コードクローン関係を考慮したソフトウェア部品グラフの視覚化手法に関する考察

2013SE040 菱川真以　2013SE140 成瀬磨耶　2013SE179 佐々木麻衣
指導教員：横森励士

1　はじめに

ソフトウェアが肥大化するにつれて，構成する部品の数や，部品間の関係が複雑になり，大量の部品で構成されるソフトウェアの把握が難しくなる．例えば，大量の部品の中には同一もしくは類似した機能を実現するために類似コード片が存在することが考えられる．これらの類似コード片をツールによって抽出した場合，抽出された類似コード片が単純に一致しているものなのか，解消すべきコード片なのかを判別するためには，抽出された類似コード片の記述を確認して類似性を判定する必要がある．

本研究では，描画された部品グラフ上でコードクローン関係を持つ部品同士がどのように配置されるかを調査することでその部品間に存在するコードクローンの類似度を推測する手法を提案する．部品グラフ上の部品の位置関係から，その部品間に存在するコードクローンの関連性の高さを推測できるのではないかと考えた．実際にオープンソースプロジェクトのソースコードに対して提案手法を適用し，部品グラフとして描画された部品同士の位置関係が部品間にまたがるコードクローンの関連性を推測するための情報として有効かどうかを評価する．

2　背景技術

2.1　ソフトウェア部品と部品グラフ

一般に，ソフトウェア部品とはモジュールや関数，クラスなどのソフトウェアの構成要素を指す [4]．以降，ソフトウェア部品を単に部品と呼ぶ．ソフトウェア部品間には，ある部品間の属性や振る舞いを利用することによる利用関係や，共通のコード断片をもつことによるコードクローン関係を定義することができる．以下では，ソフトウェア部品間の関係を表現するために Java アプリケーションを対象として部品グラフを定義する．部品グラフでは各クラスを頂点，利用関係やコードクローン関係を辺で表す．

2.2　コードクローン

コードクローンとは，ソースコード中での類似または一致したコード片を指す [5]．ソースコード内での類似した処理が各部品で行われる場合のように，同一の部品内だけにとどまらず，異なる部品間に存在する場合もある．コードクローンはソフトウェアの保守工程においてプログラム管理の手間を増大させる可能性を持つため，コードクローンとなる部品を有する部品は，常に着目する必要がある．コードクローンを検出するツールとして，CCfinder が公開されている [5]．CCfinder はソースコードをトークン列とみなし，一致したトークン列が，ある値以上の長さのものを類似コード片として抽出する．機械的に抽出されたコードクローンに対しては，それらが解消すべきコード片なのか，単に一致しただけなのかを判別するために，最終的に類似コード片同士を比較し，抽出されたコード片の意味づけを行う必要がある．

2.3　グラフの視覚化手法

グラフを視覚的に表現する手法として，多数の頂点をもつグラフなどをコンピュータ上で自動配置するための手法やツールが提案されている．Graphviz[3] はオープンソースで開発された，指定したグラフの頂点を自動的に頂点を配置し，可視化された情報として表示するツールである．利用関係を実現するソフトウェア部品グラフに対して，頂点，辺に関するデータを記述し，描画のアルゴリズムを与えることで自動的に頂点を配置し，グラフを描画する．

竹仲は，スプリングアルゴリズムを用いて部品グラフを描画する研究を行なった [1]．部品グラフに対して適用した場合，機能を単位としてそれらの機能を構成する部品がまとまって表示され，機能単位でソフトウェアを把握することに有効であることを示した．スプリングアルゴリズムとは，多数の頂点を持つグラフをコンピューター上で視覚的な面から効果的に自動配置するアルゴリズムであり，グラフ上の辺が可能な限り交差しないようにグラフを描写することができる [2]．スプリングアルゴリズムで用いられる頂点はクーロンの法則に伴う電荷を持ち，頂点同士は互いに反発する．結果として，辺により繋がった頂点同士は引力によりグラフ上の距離は短くなり，一方で繋がりを持たない頂点同士は斥力によりグラフ上の距離が遠くなる．

3　ソフトウェア部品グラフの視覚化手法に関する考察

3.1　研究の動機と提案するアプローチ

コードクローン検出ツールで検出されたコードクローンに基づいて作業を行う場合，検出されたコードクローンに対する意味付けを行うことが必要となる．このとき，意味付けを支援するための情報として，部品間の関係などが利用可能なのではないかと考えた．一方で，過去の研究 [1] で提案された手法では，機能を単位としたまとまりとして，部品をグラフ上で描画できた．描画されたグラフ上で部品同士が近くに配置された場合ではクラス間の関連性が高く，一方で遠くに配置された場合はクラス間の関連性は低いのではないかと仮定することで，描画されたグラフにおける部品間の部品同士の位置関係を，コードクローンの

関連性の強さを推測するための指標として用いることができるのではないかと考えた．

3.2 提案するアプローチと実現した分析環境

上記の案をもとに，描画された部品グラフの情報を利用して類似したコード片の関連性をクラス間の関係から推測する手法を提案する．提案手法の分析手順を以下に示す．

1 各描画手法で描かれた部品グラフを得る

(1-1) Classycle[8] を用いて利用関係を抽出する．

(1-2) 利用関係をグラフの視覚化ツールに入力として与え，部品グラフを描画する．

2 対象プロジェクトのコードクローン関係を得る．

(2-1) CCfinder[5] を用いて，コードクローン関係を持つ部品の対を得る．

(2-2) (2-1) で得た部品の対それぞれについて，コード片の記述を精査することで，コードクローンの関連性を評価レベルとして求める．

(2-3) (1-2) で作成したそれぞれのグラフ上で，それぞれの部品対間の距離を計測する．

(2-4) 描画手法ごとに，(2-2) で得たコードクローンの関連性と (2-3) で得た部品間の距離を組として，それらの値の関連性を評価する．

3 描画手法ごとに，(2-4) で得られた組の値の関係を調べる．

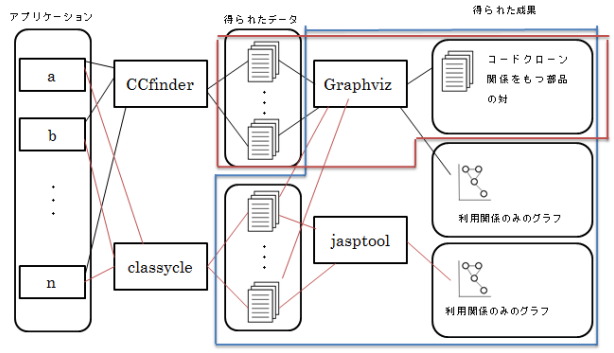

図 1 分析手順の経過

3.3 評価レベルの定義

類似したコード片の関係性の強さを，コード片の内容をもとに次のように 5 段階で表現する．複数の類似コード片が混在する場合は，一致度が一番高いものを採用する．

Lv1. メソッド呼び出しの連続など，構文が連続していることで検出されたもの．構文内の処理がまったく異なり，コードクローンとは呼びにくいもの．

Lv2. if 文や for 文などの構文で，表面上の記述内容は似ているが，中で記述されている内容が異なるもの．

Lv3. if 文や for 文などの構文中で記述されている内容がほぼ同じ内容であるもの．

Lv4. 1 つのメソッド内すべての構文や括弧内の処理が完全に一致しているもの．

Lv5. 2 つ以上のメソッド内すべての構文や括弧内の処理が完全に一致しているもの．

4 評価実験

提案手法の有効性を確認するために，実際のオープンソースプロジェクトに提案手法を適用した．過去の研究 [1] で提案された描画ツールである Jasptool(以下， Jasp) を用いるとともに，比較対象を求めるため予備実験を行なった．予備実験では，Graphviz[3] で用意されている様々な描画手法から，比較対象の候補を選出した．その中から，スケーラビリティの観点から利用可能でかつ，部品間の関係を表現できる手法を選んだ．その結果，Graphviz[3] で提供されるレイアウトの 1 つであり，ノードが大量にある場合にも対応可能な手法である Sfdp(以下， Sfdp) を比較対象として選択した．Sfdp は，Jasp[1] と同様にスプリングアルゴリズムを利用しているが，蓄えるエネルギーではなくかかる力が最小となるように描画する手法で，ノード数が多い場面でも利用可能な描画アルゴリズムである．描画手法以外から求めるアルゴリズムとして，部品グラフ上で 2 つの部品間を移動するために必要な辺を辿る回数 (以下， Trace) を比較対象として選出した．

4.1 評価基準について

表 1 で示す 11 個のオープンソースプロジェクトに対して提案手法を適用する．コードクローンを共有する部品それぞれにおいて 3 つの手法 (Jasp，Sfdp，Trace) でそれぞれ距離を求めることにより，3 つの手法それぞれについて，(距離，コードクローンの評価レベル) という値の組が入手できる．それぞれの手法における距離と評価レベルとの関係をケンドールの順位相関係数で示し，評価する．ケンドールの順位相関係数では，2 つの項目間で順位関係が一致する組の数を求め，その数に基づいて順に相関係数を算出する．相関係数の範囲は-1 から 1 の値であり，順位が完全に一致した場合では 1.0 の値をとり，一方順位が完全に逆の場合では-1.0 の値をとる．今研究では 2 つの項目である評価レベルと距離の値に対し，評価レベルが高い場合，対応する距離の値は小さく，一方で評価レベルが低い場合，対応する距離の値は大きくなることが求められているため，-1 に近い値になるほど想定した結果に近い．

4.2 評価実験の結果

表 1 は 11 個のプロジェクトに 3 つの手法（Jasp，Sfdp，Trace）を用いて部品間の距離を計測し，2 つの部品間の距離の値と評価レベルの値の間の関係をケンドールの順位相関係数で評価を行ない，その結果を示したものである．表 1 は，プロジェクトごとに，コードクローン関係を持っている部品対の数 (以下，数) を示したのちに，3 つの手法それぞれの順位相関係数，p 値を示している．p 値とは，データから算出された検定統計量の値に関して，帰無仮説が棄

表 1 Jasp，Sfdp，Trace の 3 つの手法の距離と評価レベルにおけるケンドールの順位相関係数の結果

		Jasp		Sfdp		Trace	
プロジェクト	CC 数	相関係数	p 値	相関係数	p 値	相関係数	p 値
Pixelitor(ver0.4)	22	-0.006	0.973	0.243	0.195	0.195	0.357
Jasmin	58	-0.494	2.197e-06	-0.302	0.003	-0.216	0.065
Jamod	91	-0.068	0.411	0.183	0.027	-0.113	0.225
Hodoku	50	-0.136	0.261	-0.060	0.596	0.141	0.275
Pixelitor(ver0.2)	10	-0.488	0.088	-0.684	0.017	-0.709	0.022
Solitaire	5	-0.589	0.179	-0.224	0.602	-0.5	0.269
jlGui(ver3.0)	115	-0.047	0.018	-0.194	0.018	-0.132	0.090
Lept4J(ver1.2.4)	13	-0.196	0.401	-0.058	0.799	0.276	0.295
GeoAPI	56	-0.257	0.015	-0.035	0.741	0.080	0.472
Horizon	9	-0.036	0.904	-0.495	0.111	-0.182	0.574
Turtlesports	50	-0.427	0.0001	-0.214	0.052	-0.528	1.52e-05

却される水準である．ここでは，p 値が 0.1 以下であるとき十分な標本数を持っており，有意な相関であるとする．

各手法における相関係数に着目したところ，表 1 で示す結果から以下のように比較した．

Jasp　3 つの手法の中で，一番強い負の相関を示した．相関係数が -0.2 以下の弱い負の相関を示すプロジェクトは 5 個， -0.4 以下のやや強い負の相関を示すプロジェクトは 4 個存在した．有意な相関であると認められるプロジェクトは 5 個存在し，それらの多くはやや強い負の相関を示していた．

Sfdp　負の相関を持つプロジェクトが多かったものの，正の相関を示すプロジェクトも見られた．相関係数が -0.2 以下の弱い負の相関を示すプロジェクトは 5 個，-0.4 以下のやや強い負の相関を示すプロジェクトは 2 個存在した．有意な相関であると認められるプロジェクトは 5 個存在し，それらは弱い相関を示すものから，相関がないことを示すものまでさまざまであった．

Trace　全体的に相関がないプロジェクトが多く見られた．相関係数が -0.2 以下の弱い負の相関を示すプロジェクトは 3 個，-0.4 以下のやや強い負の相関を示すプロジェクトは 2 個存在した．有意な相関であると認められるプロジェクトは 4 個存在し，それらは相関がないものから負の相関を示すものまで多種多様であった．

4.3　プロジェクトにおける結果の違い

本節では Jasmin[10] に対し 3 つの各手法を用いた場合の結果の違いについて考察する．Jasmin[10] は Java の仮想マシンのアプリケーションであり，コードクローン関係を持つ部品対が 58 組存在した．以下に挙げる図 2，図 3，図 4 では縦軸を評価レベル，横軸を各手法における部品間の距離又は辺を辿る回数として，それぞれの手法による距離と評価レベルの分布を示したものである．

Jasp　図 2 から評価レベルが高い部品について，他の手法よりも部品グラフ上で近い場所に配置されており，想定に近い結果となっている．このことが 3 つの手法の中で一番負の相関が強くなった原因であると考えられる．一方で，評価レベルが低い部品対は，部品グラフ上で近い場所にも遠い場所にも配置されていることを確認した．近くに配置されたコードクローンを持つ部品対のコード片は，関連性の高いものだけではないことが分かる．

Sfdp　図 3 から Jasp の場合と比較して，評価レベルの高い部品対が全体的に散布図上で右の方に配置されていることが分かる．評価レベルが低い部品対については，Jasp の場合と同様な結果が得られ全体としては Jasp よりも弱い負の相関がみられる結果となった．

Trace　図 4 の散布図から分かる通り，他の 2 つとは違った傾向を示した．評価レベルが 5 であった部品対はいずれも距離が 2 の部品対であり，単純に距離が短ければ評価レベルが上がるわけではないことが分かる．順位相関係数も 3 つの手法の中で最も低かった．

5　考察

今研究では描画されたグラフ上での位置関係をもとに，部品間のコードクローンの関連性を推測するための手法を提案し，評価実験を行なった．評価実験の結果，スプリングアルゴリズムを用いてバネが持つエネルギーの和を最小にする手法を用いて描画する Jasptool[1] を用いた手法が，全体として負の相関を示しやすく，他の手法よりも負の相関を示していることを確認した．部品同士が近くに配置された場合ではクラス間の関連性が高く，一方で遠くに配置された場合ではクラス間の関連性は低いという仮定が，多くのプロジェクトである程度成り立っていることが確認できた．

個別のプロジェクトである Jasmin[10] に着目し評価実験を行なった結果では，評価レベルが高いコードクローン

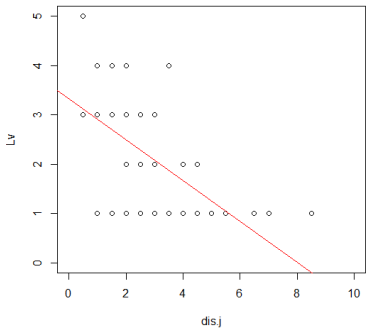

図 2　Jasmin の散布図 (Jasp)

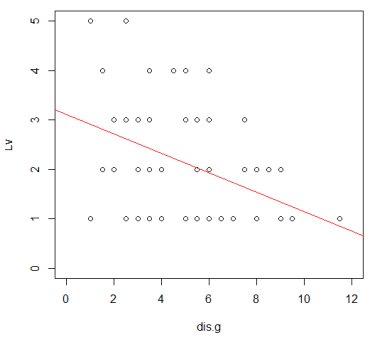

図 3　Jasmin の散布図 (Sfdp)

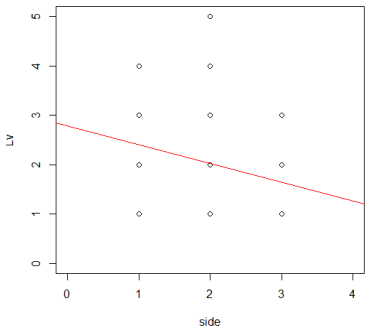

図 4　Jasmin の散布図 (Trace)

関係を持つ部品対に対しては想定に近い結果を得られた．しかし評価レベルが低い場合は，距離の遠近に関係なく存在した．例として距離が近いのにも関わらず，評価レベルが低いコードクローン関係を持つ部品対のソースコードを図 5，図 6 に挙げる．赤枠内で示す部分のみが一致しており，そこからは，派先元や実装しているインタフェースのみが一致していたことがわかる．派生元や実装しているインタフェースが一致しているという事実は，それらのクラスが関連性を持つことは示していると考えられるが，コードクローンの関連性を示すとは限らないことが分かった．部品グラフ上で辺として反映する利用関係を選ぶことで，コードクローンに関連するような利用関係のみで部品グラフを構築し，手法が想定したような方法で部品グラフを描画できるのではないかと考える．

```
public class InterfaceCP extends CP implements RuntimeConstants
{
  ClassCP clazz;
  NameTypeCP nt;

  /**
   * @param cname Name of class defining the interface
   * @param varname symbol for the interface method
   * @param sig Signature for method
   */
  public InterfaceCP(String cname, String varname, String sig)
```

図 5　InterfaceCP.java のソースコードの記述の一部

```
public class IntegerCP extends CP implements RuntimeConstants
{
  int val;

  /**
   * @param n Value for integer constant
   */
  public IntegerCP(int n)
```

図 6　IntegerCP.java のソースコードの記述の一部

6　まとめ

今研究では，描画されたグラフ上での部品の位置関係とその間に存在するコードクローンの関係性について調査を行なった．その結果，Jasptool[1] を用いて描画されたグラフにおいて，描画されたグラフ上での距離とコードクローンの関係性に弱い，もしくはやや強い相関を確認した．それらは他の手法に比べ，強い相関を示していた．

今後は他のプロジェクトに対する適用を行い，一般性を検証するとともに，部品グラフ上で表現する利用関係を精査することで，描画された部品グラフの精度の向上が見られるかを検証したい．

参考文献

[1] 竹仲　孝盛，“スプリングアルゴリズムに基づくソフトウェア部品グラフの視覚化手法の実現”，南山大学情報 理工学部 2015 年度卒業論文，2016.

[2] 臼杵正郎・杉山公造 (2003)，“群れアルゴリズムを応用したグラフ描画法”，北陸先端科学技術大学院大学：https://ipsj.ixsq.nii.ac.jp/ej/?action=repository_uri&item_id=40309&file_id=1&file_no=1.

[3] graphviz：http://graphviz.org/.

[4] C. Krueger：“Software Reuse”，ACM Computing Surveys，vol.24，no.2，pp.131-183，1992.

[5] T.Kamiya，S.Kusumoto，K.Inoue：“CCFinder：A multilinguistic token-based code clone detection system for large scale source code”，IEEE Transactions on Software Engineering，vol. 28，no. 7，pp. 654 - 670，2002.

[6] turtlesport：http://turtlesport.sourceforge.net/.

[7] Pixelitor：http://pixelitor.sourceforge.net/.

[8] Classycle：http://classycle.sourceforge.net/.

[9] R Development Core Team：https://www.r-project.org/.

[10] Jasmin：http://jasmin.sourceforge.net/.

[11] 豊田秀樹，『回帰分析入門』，東京図書，2012.