修士論文

プログラム要素の組合せ集合を用いたソースコード検索

高田 大樹
奈良先端科学技術大学院大学
先端科学技術研究科
情報理工学プログラム

主指導教員: 松本 健一 教授
ソフトウェア工学 研究室（情報科学領域）

令和 2 年 3 月 13 日提出

本論文は奈良先端科学技術大学院大学先端科学技術研究科に
修士 (工学) 授与の要件として提出した修士論文である.

高田 大樹

審査委員：

松本 健一 教授　（主指導教員，情報科学領域）
笠原 正治 教授　（副指導教員，情報科学領域）
石尾 隆 准教授　（副指導教員，情報科学領域）
畑 秀明 助教　　（副指導教員，情報科学領域）

プログラム要素の組合せ集合を用いたソースコード検索*

高田 大樹

内容梗概

本論文ではソースコード中のプログラム要素を組合せ集合として索引化するソースコード検索手法を提案する．ソースコードの実装例を検索することは，ソフトウェア開発の生産性向上に欠かせない手段の 1 つである．既存のコード検索エンジンでは検索クエリの単語とソースコードの記述が一致している必要があるが，開発者が自然言語による質問形式で検索クエリを表現する事例も多く，記述の不一致で目的のソースコード例を取得できないといった課題がある．こうした自然言語クエリによるソースコード検索に対応するため，本研究ではまず自然言語クエリの事前分析を行い，単語間の対応関係，およびソースコード中に現れない語彙の存在，という 2 つの特徴を特定した．提案手法ではそれぞれの特徴に基づき，単語間の対応関係を辺要素として追加し，またソースコード中に現れない語彙を生成モデルで補完する，というアプローチをとる．さらに組合せ集合の効率的な表現が可能な ZDD というデータ構造を用いることで，それぞれの特徴表現を反映したソースコード索引を構築する．提案アプローチの効果に関する評価の結果，ZDD を用いた索引がそれぞれの特徴を正しく反映できていることを確認できた．また既存ツールとの比較により，74.8% のクエリに対して上位 10 件以内に正解のソースコード例を推薦でき，既存手法よりも 8 ポイント高い性能を達成した．これらの結果はソースコード検索における索引化手法の新たな可能性を示した．

*奈良先端科学技術大学院大学 先端科学技術研究科 修士論文, 令和 2 年 3 月 13 日.

キーワード

ソースコード検索, 自然言語クエリ, クエリ再構成, ZDD, コード生成モデル

Code Search Using Combination Set of Program Elements*

Daiki Takata

Abstract

This thesis proposes a source code retrieval method that indexes program elements in source code as a combination set. Searching implementation examples of source code is one of the indispensable practices for improving the productivity of software development. Existing code search engines require that the words in the search query match the description in the source code. However, there are many cases where the developers express search queries in the form of natural language questions, and there is a problem that the target source code examples cannot be obtained due to mismatched descriptions. To support code search using natural language queries, this study first performed a preliminary analysis in the natural language queries and identified two features: the correspondence between words and the presence of vocabulary that does not appear in the source code. The proposed method based on each feature takes approaches of adding the correspondence between words as an edge element and complementing the vocabulary that does not appear in the source code by a generating model. Furthermore, this method uses a data structure called ZDD, which can efficiently represent the combination sets, and constructs a code index that reflects each feature representation. As a result of evaluating the effectiveness of the proposed approaches, it was confirmed that the index using ZDD correctly reflected each

*Master's Thesis, Graduate School of Science and Technology, Nara Institute of Science and Technology, March 13, 2020.

feature. In comparison with the existing tools, the correct source code examples could be recommended within the top 10 for 74.8% of the queries, and the performance was 8 points higher than the existing method. These results indicate a new possibility of an indexing approach in code search.

Keywords:

Code search, natural language query, query reformulation, ZDD, code-generating model

目 次

図 目 次

表 目 次

1. はじめに

ソースコード検索は，数十年にわたってソフトウェア開発で実践されている開発者の活動である [36]．ソフトウェア開発ではソースコードの再利用を行うことで，ソフトウェアの品質および生産性の向上が望める [12, 26, 30]．また実際のソースコード例を参考にすることで，ソフトウェアの仕様や定義のみでは理解しづらい振る舞いの理解や，実装を比較することにより明らかとなるバグ修正，パフォーマンス向上の箇所の特定など，開発者は実装方法に関する理解を深めることができる [10, 25, 28]．こうした理由から，開発者は日々多くの時間を費やしてソースコード検索を行っている [13]．

一般にソフトウェア開発者が目的のソースコード例を見つけるために必要な検索クエリを適切に記述するには，多大な時間と労力が必要である [23]．既存のコード検索エンジン [2, 4, 5] では全文検索による手法が採用されており，実際のソフトウェアが保管されたソフトウェアリポジトリ [2, 6] のソースコード群の中から，開発者が検索クエリとして指定した単語と一致するソースコード例を抽出できる．この全文検索手法で得られる推薦結果の質は，検索クエリに含まれる文字列とソースコード例の記述に大きく依存する [22]．開発者は検索結果からフィードバックして検索クエリの再構成を繰り返し行うことで，目的のソースコード例を獲得している [34]．

既存のコード検索エンジンの問題点として，ソースコード例の記述とは一致しないような検索クエリによって目的のソースコード例を見つけることが困難である点が挙げられる．Bajracharya ら [10] によると，既存のコード検索エンジンに対して自然言語による質問文の形式で検索クエリを表現することで，目的のソースコード例を収集しようとする事例が頻繁に発生する，ということが報告されている．このような自然言語による検索クエリの表現はソースコード例との語彙の不一致を引き起こし，検索の効率性に悪影響を及ぼす [18]．

本研究ではこの課題の解決方法として，ソースコード検索での自然言語クエリへの対応を目的とする．ソースコード検索において自然言語クエリを扱った既存研究 [31, 33] では，オンライン Q & A フォーラム [7] の知識を利用して検索クエリにソフトウェア固有の単語を補完するクエリ再構成手法のみを対象にしている．

本研究では自然言語クエリに存在する特徴をソースコード索引に反映させ，単語情報のみに依存しないソースコード中の要素関係に基づくソースコード検索手法を提案する．

本研究の目的達成のためのアプローチとして，まず (1) 自然言語クエリの形式についての事前分析を行う．その結果として自然言語クエリには単語間の対応関係，およびソースコード中に現れない語彙の存在という 2 つの特徴が把握できた．提案手法ではそれらの分析結果に基づき， (2) 単語間の対応関係を表現する辺要素の追加，およびソースコード中に現れない語彙を補完する拡張クエリの追加という 2 つのアプローチを行い，ソースコード中のプログラム要素を組合せ集合として索引化する．構築したソースコード検索手法について有効性を確かめるために，それぞれの提案アプローチの効果に関する評価と，既存のコード検索エンジンおよびクエリ再構成の最新手法を組み合わせたコード検索ツールとの比較を評価実験にて行い，本研究における提案手法を検証していく．

以降，本論文では 2 章で本研究の背景について述べる．3 章では本手法を提案する前段階に行った自然言語クエリの事前分析について説明する．4 章では本研究の提案手法について説明する．5 章では本手法に関する評価実験について説明する．6 章では本手法の有効性および妥当性への脅威について議論する．7 章では本研究に関連する既存研究について紹介する．最後に，8 章でまとめと今後の課題について述べる．

2. 背景：自然言語による実装知識の検索

近年，情報検索技術の発展により自然言語を介した実装知識の検索を行う機会が増えている．web 検索エンジン [1, 3, 8] を用いてソースコード実装時の疑問を検索すると，多くの疑問に対する解説記事や公式ドキュメントを見つけることができる．これらの情報には，開発者の実装に関する疑問を解消する自然言語による記述が存在している．また実装に関する知識は実際のソースコード実装例と密接な関係にあり，自然言語による文章の中でたびたびソースコード例が出現しているのを確認できる．開発者はこの見つかった実装例を文章によって理解することで，疑問を解消して開発を進めていくことができる．

自然言語によって目的の実装に合致した情報を得るために，現在ではオンライン Q & A フォーラムでの意見交換の情報が特に多く用いられている．フォーラムでは 1 つの疑問について複数の回答が見られ，ある人が回答した実装方法に対して別の人が是非をコメントし，質問者にとって一番疑問の解消に役に立った回答や，その疑問に対する回答の中で重要度の高い回答が優先的に表示される．また蓄積されたフォーラムの情報を利用して，以前に質問された類似の疑問を検索して見つけ出すことで，同様の実装方法によって効率的に疑問を解消できる場合もある．このフォーラムによる情報ではソフトウェアの公式ドキュメントとは異なり，特定の端的な実装に関する疑問を素早く解消するためのソースコード例が得られるという利点がある．リスト 2.1 には具体例として，開発者の一般的な質問に対するフォーラムでの回答ソースコード例を挙げている．この例では，質問の記述に対してソースコード中の記述が異なるような，開発者の意図に沿った適切なソースコード例が示されている．このように web 検索エンジンとは異なる方法で，開発者は分からない部分の実装に対する疑問をフォーラム上で投げかけ，その部分に関する実装例を回答によって得ることができる．

こうした対話による疑問解消の需要に伴い，フォーラムの知識を利用したソースコードの検索に関する研究が多く行われている [24, 31, 33, 39]．フォーラムには自然言語による議論と対応づいたソースコード例が膨大に存在し，頻繁に議論されている内容をパターンとして認識することで，大衆の意見に基づいたソースコード例の提示を実現できる．さらに近年の自然言語処理技術を活用することで，

```
String yourText = "YOUR_TEXT";
BufferedImage bufferedImage = new BufferedImage(
    170, 30, BufferedImage.TYPE_INT_RGB);

Graphics graphics = bufferedImage.getGraphics();
graphics.setColor(Color.LIGHT_GRAY);
graphics.fillRect(0, 0, 200, 50);
graphics.setColor(Color.BLACK);
graphics.setFont(new Font("Arial Black", Font.BOLD, 20));
graphics.drawString(yourText, 10, 25);

ImageIO.write(bufferedImage, "jpg", new File(
    "path/to/image"));
```

リスト 2.1: フォーラムでの質問 “write text onto image in Java” に対する回答ソースコード例

既存のコード検索エンジンとは異なる開発者の疑問に基づく自然なクエリ表現が可能となる．一方で，フォーラムに記述されるソースコード例はごく一部であり，目的の機能を実装するために不足した記述がある場合だけではなく，本番環境では実装すべきではないその場しのぎで不安全な例，既にライブラリやパッケージが更新されて古くなった非推奨な方法，記述された API のライセンス違反などが含まれているという報告がある [17, 37, 42, 45]．このような特性から，現在フォーラムに含まれる実装知識の適切な活用方法が検討されている [38, 43]．

本研究ではこれらのフォーラムの知識として，フォーラム中の質問で記述されているような形式の自然言語クエリを対象に分析を行い，自然言語クエリに含まれる特徴的なパターンを把握する．また開発者の目的とする機能に対して十分な実装知識を補完するため，実際のプロジェクトで使用されている完全なソースコード例のみを検索結果として推薦するシステムを構築する．

3. 自然言語クエリの事前分析

3.1 対象データの概要

実際のプロジェクトに用いられているソースコードを自然言語クエリから推薦するために，2019 年 9 月に公開された CodeSearchNet [20] というデータセットを対象に調査する．このデータセットには，表 3.1 に示すような自然言語のコメントが記述されている 496,688 件の Java によるソースコード例と，それらをもとに専門家が収集した 99 個の英語による自然言語クエリが含まれている．ソースコード例は GitHub [2] のスター数・フォーク数に基づく人気度の高いプロジェクトから順にメソッド単位で収集され，短すぎるメソッドや記述が重複したメソッドなど，ソースコード推薦として適当でないようなメソッドは除去されている．また自然言語クエリに関しては，プログラミング言語に依らない汎用的なプログラミングに関する質問を Bing [1] および StaQC [40] から取得し，収集したソースコード例と関連するクエリのみ人手により厳選して収集されたデータとなっている．

データセット中にあるソースコード例のコメントは主に Javadoc の形式で存在しており，メソッド全体に対する要約や，そのメソッドの引数，戻り値についての説明が記述されている．今回のデータセットでは，ソースコード例と自然言語の対応付けを得るために用いられており，Javadoc の形式に従って要約記述以外の情報は除去されている．

3.2 自然言語クエリの形式によるパターンの抽出

ソースコード検索で使用される自然言語クエリの特徴を把握するため，著者自身が CodeSearchNet に含まれる 99 個全ての自然言語クエリを実際に見て，クエリの特徴について調べた．本調査ではクエリの形式に着目し，特徴的なパターンに応じて分類を行った．調査により把握できた自然言語クエリの形式および実際のパターン例，分類されたクエリ数を表 3.2 に示す．

CodeSearchNet の自然言語クエリ 99 個のうち，69 個のクエリにおいて 3 つ

表 3.1: CodeSearchNet のソースコード例に含まれる情報

変数名	値
repo	javalite/activeweb
path	activeweb/src/main/java/org/javalite/activeweb/freemarker/FreeMarkerTag.java
url	https://github.com/javalite/activeweb/blob/f25f589da94852b6f5625182360732e0861794a6/activeweb/src/main/java/org/javalite/activeweb/freemarker/FreeMarkerTag.java#L435-L445

code	protected Map session(){
	Map session;
	try{
	SimpleHash sessionHash = (SimpleHash)get(”session”);
	session = sessionHash.toMap();
	}catch(Exception e){
	logger().warn(”failed to get a session map in context, returning session without data!!!”, e);
	session = new HashMap();
	}
	return Collections.unmodifiableMap(session);
	}

code_tokens	[protected, Map, session, (,), {, Map, session, ;, try, {, SimpleHash, sessionHash, =, (, SimpleHash,), get, (, ”session”,), ;, session, =, sessionHash, ., toMap, (,), ;, }, catch, (, Exception, e,), {, logger, (,), ., warn, (, ”failed to get a session map in context, returning session without data!!!”, e,), ;, session, =, new, HashMap, (,), ;, }, return, Collections, unmodifiableMap, (, session,), ;, }]
docstring	Returns reference to a current session map.\n\n@return reference to a current session map.
docstring_tokens	[Returns, reference, to, a, current, session, map, .]
language	java
partition	train

表 3.2: CodeSearchNet の自然言語クエリにおける特徴的なパターン

形式	パターン例	クエリ数
名詞句のみ	priority queue, aes encryption, ...	26
A to B	convert int to bool, string to date, ...	13
動詞＋目的語	write csv, format date, ...	30

のパターンに分類することができた．これらのパターンに分類できないクエリは複数の副詞句や形容詞句などを持ち，単一の単純なパターンとして解釈することが困難なケースである．以下では，見つかった 3 つのパターンとその特徴について説明する．

まず，名詞句のみのパターンが 26 件見つかった．また名詞句のみではない場合でも，名詞句のみがクエリを解釈するのに十分な意味を持つクエリもこの中に含んでいる．このパターンは主に複合名詞の形でクエリが構成されており，名詞句中に自然言語の語彙には通常存在しないプログラミング固有の単語が多く使われている．プログラミング固有の単語には，それ単体で実装したい機能を読み取れる記述が多く見られた．

名詞句に分類できない場合で,「A to B」の形式を持つようなパターンが 13 件見つかった.「A to B」の前には動詞の「convert」が単語として出現するパターンも多く見られた．このパターンでは A から B への要素の変換を意味する記述がされており，多くの場合で A および B の要素としてプログラミング固有の単語が指定されているのを確認できた．

さらに「A to B」にも分類できないクエリのうち,「動詞＋目的語」の形式を持つようなパターンが 30 件見つかった．このパターンに分類されるクエリは，副詞句や形容詞句などを含まないか,「動詞＋目的語」の部分だけでクエリを解釈するのに十分な意味を持つものに限定している．クエリ中の動詞には「get」,「set」,「read」などの複数のクエリで現れる共通の語彙を見ることができた．

3.3 クエリのパターンに関連したソースコード例の特徴

自然言語クエリの特徴的なパターンと実際のソースコード例との関連性を理解するため，CodeSearchNet に含まれるそれぞれの自然言語クエリに対して適切だと考えられるソースコード例をフォーラムの 1 つである Stack Overflow [7] やソフトウェアリポジトリの GitHub を用いて目視での確認を行った．さらに自然言語クエリの特徴的なパターンに関連するソースコード例の特徴について考察した．

まず「A to B」のパターンおよび「動詞＋目的語」のパターンでは，自然言語クエリ中の単語間に注目すべき対応関係が見て取れることがわかった.「A to B」

```
boolean result = false; // Boolean indicating whether directory was set
File directory;         // Desired current working directory
directory = new File(directory_name).getAbsoluteFile();
if (directory.exists() || directory.mkdirs()) {
    result = (System.setProperty(
        "user.dir", directory.getAbsolutePath()) != null);
}
return result;
```

リスト 3.1: 自然言語クエリ “set working directory” と語彙が異なる例

のパターンに関連するソースコード例では，クラス A からクラス B への型の変換操作を行っている処理がよく見られ，A と B のクラス間で保有するデータに関する対応関係をそのメソッド全体で見ることができる．また「動詞＋目的語」のパターンでは，メソッド名もしくはクラス名など，キャメルケースで書かれた複合語の要素にクエリ中の単語が出現する場合が見られ，動詞と目的語に共起的な表現としての対応関係を複合語の識別子内で見ることができる．これらの単語間の対応関係ではクエリ中の特徴的なパターンが表現されており，ソースコード例の機能的な特徴を反映しているものと考えられる．したがって，これらの特徴的なパターンに含まれる対応関係を考慮したソースコード検索に関する索引を表現する必要がある．

また，ソースコードの実装例には出現しない語彙が自然言語クエリに指定される可能性があることも改めて確認できた．一部のクエリではリスト 3.1 のように，クエリ中の単語と異なる単語がソースコードで用いられている例も確認できた．この場合，既存のコード検索エンジンのような全文検索による手法だと，ソースコード中のキーワードと語彙が一致せず，目的のソースコード例が見つからない場合が考えられる．一方で，ソースコードのコメント中に自然言語クエリの記述が含まれる例も多く見られた．このことから，自然言語クエリの語彙に対応する一般的な実装例の語彙を補完するような知識としてソースコードコメントを活用できると考えられる．

4. 組合せ集合によるソースコード検索システム

本手法では 3 章で得られた知見に基づき，自然言語クエリを利用して目的のソースコード例を提示する検索システムを提案する．システムへの入力 q は自然言語による単語列である．これに対し，目的のソースコード例の一覧として順位付けされたメソッド集合 $M = \{m_1, m_2, ...\}$ が得られるソースコード索引を構築する．基本的なアプローチとしては，まず各メソッド m_i に含まれる特徴をプログラム要素の組合せ C_{m_i} として表現し，組合せ表現の集合 $CS = \{C_{m_i}\}$ による索引化を行う．さらに自然言語クエリ q の組合せ表現 C_q を用いて，索引から取得したメソッドの組合せ表現 C_{m_j} との関連度 $E(C_q, C_{m_j})$ を評価し，組合せ集合 $CS_{match} = \{C_{m_j}\}$ に対して優先順位を付ける．最終的にはそれぞれの組合せ表現 C_{m_j} に対応するメソッド m_j を転置索引を用いて取得し，優先度により整列されたメソッド集合 M の推薦を行う．

本研究における提案システムの設計を図 4.1 に示す．なお本提案システムは主に以下の 5 つの Step により構成される．

Step 1. プログラム要素の組合せ表現化：検索対象のメソッドをプログラム要素の組合せとして表現するため，語彙的な特徴である識別子の集合を抽出し，正規化処理を施す．自然言語クエリの単語列に関しても同様の処理を行う．

Step 2. 単語の対応関係に基づく辺要素の追加：3 章の結果に基づき，対応関係にある自然言語クエリの単語同士に辺を張り，機能的な特徴を反映する組合せ表現の追加を行う．これに伴い，メソッド上で対応関係にあるプログラム要素同士で同様の辺を事前に張り，単語間の対応関係を考慮したソースコード検索を行う．

Step 3. ソースコードコメントをもとにしたクエリの拡張：3 章の結果に基づき，自然言語クエリに直接現れない語彙を補完するため，プログラム要素とソースコードコメントの組を学習したニューラル機械翻訳による生成モデルを利用して，検索クエリの拡張を図る．

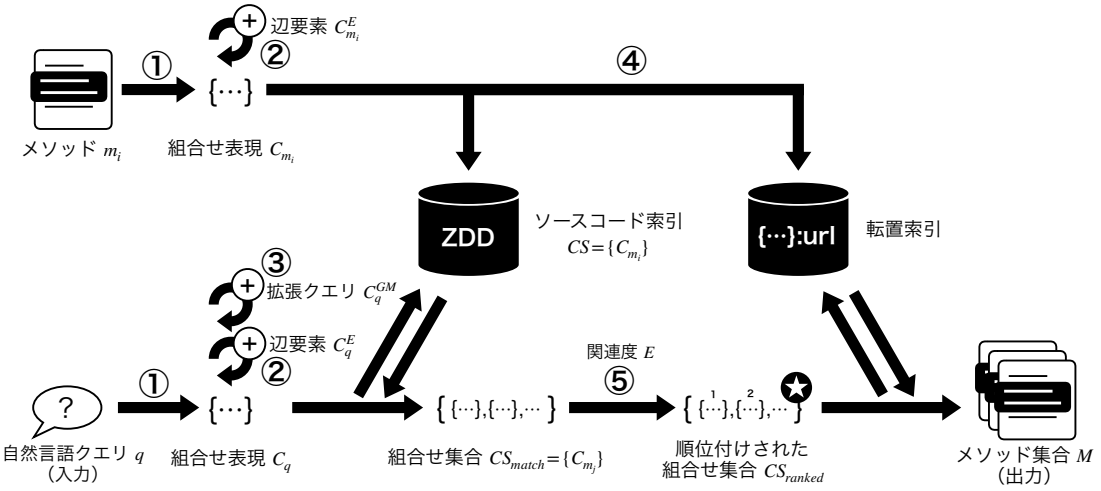


図 4.1: 提案手法の設計

Step 4. 組合せ集合によるソースコード索引化 : ZDD というデータ構造を用いて，全メソッドの組合せ表現の集合を 1 つの索引として表現する．さらに転置索引を作成することで組合せ表現に対応するソースコード例の抽出を行う．

Step 5. 推薦ソースコード例の順位付け：メソッドに含まれるプログラム要素に重み付けを行い，検索結果として列挙されたメソッドの組合せ表現に対して順位付けを行う．

また，各 Step によって再構成されたクエリの組合せ表現の具体例を表 4.1 に示す．

表 4.1: 再構成クエリの組合せ表現の例

q		“How do I decompress a GZip file in Java?”
C_q	(Step 1)	{how, do, i, decompress, a, g, zip, file, in, java}
C^E_q	(Step 2)	{decompress–file}
C^{GM}_q	(Step 3)	{byte, decompress, input, gzip, array, in, output, bao, except, stream, io}

4.1 Step 1. プログラム要素の組合せ表現化

ここでは，検索対象のメソッド m_i をプログラム要素の組合せ C_{m_i} として表現する処理を行う．プログラム要素をメソッドから抽出するため，静的構文解析ツール ANTLR[1] を用いて，Java の構文規則に従い目的のプログラム要素のみを選択している．なお本研究では，メソッド内の識別子で出現している単語のみをプログラム要素と呼ぶ．また全メソッドで取り得るプログラム要素の全体集合から，一部の集合を取り出したものを組合せと呼ぶ．

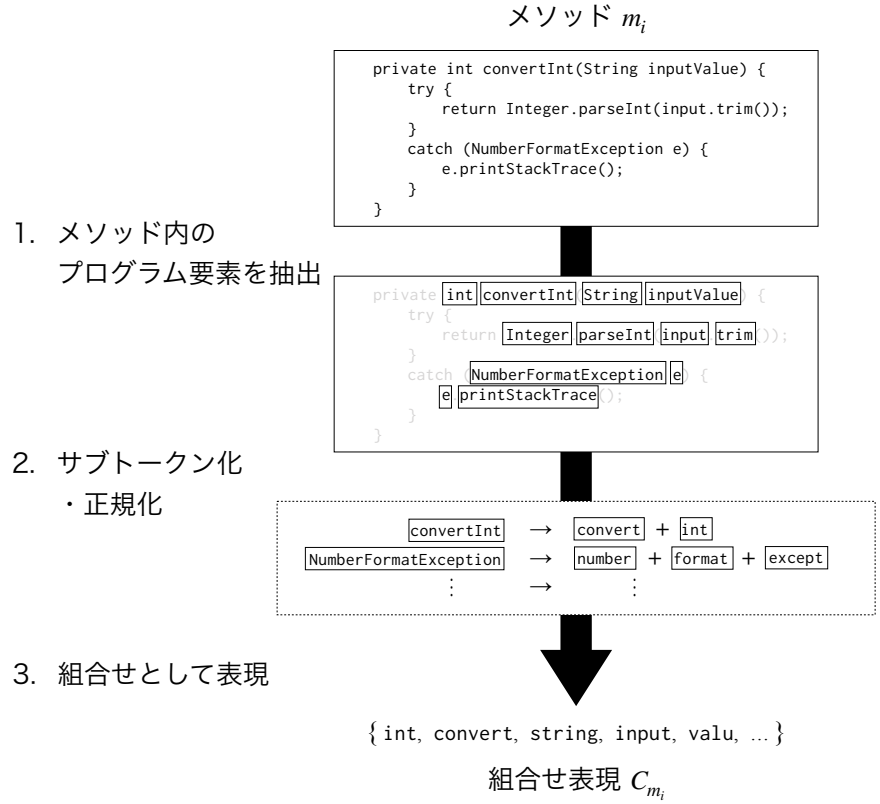


図 4.2: メソッドに含まれるプログラム要素の組合せ表現の抽出

具体的な処理としてまず，メソッドに含まれる識別子の抽出を行う．CodeSearchNet ではメソッドを単位としたソースコード例が提供されている．これに対し，

[1]ANTLR: https://www.antlr.org/

int や boolean などの Java の基本型および，変数名やクラス名，クラスのメソッドコール名のみを選択し，トークンの集合を収集する．

さらに，収集したトークン集合に対してサブトークンへの分割および正規化の処理を行う．Java の主な命名規則では，変数名やクラス名などをキャメルケースと呼ばれる，複数の単語に対してそれぞれの頭文字を大文字にして並べた一綴りの識別子の例が多く見られる．この規則を利用し，全てのトークンをキャメルケースによってトークン分割を行い，サブトークンの形で表現する．加えて語彙数の削減を図るために，NLTK[2] を用いてそれぞれのサブトークンに Lemmatization と Stemming の処理を行い，単語を正規化する [35]．まず最初に Lemmatization によって単語の活用形を統一し，その後 Stemming によって派生した同義の単語をまとめる処理を行っている．以上の一連の操作を行うことにより，メソッド内のプログラム要素を抽出している．

自然言語クエリに関しては，クエリに含まれる単語をメソッド中のプログラム要素と同等なものとして扱い，サブトークンへの分割および正規化処理を行うことで自然言語クエリの組合せ表現に変換している．

4.2 Step 2. 単語の対応関係に基づく辺要素の追加

機能的に類似したソースコード例を検索するために，3 章での分析で把握できた自然言語クエリ q 中の単語間の対応関係を辺要素 C_q^E として表現する．ソースコード索引では，あるメソッドの組合せ表現 C_{m_i} において形式的に対応がある 2 つの要素間でグラフとしての辺を張り，あらかじめ構成された組合せ表現に対して辺要素 $C_{m_i}^E$ を追加する．

本研究ではソースコード索引に ZDD というデータ構造を用いるため，組合せ表現として追加する辺要素の個数には制限がある（4.4 節を参照）．そこで本論文では ZDD の性質をうまく活用できた「動詞＋目的語」の形式を持つパターンにのみ着目し，「A to B」の形式を持つパターンを考慮の対象から外した．「A to B」のパターンでは互いに異なる識別子間で対応関係を持つため，プログラム要素のデータ依存を考える必要がある．したがって，プログラム要素の個数に対し

[2]NLTK: `http://www.nltk.org/`

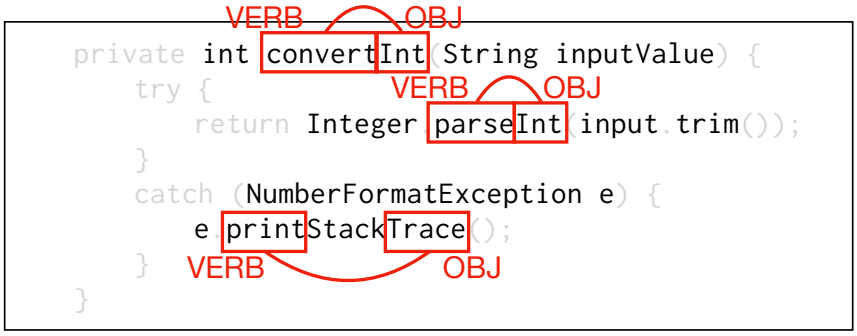


図 4.3: 「動詞」と「目的語」の対応関係にあるプログラム要素の組

追加される辺要素の数は非常に多くなり，ZDD の構築が困難となる可能性がある．実際に，予備実験としてデータフローを作成し ZDD への辺要素の追加を行ってみたが，使用した計算環境およびライブラリではメモリ制限により処理が強制終了した．なお，計算環境は 2.20 GHz Xeon®E5-2650 v4 ×2, 256 GiB RAM, Supermicro SYS-1028GR-TR を用いた．

「動詞＋目的語」のパターンでは，図 4.3 のようにサブトークン分割前の識別子内で「動詞」と「目的語」の関係にあるそれぞれのプログラム要素を自然言語処理ライブラリの spaCy[3] を用いて特定した．この対応にある要素間にのみ辺を張り，得られた辺要素の集合と Step 1 で構築した組合せ表現との和集合をとることで，新たに組合せ表現を再構築した．

4.3 Step 3. ソースコードコメントをもとにしたクエリの拡張

3 章で述べた自然言語クエリとソースコードで使用される単語の違いに対処するため，機械学習による生成モデルを利用する．ソースコードに記述されている自然言語のコメント文からソースコード中のプログラム要素の系列を出力するようにモデルを学習し，自然言語クエリ q 中に出現しないプログラム要素の組合せ表現 C_q^{GM} を追加する．この系列から系列への変換をするタスクは機械翻訳と呼ばれ，自然言語処理の分野でも多く取り組まれている．

[3]spaCy, `https://spacy.io/`

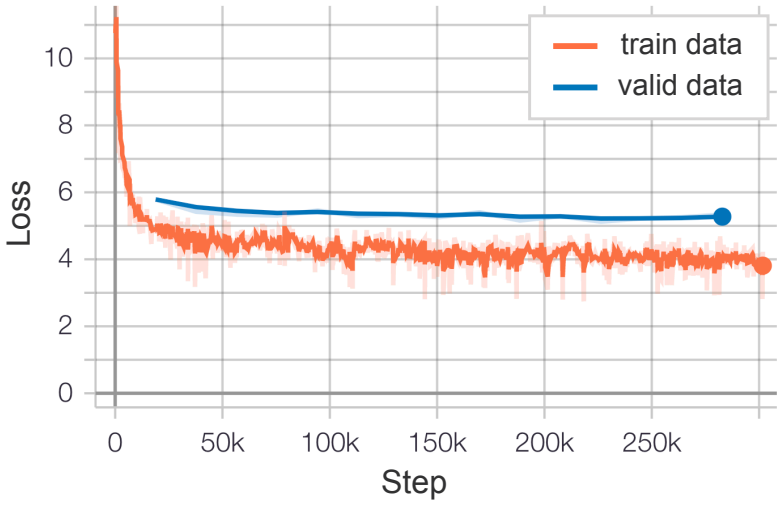


図 4.4: CodeSearchNet のデータを学習した生成モデルの学習曲線

本研究では，大量のデータと単一のモデルによって優れた評価が得られるニューラル機械翻訳を用いた手法を利用する．ニューラル機械翻訳では一般に，入力と過去に出力した単語の情報から次に出現する単語を学習済みの確率分布によって予測し，一番確率の高い単語を出力していく．このニューラル機械翻訳を行うモデル作成ツールとして fairseq[4] [32] を用いて，既にデータが学習用・テスト用・検証用に振り分けられた CodeSearchNet のデータを用いて生成モデルを学習する．

具体的には，表 3.1 の変数 “docstring” に格納されたソースコードコメントおよび，変数 “code” に格納されたメソッドのそれぞれに対して，Step 1 の前処理を加える．さらにソースコードコメントの組合せ表現からメソッドの組合せ表現に翻訳するようなモデルの学習を行う．これらの操作によって得られた生成モデルに対し，Step 1 で構築したクエリの組合せ表現を入力することで，メソッドに含まれるような語彙を補完している．なお生成モデルには CNN のアーキテクチャを利用し，ZDD と同じ計算環境に NVDIA Quadro P4000 GPU を用いて 27.3 時間 (16 epoch) 学習した．

その結果として，学習用および検証用データに対するそれぞれの損失曲線を図

[4]fairseq: `https://github.com/pytorch/fairseq`

4.4 に示す．オレンジ色の線は学習用データに対する損失であり，青色の線は検証用データに対する損失である．今回，30 万 step の学習で性能は十分と判断し，損失が最も小さなスナップショットを復元して生成モデルを利用する．

4.4 Step 4. 組合せ集合によるソースコード索引化

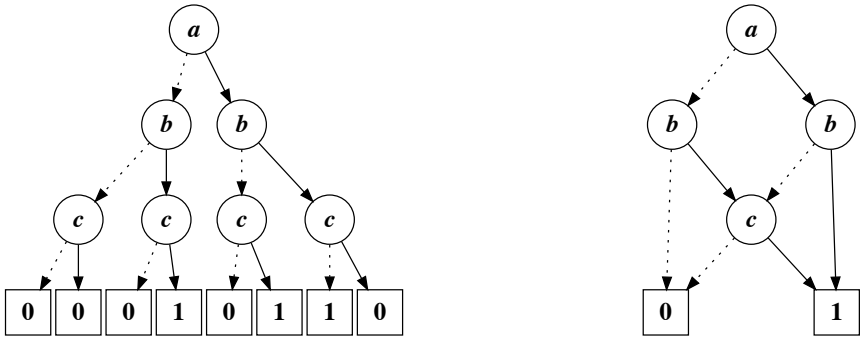
Step 1 および Step 2 で得られたメソッドの組合せ表現 C_{m_i}, $C^E_{m_i}$ を 1 つの索引で効率的に表現するため，ZDD を用いて組合せ集合 $CS = \{C_{m_i}\}$ として全メソッドを索引化する．

ZDD [29] は集合の集まりを表現するために用いられるデータ構造である．ZDD を用いることで図 4.5 のように，ある組合せの集合を表す二分木を簡約化した形で表現することができる．具体的には二分木状のグラフに対して，重複する節点の共有，冗長な節点の削除，という 2 つの簡約規則を可能な限り適用することで，グラフをコンパクトかつ一意に表現している．ZDD には組合せ集合を効率的に検索できる性質があり，ZDD を用いることで設定した問題の解集合を解の個数に依存せず高速に求めることができる．さらに，組合せ集合に対しさまざまな集合演算を適用することができ，必要な解集合を絞り込んで取得することが可能となる．一方，ZDD ではメモリの使用量が利用するデータに大きく依存するという制約も存在する．組合せ集合は本来二分木の形で構成されるため，出現する要素数 n に対し簡約化前のグラフの最悪空間計算量は $O(2^n)$ となる場合があり得る．したがって簡約化によりグラフ全体のサイズを小さくできない場合は膨大のメモリを消費する可能性がある．

本研究ではこの ZDD の性質を利用して，CodeSearchNet の全てのメソッドの組合せ表現を組合せ集合として索引化する．これを実現するため，ZDD の効率的な構築・操作が可能な Graphillion[5] というライブラリを用いる．このライブラリは組合せ集合をグラフ集合として構築しているため，組合せ表現に含まれる各要素を自己ループ辺の表現に置き換えて実装する．

また，ZDD によって表現したメソッドの組合せ集合から特定のソースコード例を検索するために，組合せをキーとしてメソッドの参照先を得ることが可能な転

[5]Graphillion: `http://graphillion.org/`



(a) 二分木　　(b) ZDD

図 4.5: 組合せ集合 $\{\{b, c\}, \{a, c\}, \{a, b\}\}$ を表す二分木および ZDD

置索引を作成する．1 つの組合せを文字列によって表現し，その後ハッシュ化の操作を行うことにより，その組合せと同じ形式を持つメソッドを高速に取得できるようにしている．また，CodeSearchNet のソースコード例は GitHub 上のソースコードに対応づいており，ソースコードの URL および行番号を参照することで実際のメソッドを取得できる．この転置索引では，メソッドに含まれるプログラム要素の集合と同じ形式をもった，実際のメソッドと対応づいた URL を記録する形で構成している．

ソースコード索引を構築するにあたり，全体の組合せ集合を 1 つの ZDD として表現するため，ZDD が保持する変数の個数は全てのメソッドから抽出されるプログラム要素の語彙数と一致する．1 つのメソッドに出現するプログラム要素の個数は全てのメソッドに対する語彙数と比べて非常に少ないため，疎な組合せ集合によって効率的な表現が可能だと考えられる [46].

4.5 Step 5. 推薦ソースコード例の順位付け

検索クエリの組合せ表現 C_q からソースコード例 M の推薦を行うために，メソッドの組合せ表現に含まれるプログラム要素を用いて組合せ表現の順位付けを行う．まず，IDF によりそれぞれのプログラム要素 t の重み $idf(t)$ を 式 4.1 の

ように定義する．

$$idf(t) = \log\left(\frac{\text{全メソッド数} + 1}{\text{プログラム要素 } t \text{ が含まれるメソッド数} + 1}\right) + 1 \tag{4.1}$$

また検索クエリの組合せ表現 C_q を用いて，あるメソッドの組合せ表現 C_{m_i} との関連度 $E(C_q, C_{m_i})$ を式 4.2 のように計算し，関連度の高いメソッドの組合せ表現から優先的に推薦する．

$$E(C_q, C_{m_i}) = \sum_{t \in C_q \cap C_{m_i}} idf(t) \tag{4.2}$$

これらの式を用いて，検索クエリに対するソースコードの順位付けを行い，検索結果としてソースコード例を推薦する．まずメソッドの推薦候補として，検索クエリの組合せ表現 C_q の要素を少なくとも 1 つ以上含むような組合せ表現の集合 $CS_{match} = \{C_{m_j}\}$ をソースコード索引の ZDD によって列挙する．この組合せ集合のそれぞれの要素 C_{m_j} を関連度 E によって優先度順に並び替え，順位付けされた組合せ集合 CS_{ranked} を得る．最終的にはそれぞれの要素 C_{m_j} に対し，転置索引を用いて収集したメソッド集合 M を推薦する．なお得られたメソッド集合の関連度 E が同じ場合には，実際の記述量が少ないメソッドを優先度が高いものとして並び替えを行っている．

5. 評価実験

本章では，自然言語クエリにおけるソースコード検索を実際に行い，4 章で述べた提案手法についての実験を行う．実験には共通の評価データセットを用いて，それぞれの提案アプローチの効果に関する評価と，既存の検索ツールとの比較評価を行う．

5.1 実験設定

評価データセットとしては，Rahman [33] らが手動で収集した 310 個の自然言語による検索クエリと，そのクエリに関連する API クラスの集合の組を用いる．検索クエリは Kode Java[6]，Java2s[7]，CodeJava[8]，JavaDB[9] の 4 つのプログラミングチュートリアルサイトが利用され，チュートリアルサイト上のプログラミングタスクに関する質問のタイトルから収集されている．また，ソースコードの説明文に記載されている API クラスがそれぞれのプログラミングタスクに関連する固有のものであるとして，そのクエリについて手動で列挙した API クラス集合を正解データと定められている．

本論文では，ソースコード検索に広く採用されている以下の 3 つのランキング評価指標を用いて提案手法を評価する．

- Top-K Accuracy / Hit@K：上位 K 個の推薦結果のうち，正解データの要素を少なくとも 1 つ含む検索クエリの割合である．
- Mean Reciprocal Rank@K (MRR@K)：MRR@K は全ての検索クエリ Q についての Reciprocal Rank@K (RR@K) の平均値である．RR@K とは検索クエリ $q \in Q$ に対する上位 K 個の推薦結果のうち，正解データの上位から数えた最初の順位 R の逆数である．なお，MRR@K は以下のような式で表される．

[6]Kode Java: https://kodejava.org/
[7]Java2s: http://www.java2s.com/
[8]CodeJava: https://www.codejava.net/
[9]JavaDB: http://www.javadb.com/

$$\mathrm{MRR@K} = \frac{1}{|Q|} \sum_{q \in Q} \frac{1}{R_q} \tag{5.1}$$

- Mean Average Precision@K (MAP@K) : MAP@K は全ての検索クエリ Q についての Average Precision@K (AP@K) の平均値である．AP@K とは検索クエリ $q \in Q$ に対する上位 K 個の推薦結果のうち，正解データの出現時点を閾値とした全ての閾値ごとの Precision の平均をとった値である．なお，AP@K および MAP@K は以下のような式で表される．

$$\mathrm{AP}_q\mathrm{@K} = \left(\sum_{k=1}^{\mathrm{K}} P(k) \cdot I_q(k) \right) \left(\sum_{k=1}^{\mathrm{K}} I_q(k) \right)^{-1} \tag{5.2}$$

$$\mathrm{MAP@K} = \frac{1}{|Q|} \sum_{q \in Q} \mathrm{AP}_q\mathrm{@K} \tag{5.3}$$

ここで用いられている $P(k)$ は式 5.4 によって計算される上位 k 個の推薦結果に対する Precision である．また $I_q(k)$ は検索クエリ q に対する順位 k の推薦結果において，正解データを含む場合は 1 を，そうでない場合は 0 を返す指示関数である．

$$P(k) = \frac{\text{上位 } k \text{ 個の推薦結果のうち正解データを含むメソッド数}}{\text{上位 } k \text{ 個で推薦したメソッド数}} \tag{5.4}$$

なお，上位 k 個の推薦結果に正解データが 1 つも含まれない場合（式 5.2 の第 2 因数に含まれる $I_q(k)$ の総和が 0 となる場合）の AP@K の値は 0 として評価される．

5.2 提案アプローチの効果に関する評価

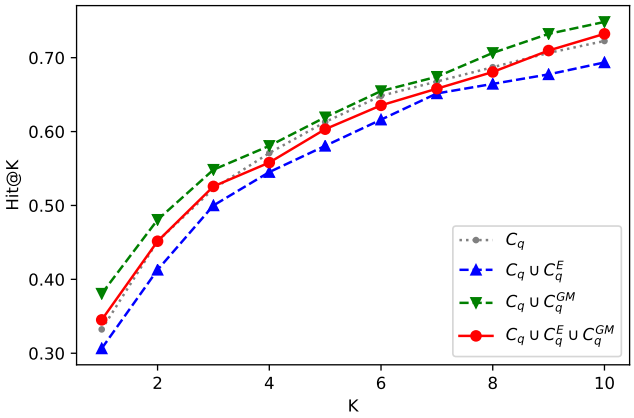
4 章で述べた本提案手法の各アプローチによってソースコード検索システムがどのくらい改善されるかを調べるため，それぞれの提案アプローチについての比較実験を行う．具体的には，検索クエリ q に Step 1 の処理を行った組合せ表現 C_q を基準とし，Step 2 の単語間の対応関係に基づく辺要素 C_q^E と，Step 3 の生成モデルでの拡張クエリ C_q^{GM} の 2 つの組合せ表現のべき集合について，上位 1 個から 10 個までソースコード例を推薦したときのランキング評価を比較する．その結果を図 5.1 に示す．

また，3 章によって得られたパターンに対する組合せ表現がソースコード検索の結果に反映できているかを確かめるため，パターンに基づいて評価データセットの目視による分類を行った．さらに,「動詞＋目的語」のパターンのみを持つ 86 個のクエリに対し，同様のランキング評価を行った．その結果を図 5.2 に示す．

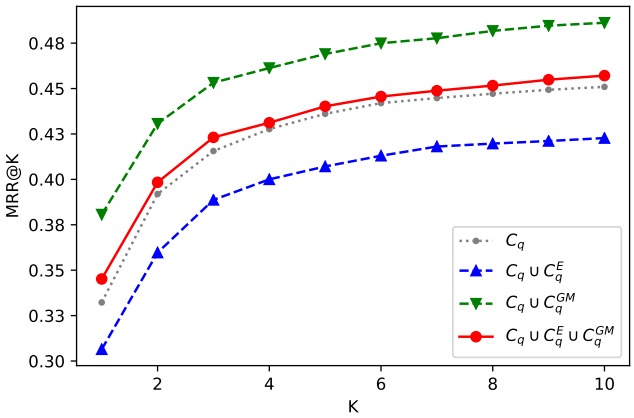
まず Step 2 で提案した，辺要素の追加のみを行った組合せ表現 $C_q \cup C_q^E$ によるアプローチでは，全体のクエリを用いたときに Hit@K, MRR@K, MAP@K の全ての評価において基準のアプローチを下回っていた．一方パターン「動詞＋目的語」のクエリのみを用いた場合，どの評価においても Step 2 のみ追加したアプローチの値はほとんど変化しなかったものの，基準のアプローチの値は下がっており，相対的に Step 2 のみ追加したアプローチが基準を上回っていた．これは K が小さいほど大きな差がみられた．

Step 3 で提案した，拡張クエリの追加のみを行った組合せ表現 $C_q \cup C_q^{GM}$ によるアプローチでは，全体のクエリを用いたときにどの評価においても基準のアプローチおよび Step 2 のみ追加したアプローチの両方の評価を上回っていた．これは「動詞＋目的語」のパターンのみにクエリを限定しても同様であったが，ランキング評価の値は絞り込みにより下がっているのを確認できた．

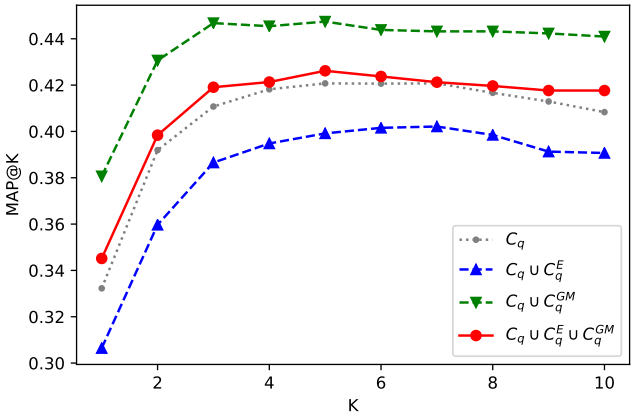
Step 2 および Step 3 の両方の追加を行った組合せ表現 $C_q \cup C_q^E \cup C_q^{GM}$ によるアプローチは，全体のクエリを用いたときにどの評価においても Step 3 のみ追加したアプローチを下回り，基準のアプローチとほとんど等しい値をとっていることが確認できた．一方パターン「動詞＋目的語」のクエリのみを用いた場合，どの評価においても Step 2 および Step 3 を追加したアプローチの値はほとんど



(a) Hit@K

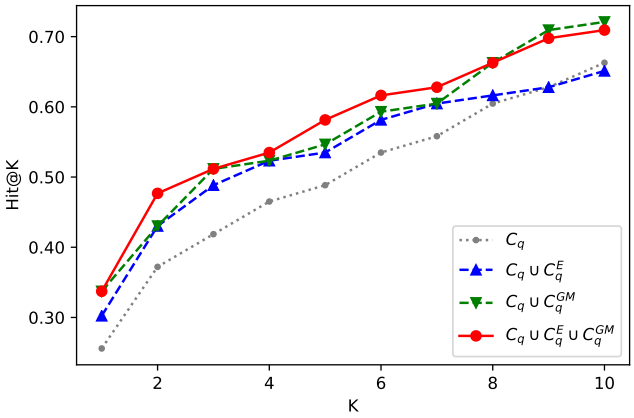


(b) MRR@K

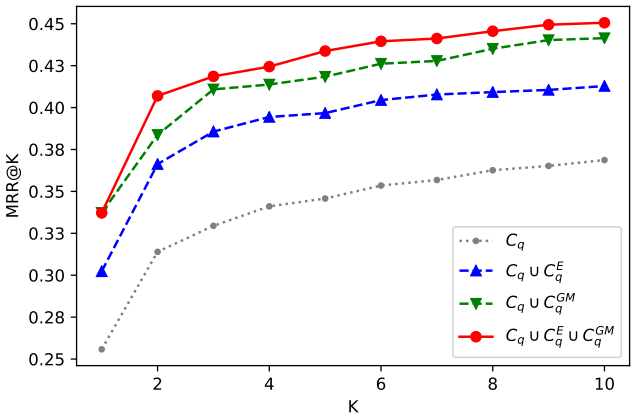


(c) MAP@K

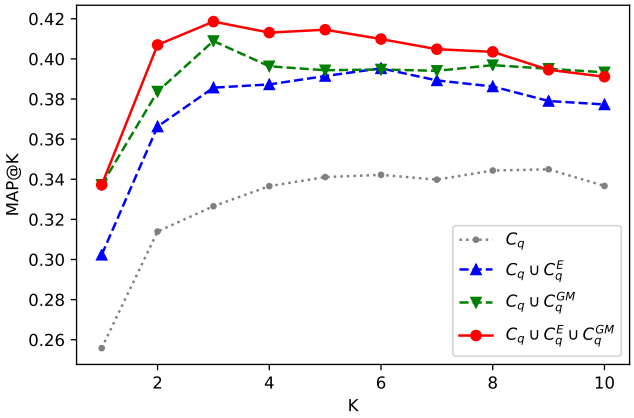
図 5.1: 提案アプローチに関する比較



(a) Hit@K



(b) MRR@K



(c) MAP@K

図 5.2: 提案アプローチに関する比較（パターン「動詞＋目的語」のクエリのみ）

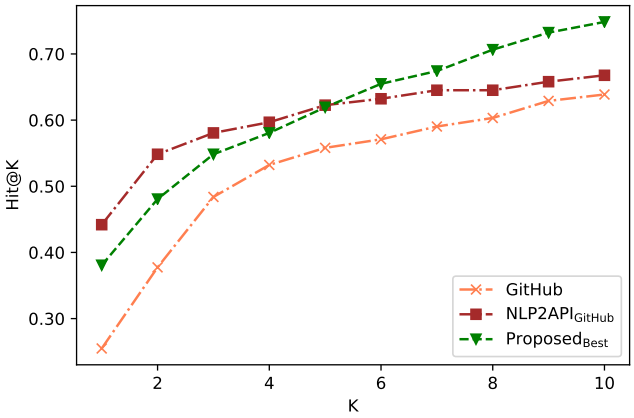
変化しなかったため，相対的に Step 3 のみ追加したアプローチを $2 \leq \mathrm{K} \leq 8$ の範囲でわずかに上回っていた．

5.3 既存のソースコード検索ツールとの比較

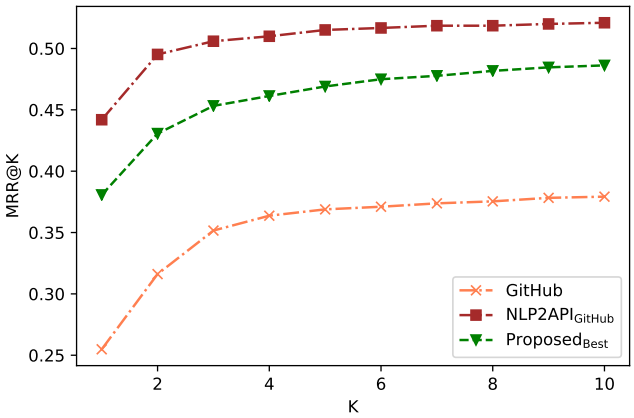
次に，本提案手法が既存のソースコード検索手法よりも優れた結果を推薦できるかを調べるため，既存ツールとの比較実験を行う．本実験では比較する既存ツールとして，現在広く利用されているコード検索エンジンの GitHub と，Rahman らが提案した検索クエリを再構成する手法 NLP2API [33] による検索結果 $\mathrm{NLP2API_{GitHub}}$ の 2 つを対象として用いる．本提案手法のメソッド単位による検索と条件を合わせるために，GitHub では GitHub API によって検索クエリと最も関連しているメソッドを抽出した．また $\mathrm{NLP2API_{GitHub}}$ では，同様の方法で得られる GitHub の上位 30 件のメソッドを対象に，NLP2API の再構成クエリを用いて再度メソッドの順位付けを行った．これらの操作により得られる検索結果と 5.2 節で得られた全体のクエリに対する最良のアプローチに基づき，上位 1 個から 10 個までソースコード例を推薦したときのランキング評価を比較する．その結果を図 5.3 に示す．

本研究で提案した最良のアプローチによる手法は，どの評価においても全文検索による既存のコード検索ツールの GitHub を上回っていた．Hit@K では，5–13% の差が見られ，K が大きくなるにつれ差が離れていくことが確認できた．MRR@K および MAP@K では，それぞれ 10–13%, 7–13% の差が見られ，K が小さいほど大きな差であることが確認できた．

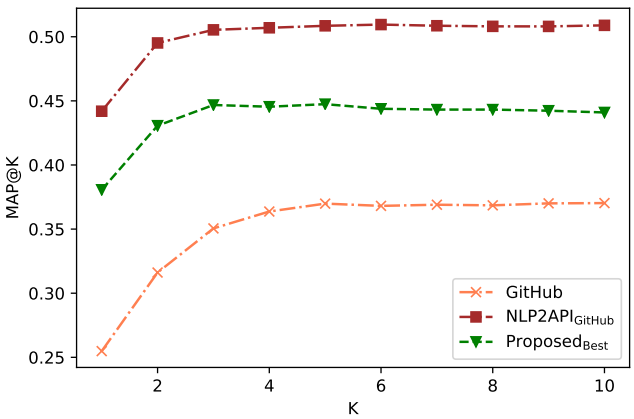
また提案手法は，Hit@K は K が小さいときに $\mathrm{NLP2API_{GitHub}}$ の手法よりも劣っているが，$\mathrm{K} = 5$ を境に K が大きくなるにつれて $\mathrm{NLP2API_{GitHub}}$ の手法を上回るようになった．さらに $\mathrm{K} = 10$ のときでは，74.8% のクエリに対して正解データを含むソースコード例を提示できることを確認でき，$\mathrm{NLP2API_{GitHub}}$ の手法よりも 8 ポイント高い性能が得られた．MRR@K および MAP@K は，いずれの K においても $\mathrm{NLP2API_{GitHub}}$ の手法よりそれぞれ 3–6%, 6–7% の差で劣っていた．また K が大きくなるにつれて MRR@K では差が縮まり，MAP@K では差が離れていくことが確認できた．



(a) Hit@K



(b) MRR@K



(c) MAP@K

図 5.3: 既存ツールとの比較

6. 議論

6.1 本手法の有効性

提案アプローチに関する比較評価では，Step 3 のみ追加したアプローチがいずれの評価においても高い値をとっていたことから，生成モデルによるクエリ拡張はソースコード検索の改善に寄与していると考えられる．またパターン「動詞＋目的語」のクエリのみを用いて評価を適用した場合に，Step 2 を用いないアプローチは Step 2 を用いたアプローチと比べて値を大きく下げてしまうことから，パターン「動詞＋目的語」のクエリに対して単語間の対応関係を反映したアプローチが効果的に作用しているとわかる．したがって，事前分析に基づいて提案したそれぞれのアプローチはソースコード検索に有効であるといえる．しかしながら，全体のクエリを用いた場合に，Step 2 と Step 3 を組合せたアプローチが Step 3 のみによるアプローチより劣っているため，パターン「動詞＋目的語」に分類されないクエリに対し，単語の対応関係を考慮したアプローチを組み込むことは有効でないと考えられる．

既存ツールとの比較評価では，本提案手法がいずれの評価においても既存のコード検索エンジンを上回っており，GitHub で用いられる全文検索の手法より有効であることが示された．また本手法は，NLP2API を用いた推薦結果の再順位付けによる手法と比べ，MRR@K および MAP@K の値が劣っているものの，K が大きい場合に Hit@K の値が優れているため，推薦の個数が多い場合に自然言語クエリに対して正しいとされる実装例を提示できる可能性が高いといえる．このことから，本手法が既存手法よりも自然言語クエリをより広く網羅しているという面で有効性を示している．

6.2 妥当性への脅威

本実験では内的妥当性への脅威がある．事前分析では目視による自然言語クエリの分類を行い，形式における特徴的なパターンを抽出している．この得られたパターンについて，実際のソースコード例をフォーラム上のクエリに関連する

回答を確認したが，回答のソースコード例が目的通り動作するものかは検証していない．したがって，実際に目的通り動作する回答例を確認した場合と異なるパターンの特徴が分析によって得られる可能性がある．さらに評価実験で用いる評価データセット中のクエリは，CodeSearchNet により得られる自然言語クエリの長さが平均的に異なり，パターンの分類および本研究で提案した手法による影響が異なる可能性がある．本研究では，評価データセット中の長いクエリ（目的語の節に含まれるストップワードを除く単語数が 5 以上のもの）を「動詞＋目的語」のパターンとして分類しないようにし，CodeSearchNet のクエリ形式と近いクエリをなるべく採用した．

本実験では外的妥当性への脅威がある．自然言語クエリの事前分析では，分析対象として CodeSearchNet の 99 個のクエリを用いたが，これらのクエリは完全な文ではなく実際のクエリとは異なる可能性がある．本論文では，クエリの分析によって得られた知見が分析していないクエリでも成立するか確認するために，評価には異なるデータセットを用いた．提案手法の評価では，正解データを API クラスの有無によって判断したが，API を用いずに独自で実装したソースコード例の検出は行えていない．また，今回実装した ZDD による検索アルゴリズムの速度はクエリに含まれる組合せ表現の要素数に大きく依存し，評価データセットの自然言語クエリに対する検索の平均時間および標準偏差時間はそれぞれ 9.4 秒，7.2 秒であった．最大で 33.6 秒かかるケースも見られたため，本実装による検索システムでは実用的な検索時間の観点で実際の自然言語クエリに対応できない可能性がある．

7. 関連研究

7.1 ソースコード検索

ソースコード検索技術は，ソフトウェア開発の支援を目的として様々な手法が提案されている．既存のソースコード検索ツールの多くはソースコードに含まれる構成要素の索引化に焦点を当てている．Inoue ら [21] はソフトウェア中の要素間の類似性や依存関係を考慮したソフトウェア要素検索ツール SPARS-J を開発した．Bajracharya ら [11] は大規模なソースコードのキーワードや構造情報を収集したソースコード検索基盤ツール Sourcerer を開発した．本研究では，検索の入力において自然言語クエリに対応することを目的とし，索引と完全には一致しないようなクエリを考慮に入れたソースコード検索を行う．これに対し，Rahman ら [33] は自然言語クエリからフォーラム上の関連する API クラスを自動識別することにより，検索クエリを再構成し既存のコード検索エンジンを改善した．我々はソースコード中の構成要素を組合せ集合として表現することにより，クエリの再構成を行った．この再構成したグラフ形式のクエリからソースコードを効率よく検索するという点で，既存のソースコード検索ツールとは異なるアプローチを提案する．

近年では，機械学習技術を用いたソースコードのモデル化について広く研究されている [9, 15]．Gu ら [19] はソースコードと自然言語の説明文を同じベクトル空間上に埋め込み，自然言語クエリに関連するソースコードをベクトルの類似度に従って取得できる手法を提案した．Chen ら [16] はソースコードと自然言語を双方向にマッピングする生成モデルを学習することにより，自然言語クエリに対するソースコード検索を実現した．本研究は実際の自然言語クエリに対して，索引による検索結果が機械学習技術を用いた手法よりも優れていると仮定し [20]，自然言語クエリの再構成方法にのみ取り組んでいる．

7.2 類似ソースコードの検出

ソースコードの再利用やバグ修正のパッチ推薦，リファクタリングを目的とした，類似ソースコードの検出に関する研究が取り組まれている．Kim ら [24] は入力のソースコード片と機能的に類似したソースコードを検索する手法として，フォーラムの投稿に基づき入力したソースコードの機能の最適な説明を抽出し，検索クエリを代替するツール FaCoY を開発した．Zhang ら [41] はソースコードを抽象構文木に変換し，語彙的・構文的特徴を捉えたベクトルに符号化することで，ソースコードの類似性をベクトル表現によって比較する手法を提案した．Zhao ら [44] はソースコードの制御フローとデータフローを高次元・疎な特徴行列に符号化し，機能的に類似したソースコードを検出する手法を提案した．本研究は自然言語クエリからソースコードを推薦するシステムであり，実装意図を適切に明示した検索を行っている点で異なる．

7.3 API パターンマイニング

実装知識の補完を目的として，大量のソースコードからアプリケーションプログラミングインタフェース (API) の使用パターンをマイニングする研究が行われている．Cai ら [14] は自然言語クエリの指定によりフォーラムで類似する API 関連の質問を検索し，ドキュメントとの類似性を考慮することで順位付けされた候補 API を推薦するツール BIKER を開発した．Liu ら [27] は入力の自然言語クエリに対し，最も関連性が高い文と最も関連性が高いメソッドが含まれる要約を生成する手法を提案した．本研究では実際のソースコードを検索し，完全な実装例を提示しているため用途が異なる．

8. おわりに

本論文では自然言語クエリを用いたソースコード検索に対し，クエリの事前分析に基づくソースコード表現の索引化を手法として提案した．本研究における事前分析では自然言語クエリの形式に対する抽出を行い，大きく 3 つのパターンに分類できた．本研究ではこの分析の中で特定できた，単語間の対応関係とソースコード中に現れない語彙の存在という 2 つの特徴を扱い，ソースコード中のプログラム要素を組合せ集合として表現した．さらにそれぞれの特徴に対応するアプローチとして,「動詞＋目的語」の対応関係に関する辺要素の追加と存在しない語彙に対するクエリの拡張を行い，これらを組合せたソースコード検索手法の評価を行った．

評価実験では，2 つの提案アプローチに対する比較と，既存のコード検索エンジン GitHub および Rahman らが提案したクエリ再構成によるツールとの比較の 2 つを一般的なランキング評価指標を用いて行った．提案アプローチによる比較では，単語間の対応関係に対するアプローチが一部のクエリにおいてソースコード検索の改善に貢献していることがわかった．さらに語彙の拡張に対するアプローチが全体のクエリを通してソースコード検索の改善に大きく影響を及ぼしていることがわかった．既存ツールとの比較では， 74.8% のクエリに対して上位 10 件以内に正解のソースコード例を提示でき，既存手法より 8 ポイント高い性能を達成した．このことから，本提案システムがさまざまな自然言語クエリに対する網羅性という観点で優れた結果を得られることがわかった．これらの結果は，ソースコード中の要素関係に基づく索引化および検索手法の新たな価値を見出したといえる．

本手法における今後の課題として，パターンの認識に対する自動化と検索結果を推薦するランキングアルゴリズムの改善が挙げられる．評価実験において,「動詞＋目的語」のパターンに分類されないクエリに対する検索結果に悪影響を及ぼしている可能性があるため，クエリのパターンを自動的に認識することで各アプローチを適用するシステムがより効果的だと考えられる．また本手法は，既存手法と比べてより上位の推薦結果のランキング評価が劣っているため，ランキング部分に関するアルゴリズムにおいて改善の余地があると考えられる．

謝辞

本論文は筆者が先端科学技術研究科情報科学領域博士前期課程に在籍中の研究成果をまとめたものである．同研究科ソフトウェア工学研究室教授 松本 健一 先生には，本研究実施の機会を与えて頂き，またその遂行にあたって様々なご指導・ご助言を頂いた．ここに感謝の意を表する．同研究科大規模システム管理研究室教授 笠原 正治 先生には，副審査委員として研究の大筋に関する貴重なご助言を頂いた．ここに感謝の意を表する．同研究科ソフトウェア工学研究室准教授 石尾 隆 先生 には，本研究の手法の提案にご協力頂き，研究を進める上での有益なご助言を頂いた．ここに感謝の意を表する．同研究科ソフトウェア工学研究室助教 畑 秀明 先生には，本研究の方針をはじめ細部にわたって議論を尽くし，多大なご指導・ご助言を頂いた．ここに感謝の意を表する．本研究における手法では京都大学情報学研究科通信情報システム専攻コンピュータアルゴリズム分野准教授 川原 純 先生にご相談頂き，理解を深める上で重要な知識をご教示頂いた．ここに感謝の意を表する．

また 2 年間の有意義な研究活動を過ごすにあたって，同研究科ソフトウェア工学研究室およびソフトウェア設計学の皆様には日頃より多くの相談や議論をして頂き．研究に対する自らの理解を深めるに至った．この活動を通じてご協力いただいた皆様に感謝の意を表する．最後に，本大学院への進学を受け入れ，様々なご支援をして頂いた家族に心よりの謝意を表する．

参考文献

[1] Bing. https://bing.com/.

[2] GitHub. https://www.github.com/.

[3] Google. https://www.google.com/.

[4] Krugle. https://www.krugle.com/.

[5] searchcode. https://searchcode.com/.

[6] SourceForge. https://sourceforge.net/.

[7] Stack Overflow. https://stackoverflow.com/.

[8] Yahoo! JAPAN. https://www.yahoo.co.jp/.

[9] Miltiadis Allamanis, Earl T. Barr, Premkumar Devanbu, and Charles Sutton. A survey of machine learning for big code and naturalness. *ACM Computing Surveys*, 51(4), 2018.

[10] Sushil K. Bajracharya and Cristina V. Lopes. Analyzing and mining a code search engine usage log. *Empirical Software Engineering*, 17(4-5):424–466, 2012.

[11] Sushil K. Bajracharya, Joel Ossher, and Cristina V. Lopes. Sourcerer: An infrastructure for large-scale collection and analysis of open-source code. *Science of Computer Programming*, 79:241–259, 2014.

[12] Victor R. Basili, Lionel C. Briand, and Walcélio L. Melo. How reuse influences productivity in object-oriented systems. *Communications of the ACM*, 39(10):104–116, 1996.

[13] Joel Brandt, Philip J. Guo, Joel Lewenstein, Mira Dontcheva, and Scott R. Klemmer. Two studies of opportunistic programming: Interleaving web foraging, learning, and writing code. In *Proceedings of the Conference on Human Factors in Computing Systems*, page 1589–1598, 2009.

[14] Liang Cai, Haoye Wang, Qiao Huang, Xin Xia, Zhenchang Xing, and David Lo. BIKER: A tool for Bi-information source based API method recommendation. In *Proceedings of the Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 1075–1079, 2019.

[15] Jose Cambronero, Hongyu Li, Seohyun Kim, Koushik Sen, and Satish Chandra. When deep learning met code search. In *Proceedings of the Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 964–974, 2019.

[16] Qingying Chen and Minghui Zhou. A neural framework for retrieval and summarization of source code. In *Proceedings of the International Conference on Automated Software Engineering*, pages 826–831, 2018.

[17] Felix Fischer, Konstantin Böttinger, Huang Xiao, Christian Stransky, Yasemin Acar, Michael Backes, and Sascha Fahl. Stack overflow considered harmful? the impact of copy paste on android application security. In *Symposium on Security and Privacy*, pages 121–136, 2017.

[18] George W. Furnas, Thomas K. Landauer, Louis M. Gomez, and Susan T. Dumais. The vocabulary problem in human-system communication. *Communications of the ACM*, 30(11):964–971, 1987.

[19] Xiaodong Gu, Hongyu Zhang, and Sunghun Kim. Deep code search. In *Proceedings of the International Conference on Software Engineering*, pages 933–944, 2018.

[20] Hamel Husain, Ho-Hsiang Wu, Tiferet Gazit, Miltiadis Allamanis, and Marc Brockschmidt. CodeSearchNet Challenge: Evaluating the state of semantic code search. *arXiv e-prints*, page arXiv:1909.09436, 2019.

[21] Katsuro Inoue, Reishi Yokomori, Tetsuo Yamamoto, Makoto Matsushita, and Shinji Kusumoto. Ranking significance of software components based on use relations. *IEEE Transactions on Software Engineering*, 31(3):213–225, 2005.

[22] Iman Keivanloo, Juergen Rilling, and Ying Zou. Spotting working code examples. In *Proceedings of the International Conference on Software Engineering*, page 664–675, 2014.

[23] Katja Kevic and Thomas Fritz. Automatic search term identification for change tasks. In *Companion Proceedings of the International Conference on Software Engineering*, page 468–471, 2014.

[24] Kisub Kim, Dongsun Kim, Tegawendé F. Bissyandé, Eunjong Choi, Li Li, Jacques Klein, and Yves L. Traon. FaCoY: A code-to-code search engine. In *Proceedings of the International Conference on Software Engineering*, pages 946–957, 2018.

[25] Andrew J. Ko, Brad A. Myers, Michael J. Coblenz, and Htet H. Aung. An exploratory study of how developers seek, relate, and collect relevant information during software maintenance tasks. *IEEE Transactions on Software Engineering*, 32(12):971–987, 2006.

[26] Wayne C. Lim. Effects of reuse on quality, productivity, and economics. *IEEE Software*, 11(5):23–30, 1994.

[27] Mingwei Liu, Xin Peng, Andrian Marcus, Zhenchang Xing, Wenkai Xie, Shuangshuang Xing, and Yang Liu. Generating query-specific class API

summaries. In *Proceedings of the Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 120–130, 2019.

[28] Lee Martie, André van der Hoek, and Thomas Kwak. Understanding the impact of support for iteration on code search. In *Proceedings of the Joint Meeting on Foundations of Software Engineering*, page 774–785, 2017.

[29] Shin-ichi Minato. Zero-suppressed BDDs for set manipulation in combinatorial problems. In *Proceedings of the International Design Automation Conference*, pages 272–277, 1993.

[30] Parastoo Mohagheghi, Reidar Conradi, Ole M. Killi, and Henrik Schwarz. An empirical study of software reuse vs. defect-density and stability. In *Proceedings of the International Conference on Software Engineering*, pages 282–291, 2004.

[31] Liming Nie, He Jiang, Zhilei Ren, Zeyi Sun, and Xiaochen Li. Query expansion based on crowd knowledge for code search. *IEEE Transactions on Services Computing*, 9(5):771–783, 2016.

[32] Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. fairseq: A fast, extensible toolkit for sequence modeling. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2019.

[33] Mohammad M. Rahman and Chanchal K. Roy. Effective reformulation of query for code search using crowdsourced knowledge and extra-large data analytics. In *Proceedings of the International Conference on Software Maintenance and Evolution*, pages 473–484, 2018.

[34] Caitlin Sadowski, Kathryn T. Stolee, and Sebastian Elbaum. How developers search for code: A case study. In *Proceedings of the Joint Meeting on Foundations of Software Engineering*, page 191–201, 2015.

[35] Hinrich Schütze, Christopher D. Manning, and Prabhakar Raghavan. *Introduction to information retrieval*. Cambridge University Press, 2008.

[36] Janice Singer, Timothy Lethbridge, Norman Vinson, and Nicolas Anquetil. An examination of software engineering work practices. In *Conference of the Center for Advanced Studies on Collaborative Research*, page 174–188, 2010.

[37] Christoph Treude and Martin P. Robillard. Understanding stack overflow code fragments. In *Proceedings of the International Conference on Software Maintenance and Evolution*, pages 509–513, 2017.

[38] Gias Uddin and Foutse Khomh. Automatic mining of opinions expressed about apis in stack overflow. *IEEE Transactions on Software Engineering*, pages 1–1, 2019.

[39] Bowen Xu, Zhenchang Xing, Xin Xia, David Lo, and Xuan-Bach D. Le. XSearch: A domain-specific cross-language relevant question retrieval tool. In *Proceedings of the Joint Meeting on Foundations of Software Engineering*, page 1009–1013, 2017.

[40] Ziyu Yao, Daniel S. Weld, Wei-Peng Chen, and Huan Sun. StaQC: A systematically mined question-code dataset from stack overflow. In *Proceedings of the World Wide Web Conference*, page 1693–1703, 2018.

[41] Jian Zhang, Xu Wang, Hongyu Zhang, Hailong Sun, Kaixuan Wang, and Xudong Liu. A novel neural source code representation based on abstract syntax tree. In *Proceedings of the International Conference on Software Engineering*, page 783–794, 2019.

[42] Tianyi Zhang, Ganesha Upadhyaya, Anastasia Reinhardt, Hridesh Rajan, and Miryung Kim. Are code examples on an online Q&A forum reliable?: A study of API misuse on stack overflow. In *Proceedings of the International Conference on Software Engineering*, pages 886–896, 2018.

[43] Tianyi Zhang, Di Yang, Crista Lopes, and Miryung Kim. Analyzing and supporting adaptation of online code examples. In *Proceedings of the International Conference on Software Engineering*, page 316–327, 2019.

[44] Gang Zhao and Jeff Huang. DeepSim: Deep learning code functional similarity. In *Proceedings of the Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 141–151, 2018.

[45] Jing Zhou and Robert J. Walker. API deprecation: A retrospective analysis and detection method for code examples on the web. In *Proceedings of the International Symposium on Foundations of Software Engineering*, page 266–277, 2016.

[46] 湊真一. BDD/ZDD を用いたグラフ列挙索引化技法. **オペレーションズ・リサーチ**, 57(11):597–603, 2011.

発表論文一覧

Daiki Takata, Abdulaziz Alhefdhi, Maipradit Rungroj, Hideaki Hata, Hoa K. Dam, Takashi Ishio, and Kenichi Matsumoto. Catalogen: Generating catalogs of code examples collected from OSS. In *Proceedings of the International Workshop on Dynamic Software Documentation*, page 11–12, 2018.