社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

構文誤りを含むプログラムを評価可能なソースコード用自動評価尺度の調査

髙市　陸† 肥後　芳樹† 柗本　真佑† 楠本　真二†
倉林　利行†† 切貫　弘之†† 丹野　治門††

† 大阪大学大学院情報科学研究科
†† 日本電信電話株式会社

あらまし　近年，コード翻訳や自動プログラム生成等のソースコードを出力する手法の研究が盛んに行われている．それらの手法は発展途上で，構文的に間違っているプログラムが生成されることも少なくない．そのため，既存手法の評価には，自然言語処理で用いられる自動評価尺度が使われることがある．しかし，それらの評価尺度のうち，どれが自動生成されたプログラムの評価に適しているかは不明である．そこで本研究では，構文的に正しくないプログラムに対しても適用可能で，自然言語で記述された要求からプログラムを生成する手法の評価に適した自動評価尺度を明らかにする．

キーワード　自動評価尺度，自動プログラム生成，深層学習

1. まえがき

近年，コード翻訳や自動プログラム生成（Automated Program Generation: APG）等のソースコードを生成する手法の研究が盛んに行われている [1] [2] [3] [4] [5]．これらの研究では，生成されたコードの評価に自動評価尺度が用いられている．

ソースコードのための自動評価尺度はいくつか提案されている [6] [7]．それらの評価尺度は，コードが構文的に正しいことを前提として抽象構文木やプログラム依存グラフを用いている．しかし，コードを生成する手法の研究は発展途上で，構文的に間違っているプログラムが生成されることも少なくない．例えば，近年提案された自動プログラム生成モデルである SNM [4] では 7.0 %，Coarse-to-Fine [5] では 90 % の出力プログラムが構文的に正しくなかったという報告がある [8]．そのため，コードが構文的に正しいことを前提としているソースコードのための自動評価尺度は使えないことがある．

コードを生成する手法の評価には，ソースコード用の自動評価尺度ではなく，自然言語処理で用いられる自動評価尺度が使われている場合もある [9] [10]．特に，BLEU は生成されたソースコードの品質評価に頻繁に用いられているが，ソースコードを評価するには限界があるといわれている [6] [11]．生成コードの評価に用いられておりコードの構文的正しさを前提としない評価尺度のうち，どれが自動生成されたプログラムの評価に適しているかは不明である．

そこで本研究では，構文的に正しくないコードに対しても適用可能なコードの評価に適した自動評価尺度を明らかにする．特に自動プログラム生成タスクに着目し，自然言語で記述された要求から自動的に生成されたコードの評価に適した自動評価尺度を明らかにする．

現状，自然言語で記述された要求を入力とする自動プログラム生成では，要求を満たすコードの生成は困難である [8]．自動プログラム生成を利用する際，生成されたコードを人間が修正することにより，要求を満たすコードにする必要がある．そのため，生成されたコードは要求を満たすコードへの修正が容易であることが望ましい．自動評価尺度が生成されたコードの評価に適しているかどうかは，生成されたコードから要求を満たすコードへの修正の容易さによって判断できる．

生成されたコードから要求を満たすコードへの修正容易性を測る被験者実験を行った．その結果，生成コードの修正容易性と相関が強く，自動プログラム生成によって生成されたコードの評価に適した自動評価尺度は METEOR [12] であることが明らかになった．

2. Research Question

自動プログラム生成によって生成されたコードの評価に適した自動評価尺度を明らかにするために，以下 2 つの Research Question（RQ）を設定した．

RQ1: 修正時間を用いて測定する生成コードの修正容易性を適切に評価できる自動評価尺度はどれか？

生成コードの修正容易性は，開発者による修正に要した時間で評価できる．生成コードの修正容易性を修正時間で評価するとき，修正容易性を適切に評価できる自動評価尺度を調査する．調査では，修正時間と自動評価尺度の評価値の相関をみる．対象となる自動評価尺度のうち最も相関が強い尺度が修正容易性を適切に評価できるとみなす．

RQ2: 修正量を用いて測定する生成コードの修正容易性を適切に評価できる自動評価尺度はどれか？

生成コードの修正容易性は，開発者による修正前後のコードの修正量によっても評価できる．生成コードの修正容易性を修正量で評価するとき，修正容易性を適切に評価できる自動評価尺

度を調査する．調査では RQ1 と同様に，修正量と自動評価尺度の評価値の相関をみる．対象となる自動評価尺度のうち最も相関が強い尺度が修正容易性を適切に評価できるとみなす．

3. 準　　備

3.1 自動プログラム生成（APG）

自動プログラム生成とは，プログラムを全自動で生成する技術である．自動プログラム生成手法は，入力として与える要求の形式とプログラム生成手段の 2 つの組み合わせで分類できる．入力の形式には自然言語の文章 [8]，DSL [13]，入出力例 [14] 等がある．手段には，翻訳ベース [8]，探索ベース [15] 等がある．本研究では特に，自然言語の文章で書かれた要求を入力とする翻訳ベースの自動プログラム生成手法を想定する．

3.2 編 集 距 離

編集距離 EditDistance [16] は，ある列 X を別の列 Y と同じ列にするための編集操作の最小回数である．編集操作には要素の追加，削除，置換の 3 種類がある．

0 以上 1 以下の値に正規化された編集距離の求め方を (1) 式に示す．なお，列 S の長さを $\mathrm{length}(S)$ で表している．

$$\mathrm{NormarizedEditDistance} = \frac{\mathrm{EditDistance}\left(X, Y\right)}{\max\left(\mathrm{length}\left(X\right), \mathrm{length}\left(Y\right)\right)} \tag{1}$$

3.3 自動評価尺度

自動評価尺度は，翻訳結果の品質を自動的に評価するための尺度である．翻訳結果の手動評価は高コスト [17] なため，低コストで評価可能な自動評価尺度が提案されている．自動評価尺度によって手動評価を代替するという目的を考えると，自動評価尺度による評価と手動評価の相関は高いほどよい．

以下では，本研究で用いた自動評価尺度について説明する．いずれも，評価値は 0 以上 1 以下の数値で表現され，1 に近いほど高い評価を意味する．また，説明では生成コードのトークン列を pre，正解コードのトークン列を ref，正解コードのトークン列の集合を refs として用いる．

BLEU

BLEU（BiLingual Evaluation Understudy）[17] は，自然言語の機械翻訳結果の評価のために考えられた自動評価尺度である．BLEU は，pre と ref の n-gram の精度 p_n と重み w_n，生成コードの長さに対するペナルティ BP を用いて (2) 式で表される．

$$\begin{aligned}\mathrm{BP} &= \min\left(1, e^{1-\mathrm{length(ref)}/\,\mathrm{length(pre)}}\right)\\ \mathrm{BLEU}(\mathrm{pre}, \mathrm{ref}) &= \mathrm{BP}\cdot\exp\left(\sum_{n=1}^{N} w_n \log p_n\right)\end{aligned} \tag{2}$$

本研究では，$N = 4$，$w_n = 1/N$ とする．これは，BLEU を提案した研究 [17] で用いられた設定である．例えば，生成コードが図 1(a) で正解コードが図 1(b) のときの BLEU は次のように計算する．まず，pre に対する ref の n-gram の精度を考える．$n = 4$ のとき，pre の 4-gram は 8 個あり，そのうち ref にも存在する 4-gram は “`a [ i ]`” のみで 1 個なので $p_4 = 1/8$ である．同様に $p_1 = 11/11$，$p_2 = 7/10$，$p_3 = 3/9$ となる．また，$\mathrm{length(ref)} = 13$，$\mathrm{length(pre)} = 11$ より

```
S += b [ j ] + a [ i ]
```

(a) 生成コード pre

```
S += a [ i ] + b [ j + 1 ]
```

(b) 正解コード ref

図 1: 自動評価尺度の計算手順説明用コード

$$\begin{aligned}\mathrm{BP} &= e^{-2/11}\\ \mathrm{BLEU}(\mathrm{pre}, \mathrm{ref}) &= \mathrm{BP}\cdot\exp\left(\sum_{n=1}^{4}\frac{\log p_n}{N}\right)\\ &\simeq 0.345\end{aligned}$$

となる．

正解コードが複数ある場合の BLEU は，(3) 式で算出される．

$$\mathrm{BLEU}(\mathrm{pref}, \mathrm{refs}) = \max_{\mathrm{ref}\in\mathrm{refs}} \mathrm{BLEU}(\mathrm{pref}, \mathrm{ref}) \tag{3}$$

STS

STS（STring Similarity）[6] は，編集距離を用いた自動評価尺度で (4) 式で表される．

$$\mathrm{STS}(\mathrm{pre}, \mathrm{ref}) = 1 - \mathrm{NormarizedEditDistance}(\mathrm{pre}, \mathrm{ref}) \tag{4}$$

例えば，生成コードが図 1(a) で正解コードが図 1(b) のときの STS は次のように計算する．まず，pre に対する ref の編集距離を考える．pre の左から `b` を `a` に，`j` を `i` に，`a` を `b` に，`i` を `j` にそれぞれ置換し，`+` と `1` を挿入すれば ref になる．よって，編集距離 $\mathrm{EditDistance}(\mathrm{pre}, \mathrm{ref}) = 6$ となる．また，$\mathrm{length(ref)} = 13$，$\mathrm{length(pre)} = 11$ なので

$$\begin{aligned}\mathrm{STS}(\mathrm{pre}, \mathrm{ref}) &= 1 - 6/\max(13, 11)\\ &\simeq 0.538\end{aligned}$$

となる．

正解コードが複数ある場合の STS は，(5) 式で算出される．

$$\mathrm{STS}(\mathrm{pref}, \mathrm{refs}) = \max_{\mathrm{ref}\in\mathrm{refs}} \mathrm{STS}(\mathrm{pref}, \mathrm{ref}) \tag{5}$$

ROUGE-L

ROUGE（Recall-Oriented Understudy for Gisting Evaluation）[18] は，自然言語要約の品質評価のために考えられた自動評価尺度である．ROUGE-L [18] は ROUGE の一種で，2 つの列における最長共通部分列（Longest Common Subsequence: LCS）の長さを用いて計算される．ROUGE-L は，重み β を用いて (6) 式で表される．

$$\begin{aligned}R(\mathrm{pre}, \mathrm{ref}) &= \frac{\mathrm{LCS}(\mathrm{pre}, \mathrm{ref})}{\mathrm{length(ref)}}\\ P(\mathrm{pre}, \mathrm{ref}) &= \frac{\mathrm{LCS}(\mathrm{pre}, \mathrm{ref})}{\mathrm{length(pre)}}\\ \mathrm{ROUGE\text{-}L}(\mathrm{pre}, \mathrm{ref}) &= \frac{(1+\beta^2)RP}{R+\beta^2 P}\end{aligned} \tag{6}$$

本研究では，R と P を同じ重要度として扱うために $\beta = 1$ とす

る．例えば，生成コードが図 1(a) で正解コードが図 1(b) のときの ROUGE-L は次のように計算する．まず，pre に対する ref の LCS を考える．LCS は “`s += [ ] + [ ]`” なので，$\mathrm{LCS}(\mathrm{pre}, \mathrm{ref}) = 7$ となる．また，$\mathrm{length}(\mathrm{ref}) = 13$，$\mathrm{length}(\mathrm{pre}) = 11$ なので

$$
\begin{aligned}
R(\mathrm{pre}, \mathrm{ref}) &= 7/13 \\
P(\mathrm{pre}, \mathrm{ref}) &= 7/11 \\
\text{ROUGE-L}(\mathrm{pre}, \mathrm{ref}) &= \frac{2 \cdot 7/13 \cdot 7/11}{7/13 + 7/11} \\
&\simeq 0.614
\end{aligned}
$$

となる．

正解コードが複数ある場合の ROUGE-L は，(7) 式で算出される．

$$
\begin{aligned}
R(\mathrm{pref}, \mathrm{refs}) &= \frac{\sum_{\mathrm{ref} \in \mathrm{refs}} \mathrm{LCS}(\mathrm{pref}, \mathrm{ref})}{\sum_{\mathrm{ref} \in \mathrm{refs}} \mathrm{length}(\mathrm{ref})} \\
P(\mathrm{pref}, \mathrm{refs}) &= \frac{\sum_{\mathrm{ref} \in \mathrm{refs}} \mathrm{LCS}(\mathrm{pref}, \mathrm{ref})}{\mathrm{length}(\mathrm{pref})} \\
\text{ROUGE-L}(\mathrm{pref}, \mathrm{refs}) &= \frac{(1+\beta^2) R(\mathrm{pref}, \mathrm{refs}) P(\mathrm{pref}, \mathrm{refs})}{R(\mathrm{pref}, \mathrm{refs}) + \beta^2 P(\mathrm{pref}, \mathrm{refs})}
\end{aligned}
\tag{7}
$$

METEOR

METEOR は，自然言語の機械翻訳結果の評価のために考えられた自動評価尺度で，BLEU の改良版として設計されている [19]．METEOR では，トークンの一致度を以下のように分類する．

（1）完全一致
（2）語幹が一致
（3）同義語
（4）フレーズが言い換え可能

ある文 A に含まれる単語 a と別の文 B に含まれる単語 b が一致するとき，一致度がどれに分類されるか判定する関数を $M(a, b)$ とする．このとき，A に含まれる単語であって B にも含まれる単語との一致度が（1）から（4）に分類される単語数をそれぞれ $m_1(A, B)$ から $m_4(A, B)$ で表す．特に，B が自明な場合は $m_i(A)$ と書く．また，A と B で同一順序で出現し，連続して一致する単語の列をチャンクという．チャンクの数を $ch(A, B)$ で表す．METEOR は，生成されたテキスト内の内容語 h_c と 機能語 h_f，参照テキスト内の内容語 r_c と機能語 r_f，生成されたテキストと参照テキストそれぞれの一致する単語数の平均 m および単語の一致度ごとの重み w_i を用いて (8) 式で表される．

$$
\begin{aligned}
P &= \frac{\sum_i w_i \cdot (\delta \cdot m_i(h_c) + (1-\delta) \cdot m_i(h_f))}{\delta \cdot |h_c| + (1-\delta) \cdot |h_f|} \\
R &= \frac{\sum_i w_i \cdot (\delta \cdot m_i(r_c) + (1-\delta) \cdot m_i(r_f))}{\delta \cdot |r_c| + (1-\delta) \cdot |r_f|} \\
F_{\mathrm{mean}} &= \frac{P \cdot R}{\alpha \cdot P + (1-\alpha) \cdot R} \\
\mathrm{Pen} &= \gamma \cdot \left(\frac{ch}{m}\right)^{\beta} \\
\mathrm{METEOR} &= (1 - \mathrm{Pen}) \cdot F_{\mathrm{mean}}
\end{aligned}
\tag{8}
$$

α，β，γ，δ および w_i はパラメータである．本研究では，ソースコードが英単語を用いて書かれる場合が多い点に着目して，生成コードと正解コードをそれぞれ生成されたテキストと参照テキストと見なして評価した．評価では，METEOR の実装である `meteor-1.5.jar` [12] に `-lang en` オプションを渡して言語設定を英語にし，コードを英語のテキストとみなした．各パラメータは，言語設定を英語にしたときのデフォルトの値である．

正解コードが複数ある場合の METEOR は，(9) 式で算出される．

$$
\mathrm{METEOR}(\mathrm{pref}, \mathrm{refs}) = \max_{\mathrm{ref} \in \mathrm{refs}} \mathrm{METEOR}(\mathrm{pref}, \mathrm{ref}) \tag{9}
$$

4. 実　　験

被験者実験によって，自動プログラム生成モデルの生成したコードを要求を満たすコードに修正する修正容易性を測定する．修正容易性は，修正時間と修正量で測る．修正量は，生成コードと修正済コードの正規化された編集距離によって求める．測定した修正容易性と自動評価尺度による生成コードの評価の相関をとる．修正時間と修正量は高いほど修正容易性が低いので，負の相関が強いほど，その自動評価尺度は自然言語で記述された要求からプログラムを生成する手法の評価に適しているといえる．負の相関が強いということは，修正容易性が高いほど評価が高くなる傾向が強いからである．

4.1 自動プログラム生成モデルの作成

自動プログラム生成モデルの作成のために，GitHub 上で公開されている自然言語翻訳用のネットワーク(注1)を利用した．

学習には，プログラミングコンテストの問題文と正解コード，テストケースの組からなるデータセット ReCa [8] を用いた．データセットには，5,149 問の問題文と 16,673 個の Python コードが含まれている．300 問をテスト用，200 問を検証用，残りを訓練用として学習を行い自動プログラム生成モデルを作成した．

作成したモデルの入出力例を図 2(b)(c)(d) に示す．入力は，図 2(a) のような英語の問題文に対して小文字化，レンマ化，ストップワード除去などの前処理を施した図 2(b) のような文である．出力は，図 2(c) のような Python コードのトークン列である．図 2(c) のようなトークン列は，適当な処理によって自動的に図 2(d) のような Python コードに変換できる．以下では，図 2(d) のような自動プログラム生成モデルの出力から得られるコードを生成コードと呼ぶ．図 2(d) は要求を満たさないコードである．例えば，図 2(g) のようなテストケースの入力が与えられた場合に，図 2(d) は入力の受け取りに失敗する．入力の 1 行目には整数が 1 つしかないが，図 2(d) の 1 行目では `n` と `k` の 2 つの整数を受け取ろうとしているからである．問題の正答コードのひとつであり，要求を満たす図 2(e) では，図 2(g) の入力を与えると期待される出力と同じ出力が得られる．

4.2 対　　象

被験者実験には，テスト用データ 300 問からランダムにサン

(注1)：`https://github.com/nazim1021/neural-machine-translation-using-gan`

```
Suppose we have a sequence of non-negative integers, Namely a_1, a_2, ... ,a_n. At each time we can choose one term a_i with 0 < i < n and we subtract 1 from both a_i and a_i+1. We wonder whether we can get a sequence of all zeros after several operations.
```

(a) 問題文

```
suppose we have sequence of non-negative integer, namely a_1, a_2,..., a_n. at each time we can choose one term a_i with 0 < i < n and we subtract 1 from both a_i and a_i+1. we wonder whether we can get sequence of all zero after several operation.
```

(b) 前処理された問題文

```
n , k = map ( int , input ( ) . split ( ) ) &n a = list ( map ( int , input ( ) . split ( ) ) ) &n a . sort ( ) &n ans = 0 &n for i in range ( k ) : &n & indent if a [ i ] == a [ i - 1 ] : &n &indent &indent ans += 1 &n &indent &dedent print ( ans ) &n
```

(c) 自動プログラム生成モデルの出力

```
n , k = map ( int , input ( ) . split ( ) )
a = list ( map ( int , input ( ) . split ( ) ) )
a . sort ( )
ans = 0
for i in range ( k ) :
    if a [ i ] == a [ i - 1 ] :
        ans += 1
print ( ans )
```

(d) 生成コード

```
Len = int(input())
List = list(map(int, input().split()))
assume = True
res = 0
for i in range(Len):
    res = List[i] - res
    if res < 0:
        assume = False
        break
if assume and res == 0:
    print("YES")
else:
    print("NO")
```

(e) 正解コード

```
n = int ( input ( ) )
a = list ( map ( int , input ( ) . split ( ) ) )

for i in range ( n-1 ) :
    a[i+1]-=a[i]
    a[i]=0


if(a[-1]==0):print("YES")
else:print("NO")
```

(f) 修正済コード

入力

```
2
2 2
```

期待される出力

```
YES
```

(g) テストケース

図 2: 実験データの例

プリングした 10 問を用いた．サンプリングされた問題には，1 問あたり平均 53.4 個のテストケースがある．

被験者は，大阪大学大学院情報科学研究科に所属する教員 1 名，同研究科に所属する修士の学生 8 名，大阪大学基礎工学部情報科学科に所属する学部生 2 名の計 11 名である．被験者ごとに Python の習熟度は異なる．習熟していない被験者のために，コードの修正時に必要と思われる処理のチートシートを配布した．

4.3 手　　順

被験者実験は，問題ごとに以下の手順で行った．

STEP-1 （要求の理解）被験者には図 2(a) のような問題文と図 2(g) のようなテストケースが与えられる．被験者は問題文を読み，要求を理解する．

STEP-2 （生成コードの修正）被験者には図 2(d) のような生成コードが与えられる．被験者は STEP-1 で与えられた問題の解答コードとして正しいコードになるように生成コードを修正する．すなわち，要求を満たすように生成コードを修正する．

STEP-3 （テスト）STEP-2 で修正したコードが STEP-1 で与えられたすべてのテストケースに通過するか確かめる．通過すれば STEP-3 は終了する．通過しなければ STEP-2 に戻り生成コードをさらに修正する．

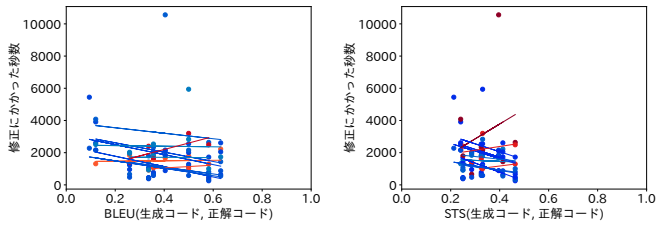
生成コードを修正するまでに要した時間として，STEP-2 から STEP-3 終了までの秒数を測定した．以上の手順で図 2(f) のように修正された生成コードを，修正済コードと呼ぶ．STEP-2 や STEP-3 において生成コードを修正できなかった場合は，修正済コードおよび修正時間は得られない．そのため，得られる修正済コードと修正時間の数は被験者ごとに異なる．

4.4 結果と考察

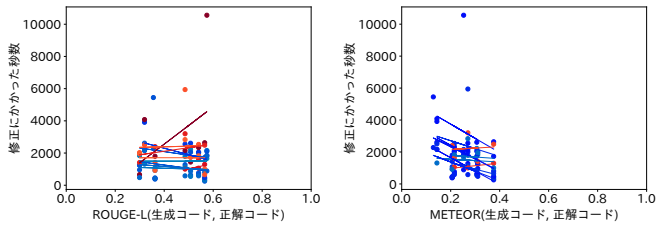
実験結果から，2 つの RQ に回答する．

RQ1: 修正時間を用いて測定する生成コードの修正容易性を適切に評価できる自動評価尺度はどれか？

表 1 に各自動評価尺度の評価値と修正時間のピアソンの相関係数，有意性検定の結果である p 値を示す．表 1 から，評価値と修正時間の相関は METEOR が最も強いことがわかる．また，



(a) BLEU (b) STS



(c) ROUGE-L (d) METEOR

図 3: 各メトリクスによる生成コードと正解コードの類似度に対する生成コード修正時間の分布

METEOR に関しては，有意水準 5 % で有意に相関があることもわかる．

図 3 に，各自動評価尺度による生成コードと正解コード間の類似度に対する生成コード修正時間の分布を示す．図中の線分は各被験者の分布に対する回帰直線を表している．直線と分布の点の色は，回帰直線の傾きが正の場合は暖色，負の場合は寒色で，傾きが強いほど赤または青が濃くなっている．図 3 をみると，ほとんどの被験者は評価値が高いほど修正時間が短くなっている．一方で，一部の被験者に関しては評価値が高いほど修正時間が長くなっている．そのような被験者による修正時間と各自動評価尺度における評価値の相関は統計的に有意とは言えない．

RQ1 への回答： 生成コードの修正時間による評価の代替として優れている自動評価尺度は METEOR である．また，生成コードの評価によく用いられている BLEU は修正時間による評価の代替として優れているとは言えない．

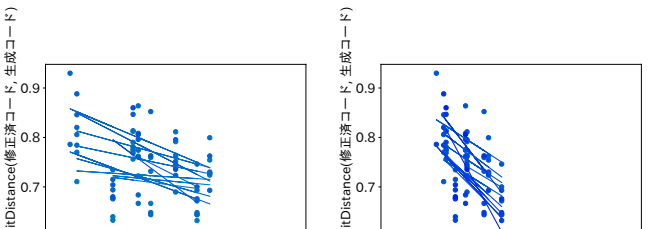
RQ2: 修正量を用いて測定する生成コードの修正容易性を適切に評価できる自動評価尺度はどれか？

表 2 に各自動評価尺度の評価値と修正量の相関係数，有意性検定の結果である p 値を示す．表 2 から，評価値と修正量の相関は METEOR が最も強いことがわかる．また，全ての自動評価尺度において，有意水準 5 % で有意に相関があることもわかる．

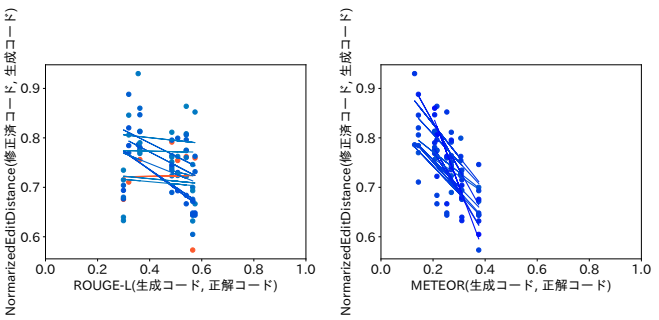
図 4 に，各自動評価尺度による生成コードと正解コード間の類似度に対する生成コード修正量の分布を示す．図 4 をみると，ROUGE-L 以外の自動評価尺度では，すべての被験者において

表 1: 各自動評価尺度の評価値と修正時間の相関

自動評価尺度	相関係数	p 値
BLEU	−0.181	0.117
STS	−0.100	0.389
ROUGE-L	0.011, 5	0.921
METEOR	−0.251	0.028, 6



(a) BLEU (b) STS



(c) ROUGE-L (d) METEOR

図 4: 各メトリクスによる生成コードと正解コードの類似度に対する生成コードと修正コードの編集距離の分布

評価値が高いほど修正量が少なくなっている．ROUGE-L で修正量と評価値の相関が低い原因は，ROUGE-L が正解コードの長さを考慮しすぎているからだと考えられる．ROUGE-L 以外の 3 つの評価尺度では，正解コードが複数あるとき，各正解コードを用いた場合の評価値の最大値を全体の評価値とすることで，1 つの正解コードの長さのみを考慮している．一方で ROUGE-L は，各正解コードの和を用いており，すべての正解コードの長さを考慮している．実際，正解コードの長さをあまり考慮しないように $\beta = 2$ とした場合，相関係数は −0.573 と改善する．これは，METEOR に次ぐ相関の強さである．反対に，$\beta = 0.5$ とした場合，相関係数は −0.267 と悪化する．

RQ2 への回答： 生成コードの修正量による評価の代替として優れている自動評価尺度は METEOR である．また，生成コードの評価によく用いられている BLEU は修正量による評価の代替として優れているとは言えない．

RQ1 および RQ2 の結果を踏まえると，自然言語の文章で記述された要求からプログラムを生成する手法の評価に適した自動評価尺度は METEOR であると言える．

表 2: 各自動評価尺度の評価値と修正量の相関

自動評価尺度	相関係数	p 値
BLEU	−0.392	4.64×10^{-4}
STS	−0.555	1.99×10^{-7}
ROUGE-L	−0.481	1.08×10^{-5}
METEOR	−0.696	3.03×10^{-12}

5. 妥当性の脅威

本研究では，自然言語用のネットワークから自動プログラム生成モデルを作成した．そのため，最新の自動プログラム生成モデルよりも性能が低く，正解コードと大きく異なるコードが生成されている可能性がある．生成コードの質が異なれば，異なる実験結果となるかもしれない．

被験者実験は，11 人の被験者と最大 10 問分のデータで実施された．統計的な分析をするには規模が小さすぎる可能性がある．より多くの被験者とデータ数で実験した場合，同様の結果が得られるとは限らない．

被験者実験で使用した問題が被験者に難しすぎた可能性もある．簡単な問題の正解コードは短く単純で，難しい問題の正解コードは長く複雑かもしれない．もし，実験に使用したモデルが短く単純なコードを生成しやすければ，自動評価尺度による評価は単に問題の難しさを表すだけとも考えられる．被験者にとって問題が難しすぎる場合，被験者は生成コードの理解と修正だけでなく，問題を解くためのアルゴリズムを考察する時間が余分にかかってしまう．その結果，簡単な問題の時は高評価かつ修正容易，難しい問題のときは低評価かつ修正困難となってしまう．考察の時間を考慮して被験者実験を行い修正容易性を測定した場合，実験結果が異なる可能性がある．

6. あとがき

本研究では，構文的に正しくないコードに対しても適用可能なコードの評価に適した自動評価尺度を明らかにすることを目的に，被験者実験によって得られた評価値と相関の強い自動評価尺度を調査した．調査の結果，自動プログラム生成による生成コードを要求を満たすコードへ修正するまでの時間と修正量の両方で比較的強い相関を示した自動評価尺度は METEOR であった．よって，METEOR がコードの評価に適した自動評価尺度であると言える．

今後の研究課題として，問題文だけを読んで一からコードを書いたときの時間と生成コードの修正時間との差分によって生成コードを評価した場合と相関の強い自動評価尺度の調査を考えている．このような手動評価尺度を用いることで，問題の難しさによる考察時間の影響を小さくした評価ができる可能性がある．また，コード生成に特化した最新の自動プログラム生成モデルの利用も検討している．生成コードの品質が向上すれば，問題の難しさによる考察時間の影響や Python への習熟度の影響を小さくできるかもしれないからである．

謝辞 本研究は JSPS 科研費（JP20H04166，JP21K18302，JP21K11820，JP21H04877，JP22H03567，JP22K11985）の助成を得て行われた．

文　献

[1] D. Guo, S. Ren, S. Lu, Z. Feng, D. Tang, S. LIU, L. Zhou, N. Duan, A. Svyatkovskiy, S. Fu, M. Tufano, S.K. Deng, C. Clement, D. Drain, N. Sundaresan, J. Yin, D. Jiang, and M. Zhou, “GraphCodeBERT: Pre-training code representations with data flow,” Proc. International Conference on Learning Representations, 2021.

[2] B. Roziere, M.A. Lachaux, L. Chanussot, and G. Lample, “Unsupervised translation of programming languages,” Proc. International Conference on Neural Information Processing Systems, 2020.

[3] W. Ahmad, S. Chakraborty, B. Ray, and K.W. Chang, “Unified pre-training for program understanding and generation,” Proc. Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, June 2021.

[4] P. Yin and G. Neubig, “A syntactic neural model for general-purpose code generation,” Proc. Annual Meeting of the Association for Computational Linguistics, 2017.

[5] L. Dong and M. Lapata, “Coarse-to-Fine decoding for neural semantic parsing,” Proc. Annual Meeting of the Association for Computational Linguistics, 2018.

[6] N. Tran, H. Tran, S. Nguyen, H. Nguyen, and T. Nguyen, “Does BLEU score work for code migration?,” Proc. 2019 IEEE/ACM International Conference on Program Comprehension, 2019.

[7] G. Zhao and J. Huang, “Deepsim: deep learning code functional similarity,” Proc. ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, 2018.

[8] H. Liu, M. Shen, J. Zhu, N. Niu, G. Li, and L. Zhang, “Deep learning based program generation from requirements text: Are we there yet?,” IEEE Transactions on Software Engineering, vol.48, no.4, pp.1268–1289, 2022.

[9] A. Svyatkovskiy, S.K. Deng, S. Fu, and N. Sundaresan, “Intellicode compose: Code generation using transformer,” Proc. ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, 2020.

[10] B. Wei, G. Li, X. Xia, Z. Fu, and Z. Jin, “Code generation as a dual task of code summarization,” Proc. International Conference on Neural Information Processing Systems, 2019.

[11] S. Karaivanov, V. Raychev, and M. Vechev, “Phrase-based statistical translation of programming languages,” Proc. 2014 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming & Software, 2014.

[12] M. Denkowski and A. Lavie, “Meteor universal: Language specific translation evaluation for any target language,” Proc. Ninth Workshop on Statistical Machine Translation, 2014.

[13] M. Rabinovich, M. Stern, and D. Klein, “Abstract syntax networks for code generation and semantic parsing,” 2017. https://arxiv.org/abs/1704.07535

[14] E. Parisotto, A. Mohamed, R. Singh, L. Li, D. Zhou, and P. Kohli, “Neuro-symbolic program synthesis,” Proc. International Conference on Learning Representations, 2017.

[15] L. Spector, “Autoconstructive evolution: Push, PushGP, and Pushpop,” Proc. Genetic and Evolutionary Computation Conference, 2001.

[16] V.I. Levenshtein, “Binary codes capable of correcting deletions, insertions, and reversals,” Soviet physics doklady, vol.10, 1966.

[17] K. Papineni, S. Roukos, T. Ward, and W.J. Zhu, “Bleu: a method for automatic evaluation of machine translation,” Proc. Annual Meeting of the Association for Computational Linguistics, 2002.

[18] C.Y. Lin, “ROUGE: a package for automatic evaluation of summaries,” Proc. ACL Workshop on Text Summarization Branches Out, 2004.

[19] S. Banerjee and A. Lavie, “METEOR: An automatic metric for MT evaluation with improved correlation with human judgments,” Proc. ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization, 2005.