる特徴点には出現頻度によって，適切なウイルス確率が与えられる．ウイルスと一般の実行ファイルに含まれるすべての特徴点に対して式 (1) の計算を繰り返すことで，特徴点のウイルス確率が大量に収められたデータベースができることになる．以後これを提案シグネチャと呼ぶ．

3.4.2 検出部分

次にウイルス検出部分の流れについて述べる．以下は，学習後にある実行ファイルが入力されウイルス判定されるまでの流れである．

(1) 新規に入力された実行ファイルから特徴点である strings を抽出する．

(2) 実行ファイルから得られたすべての特徴点に対して，特徴点のウイルス確率を割り当てる．特徴のウイルス確率は先ほど生成した提案シグネチャから得る．提案シグネチャ中に対応する特徴点のウイルス確率が存在しない場合には，デフォルト値である 0.40 が割り当てられる．

提案シグネチャ内に存在しない特徴点は，学習時にウイルスや一般ファイルにほとんど出現しなかったものである．よって，入力にそのような特徴点が含まれた場合にも，どちらか一方の性質を反映しているものではなくその特徴点を含む実行ファイル固有のものであると考え，やや無害な 0.40 を割り当てる．

(3) 確率 P_{base} からの差が大きい順に特徴点を 15 個だけ選択して，後の計算に用いる．オリジナル Paul Graham ベイズでは P_{base} を 0.50 と設定するが，今回は 0.499 に設定する．入力されファイルがウイルスであれば 0.99 に近い値が多く選択されることになり，一般の実行ファイルであれば 0.01 に近い値が多く選択されることになる．なお，P_{baee} を 0.499 に設定した理由は 3.4.3 項で詳しく述べる．

(4) 選択された 15 個の特徴点の複合確率を求めることで，ファイル全体のウイルス確率が求まる，選択された特徴点のウイルス確率を P_i とすると，ファイル全体のウイルスで確率 P_v は式 (2) のように表せる．

$$P_v = \frac{\prod_{i=1}^{15} P_i}{\prod_{i=1}^{15} P_i + \prod_{i=1}^{15}(1 - P_i)} \quad (2)$$

特徴点を 15 個だけ選択するのは，入力された実行ファイルの性質を強く表している特徴点だけを選択するためである．提案手法では P_{base} という基準となる値を決め，その値から極端に離れている特徴点を，その実行ファイルの性質を強く表すものとしている．一部の特徴点だけを P_v の計算に用いていることで，ウイルス作者は提案手法による検出を回避するのが難しくなる．もし，すべての特徴点に注目する検出アルゴリズムを採用している場合には，ウイルス作者は無害な API などをウイルスに大量に追加することで P_v の値を容易に低下させられる．それにより，検出を回避されてしまう可能性がある．一方，提案手法のように 15 個の特徴を優先するアルゴリズムであれば，ウイルス作者が P_v を低下させようと無害な命令をウイルスに大量に追加しても，ウイルスらしい特徴が含まれている限りは検出が可能となる．

(5) P_v がしきい値である 0.90 を超えたものをウイルスとして検出する．

提案手法ではしきい値として 0.90 が設定されている．しかし，本手法にとってしきい値はそれほど重要なものではない．なぜなら，式 (2) の特性上，P_v は 0 か 1 に極端に近い値しかとらないからである．6.2 節で行った実験でも $0.10 < P_v < 0.90$ になるようなファイルは全体の 4.4%程度しか存在せず，ほとんどのファイルの P_v は 0 か 1 に近い部分にのみ分布していることが分かっている．よって，しきい値として 0.90 を設定すれば，ウイルスらしい高い P_v の値を持つファイルを適切に選択できる．

3.4.3 オリジナルからの変更点

P_{base} の値をオリジナルから変更したのはウイルスらしい特徴点が，より選択されるようにするためである．Paul Graham ベイズでは P_{base} との差が大きい順にウイルス確率を選択すると定義されている．しかし，P_{base} が 0.50 のときにウイルス確率 0.01 と 0.99 のどちらを選択するかは定義されていない．たとえば，入力されたファイルからウイルス確率 0.99 の特徴点が 10 個，0.01 の特徴点が 100 個，抽出できたとすると，最終的にどれが選択されるかはランダムである．

表 3 圧縮ウイルスを学習した際の一般の実行ファイルに対する誤検出率 (%)
Table 3 False detection rate between packed virus and nonvirus (%).

Signature generated from packed Virus	Nonvirus Input					
	FSG	PEX	PEtite	UPX	tElock	Nonvirus2
FSG	60.5	0.0	0.0	0.0	0.0	0.0
PEX	2.5	35.5	1.0	2.3	2.5	0.0
PEtite	26.0	0.0	50.0	11.4	10.0	0.0
UPX	3.5	26.5	0.5	64.0	0.5	0.0
tElock	0.0	0.0	0.0	0.0	37.5	0.0

しかし実際には出現回数が多いものが選ばれるため，この場合入力されたファイルは安全な実行ファイルとされる確率が高い．6.2 節の実験では実行ファイルから抽出された特徴点の 95%がウイルス確率 0.01 に割り当てられることが分かっており，仮に P_{base} が 0.50 の状態でカテゴリ分類を行った場合，多くの場合は，単純に数の多い 0.01 側が選択されることになる．そこで P_{base} を 0.499 と設定することで，0.99 側のウイルス確率が存在する場合には優先して選択されるようにした．

4. 準　　備

4.1 実験データ

ここでは実験に用いるデータについて述べる．以下にあげる各種ファイルから strings を取り出しておき，実験データとして用いた．Paul Graham ベイズは約 400 個のファイルの学習でも十分な性能を得られることが報告[17] されているため，今回用いる実験データも一般の実行ファイルが 200 個，ウイルス側が 200 個の学習が行えるような実験データを用意した

圧縮ウイルス　実行可能圧縮形式に変換が行われているウイルスのことである．アンチウイルスの F-Prot で検出したウイルスを PEiD によって，FSG, PEX, PEtite, UPX, tElock の圧縮形式ごとのグループに分けた．各グループのファイル数は 200 個である．

圧縮実行ファイル　実行可能圧縮形式に変換が行われている無害な一般の実行ファイルのことである．研究室で使用されている PC から無作為に実行ファイルを選び出し，FSG, PEX, PEtite, UPX, tElock の各圧縮形式に変換を行った．各グループのファイル数は 200 個である．

実行ファイル　実行可能圧縮形式に変換が行われていない無害な一般の実行ファイルのことである．研究室で使用されている PC から無作為に実行ファイル 400 個を選び出し，PEiD を用いて圧縮が行われていないことを確認した．そして，それらを 200 個の 2 グループに分けて Nonvirus1, Nonvirus2 とした．

亜種ウイルス　今回使用する亜種は岩手大学総合情報処理センターにおいて収集された Netsky の B, C, D, J, M, N, P, Q, S, T, W, X の計 12 種類と Bagle の J, K, N, O, Y, Z, AB, AE の計 8 種類である．これらを PC 上で一度実行して，動作中のメモリの内容をダンプすることで実行可能圧縮に変換されていない状態のウイルスを取り出した．Netsky.B を 200 個，Netsky.C を 200 個というように，亜種ごとに 200 個用意した．

5. 圧縮ウイルスの学習

5.1 圧縮形式判定器

ここでは圧縮ウイルスをそのまま学習すると，圧縮形式の特徴点を学習することになり，ウイルス検出器ではなく，圧縮判定器の性質を持ったものになってしまうことを示す．

Naive ベイズを用いて，圧縮形式ごとに圧縮ウイルスと実行ファイル Nonvirus1 を学習させ，入力として圧縮実行ファイルと実行ファイル Nonvirus2 を入力した場合の誤検出率を調べる．ここでいう誤検出率とは，一般のファイルを間違ってウイルスカテゴリに分類してしまった確率のことである．実験を行う際の strings の最短長 l は 4 に設定した．なお，これ以降の実験は Pentium III 1 GHz，メモリ 512 MB を搭載した PC で行う．

結果は**表 3** である．圧縮ウイルスと実行ファイルの特徴点を学習し，正しくウイルスらしさを認識していれば，大多数の圧縮実行ファイルは，ウイルスへのカテゴリ分類は行われないはずである．しかし，この例では同じ形式で圧縮された一般のファイルを誤検出する傾向があるのが分かる．これは 2.2 節で述べたように，元の特徴点が圧縮によって認識できなくなっているのが原因である．つまり圧縮後に残っているのは，あくまで圧縮形式ごとの特徴であり，元々がウイルスであったのか一般の実行ファイルであったのか区別が

つかなくなっている．たとえば，UPX 形式は一般の実行ファイルが圧縮されることで，64%がウイルスであると誤検出されてしまっている．逆に実行ファイル Nonvirus2 に対する誤検出はまったく見られない．これは圧縮ウイルスと圧縮実行ファイルは共通点を多く持っているが，圧縮されていないものとは明らかな違いがあることを示している．

このことからウイルス検出を目的とした場合に，圧縮ウイルスを学習させると検出率に悪影響を及ぼすといえる．圧縮ウイルスを学習させることで蓄積される特徴点は UPX などの圧縮形式ごとの特徴であり，ウイルスそのものの特徴点ではない．仮に一般の実行ファイルが UPX で圧縮されていた場合には表 3 のように高い確率で誤検出することとなる．これは Naive ベイズに限らず Paul Graham ベイズについても同様である．つまり，ウイルスの特徴点をとらえるには圧縮ウイルスを必ず解凍する必要があるといえる．次節では解凍したウイルスを提案手法で学習した際の評価を行う．

6. 提案手法の評価と考察

ここでは提案手法の評価を行う．まず最短長 l と処理速度の関係について述べ，次に最短長 l と検出率，誤検出率の関係について述べる．

6.1 処理速度

まず，Naive ベイズと Paul Graham ベイズの速度比較を行う．strings と見なす最短長 l を 4 から 28 まで変化させて，学習および入力をカテゴリに分類するのに要した CPU 時間を比較する．学習に 400 個，分類されるものに 400 個のファイルを用意した．

結果は**図 2** である．すべての区間で Paul Graham ベイズは Naive ベイズに比べて約 10 倍の速度が出ている．ウイルスの流行を防ぐためには，いかに素早く対応をしていくのかが重要な点であり，Paul Graham ベイズによるウイルスフィルタを実際のサーバやクライアントに実装する場合には，高速な処理速度が大きな利点となる．

次に提案手法の処理速度についてさらに詳しく見てゆく．**図 3** は図 2 から Paul Graham ベイズのグラフを抜き出し，1 分間にカテゴリ分類できるファイル数のグラフを追加したものである．処理時間は l が小さくファイルから抽出できる特徴点が多い場合には，約 4 分 30 秒を必要としているが，$20 \leq l$ の区間では約 30 秒に収束していることが分かる．これは $20 \leq l$ の区間ではすでに十分に長い strings しか残っていないため，l を多少変化させた程度では抽出される特徴点

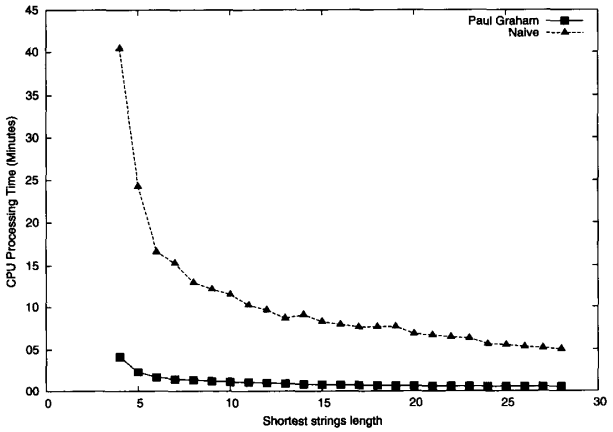


図 2 Naive ベイズと Paul Graham ベイズが学習とカテゴリ分類に要した時間
Fig. 2 Processing time Naive Bayes and Paul Graham Bayes.

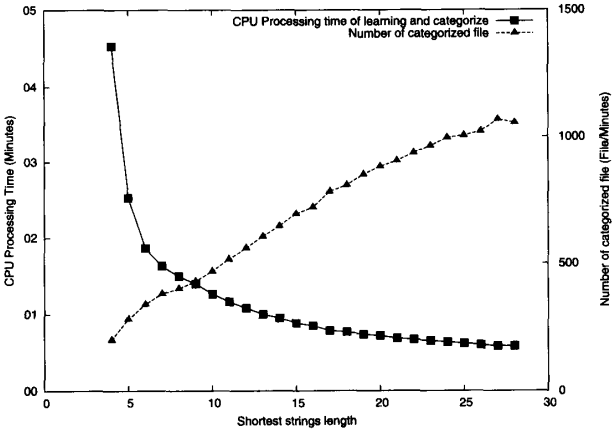


図 3 Paul Graham ベイズの処理時間
Fig. 3 Paul Graham Bayes processing time.

の数が変化せず計算時間には影響がないためである．また，1 分間にカテゴリ分類できる数は l に比例して大きくなっている．l によって 1 分間に 200 から 1,000 個ほどのファイルを処理できており，l に適切な値を設定することで，サーバやクライアントなどに提案手法を適用しても十分現実的な時間で処理が行えるといえる．

以上の結果より，提案手法を運用する場合に設定する l の値としては，$4 \leq l \leq 28$ の範囲がで適切であると考える．まず，l を 4 に設定すると図 3 のように，処理に極端に時間がかかるようになる．既存のアンチウイルスの問題点は，シグネチャ生成の遅延のために，未知のウイルスが検出できない期間が存在してしまうことである．提案手法は未知ウイルスを検出できる特徴を持っているが，そのためには提案シグネチャが生成されている必要がある．よって，l をこれ以上小さくし，学習時間をさらに延ばすと提案シグネチャの生成が遅れ，既存のアンチウイルスと同様に未知ウイル

スを検出できなくなってしまう可能性がある．ゆえに，最小値を 4 とする．また，図 3 より，l がある程度大きくなると，抽出できる特徴点の数に大きな変化がなくなり，処理時間が一定値に収束している．それゆえ，実験時の l 最大値を 28 としている．

6.2 検出率・誤検出率

ここでは l と検出率および誤検出率との関係を調べるために以下の実験を行う．まず，Netsky.B から Netsky.X までの 12 種類の Netsky の亜種ウイルスと Nonvirus1 とを学習し，各 Netsky の亜種向けの提案シグネチャを 12 個生成する．同時にすべての Netsky の亜種ウイルスを均等に混ぜ合わせて 200 個にしたものと Nonvirus1 を学習し，Mix 提案シグネチャを生成する．次に生成した各 Netsky の亜種向けの提案シグネチャを用いて，すべての Netsky の亜種ウイルスと Nonvirus2 を総当たりでカテゴリ分類する．つまり Netsky.B から生成した提案シグネチャを用いて，すべての Netsky の亜種ウイルスと Nonvirus2 を分類し，次は Netsky.C で作った提案シグネチャで分類を行う．これを Mix 提案シグネチャまで繰り返す．この実験を l を 4 から 28 まで変化させながら，検出率および誤検出率の関係を調べる．また，Bagle の亜種ウイルスに対しても同様にシグネチャを生成し，l を変化させながら Bagle の亜種ウイルスと Nonvirus2 をカテゴリ分類する．

6.2.1 全般的な傾向

結果は**図 4**，**図 5** である．図 4 における検出率とは，Netsky.B から Netsky.X の 12 個の提案シグネチャを使ってカテゴリ分類した場合に，それぞれの提案シグネチャでウイルスをウイルスであると検出した確率の平均である．また，誤検出率とは Netsky.B から Netsky.X の 12 個の提案シグネチャを使ってカテゴリ分類した場合に，それぞれの提案シグネチャで一般の実行ファイルである Nonvirus2 を誤ってウイルスであると検出した確率の平均である．また，Bagle に関しても同様の操作で図示したものが図 5 である．

まず Netsky に絞って話を進める．図 4 からすべての l において検出率が約 70%で推移していることが分かる．これは l によって検出率が大きな影響を受けず，各 l ごとに適切な特徴点を Paul Graham ベイズによって選択されていることを示している．

実際にどのような特徴点をウイルス確率計算に使ったかを示したのが**表 4** である．$l = 4$ のように極端に小さな値のときには特に意味を持たない strings が計算に使われているのが分かる．これらの string の多くは API や DLL を表したものではなく実行ファイルの

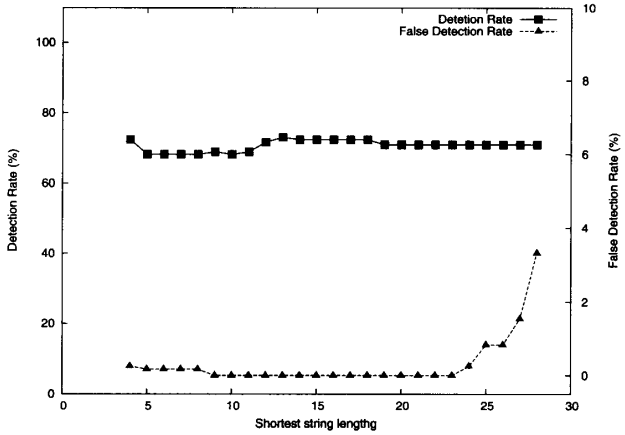


図 4 Netsky の検出率および誤検出率
Fig. 4 Netsky detection rate and false detection rate.

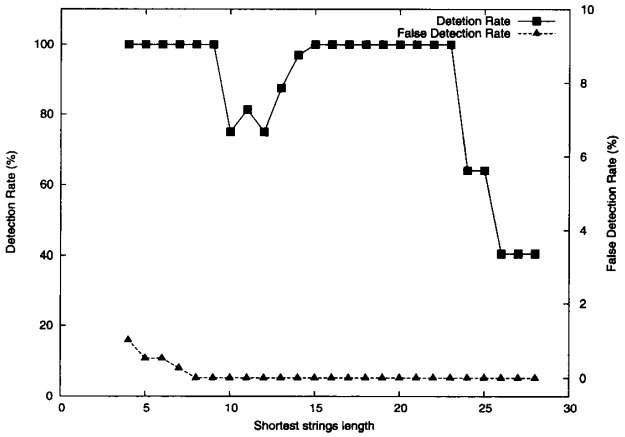


図 5 Bagle の検出率および誤検出率
Fig. 5 Bagle detection rate and false detection rate.

バイナリ列の一部が偶然 ASCII コードの並びであったものである．これらの string に高いウイルス確率が与えられている場合には，この特徴点がウイルスカテゴリに頻繁に現れることを示している．バイナリの並びはウイルスを作るときの技法やコンパイラの出力の仕方などに依存しているので，ウイルスカテゴリに多く現れるこれらの特徴点は，ウイルス作者が頻繁に用いる何らかの命令や技法である可能性がある．よって，これらの string は特徴点としての性質を十分に有しているものである．しかし，l が小さな値であると膨大な数の特徴点が抽出されるため，図 3 のように学習に大きな時間がかかるようになる．よって，頻繁に学習を行いたい場合には極端に小さな l は避けるのが好ましい．

$l = 13$ のような長さのときには，使用している API，書き込むレジストリのキーや内部で使用している値などが計算に使われていることが分かる．この中で注目すべきはレジストリのキー部分である．ほとんどのウイルスは再起動時にも動作ができるように，Run や RunServices の項目に書き込みを行う．この例ではそ

表 4 ウイルス確率計算に使われた特徴点
Table 4 Features used for virus probability calculation.

Feature	Virus Probabililty
$l = 4$	
D$(j	0.01
exit	0.01
QQSV	0.01
_WVP	0.98
}f:F	0.99
talu	0.99
s55.	0.99
$l = 13$	
CoMarshalInterThreadInterfaceInStream	0.01
For more information about these matters, see the files	0.01
GetSecurityDescriptorDacl	0.01
SOFTWARE/Microsoft/Windows/CurrentVersion/RunServices	0.99
SOFTWARE/Microsoft/Windows/CurrentVersion/Run	0.98
Windown Longhorn Beta Leak.exe	0.99
GdipDisposeImage	0.99

れらの strings を学習し，人間が意図的に値を調整しなくても危険な特徴点として認識している．

次に l が極端に大きくなった場合に誤検出が増えている原因について述べる．これはファイルから抽出できる特徴点が少なくなっているため，1 つの特徴点のウイルス確率が全体のウイルス確率に大きな影響を与えるためである．たとえば，$l = 27$ までは 2 つの特徴点を抽出でき，それらに 0.90, 0.10 というウイルス確率が与えられているとする．すると，ファイル全体としてのウイルス確率は 0.50 である．しかし $l = 28$ になって 0.90 側の特徴点しか抽出できなくなったとすれば，ファイル全体としてのウイルス確率も 0.90 になってしまう．これは，たかだか 1 つの特徴点を見てウイルスだと判定しており，正しい判定である可能性は低い．特徴点を絞り込みすぎることで，この例のようなカテゴリ分類が行われてしまう恐れが高いので極端に大きな l は避けるべきである．

次に Bagle について述べる．l が大きくなると検出率が大きく下がるのが特徴である．これは上でも述べたように特徴点が少なくなるのが原因である．**図 6** は Bagle から抽出できる strings の数と l の関係を表したものである．Bagle.AB は $19 < l$ の部分で，抽出できる strings が 15 個を下回っているのが分かる．よって，これより先は 1 つの特徴点が全体に与える影響力が大きくなっていき，公正な判断がされにくくなる．$l = 28$ 時点では特徴点が 1 つしか抽出できなくなり，これらのウイルスで生成したシグネチャで他の亜種を検出できず，さらには自分自身も検出されないという状況がみられた．このような現象は Bagle.Z, Bagle.AE でも発生し，それによって全体の検出率が大きく下がった．

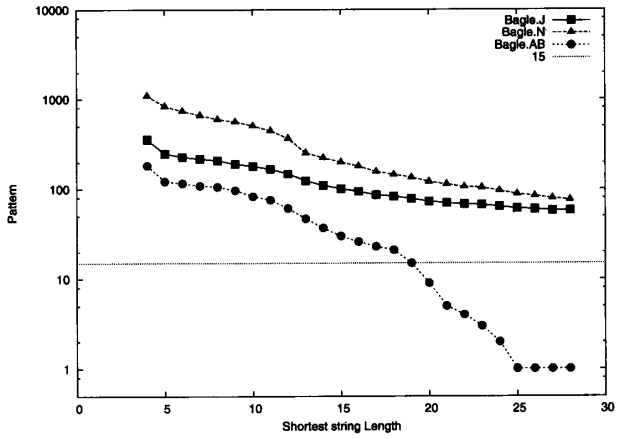


図 6 文字列のパターン数
Fig. 6 Number of strings pattern.

以上のことをふまえて，Netsky, Bagle を検出する際には最短長 l に 19 より小さな値を設定することが好ましいといえる．

6.2.2 シグネチャごとの検出の傾向

ここでは $l = 13$ のときに，個々の提案シグネチャが検出しているウイルスの様子を示す．$l = 13$ の点を選んだのは Netsky の亜種検出実験において最も良い結果が得られたからである．

結果は**表 5**, **表 6** である．表は各シグネチャに対して特定の亜種を入力した場合に，どの程度検出を行っているかを示しており，○は提案シグネチャで 100%正しくカテゴリ分類ができたことを示している．つまり，入力がウイルスの部分での○はウイルスを検出し，入力が Nonvirus2 部分ではウイルスの誤検出をしなかったことを示している．空白の場所は正しいカテゴリ分類がまったくできなかったことを示している．

表 5 Netsky の亜種検出率 (l=13)
Table 5 Variety of Netsky detection rate (l=13).

	Input												
Signature	B	C	D	J	M	N	P	Q	S	T	W	X	Nonvirus2
Netsky.B	○	○	○	○	○	○			○	○	○	○	○
Netsky.C	○	○	○	○	○	○	○		○	○	○	○	○
Netsky.D	○	○	○	○	○	○			○	○	○	○	○
Netsky.J	○	○	○	○	○	○			○	○	○	○	○
Netsky.M	○	○	○	○	○	○			○	○	○	○	○
Netsky.N	○	○	○	○	○	○			○	○	○	○	○
Netsky.P		○					○	○					○
Netsky.Q								○					○
Netsky.S	○	○	○	○	○	○			○	○	○	○	○
Netsky.T	○	○	○	○	○	○			○	○	○	○	○
Netsky.W	○	○	○	○	○	○			○	○	○	○	○
Netsky.X	○	○	○	○	○	○			○	○	○	○	○
Mix	○	○	○	○	○	○	○	○	○	○	○	○	○

表 6 Bagle の亜種検出率 (l=13)
Table 6 Variety of Bagle detection rate (l=13).

	Input								
Signature	J	K	N	O	Y	Z	AB	AE	Nonvirus2
Bagle.J	○	○	○	○					○
Bagle.K	○	○	○	○					○
Bagle.N	○	○	○	○	○	○	○	○	○
Bagle.O	○	○	○	○	○	○	○	○	○
Bagle.Y	○	○	○	○	○	○	○	○	○
Bagle.Z	○	○	○	○	○	○	○	○	○
Bagle.AB	○	○	○	○	○	○	○	○	○
Bagle.AE	○	○	○	○	○	○	○	○	○
Mix	○	○	○	○	○	○	○	○	○

ここからは，Netsky の検出の様子を表 5 と図 4 の関係から述べる．図 4 の $l = 13$ 部分での検出率が約 70%であるのは，表 5 のように，ほとんどの提案シグネチャで Netsky.P，Netsky.Q を検出することができず，さらには，Netsky.P と Netsky.Q から生成した提案シグネチャで，他の Netsky の亜種を検出できないことが原因である．このような傾向はすべての l で見られ，これによってシグネチャごとの検出率の平均を大きく下げている．また，Netsky.B シグネチャは l によって Netsky.S，Netsky.T，Netsky.X を検出できない区間があり，それにより図 4 の検出率に多少の影響を与えている．

次に，シグネチャと入力の組合せによって，完全に検出できる場合とまったく検出できない場合とに 2 極化している理由について述べる．これは，1 種類の亜種の中ではファイル間に多様性がほとんどないためである．たとえば，Netsky.B，Netsky.C，Netsky.D，Netsky.Q，Netsky.P はそれぞれの 200 個のファイル中で複数のパターンに分かれているものの，strings を抽出すると同じ亜種のファイルからはほぼ同一の strings が得られる．それゆえ，シグネチャと入力の組合せによって完全に検出できる，あるいはまったく検出できないという現象が起きる．このような理由により l を変化させたときにも，特定のシグネチャと入力の組合せによる検出率は 0%か 100%という単位で変化することとなる．ただし，一般の実行ファイルの入力である Nonvirus2 は多様性があるため，ファイル全体のウイルス確率も様々な値をとり，しきい値を超えたものに関しては図 4 において，誤検出率として表現されている．

次に，提案シグネチャで未来の亜種を検出できることについて述べる．Netsky.B から生成した提案シグネチャは，Netsky.P，Netsky.Q を除いてすべての亜種ウイルスを検出しており誤検出もまったくない．このとき，Netsky.B にとって他の亜種は未来に発生するものだが，Netsky.B が手に入った時点で提案シグネチャを生成すれば，未来の亜種も検出できることを示している．これは，亜種どうしは互いに似たような特徴点を持ち，それらの特徴点は一般の実行ファイル中には頻繁に現れないことを示している．よって 3.2 節で述べた亜種どうしは似ているという仮定を証明しているといえる．この性質により，Netsky.B 以外から生

成した提案シグネチャも，未来に発生する似た亜種を検出できている．逆に提案手法は同じ特徴点を持っていないものは検出ができない．Netsky.P と Netsky.Q に関しては，抽出できる strings が少なく，他の Netsky の亜種との共通点も少ないため，他の提案シグネチャで Netsky.P，Netsky.Q を検出することができず，さらには Netsky.P と Netsky.Q から生成した提案シグネチャで，他の亜種を検出できないという現象が起きた．ただし，Netsky シリーズの中で Netsky.P と Netsky.Q だけは他の亜種と動作が大きく異なるという報告[8] もあり，この 2 種を Netsky の亜種と見なすべきなのか議論の余地があることを付け加えておく．

6.3 提案手法の運用

提案手法と既存のアンチウイルスを組み合わせて運用することで，シグネチャ更新まで未知ウイルスが検出できないという，既存のアンチウイルスの欠点を補完することができる．

既存のアンチウイルスによる検出では，未知ウイルスの発生が確認されてからシグネチャが更新されるまでの時間は，最も対応が速いアンチウイルスメーカで約 4 時間を必要とし，アンチウイルスメーカ各社の平均では約 10 時間を要する[18]．そのため表 5 の 12 種類の Netsky が発生していくと既存のアンチウイルスでは合計で $12 \times 10 = 120$ 時間の間，未知ウイルスが検出不可能な時間が存在することになる．

以下では上記の時間を最小限に抑える運用方法を表 5 を用いて説明する．前提として 10 時間ごとにアンチウイルスのシグネチャ更新が行われ，更新の直後に未知ウイルスが発生するものとする．さらに，アンチウイルスはシグネチャの更新直後に発生したウイルスだけは検出できないものとする．

まず Netsky.B が発生する．しかしシグネチャには対応する情報がないので，アンチウイルスでは検出ができない．10 時間後にシグネチャが更新されて Netsky.B が検出できるようになったら Netsky.B 用の提案シグネチャを生成する．これによって，Netsky.B が検出可能となり検出されたものがウイルスカテゴリへ分類されるようになる．

次に Netsky.C が発生すると，発生初期にはアンチウイルスでこれを検出することができない．しかし，先ほど生成しておいた Netsky.B の提案シグネチャでは Netsky.C も検出することができるため，アンチウイルスのシグネチャの更新を待たずに Netsky.C はウイルスカテゴリへの分類されるようになる．Netsky.C が発生してから 10 時間後にはアンチウイルスでも Netsky.C を検出可能となる．そのときにアンチウイルスでウイルスカテゴリ内をスキャンすると Netsky.B，Netsky.C，その他のファイルの 3 種類が存在する可能性がある．そこで，この中から Netsky.B と Netsky.C だけを取り出して均等に混ぜたものを提案手法により学習させ，新たな提案シグネチャを生成し直す．この提案シグネチャは，すべての Netsky を含んだ Mix 提案シグネチャと同様に，混ぜ合わされたウイルスの特徴点を含んだものであるため，Netsky.B と Netsky.C の性質を継承し Netsky.Q 以外を検出できることが確認されている．

次に Netsky.D が発生するが，先ほどの提案シグネチャで検出することができる．10 時間後にアンチウイルスで Netsky.D が検出できるようになったら，Netsky.B，Netsky.C，Netsky.D を均等に混ぜたものを学習させる．このように定期的に提案シグネチャを作り直していくことで，Netsky.N までを検出し続けていく．

その次に Netsky.P が発生するが，その時点では Netsky.C の情報が混ぜ合わされた提案シグネチャが生成されているため，Netsky.P も検出可能であり，さらに Netsky.P の情報より Netsky.Q も検出可能となる．このような方法でシグネチャを作り直していくことで，誤検出もなく最終的にはすべての Netsky の特徴点を含んだ Mix 提案シグネチャが生成されることが確認できた．

このように既存のアンチウイルスのシグネチャ更新に合わせて提案シグネチャを更新していくことで，Netsky.B から提案シグネチャを生成した後は，すべての Netsky を検出し続けることができる．ゆえに，Netsky.B が発生してからアンチウイルスによって検出可能になるまでの 10 時間が，上記の運用における検出不可能な時間である．アンチウイルスのみによる運用では検出不可能な時間が合計で 120 時間存在したので，上記の運用で大幅に低減できたといえる．

6.4 検出回避の可能性

最後にウイルスが提案手法の検出を回避する可能性について述べる．既存のアンチウイルスは，ウイルス判定をファイルのごく一部がシグネチャと一致するか否かで行っている．よって，ウイルス作者は，ウイルス全体の複数箇所の修正を行って亜種とすることで検出を回避してきた．しかし，提案手法はウイルス確率の高い特徴点を 15 個だけ選び出すため，複数箇所を変更した亜種でもオリジナルが持っている特徴点が残っている限りは検出できることが 6.2.2 項の結果からも明らかになった．

亜種は 2.1 節で述べたように，オリジナルのウイ

ルスを少しだけ改造して生成されるものである．それゆえ，ウイルス作者はコストをかけずに次々と新しい亜種を産み続けることができる．しかし提案手法ではオリジナルの特徴点が残っている限りは亜種の多くを検出できるので，ウイルス作者がこれを回避するためには，オリジナルのウイルスに大幅な改造を加える必要がでてくる．よって，既存のアンチウイルスに比べ検出回避が困難である．また，大幅な改良を加えることで検出が回避される可能性は高まるものの，ウイルス作者にそれなりの改造コストを強いることになる．よって，提案手法が普及すれば，未知ウイルスの発生速度そのものを低減させられる可能性がある．

7. おわりに

本論文では，実行可能圧縮形式がウイルス検出に与える悪影響を述べるとともに，解凍したウイルスの特徴点を Paul Graham ベイズで学習し，共通点を持つ未知のウイルスを検出する手法を提案した．提案手法に用いた Paul Graham ベイズは Naive ベイズより処理速度が約 10 倍速く，学習データ strings の最短長 l を 19 より小さく設定することで，Netsky の亜種と Bagle の亜種を短時間に検出できることを明らかにした．また，提案手法と既存のアンチウイルスを組み合わせて用いることで，既存のアンチウイルスが持つ未知ウイルスが検出不可能な時間を最小限に抑え，未知ウイルスの発見速度にシグネチャの生成が間に合わないという問題を解決できることを示した．なお，提案手法は特定の感染経路向けに特化したものではない．実装の方法によっては，P2P を媒介とするような，現在より感染力の強いウイルスにも対抗できる可能性がある．

今後は，亜種ウイルスに限らず，未知ウイルス全般を使っての提案手法の評価を行っていく予定である．

参考文献

1) IPA: 2004 年第 3 四半期コンピュータウイルス届出状況，技術報告，情報処理推進機構セキュリティセンター (2004). http://www.ipa.go.jp/security/txt/2004/documents/2004q3-v.pdf
2) Christodorescu, M. and Jha, S.: Static analysis of executables to detect malicious patterns, *12th USENIX Security Symposium*, Washington, DC, Advanced Computing Systems Association (2003).
3) Kephart, J. and Arnold, B.: Automatic Extraction of Computer Virus Signatures, *the 4th Virus Bulletin International Conference*, England, Virus Bulletin Ltd, pp.178–184 (1994).
4) 中谷直司，小池竜一，厚井裕司，吉田等明：メール型未知ウイルス感染防御ネットワークシステムの提案，情報処理学会論文誌，Vol.45, No.8, pp.1908–1920 (2004).
5) 神薗雅紀，白石善明，森井昌克：仮想ネットワークを使った未知ウイルス検知システム，コンピュータセキュリティ研究会，Vol.16, No.22, pp.113–120 (2003).
6) Schultz, M.G., Eskin, E., Zadok, F. and Stolfo, S.J.: Data Mining Methods for Detection of New Malicious Executables, *Security and Privacy*, Oakland, CA, USA, IEEE, pp.38–49 (2001).
7) Schultz, M.G., Eskin, E., Zadok, E., Bhattacharyya, M. and Stolfo, S.J.: MEF: Malicious Email Filter — A UNIX Mail Filter that Detects Malicious Windows Executables, *Proc. Annual USENIX Technical Conference, FREENIX Track*, Boston, MA, pp. 245–252 (2001).
8) Symantec Corporation: Security Response. http://securityresponse.symantec.com/
9) IPA：「W32/Netsky」ウイルスの亜種（Netsky.Q）に関する情報 (2004). http://www.ipa.go.jp/security/topics/newvirus/netsky-q.html
10) Microsoft Corporation: *Microsoft Portable Executable and Common Object File Format Specification* (1999).
11) FRISK Software International: F-Prot. http://www.f-prot.com/index.html
12) PEiD. http://peid.has.it/
13) Virus Bulletin: the VB 100% award. http://www.virusbtn.com/vb100/
14) Graham, P.: A Plan for Spam (2002). http://www.paulgraham.com/spam.html
15) Graham, P.: Better Bayesian Filtering (2003). http://www.paulgraham.com/better.html
16) 渡部 洋：ベイズ統計学入門，福村出版 (1999).
17) Massey, B., Thomure, M., Budrevich, R. and Long, S.: Learning Spam: Simple Techniques For Freely-Available Software., *USENIX Annual Technical Conference, FREENIX Track*, USENIX, pp.63–76 (2003).
18) Marx, A.: Anti-Virus Outbreak Response Testing and Impact, *Virus Bulletin 2004 Conference Presentation*, Chicago (2004).

(平成 16 年 11 月 29 日受付)
(平成 17 年 6 月 9 日採録)

小池　竜一

2003年岩手大学工学部情報工学科卒業．2005年同大学院工学研究科博士前期課程修了，同年同大学院工学研究科博士後期課程入学，現在に至る．ネットワークセキュリティに関する研究に従事．電子情報通信学会学生員．

中谷　直司

1994年埼玉大学工学部電子工学科卒業．1996年同大学院博士前期課程修了．1999年同大学院博士後期課程修了．同年岩手大工学部情報システム工学科教務職員．2001年同科助手，現在に至る．進化型アルゴリズム，ネットワークセキュリティに関する研究に従事．博士（学術）．電子情報通信学会会員．

萩原由香里

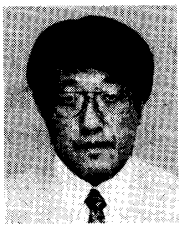
1993年東京農工大工学部機械システム工学科卒業．1995年電気通信大学大学院情報システム学研究科博士前期課程修了．同年東京農工大学工学部技術職員，2002年～岩手大学工学部技術職員．コンピュータセキュリティ，パターン認識の研究に従事．

厚井　裕司（正会員）

1970年東京理科大学理学部応用物理学科卒業．同年三菱電機（株）入社．2001年岩手大学工学部情報システム工学科教授，現在に至る．主として，マルチメディアネットワーク，ネットワークセキュリティ，RF-IDタグに関する研究に従事．工学博士．IEEE，電子情報通信学会各会員．

高倉　弘喜（正会員）

1990年九州大学工学部情報工学科卒業．1992年同大学院情報工学専攻修了．1995年京都大学大学院博士課程修了．博士（工学）．イリノイ大学訪問研究員，奈良先端科学技術大学院大学情報科学研究科助手，京都大学大学院工学研究科講師，同大大型計算機センター助教授を経て，2002年同大学術情報メディアセンター助教授．大規模ネットワークにおけるセキュリティの研究，地理情報システム等の研究に従事．地理情報システム学会，システム制御情報学会，ACM各会員．



吉田　等明（正会員）

1987年東北大学大学院博士後期課程化学専攻修了．同年筑波大学技官．1989年同大学化学系助手．1991年岩手大学工学部助手．1995年同学部助教授，現在に至る．計算機化学，ニューラルネットワーク，遺伝的アルゴリズム，暗号等に関する研究に従事．理学博士．計測自動制御学会，日本化学会，米国化学会各会員．