コードクローンを用いたコンポーネントランク拡張手法の改良を目的とした評価実験
―コードクローン検出の粒度による影響の調査―

2011SE066 伊原大輝　2011SE082 石川稜
指導教員：横森励士

1　はじめに

近年ソフトウェアが大規模化している中で，それに伴いソフトウェア部品の数も増大化し，内部の関係も複雑化している．このような環境下では，ソフトウェアをモデル化して，メトリクスに基づいてある特性を持つ部品を抽出するという方法が有効であると考える．千賀が行った先行研究では，コンポーネントランクの計算にコードクローンを反映させることで，コードクローンから共通して利用されやすい部品を抽出する手法 [3] を提案した．評価実験 [5][6] では，手法の有効性を確認したが，実際の利用のためにさらなる精度の向上が必要であることがわかった．

本研究では，コードクローン検出の条件である，検出対象となる連続したトークンの長さを変えることで，手法の精度が向上するかを確認する．連続したトークンの長さが長くなるほどコードクローンの粒度は大きくなり，粒度の大きいコードクローンはその分一致する部分が多くなるので，意味的にも関連高いものだけが検出され，精度が向上するようになるという仮説に基づいている．実際に，オープンソースの開発プロジェクトに対して適用を行い，提案する手法が精度の向上に役立つかを検証する．[3] の手法が実際に利用可能な精度を持つようになることで，ソフトウェアの保守作業の支援に役立てることができると考えられる．

2　背景技術

2.1　部品グラフとコンポーネントランク

一般に，ソフトウェア部品とはモジュールや，クラスなどのソフトウェアの構成要素を指す．以降，ソフトウェア部品を単に部品と呼ぶ．ソフトウェアは構成要素の部品間で相互に属性やふるまいを利用しあう事で 1 つの機能を提供する．本研究では，ある部品がある部品を利用するとき，この部品間に利用関係が存在すると考え，部品グラフとしてモデル化する．部品を頂点，利用関係を有向辺で表したグラフを部品グラフと呼ぶ．以降，V を部品（頂点）の集合，E を有向辺の集合として，部品グラフを $G = (V, E)$ と表現する．

コンポーネントランク [1] は利用頻度に基づいて部品グラフからそのソフトウェアにおける各部品の重要度となる評価値を算出する手法である．コンポーネントランクでは，部品グラフ $G = (V, E)$ 上の個々の辺および頂点に対して重みを繰り返し計算し，その後，対応する頂点の重みを各部品の評価値として，順位付けを行い評価を行う．重みとは，次のように定義される．

頂点の重み

部品グラフ G 上の各頂点 v は，$0 \leqq w(v) \leqq 1$ の重みを持ち，G の頂点の重みの総和は 1 となる．

辺の重み

頂点 v_i から v_j への辺 e_{ij} に関する辺の重み $w'(e_{ij})$ を式 (1) のように定義する．

$$w'(e_{ij}) = d_{ij} \times \ w(v_i) \tag{1}$$

d_{ij} は配分率と呼び，$0 \leqq d_{ij} \leqq 1$ かつ $\sum_i d_{ij}=1$ を満たす値とする．原点 v_i から v_j へ利用関係が存在しない場合，$d_{ij}=0$ とする．この配分率 d_{ij} は，頂点の重みの再計算において，有向辺の終点となる頂点の重みの決定に利用され，各頂点の重みが利用関係の先の頂点に分配される．

繰り返し計算の手順

1. 評価値の初期値として，各頂点に正の値を与える．
2. 値の変化が一定以下になるまで，次の計算を繰り返す．
   - 辺の重みを現在の頂点の重みから決定する．
   - その頂点に入ってくる辺の重みの和として，頂点の重みを再計算する．

2.2　コードクローン関係を考慮したコンポーネントランクを拡張する方法

コードクローン [2] とは，ソースコード中での類似または一致した部分であり，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．コードクローンは無意味に出現するものではなく，同一処理が必要になるなど，開発の何らかの意図によって作りこまれる場合が多い．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ので，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．過去の研究では，コードクローンを持つ部品同士を部品グラフで結合する前後でコンポーネントランクを比較すると，結合する部品から共通して利用される部品の評価値が下がることに着目し，結合する前後での各部品の評価値の変化を表示するツールを作成した [3].

実際のオープンソースプロジェクトに対して適用を行い，評価値の変化が想定した通りに起こっているか [5]，評価値が減少した部品が想定した通りコードクローン内で利用されるような部品であるか [6] を検証した．実験結果からは，想定したこれらの事象がある程度確認でき，ある程度の有効性を確認したが，ツールなどで有効に利用する

ためには，さらなる精度の向上をはかることが必要であることを確認した．

3　コードクローン検出時のしきい値の変化によるコンポーネントランク拡張手法の改良

3.1　実験の動機

[5][6] で行われた研究の考察によると，関連性の弱いコードクローンに基づいてノード結合を行うと，関係の弱い部品同士の結合によって部品グラフの構造が変化し，それによって想定した評価値の変動が得られない場合が多いことがわかった．このことから，精度向上のための方法として，コードクローン検出時のしきい値を上げ，より大きいかたまりのコードクローンのみを検出することで，より関係性の強いコードクローンのみが検出でき，想定した結果が得られやすくなるのではないかと考えた．

3.2　実験の手順

1. 実際のオープンソースプロジェクトを収集し，各プロジェクトから 1 バージョン入手する．
2. CCFinder[2] を用いてクラス間のコードクローンの関係を抽出する．CCFinder は，連続して一致するトークンが一定以上のコード片をコードクローンとして認識するが，そのしきい値を，30 以上，40 以上，50 以上，70 以上の計 4 通りで設定し，それぞれの場合の結果を入手する．コードクローン抽出の条件から，これらの得られたコードクローン集合間の関係は包含関係にあることも確認している．
3. Classycle[5] を用いてクラスの利用関係の抽出を行う．
4. 3 で得られた結果から部品グラフを構築する．
5. それぞれの検出結果においてコードクローン関係を持つ部品を結合し，コンポーネントランクを 4 通り計算する．
6. 4 つのコンポーネントランクを結合前のコーポンネントランクと比較し，評価項目に基づいて評価を行う．

3.3　部品の分類について

コードクローン関係により結合されなかった部品に着目し，それらの部品を結合した部品群からの利用関係を用いて 3 つのグループに分け，以下の項目で評価する．

結合されなかった部品の分類

GroupA コードクローン関係により結合された部品群内の部品の 2 つ以上から共通して利用される部品 (以下，A)

GroupB コードクローン関係により結合された部品群内の部品の 1 つが共通して利用される部品 (以下，B)

GroupC コードクローン関係により結合された部品群から全く利用されていない部品 (以下，C)

3.4　評価項目

項目 1 A，B，C それぞれに属する部品の数の変化

4 つのノード結合の条件それぞれの適用結果の部品グラフに対して分類を行い，A，B，C に属する部品の数の違いを調査する．予想される傾向として，トークン数が増大していくとコードクローンによる部品の結合が起こりにくくなり，A の部品が減少することが考えられる．

項目 2 A，B，C に属する部品の平均変動率の変化

項目 1 と同様に 4 つのノード結合の条件で得られた部品グラフに対し分類を行い，A，B，C に属する部品の平均変動率を調査する．ここでいう変動率とは，統合前後の各頂点の重みの変動率を用いる．予想される結果として，不必要な結合が起こりにくくなることで，A に属する部品の評価値が下がることが期待される．

変動率の定義

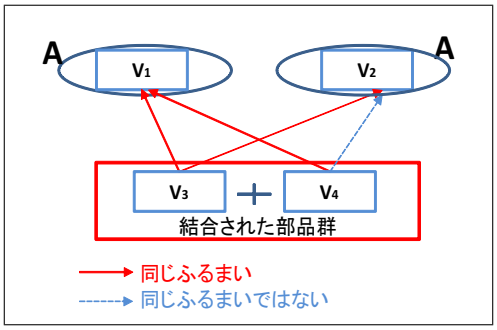
部品数により 1 つの頂点あたりの重みが異なるので，変動率にはグラフにおける頂点の総数を考慮する．ある頂点 v_i の頂点統合前のグラフでの評価値を $w(v_i')$，また，頂点統合のグラフにおける頂点の総数を $|V|$，頂点統合後のグラフにおける頂点の総数を $|V'|$ とする．このとき頂点 v_i の評価値の変動率を式 (3) で定義する．

$$\frac{w(v_i') \times |V'|}{w(v_i) \times |V|} \times 100 \tag{2}$$

項目 3 A に属する部品について，結合された部品群から同じように利用されているかどうか

4 つのノード結合の条件で得られた A の部品それぞれについて，結合された部品群から同じように利用されているかを調査する．図 1 はその例を示しており，V_1，V_2 は A に属している．V_1 は V_3，V_4 の結合している部品から同じように利用されているので該当するが，V_2 は該当しない．表では，該当する A の部品の数を原因部品数と表現する．また，適合率と再現率を定義し，各条件で A の部品のうち結合部品群から同じように利用されていた部品である割合と，各条件でそれらがどれくらい補足できたかを示す．

図 1　結合部品群から同じように利用された部品



適合率と再現率の定義

A の部品数 N に対する，結合された部品群から同じように利用されている A の部品数 R の割合を適合率として式 (3) で定義する．

$$\text{適合率} = \frac{R}{N} \times 100 \tag{3}$$

同様に 4 つの条件のどれかで同じように利用されている A の部品の集合の要素 (要素数 C) に対して，各条件結合された部品群から同じように利用されている A の部品数 R の割合を再現率として式 (4) で定義する．

$$\text{再現率} = \frac{R}{C} \times 100 \tag{4}$$

4 実験の結果

31 種類のオープンソースプロジェクトに対して適用した結果，上手くいかなかった事例が 19 種類あった．A の部品が無かったものが 2 種類，検出の条件を変えても結果が変わらなかったものが 4 種類，ある一定以上のトークン数で全く検出できなくなったものが 13 種類といった結果だった．その他の 12 プロジェクトではおおむね想定した結果を示した．以降，12 の事例のうち，3 事例を中心として結果の紹介を行う．

事例 1 ：JmDNS に対する分析結果

JmDNS とは，マルチキャスト DNS の機能を Java で実装したソフトウェアである，表 1 は JmDNS に対する適用結果で，以下の事が得られた．

- 検出時のトークン数が上がるにつれて結合された部品の数も減少しており，それに伴い A の部品数は減少している．特に，50 から 70 に変えた時，A の部品数が大きく減少しており，70 では満足に検出できていないことがわかる．
- A に属する部品の評価値は，トークン数が上がるにつれて減少する傾向にある．30 では C に属する部品のほうが平均して減少していたが，それ以外では，想定したとおりの減少をしている．
- トークン数が上がるにつれて結合され同じように利用されている部品 (表中の原因部品数) が A に属している数も減少しているが，その減少幅は A の部品数の減少の度合いよりも小さいので，結合され同じように利用されている部品の適合率はトークン数が大きくなることで増加する傾向にある．トークン数が上がるにつれて結合され同じように利用されている部品の一部が A に属さなくなるので，再現率はトークン数が大きくなることで減少している．特に，トークン数が 50 から 70 になったとき，結合され同じように利用されている部品が 6 つ減少しており，再現率が大幅に下がった．適合率と再現率の調和平均である f 値 [7] からは，トークン数を 40 にした場合が最もよい結果となった．

事例 2 : JavaS に対する分析結果

JavaS とは，VoIP 技術を利用したソフトウェアフォンのためのソフトウェアで，表 2 は JavaSIP に対する適用結果で，以下の事が得られた．

- 検出時のトークン数を 30 から 40 に変えた時，A の部品数が増加した．これはトークン数が 30 のときに結合されていた部品が，トークン数が 40 になったとき非結合部品となり，A の部品に分類されたからである．その結果，結合され同じように利用されている部品の数が増え再現率が上がる原因となった．
- A の評価値は，トークン数が上がるにつれて増加する傾向にあるが，B や C よりは減少していた．
- 結合され同じように利用されている部品の適合率はトークン数が大きくなるにつれて上がっている．一方，再現率はトークン数が大きくなることで減少しているが，最大となるのは，前述のとおりトークン数が 40 のときである．f 値からは，トークン数を 50 にした場合が最もよい結果となった．

表 1 JmDNS に対する分析結果

トークン数	30	40	50	70
A の部品数	15	13	11	3
結合部品数	20	12	8	2
A の平均	103 %	90 %	92 %	76 %
B の平均	110 %	103 %	105 %	102 %
C の平均	87 %	102 %	99 %	102 %
原因部品数	10	9	8	2
適合率	50 %	69 %	72 %	66 %
再現率	90 %	81 %	72 %	18 %
f 値	0.64	0.74	0.72	0.28

表 2 JavaSIP に対する分析結果

トークン数	30	40	50	70
A の部品数	26	28	17	16
結合部品数	17	11	5	4
A の平均	89 %	94 %	95 %	97 %
B の平均	102 %	104 %	105 %	104 %
C の平均	98 %	100 %	100 %	99 %
原因部品数	7	8	6	5
適合率	27 %	29 %	35 %	31 %
再現率	88 %	100 %	86 %	71 %
f 値	0.41	0.45	0.50	0.43

事例 3 : goGUI に対する分析結果

goGUI は，囲碁の碁盤表示や操作を行う GUI ソフトで，表 3 は goGUI に対する適用結果で，以下の事が得られた．

- 検出時のトークン数が上がるにつれて A の部品数は減少している．トークン数が 50 から 70 になったとき，結合され同じように利用されている部品が 21 個減少しており，再現率が大幅に下がる原因となった．
- A の評価値は，トークン数 50 まで増加傾向にあったが，トークン数が 70 の場合想定したとおり評価値は減少した．
- 結合され同じように利用されている部品の適合率は

トークン数が大きくなるにつれて増加する．トークン数が 30 から 40 にしたとき結合され同じように利用されている部品が A に属している数も減少しているが，その減少幅は A の部品数の方が大きい．f 値からは，トークン数を 40 にした場合が最もよい結果となった．

表 3 goGUI 分析結果

トークン数	30	40	50	70
A の部品数	53	37	33	3
結合部品数	34	20	16	4
A の平均	92 %	92 %	94 %	90 %
B の平均	100 %	97 %	98 %	97 %
C の平均	97 %	102 %	101 %	100 %
原因部品数	29	27	24	3
適合率	55 %	72 %	72 %	100 %
再現率	100 %	90 %	80 %	10 %
f 値	0.71	0.80	0.76	0.18

5 考察

5.1 分析結果の傾向について

31 種類のプロジェクトに手法を適用した全体の傾向として，以下の事が得られた．

1. 19 種類では条件を満たす部品が存在せず，必ずしも提案手法を実現できるわけではないことがわかる．
2. 全体的にトークン数が上がるにつれて結合された部品の数も減少しており，それに伴い A の部品数は減少している．しかし，トークン数を上げたときに A の部品数が増加することが 4 種類のプロジェクトにみられた．これはトークン数が低い条件では結合された部品が非結合部品となり，分類の対象となったためである．
3. A の評価値はトークン数が大きくなることで減少する傾向にあるが，変化の度合いが小さくなっていくプロジェクトもみられた．その際でも，A の変動率は B，C の変動率より低かった．
4. 結合され同じように利用されている部品の割合はトークン数が大きくなることで増加する傾向にある．これは，A の部品数の減少幅が，同じように利用される部品の減少幅より大きいからである．一方で，結合され同じように利用されている部品の再現率はトークン数が大きくなることで減少する傾向にある．適合率と再現率の平均である f 値の観点からは，最適な場合は 30 から 50 の範囲に混在しており，どのしきい値がよいかについては判断がつかなかった．

5.2 実利用における適切な値

今回の実験では f 値の観点からは，しきい値が 30 から 50 の事例において一番よいケースがみられ，プロジェクト毎にまちまちであった．またトークン数の値を大きくしすぎると，十分な精度が得られない事もわかった．トークン数が大きいほど A の評価値が低下しやすいことを考慮すると，最初のしきい値として 50 などのある程度大きい値を与えた上で必要に応じて条件をゆるくしていき，どう変化していくのかを確認していく方法がよいと思われる．

6 まとめ

本研究では，コードクローン検出時のしきい値で，ある連続したトークン数の長さを変更することで，コンポーネントランク拡張手法の精度がどう変わるかを検証した．結果として，効果のあるプロジェクトにおいてはトークン数のしきい値がある程度の値になるまでは，トークン数が大きくなることで評価値が減少し，検出できた部品中の適合率は増加する一方で，再現率は減少することが分かった．実験結果に基づいて実利用におけるトークン数に関する条件のしきい値を適切に設定することで，手法の精度向上につなげる事ができると考える．今後の課題として他の改善手法の適用を実現することで，さらなる精度の向上をはかる事が求められる．

参考文献

[1] Katsuro Inoue, Reishi Yokomori, Tetsuo Yamamoto, Makoto Matsushita，and Shinji Kusumoto，”Ranking Signi cance of Software Components Based on Use Relations”，Transaction on Software Engineering，vol．31，no．3，pp．213-225，2005.

[2] Toshihiro Kamiya，Shinji Kusumoto，and Katsuro Inoue，”CCFinder: A Multi-Linguistic Tokenbased Code Clone Detection System for Large Scale Source Code”，Transaction on Software Engineering，vol．28，no．7，pp．654-670，2002.

[3] 千賀 英佑：”コードクローンを利用したソフトウェア部品の評価手法についての考察”，南山大学大学院数理情報研究科 2013 年度修士論文，2014.

[4] Classycle: Analysing Tools for Java Class and Package Dependencies，http://classycle.sourceforge.net/

[5] 奥田 黎哉，小林 登夢，村瀬 慶紀：”コードクローンを用いたコンポーネントランク拡張手法の有効性評価 —部品グラフの変化についての分析—”，南山大学情報理工学部 2014 年度卒業論文，2015.

[6] 安藤 正憲，堀江 大河，井田 裕之：”コードクローンを用いたコンポーネントランク拡張手法の有効性評価 —変化の影響を受けた部品について分析 —”，南山大学情報理工学部 2014 年度卒業論文，2015.

[7] 徳永 健伸：”情報検索と言語処理”，東京大学出版会，1999.