修 士 論 文

題　目　　ソースコードにおける
　　メソッドの抽出的要約手法の提案

主任指導教員　　水野　修　教授

京都工芸繊維大学大学院 工芸科学研究科

情報工学専攻

学生番号　　17622014

氏　　名　　小林　勇揮

平成30年2月8日提出

様式 2 号

学位論文の要旨（和文）

平成 30 年 2 月 8 日

京都工芸繊維大学大学院
工芸科学研究科長　殿

工芸科学研究科　　情報工学専攻
平成 29 年入学
学生番号　　17622014
氏　　名　　小林　勇揮　㊞

（主任指導教員　　水野　修　㊞　）

本学学位規則第 4 条に基づき、下記のとおり学位論文内容の要旨を提出いたします。

1.　論文題目

ソースコードにおけるメソッドの抽出的要約手法の提案

2.　論文内容の要旨（400 字程度）

ソフトウェアの大規模化に伴ってソフトウェアが複雑化することで様々な問題が起こる．例えば，プログラマがプログラムに変更を加える際にソースコードを書くよりも読んだり眺めたりすることに時間を費やす傾向がある．このソースコードの読解にかかる時間を短縮するために自然言語処理の技術を利用して，ソースコードを要約する研究が行われている．その中でソースコード内にある重要なメソッドを抽出し，ソースコードの折り畳み機能を用いて，ソースコードを要約する方法に着目した．本研究では，プログラマがソースコードを要約する際の特徴を参考に，既存の自然言語処理の技術である NgramIDF と LexRank を応用したソースコードにおける新しいメソッドの抽出的要約手法を提案する．情報工学専攻の学生 9 名による実験により，Github 上にある 5 つのプロジェクトにおいて，提案手法が他の既存手法よりも被験者が重要だと考えるメソッドを抽出できるという点で優れていることを示した．さらに被験者にソースコードの要約を提示したところ，提案手法による要約でソースコードを読みやすく感じる学生が多く，ソースコードの概念的な理解に繋がったと考えられる．以上の実験結果より，提案したソースコード要約手法によって，プログラム理解に貢献できる可能性を示したと考える．

Proposal of source code summarization for extracting methods

2018 17622014 *KOBAYASHI Yuki*

Abstract

Large software has various problems due to the inherent complexity of software. One problem is that programmers tend to spend much time for reading source code. To reduce time to read source code, there is much research using natural language processing techniques to summarize source code. Among them, we focused on the source code summarization using source code folding functions. In our research, we propose a new source code summarization method to extract methods in source code using the NgramIDF and the Lexrank considering the characteristics when the programmer summarizes source code. In the five projects on Github, experiment showed that our proposed method was superior to other existing methods in extracting the methods that programmers consider important. Furthermore, we presented a summary of the source code to nine master course students in the department of information science. We found that many students felt that the source code was easy to read due to the summary by our proposed method, and we conclude that our method help them to gain a conceptual understanding of the source code. From the result of experiment, we believe that the source code summarization by our proposed method can demonstrate the contribution to the program comprehension.

目　次

1. 緒言

近年，ソフトウェアの大規模化に伴って，ソースコードの行数の増加やソースコードのファイル数が増加することにより，開発者がプログラムに変更を加える際にソースコードを書くよりも読んだり眺めたりすることに時間を費やす傾向がある [1]．また，プログラム理解の研究によると，開発者はソフトウェア保守の際にコードの小さな部分に焦点を当て，システム全体を把握することを避けようとする [2]．そのため，しばしば開発や保守の期間に，開発者はコードを素早く理解するために流し読みをする [3]．このことから開発者のプログラム理解を支援するために，自然言語処理の技術を利用して，ソースコードを要約する，つまりソースコードからプログラム理解に繋がるコードを抽出するという研究が行われている．

例えば，大場らによる TF-IDF を応用したプログラム中に存在するコンセプトキーワードの抽出といった研究 [4] や，Haiduc らによるベクトル空間モデル (VSM) を用いた自動テキスト要約に基づくアプローチによるソースコード要約 [5]，Ying らによる機械学習を利用したコード断片の強調による要約 [6]，Fowkes らによるエディタのコード折り畳み機能を利用したメソッド単位のトピックモデルによるソースコード要約 [7] など，現在に至るまで様々な研究が行われている．

筆者はこの中で一番実現性の高いエディタのコードの折り畳み機能を利用したソースコード要約に着目してソースコードの要約を考えた．コードの折り畳み機能は多くのエディタや IDE の機能に含まれており，開発者によってソースコードを展開するか，折り畳むかを選択することができる．しかし，現在のコードの折り畳み機能は，コードの折り畳みの有無を開発者の手作業によって決められるため実用的ではない．また，開発者は何を折り畳むべきなのかを決定するために，ソースコードを理解していなければならないので鶏卵問題が発生する．そのため，本研究では自然言語処理の技術を応用して折り畳むべきコードを自動的に選択し，ユーザに提示することでこの問題を解決する．例えば，エディタの機能を利用して，図 1.1 のような折り畳まれていないコードを図 1.2 のような折り畳まれているコードにすることでソースコード要約を達成する．

次に，この要約を実現するためにメソッド単位のソースコードの要約を行う．ソースコードを要約するためにはソースコードから特徴量を得る必要があり，その際に

```
66      @Override
67      public void execute() throws BuildException {
68          File savedFile = myFile; // may be altered in validate
69
70          validate();
71
72          final Properties properties = loadProperties();
73          final int buildNumber = getBuildNumber(properties);
74
75          properties.put(DEFAULT_PROPERTY_NAME,
76              String.valueOf(buildNumber + 1));
77
78          // Write the properties file back out
79
80          try (OutputStream output = Files.newOutputStream(myFile.toPath())) {
81              properties.store(output, "Build Number for ANT. Do not edit!");
82          } catch (final IOException ioe) {
83              throw new BuildException("Error while writing " + myFile, ioe);
84          } finally {
85              myFile = savedFile;
86          }
87
88          //Finally set the property
89          getProject().setNewProperty(DEFAULT_PROPERTY_NAME,
90              String.valueOf(buildNumber));
91      }
92
93      private int getBuildNumber(final Properties properties) throws BuildException {
94          final String buildNumber =
95              properties.getProperty(DEFAULT_PROPERTY_NAME, "0").trim();
96
97          // Try parsing the line into an integer.
98          try {
99              return Integer.parseInt(buildNumber);
100         } catch (final NumberFormatException nfe) {
101             throw new BuildException(
102                 myFile + " contains a non integer build number: " + buildNumber,
103                 nfe);
104         }
105     }
```

図 1.1　折り畳まれていないソースコード．66 行から 105 行までのコードが表示されている．ここには表示されてないが 105 行以降に 2 つのメソッドがある．

```
66      @Override
67      public void execute() throws BuildException {
68          File savedFile = myFile; // may be altered in validate
69
70          validate();
71
72          final Properties properties = loadProperties();
73          final int buildNumber = getBuildNumber(properties);
74
75          properties.put(DEFAULT_PROPERTY_NAME,
76              String.valueOf(buildNumber + 1));
77
78          // Write the properties file back out
79
80          try (OutputStream output = Files.newOutputStream(myFile.toPath())) {
81              properties.store(output, "Build Number for ANT. Do not edit!");
82          } catch (final IOException ioe) {
83              throw new BuildException("Error while writing " + myFile, ioe);
84          } finally {
85              myFile = savedFile;
86          }
87
88          //Finally set the property
89          getProject().setNewProperty(DEFAULT_PROPERTY_NAME,
90              String.valueOf(buildNumber));
91      }
92
93      private int getBuildNumber(final Properties properties) throws BuildException {...
105     }
106
107     private Properties loadProperties() throws BuildException {...
115     }
116
117     private void validate() throws BuildException {...
139     }
140 }
```

図 1.2　折り畳まれたソースコード．66 行から 140 行までのコードが表示されている．94 行から 104 行，106 行から 114 行，116 行から 138 行が折り畳まれている．

ソースコードの要約を生成するために適切な特徴量を付ける必要がある．よって，プログラマがソースコードを要約する際の特徴を利用してメソッドの要約を行いたいと考えた．Rodegheroらによるアイトラッキングを利用した研究 [8] によると，プログラマがソースコードを要約する際に眺める時間が長かったものにメソッドのシグネチャとメソッドの呼び出しがある．そこで本研究ではメソッドのシグネチャとメソッドの呼び出しに着目してメソッドの要約を行った．この2つを利用するために，まずメソッドのシグネチャ，特にプログラマの意図が入りやすいメソッド名に対してTF-ngramIDF[9] を利用して各メソッドを重み付けした．次に，その各メソッドの呼び出し関係よりメソッドコールグラフを作成することで，既存の自然言語処理の技術であるLexRank[10] を応用し，メソッドの要約を行った．また，メソッドを折り畳むか折り畳まないかの選択は0-1ナップサック問題を参考にすることで解決した．

本報告の以降の構成は次のとおりである．まず第2章で本研究で利用する語の重み付け手法や抽出的要約手法について説明する．次に第3章でメソッドの抽出的要約についての説明や本研究で提案するソースコードの要約手法について詳しく説明する．次に第4章で研究設問や実験で使うプロジェクトの説明，被験者による評価実験の説明をし，第5章で第4章で設けた研究設問に沿って，実験結果および考察を述べる．そして，第6章で本研究の実験結果や実験方法における妥当性への脅威について述べる．最後に，第7章で実験方法や実験結果，今後の課題について簡潔に述べる．

2. 準備

2.1 語の重み付け手法

語の重み付け手法 (term weighting scheme) はテキスト解析において重要な技術である．代表的なものとして TF-IDF (Term Frequency-Inverse Document Frequency) がある．TF-IDF は，文書中に出現する特定の単語がどのくらい特徴的であるかを識別するための指標で，情報検索 [11]，文書クラスタリング [12]，特徴語抽出 [13]，映像中のオブジェクトマッチング [14] など，幅広いアプリケーションで利用することができる．また，単語と文書の共起性や情報検索の確率モデルなどの観点 [15][16][17] から理論的説明が与えられており，多くの人が TF–IDF を利用する根拠となっている．本研究では，この語の重み付け手法をソースコードに対して利用することにした．次の節で，TF-IDF について詳しく説明していく．

2.1.1 TF-IDF

TF-IDF のような語の重み付け手法は，一般的に，局所的重み付けと大域的重み付けの二つの要素からなる．局所的重み付けは，ある文書に対する語の出現頻度から計算され，対象とする文書によって重みは変化する．一方，大域的重み付けは，文書集合全体における語の文書頻度から計算され，語の重みは対象とする文書によらず一定である．TF-IDF は具体的には，次の式（2.1）により語 t の文書 $d \in D$ における重みを計算する．

$$TF\text{-}IDF(t,d) = tf(t,d) \cdot \log \frac{|D|}{df(t)} \tag{2.1}$$

ここで，$tf(t,d)$ は文書 d における語 t の出現頻度，$df(t)$ は文書集合 D における t の文書頻度，$|D|$ は D の文書数である．

この式（2.1）の右辺は，局所的重み付けと大域的重み付けの二つに分けることができる．局所的重み付けは次の式（2.2）で表される．

$$TF(t,d) = tf(t,d) \tag{2.2}$$

また，大域的重み付けは次の式（2.3）で表される．

$$IDF(t) = \log \frac{|D|}{df(t)} \tag{2.3}$$

特に，大域的重み付けである IDF[18] は様々な語の重み付け手法で採用されている．IDF が様々な語の重み付け手法として採用されている理由として，式（2.3）のような簡潔さとロバスト性が挙げられる．実際に，IDF は様々な文献 [15][16][19] において理論的にロバストであることが示されている．

2.1.2 NgramIDF

従来の IDF の欠点として，連続した N 個の単語の組み合わせである単語 Ngram に対して適用できないことがあった．単語 Ngram に対する IDF は，その連続する単語の繋がりが不自然であるほど大きい重み付けをしてしまっていた．例えば,「Osaka University」と「Osaka be」では単語の繋がりが不自然な「Osaka be」の方が文書に出現する割合が低くなるため,「Osaka be」の方が大きい重み付けをしてしまう．しかし，大阪大学大学院情報科学研究科の白川真澄らによる研究 [9] により，単語 Ngram に対しても IDF を適用することができることが発見された．この手法は，文字列が出現する Web ページをシャノン・ファノ符号によって表現したとき，空文字からの情報距離が対象の文字列の IDF となることを利用している．具体的には，次の式（2.4）によって単語 Ngram g に対する IDF を計算する．

$$IDF_{N\text{-}gram}(g) = \log \frac{|D| \cdot df(g)}{df(\theta(g))^2} \tag{2.4}$$

ここで，$|D|$ は D の文書数，$df(g)$ は文書集合 D における g の文書頻度，$\theta(g)$ は g を構成する各単語の論理積，$df(\theta(g))$ は文書集合 D における $\theta(g)$ を満たす文書数である．また，各単語の論理積とは，ある文書において g を構成する各単語が含まれていることである．

2.2 抽出的要約手法

抽出的要約手法とは，文書を要約する際に用いられる手法の一つである．一般的に文書を要約する方法には二つあり，抽出的要約手法と生成的要約手法が考えられる．生成的要約手法とは，要約する文書で用いられている文や単語だけでなく，様々な言語表現を用いて要約を生成する手法である．対して，抽出的要約手法とは，要約する文書の一部（文や単語など）を選択して並べることで要約を生成する手法で

ある．抽出的要約手法では，要約に含める文書の一部を選択するために TF-IDF などの語の重み付け手法を利用することがある．例えば，VSM (Vector Space Model) を利用した要約手法であったり，LexRank と呼ばれる要約手法などがある．次の節でこれらについて説明する．

2.2.1 VSM

VSM (Vector Space Model) とは，情報検索における標準的な手法であり，文書は TF-IDF などを用いたベクトルで表現され，類似度はコサイン類似度などの測定基準によって計算される [20][21]．VSM を抽出的要約に適用するために，文書全体のベクトルと文書の一部のベクトルを比べて，最も類似度の高いものを要約に含める．具体的には，ある文書 d のベクトルを v_d とし，潜在的な要約 u のベクトルを v_u とすると，要約 u の類似度 $\sigma(u)$ について次のような式（2.5）ができる．

$$\sigma(u) = csim(v_d, v_u) \tag{2.5}$$

ここで $csim$ とはコサイン類似度であり，TF-IDF を利用して次のような式（2.6）で求めることができる．

$$csim(x, y) = \frac{\sum_{\omega \in x,y} tf(\omega, x) \cdot tf(\omega, y) \cdot (idf(\omega))^2}{\sqrt{\sum_{x_i \in x} (tf(x_i, x) idf(x_i))^2} \cdot \sqrt{\sum_{y_i \in y} (tf(y_i, y) idf(y_i))^2}} \tag{2.6}$$

もし潜在的な要約 u の集合があれば，$\sigma(u)$ が最大化する u を最適な要約として選ぶことができる．

2.2.2 LexRank

LexRank[10] とは，文単位の相対的重要性を計算するための確率的グラフベースな要約手法であり，Page と Brin による PageRank[22] と呼ばれる Web ページの重要度を決定するアルゴリズムに着想を得て考えられた．要約方法としては，要約する文書の各文を TF-IDF によりベクトル化し，各文のコサイン類似度を求め，その値がある閾値を越えるとその文同士をエッジで繋ぎグラフを作成する．これにより隣接行列を生成できる．次にその隣接行列の各要素をそれぞれのエッジの数で割ることで状態遷移確率を求める．最後に，マルコフ連鎖の定常分布を用いて，その状態遷移確率の定常分布を求めることで文単位の相対的重要性を計算する．

3. 提案手法

3.1 メソッドの抽出的要約

メソッドの抽出的要約とは，ソースコードをメソッドの集合であると考えて，そのメソッド集合からそのソースコード全体を要約しているメソッド（以下，要約メソッド）を選択するものである．また，プログラマに対して要約メソッドを提示する際には，Fowkes らによる研究 [7] を参考にして，ソースコードを表示するエディタのコードの折り畳み機能を利用して実現する．具体的には図 1.1 から図 1.2 のように要約メソッドを展開し，それ以外のメソッドを折り畳む．

ここで，ソースコードは一般的にメソッド名や変数名などを識別するために識別子と呼ばれる英字や数字からなる文字列で構成される．また，識別子はプログラムの可読性を上げるために，単一または複数の単語から構成されることがある [23][24][25]．さらに，ソースコードにはコメントと呼ばれるプログラムの説明や補足をするための文字列も含まれる．このようにソースコードは記号だけではなく，自然言語からも構成されており，実際にソースコードを読んでみると，そのような自然言語の方が大半を占めていることがわかる．よって，ソースコードに対して自然言語処理の技術を利用することは可能であると考える．

また，ソースコードはコードの冗長性を避けるためや機能性の向上，保守性の向上のために複数のメソッドから構成されることが多く，メソッドごとに機能がまとまっていることが多い．よって，ソースコードをメソッドの集合として考えることもできる．

したがって，本研究では，自然言語処理の技術をソースコードに対して適用し，ソースコードをメソッドの集合であると考え，そのメソッド集合から要約メソッドを選択する手法を提案する．また関連研究とは異なる点として，自然言語処理の技術を利用する際にプログラマがソースコードを読むときの特徴を利用する点である．Rodegero らによるプログラマのソースコードのメソッドを要約する際のアイトラッキングの研究 [8] によると，プログラマはソースコードを要約する際にメソッドのシグネチャ，メソッドの呼び出し，制御文の順に注視していることがわかった．本研究ではこの結果から，メソッドのシグネチャとメソッドの呼び出しに着目して，自然

言語処理の技術を応用した新しいソースコードの要約手法を提案する．

3.2 提案手法の解説

初めに，メソッドのシグネチャとメソッドの呼び出しをどのように自然言語処理の技術に応用していくのかを説明する．

メソッドのシグネチャ

まずメソッドのシグネチャとは，プログラムにおけるメソッドの宣言部のことを指す．例えば，プログラミング言語の Java においては，返り値の型，メソッド名，パラメータ名，その型などから構成される．この中で，プログラマの意図が多く反映されるものにメソッド名がある．メソッド名は識別子であり，メソッドの処理を説明するために複数の単語を組み合わせて命名されることが多い．実際に，1 文字や略語で構成される識別子よりも複数の辞書語で構成された識別子の方が理解しやすい [23][24] という研究結果があるように，メソッド名などの識別子を複数の語で構成することはソースコードの可読性及び保守性の向上の観点から多くのプロジェクトで使われている．また，識別子を複数の語で構成する際に，単語ごとの可読性を上げるためにいくつかの命名記法がある．代表的なものとしては，キャメルケース記法とアンダースコア記法がある．キャメルケース記法とは，単語の先頭を大文字にして複数の単語を繋げる記法である．次に，アンダースコア記法とは，単語の間にアンダースコアを入れることで単語を繋げる記法である．このようにメソッド名のような識別子は複数の単語で構成され，また命名記法から各単語を分割することも可能である．よって，語の繋がりを考慮できる語の重み付け手法である NgramIDF を利用できると考えた．

メソッドの呼び出し

次にメソッドの呼び出しとは，プログラムで定義されたメソッドを呼び出すことである．このメソッド呼び出しを利用して，メソッドコールグラフと呼ばれるメソッドをノード，呼び出し関係をエッジとしたグラフを作ることができる．実際にこのメソッドコールグラフの特徴を利用した研究もある．例えば，Kato らによる研究 [26] では，メソッドコールグラフとグラフカットの技術を利用して，あるシナリオに基づく特徴的なメソッドを抽出する手法を提案している．本研究では，LexRank の文

間の類似度によりグラフを作成する処理をメソッドコールグラフに置き換えて，メソッドの抽出的要約を行う．

以上のことを踏まえて，提案するメソッドの抽出的要約の方法について説明する．要約対象は，あるプロジェクトに含まれるあるソースコードとする．以下のような手順で要約を行う．

1. 対象のプロジェクトからコーパスの生成
2. 対象のソースコードから AST (Abstract Syntax Tree) の解析によりメソッド名とメソッドの呼び出し関係，範囲，LoC (Lines of Code) の取得
3. コーパスとメソッド名から TF-NgramIDF の計算
4. メソッドの呼び出し関係から隣接行列の生成
5. 3. と 4. の結果から状態遷移確率の生成
6. マルコフ連鎖の定常分布により要約メソッドを生成
7. 要約メソッドと各メソッドの範囲と LoC から折り畳むメソッドを選択

図 3.1 に手順の概要を示す．

3.2.1 コーパスの生成

要約対象のプロジェクトに属する全てのソースコードに対して，コーパスを生成する．コーパスに含めるのはソースコードに含まれる英字や数字からなる文字列や識別子とし，複数の単語からなる識別子は命名記法により各単語に分割する．分割する際に利用した命名記法はキャメルケース記法とアンダースコア記法である．例えば「getValue」や「get_value」などの識別子は「get」と「value」に分割する．また，「=」や「(」などの記号はストップワードとして取り除いた．

3.2.2 メソッドのランクの生成

AST 解析により対象のソースコード s のメソッド名 m を取得できたら，対象のプロジェクトに属するソースコードの集合 S から求めたコーパスを利用し，各メソッド名の特徴量を TF と NgramIDF を掛けた次の式（3.1）で計算する．

$$TF\text{-}IDF_{N\text{-}gram}(m) = tf(m,s) \cdot \log \frac{|S| \cdot df(m)}{df(\theta(m))^2} \tag{3.1}$$

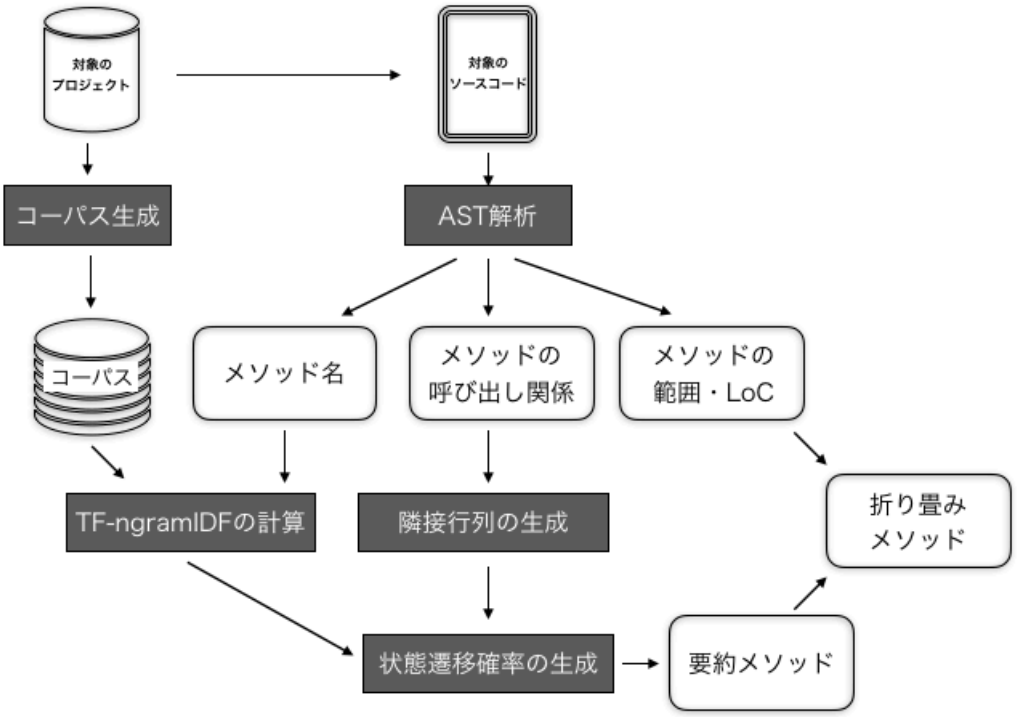

図 3.1　メソッドの抽出的要約の手順

ここで，NgramIDF に TF を掛けた理由は，メソッド名の出現回数が多いほど，そのメソッドが利用されていることに繋がり重要なメソッドと考えることができるからである．

また，AST 解析により取得したメソッド呼び出し関係から隣接行列を生成し，その隣接行列のエッジの重みを式（3.1）により決定する．さらに隣接行列の生成については以下の追加の処理が 2 つある．

- 1 つ目は，再帰的なメソッドであるなしに関わらず自ノードに遷移するエッジを加えることである．隣接行列のエッジの重み付けは呼び出されるメソッド名の特徴量から求められるために，呼び出しを行うメソッドの重み付けは評価されない．よって，呼び出しを行うメソッドのノードの評価を行いたいため，自ノードに遷移する確率を追加した．
- 2 つ目は，全てのメソッドへのエッジを持つノードを追加することである．これはソースコードによってはメソッド呼び出しを実行しないものがあるため，その際にメソッドコールグラフを作成するとグラフが非連結グラフになり，マルコフ連鎖の定常分布により要約メソッドがうまく求めることができない．そのため，全てのメソッドへのエッジを持つノードを追加することでこれを回避する．また，追加ノードに限っては各メソッドから追加ノードへのエッジの重みを求める．追加ノードへのエッジの重みは，ノードを追加することによる影響を少なくするためにエッジの取りうる値の最小値とする．

以上より，式（3.1）によりエッジの重みを付けた隣接行列 A を求めるとき，その各成分 $a(i,j)$ のエッジの重みは，メソッド数を n とし，ノード 0 を追加の処理の 2 つ目で追加したノードとすると，次の式（3.2）によって表される．

$$a(i,j) = \begin{cases} TF\text{-}IDF_{N\text{-}gram}(m(j)) & (i=0, j\neq 0) \\ \min\{TF\text{-}IDF_{N\text{-}gram}(m(k)), k=1,2,\cdots,n\} & (i\neq 0, j=0) \\ TF\text{-}IDF_{N\text{-}gram}(m(i)) & (i,j\neq 0, i=j) \\ a(j,i) = TF\text{-}IDF_{N\text{-}gram}(m(j)) & (i,j\neq 0, j\in inv(i)) \end{cases} \tag{3.2}$$

ここで，$a(i,j)$ はノード i からノード j へのエッジの重みを表し，エッジがない場合は 0 になる．$m(j)$ はノード j のメソッド名を表し，そして，$inv(i)$ はノード i の

メソッドが呼び出すメソッドのノードの集合を表す．また，i と j の取りうる値は $\{0, 1, 2, \cdots, n\}$ である．式（3.2）において，$i = 0, j \neq 0$ および $i \neq 0, j = 0$ のときは 2 つ目の追加の処理によって追加されたエッジの重みの計算であり，$i, j \neq 0, i = j$ のときは 1 つ目の追加の処理によって追加されたエッジの重みの計算である．

次に，式（3.2）より，エッジの重みを付けた隣接行列 A から状態遷移確率 P を求めるとき，その各成分 $p(i, j)$ は次の式（3.3）によって表される．

$$p(i, j) = \frac{a(i, j)}{\sum_{k=0}^{n} a(i, k)} \tag{3.3}$$

ここで，$p(i, j)$ はノード i からノード j に遷移する確率を表す．

次に，この状態遷移確率 P を用いて，マルコフ連鎖の定常分布を求める．この定常分布 π は，次の式（3.4）で求めることができる．

$$\pi P = \pi \tag{3.4}$$

ここで，π はベクトルであり，$\pi(y) \geq 0$ かつ，$\sum_y \pi(y) = 1$ を満たす．

実際に，この定常分布 π を計算機上で求めるときは，状態遷移確率 P の極限分布が定常分布になる性質を利用して求める．その極限分布は，任意の確率ベクトルを t とすると，次の式（3.5）で表される．

$$\lim_{n \to \infty} sP^n = \pi \tag{3.5}$$

そして，簡単のために次の式（3.6）で定常分布の近似値 π' を求める．

$$\pi' = sP^n - sP^{n-1} \leq \epsilon \qquad (\exists n \in \{1, 2, \cdots\}) \tag{3.6}$$

ここで，ϵ を許容誤差とし，この許容誤差 ϵ を満たす n を求めることで，定常分布の近似値 π' を求め，その分布で高い値を持つメソッドを要約メソッドとして抽出する．

3.3 要約メソッドの提示方法

要約手法の提示方法は，エディタなどで使われるコードの折り畳み機能を利用して行う．よって，式（3.6）で得られた定常分布の近似値 π' の値を利用し，各メソッドを折り畳むか，折り畳まないかの 2 値分類問題として考えることができる．しかし，メソッドにはその値だけでなく，メソッドの行数なども考慮する必要があるた

めに，単純に閾値を設定して分類する方法では要約後のソースコードの行数にばらつきが出てしまう．

したがって，0-1 ナップサック問題を参考にして，折り畳むメソッドを 0，折り畳まないメソッドを 1 として，提示する要約メソッドを選択する．メソッドの集合を I とし，各メソッド $i \in I$ の行数を l_i，価値を v_i，展開できるメソッドの最大行数を L_{max} とすると以下の式（3.7）（3.8）（3.9）を満たす x_i の集合から折り畳むメソッドを決定する．

$$\max \qquad \sum_{i \in I} v_i x_i \tag{3.7}$$

$$s.t. \qquad \sum_{i \in I} l_i x_i \leq L_{max} \tag{3.8}$$

$$x_i \in \{0, 1\} \qquad (\forall i \in I) \tag{3.9}$$

ここで，メソッドの価値 v_i は，定常分布の近似値 π' に正規化などの調整をすることで求める．

また，展開できるメソッドの最大行数 L_{max} は，全メソッドの行数の合計からの圧縮率 r を指定して，次の式（3.10）により求める．

$$L_{max} = r \cdot \sum_{i \in I} l_i \tag{3.10}$$

これにより，指定した圧縮率 r でソースコードを要約することができる．

4. 実験方法

提案手法を評価するために，研究設問を設け，被験者によるメソッドの評価の実験とコードの折り畳み機能によるソースコードの要約を提示し読んでもらう実験を行なった．実験で被験者に読んでもらうプログラミング言語は Java とした．

4.1 研究設問

提案手法を評価するために 3 つの研究設問を設けた．

RQ1　NgramIDF で特徴のあるメソッド名の重み付けができるのか

NgramIDF は提案手法において，状態遷移確率を生成するための隣接行列のエッジの重み付けに使われている．NgramIDF は式（2.4）により，語の繋がりを考慮して語の重み付けができるため，例えばソースコードにおいて，get や set，put といった文字列が含まれるメソッド名の NgramIDF は低くなる傾向がある．つまり，複数のソースファイルで出現する文字列の組み合わせからなるメソッド名の NgramIDF は低くなる傾向がある．また，それらの文字列からなるメソッド名を持つメソッドは他のファイルに存在するメソッドと似た処理をすることもあり，メソッドの処理をある程度予測できる．例えば，getName の時は変数 name を返り値として返すことが予測できる．したがって，メソッド名の NgramIDF の値が低ければ低いほど，メソッドの処理が想像しやすい可能性がある．よって，状態遷移確率を生成するために NgramIDF を隣接行列の重みに使用している理由は，メソッドのボディ部分を読んでいる途中に出現するメソッド名の値が低いほど，そのメソッドの処理が想像できるためにそのメソッドを読む必要がなくなり，逆に，メソッド名の値が高いほど，そのメソッドの処理が想像できないためにそのメソッドを読む必要が出てくると考えたためである．

したがって，NgramIDF がソースコードにおいても語の繋がりを考慮して語の重み付けができるのかどうか，また，NgramIDF の値が低いほど処理が想像しやすく，高いほど処理が想像しづらいのかを調べたいため，この設問を設けた．

RQ2　提案手法でプログラマが重要だと思うメソッドを高く評価でき，要約メソッドを提示できるか

提案手法では，メソッド名とメソッド呼び出しといったプログラマがソースコードを要約する際に注目する部分を利用して，メソッドの要約手法を考えたが，その手法でプログラマが重要だと考えるメソッドを正しく評価できているかどうかを調べたいため，この設問を設けた．

RQ3　要約したソースコードを提示することでプログラム理解に貢献できるのか

提案手法で，実際にエディタのコード折り畳み機能を利用してソースコードの要約を提示するが，この方法で提示したソースコードをプログラマが読んだ時にソースコードの可読性が上がるのか，また，提案手法で選択した折り畳まないメソッドが実際にソースコードにおいて重要なメソッドかどうかを調べたいため，この設問を設けた．

4.2　対象プロジェクトおよびソースコード

本研究で行なった実験で使用したソースコードは，Github で公開されている 5 つの Java のオープンソースプロジェクト [27][28][29][30][31] から，それぞれ約 300 行以下のソースコードを 1 つずつ選択した．また，ソースコードの要約を提示するソースコードはプロジェクト ant[27] から 1 つ選択した．表 4.1 に選択したオープンソースプロジェクトとその説明や LoC などを載せた．さらに，表 4.2 と表 4.3 には実験で使用したソースコードを載せた．この実験で使用したプロジェクトは Fowkes らによる関連研究 [7] で利用されているものを参考に選択した．また，使用したソースコードは実験時間や被験者の負担を考えて，約 300 行以下の比較的行数の短いものにした．

4.3　実装したプログラムについて

提案手法を実装するにあたり，NgramIDF については白川らによって Github 上で公開されている実装 [32] を参考にして，ソースコードに対して NgramIDF の計算を行えるように改良を加え，新たに Python を用いて自ら実装を行なった．

また，提案手法では Java のソースコードからメソッド名とメソッド呼び出し，メソッドの範囲などを取得する必要があるため，これらを Java の AST (Abstract Syntax Tree) を解析することで取得した．AST を解析する際には Eclipse Foundation(注 1) に

(注 1)：https://www.eclipse.org/

表 4.1　Github 上にある Java のオープンソースプロジェクト

プロジェクト	説明	LoC	Java ファイル数	取得日時
ant	ビルドツールソフトウェア	282,269	1,310	2018/12/21 17:12
elasticsearch	分散型 RESTful 検索/分析エンジン	1,817,221	10,415	2018/12/21 14:52
libgdx	ゲーム開発フレームワーク	406,327	2,329	2018/12/21 14:57
netty	ネットワークアプリケーションフレームワーク	406,098	2,422	2018/12/21 14:58
spring-framework	アプリケーションフレームワーク	1,107,278	6,922	2018/12/21 15:58

表 4.2　メソッドの評価の実験で使用したソースコード

プロジェクト	ソースコード	主な説明	LoC	メソッド数
ant	Mkdir.java	与えられたディレクトリを作成する	115	4
elasticsearch	FileWatcher.java	ファイルリソースの監視	319	17
libgdx	OrderedSet.java	挿入順序を使用して key 値も Array に格納する	157	18
netty	SctpMessage.java	SCTP データチャンクの表現	207	19
spring-framework	CallableStatementCreatorFactory.java	あるオブジェクトを作成するためのヘルパークラス	237	14

表 4.3　ソースコードの要約を提示するソースコード

プロジェクト	ソースコード	主な説明	LoC	メソッド数
ant	BuildNumber.java	ビルドナンバーを更新する	166	5

よって提供されている JDT (Java Development Tools) を利用した．JDT の中で AST に関する処理は「org.eclipse.jdt.core.dom」というパッケージにまとめられている．

ソースコードの要約を提示するための実装としては，ソースコードなどを編集する際に利用されるエディタである Sublime Text(注 2) のユーザが自由に定義できるプラグインの API (Application Programming Interface)[33] を利用し，メソッドの折り畳み機能を実装した．

4.4 被験者による評価実験

提案手法を評価するために，Java のプログラミング経験のある情報工学専攻の学生 9 名に協力してもらい実験を行なった．実験時間は約 1 時間ほどで，表 4.2 にある 5 つのソースコードを読んでもらい，ソースコード内にある各メソッドに対して，以下の質問に 5 段階評価で答えてもらった．

- シグネチャから処理が想像できるかどうか
- 機能の概要から重要なメソッドと判断できるかどうか

まず「シグネチャから処理が想像できるかどうか」については，被験者にメソッドのシグネチャを提示し，そのシグネチャから処理が想像できるなら高い評価にしてもらった．次に「機能の概要から重要なメソッドと判断できるかどうか」については，重要なメソッドかどうか判断するための指針として，それぞれのソースコードに対して機能の概要を提示した．機能の概要はソースコードに含まれるコメントを参考にして作られ，その概要に沿うメソッドであるほど高い値を選択してもらった．

そして，最後に表 4.3 にある 1 つのソースコードの要約を提示したソースコードを読んでもらい，以下の質問に答えてもらった．

- メソッドが折り畳まれたことによりソースコードを読みやすくなったように感じたか
- ソースコードの理解に必要そうなメソッドが展開されていたか

この質問に対しては 5 つの評価尺度を与えて，それを選択した理由も併せて答えてもらった．

(注 2)：https://www.sublimetext.com/

5. 結果および考察

第 4.1 節の研究設問に沿って結果および考察を述べる．

5.1 RQ1:NgramIDF で特徴のあるメソッド名の重み付けができるのか

まず，NgramIDF は自然言語で利用される語の重み付け手法であるが，これがソースコードに対しても単語の繋がりを考慮に入れて重み付けができるのかどうかを示していく．表 4.1 の ant におけるメソッド名の集合に対して，NgramIDF を求め，その値の分布を調べた．比較のために IDF の値も求め，それを箱ひげ図にしたものが図 5.1 である．図 5.1 より，IDF は比較的に高い値を取ることが多く，NgramIDF に比べてうまく重み付けができていないことが分かる．また，NgramIDF は IDF に比べて値が分散していて，うまく重み付けができていることが分かる．図 5.1 に，NgramIDF と IDF の平均，中央値，分散を求めた．これらの値からも IDF に比べて，NgramIDF の方がうまくメソッド名の重み付けができるということが分かる．よって，ソースコードに対しても，NgramIDF で語の重み付けをすることは可能であることが分かる．

次に，第 4.4 節のプログラマによる評価実験により，NgramIDF により高い重みが付けられたメソッド名は処理が想像しづらいのか，また，低い重みが付けられたメソッド名は処理が想像しやすいのかを示していく．9 名の被験者によって「シグネチャから処理が想像できるかどうか」の 5 段階評価の平均値を取り，その平均値と NgramIDF の値の相関関係を求めた．その結果が図 5.2 になり，相関係数は -0.26 を取り，弱い負の相関が見られた．図 5.2 を見ると，左下の NgramIDF の値が低く，処理の想像がしづらい値になるプロットが少ないことより，NgramIDF の値が低いメソッド名は処理が想像しやすい傾向が見れる．対して，右上の NgramIDF の値が高く，処理の想像がしやすい値になるプロットが少なくないことより，NgramIDF の値が高いメソッド名は処理の想像がしづらいという傾向はあまりないように見える．つまり，NgramIDF の値が低くなるメソッド名は処理の想像がしやすくなる傾向は見られるが，NgramIDF の値が高くなるメソッド名は処理の想像がしづらい傾向はあまり見られなかったことがわかった．

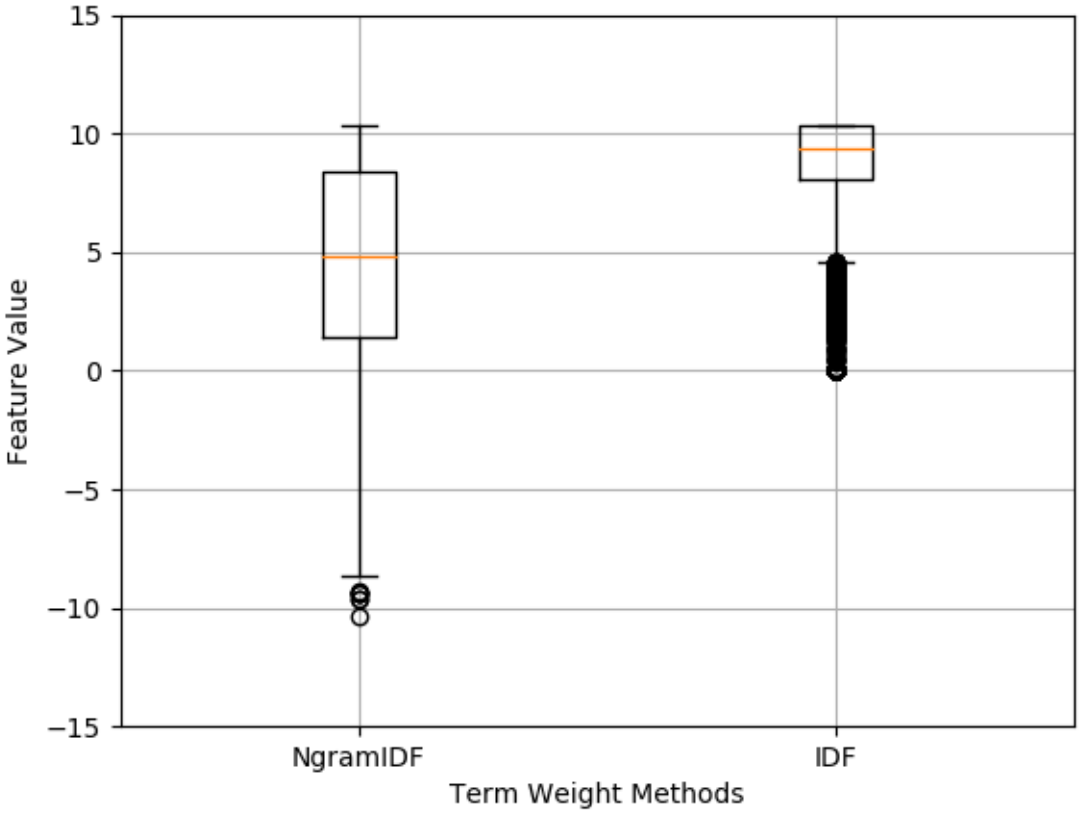

図 5.1　ant におけるメソッド名の NgramIDF と IDF の値の箱ひげ図

表 5.1　NgramIDF と IDF の平均，中央値，分散

語の重み付け手法	平均	中央値	分散
NgramIDF	4.60	4.80	17.60
IDF	8.80	9.36	4.80

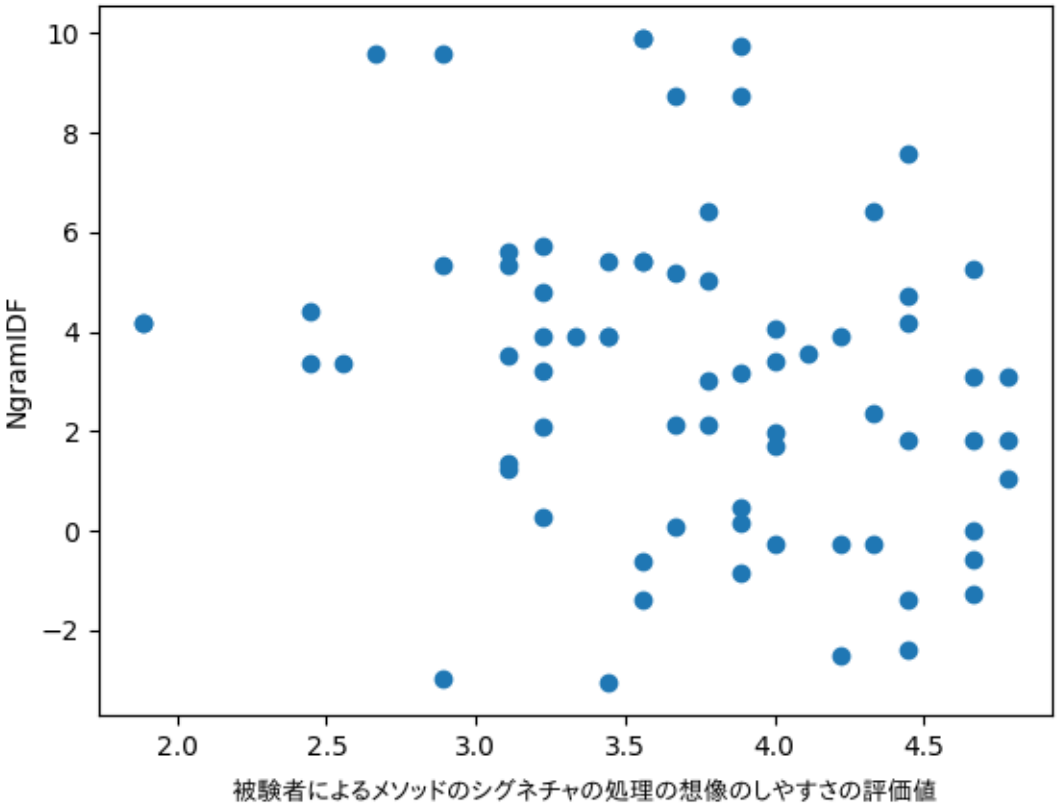

図 5.2　NgramIDF と被験者によるメソッドのシグネチャから処理の想像のしやすさの評価値の相関関係（相関係数 = -0.26）

具体例として，表 5.2 にその傾向が見られるものを載せた．また，表 5.3 にはその傾向が見られないものを載せた．

表 5.2 と表 5.3 を見ると，予想通り set や get といった単語が付くメソッド名は NgramIDF の値が低いことがわかる．しかし，NgramIDF の値が低い場合に必ず被験者の評価が高くなるとは限らず，setResultSetType のように被験者の評価が低くなる名前もあった．setResultSetType の処理は変数の ResultSetType に引数の値を代入することが予想できるが，名前を見ると set という単語が 2 回出現し，set がどこに掛かってるのか分かりづらいために被験者の評価が低くなったと考える．また，表 5.3 では，checkAndNotify と mkdirs が相関が見られなかった．これらのメソッド名の処理は，被験者に提示したソースコードの機能の概要に関係がある．実験の手順を考慮すると，この機能の概要を見た後に「シグネチャから処理が想像できるかどうか」という質問に答えることによって処理が想像しやすくなり，被験者の評価が高くなったことが考えられる．実際に被験者から「概要を見たからこのメソッドのシグネチャから処理が想像しやすかった」という意見もあった．

また，提案手法ではメソッドのシグネチャのメソッド名のみの特徴付けを考えたが，実際のプログラマによる評価実験では図 5.3 のようにメソッドのシグネチャから処理を想像できるかどうかを答えてもらった．そのため，同じメソッド名であっても図 5.4 のようにパラメータが違う場合などがあり，表 5.4 のように被験者の評価が変わっていた．表 5.4 を見ると，同じメソッド名でもパラメータを持つメソッド名の方が処理を想像しづらいことがわかる．したがって，シグネチャの評価をする際にはメソッド名だけではなく，パラメータなどの要素も考慮に入れて特徴付けをする必要があることがわかった．

以上の結果より，NgramIDF はソースコードのメソッド名に対してうまく特徴付けができ，またその得られた特徴量から特に NgramIDF の低いメソッド名に対して処理の想像のしやすさに少しの相関があることがわかり，隣接行列のエッジの重み付けにおいても少しの適正があることがわかった．しかし，同じメソッド名だとしてもパラメータの違いによってメソッドの想像のしやすさに影響があるので，今後その点に考慮する必要があると考える．

表 5.2　相関があったメソッド名の例

メソッド名	NgramIDF	被験者の評価の平均偏差
getDir	-0.57	0.95
setDir	-0.26	0.50
retainedDuplicate	4.41	-1.27
CallableStatementCreatorImpl	9.59	-1.05

表 5.3　相関がなかったメソッド名の例

メソッド名	NgramIDF	被験者の評価の平均偏差
setResultSetType	-3.06	-0.27
clearState	-2.97	-0.83
checkAndNotify	4.70	0.73
mkdirs	5.27	0.95

```
public void execute() throws BuildException { ... }
```

```
public void setDir(File dir) { ... }
```

```
public File getDir() { ... }
```

```
private boolean mkdirs(File f) { ... }
```

図 5.3　被験者に提示したシグネチャの例

```
public void clear(int maximumCapacity) { ... }

public void clear() { ... }
```

```
public String toString() { ... }

public String toString(String separator) { ... }
```

図 5.4　同じメソッド名でパラメータに違いがあるシグネチャ

表 5.4　同じメソッド名でパラメータに違いがあるメソッド名の例

メソッド名 (パラメータ型)	NgramIDF	被験者の評価の平均偏差
clear(int)	2.14	-0.05
clear()	2.14	0.06
toString()	-1.38	0.73
toString(String)	-1.38	-0.16

5.2 RQ2:提案手法でプログラマが重要だと思うメソッドを高く評価でき，要約メソッドを提示できるか

提案手法を評価するために，まず第 4.4 節で説明した「機能の概要から重要なメソッドと判断できるかどうか」の 9 名の被験者の評価の平均値を取ることで，被験者による各メソッドの評価を作成した．この実験ではこの評価を正解データとして扱う（以降，被験者評価と呼ぶ）．また，提案手法と被験者評価だけでは適切な評価が難しいため，既存手法である VSM と LexRank を用いて，それぞれの結果を比較して提案手法を評価することにする．

まず，被験者評価と各手法によるメソッドの評価値の相関関係を求める．被験者評価と各手法のそれぞれのソースコードにおける相関係数とそれらの平均値を求めて表にしたのが，表 5.5 である．表 5.5 を見ると，提案手法が VSM や LexRank と比べて，FileWatcher.java を除く実験対象について安定した被験者評価との相関を持っていることがわかる．各ソースコード 5 つの相関係数の平均値を見てみると，提案手法が他の手法に比べて被験者評価と相関があることがわかる．しかし，FileWatcher.java に対してはいずれの手法も被験者評価と相関を持たなかった．この原因の一つとして，メソッド名に使用される単語があまり特徴的な単語ではなかったためであると考える．例えば，このソースコードはファイルリソースを監視するためのものであるため，重要であると判断できるようなメソッド名の一部に「File」といった特徴量の低い単語が含まれていた．これによって，TF-IDF などの語の重み付けがうまくできなかった結果，被験者評価と相関を持たなかったと考える．

次に，被験者評価と各手法を用いた要約メソッドの選択による評価を行う．要約メソッドの選択方法は第 3.3 節の方法により求め，各メソッドを折り畳むか折り畳まないかの 2 値分類問題として考え，正解率と適合率，再現率，F 値をそれぞれ求めた．表 5.6 に，被験者評価と各手法による圧縮率ごとの折り畳みメソッドの正解率を載せた．圧縮率は 70[%] や 80[%] ではソースコードを要約したとは考えにくいことから，10[%] から 60[%] までの 6 段階にした．表 5.6 の太字で強調された数値を見ると，提案手法の正解率が既存手法を下回ることもあるが，全体的に見ると既存手法を上回ることが多いことがわかる．また，表 5.7 は，全体の平均値を取った結果である．表 5.7 からも提案手法が既存手法よりも平均的に優れていることがわかる．

表 5.5　被験者評価と各手法によるメソッド評価値の相関関係

ソースコード	手法	相関係数
Mkdir.java	提案手法	0.87
	VSM	0.51
	LexRank	0.62
FileWatcher.java	提案手法	-0.05
	VSM	-0.29
	LexRank	-0.01
OrderedSet.java	提案手法	0.42
	VSM	0.55
	LexRank	0.04
SctpMessage.java	提案手法	0.54
	VSM	0.57
	LexRank	0.53
CallableStatementCreatorFactory.java	提案手法	0.34
	VSM	0.17
	LexRank	0.03
5 つの平均値	提案手法	0.42
	VSM	0.30
	LexRank	0.24

表 5.6　圧縮率ごとにおける各手法での折り畳みメソッドと被験者評価との正解率

ソースコード	手法	10[%]	20[%]	30[%]	40[%]	50[%]	60[%]
	提案手法	1.00	1.00	1.00	1.00	1.00	1.00
Mkdir.java	VSM	1.00	1.00	0.50	1.00	1.00	1.00
	LexRank	1.00	1.00	1.00	1.00	1.00	1.00
	提案手法	0.71	0.53	0.47	**0.76**	0.53	0.47
FileWatcher.java	VSM	0.65	0.53	0.47	0.41	0.53	0.41
	LexRank	**0.82**	**0.71**	**0.65**	0.59	0.53	**0.53**
	提案手法	**0.89**	**0.72**	0.61	0.61	0.61	0.61
OrderedSet.java	VSM	0.72	0.56	**0.72**	**0.67**	0.50	**0.67**
	LexRank	0.78	0.61	0.44	0.50	**0.67**	0.50
	提案手法	0.89	**0.89**	0.72	0.50	**0.56**	**0.72**
SctpMessage.java	VSM	0.89	0.78	0.50	0.44	0.44	0.56
	exrank	0.89	0.78	0.72	**0.67**	0.44	0.44
	提案手法	**0.71**	**0.79**	0.71	1.00	1.00	1.00
CallableStatementCreatorFactory.java	VSM	0.64	0.64	0.71	1.00	1.00	1.00
	LexRank	0.57	0.50	0.71	1.00	1.00	1.00

表 5.7　被験者評価と各手法の正解率，適合率，再現率，F 値の各平均値

手法	accuracy	precision	recall	f-measure
提案手法	**0.77**	**0.67**	**0.67**	**0.67**
VSM	0.70	0.59	0.58	0.58
LexRank	0.74	0.63	0.62	0.62

以上の結果より，提案手法はプログラマが重要だと思うメソッドを高く評価でき，また要約メソッドの提示もできることがわかった．しかし，FileWatcher.java のようにメソッド名の単語の組み合わせによっては，性能が左右される場合があるので注意が必要である．

5.3 RQ3:要約したソースコードを提示することでプログラム理解に貢献できるのか

第 4.4 節にあるように，表 4.3 にある 1 つのソースコードを提案手法により要約を提示したソースコードを被験者 9 名に読んでもらった．被験者に読んでもらった後に以下の 2 つの質問に答えてもらった．

- メソッドが折り畳まれたことによりソースコードを読みやすくなったように感じたか
- ソースコードの理解に必要そうなメソッドが展開されていたか

その質問に答えてもらった結果，まず「メソッドが折り畳まれたことによりソースコードを読みやすくなったように感じたか」に対して,「とても読みやすく感じた」が 1 名,「少し読みやすく感じた」が 6 名,「どちらも感じなかった」が 2 名,「少し読みづらく感じた」が 0 名,「とても読みづらく感じた」が 0 名であった．その結果を円グラフにしたものが図 5.5 である．

結果より,「とても読みやすく感じた」が 1 名,「少し読みやすく感じた」が 6 名とあるように，ソースコードが折り畳まれたことによりソースコードが読みやすくなったように感じることがあることがわかった．このようになった理由としては被験者のコメントより,

- getter や setter のような名前を見ただけで処理が分かるメソッドが折り畳まれていて中身を読む必要がなくなったため（2 名）
- メソッド名がメソッドの中身を的確に表していたため（2 名）
- 関数が折り畳まれていてもメソッドの上部に処理の内容がコメントで書かれていたため（1 名）
- 全体のコードの量が少なく感じたため（1 名）

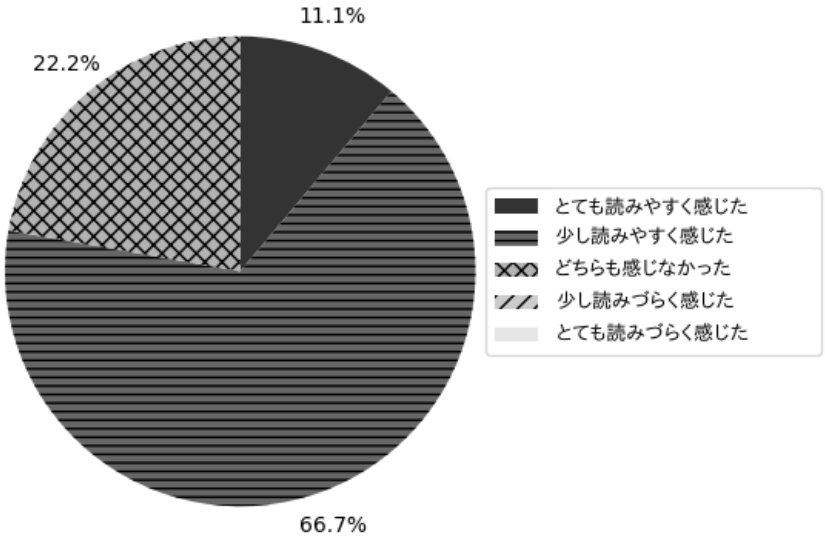

図 5.5　メソッドが折り畳まれたことによりソースコードを読みやすくなったように感じたか

などが挙げられる.

また,「どちらも感じなかった」が 2 名いるが，その主な理由として被験者のコメントより,

- 折り畳まれたことにより全体の把握はしやすくなったが，折り畳まれたメソッドの処理の内容が正確に把握できなくなったため（1 名）

という理由が挙げられた.

次に,「ソースコードの理解に必要そうなメソッドが展開されていたか」に対して,「ほとんど展開されていた」が 3 名,「少し展開されていた」が 6 名,「どちらでもなかった」が 0 名,「あまり展開されていなかった」が 0 名,「ほとんど展開されていなかった」が 0 名であった．その結果を円グラフにしたものが図 5.6 である.

結果より,「ほとんど展開されていた」が 3 名,「少し展開されていた」が 6 名となり，提案手法により展開されたメソッドがソースコードの理解に必要そうなメソッドであると判断できそうである．このようになった理由としては被験者のコメントより,

- 重要なメソッドは展開されているメソッドに集約されており，他のメソッドは比較的重要度が低く，また，メソッドのシグネチャから内容が推測可能であったため.（1 名）
- 肝となりそうな処理は展開されていたが，そこから呼び出される細かい処理の部分が折りたたまれており，完全に理解することはできないと思ったため（3 名）
- 大まかな部分を理解できる程度ではあったため（3 名）

などが挙げられた.

以上の結果より，メソッドが折り畳まれることによりソースコードが読みやすくなり，また提案手法により展開されたメソッドはソースコードの理解に必要そうなメソッドであるために，そのメソッドを見ることでそのソースコードの概念的な処理を理解することができることがわかった．また，折り畳まれたことによりソースコードやメソッドの正確な処理を把握できなくなるというコメントがあったが，その点については，ユーザが折り畳まれたコードを展開できるようにすることで対応

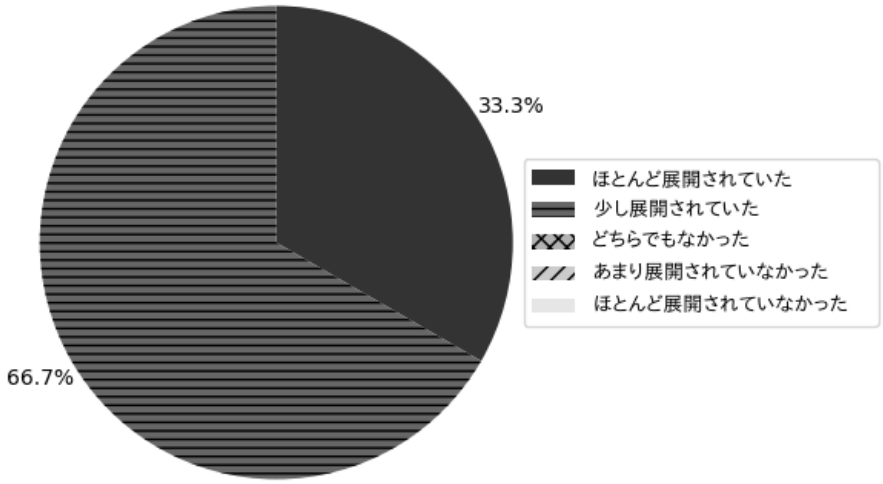

図 5.6　ソースコードの理解に必要そうなメソッドが展開されていたか

できると考えている．さらに，この実験で用いた表 4.3 のソースコードはメソッドの先頭にメソッドの処理を示すコメントが記述されていたので，そのコメントの影響で読みやすく感じた可能性もあると考える．つまり，折り畳まれたメソッドの先頭にあるコメントが無ければ，読みづらく感じる可能性もあると考えた．

6. 妥当性への脅威

6.1 外的妥当性

外的妥当性では実験結果の一般性についての妥当性への脅威について考える．

対象プロジェクトおよびソースコードについて

プログラミング言語はJavaに絞り，また実験に使用したプロジェクトはGithub上で公開されている5つのプロジェクトである．よって，Java以外のプログラミング言語や，Github上などで公開されていないプロジェクトでは違う結果になる可能性がある．また，実験で利用したソースコードは被験者の負担を考え，約100行から約300行のソースコードを選択した．実際にプロジェクトに存在するソースコードの行数は約100行から約300行のソースコードだけではなく，中には1,000行を超えるソースコードもあるために，そのようなソースコードに対して実験を行うと違う結果になることも考えられる．したがって，実験で使用するプロジェクトの違いやソースコードの違いによって実験結果に影響があることが考えられるので，この点は外的妥当性への脅威になりうる．また，実験に使用したソースコードの数は6つで，その内5つが被験者によるメソッドの評価に利用し，残りの1つは被験者によるソースコードの折り畳み実験のアンケートに使用した．よって，実験で使用したソースコードの少なさも外的妥当性への脅威となりうるため，留意する必要がある．

被験者によるメソッドの評価について

また今回の実験では被験者によるメソッドの評価を正解データとして扱った．被験者はJavaのプログラミング経験のある情報工学専攻の学生9名を集めた．しかし，被験者のほとんどがJavaのプログラミング経験が数ヶ月であるため，中にはJava特有のクラスの書き方やメソッド名の書き方に慣れてない人もいた．また，実験で使用したソースコードによっては出現する単語が専門的な単語であるためにソースコードを完全に理解できないといった問題も挙げられた．よって，Javaのプログラミング経験がもっと長い被験者やプロジェクトに関係のある被験者を集めると，実験結果より良い正解データになることが考えられるので，この点についても外的妥当性への脅威になりうる．

6.2 内的妥当性

内的妥当性では実験方法の妥当性への脅威について考える．

被験者によるメソッドの評価方法について

被験者によるメソッドの評価方法については，被験者によって重要なメソッドの評価の仕方が変わると問題があるので，ソースコードのコメントを参考に実験者がソースコードの機能の概要を提示し，被験者にはそれに沿うメソッドの評価を高くしてもらった．しかし，実験者は対象プロジェクトの開発者ではないために，対象ソースコードを完全に理解できているわけではなく，またソースコードのコメントによって機能の概要を作成する必要があるために，実験者はその機能の概要が完全に正しいと判断することはできない．したがって，被験者に提示した正確さの不明な機能の概要によるメソッドの評価は内的妥当性への脅威となりうる．また，被験者に「シグネチャから処理が想像できるかどうか」によるメソッドの評価は，ソースコードの機能の概要を読んでもらった後に行われるため，被験者によっては概要を読む前後で評価が変わってしまう可能性もある．実際に被験者からもそのようなコメントを頂いたので，この点も内的妥当性への脅威となりうると考える．

実装したプログラムの正確さについて

本実験で利用したプログラムは重要な部分についてはテスト駆動で開発を行うことなどで不具合を取り除いたが，まだ未発見の不具合が残っている可能性はある．そのため，プログラムに内在するバグが内的妥当性への脅威となりうる．

7. 結言

本研究では，プログラマがソースコードを要約する際に重要だと考えるメソッドのシグネチャとメソッドの呼び出しに着目し，既存の自然言語処理の技術である NgramIDF と LexRank を応用した新しいソースコードにおけるメソッドの抽出的要約手法を提案した．

Github 上にある公開されている Java の 5 つのプロジェクトに対してそれぞれソースコードを一つずつ選び，情報工学専攻の学生 9 名にメソッドの評価をしてもらった．その結果，NgramIDF によるメソッド名の特徴付けにより少しであるが特徴のあるメソッドを評価できることがわかった．また既存手法である VSM や LexRank と比較実験を行なった結果，提案手法によるメソッドの評価と被験者によるメソッドの評価に，既存手法よりも相関があることがわかった．また，提案手法による折り畳みメソッドの選択によるソースコード要約の精度についても，既存手法よりも優れていることがわかった．

さらに情報工学専攻の学生 9 名に提案手法によるソースコードの要約を提示したところ，要約によってソースコードが読みやすく感じる学生が多く，また，提案手法によるソースコードの要約によりソースコードを理解する上で重要なメソッドから読むことができ，ソースコードの概念的な理解に繋がったと考えられる．

以上の点から，提案手法によるソースコード要約によって，プログラマのプログラム理解に貢献できる可能性を示したと考える．今後の課題としては，RQ1 への考察結果からメソッドのシグネチャの内のメソッド名だけでなく，例えばパラメータ名などを利用した特徴付け手法の考慮がある．また本研究ではメソッドのコールグラフはソースコード単位で作成していたので，プロジェクト単位で作成するとまた結果が変わってくると考えている．

謝辞

本研究を行うにあたり，研究課題の設定や研究に対する姿勢，本報告書の作成に至るまで，全ての面で丁寧なご指導を頂きました，本学情報工学・人間科学系 水野修教授に厚く御礼申し上げます．本報告書執筆にあたり実験協力や貴重な助言を多

数頂きました，本学情報工学専攻の小西健太君，前田竜輝君，木本大介君，およびソフトウェア工学研究室の皆さん，そして学生生活を通じて著者の支えとなった家族や友人に深く感謝致します.

参考文献

[1] A.J. Ko, B.A. Myers, M.J. Coblenz, and H.H. Aung, “An exploratory study of how developers seek, relate, and collect relevant information during software maintenance tasks,” IEEE Transactions on Software Engineering, vol.32, pp.971–987, Dec. 2006.

[2] A.J. Ko, B.A. Myers, M.J. Coblenz, and H.H. Aung, “Maintaining mental models: a study of developer work habits,” In Proceedings of the 28th International Conference on Software Engineering, pp.492–501, 2006.

[3] J. Starke, C. Luce, and J. Sillito, “Searching and skimming: An exploratory study,” In Proceedings of the 2009 International Conference on Software Maintenance, pp.157–166, 2009.

[4] 大場　勝，権藤克彦，“プログラム理解を支援するコンセプトキーワードの自動抽出法 cktf/idf 法の提案,” 情報処理学会論文誌，vol.48，no.8，pp.2596–2607，Aug. 2007.

[5] S. Haiduc, J. Aponte, L. Moreno, and A. Marcus, “On the use of automated text summarization techniques for summarizing source code,” In Proceedings of the 2010 17th Working Conference on Reverse Engineering, pp.35–44, 2010.

[6] A.T.T. Ying and M.P. Robillard, “Code fragment summarization,” In Proceedings of the 9th Joint Meeting on Foundation of Software Engneering, pp.655–658, 2013.

[7] J. Fowkes, P. Chanthirasegaran, R. Ranca, M. Allamanis, M. Lapata, and C. Sutton, “Autofolding for source code summarization,” IEEE Transactions on Software Engineering, no.12, pp.1095–1109, Dec. 2017.

[8] P. Rodeghero，C. McMillan，P.W. McBurney，N. Bosch，S.D ’Mello，“Improving automated source code summarization via an eye-tracking study of programmers,” In Proceedings of the 36th International Conference on Software Engineering, pp.390–401, May 2014.

[9] 白川真澄，原隆浩，西尾章治郎，“コルモゴロフ複雑性に基づく idf の単語 n-gram への適用,” DEIM Forum，vol.A，pp.3–5，2015.

[10] E. G and R.D. R, "Lexrank: graph-based lexical centrality as salience in text summarization," Journal of Artificial Intelligence Research, pp.457–479, 2004.

[11] H.C Wu, R.W.P. Luk, K.F Wong, and K.L. Kwok, "Interpreting tf-idf term weights as making relevance decisions," ACM Transactions on Information Systems, vol.26, no.3, pp.13:1–13:37, June 2008.

[12] B.C. Fung, K. Wang, and M. Ester, "Hierarchical document clustering using frequent itemsets," Proceedings of SIAM International Conference on Data Mining(SDM), pp.59–70, May 2003.

[13] K.S. Hasan and V. Ng, "Conundrums in unsupervised keyphrase extraction: Making sense of th state-of-the-art," Coling, pp.365–373, Aug. 2010.

[14] J. Sivic and A. Zisserman, "Video google: A text retrieval approach to object matching in videos," ICCV, pp.1470–1477, Oct. 2003.

[15] A. Aizawa, "An information-theoretic perspective of tf-idf measures," Information Processing and Management, vol.39, pp.45–65, 2003.

[16] S. Robertson, "Understanding inverse document frequency: On theoretical arguments for idf," Journal of Documentation, vol.60, no.5, pp.503–520, 2004.

[17] D. Hiemstra, "A probabilistic justification for using tfxidf term weighting in information retrieval," International Journal on Dgital Libraries, vol.3, no.2, pp.131–139, Sept. 2000.

[18] K.S. Jones, "A statistical interpretation of term specificity and its application in retrieval.," Journal of Documentation, vol.28, no.1, pp.11–21, 1972.

[19] D. Metzler, "Generalized inverse document frequency," CIKM, pp.26–30, Oct. 2008.

[20] C.D. Manning, P. Raghavan, and H. Schutze, Introduction to Information Retrieval, U.K.: Cambridge Univ. Press, Cambridge, 2008.

[21] X. Liu, J.J. Webster, and C. Kit, "An extractive text summarizer based on significant words," In Computer Processing of Oriental Languages. Language Technology for the Knowledge-based Economy, pp.168–178, 2009.

[22] L. Page, S. Brin, R. Motwani, and T. Winograd, “The pagerank citation ranking: Bringing order to the web,” 1998.

[23] D. Lawrie, C. Morrell, H. Field, and D. Binkley, “What ’s in a name? a study of identifiers,” 14th IEEE International Conference on Program Comprehension, pp.3–12, June 2006.

[24] J. Hofmeister, J. Siegmund, and D.V. Holt, “Shorter identifier names take longer to comprehend,” 24th IEEE International Conference Software Analysis Evolution and Reengineering, pp.217–227, Feb. 2017.

[25] S.L. Abebe, S. Haiduc, P. Tonella, and A. Marcus, “The effect of lexicon bad smells on concept location in source code,” In Proceedings of the 2011 International Working Conference Source Code Analysis Manipulation, pp.125–134, 2011.

[26] T. Kato, S. Hayashi, and M. Saeki, “Cutting a method call graph for supporting feature location,” 2012 Fourth International Workshop on Empirical Software Engineering in Practice, pp.55–57, 2012.

[27] GitHub, Inc., GitHub - apache/ant: Mirror of Apache Ant, github（オンライン）, 入手先〈https://github.com/apache/ant〉（参照 2018-12-21）.

[28] GitHub, Inc., GitHub - elastic/elasticsearch: Open Source, Distributed, RESTful Search Engine, github（オンライン）, 入手先〈https://github.com/elastic/elasticsearch〉（参照 2018-12-21）.

[29] GitHub, Inc., GitHub - libgdx/libgdx: Desktop/Android/HTML5/iOS Java game development framework, github（オンライン）, 入手先〈https://github.com/libgdx/libgdx〉（参照 2018-12-21）.

[30] GitHub, Inc., GitHub - netty/netty: Netty project - an event-driven asynchronous network application framework, github（オンライン）, 入手先〈https://github.com/libgdx/libgdx〉（参照 2018-12-21）.

[31] GitHub, Inc., GitHub - spring-projects/spring-framework: Spring Framework, github（オンライン）, 入手先〈https://github.com/spring-projects/spring-framework〉（参照 2018-12-21）.

[32] GitHub, Inc., GitHub - iwnsew/ngweight: N-Gram Weighting Scheme, github（オンライン）, 入手先〈https://github.com/iwnsew/ngweight〉（参照 2019-2-4）.

[33] Sublime HQ Pty Ltd, API Reference - Sublime Text 3 Documentation, sublimetext（オンライン）, 入手先〈http://www.sublimetext.com/docs/3/api_reference.html〉（参照 2019-2-4）.