命令の乱雑さに基づくプログラム理解性の評価

Evaluation of Software Understandability Based on the Randomness of Instructions

二村 阿美[*] **門田 暁人**[†] **玉田 春昭**[‡] **神崎 雄一郎**[§] **中村 匡秀**[¶] **松本 健一**[‖]

あらまし 本論文では，情報理論からのアプローチにより，プログラムの理解性を定量化する．基本アイデアは，各命令が全くランダムに（乱雑に）プログラム中に出現する場合に，プログラムの理解性が最も低い状態であると考える．命令がランダムに出現するとは，(1) 各命令の出現頻度が等しく，かつ，(2) その並びに規則性がない状態であると考える．前者はエントロピー，後者はコルモゴロフ複雑性の概念を用いて定量化する．さらに，コルモゴロフ複雑性により，プログラマが理解すべき最低限の情報量を定量化する．ケーススタディを通して，本提案の妥当性や今後の課題を考察する．

1 はじめに

近年，不正行為を目的としたプログラム解析を防ぎたいという社会的要求が高まっており，数多くのプログラム難読化法が提案されてきた [3]．

しかし，プログラム難読化の度合い（理解性）やその効果を評価することは容易でない．プログラム理解の目的や手段が多種多様なためである．例えば，プログラムの複雑さメトリクスにより理解の困難さを評価する試みがある [2] が，複雑さメトリクスとプログラム理解に要する時間との間には必ずしも相関がないことも示されている [10]．また，人間にプログラム解析を行わせて評価する方法 [6] やゴール木による評価 [14] も行われているが，評価に多大な労力を要する点が問題となる．

本論文では，情報理論からのアプローチにより，プログラムの理解性を定量化することを試みる．基本アイデアは，プログラムを読んでもそれがノイズにしか見えない状態，つまり，各命令が全くランダムに（乱雑に）プログラム中に出現する場合に，プログラムの理解性が最も低い状態であると考える．そして，命令の乱雑さを定量化することにより，理解性の評価を行う．ここで，命令の出現がランダムであるとは，(1) 各命令の出現頻度が等しく，かつ，(2) その並びに規則性がない状態であると考える．本論文では，前者はエントロピーの概念を，後者はコルモゴロフ複雑性の概念を用いることで評価する．

また，本論文では，理解すべき情報量が多いプログラムほど理解性が低い（理解により多くの時間を要する）と考え，コルモゴロフ複雑性の概念を用いて，プログラマが理解すべき最低限の情報量を定量化する．

本アプローチは，低コストで評価値の算出が可能であり，プログラム中の命令が「ランダムといえる状態にどの程度近いか」，および,「理解すべき情報量」という観点での理解性を評価できる．

なお，本論文では，理解性の評価対象として，オペコードのみを扱う．オペランドの扱いについては今後の課題とする．また，ケーススタディでは，C 言語をコン

[*]Ami Futamura, 奈良先端科学技術大学院大学 情報科学研究科

[†]Akito Monden, 奈良先端科学技術大学院大学 情報科学研究科

[‡]Haruaki Tamada, 京都産業大学 コンピュータ理工学部

[§]Yuichiro Kanzaki, 熊本高等専門学校 人間情報システム工学科

[¶]Masahide Nakamura, 神戸大学 大学院工学研究科

[‖]Kenichi Matsumoto, 奈良先端科学技術大学院大学 情報科学研究科

パイルして得られる x86 アセンブリプログラムを対象とした実験を行う．以降では，提案方法，および，ケースタディについて述べる．

2 提案手法

2.1 エントロピーによる命令出現頻度の乱雑さの評価

エントロピーとは，情報の乱雑さ（無秩序さ）を定量的に表す指標である [13]．シャノンの情報理論においては，1 つの事象（情報源シンボル）が生起したことを知ったことにより得られる情報量の期待値 (平均情報量) として位置づけられる [8]．エントロピー $H(S)$ は，式 (1) で定義される．

$$H(S) = -\sum_{i=1}^{n} P(s_i) \log_2 P(s_i) \qquad [\text{ビット/シンボル}] \tag{1}$$

ここで，s_i は情報源シンボル，$S = \{s_1, s_2, ..., s_n\}$ は情報源アルファベット，$P(s_i)$ は s_i の生起確率を表し，$\sum_{i=1}^{n} P(s_i) = 1$ である．

本論文では，s_i を命令（オペコード），S を全命令の集合とし，$P(s_i)$ は与えられたプログラムにおける命令 s_i の出現頻度とする．プログラム中の全ての命令の出現頻度が等しいとき，エントロピー $H(S)$ は最大値 $\log_2 n$ をとり，最も難読化されている状態であると考える．

ただし，一般に，n が大きいほどエントロピーが大きくなるため，最大エントロピーで正規化した値として，(2) 式のとおり相対エントロピー $h(S)$ が定義される．

$$h(S) = \frac{H(S)}{\log_2 n} \tag{2}$$

本論文では，難読化の評価に，この相対エントロピー $h(S)$ を用いる．全ての命令の出現頻度が等しいとき，つまり，最も難読化されている場合，$h(S) = 1$ となる．

従来研究において，神崎ら [4] は，(一般的なプログラムと比較して）出現頻度が著しく低い命令がプログラム中に含まれる場合，その命令がプログラム解析の手がかりとなり得ることを示している．本論文のアプローチでは，相対エントロピー $h(S)$ が 1 に近づくと，どの命令も出現頻度がほぼ等しくなるため，プログラム解析の手がかりが少なくなることになる．

なお，上記のエントロピーの定義では，記憶のない情報源が仮定されているが，実際のプログラムでは，各命令は互いに独立に出現するとはいえず，エルゴードマルコフ情報源を仮定する方が望ましい．ただし，その場合，各命令の出現パターンを長期間観測する必要があり，相当に長いプログラムでなければエントロピーを正しく算出できないという問題があるため，今後の課題とする．

2.2 コルモゴロフ複雑性による命令出現順序の乱雑さの評価

2.2.1 コルモゴロフ複雑性の概念

プログラムに含まれる命令群の乱雑さを評価するにあたって，(記憶のない情報源を仮定した）エントロピーによる評価だけでは不十分である．エントロピーが最大である状態，つまり，各命令の出現頻度が一様であっても，プログラム中の命令順序に何らかのパターン (規則性）がある場合，プログラムの理解性を高め得るためである．

ここでは，ADD と SUB からなる命令数 10 の次の 2 つのプログラムを考える．

プログラム 1: ADD SUB ADD SUB ADD SUB ADD SUB ADD SUB

プログラム 2: SUB ADD SUB SUB ADD ADD SUB ADD ADD SUB

いずれのプログラムも ADD と SUB の出現回数は同じであるためエントロピーは等しいが，前者は”ADD SUB”のパターンが 5 回繰り返されるのに対し，後者は特段のパターンが見られないため，直感的には後者の方が理解性が低いといえる．本論

文では，この 2 者を区別する概念としてコルモゴロフ複雑性を導入する．

コルモゴロフ複雑性は，文字列の複雑さを表現する概念であり，与えられた文字列を出力するプログラムの最小の長さとして定義される．前述の 2 つの命令列の例では，前者は，例えば，

print "ADD SUB"× 5

のように短く書くことができるが，後者は，前者よりは短く書くことができない．従って，後者は前者よりも大きなコルモゴロフ複雑性を持つことになる．

より厳密には，コルモゴロフ複雑性 $K_u(x)$ は，有限長の文字列 x，および，万能計算機（万能チューリングマシン）u が与えられたときに，計算機 u のための x を出力するプログラム p の最小の長さとして定義される．なお，コルモゴロフ複雑性は，計算機（プログラミング言語）に依存した関数であるが，言語の選択は本質ではない．言語の選択によってコルモゴロフ複雑性はたかだか定数分しか増減しないためである．

プログラム理解においては，プログラム p のコルモゴロフ複雑性 $K_u(p)$ は，プログラマが理解すべき最低限の情報量を表していると考えられる．プログラム p が大きくても，$K_u(p)$ が小さいならば，理解すべき情報量は少なくて済む．例えば，前述のプログラム 1 のように繰り返しを含む場合，そうでないプログラム 2 よりも理解すべき情報量は小さい．

本論文では，p に含まれる命令列の「規則性のなさの度合い」を評価したいため，プログラム p の長さ $l(p)$ に対する $K_u(x)$ の比を，相対コルモゴロフ複雑性 $k_u(p)$ として次の通り定義する．

$$k(p) = \frac{K(p)}{l(p)} \tag{3}$$

p の命令順序に規則性が全くない場合（つまり，p が最も難読化されている場合），p を出力するプログラムを短く記述できなくなり，$k(p) \simeq 1$ となる．

なお，相対コルモゴロフ複雑性が最大値を取るためには，相対エントロピー $h(S) = 1$ を満たすことが必要条件となる．$h(S) < 1$ の場合，すなわち命令の出現頻度に偏りがある場合は，出現頻度の高い命令をより短いビット列で符号化して保持し，それを復号するようなプログラムを書くことで，短い $K(p)$ を作成できるためである．

2.2.2 コルモゴロフ複雑性の算出

実は，コルモゴロフ複雑性は計算不能である [7]．そのため，文字列 x を可逆圧縮により究極に圧縮した時のビット数 $C(x)$ として近似する方法が提案されている [1]．文字列 x に何らかの規則性や冗長性が含まれる場合，x は圧縮可能なため $C(x)$ は x のサイズ $l(x)$ より小さくなり，x に規則性や冗長性が含まれない場合は圧縮できないため $C(x) \simeq l(x)$ となる．なお，$C(x)$ は圧縮ツールに依存するため，できる限り圧縮効率の高いツールを選ぶ必要がある．(本論文では，既存の圧縮ツールの中で圧縮効率の高い bzip2 を用いる．)

ただし，実際の命令列 p を圧縮して $C(p)$ を求める場合には注意が必要である．命令の乱雑性を正しく評価するために，各命令をなるべく対等に表現（符号化）しておく必要がある．例えば，ある命令の表現が別の命令の表現のサブセットになっている場合（たとえば，mul と imul），そうでない場合（add と sub）よりも圧縮できてしまい，命令の乱雑さを正しく評価できなくなる．

この問題を緩和するために，本論文では，プログラムの圧縮に先立って，各命令を固定長のビット列により符号化しておくこととする．各命令を区別可能な最小のビット長を決め，各命令にビット列を割り当ててプログラムを表現する．命令の種類数を r とすると，例えば，$r = 32$ の場合，$5(= \log_2 32)$ ビットで全命令を区別して表現できる．本論文では，コルモゴロフ複雑性の近似値 $K(p)$ は，符号化されたプログラム p を圧縮したときの $C(p)$ として定義する．

次に，相対コルモゴロフ複雑性 $k(p)$ の算出について述べる．ここで注意すべきことは，命令の種類数 r が 2 のべき乗でない場合，命令が割り当てられない（つまりプログラム中に出現しない）ビットパターンが生じることである．例えば，$r = 33$ の場合，5 ビットではぎりぎり全命令を区別できないため，1 命令あたり 6 ビット必要になるが，冗長な（命令が割り当てられない）ビットパターンが多数存在することになる．このような場合，たとえ命令の出現順序が（究極的に）乱雑であっても，プログラムを圧縮できてしまい，乱雑さの度合いを正しく評価できないことになる．

そこで，圧縮前のプログラムの長さ $l(p)$ を，冗長なビットがないと仮定した場合のプログラムの長さとして与えることとする．具体的には次式の通りである．

$$l(p) = n \times \log_2 r \tag{4}$$

ここで，n はプログラム p に含まれる命令数である．これを用いて，相対コルモゴロフ複雑性の近似値 $k(p)$ を次式により求める．

$$k(p) = \frac{K(p)}{l(p)} \tag{5}$$

命令順序に規則性が全くない場合，p を圧縮できたとしても圧縮後のサイズは高々 $l(p)$ となり，$k(p) \simeq 1$ となる．

3　ケーススタディ 1：難読化の評価

3.1　目的

本ケーススタディの目的は，難読化前のプログラムと難読化後のプログラムで，相対エントロピー $h(S)$，コルモゴロフ複雑性 $K(p)$，相対コルモゴロフ複雑性 $k(p)$ をそれぞれ計測し，これらの尺度が難読化の評価に利用できるかどうかを検討することである．

3.2　題材

評価対象は，AES(Advanced Encryption Standard) [11] による暗号化・復号を行う C プログラムをコンパイルして得られる x86 アセンブリプログラムである．コンパイルには，cygwin 環境の gcc 4.5.3 を用いた．

この AES プログラムに対し，次の 5 つの難読化手法を適用した．

- O_1　ループ難読化 [9]
- O_2　関数の分割・マージ [3]
- O_{3a}　偽装コードの追加（ランダム） [15]
- O_{3b}　偽装コードの追加（パターンを消す） [15]
- O_4　高レベル命令を低レベル命令に書き換える [15]

O_1,O_2 はソースコードを，O_{3a},O_{3b},O_4 はアセンブリコードを対象としている．難読化にあたっては，まず，対象の C プログラムに難読化 O_1,O_2 をそれぞれ適用する．難読化後のプログラムをコンパイルして，アセンブリプログラムを作成する．それぞれ $O_1(p)$,$O_2(p)$ とする．一方，難読化 O_{3a},O_{3b},O_4 については，対象の C プログラムをコンパイルして得られたアセンブリプログラムに適用する．難読化後のプログラムをそれぞれ $O_{3a}(p)$,$O_{3b}(p)$,$O_4(p)$ とする．

アセンブリプログラムの命令数および命令の種類数を表 1 に示す．表 1 に示されるように，命令数は，難読化後に増加している．命令の種類数は，難読化後に概ね増加しているが，$O_4(p)$ は，高レベル命令がプログラム中から存在しなくなったため，難読化前と比較して命令の種類数が減少している．

次節では，それぞれの難読化について簡単に説明する．

表 1　アセンブリプログラムの命令数および命令の種類数

プログラム	命令数 n	命令の種類数 r
p (難読化前)	1355	31
$O_1(p)$	1411	32
$O_2(p)$	1388	32
$O_{3a}(p)$	1384	31
$O_{3b}(p)$	1717	33
$O_4(p)$	1429	27

3.3　難読化手法

3.3.1　O_1 ループの難読化

ループの制御構造を複雑にする難読化である．
O_1 を適用した例を図 1 に示す

```
void Mult(int M,int A[]){
  int i;

  for(i=0;i<M;i++){
    if(A[i]) A[i] *= 2;
    printf("A[%d]=%d\n",i,A[i
        ]);
  }
}
```

```
void Mult(int M,int A[]){
  int i=0;

  if(i<M){
    while(1){
      if(A[i]) A[i] *= 2;
      printf("A[%d]=%d\n",i,A[
          i]);
      i++;
      if(!(i<M)) break;
    }
  }
}
```

図 1　O_1 の例

3.3.2　O_2 関数の分割・マージを行う難読化

機能別に分けられた関数を分割または統合する難読化である．関数の機能の抽象度が増すことで，プログラムの理解性が低下すると考えられる．

3.3.3　O_{3a} 偽装コードの追加（ランダム）

偽装コードの生成方法には，新規に偽装コードを生成する方法と，オリジナルコードを複製・改変してそれを偽装コードとして使用する方法がある [15]．

O_{3a} では，後者を行った．O_{3a} を適用した例を図 2 に示す．

3.3.4　O_{3b} 偽装コードの追加（パターンを消す）

O_{3b} ではランダムに偽装コードを追加したが，O_{3b} では命令順序のパターンを消去するように偽装コードを追加する．オペコードの命令順序についてパターンを計測し，出現数の多いパターンを削除するために，偽装コードを追加する．追加する命令は，実行に影響を与えないものである．O_{3b} を適用すると，命令順序に規則性がなくなり，コルモゴロフ複雑性は増大すると考えられる．O_{3b} を適用した例を図 3 に示す．

3.3.5　O_4 高レベル命令を低レベル命令に書き換える難読化

アセンブリ命令には様々な種類が存在し，特定の命令を他の複数の命令で書き換えることも可能である．O_4 は，高レベル命令を add,mov,jmp,or,and などの低レベル命令に書き換える．O_4 を適用した例を図 4 に示す．

村山ら [15] は，解析しづらいプログラムを生成するためには，読む人間に情報を与えないようにプログラムを低レベル命令で構成すべきであると主張している．一

```
  leal  28(%esp), %eax
  addl  44(%esp), %eax
  movzbl  (%eax), %eax
  movzbl  %al, %eax
  movl  %eax, 4(%esp)
  movl  $LC2, (%esp)
  call  _printf
  addl  $1, 44(%esp)
L2:
  cmpl  $15, 44(%esp)
```

```
  leal  28(%esp), %eax
  addl  44(%esp), %eax
  movzbl  (%eax), %eax
  movzbl  %al, %eax
  jmp L10
  movl  %eax, 4(%esp)
  movl  $LC2, (%esp)
  call  _printf
  addl  $1, 44(%esp)
L10:
  movl  %eax, 4(%esp)
  movl  $LC2, (%esp)
  call  _printf
  addl  $1, 44(%esp)
L2:
  cmpl  $15, 44(%esp)
```

図 2 O_{3a} **の例**

```
  addl  44(%esp), %eax
  movzbl  (%eax), %eax
  movzbl  %al, %eax
  movl  %eax, 4(%esp)
  movl  $LC2, (%esp)
```

```
  addl  44(%esp), %eax
  movzbl  (%eax), %eax
  movzbl  %al, %eax
  movl  %eax, 4(%esp)
  movl  $LC2, (%esp)
  jmp L10
  addl  44(%esp), %eax
L10:
```

図 3 O_{3b} **の例**

```
  leal  3(%eax), %ecx
  testl %eax, %eax
  cmovs %ecx, %eax
  sarl  $2, %eax
```

```
  movl  %eax, %ecx
  addl  $3, %ecx
  testl %eax, %eax
  cmovs %ecx, %eax
  sarl  $2, %eax
```

図 4 O_4 **の例**

般に，プログラム中の高レベル命令は出現頻度が低く，低レベル命令は出現頻度が高い．出現頻度の低い命令（高レベル命令）を，出現頻度の高い命令（低レベル命令）に書き換えることで，命令分布を一様にし，プログラムから得られる情報を小さくする狙いがある．

3.4 結果と考察

エントロピーおよびコルモゴロフ複雑性を計測した結果を表 2 に示す．

相対コルモゴロフ複雑性 $k(p)$ に着目すると，$O_{3b}(p)$ を除いて 0.1 未満と非常に小さな値となった．この理由は，難読化前の AES プログラムが極めて分かりやすく書かれたサンプル実装であり，AES の暗号化・復号アルゴリズムにおける類似の処理をあえて共通化せずに実装されているため，随所に規則性が見られ，圧縮性が高くなったためと考えられる．

プログラム $O_1(p)$ では，コルモゴロフ複雑性 $K(p)$ が 509 から 533 へと増大し，相

表 2　評価結果

	$h(S)$	$K(p)$	$k(p)$
p (難読化前)	0.74	509	0.076
$O_1(p)$	0.74	533	0.078
$O_2(p)$	0.72	531	0.077
$O_{3a}(p)$	0.74	530	0.077
$O_{3b}(p)$	0.76	1022	0.118
O_4	0.69	487	0.072

対コルモゴロフ複雑性 $k(p)$ もわずかに増加した（0.076 から 0.078）．また，エントロピーはほとんど変化しないという結果となった．難読化前のプログラムでは，類似するループが多数存在しており，規則性が見られたが，ループを難読化した結果，規則性が減少するとともに，難読化にともなってプログラムサイズが増大したことから，$K(p)$ が増大したと考えられる．ただし，プログラムには依然として類似の処理が多いことから，$k(p)$ は低い値にとどまっている．

プログラム $O_2(p)$ も $O_1(p)$ とほぼ同様の結果となった．関数を分割・マージすることで，プログラムサイズが増大するとともに，プログラムに含まれる規則性が減ったことで $K(p)$ が増大した．プログラム $O_{3a}(p)$ も同様の結果であった．

プログラム $O_{3b}(p)$ では，$K(p)$ が 509 から 1022 と 2 倍以上増大し，$k(p)$ も 0.076 から 0.11 へと増大した．O_{3b} はパターンを消す難読化であるため，圧縮効率が大きく低下したことが原因であると考えられる．

O_{3b} は，プログラム中の命令順序のパターンをどの程度消去するかによって，結果が異なる．そこで，パターン消去の度合い別に測定した結果を図 5 に示す． 命令

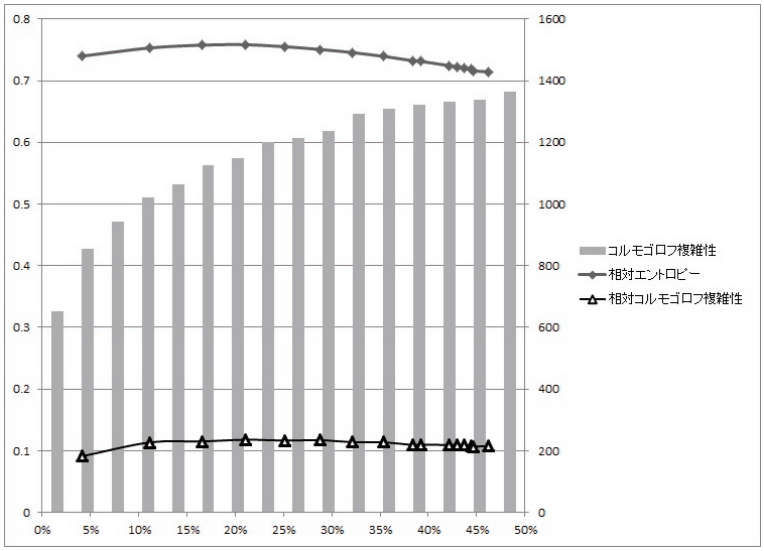

図 5　評価結果

順序のパターンを消去するほど，コルモゴロフ複雑性が増大している．一方，相対エントロピーおよび相対コルモゴロフ複雑性については，21%を境に値が減少傾向にある．

プログラム $O_4(p)$ では，$h(S)$，$K(p)$，$k(p)$ がすべて減少するという結果となった．$h(S)$ が小さくなった理由は，高レベル命令を低レベル命令の系列に置き換えた

ため，命令の出現頻度の分布に偏りが生じたためと考えらえる．また，$K(p)$ と $k(p)$ が減少した理由は，置き換え規則が単純であったために，同じ命令系列が頻出することになったことと，そもそも命令の種類数が減ったことで，命令の系列にパターンが生じやすくなり，圧縮性が高まったためと考えられる．

O_4 については，難読化の実装方法にそもそもの問題があったといえるが，特に，相対コルモゴロフ複雑性 $k(p)$ については，命令の種類数の影響を受けるため，今後，命令の種類数を考慮した尺度へと改良する必要があると考えられる．そこで，$k(p)$ を命令の種類数 n で正規化した値は，表 3 のようになった．

表 3　評価結果

	$k(p)/r$
p (難読化前)	0.0024
$O_1(p)$	0.0024
$O_2(p)$	0.0024
$O_{3a}(p)$	0.0025
$O_{3b}(p)$	0.0036
$O_4(p)$	0.0027

$O_4(p)$ は難読化前と比較して $h(S)$，$K(p)$，$k(p)$ が減少していたが，$k(p)/r$ は増加している．$k(p)/r$ のみに着目すると，O_4 によってプログラムが読みづらくなったといえる．$O_{3b}(p)$ については，やはり他のプログラムと比較して値が増加している．$K(p)$,$k(p)$,$k(p)/r$ は，プログラムの理解性の評価に役立ちそうである．また，エントロピー $h(S)$ は，現時点では役に立つ尺度とはいえないことが分かった．

4　ケーススタディ２：異なる開発者によるプログラムの評価

4.1　目的

本手法は，難読化されていないプログラムの理解性の評価にも利用できる可能性がある．そこで，同じ仕様で異なる 4 人の人物が作成したプログラムを対象に，本手法を用いて理解性を評価する．

4.2　題材

プログラムの仕様は，与えられたテキスト入力をハフマン符号により符号化するものである．4 名の実装者は，奈良先端科学技術大学院大学の 1 名の大学教員と 3 名の社会人学生（いずれもソフトウェア開発経験がある）である．

アセンブリプログラムの特徴を表 4 に示す．

表 4　アセンブリプログラムの特徴

プログラム	命令数 n	命令の種類数 r	使用している技術	主観評価
p_a	372	38	構造体	理解しやすい
p_b	322	29	構造体，ポインタ，木構造	理解しにくい
p_c	356	27	構造体，ポインタ，木構造	理解しにくい
p_d	478	26	構造体，木構造	理解しやすい

主観的には，p_a,p_d がシンプルな実装となっており，理解しやすい．p_b と p_c はポインタを多用しているため p_a,p_d よりも理解しにくい．なお，この主観評価は，ソースプログラムに対するものである．

4.3 結果と考察

エントロピーおよびコルモゴロフ複雑性を評価した結果を表 5 に示す.

表 5 評価結果

	$h(S)$	$K(p)$	$k(p)$	$k(p)/r$
p_a	0.74	242	0.12	0.0033
p_b	0.66	214	0.14	0.0047
p_c	0.61	214	0.13	0.0047
p_d	0.61	219	0.097	0.0037

相対コルモゴロフ複雑性 $k(p)$ に着目すると，p_d が理解しやすいという主観評価に合致する．また，コルモゴロフ複雑性 $K(p)$ に着目すると p_a の値が最も大きくなっている．p_a は，命令の種類数が多いことが原因であると考えられる．$k(p)/r$ に着目すると，p_a と p_d が理解しやすいという主観評価に合致する．

$k(p)/r$ については，ケーススタディ 1 と同様，プログラムの理解性の評価に役立ちそうである．一方，エントロピー $h(S)$ も同様に，現時点では役に立つ尺度とはいえないようである．

5 関連研究

Kirk ら [5] は，本論文と同様，コルモゴロフ複雑性をソフトウェアメトリクスとして提案し，デザインメトリクスの代替となり得るかを検討している．ただし，Kirk らは，Java クラスファイル全体を圧縮して得られるビット数をコルモゴロフ複雑性として用いており，クラスファイル中の多様な要素（クラスヘッダ，コンスタントプール，フィールド情報など）のいずれがコルモゴロフ複雑性に寄与しているかは不明確であった．また，相対コルモゴロフ複雑性に該当するメトリクスは提案しておらず，命令のランダム性を評価していない．本論文は，プログラム中からオペコードの系列だけを取り出して，各オペコードが対等になるように符号化してから（相対）コルモゴロフ複雑性を求めることで，命令のランダム性を評価している点が異なる．

Buse ら [16] は，コードの可読性についての論文である．Buse らは，プログラム中の変数名の長さやインデント，記号など複数の要因に着目して可読性を評価している．我々は，プログラムのオペコードに絞って評価しているという点が異なる．

6 終わりに

本論文では，エントロピーとコルモゴロフ複雑性に基づいてプログラムの理解性を定量化する方法を提案した．ケーススタディの結果，コルモゴロフ複雑性 $K(p)$，相対コルモゴロフ複雑性 $k(p)$，および，$k(p)$ を命令の種類数 r で正規化した $k(p)/r$ は，プログラム理解性の評価に役立つことが示唆された．一方，記憶のない情報源を仮定した相対エントロピー $h(S)$ は，あまり有用でないことが分かった．

本論文の制約は次の通りである．まず，本論文では，オペコードを対象とし，その系列の理解性のみを評価している．オペランドも含めた理解性評価や，名前（関数名や変数名など），コメント文などの理解性評価は対象外である．また，本論文では，他の理解性尺度との比較を行っていない．その理由は，本論文ではオペコードの理解性のみを評価しているため，他の尺度との直接的な比較が困難なためである．今後，オペコード以外の要素についても評価方法を検討し，他の尺度との比較を行っていく予定である．また，本論文では，一部の難読化法のみを用いた評価となっており，他の難読化法を用いた評価も今後必要となる．

今後の課題は次の通りである．まず，エントロピーの算出に，エルゴードマルコ

フ情報源を仮定した方法を採用することである．また，命令分布や系列が，一般的なプログラムとどれだけ異なっているか，という観点で評価する方法についても検討する予定である．さらに，プログラムの実行履歴を対象に評価を行うことも検討している．

参考文献

[1] Rudi Cilibrasi and Paul M. B. Vitányi, “Similarity of objects and meaning of the words,” 3rd Int’l Conf. on Theory and Applications of Models of Computation, Lecture Notes in Computer Science, Vol. 3959, pp.21-45,2006.

[2] Christian Colberg, Clark Thomborson, Douglas Low, “A taxonomy of obfuscation transformations”, Technical report 148, Department of Computer Science, the University of Aukland, Aukland, New Zealand,1997.

[3] Christian Collberg, Jasvir Nagra, “Surreptitious software,” Addison-Wesley Professional, Aug,2009.

[4] 神崎雄一郎，門田暁人, “ソフトウェア保護機構を構成するコードの特徴評価の試み,” コンピュータセキュリティシンポジウム 2011(CSS2011) 予稿集, pp.827-832, October,2011.

[5] S. R. Kirk and S. Jenkins, “Information theory-based software metrics and obfuscation”, Journal of Systems and Software, Vol. 72, Issue 2, pp. 179-186, July,2004.

[6] 松岡賢, 赤井健一郎, 松本勉, “鍵内蔵型暗号ソフトウェアの人手による耐タンパー性評価”, No. 6C-2, 2002 年暗号と情報セキュリティシンポジウム（SCIS2002）, 2002.

[7] Peter Miltersen, ”Course notes for Data Compression - 2 Kolmogorov complexity Fall 2005”, University of Aarhus, Sep. 2005. http://www.daimi.au.dk/ bromille/DC05/Kolmogorov.pdf

[8] 宮川洋, “情報理論,” コロナ社, 1979.

[9] 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.644–652, July,1997.

[10] Masahide Nakamura, Akito Monden, Tomoaki Itoh, Ken-ichi Matsumoto, Yuichiro Kanzaki, and Hirotsugu Satoh, “Queue-based cost evaluation of mental simulation process in program comprehension”, Proc. 9th IEEE International Software Metrics Symposium (METRICS2003), pp. 351-360, Sydney, Australia, Sep, 2003.

[11] National Institute of Standards and Technology (NIST), “Advanced Encryption Standards (AES)”, Federal Information Processing Standards Publication 197, Nov,2001.

[12] T. Ogiso, Y. Sakabe, M. Soshi, and A. Miyaji, “Software obfuscation on a theoretical basis and its implementation”, IEICE Trans., Fundamentals, vol. E86-A, No. 1,pp.176-186,2003.

[13] 山下洋史,“エントロピー・モデルにおけるエントロピーの役割”, 明大商学論叢”,Vol.84,No.2,pp.71-88,2002.

[14] H. Yamauchi, A. Monden, M. Nakamura, H. Tamada, Y. Kanzaki, and K. Matsumoto, “A goal-oriented approach to software obfuscation”, Int’l J. of Computer Science and Network Security, Vol.8, No.9, pp.59-71, Sep,2008.

[15] 村山隆徳, 満保雅浩, 岡本栄司, 植松友彦,“プログラムコードの難読化について”,SCIS96-8D,pp.1-6,1996.

[16] R.Buse and W.Weimer,“Learning a Metric for Code Readability”,Software Engineering,IEEE Transactionson,36(4),pp.546-558,2010.