B-010

ソフトウェア構造グラフの分散表現に基づくデザインパターン検出の評価

Evaluation on detection of design pattern based on distributed representation of software structure graph

若松 一樹† 梅澤 猛† 大澤 範高†
Kazuki Wakamatsu Takeshi Umezawa Noritaka Osawa

1. はじめに

ソフトウェアに使用されているデザインパターンを検出することができれば、プログラムの機能や目的の理解に役立つ。Nazar らは、Java プログラムのクラスやメソッドを構成するコード特徴量を分散表現し、機械学習によってデザインパターンを検出するモデルを作成した[1]。クラスやメソッドの名称を基にしたこの手法は、平均で適合率 74.1%、再現率 71.5%と比較的検出精度が高いものの、適切に命名されていない場合には適用できない。

Zhang らは、典型的なデザインパターンのグラフ表現を定義し、プログラムコードの中から同型のグラフとなる箇所を探索することで検出を行っている[2]。この手法では、命名の適切性に依存せずに検出が可能ではあるが、デザインパターンのグラフ表現と同形のものを探索するため、実装が一致しない場合検出ができないという課題が残る。

そこで、本研究では、デザインパターンは、周辺要素にも影響を与えるという仮説をたて、デザインパターン周辺のコンテキストを含めて学習を行った際のデザインパターン検出モデルの評価を行う。本研究ではプログラムコードを、クラス/インタフェース間の関連により有向グラフに変換し、それから抽出した部分グラフの分散表現に基づいたデザインパターン分類モデルを構築する。

2. 実験手法

本研究ではオープンソースソフトウェアのソースコードからクラス、インタフェースを検出し、それらの継承関係、メンバ変数の参照、メソッドの呼び出し関係などをノードV、エッジEで構成される有向グラフ$G = (V, E)$として表し、その部分グラフを分散表現したものを用いてデザインパターンの検出を行う。

グラフの分散表現とは、グラフを固定次元に変換した埋め込みベクトルである。固定次元のベクトルであることから、機械学習アルゴリズムを利用しやすい。

今回の実験は、データセットの生成、分散表現モデルの訓練、分散表現を入力とする分類モデルの構築、分類モデルの評価の 4 フェーズにより構成される。図 1 に実験のフェーズ間の関係を示す。

2.1 データセットの生成

オープンソースソフトウェアのソースコードセット S からクラス・インタフェースを検出、関連や継承関係を解析し、ソフトウェア構造を表すグラフGを作成する。その後、Gの部分グラフで構成される部分グラフ集合Pを作成する。部分グラフ集合Pの要素であるp_vは、ノード$v \in V$を中心として、距離が 3 以内の部分グラフである。

† 千葉大学 Chiba University

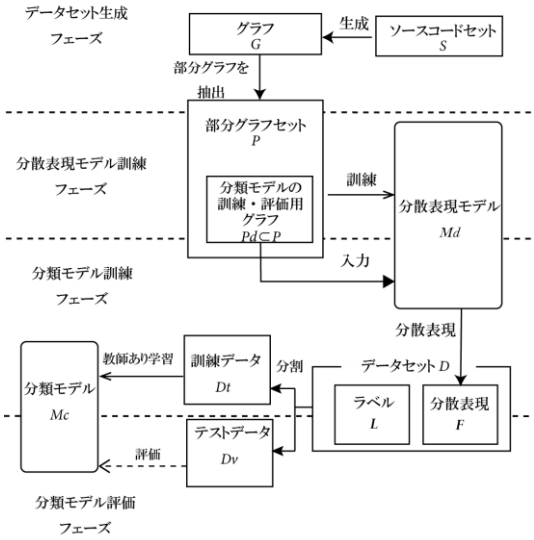

図 1　実験のフェーズ間の関係

部分グラフ集合Pのうち、デザインパターンの一部であることを表すラベルを付けたものの集合を$Pd \subset P$とする。Pdの要素$p_i \in Pd$に対応するラベルを$l_i \in L$とする。ここで、Lはデザインパターンの種類を表すラベルの集合である。

基本データ型はクラスではないためノードに含めない。エッジ生成ルールを表 1 に示す。

図 2 に、クラス A、B、C、D、E、F を定義するコード例と、それから作成されるグラフを示す。図 2 には説明のためにノード・エッジ種別を示しているが、実際のグラフのノード、エッジに区別はない。

表 1. エッジ作成ルール

エッジ種別	ルール
Associates	メンバ変数として持つ
Extends	クラスを継承する
Implements	インタフェースを実装する
Invokes	対象のメソッドを呼び出す
Returns	対象を返すメソッドがある
Accesses	対象クラスのメンバにアクセスする

本研究で用いたデータセットは Nazar らが公開しているデータセットをもとに作成した。Nazar らのデータセットは、12 種類のデザインパターンを含むプログラムとデザインパターンを含まないプログラムを各 100 件ずつ含んでいる。各プログラムの一部のクラスには、デザインパターンの種類を表すラベルが付与されている。今回はその内、Factory、Adapter、Builder、None(デザインパターンではない)を検出対象とした。すなわち、ラベルの種類数$|L|$=4 である。

今回は Nazar らが使用したソフトウェアの内、ソースコードが取得可能だった 11 件のオープンソースソフトウェアから、上記の手順に従いグラフGを生成し、25,633 件の部分グラフからなる部分グラフセットPを得た。Pの要素のう

ち、中心にするノード にラベルが与えられているもので Pdを構成した。表 2 にPdに含まれるパターンと、パターンごとのグラフの数を示す。

表 2. Pd に含まれるパターンごとのグラフ数

パターン名	グラフ数
Factory	36
Adapter	47
Builder	49
None	72

```
class A extends B{
  C func(){return D();}
}
class B{
  C func(){return C();}
}
class C{
  void func(){}
}
class D extends C{
  F member;
  void func(){
    member.func();
    member.data+=1;
  }
}
interface E{
  public void func();
}
Class F implements E{
  public int data=0;
  public void func(){}
}
```

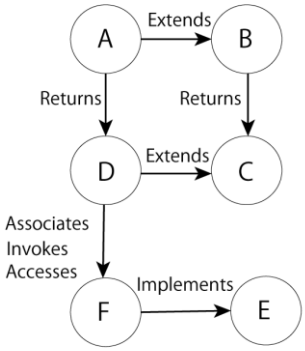

図 2. コード例と作成されるグラフ

2.2 分散表現モデルの訓練

部分グラフセットPを用いて、分散表現モデルMdを graph2vec で訓練する。graph2vec のハイパーパラメータとして、学習率 0.025、エポック数 1 で 128 次元のベクトルを生成するよう設定した。

部分グラフ$p_i \in Pd$をMdに入力して得た分散表現$f_i \in F$と、ラベル l_iの組(f_i, l_i)を作成し、これを要素として持つデータセットDを構築する。ここで$F = Md(Pd)$である。

2.3 分類モデルの訓練

分散表現とラベルの組の集合Dを用いた教師あり学習で、分類モデルMcを訓練するデータセットをDt、テストデータセットをDvとする。今回の実験では、Dvは、Dから抽出した 30%を使用した。$Dt = D$とした場合の分類モデルを$Mc0$とし、$Dt = D - Dv$とした場合の分類モデルを $Mc1$ とする。

分類アルゴリズムには SVM、ランダムフォレストを用いた。SVM は rbf カーネルを使用した。ランダムフォレストの分類木数は 20、最大深さは 5 とした。

2.4 分類モデル評価フェーズ

分類モデル$Mc0, Mc1$によってテストデータDvの分類を行ったときの適合率、再現率を表 3 に示す。また、分類モデル$Mc1$を用いてランダムフォレストにより分類を行った際の結果を混同行列で図 3 に示す。

表 3 より、テストデータが訓練データと重複するモデル $Mc0$ では、90%以上の高い精度で分類ができるため、学習は行われていることがわかる。しかし表 3 より、訓練データとは異なるテストデータで評価する$Mc1$ では、適合率、再現率ともに 0.45 程度と低いことがわかる。これらから、過学習が起きていることが疑われる。

過学習が起きている原因としては、今回の実験で使用したグラフ数が 204 件と少なかったために、十分な汎化が行われなかったためと考えられる。

表 3 分類結果

	分類モデル：$Mc0$		分類モデル：$Mc1$	
	適合率	再現率	適合率	再現率
SVM	0.90	0.93	0.45	0.46
ランダム フォレスト	0.93	0.91	0.46	0.45

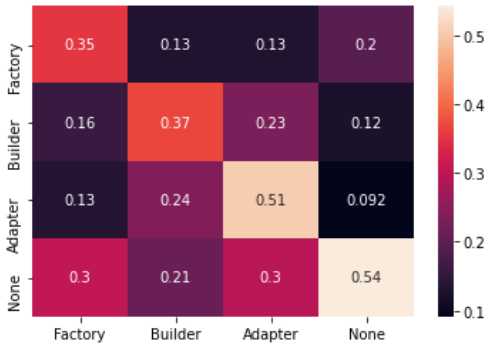

図 3. 分類モデル$Mc1$を用いてランダムフォレストによる分類を行った際の混同行列

3. おわりに

ソフトウェア構造を表すグラフの分散表現を用いたデザインパターン検出手法の検証を行った。結果として、適合率・再現率ともに最大で 0.46 となった。デザインパターンとそのコンテキストのグラフによる学習が可能であることは示せたが、精度を向上させる必要がある。

今後は過学習を防ぐために、ソフトウェアレポジトリから多くのソースコードを取得し、パターンごとのグラフ数を増やすことで、十分に汎化された分類モデルの構築を目指す。

参考文献

[1] N. Nazar, A. Aleti, "Feature-Based Software Design Pattern Detection" , Computing Research Repository (CoRR) (2020)
[2] P. Zhang, D. Yu and J. Wang, "A Degree-Driven Approach to Design Pattern Mining Based on Graph Matching" , 2017 24th Asia-Pacific Software Engineering Conference (APSEC) (2017), pp. 179-188,
[3] A. Narayanan, M. Chandramohan R. Venkatesan, L. Chen, Y. Liu and S. Jaiswal, "graph2vec: Learning Distributed Representations of Graphs." (2017).