テクニカルノート

# コンポーネントランク法による ソフトウェアクラスタリング結果の理解性向上

中塚 剛† 松下 誠† 井上克郎†

巨大なシステムを複数のサブシステムに分解するソフトウェアクラスタリング手法は，大規模なソフトウェアを理解するために有用である．しかし，システム内には，特定のサブシステムに属するべきでないライブラリのような部品も存在する．そのような部品を事前に特定することにより，クラスタリング結果が理解しやすくなると期待できる．本稿では，ソフトウェア部品の重要度を測定するコンポーネントランク法を用いることにより，特定のサブシステムに属するべきではない部品を特定する手法の提案を行う．提案する手法を用いたクラスタリングツールを用いて実験を行い，得られた結果に対する考察を行った．その結果，従来手法よりも優れたクラスタリング結果を得られることが分かった．

# Improvement in Understandability of the Software Clustering Results by Component Rank Model

Gou Nakatsuka,† Makoto Matsushita† and Katsuro Inoue†

Decomposing a large software system into some subsystems helps us understand its structure and functions. In the system, however, there are special modules such as libraries that should not belong to a certain subsystem, and pre-detecting such modules can make the clustering results more understandable. This paper proposes a new method of detecting such modules by Component Rank Model, that is used to measure mutual significance of software modules. We performed experiments with a new clustering tool combined with our detecting method and investigated the clustring results. As a result, our method turned out better than an existent method.

## 1. はじめに

仕様書等が十分でないソフトウェアシステム（以降，システム）の構造や機能の理解を助ける手法の１つとして強凝集かつ低結合の原則に基づくソフトウェアクラスタリングが研究されている[1),2)]．これらの研究では，システムは多くのモジュール（たとえば Java の場合，クラスやインタフェース）から構成されると仮定し，密に依存しあっているモジュールをクラスタとすることによってサブシステムを構築し，サブシステム間の依存関係をできるだけ少なくするという手法が用いられている．

このようなソフトウェアクラスタリングにおいて問題となるのは，システム中に存在する遍在モジュール（omnipresent module）である[3)]．遍在モジュールとは多くのモジュールから直接的あるいは間接的に依存されているモジュールである．遍在モジュールは特定のサブシステムに属するべきでないため[3)]，適切に特定してクラスタリング対象から除去する必要がある．しかし従来手法では，依存関係を持つモジュールの数が閾値以上のモジュールを遍在モジュールとするという単純な手法で特定しており[1),3)]，遍在モジュールに関して詳細な考察は行われていない．

一方，Java ソフトウェア部品検索システム SPARS-J[4)] では，コンポーネントランク法と呼ばれる手法が用いられている．この手法は，あるシステム内のソフトウェア部品の相互利用関係を解析することによって，各ソフトウェア部品の重要度を計算する．ここで上位にランク付けされたクラスは，ソフトウェア中の多くから利用される重要な部品であるが[5)]，これは遍在モジュールの満たすべき性質と似ている．

† 大阪大学大学院情報科学研究科
Graduate School of Information Science and Technology, Osaka University

そこで本稿では，コンポーネントランク法を利用して遍在モジュールを特定する手法を，実験によって評価，決定する．その結果，本手法が従来のソフトウェアクラスタリングアルゴリズムを改良することを示す．

以降，2 章で本研究に関連する 2 つの手法について説明し，3 章で，コンポーネントランク法を用いた遍在モジュール特定手法を決定するための実験と結果について説明する．最後に，4 章でまとめと今後の課題について述べる．

## 2. 関 連 研 究

### 2.1 ソフトウェアクラスタリングシステム Bunch

Bunch[1] は，強凝集・低結合の原則に基づいたクラスタリングを実装したシステムで，NP 困難であるクラスタリングを，局所探索法を用いることで現実的な実行時間で解く．Bunch では，4 種類の遍在モジュールを定めており，それぞれ以下のような特定条件を与えている．

- **supplier**：入次数が平均次数の 3 倍以上
- **client**：出次数が平均次数の 3 倍以上
- **central**：supplier かつ client
- **library**：出次数が 0

本研究は，Bunch におけるこれら 4 種の遍在モジュールを特定するための適切な条件を定めることで，Bunch の出力する解を改良することを目指す．

### 2.2 コンポーネントランク法

Java ソフトウェア部品の再利用を助けるために，部品検索システム SPARS-J が開発されている[4]．このシステムは，与えられた検索単語に対して関連する Java ソフトウェア部品を提示する．その際，ソフトウェア部品を提示する順番を決定する手法として，コンポーネントランク法が用いられる．

コンポーネントランク法では，各モジュールの直接的あるいは間接的な利用関係に基づいて，総和が 1 となるようにコンポーネントランク値（CRV）を振り分ける．この際，たくさんの部品から使用されている部品の CRV は高くなり，また，CRV の高い部品から使用されている部品の CRV も高くなる．

本稿では，「CRV の高い部品は，多くの部品に直接的あるいは間接的に使用されている部品であり，遍在モジュールの候補ではないか」，という点に注目して，CRV を用いた遍在モジュール特定手法を提案する．

## 3. 遍在モジュール特定手法

Bunch で定義されている 4 種類の遍在モジュールを CRV，次数によって特定するために，どのような条件を用いれば解を改良できるか，検証実験を行う．ここでは 40 の特定条件を定義し，各条件がどのような結果を導くかを比較するために 2 種類の実験を行い，最適な条件を決定する．

### 3.1 実験対象となる特定条件

本実験では，**表 1** に示すように C1〜C40 の特定条件を実験対象として定めた．ここで in，out，CRV の列はそれぞれ入次数，出次数，CRV の条件を表し，数字は平均値からの倍率を表している．数字が正の場合，たとえば 3 の場合は平均値の 3 倍以上ならば該当とする．ただし，数字が 0 の場合は，値が 0 のときのみ該当とする．

C1 は Bunch で用いられている条件であり，C2 は遍在モジュールを考慮していない．C3〜C6 は，CRV を用いておらず，C7 以降は，supplier，central，library の条件に CRV を使用している．この際，CRV が平均を超えるモジュールは全体の約 20%と少ないため，CRV の条件としては平均以上のみを考慮して，次数

**表 1** 遍在モジュール特定条件（数字は平均からの倍率）
Table 1 Conditions for omnipresent modules.

| No. | supplier | | client | central | | | library | | |
|---|---|---|---|---|---|---|---|---|---|
| | in | CVR | out | in | out | CVR | in | out | CVR |
| C1 | 3 | | 3 | 3 | 3 | | | 0 | |
| C2 | | | | | | | | | |
| C3 | 2 | | 2 | 2 | 2 | | | 0 | |
| C4 | 2.5 | | 2.5 | 2.5 | 2.5 | | | 0 | |
| C5 | 3.5 | | 3.5 | 3.5 | 3.5 | | | 0 | |
| C6 | 3 | | 3 | 3 | 3 | | | | |
| C7 | | 1 | 3 | | 3 | 1 | | 0 | |
| C8 | | 1 | 3 | | 3 | 1 | | | |
| C9 | | 1 | 3 | | 3 | 1 | | 0 | 1 |
| C10 | | 1 | 3 | | 3 | 1 | 1 | 0 | 1 |
| C11 | | 1 | 3 | 3 | 3 | 1 | 1 | 0 | 1 |
| C12 | 1 | 1 | 3 | 1 | 3 | 1 | | 0 | |
| C13 | 1 | 1 | 3 | 1 | 3 | 1 | | | |
| C14 | 1 | 1 | 3 | 1 | 3 | 1 | | 0 | 1 |
| C15 | 1 | 1 | 3 | 1 | 3 | 1 | 1 | 0 | 1 |
| C16 | 1 | 1 | 3 | | 3 | 1 | 1 | 0 | 1 |
| C17 | 1 | 1 | 3 | 3 | 3 | 1 | 1 | 0 | 1 |
| C18 | 1.5 | 1 | 3 | 1.5 | 3 | 1 | | 0 | |
| C19 | 1.5 | 1 | 3 | 1.5 | 3 | 1 | | | |
| C20 | 1.5 | 1 | 3 | 1.5 | 3 | 1 | | 0 | 1 |
| C21 | 1.5 | 1 | 3 | 1.5 | 3 | 1 | 1 | 0 | 1 |
| C22 | 1.5 | 1 | 3 | | 3 | 1 | 1 | 0 | 1 |
| C23 | 1.5 | 1 | 3 | 3 | 3 | 1 | 1 | 0 | 1 |
| C24 | 2 | 1 | 3 | 2 | 3 | 1 | | 0 | |
| C25 | 2 | 1 | 3 | 2 | 3 | 1 | | | |
| C26 | 2 | 1 | 3 | 2 | 3 | 1 | | 0 | 1 |
| C27 | 2 | 1 | 3 | 2 | 3 | 1 | 1 | 0 | 1 |
| C28 | 2 | 1 | 3 | | 3 | 1 | 1 | 0 | 1 |
| C29 | 2 | 1 | 3 | 3 | 3 | 1 | 1 | 0 | 1 |
| C30 | 2.5 | 1 | 3 | 2.5 | 3 | 1 | | 0 | |
| C31 | 2.5 | 1 | 3 | 2.5 | 3 | 1 | | | |
| C32 | 2.5 | 1 | 3 | 2.5 | 3 | 1 | | 0 | 1 |
| C33 | 2.5 | 1 | 3 | 2.5 | 3 | 1 | 1 | 0 | 1 |
| C34 | 2.5 | 1 | 3 | | 3 | 1 | 1 | 0 | 1 |
| C35 | 2.5 | 1 | 3 | 3 | 3 | 1 | 1 | 0 | 1 |
| C36 | 3 | 1 | 3 | 3 | 3 | 1 | | 0 | |
| C37 | 3 | 1 | 3 | 3 | 3 | 1 | | | |
| C38 | 3 | 1 | 3 | 3 | 3 | 1 | | 0 | 1 |
| C39 | 3 | 1 | 3 | 3 | 3 | 1 | 1 | 0 | 1 |
| C40 | 3 | 1 | 3 | | 3 | 1 | 1 | 0 | 1 |

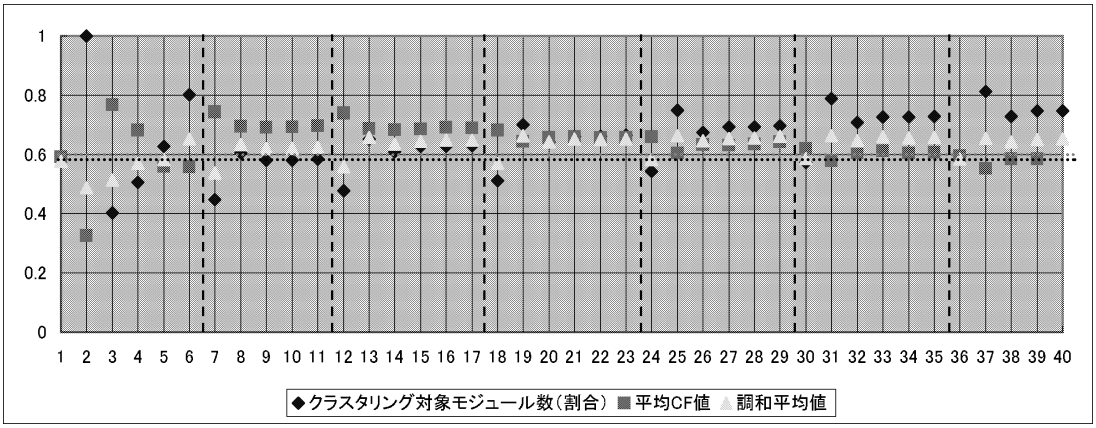


**図 1** 実験 1 の結果
Fig. 1 Result for experiment 1.

条件を組み合わせることで条件を厳しくしている．各遍在モジュールに対して条件を定め，その直積の一部を条件とした．

### 3.2 実験 1：結果の理解しやすさによる評価

Bunch では，各クラスタの凝集度と結合度のトレードオフを $CF$（Cluster Factor）という関数で表している．$CF$ は，各クラスタに関連する辺のうち，クラスタ内部にある辺の割合を表し，0〜1 の値をとる．つまり，$CF$ 値が高いほどクラスタ間に辺が少なくなり，結果が理解しやすいものとなる．すなわち，平均 $CF$ 値（全クラスタの $CF$ 値の平均値）が高い方が全体の結果が理解しやすいといえる．

一般的に，遍在モジュールを多く特定するほど，平均 $CF$ 値が高くなりやすい．この場合，クラスタリングされたモジュールは理解しやすいといえるが，遍在モジュールの数が増えるため，個別に理解する必要があるモジュール数が増加することになる．この結果，全体として理解にかかるコストが増加してしまう場合がある．よって，平均 $CF$ 値よりも，クラスタリング対象モジュール数（遍在モジュールではないモジュールの数）を重視し，2 つの値がともに高い条件が優れているとする．

本実験では，15 種類の Java で書かれたオープンソースシステムに対して 40 の特定条件を適用した．

**結果と考察**

15 システムに適用した結果の平均値を**図 1** に示す．横軸は 40 の条件を表し，縦軸は，クラスタリング対象モジュール数の全モジュール数に対する割合，平均 $CF$ 値と両者の調和平均値を示している（一部，記号が重なっている）．

左端の C1 の結果が Bunch オリジナル条件での結果であり，C2 と比べると，平均 $CF$ 値が約 2 倍に増加した一方，遍在モジュール数が多過ぎることが分かる．なお，この結果が比較の基準となるため，横軸に対して平行に破線で示している．

C1，C3〜C5 の結果を見ると，クラスタリング対象モジュール数と平均 $CF$ 値がトレードオフの関係となっていることが分かる．また，C6 の結果は，出次数が 0 という条件によって library がいかに多く選ばれていることを示している．

C7 以降の結果は，C1 に比べて，2 つの評価値がともに増加している条件が多数存在している．supplier に対する条件が厳しくなるにつれ，クラスタリング対象モジュール数は増加し，逆に平均 $CF$ 値が減少している．つまり，トレードオフとなっているが，調和平均値が改善していることが分かる．クラスタリング対象モジュール数，平均 CF 値がともに C1 の結果より高い結果の中で，前述のように，クラスタリング対象モジュール数を重視すると，C27〜C29 の条件が優れているといえる．

### 3.3 実験 2：ベンチマークによる評価

クラスタリング結果構造を評価するためによく用いられる手法として，適切と考えられる解をベンチマークとして作成して結果と比較するという手法がある[6),7)]．本実験では，実験 1 で用いたシステムの 1 つである GNUJpdf（25 モジュール）に対してベンチマークを作成し，結果と比較を行った．

ここでは，ベンチマークと解を比較するために，EdgeSim と MeCl という 2 種類のメトリクス[7)] を用いた．EdgeSim は，辺の類似度を測定するメトリクスで，MeCl は，片方のグラフを分割した後マージすることで，もう片方のグラフと一致させるために必要なコストを測定するメトリクスである．ともにパーセント尺度で，値が高いほど 2 つのグラフが類似して

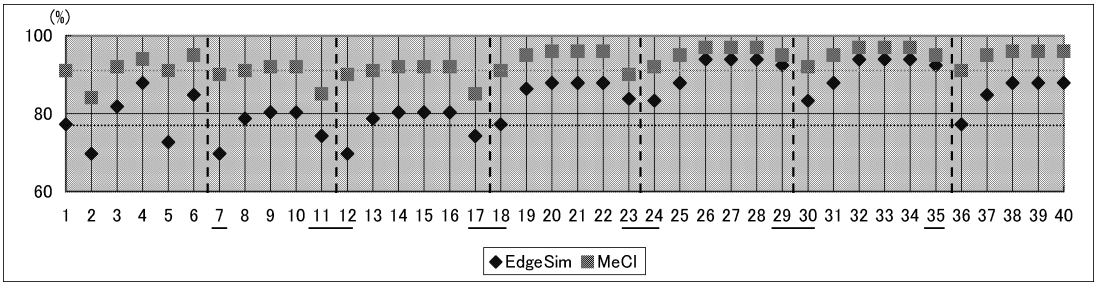


**図 2** 実験 2 の結果
Fig. 2 Result for experiment 2.

いることを意味する．ただし，これらは遍在モジュールを考慮しないため，正しい評価が行えない場合がある．そのため，目視評価によってその点を補う必要がある．

**結果と考察**

実験 2 の結果を**図 2** に示す．横軸は，40 の条件を表している．横軸の値の一部に付けられた下線は，遍在モジュールの選び方によって Bunch の解に一部のモジュールが存在しないというバグのため，解を一部書き換えて測定したことを表している．縦軸は，EdgeSim，MeCl を表している．両メトリクスの相関係数は 0.88 と非常に高い．

この結果から，EdgeSim，MeCl ともに値が改善されている条件が多数存在することが分かる．実験 1 で結果の良かった C27〜C29 は，実験 2 でも非常に良い結果を示している．ただし，C29 は，前述の Bunch のバグのため，正しい解を出力していない．なお，GNUJpdf はモジュール数が少ないため，同じクラスタリング結果を出力する条件があり，図 2 で同じ値を示しているものが見られる．

### 3.4 実験のまとめ

GNUJpdf では C27，C28 が優れた結果を示しているが，central の条件が supplier かつ client でない場合に不適切な結果を出力しうることがあった．そこで，C28 は不適切となる．よって，定義した 40 の条件の中では，以下に示す C27 が，最も Bunch のアルゴリズムを改善することが分かった．

- **supplier**：入次数が平均次数の 2 倍以上，かつ CRV 平均以上
- **client**：出次数が平均次数の 3 倍以上
- **central**：supplier かつ client
- **library**：入次数が平均以上，かつ出次数が 0，かつ CRV 平均以上

## 4. むすび

本稿では，コンポーネントランク法によって測定されるコンポーネントランク値を用いて遍在モジュールを特定し，高凝集・低結合の原則に基づくソフトウェアクラスタリングを改良できることを確認した．

今後の課題としては，より大規模なシステムでのベンチマーク実験による評価があげられる．

## 参考文献

1) Mitchell, B.S. and Mancoridis, S.: On the Automatic Modularization of Software Systems Using the Bunch Tool, *IEEE Trans. Softw. Eng.*, Vol.32, No.3, pp.193–208 (2006).
2) Schwanke, R.W.: An Intelligent Tool For Re-engineering Software Modularity, *Proc. Int'l Conf. on Softw. Eng.*, pp.83–92 (1991).
3) Müller, H.A., Orgun, M.A., Tilley, S.R. and Uhl, J.S.: A Reverse Engineering Approach To Subsystem Structure Identification, *Journal of Softw. Maintenance: Research and Practice*, Vol.5, No.4, pp.181–204 (1993).
4) Inoue, K., Yokomori, R., Yamamoto, T., Matsushita, M. and Kusumoto, S.: Ranking Significance of Software Components Based on Use Relations, *IEEE Trans. Softw. Eng.*, Vol.31, No.3, pp.213–225 (2005).
5) 市井 誠，横森励士，松下 誠，井上克郎：コンポーネントランクを用いたソフトウェアのクラス設計に関する分析手法の提案，電子情報通信学会技術研究報告，SS2005-37, Vol.105, No.229, pp.25–30 (2005).
6) Wen, Z. and Tzerpos, V.: An effectiveness measure for software clustering algorithms, *Proc. Int'l Workshop on Program Comprehension* (*IWPC '04*), pp.194–203 (2004).
7) Mitchell, B.S. and Mancoridis, S.: Comparing the Decompositions Produced by Software Clustering Algorithms using Similarity Measurements, *Proc. Int'l Conf. on Softw. Maintenance*, pp.744–753 (2001).

(平成 19 年 4 月 4 日受付)
(平成 19 年 6 月 5 日採録)

**中塚　剛**

平成 19 年大阪大学大学院情報科学研究科コンピュータサイエンス専攻博士前期課程修了．在学中，ソフトウェアクラスタリングの研究に従事．修士（情報科学）．

**松下　誠**（正会員）

平成 5 年大阪大学基礎工学部情報工学科卒業．平成 10 年同大学大学院博士後期課程修了．同年同大学基礎工学部情報工学科助手．平成 14 年大阪大学大学院情報科学研究科コンピュータサイエンス専攻助手．平成 17 年同専攻助教授，平成 19 年同専攻准教授．博士（工学）．

**井上　克郎**（正会員）

昭和 54 年大阪大学基礎工学部情報工学科卒業．昭和 59 年同大学大学院博士課程修了．同年同大学基礎工学部情報工学科助手．昭和 59〜61 年ハワイ大学マノア校情報工学科助教授．平成元年大阪大学基礎工学部情報工学科講師．平成 3 年同学科助教授．平成 7 年同学科教授．平成 14 年大阪大学大学院情報研究科コンピュータサイエンス専攻教授．工学博士．