

修 士 論 文

命令と呼び出しグラフによる 類似コード推薦法

令和 5 年度 修了

三重大学大学院 工学研究科

博士前期課程 情報工学専攻

コンピュータソフトウェア研究室

大野順也

概要

プログラミングにおいて、コードの再利用やライブラリの活用はその作業を効率化させる。しかし、ユーザがそれらの存在を知らずに、類似処理をするコードを記述してしまい、プログラミングが非効率になることがある。効率化のためには、ユーザが記述したコード片に対して類似処理をするコード片やライブラリを自動で推薦するシステムが必要である。

コード片やライブラリの推薦システムは主にコンテンツベースフィルタリングと協調フィルタリングの2種に分けられる。前者はコード片が持つ情報を用いるため、ソースコードの構造やその処理内容などを考慮して推薦できる。一方、後者はユーザが記述したコード片やソースコードレポジトリなどで使用される関数やライブラリの使用傾向により推薦されるため、類似処理のコード片が推薦されない可能性がある。

また、ユーザが記述したコード片と推薦候補のコード片の間には、字句や構文が完全に一致しないことが多い。しかし、コンパイラによって翻訳された低水準なバイトコードや機械語レベルではコード片が類似するため、ソースコードの字句や構文に差異があっても推薦できる。また、コード片内で呼び出される関数やメソッドが全く異なる処理をする場合、コード片全体の処理が異なり、推薦対象とならない。そのため、呼び出す関数やメソッドの情報も考慮しなければならない。

そこで本研究では、命令と呼び出し関係を用いたコンテンツベースの類似コードの推薦手法を提案する。提案手法で扱う命令と呼び出し関係は、それぞれ特徴ベクトルと呼び出しグラフで扱う。特徴ベクトルはバイトコードや機械語から得られる命令の出現回数をベクトルにしたもので、呼び出しグラフは関数・メソッドの間の呼び出し関係を表すグラフである。また、呼び出しグラフの頂点は関数やメソッド、辺は関数・メソッド間の呼び出し関係を表し、頂点と辺には関数・メソッドの情報を属性に持つ。特徴ベクトルと呼び出しグラフから類似度を測り、それらの閾値の範囲を満たしたものでランキングし、類似処理をするコード片を推薦する。類似度を測る指標として、特徴ベクトルはコサイン相違度、呼び出しグラフは正規化グラフ編集距離を用いた。また、正規化グラフ編集距離において、置換操作にかかるコストを呼び出しグラフの頂点と辺が持つ属性で計算し、重みでどの属性を重視するか調整できるようにした。

提案手法の実験と評価のために、提案手法の実装と実験データの作成を行った。また、実験データを用いて提案手法での適切な重みと閾値の範囲を求めた。推薦の精度を測るために推薦結果上位 k 位の精度を表す適合率@ k 、再現率@ k を用いて評価した。

目次

第 1 章	はじめに	3
1.1	研究背景	3
1.2	研究目的	3
1.3	論文の構成	4
第 2 章	関連研究	5
2.1	コード片の推薦	5
2.2	コードクローン検出	5
2.3	ライブラリ推薦	6
第 3 章	提案手法	7
3.1	推薦の流れ	7
3.2	抽出する特徴	8
3.2.1	命令	8
3.2.2	呼び出しグラフ	8
3.3	コサイン相違度	11
3.4	正規化グラフ編集距離	11
3.4.1	編集コスト	12
3.4.2	コサイン相違度と正規化グラフ編集距離の総合的な指標	13
第 4 章	実験・評価	14
4.1	実験準備	14
4.1.1	実験データ	14
4.1.2	実装	14
4.2	提案手法の評価	15
4.2.1	重みと閾値の範囲の選定	15
4.2.2	推薦結果上位 k 位の評価	18
4.3	考察	21
4.3.1	重みの組	21
4.3.2	閾値範囲	21
4.3.3	推薦結果の評価	21
第 5 章	まとめと今後の課題	22
5.1	まとめ	22
5.2	今後の課題	22

5.2.1	置換コストに用いる特徴	22
5.2.2	実装・実験	22
5.2.3	推薦にかかる時間	23
参考文献		25
付録 A	実験で用いたソースコード	26

第 1 章

はじめに

1.1 研究背景

プログラミングにおいて、コードの再利用やライブラリの活用はその作業を効率化させる。しかし、ユーザがそれらの存在を知らずに、類似処理をするコードを記述してしまい、プログラミングが非効率になることがある。同様に、大規模なソースコードレポジトリからユーザがクエリを作成し、ユーザが求める機能のコードやライブラリを探すことも非効率である。プログラミング効率化のためには、ユーザが記述したコード片に対し、類似処理をするコード片やライブラリなどを自動で推薦するシステムが必要である。

コード片やライブラリの推薦システムは主にコンテンツベースフィルタリングと協調フィルタリングの 2 種に分けられる。前者はコード片が持つ情報を用いるため、ソースコードの構造や処理内容などを考慮して推薦できる。対して、後者はユーザが記述したコード片やソースコードレポジトリなどで使用される関数やライブラリの使用傾向を用いて推薦するため、類似処理のコード片を推薦しない可能性がある。

また、ユーザが記述したコード片と推薦候補の類似処理するコード片の間には字句や構文レベルで差異があり、必ずしも一致しない。しかし、コンパイラによって変換された低水準なバイトコードや機械語レベルでは類似するため、字句や構文が異なるコード片でも推薦できる。また、コード片内で呼び出される関数やメソッドが全く異なる処理をする場合、コード片全体の処理が異なり、推薦対象とならない。そのため、呼び出す関数やメソッドの情報も考慮しなければならない。

1.2 研究目的

本研究では、ユーザが記述したコード片に対し、コードを蓄えているコーパス内の類似処理をするコードの推薦を目的とし、命令と呼び出し関係を用いたコンテンツベースの推薦手法を提案する。本研究で用いる命令はバイトコードや機械語から得られる命令とする。また、コード片の命令は類似しているが、呼び出している関数やメソッドでは命令が違い全く異なる処理をし、コード全体として処理が異なる場合がある。そのため、命令に加え、推薦において関数やメソッド間の呼び出し関係も考慮する。命令と呼び出し関係からそれぞれ特徴ベクトルと呼び出しグラフを生成する。特徴ベクトルは命令の出現回数で、呼び出しグラフは頂点が関数やメソッド、辺が関数・メソッド間の呼び出し関係を表す。これらから指標としてコサイン相違度と正規化グラフ編集距離を用いて、類似処理をするコード片を推薦する。

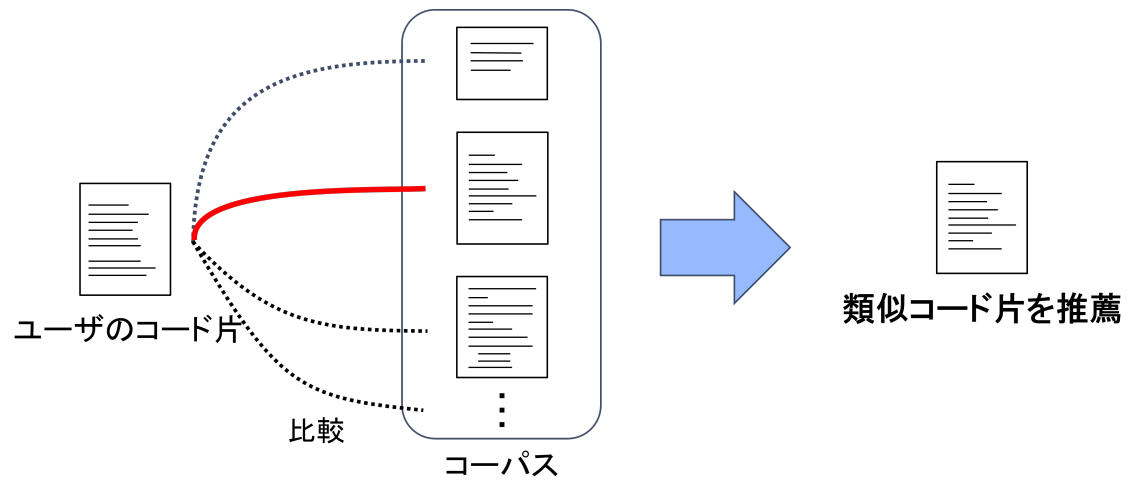


図 1.1 推薦のイメージ図

1.3 論文の構成

本論文の構成は次の通りである．第 2 章で，類似コード推薦の関連研究について述べ，第 3 章で，本研究の提案手法として，推薦の流れ，推薦に用いる命令と呼び出しグラフ，類似指標について述べる．第 4 章で，提案手法の実装と実験，考察について述べ，第 5 章で，まとめと今後の課題を述べる．

第 2 章

関連研究

コード片推薦やライブラリ推薦の研究では、プログラマが記述したコード片をクエリとして、コードを蓄えているコーパスやソースコードレポジトリからソースコードを取得し、クエリのソースコードと比較し、類似するものを推薦する。

2.1 コード片の推薦

Luan ら [7] はコード片の構造を用いたコード片推薦システム Aroma を提案している。始めにクエリとコーパスのコードを抽象構文木に変換する。各抽象構文木から構文木上の親子・兄弟関係や変数の相互の定義・使用関係をコード片の特徴として抽出する。その特徴の集合間で共通部分を求めることで、クエリに類似するコード片を推薦している。コード片の構造の特徴を用いているため、構造や構文は異なるが類似処理をするコード片の推薦を漏らす可能性がある。

Diamantopoulos ら [3] は、コード片の識別子名や型情報を用いたコード片の推薦システム QualBoa を提案している。クエリとレポジトリのソースコードを抽象構文木に変換し、抽象構文木からクラスやメソッドの識別子名や引数・返り値の型を抽出する。次に識別子名をキャメルケースに従い単語に分割して識別子名の集合とし、引数・返り値の型はそれぞれを要素とする集合を生成する。コード片間の各集合の共通部分の割合を求め、その割合から総合的な類似度をコード片間の類似度とし、類似度のソートによりランキングして結果を推薦する。しかし、識別子名はプログラマやプロジェクトによりその命名規則は異なり、類似処理するクラスやメソッド名に同じ単語を用いるとは限らない。そのため、識別子名の差異により、引数や返り値で類似していても推薦できないことがある。

2.2 コードクローン検出

類似コード推薦に関連する研究でコードクローン検出がある。コードクローンとは、ソースコード中に存在する互いに一致もしくは類似するコード片の対である。コードクローンはコード片の構造や処理内容などの一致に応じて 4 段階の種類に分けられていて [9]、その中で類似処理をするが構文が異なる対である第 4 種のクローンを検出することは、類似コード推薦において特に関連する。

Yu ら [10] はバイトコード上の命令列とメソッド呼び出しを用いたコードクローン検出法を提案している。ソースコードをバイトコードへ変換し、バイトコード上の命令とメソッド呼び出しそれぞれの類似度を求める。命令はコード片間で共通する命令の個数で類似度を算出し、メソッド呼び出しは呼び出される各メソッドの長さで類似度を算出する。これら 2 つの類似度を統合した類似度が閾値を満たすとき、対象の 2 片のコード片はコードクローンとして検出される。しかし、この検出法ではメソッド呼

び出しについて、メソッドの長さしか考慮されていない。呼び出すメソッドが全く異なる処理をする場合、第4種のクローンを検出できない可能性がある。

2.3 ライブラリ推薦

コード片のまとまりであるライブラリの推薦について述べる。ライブラリ推薦の研究には、既に使用しているライブラリから機能が同等または類似である他のライブラリへ移行を推薦するライブラリ移行推薦の研究がある。He ら [6] はソフトウェアの開発履歴から得られる4つのメトリクスを用いた協調フィルタリングの手法を提案している。メトリクスには、レポジトリから得られるバージョンやリビジョン、それに関連する特徴を用いている。あるライブラリから特定のライブラリへ移行（あるライブラリが削除され、特定のライブラリを追加）するときにおいて、その頻度、コミットメッセージなどでその移行の内容を示しているか、移行（削除と追加）のリビジョンの近さ、APIの移行頻度、これら4つを指標とする。これら指標を総乗した値でソートし、ランク付けされた移行先の候補一覧を推薦する。この手法は、他の使用傾向を利用する協調フィルタリングであるため、特定のライブラリへの移行が少ない場合、そのライブラリが類似処理をするにも関わらず、推薦できない可能性がある。

第 3 章

提案手法

本研究では、バイトコードの命令と関数・メソッドの呼び出し関係から得られる特徴を用いて、類似コードの推薦手法を提案する。本研究での対象の記述言語を Python とし、また、推薦するコード片の粒度は関数やメソッド単位とする。

3.1 推薦の流れ

まず、ユーザとコーパスのコード片からバイトコードと抽象構文木を生成する。バイトコードから各関数・メソッド毎に命令の出現回数を抽出し、関数・メソッドの特徴ベクトルとする。抽象構文木からは、関数・メソッド呼び出しを解析し、呼び出しグラフを生成する。コード片間の類似度を測るために、特徴ベクトルと呼び出しグラフの類似度指標としてコサイン相違度と正規化グラフ編集距離を用いる。コサイン相違度と正規化グラフ編集距離のそれぞれの閾値の範囲を満たすものでフィルタリングし、コサイン相違度と正規化グラフ編集距離の総合的な指標により、推薦対象に対して推薦するコード片をランキングして推薦する。推薦の概要を図 3.1 に示す。

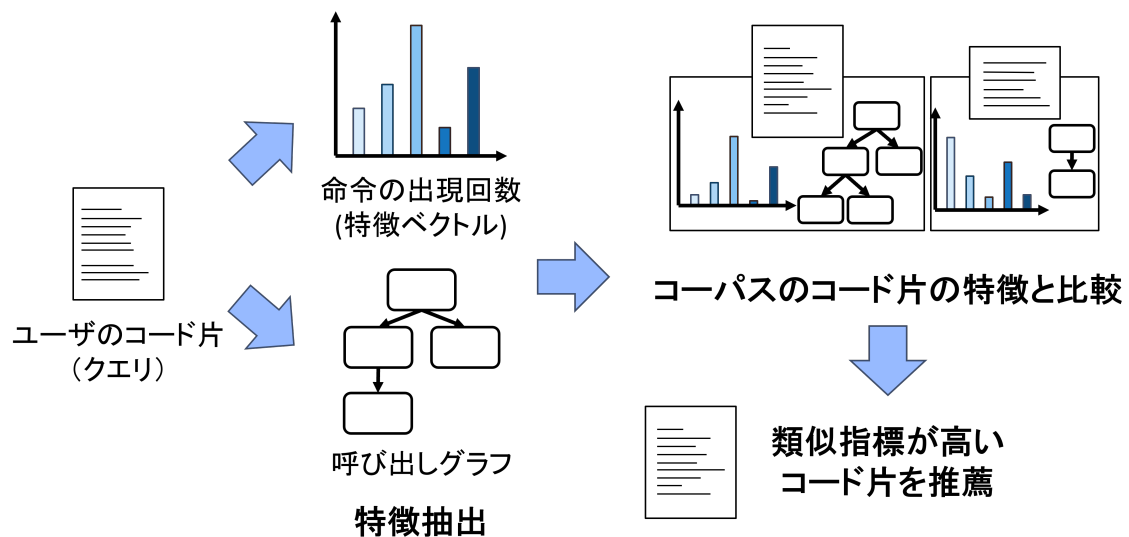


図 3.1 推薦の概要図

3.2 抽出する特徴

バイトコードの命令から関数・メソッド内での各命令の種類毎の出現回数を抽出し、抽象構文木から関数・メソッド間の呼び出し関係を表す呼び出しグラフを抽出する。3.2.1, 3.2.2 節で、2次元空間での L_1 距離を求める Python で記述したソースコード 3.1 を例に命令と呼び出しグラフについて説明する。

ソースコード 3.1 2次元空間での L_1 距離を求める Python プログラム

```
1 def myabs(a):
2     if a >= 0:
3         return a
4     else:
5         return -a
6
7 def l1distance(x1, y1, x2, y2):
8     return myabs(x1 - x2) + myabs(y1 - y2)
```

3.2.1 命令

関数・メソッド内の各命令の種類毎の出現回数を抽出する。ソースコード 3.1 から抽出した一部の命令の出現回数を表 3.1 に示す。命令の出現回数は、関数やメソッドの呼び出しがある場合、その呼び

表 3.1 ソースコード 3.1 の命令の出現回数

	RESUME	LOAD_GLOBAL	BINARY_OP	...	CALL	RETURN_VALUE
myabs	1	0	0	...	0	2
l1distance	3	2	3	...	2	5

出される関数・メソッド内の命令も関数・メソッド全体の出現回数として考慮する。そのため、関数 `l1distance` はソースコード上では `return` 文が 1 つしか無いため、命令の `RETURN_VALUE` の出現回数は 1 回となるが、関数 `myabs` を 2 回呼び出しているため、`RETURN_VALUE` は 5 回出現するとみなす。なお、本研究では、命令の種類は Python3.11 のバイトコードの仕様 [5] に従って分ける。

抽出した関数・メソッド内の命令の出現回数をその関数・メソッドの特徴ベクトルとする。ベクトルの各要素は出現回数を表すため 0 を含む自然数である。

3.2.2 呼び出しグラフ

本研究で扱う呼び出しグラフの説明をする。呼び出しグラフは、始点を持つ属性付き有向多重グラフであり、頂点が関数・メソッド、辺が呼び出し関係を表す。関数・メソッドを複数回呼び出している場合は多重辺、再帰呼び出しの場合は自己ループで表す。図 3.2 にソースコード 3.1 の呼び出しグラフを示す。頂点のラベルは元のソースコードでの関数定義の開始行番号と関数名を表し、後述する組み込み関数や演算関数は 0 行目とする。なお、属性についても後述するが、多数あるため図では省略する。また、ソースコード 3.2 のようにモジュールをインポートしその関数を呼び出している場合、呼び出しグラフではモジュールを展開してグラフを構成する。その例を図 3.3 に示す。

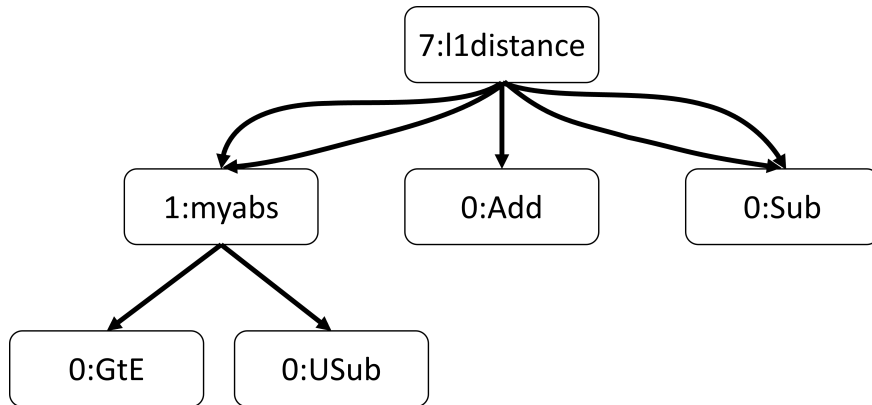


図 3.2 ソースコード 3.1 の呼び出しグラフ

ソースコード 3.2 2次元空間での L_2 距離を求める Python プログラム

```

1 import mymath
2
3 def l2distance(x1, y1, x2, y2):
4     return mymath.mysqrt((x1 - x2)**2 + (y1 - y2)**2)

```

ソースコード 3.3 数学関数をまとめたモジュール mymath.py の一部

```

1 def mysqrt(a):
2     return a**0.5

```

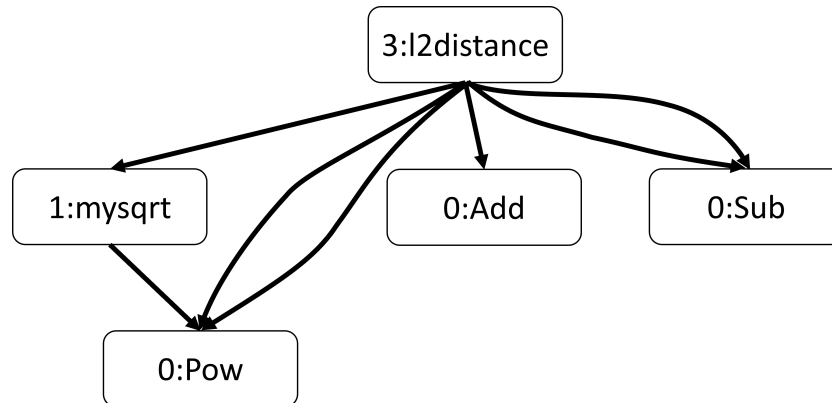


図 3.3 モジュール内の関数呼び出しの展開を考慮した呼び出しグラフ

本手法ではコード片の関数・メソッド単位でグラフを構成するため、始点は推薦の比較対象の関数・メソッドとなる。

頂点の属性は、引数の数、return 文の数、返り値の種類、関数・メソッドの種類を持つ。返り値の種類とは return 文で返されるデータ型が単数か複数のどちらであるかの 2 種で、Python の組み込みのデータ型である list, tuple, range, str, bytes, bytearray, set, frozenset, dict を複数とし、複数以外のデータ型を単数とする。関数・メソッドの種類は関数、メソッド、組み込み関数、演算関数に分ける。演算関数は、Python の構文の算術演算や比較演算が記述されていたら特別にその演算の関数を呼び出したと扱う。これは、コード片が短く得られる情報が少ない場合があるため、演算を関数呼び出しとして考慮する。図 3.2, 3.3 での Add や GtE がその例である。

辺の属性は、呼び出し元の文脈と位置についての3つの情報を持つ。1つ目は、呼び出しがどの構文中で呼び出されるかの情報で、ソースコード 3.1 の8行目の関数 `myabs` の呼び出しは、算術演算内で呼ばれているので、その情報を持つ。言い換えると、抽象構文木上の関数・メソッド呼び出しを表す頂点の親を情報に持つ。2つ目は、何番目の引数で呼ばれたのかである。別の関数・メソッドの引数で呼び出しがある場合、何番目の引数で呼ばれるかを考慮する。ただし、関数・メソッドの引数で呼ばれない場合は、0番目の引数で呼ばれるとして扱う。3つ目は、呼び出し元の関数・メソッド内における呼び出し位置の割合である。ソースコード 3.4 を例に説明する。関数・メソッド内の有効な行のうち、どこで呼び出しがあるのかを考える。有効な行とは、コメント文のみの行と関数・メソッドの識別子名が定義される行以外の行とする。この場合、`[3, 4, 5, 6, 7, 8, 9, 11, 12, 13, 14, 15]` が有効な行となる。9行目の関数呼び出しは通算7番目の行となるが、3行目の `if` 文の条件で偽のとき、実際は5番目に実行され、順序に齟齬が出る。そのため、制御文の `if` 文や `try` 文などでは、分岐するパスの数だけ有効な行の列を考え、この例においては `if` 文の分岐により、`[3, 4, 5, 11, ...]`、`[3, 6, 7, 8, 9, 11, ...]` と2つの実行パスの有効な行を考慮する。また、12行目の呼び出しについて、前述のような3行目から9行目にかけての `if` 文の分岐と、12行目から14行目の `while` 文による分岐の4パターンの実行パスが存在する。12行目や16行目の呼び出しのように実行パスの分岐前や収束後に呼ばれるものに対しては、分岐したパスの中で有効な行数が多いパスを採用する。つまり、`[3, 6, 7, 8, 9, 11, 12, 13, 14, 15, 16, 17]` のパスを採用する。そのため、`for` 文や `while` 文などの反復の制御文については、イテレータや条件の真偽に関わらず、各構文の複文は必ず1回だけ実行されループしないものとして扱い、構文の複文が1回も実行されないときのパスは常に考慮しない。故に、この例で考えられる実行パスは `[3, 4, 5, 11, 12, 13, 14, 15, 16, 17]`、`[3, 6, 7, 8, 9, 11, 12, 13, 14, 15, 16, 17]` の2つのみである。5行目の関数呼び出しについての割合は、実行パスは前者であり、5行目は3番目であるため、 $3/10$ となる。同様に、9行目の関数呼び出しは後者のパスより $5/12$ となる。12行目、15行目、16行目の呼び出しは、有効な行が多い後者のパスよりそれぞれ $7/12$ 、 $10/12$ 、 $11/12$ となる。

ソースコード 3.4 呼び出し位置の割合の例

```

1 def func(a):
2     """comment"""
3     if a < 0:
4         b = -1
5         c = func2(b)
6     else:
7         a += 3
8         b = a * 2
9         c = func2(b)
10    #comment
11    d = 0
12    func3(c)
13    while c > 0:
14        c -= 1
15        d += func4(c)
16    e = func5(d)
17    return e

```

3.3 コサイン相違度

特徴ベクトルの類似度指標にコサイン相違度を定義して用いる。コサイン相違度は、1 からコサイン類似度を引いたものである。2 つの特徴ベクトルのなす角を表すコサイン類似度を求め、コサイン相違度を算出する。特徴ベクトルの各要素は 0 を含む自然数であり、コサイン類似度は $[0,1]$ の実数を取るため、コサイン相違度も $[0,1]$ の実数の範囲で 0 に近づくほど 2 つのベクトルは類似し、1 に近づくほど相違する。

$$\text{cosdiff}(\text{vec}_1, \text{vec}_2) = 1 - \cos(\text{vec}_1, \text{vec}_2) \quad (3.1)$$

3.4 正規化グラフ編集距離

グラフ編集距離 [8] とは、あるグラフから別のグラフへ変換するとき、頂点と辺に対して、編集操作 (追加, 削除, 置換) に要するコストの総和の最小値である。なお、本手法ではコード片を対象としているため、グラフの始点は必ず別のグラフの始点に置換されると制約する。各編集操作にかかるコストは 3.4.1 節で説明する。グラフ G_1, G_2 間のグラフ編集距離は以下の式 (3.2) で定義できる。 $\Upsilon(G_1, G_2)$ はグラフ G_1 から G_2 への変換に伴う全ての編集パスの集合、 $c(e_i)$ は編集 e_i にかかるコスト、 e_i は頂点と辺を編集 (追加, 削除, 置換) する操作、 $\text{dist}(G_1, G_2)$ は $\Upsilon(G_1, G_2)$ の中でグラフ G_1 から G_2 への編集にかかる最小の総コストを表す。

$$\text{dist}(G_1, G_2) = \min_{\lambda \in \Upsilon(G_1, G_2)} \sum_{e_i \in \lambda} c(e_i) \quad (3.2)$$

図 3.4 に呼び出しグラフ G_1 から G_2 へ変換に伴う編集の一例を挙げる。ここで、頂点集合を V 、辺集合を $E \subseteq V \times V$ とし、各頂点を $v_i \in V$ 、各辺を $(v_i, v_j) \in E$ で表す。また、各編集操作の表記について、頂点と辺の追加はそれぞれ $(\varepsilon \rightarrow v)$, $(\varepsilon \rightarrow (v_i, v_j))$ 、削除はそれぞれ $(v \rightarrow \varepsilon)$, $((v_i, v_j) \rightarrow \varepsilon)$ 、置換はそれぞれ $(v_i \rightarrow v'_i)$, $((v_i, v_j) \rightarrow (v'_i, v'_j))$ と表し、 ε は空を表す。図 3.4 の G_2 での赤い実線の頂点と辺は追加操作、緑色の破線の頂点と辺は削除操作、 G_1 の頂点から G_2 の頂点へ伸びている青色の点線は置換操作が行われることを示す。この例での編集パスは $\{(u_1 \rightarrow v_1), (u_2 \rightarrow v_2), (u_3 \rightarrow v_3), (u_4 \rightarrow v_5), (u_5 \rightarrow \varepsilon), (\varepsilon \rightarrow v_4), (\varepsilon \rightarrow (v_2, v_4)), ((u_4, u_5) \rightarrow \varepsilon)\}$ となる。

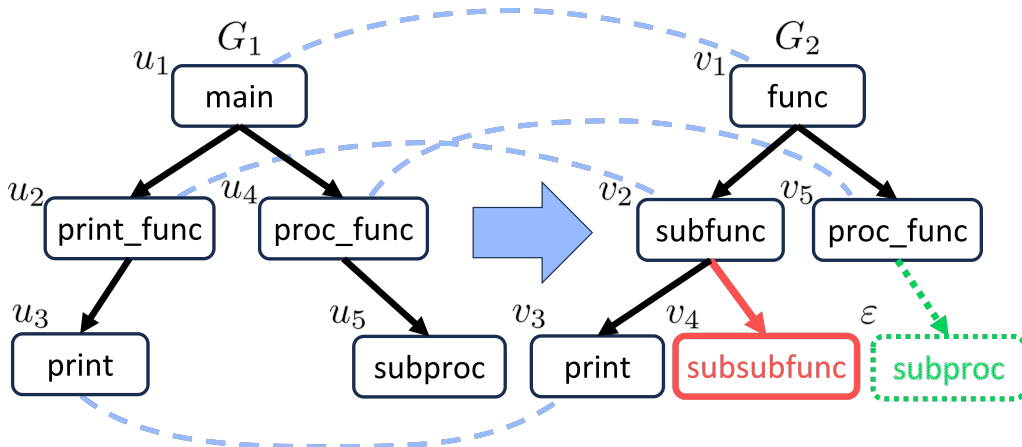


図 3.4 グラフ編集距離における編集の一例

本研究では、グラフ編集距離を $[0, 1]$ に抑えるために、 x 軸方向に b だけ平行移動させたシグモイド関数をかけて正規化する。平行移動させているのは、グラフ編集距離が 0 のとき、限りなく 0 に近い値に写すためである。以下、正規化したグラフ編集距離を正規化グラフ編集距離と呼ぶ。

$$\text{dist}'(G_1, G_2) = \varsigma_a(x) \times \text{dist}(G_1, G_2) = \frac{1}{1 + e^{-a(x-b)}} \times \text{dist}(G_1, G_2) \quad (3.3)$$

3.4.1 編集コスト

編集操作にかかるコストは、挿入と削除は 1、置換は $[0, 1]$ の範囲で頂点は (3.4) 式、辺は (3.7) 式によって決める。

頂点の置換コストは比較する関数・メソッド f_1, f_2 が同じ種類の関数・メソッドであるかで同じ種類同士なら式 (3.5) の $\text{cost}_{\text{func}}(f_1, f_2)$ 、異なるなら 1 となる。なお、関数とメソッドの比較は類似コードとして推薦できる可能性があるため、同じ種類同士の比較と同様に扱う。式 (3.5) の $\text{cost}_{\text{func}}(f_1, f_2)$ は、それぞれ引数、return 文、返り値の種類のコストを重み α, β, γ で加重平均を取っている。各コスト $\text{cost}_{\text{arg}}, \text{cost}_{\text{ret}}, \text{cost}_{\text{retseq}}$ はそれぞれ引数の個数の相違度、return 文の数の相違度、返り値の種類の不一致度である。 $\text{cost}_{\text{retseq}}$ は、比較する関数・メソッド内の return の全ての組み合わせについて、一致するかどうか計算する。3つの指標は $[0, 1]$ の範囲を取り、0 に近づくほど類似し、1 に近づくほど相違する。 $n_{\text{arg}}(f), n_{\text{ret}}(f)$ をそれぞれ関数・メソッド f の引数の個数、return 文の数とし、 $\text{type}_{\text{func}}(f)$ は関数・メソッド f の種類とする。列 $A^f = a_1^f, a_2^f, \dots, a_i^f, \dots, a_{n_{\text{ret}}(f)}^f$ は関数・メソッド f に存在する各 return 文が単数か複数のどちらを返すかの 2 値を順に並べた列である。

また、辺の置換コストも式 (3.7) により、呼び出しがどの構文中で呼ばれるか、何番目の引数で呼ばれるか、呼び出し元における呼び出し位置の割合に対して、重み δ, ϵ, ζ による加重平均で求める。 e_{f_1, f_3} は関数・メソッド f_1 内で f_3 を呼び出していることを表す。 $\text{kind}_{\text{caller}}(e_{f_1, f_3})$ は f_3 をどの構文内で呼んだかを表す。 $\text{num}_{\text{arg}}(e_{f_1, f_3})$ は f_3 をある関数・メソッド呼び出しの引数内で呼び出した場合、何番目の引数で呼んだかを表し、関数・メソッド呼び出し以外で f_3 が呼ばれていた場合、0 とする。 $\text{ratio}_{\text{lines}}(e_{f_1, f_3})$ は、呼び出し元 f_1 の有効な行に対して、 f_3 の呼び出しの位置の割合を表す。

$$\text{cost}_{\text{node}}(f_1, f_2) = \begin{cases} \text{cost}_{\text{func}}(f_1, f_2) & (\text{kind}_{\text{func}}(f_1), \text{kind}_{\text{func}}(f_2) \in \{\text{function}, \text{method}\} \\ & \forall \text{kind}_{\text{func}}(f_1) = \text{kind}_{\text{func}}(f_2)) \\ 1 & \text{otherwise} \end{cases} \quad (3.4)$$

$$\text{cost}_{\text{func}}(f_1, f_2) = \alpha \times \text{cost}_{\text{arg}}(f_1, f_2) + \beta \times \text{cost}_{\text{ret}}(f_1, f_2) + \gamma \times \text{cost}_{\text{retseq}}(f_1, f_2) \quad (3.5)$$

where:

$$\begin{cases} \alpha + \beta + \gamma = 1 & (0 \leq \alpha, \beta, \gamma \leq 1) \\ \text{cost}_{\text{arg}}(f_1, f_2) = \begin{cases} 0 & (n_{\text{arg}}(f_1) = 0 \wedge n_{\text{arg}}(f_2) = 0) \\ \frac{|n_{\text{arg}}(f_1) - n_{\text{arg}}(f_2)|}{\max(n_{\text{arg}}(f_1), n_{\text{arg}}(f_2))} & \text{otherwise} \end{cases} \\ \text{cost}_{\text{ret}}(f_1, f_2) = \begin{cases} 0 & (n_{\text{ret}}(f_1) = 0 \wedge n_{\text{ret}}(f_2) = 0) \\ \frac{|n_{\text{ret}}(f_1) - n_{\text{ret}}(f_2)|}{\max(n_{\text{ret}}(f_1), n_{\text{ret}}(f_2))} & \text{otherwise} \end{cases} \\ \text{cost}_{\text{retseq}}(f_1, f_2) = \begin{cases} 0 & (n_{\text{ret}}(f_1) = 0 \wedge n_{\text{ret}}(f_2) = 0) \\ 1 - \frac{\sum_{i=1}^{n_{\text{ret}}(f_1)} \sum_{j=1}^{n_{\text{ret}}(f_2)} eq(a_i^{f_1}, a_j^{f_2})}{n_{\text{ret}}(f_1) \cdot n_{\text{ret}}(f_2)} & (n_{\text{ret}}(f_1) \neq 0 \wedge n_{\text{ret}}(f_2) \neq 0) \\ 1 & \text{otherwise} \end{cases} \\ eq(a_i^{f_1}, a_j^{f_2}) = \begin{cases} 1 & (a_i^{f_1} = a_j^{f_2}) \\ 0 & \text{otherwise} \end{cases} \end{cases} \quad (3.6)$$

$$\begin{aligned} \text{cost}_{\text{edge}}(e_{f_1, f_3}, e_{f_2, f_4}) &= \delta \times \text{cost}_{\text{caller}_{\text{kind}}}(e_{f_1, f_3}, e_{f_2, f_4}) \\ &\quad + \epsilon \times \text{cost}_{\text{num}_{\text{arg}}}(e_{f_1, f_3}, e_{f_2, f_4}) \\ &\quad + \zeta \times \text{cost}_{\text{line}}(e_{f_1, f_3}, e_{f_2, f_4}) \end{aligned} \quad (3.7)$$

where:

$$\begin{cases} \delta + \epsilon + \zeta = 1 & (0 \leq \delta, \epsilon, \zeta \leq 1) \\ \text{cost}_{\text{caller}_{\text{kind}}}(e_{f_1, f_3}, e_{f_2, f_4}) = \begin{cases} 0 & (\text{kind}_{\text{caller}}(e_{f_1, f_3}) = \text{kind}_{\text{caller}}(e_{f_2, f_4})) \\ 1 & \text{otherwise} \end{cases} \\ \text{cost}_{\text{num}_{\text{arg}}}(e_{f_1, f_3}, e_{f_2, f_4}) = \begin{cases} 0 & (\text{num}_{\text{arg}}(e_{f_1, f_3}) = \text{num}_{\text{arg}}(e_{f_2, f_4})) \\ 1 & \text{otherwise} \end{cases} \\ \text{cost}_{\text{line}}(e_{f_1, f_3}, e_{f_2, f_4}) = |\text{raito}_{\text{lines}}(e_{f_1, f_3}) - \text{raito}_{\text{lines}}(e_{f_2, f_4})| \end{cases} \quad (3.8)$$

3.4.2 コサイン相違度と正規化グラフ編集距離の総合的な指標

推薦対象のコード片に対して、コーパスの推薦するコード片をランキングするために、式 (3.9) のコサイン相違度と正規化グラフ編集距離の総合的な相違指標を用いる。式 (3.9) は論理和を $[0, 1]$ の連続値で扱えるよう拡張した式であり、 cosdiff , dist' はともに $[0, 1]$ の範囲で、0 に近づくほど類似し、1 に近づくほど相違する相違度であるので、 totalindex は $[0, 1]$ の範囲であり、 cosdiff , dist' のどちらかでも 1 に近ければ、 totalindex も 1 に近づく、つまり総合的に相違を示す。逆に cosdiff と dist' とともに 0 に近ければ、 totalindex も 0 に近づき、総合的に類似を示す。

$$\text{totalindex}(\text{vec}_1, \text{vec}_2, G_1, G_2) = 1 - (1 - \text{cosdif}(\text{vec}_1, \text{vec}_2))(1 - \text{dist}'(G_1, G_2)) \quad (3.9)$$

第 4 章

実験・評価

提案手法の評価について、この章では実験準備、評価指標の説明、適切な重みと閾値の範囲の決め方、実験結果、考察について述べる。

4.1 実験準備

4.1.1 実験データ

自作ソースコードの 2 次元空間での L_1 距離と L_2 距離の算出、商品データを扱う小規模のクラス、自作の数学関数など 15 個のプログラム、35 個の関数・メソッド (付録 A を参照) を対象に実験を行った。

また、これら関数・メソッドに対して、全 595 対のペアを作り、このペアの間で推薦を行うとき、推薦をしてユーザが利用できるか、できないかをそれぞれ True, False の 2 つのラベル付けしたベンチマークを作成した。True の判断基準として、推薦による置き換え前後で関数・メソッドの入力である引数と出力が変わらないもの、入出力が異なりそのまま置き換えできないが一部の処理が類似しているものを対象とした。後者の例として、ソースコード 4.1, 4.2 を挙げる。これら 2 関数は引数が異なるが、関数 `mypow` の引数 `n` に 0.5 を渡すことで 2 関数は同じものとしてみなせる。

ソースコード 4.1 平方根を返す関数

```
1 def mysqrt(a):  
2     return a**0.5
```

ソースコード 4.2 累乗を返す関数

```
1 def mypow(a, n):  
2     return a ** n
```

4.1.2 実装

与えられる 2 つの Python ソースコードから Python バイトコードと抽象構文木へ変換し、それぞれ命令の特徴ベクトルと呼び出しグラフを抽出し、コサイン相違度と正規化グラフ編集距離を求めるよう実装した。グラフ編集距離を求める処理については、Python のグラフやネットワークを扱うライブラリの NetworkX[2] の関数 `graph_edit_distance` を用いた。なお、NetworkX のグラフ編集距離の算出処理は、与えるグラフの構造によりその時間が変動するため、2 秒で処理を打ち切るようにして、打ち切ったときの最小のグラフ編集距離をデータとして扱った。なお、式 (3.3) の正規化グラフ編集距離に使うシグモイド関数の a は 0.5, b は 10 とした。 $a = 0.5$, $b = 10$ とした理由として、全ての編集操作

にかかるコストを一律に 1 とした場合、実験データを対象にグラフ編集距離を算出するとグラフ編集距離は $[0, 21]$ の整数値を取るためである。 a についてはグラフ編集距離の 0 と 21 をそれぞれ 0 と 1 に限りなく近い値へ写し、シグモイド関数の曲線がなめらかで正規化後に偏りが生じない 0.5 を選んだ。

4.2 提案手法の評価

提案手法の式 (3.4), (3.7) の重み $\alpha, \beta, \gamma, \delta, \epsilon, \zeta$ での適切な値の組とコサイン相違度・正規化グラフ編集距離の閾値範囲を求め、推薦結果の精度の評価を行った。

4.2.1 重みと閾値の範囲の選定

類似コード推薦に適切な重みの組とコサイン相違度・正規化グラフ編集距離の閾値の範囲を選ぶために F 値を用いた。F 値は二値分類の評価指標に広く使われ、同様の評価指標である適合率、再現率の調和平均によって求められる。適合率は正確性の指標で、再現率は網羅性の指標である。適合率、再現率、F 値は、二値分類の結果をまとめた表 4.1 の混同行列の TP, FP, FN, TN により、式 (4.1) から (4.3) に従って求める。

表 4.1 混同行列

		本手法の推薦結果	
		利用可 (Positive)	利用不可 (Negative)
ベンチマーク	利用可 (True)	TP(True Positive) 利用可と推薦したものの内 実に利用可である数	FN(False Negative) 利用不可として推薦しなかったものの内 実は利用可であった数
	利用不可 (False)	FP(False Positive) 利用可と推薦したものの内 実は利用不可であった数	TN(True Negative) 利用不可として推薦しなかったものの内 実に利用不可である数

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \quad (4.1)$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \quad (4.2)$$

$$\text{F-measure} = \frac{2}{\text{Precision}^{-1} + \text{Recall}^{-1}} \quad (4.3)$$

重みの組と閾値範囲は、以下の局所探索法に沿って求める。探索終了時の最大 F 値の重みの組と閾値範囲が解となる。

- (1) 重みの組の集合からランダムに重みの組を選ぶ
- (2) その重みの組での最大となる F 値を求め、この重みの組を最良の重みの組とする
- (3) 最良の重みの組の近傍から一つ重みの組を選ぶ
- (4) 同様に近傍の重みの組で最大となる F 値を求める
- (5) (a) 近傍の重みの組での最大 F 値が最良の重みの組での最大 F 値を越えていたら、近傍の重みの組を最良の重みの組とし、3. へ戻る

(b) 越えていない場合、残りの近傍があれば、一つ近傍の重みの組を選び 4. へ戻る

(c) 残りの近傍がなければ探索を終了する

(1) の重みの組の集合とは、各重みを式 (3.4), (3.7) の $\alpha + \beta + \gamma = 1$ ($0 \leq \alpha, \beta, \gamma \leq 1$) と $\delta + \epsilon + \zeta = 1$ ($0 \leq \delta, \epsilon, \zeta \leq 1$) を満たしながら $1/12$ 刻みで変えた $8281 (= 91^2 = (\sum_{i=1}^{13} i)^2)$ 通りの組からなる。(2) や (4) での最大の F 値の求め方は、その重みの組でのコサイン相違度と正規化グラフ編集距離の各閾値の範囲を変化させ、推薦結果を変えることで、最大となる F 値を求めている。例として、図 4.1 の各重みが全て $1/3$ のときの散布図において、コサイン相違度、正規化グラフ編集距離ともに 1 に近づくほど False が多いため、閾値の範囲を 0 に近づけて狭めるほど F 値が高まることから、コサイン相違度の閾値の範囲が $[0.00, 0.15]$ 、正規化グラフ編集距離の閾値の範囲が $[0.00, 0.27]$ のとき、F 値が 0.58 で最大となる。

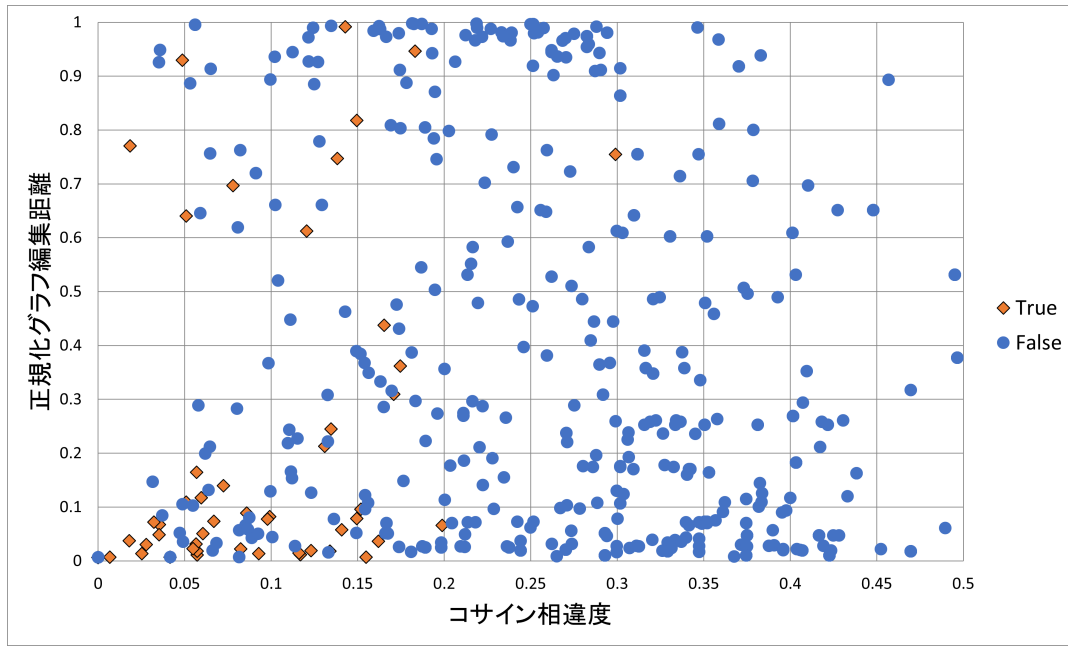


図 4.1 各重みが全て $1/3$ のときのコサイン相違度と正規化グラフ編集距離の散布図

また、近傍とは、選んだ重みの組で式 (3.4), (3.7) を満たしながら、頂点の 2 つの重みと辺の 2 つの重みを $\pm 1/12$ した組と、頂点か辺のどちらかの 2 つの重みを $\pm 1/12$ した組の集合とする。例えば $\alpha = \beta = \gamma = \delta = \epsilon = \zeta = 1/3$ の近傍は、 $\{(5/12, 1/4, 1/3, 1/3, 1/3, 1/3), (5/12, 1/4, 1/3, 5/12, 1/4, 1/3), \dots\}$ の 48 組の集合となる。なお、各組は順に $\alpha, \beta, \gamma, \delta, \epsilon, \zeta$ の値を表す。

この局所探索を 250 回行い、F 値の降順で上位 10 位までの重みの組と閾値範囲の結果を表 4.2 に示す。なお、複数の組が同じ F 値を取る場合があったため、結果は一部抜粋して示す。また、F 値が 0.6 となるもので適合率、再現率がそれぞれ 0.5, 0.75 と 0.49, 0.75 のものなどがあるが、有効数字による四捨五入によるものである。

表 4.2 から F 値が 0.6 となる重みの組 $(5/12, 0, 7/12, 1, 0, 0)$, $(0, 1/4, 3/4, 7/12, 1/3, 1/12)$, $(1/6, 1/3, 1/2, 3/4, 1/4, 0)$, $(1/12, 1/3, 7/12, 2/3, 1/4, 1/12)$, $(2/3, 1/12, 1/4, 5/12, 0, 7/12)$, $(1/2, 1/6, 1/3, 1, 0, 0)$ とその組での閾値範囲を推薦結果の精度の評価実験に使う。また、各 250 回の局所探索において、近傍を最良の重みの組にする際に、元の最良の重みの組からどの重みを $\pm 1/12$ したかをカウントした結果を表 4.3 に示す。

表 4.2 F 値の上位 10 位での重みの組と閾値範囲

$\alpha, \beta, \gamma, \delta, \epsilon, \zeta$	コサイン相違度 閾値範囲	正規化グラフ編集距離 閾値範囲	適合率	再現率	F 値
5/12,0,7/12,1,0,0	0.0, 0.17	0.0067, 0.29	0.5	0.75	0.6
0,1/4,3/4,7/12,1/3,1/12	0.0, 0.17	0.0067, 0.28	0.49	0.77	0.6
1/6,1/3,1/2,3/4,1/4,0	0.0, 0.17	0.0067, 0.29			
1/12,1/3,7/12,2/3,1/4,1/12	0.0, 0.17	0.0067, 0.29			
2/3,1/12,1/4,5/12,0,7/12	0.0, 0.16	0.0067, 0.15	0.54	0.66	0.6
1/2,1/6,1/3,1,0,0	0.0, 0.17	0.0067, 0.30	0.49	0.75	0.6
1/6,3/4,1/12,2/3,1/3,0	0.0, 0.16	0.0067, 0.28	0.5	0.73	0.59
0,1/2,1/2,1/2,5/12,1/12	0.0, 0.16	0.0067, 0.27			
0,5/6,1/6,7/12,5/12,0	0.0, 0.16	0.0067, 0.27			
1/3,1/3,1/3,1,0,0	0.0, 0.16	0.0067, 0.29	0.51	0.71	0.59
1/12,5/12,1/2,1,0,0	0.0, 0.16	0.0067, 0.27			
1/2,5/12,1/12,1,0,0	0.0, 0.16	0.0067, 0.3			
5/12,1/4,1/3,1/2,1/12,5/12	0.0, 0.16	0.0067, 0.15	0.54	0.66	0.59
1/2,1/2,0,1/2,1/6,1/3	0.0, 0.16	0.0067, 0.15			
7/12,1/3,1/12,1/2,1/6,1/3	0.0, 0.16	0.0067, 0.16			
7/12,0,5/12,1/2,1/12,5/12	0.0, 0.16	0.0067, 0.16			
3/4,1/12,1/6,5/12,1/6,5/12	0.0, 0.16	0.0067, 0.15			
1/2,0,1/2,1/2,1/12,5/12	0.0, 0.16	0.0067, 0.16			
1/2,1/2,0,1/2,1/12,5/12	0.0, 0.16	0.0067, 0.16			
7/12,1/12,1/3,1/2,1/6,1/3	0.0, 0.16	0.0067, 0.16			
7/12,1/3,1/12,1/2,1/4,1/4	0.0, 0.16	0.0067, 0.15			
1/3,1/12,7/12,5/6,0,1/6	0.0, 0.17	0.0067, 0.29	0.49	0.75	0.59
1/6,0,5/6,5/6,0,1/6	0.0, 0.17	0.0067, 0.28			
1/6,5/6,0,5/6,0,1/6	0.0, 0.17	0.0067, 0.32			
0,1,0,5/6,0,1/6	0.0, 0.17	0.0067, 0.32			
1/12,1/4,2/3,5/6,1/6,0	0.0, 0.17	0.0067, 0.29			
1/3,1/4,5/12,5/6,1/6,0	0.0, 0.17	0.0067, 0.3			
1/12,1/12,5/6,2/3,0,1/3	0.0, 0.17	0.0067, 0.28			
1/3,1/3,1/3,5/6,0,1/6	0.0, 0.17	0.0067, 0.3			
0,1/3,2/3,3/4,0,1/4	0.0, 0.17	0.0067, 0.29			
5/12,7/12,0,2/3,1/4,1/12	0.0, 0.15	0.0067, 0.16	0.55	0.64	0.59
1/3,1/3,1/3,2/3,0,1/3	0.0, 0.15	0.0067, 0.16			

表 4.3 近傍探索時の最良の重みを $\pm 1/12$ した回数

	α	β	γ	δ	ϵ	ζ
+1/12 した回数	95	130	118	251	65	55
-1/12 した回数	122	118	103	66	141	164

4.2.2 推薦結果上位 k 位の評価

重みと閾値範囲を決めるために、推薦結果全体での評価指標である適合率と再現率を用いたが、推薦対象に対して正確に推薦できるかを評価する指標として、推薦結果上位 k 位に限定した適合率@ k と再現率@ k がある。適合率@ k と再現率@ k は式 (4.5), (4.7) で定義され、それぞれ個々の推薦対象の評価指標の平均を取る。 $R_i(k)$ は推薦対象 i に対して推薦結果の上位 k 位の集合、 G は推薦対象に対して推薦コード片が True とラベル付けしたものの集合、 $TP_i@k$ は推薦対象 i に対して推薦結果の上位 k 位の内、 True とラベル付けされている数、 FN_i は推薦対象 i に対して推薦結果の k 位未満で推薦しなかったものの内、 True とラベル付けされていた数、 I は推薦対象の集合 (実験ではソースコードの全ての関数・メソッド) を表す。

適合率@ k

$$\text{Precision}_i@k = \frac{|R_i(k) \cap G|}{|R_i(k)|} = \frac{TP_i@k}{k} \quad (4.4)$$

$$\text{Precision}@k = \frac{1}{|I|} \sum_{i \in I} \text{Precision}_i@k \quad (4.5)$$

再現率@ k

$$\text{Recall}_i@k = \frac{|R_i(k) \cap G|}{|G|} = \frac{TP_i@k}{TP_i@k + FN_i} \quad (4.6)$$

$$\text{Recall}@k = \frac{1}{|I|} \sum_{i \in I} \text{Recall}_i@k \quad (4.7)$$

4.2.1 節で決めた 6 つの重みの組と閾値範囲を基に、上位 1 位から 5 位までの適合率@ k と再現率@ k を算出する。また、比較として、正規化グラフ編集距離において、頂点と辺の置換コストが常に 0 もしくは 1 を返す 2 つのパターンも加えて実験する。なお、これらのパターンでの各閾値範囲は 4.2.1 節の近傍探索の最大の F 値を求め方と同様に求め、常に 0 を返すときはコサイン相違度が $[0.0, 0.15]$ 、正規化グラフ編集距離が $[0.0067, 0.076]$ 、最大となる F 値は 0.55、常に 1 を返すときはコサイン相違度が $[0.0, 0.17]$ 、正規化グラフ編集距離が $[0.0067, 0.38]$ 、最大となる F 値は 0.54 となった。実験結果を表 4.4 から表 4.8 に示す。

表 4.4 適合率@1 と再現率@1

重みの組 $\alpha, \beta, \gamma, \delta, \epsilon, \zeta$ または置換コスト	コサイン相違度 閾値範囲	正規化グラフ編集距離 閾値範囲	適合率@1	再現率@1
5/12,0,7/12,1,0,0	0.0, 0.17	0.0067, 0.29	0.47	0.18
0,1/4,3/4,7/12,1/3,1/12	0.0, 0.17	0.0067, 0.28	0.38	0.12
1/6,1/3,1/2,3/4,1/4,0	0.0, 0.17	0.0067, 0.29	0.38	0.12
1/12,1/3,7/12,2/3,1/4,1/12	0.0, 0.17	0.0067, 0.29	0.38	0.12
2/3,1/12,1/4,5/12,0,7/12	0.0, 0.16	0.0067, 0.15	0.41	0.15
1/2,1/6,1/3,1,0,0	0.0, 0.17	0.0067, 0.3	0.47	0.18
置換コストが常に 0	0.0, 0.15	0.0067, 0.076	0.26	0.04
置換コストが常に 1	0.0, 0.17	0.0067, 0.38	0.26	0.04

表 4.5 適合率@2 と再現率@2

重みの組 $\alpha, \beta, \gamma, \delta, \epsilon, \zeta$ または置換コスト	コサイン相違度 閾値範囲	正規化グラフ編集距離 閾値範囲	適合率@2	再現率@2
5/12,0,7/12,1,0,0	0.0, 0.17	0.0067, 0.29	0.37	0.24
0,1/4,3/4,7/12,1/3,1/12	0.0, 0.17	0.0067, 0.28	0.32	0.16
1/6,1/3,1/2,3/4,1/4,0	0.0, 0.17	0.0067, 0.29	0.31	0.16
1/12,1/3,7/12,2/3,1/4,1/12	0.0, 0.17	0.0067, 0.29	0.32	0.16
2/3,1/12,1/4,5/12,0,7/12	0.0, 0.16	0.0067, 0.15	0.37	0.24
1/2,1/6,1/3,1,0,0	0.0, 0.17	0.0067, 0.3	0.37	0.24
置換コストが常に 0	0.0, 0.15	0.0067, 0.076	0.25	0.09
置換コストが常に 1	0.0, 0.17	0.0067, 0.38	0.25	0.09

表 4.6 適合率@3 と再現率@3

重みの組 $\alpha, \beta, \gamma, \delta, \epsilon, \zeta$ または置換コスト	コサイン相違度 閾値範囲	正規化グラフ編集距離 閾値範囲	適合率@3	再現率@3
5/12,0,7/12,1,0,0	0.0, 0.17	0.0067, 0.29	0.35	0.31
0,1/4,3/4,7/12,1/3,1/12	0.0, 0.17	0.0067, 0.28	0.29	0.21
1/6,1/3,1/2,3/4,1/4,0	0.0, 0.17	0.0067, 0.29	0.28	0.20
1/12,1/3,7/12,2/3,1/4,1/12	0.0, 0.17	0.0067, 0.29	0.28	0.20
2/3,1/12,1/4,5/12,0,7/12	0.0, 0.16	0.0067, 0.15	0.33	0.30
1/2,1/6,1/3,1,0,0	0.0, 0.17	0.0067, 0.3	0.35	0.31
置換コストが常に 0	0.0, 0.15	0.0067, 0.076	0.24	0.13
置換コストが常に 1	0.0, 0.17	0.0067, 0.38	0.24	0.13

表 4.7 適合率@4 と再現率@4

重みの組 $\alpha, \beta, \gamma, \delta, \epsilon, \zeta$ または置換コスト	コサイン相違度 閾値範囲	正規化グラフ編集距離 閾値範囲	適合率@4	再現率@4
5/12,0,7/12,1,0,0	0.0, 0.17	0.0067, 0.29	0.30	0.33
0,1/4,3/4,7/12,1/3,1/12	0.0, 0.17	0.0067, 0.28	0.28	0.25
1/6,1/3,1/2,3/4,1/4,0	0.0, 0.17	0.0067, 0.29	0.26	0.23
1/12,1/3,7/12,2/3,1/4,1/12	0.0, 0.17	0.0067, 0.29	0.28	0.25
2/3,1/12,1/4,5/12,0,7/12	0.0, 0.16	0.0067, 0.15	0.30	0.33
1/2,1/6,1/3,1,0,0	0.0, 0.17	0.0067, 0.3	0.30	0.33
置換コストが常に 0	0.0, 0.15	0.0067, 0.076	0.22	0.17
置換コストが常に 1	0.0, 0.17	0.0067, 0.38	0.23	0.18

表 4.8 適合率@5 と再現率@5

重みの組 $\alpha, \beta, \gamma, \delta, \epsilon, \zeta$ または置換コスト	コサイン相違度 閾値範囲	正規化グラフ編集距離 閾値範囲	適合率@5	再現率@5
5/12,0,7/12,1,0,0	0.0, 0.17	0.0067, 0.29	0.27	0.35
0,1/4,3/4,7/12,1/3,1/12	0.0, 0.17	0.0067, 0.28	0.25	0.27
1/6,1/3,1/2,3/4,1/4,0	0.0, 0.17	0.0067, 0.29	0.25	0.26
1/12,1/3,7/12,2/3,1/4,1/12	0.0, 0.17	0.0067, 0.29	0.26	0.27
2/3,1/12,1/4,5/12,0,7/12	0.0, 0.16	0.0067, 0.15	0.26	0.35
1/2,1/6,1/3,1,0,0	0.0, 0.17	0.0067, 0.3	0.27	0.35
置換コストが常に 0	0.0, 0.15	0.0067, 0.076	0.19	0.18
置換コストが常に 1	0.0, 0.17	0.0067, 0.38	0.20	0.19

4.3 考察

4.3.1 重みの組

F 値の上位 10 位では辺の重みについて、呼び出し元の種類の重み δ は $1/2$ 付近もしくはそれ以上の傾向が強く、呼び出し元の位置情報の重みの ϵ と ζ は 0 に近い傾向がある。また、表 4.3 より近傍探索において、 δ を上げ、 ϵ と ζ を下げれば、F 値が高くなる傾向がある。つまり、本実験データにおいて、呼び出し関係では呼び出し元の文脈情報を重要視すれば、F 値が高まると言える。これは、重み ϵ の呼び出し元が何番目の引数で呼ばれているのかの情報が限定的であり、実験データの関数やメソッドの行数が短く呼び出し位置の割合の影響が弱かったため、重み ϵ , ζ が低くなったと考える。

対して、頂点の重みでは、表 4.3 より、 α を $+1/12$ した回数が少なく、 $-1/12$ した回数が多い。 β と γ については、 β は $+1/12$, $-1/12$ ともに回数が多く、 β と γ の間で単純な優劣が付けられない。これは β , γ がそれぞれ return 文の個数と返り値の種類に関するコストの重みで、return 文の個数と返り値の種類に相関があるためだと考える。よって、頂点の重みは α を下げ、 β もしくは γ を上げれば、F 値が高まる。

4.3.2 閾値範囲

コサイン相違度の閾値範囲に微小な変化があった。これは重みの変化により、各関数・メソッドのペアの正規化グラフ編集距離が変化し、その閾値範囲も変化することで、閾値範囲内に True を多く分布させようとコサイン相違度の閾値範囲も変化したと考える。また、正規化グラフ編集距離の閾値範囲の最小値が 0.0067 となっているが、これは元のグラフ編集距離は 0 であったものがシグモイド関数より 0.0067 に写された結果である。シグモイド関数は単調増加であり、元のグラフ編集距離の順列は写した後も保たれるので、0.0067 を最小値として扱っても問題ない。

4.3.3 推薦結果の評価

全体的に k が小さいと適合率@ k が高く、大きいと再現率@ k が高くなった。 k が小さいと適合率@ k が高くなる理由は、ベンチマークの関数・メソッドの数が少ないためである。その数が少ないゆえに、推薦対象に推薦して True となる推薦すべき関数・メソッドが少なく、totalindex が高い False を推薦してしまい、 k の増加に伴って適合率@ k が低くなる。改善策として、True ラベルの数を増やせばよいと考えられるが、現実において推薦対象のコードに対して False となるコード片が多数存在するので、この策は適切ではない。従って、本実験でのベンチマークの関数・メソッドの数に対して、True の数は適切であると考え、適合率@ k の傾向も適切である。再現率@ k について、その実験結果から推薦結果 1 位が True である信頼、つまり 1 位で推薦された関数・メソッドの置き換えが可能である信頼は低く、推薦された 1 位から 5 位までの関数・メソッドの内、どれかは置き換え可能の信頼が高いと言える。しかし、推薦において上位であるほど True でなければいけないため、再現率@ k の実験結果は悪い。また、適合率@ k , 再現率@ k ともに全体的に低く、推薦の精度は悪い。そのため、本実験のベンチマークのような小さな規模の関数・メソッドでも適する特徴の抽出が必要である。

また、置換コストが常に 0 または 1 のときと比較して、全体的に適合率@ k , 再現率@ k ともに高い結果となり、提案手法は置換コストに関数・メソッドの情報を考慮することで、利用可能か不可能かの大まかな違いの判別ができると言える。

第 5 章

まとめと今後の課題

5.1 まとめ

ユーザが記述したコード片に対し、類似処理をするコードの推薦を目的とし、命令と呼び出しグラフを用いた推薦手法を提案した。命令は種類毎の出現回数、呼び出しグラフは、関数やメソッド、その間の情報について考慮した。特に命令は呼び出されている関数・メソッド内の命令も考慮し、命令の相違の指標にコサイン相違度を用いた。また、呼び出しグラフもモジュールの関数・メソッドも展開してグラフに加えるよう考慮し、グラフ編集距離を相違の指標にして、置換操作のコストでは呼び出しグラフが持つ返り値の種類や呼び出し元の文脈などを基に算出した。置換操作のコストの重みと、コサイン相違度と正規化グラフ編集距離の閾値範囲を、F 値を用いて推薦に適するものを求め、6 つ選択した。求めた重みと閾値範囲と、置換操作のコストが常に 0 か 1 のもので、推薦結果の精度の比較実験を行った。その結果、重みを導入したときが、適合率@ k 、再現率@ k ともに上回ったが、その値は決して高いものではなかった。それは、実験データが小さな規模であるためだと考える。

5.2 今後の課題

5.2.1 置換コストに用いる特徴

提案手法では、呼び出しグラフでのグラフ編集距離の置換操作のコストの算出に、関数・メソッドの情報を用いたが、実験結果より、辺における呼び出し元の文脈情報が重要視されていなかった。その理由として実験データの規模の小ささによると考察して、対象の関数・メソッドの規模が大きくなるほど、効果的になると期待される。加えて、効果的な推薦のために関数・メソッドの規模によらずに、適合率@ k 、再現率@ k を向上させる特徴の検討が課題である。

5.2.2 実装・実験

本手法の実装について、命令は Python3.11 のバイトコード命令を用いたが、その仕様により命令が構文レベルで分けられている。例えば、BUILD_TUPLE や LIST_EXTEND, SET_UPDATE などがある。そのため、この実装では構文の違いがあっても処理内容が類似するコードを推薦できないことがある。Python のバイトコード命令をより低級な機械語での命令へ翻訳しそれを用いた実装をしなければならない。

本研究では、実験対象にコード数が少ない自作のプログラムのベンチマークを用いて実験し評価したが、他のベンチマークや大規模なレポジトリなど対象にした実験、評価が求められる。

また、本研究の関連研究の多くは Java を対象としたものやその実験が行われているものが多く、本研究の対象である Python との比較がしにくい。関連研究であるコードクローン検出のベンチマークも同様に Java のベンチマークが広く使われているが、Python を対象としたベンチマークは少ない。クローン検出との比較実験のために、Python 以外の言語での実装が求められる。また、大規模なレポジトリからソースコードを得るために、関連研究の QualBoa で使用される Boa[4] などのコード検索エンジンがあり、それらを用いる実装の検討が必要である。

自作のベンチマークについて、筆者 1 人でラベル付けをしたため、ラベルの偏りの可能性がある。偏りをなくするために複数人でのラベル付けが必要であり、異なるラベルを付けた場合はそれを集約する方法の検討も必要である。また、ラベル付けに関しては筆者の主観に基づいているため、関数・メソッドのペアが類似していると形式的検証で示す必要もある。

推薦結果の評価指標に適合率@ k と再現率@ k を用いたが、他の指標に MRR (Mean Reciprocal Rank) や NDCG (Normalized Discounted Cumulative Gain) などがある。本手法のさらなる精度評価のためにこれらでの評価が必要である。

5.2.3 推薦にかかる時間

厳密なグラフ編集距離の算出の時間は対象となるグラフの構造により変化し、実装に用いた NetworkX の `graph_edit_distance` が用いているアルゴリズムは最悪計算量は $O(n^3)$ とされている [1]。 n はグラフの頂点もしくは辺の数である。グラフの構造が複雑になるほど計算に時間がかかり、推薦の実用では致命的である。本実験においては、グラフ編集距離の算出で 2 秒以上経過したら打ち切るようにしたが、その打ち切り時間が適切であるか定かでなく、その評価も必要である。また、近似的なグラフ編集距離の算出のアルゴリズムを実装したライブラリの使用もしくは実装の検討も必要となる。特に本研究では始点を持つ有向多重グラフを考えていて、始点から始点へ必ず置換させる制約があるため、それに適したアルゴリズムが求められる。

謝辞

本研究を進めるにあたり，日頃からご指導を頂いた山田俊行講師，河内亮周教授，多くの意見や助言を下されたコンピュータソフトウェア研究室の皆様，並びに，研究活動においてお世話になりました森岡幸音事務員，小野寺香里事務員に感謝致します．

参考文献

- [1] Zeina Abu-Aisheh, Romain Raveaux, Jean-Yves Ramel, and Patrick Martineau. "An exact graph edit distance algorithm for solving pattern recognition problems". In *4th International Conference on Pattern Recognition Applications and Methods 2015*, 2015.
- [2] NetworkX developers. "Networkx Network Anlysis in Python", 2024.
<https://networkx.org/>.
- [3] Themistoklis Diamantopoulos, Klearchos Thomopoulos, and Andreas Symeonidis. "Qualboa: reusability-aware recommendations of source code components". In *Proceedings of the 13th International Conference on Mining Software Repositories*, pp. 488–491, 2016.
- [4] Robert Dyer, Hoan Anh Nguyen, Hridesh Rajan, and Tien N Nguyen. "Boa: Ultra-large-scale software repository and source-code mining". *ACM Transactions on Software Engineering and Methodology (TOSEM)*, Vol. 25, No. 1, pp. 1–34, 2015.
- [5] Python Software Foundation. 「dis — python バイトコードの逆アセンブラ」, 2024.
<https://docs.python.org/ja/3.11/library/dis.html>.
- [6] Hao He, Yulin Xu, Yixiao Ma, Yifei Xu, Guangtai Liang, and Minghui Zhou. "A multi-metric ranking approach for library migration recommendations". In *2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pp. 72–83. IEEE, 2021.
- [7] Sifei Luan, Di Yang, Celeste Barnaby, Koushik Sen, and Satish Chandra. "Aroma: Code recommendation via structural code search". *Proceedings of the ACM on Programming Languages*, Vol. 3, No. OOPSLA, pp. 1–28, 2019.
- [8] Kaspar Riesen. "*Structural Pattern Recognition with Graph Edit Distance: Approximation Algorithms and Applications*". Springer Cham, 2015.
- [9] Chanchal Kumar Roy and James R Cordy. "A survey on software clone detection research". Queen' s School of computing TR, Vol. 541, No. 115, pp. 64–68, 2007.
- [10] Dongjin Yu, Jiazha Yang, Xin Chen, and Jie Chen. "Detecting java code clones based on bytecode sequence alignment". *IEEE Access*, Vol. 7, pp. 22421–22433, 2019.

付録 A

実験で用いたソースコード

ソースコード A.1 l1distance.py

```
1 def l1distance(x1, y1, x2, y2):
2     if x1 > x2:
3         tx = x1 - x2
4     else:
5         tx = x2 - x1
6     if y1 > y2:
7         ty = y1 - y2
8     else:
9         ty = y2 - y1
10    return tx + ty
```

ソースコード A.2 l1distance2.py

```
1 def myabs(a):
2     if a >= 0:
3         return a
4     else:
5         return -a
6
7 def l1distance2(x1, y1, x2, y2):
8     return myabs(x1 - x2) + myabs(y1 - y2)
```

ソースコード A.3 l1distance3.py

```
1 import mymath
2
3 def l1distance3(x1, y1, x2, y2):
4     return mymath.mymax(x1, x2) - mymath.mymmin(x1, x2) + mymath.mymax(y1, y2) -
        mymath.mymmin(y1, y2)
```

ソースコード A.4 l2distance.py

```
1 import mymath
2
3 def l2distance(x1, y1, x2, y2):
4     return mymath.mysqrt((x1 - x2)**2 + (y1 - y2)**2)
```

ソースコード A.5 l2distance2.py

```
1 def l2distance2(x1, y1, x2, y2):
2     tmp1 = (x1 - x2) * (x1 - x2)
3     tmp2 = (y1 - y2) * (y1 - y2)
4     tmp3 = tmp1 + tmp2
5     return tmp3 ** (1/2)
```

ソースコード A.6 l2distance3.py

```
1 import mymath
2
3 def l2distance3(x1, y1, x2, y2):
4     t = mymath.mypow(x1 - x2, 2) + mymath.mypow(y1 - y2, 2)
5     return mymath.mypow(t, 0.5)
```

ソースコード A.7 myitem.py

```
1 class Item:
2     def __init__(self, name, price):
3         self.name = name
4         self.price = price
5     def get_name(self):
6         return self.name
7     def get_price(self):
8         return self.price
9
10 class Items:
11     def __init__(self, items):
12         self.items = items
13     def get_sumprice(self):
14         total = 0
15         for i in self.items:
16             total += i.get_price()
17         return total
18
19 if __name__ == '__main__':
20     a = Item('itemA',100)
21     b = Item('itemB',200)
22     c = Item('itemC',300)
23     items = Items([a,b,c])
24     print(items.get_sumprice())
```

ソースコード A.8 myitem2.py

```
1 class Item:
2     def __init__(self, name, price, num):
3         self.name = name
4         self.price = price
5         self.num = num
6     def get_name(self):
```

```

7         return self.name
8     def get_price(self):
9         return self.price
10    def get_num(self):
11        return self.num
12
13 class Items:
14     def __init__(self, items):
15         self.items = items
16     def print_name_price_num(self):
17         for i in self.items:
18             print(i.name, i.price, i.num)
19     def get_average(self):
20         result = 0
21         total_num = 0
22         for i in self.items:
23             result = result + i.price * i.num
24             total_num = total_num + i.num
25         result = result / total_num
26         return result
27
28 if __name__ == '__main__':
29     a = Item('itemA',100,2)
30     b = Item('itemB',200,1)
31     c = Item('itemC',300,4)
32     items = Items([a,b,c])
33     items.print_name_price_num()
34     print(items.get_average())

```

ソースコード A.9 mymath.py

```

1 def mymax(a, b):
2     if a >= b:
3         return a
4     else:
5         return b
6
7 def mymin(a, b):
8     if b >= a:
9         return a
10    else:
11        return b
12
13 def myabs(a):
14     return -a if a < 0 else a
15
16 def mysqrt(a):
17     return a**0.5
18

```

```
19 def mypow(a, n):
20     return a ** n
```

ソースコード A.10 mymath2.py

```
1 def mysum(l):
2     t = 0
3     for i in l:
4         t += i
5     return t
6
7 def mysum2(l):
8     t = 0
9     for i in range(len(l)):
10         t += l[i]
11     return t
```

ソースコード A.11 myaverage.py

```
1 def myaverage(l):
2     t = 0
3     for i in l:
4         t += i
5     a = t / len(l)
6     return a
7
8 def myaverage2(l):
9     t = 0
10    for i in range(len(l)):
11        t += l[i]
12    a = t / (i + 1)
13    return a
```

ソースコード A.12 myaverage2.py

```
1 import mymath_b
2
3 def myaverage3(l):
4     t = mymath_b.mysum(l)
5     a = t / len(l)
6     return a
7
8 def myaverage4(l):
9     t = mymath_b.mysum2(l)
10    n = 0
11    try:
12        while True:
13            l[n]
14            n += 1
15    except IndexError:
```

```
16         pass
17     a = t / n
18     return a
```

ソースコード A.13 mymode.py

```
1 import mysort
2 def mymode(l):
3     c_max = 0
4     for i in range(len(l)):
5         c = 1
6         for j in range(i+1, len(l)):
7             if l[i] == l[j]:
8                 c += 1
9             if c > c_max:
10                 c_max = c
11                 mode = l[i]
12
13     return mode
14
15 def mymode2(l):
16     histogram = dict()
17
18     for i in l:
19         if i in histogram:
20             histogram[i] += 1
21         else:
22             histogram[i] = 1
23
24     max_c = 0
25     for k, v in histogram.items():
26         if max_c < v:
27             max_c = v
28             mode = k
29     return mode
30
31 def mymode3(l):
32     mysort.mybubble_sort(l)
33     current = None
34     c = 1
35     max_c = 1
36     for i in l:
37         if i == current:
38             c += 1
39         else:
40             current = i
41             c = 1
42
43     if c > max_c:
```



```
44         max_c = c
45         mode = current
46
47     return mode
```

ソースコード A.14 myharmonicaverage.py

```
1 def myharmonicaverage(l):
2     total = 0
3     try:
4         for i in l:
5             total += 1.0 / i
6         result = len(l) / total
7     except ZeroDivisionError:
8         result = 0
9     return result
```

ソースコード A.15 mysort.py

```
1 def mybubble_sort(l):
2     for i in range(len(l)):
3         for j in range(len(l)-1-i):
4             if l[j] > l[j+1]:
5                 tmp = l[j]
6                 l[j] = l[j+1]
7                 l[j+1] = tmp
```
