6K-08

循環的複雑度を用いたAlphaCodeが生成したソースコードの評価

寒川楽人† 増田聡‡

東京都市大学† ‡

1. はじめに

近年，プログラミングの需要は高まりつつある．一方，新たなプログラミング言語の学習には，多大なコストと時間要する．その為，これら開発者の負担を軽減する可能性を秘めた人口知能(AI)としてGoogleのAI部門であるDeepMind社からAlphaCodeという競技プログラミング用の自動プログラミングツールが発表された．これは，人工知能を研究する非営利団体であるOpenAIが開発したCodexと呼ばれるAIを改良されて作られた．AlphaCodeはこのCodexを改良し，競技プログラミングが可能になるまで精度を向上されている．これはCodeforceにより実施されたプログラミングコンテストで，5000人を超える参加者のうち上位54.3%という結果を残している．

しかし，AlphaCodeが生成したプログラムのソースコード(以降コードとする)は，競技プログラミングでは良い成績を収めたが，これらのコードを人が見ると，決して可読性に優れたものとは言えない．将来の実用化を目指し，コードを生成し活用する上で，人が行う作業として効率的に保守や管理等を行う必要があることが予想される．

本研究では図1のようなAlphaCodeが生成したコードからサイクロマティック複雑度(Cyclomatic Complexity Number．以降CCNとする)を調査した．AlphaCodeのデモンストレーションサイト[1]に記載されているコードを静的解析ツールであるLizardでCCNを計算し，散布図としてまとめることで，コード行数(Lines Of Code．以降LOCとする)とCCNの相関を求める．そこから外れ値を持つコードとして，いくつかサンプルを取り出し，調査を行った．また，それらのコードとの比較を行う為，IBMが公開しているCodeNetと呼ばれる，人がコーディングしたC++コードのデータセットも調査対象とした．本研究では，相関係数の算出に加え，散布図に回帰直線の描画を行い，AlphaCodeが生成したコードをLOCとCCNの関係の点から評価を行う．

AlphaCode source code evaluation using cyclomatic complexity.

†Gakuto Sangawa, Tokyo City University

‡Satoshi Masuda, Tokyo City University

```cpp
#include <bits/stdc++.h>
using namespace std;
const int N = 2e5 + 10;
vector<int> g[N];
int n, d[N], cnt[N];
void dfs(int x, int fa) {
  d[x] = d[fa] + 1;
  for (int i = 0; i < g[x].size(); i++) {
    int v = g[x][i];
    if (v == fa) continue;
```

図1．AlphaCodeが生成したソースコードの例

2. CCNの調査

2.1 調査方法

本研究では，CCNとLOCの相関[2]に焦点を当て実験を行う．一般的に，CCNについて表1のように言われている．

表1．循環的複雑度の目安

CCN	目安
1～10	理解し易くテストも容易
10～20	複雑だが理解可能，テストは難しい
20～	テスト困難
50～	メンテナンス不可

表1のCCNの目安から，CCNは20以下が望ましいと考えられる．また，相関係数の目安から，相関の有無を調べる．また，AlphaCodeにより生成されたコードに加えCodeNetのデータセットを用いて散布図をプロットし，AIが生成したコードと人間が記述したコードにどのような差異が存在するのか調査を行う．

調査対象とするコードは，AlphaCodeのデモサイトに掲載されているコード並びに，IBMが公開しているデータセットCodeNetを調査対象とする．まずコード静的解析ツールLizardを用いてコード行数およびCCNの計算を行いCSVファイルに出力する．そして，Pythonのグラフ作成ライブラリであるMatplotlibを用いて，調査したデータを散布図として，グラフ上にプロットを行った．また，散布図ではLOCとCCNの関係を可視化するため，横軸をLOC，縦軸をCCNとしプロットした後に，相関係数の算出および回帰直線の描画を行った．以上の調査を終えた後に，プロットされたデータから相関を求め，外れ値をいくつかサンプルとして取り出しコードレビューの対象とした．

2．2 調査結果

調査の結果から，得られた LOC と CCN の散布図を図 2 に示す．

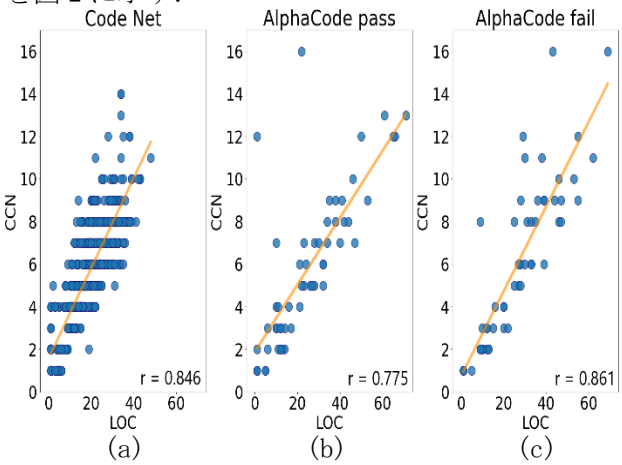

図 2．LOC と CCN の相関

図 2 では横軸が，LOC 縦軸が CCN となる．この図では，以下の基準でコードを分類している．

- 図 2(a) ： CodeNet の C++のデータセット．
- 図 2(b) ： AlphaCode の出力ソースコードで全てのテストケースを通過したコード．
- 図 2(c) ： AlphaCode の出力ソースコードでテストケースを通過しなかったコード．

以上の調査結果から，図 2(a)の CodeNet の C++のデータセットでは，相関係数が 0．846 となり，強い相関関係があることが分かる．また，CCN の最大値 14，中央値は 5 となった．表 1 の目安から複雑だが理解可能と結果が出た．図 2(b)の AlphaCode のすべてのテストケースを通過したコードでは，CCN の中央値が 5 となったが，図から見て取れるように，CCN が最大値 16 のサンプルが見られる．これは表 1 の目安から，複雑だが理解可能と判断出来る．図 2(c)の AlphaCode のテストケースを通過しなかったコードでは，先ほどと同じく CCN が最大値 16 のサンプルが存在した．しかし，CCN の中央値は先ほどの結果とは異なり 6 となった．

3．考察

図 2 の中から外れ値としていくつかサンプルを取り出し，コードの確認を行った．他のサンプルと比較すると，明確な違いが見られた．他のサンプルと比較して，ソースコードの行数あたりの if-else 文による条件分岐の回数および for 文の使用頻度が高いことが分かった．CCN の計算では，for 文も複雑度の計算に加味されるため，他のサンプルと比較して，for 文が多かったことも外れ値となった理由だろう．外れ値となったサンプルの中には，処理内容自体は単純であり，コード行数はそれほど多くないが，入力された数字を読み取る際に複数回に渡り，for 文を使用していたコードも見受けられた．これは，外れ値となりうる理由と言えるだろう．外れ値にならなかったサンプルの中にも，人間にとって「読みにくい」と感じられるコードが見受けられた．変数を使い回している為，何を意味しているのか理解に苦しむ例やサブルーチンを定義しているが，不要である場合などいくつか見受けられた．また，定義済みだが使用していない変数の存在なども，「読みにくい」感じる理由の 1 つと言えるだろう．これらを削除することで，可読性の向上につながると考える．

本調査では CCN に焦点を当てて調査を行った．CCN の計算では，コード内の条件分岐の回数をカウントしていると考えてよい．その為，CCN を下げることを目的とする場合，分岐を減らすことで，CCN の上昇を回避できる．具体的に言えば，if 文での複数回の条件分岐を減らし，1 メソッドあたりの，分岐を減らす．また，複数回にわたり，条件分岐を行う必要がある場合は，別の場所で処理を行うことにより，1 つの関数内で行われる条件分岐の回数を減らすことも効果的だと考える．

ソフトウェアを開発する場合は，より CCN を下げることに力を入れる事が予想される．CCN が高いことで，テストや保守が困難になりやすくなることは事実である．しかし，CCN が低いコードにも，バグが存在している可能性がある．また，CCN は制御フローグラフのノード数に注目している為，人間にとっての”複雑さ”を判断するための指標としては，適切でないとの意見もある．その為，他のメトリクスと併用しコードを評価することが適切だと考える．

4．まとめ

本研究では，循環的複雑度を用いて AlphaCode が生成したソースコードの評価を行った．今後は，他の自動プログラミングツールにも対象を広げ各種メトリクスも調査し，自動プログラミングの実用化への貢献を目指す．

参考文献

[1] Deepmind, AlphaCode Attention Visualization, https://alphacode.deepmind.com/, (参照 2022/11/30)
[2] 阿萬裕久；野中誠；水野修．ソフトウェアメトリクスとデータ分析の基礎，コンピュータソフトウェア，2011，28．3：3_12-3_28，