社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

プログラム構造が自動生成テストの網羅率に与える影響の調査

渡邉　凌雅†　　肥後　芳樹†　　楠本　真二†

† 大阪大学大学院情報科学研究科
E-mail: †{ryg-wtnb,higo,kusumoto}@ist.osaka-u.ac.jp

あらまし　ソフトウェア開発工程の 1 つに単体テストがある．この工程では，関数やメソッドといったプログラムを構成する単位が開発者の想定通りに振る舞うかどうかを検証する．単体テストを実施するためには，テスト対象ユニットに対応するテストスイートを用意する必要がある．しかし，ユニット全体を網羅できるテストスイートを作成する作業には多大な労力を要する．そのため，単体テストのテストスイートを自動的に生成する研究が行われており，その成果として，さまざまなテスト生成ツールが公開されている．だが，これらのツールは入力したプログラムを完全に網羅したテストスイートを生成できない場合がある．本研究では，この問題の原因を入力プログラムの構造に着目して調査する．調査の結果，テスト生成ツールが生成したテストスイートで網羅できない原因として 4 つのパターンに分類できることを発見した．また，これらのパターンに対し網羅できない原因を解決する方法が存在するか考察した．

キーワード　単体テスト，網羅率，テスト生成ツール

1. はじめに

ソフトウェア開発工程の 1 つに単体テストがある．この工程では，プログラムを構成する単位が開発者の想定通りに振る舞うかどうかを検証する．プログラム開発の早期段階でバグや問題を特定できるため，単体テストはソフトウェア開発において欠かせない工程となっている．単体テストを実施するためには，テスト対象ユニットに対応するテストスイートを用意する必要がある．しかし，ユニット全体を網羅したテストスイートを作成する作業は多大な労力を要することから，単体テストのテストスイートを自動的に生成する研究が行われている [1]．その成果として，EvoSuite [2] や Randoop [3]，SUSHI [4] といったテスト生成ツールが公開されている．

しかし，既存のテスト生成ツールでは，テスト対象ユニットを完全に網羅したテストスイートを生成できない場合がある．この問題は，テスト生成ツールの生成アルゴリズムやテスト生成時のパラメータ設定，あるいは入力プログラムの構造など，さまざまな要因に左右されると考えられる．本研究では，それらの要因の 1 つとして入力プログラムの構造に着目しこの問題の原因を調査する．

まず，テスト生成ツールが生成したテストスイート（以下，生成テストスイート）で網羅できないコードを持つメソッドに対し，そのコードが網羅されなかった原因を調査した．その後，そのコードを網羅できない原因の共通点を考察し，得られた共通点をもとにパターン化した．調査の結果，網羅できない原因に見られる特徴として，「特定の値を要求」「特定の型を要求」「実行不可能コード」「マルチスレッド処理」の 4 つのパターンに分類できることを発見した．また，これらのパターンに対し網羅できない原因を解決する方法が存在するかどうかを，EvoSuite の実装と入力プログラムの構造という 2 つの観点から考察した．

2. 準　　備

2.1 単体テスト

ソフトウェア開発では，プログラムが開発者の想定通りに振る舞うかどうかを検証するソフトウェアテストが実施される．ソフトウェアテストはテスト対象の粒度に応じて数段階に分けて実施される．

単体テストとは，テスト対象の単位としてもっとも粒度の小さい単位である関数やメソッドに対しテストを実施する工程である．プログラム実装後の早期段階で実施されるため，比較的早期にバグや問題を特定できるという利点がある．そのため，単体テストはソフトウェア開発において欠かせない工程となっている．

単体テストでは，作業効率化のためにテスト自動生成フレームワークが使われる．テスト自動化フレームワークとは，単体テストを実行するためのソフトウェアおよびテストケースの記述方法を提供するフレームワークである．テスト実行の自動化や，実行結果をまとめたレポートの自動生成といった機能を持つ．Java 用のテスト自動化フレームワークとしては JUnit(注1)があり，Eclipse(注2)や Intellij IDEA(注3)といった代表的な Java 向け統合開発環境でサポートされている．

単体テストを実施するためには，テスト対象ユニットに対応するテストスイートを用意する必要がある．テストスイートとは，何らかのテスト目標を達成するために作成されたテストケースの集合を指す．テストケースとは，テスト項目の最小単位であり，テスト対象への入力と想定される結果の組み合わせで構成される．

JUnit におけるテストスイートの例を図 1 に示す．図 1(a) は，

(注1)：https://junit.org/junit5/
(注2)：https://www.eclipse.org/
(注3)：https://www.jetbrains.com/ja-jp/idea/

```java
public class Sample {
    public boolean isOdd(int n) {
        boolean result = false;
        if (n % 2 != 0) {
            result = true;
        }
        return result;
    }
}
```

(a) テスト対象クラスの例

```java
import org.junit.Test;
import static org.junit.Assert.assertFalse;
public class SampleTest {
    @Test
    public void testEven() throws Exception {
        Sample sample = new sample();
        int n = 2;
        boolean result = sample.isOdd(n);
        assertFalse(result);
    }
}
```

(b) テストスイートの例

図 1　テスト対象クラスとテストスイートの例

この例においてテスト対象となるクラス `Sample` である．このクラスに含まれるメソッド `isOdd()` は，入力された値が奇数であるかどうかを返すメソッドである．図 1(b) は，`Sample` クラスをテストするためのクラス `SampleTest` である．`SampleTest` に含まれるメソッド `testEven()` は，`Sample.isOdd()` に対し正の偶数を入力したときの動作を検証するテストケースである．この場合，想定結果は `false` となるため，`assertFalse()` で返り値が `false` であると主張する．

2.2　網　羅　率

網羅率とは，テスト対象ユニットにおいてテストされた割合を表す指標である．網羅率を計測する際の基準を網羅基準と呼ぶ．網羅基準にはさまざまな種類があり，代表的な基準として命令網羅（C0），分岐網羅（C1），条件網羅（C2）がある．ここでは，本研究で調査基準として選択した行網羅(注4)と分岐網羅について説明する．

行網羅では，テスト対象ユニットに含まれる実行可能な行のうち，テストスイートによって実行された割合を計測する．行網羅によって網羅率を計測する場合，以下の式を用いる．

$$\text{網羅率} = \frac{\text{テストスイートによって実行された行数}}{\text{テスト対象ユニットに含まれる実行可能な行の総数}}$$

分岐網羅では，テスト対象ユニットに含まれる分岐に対し，その分岐条件が真となる場合と偽となる場合をテストスイートによって網羅できたかどうかで計測する．分岐網羅によって網羅率を計測する場合，以下の式を用いる．

$$\text{網羅率} = \frac{\text{テストスイートによって実行された分岐の数}}{\text{テスト対象ユニットに含まれる分岐の総数}}$$

これらの 2 つの網羅基準を比較すると，行網羅よりも分岐網羅の方が網羅するための条件が厳しい．例として，図 1(a) に含まれるメソッド `isOdd()` を完全に網羅するために必要なテストケースを網羅基準ごとに比較する．表 1 は，2 つのテストスイートを考えた際に，それぞれのテストスイートによって `isOdd()` を完全に網羅できるかどうかを示している．テストスイート 1 では，4 行目の分岐が `true` に分岐し 5 行目が実行されるため，行網羅の基準では `isOdd()` を完全に網羅できる．一方で，`false` には分岐しないため分岐網羅の基準では `isOdd()` を完全には網羅できない．完全に網羅するためには，テストスイート 2 のように，奇数を入力するテストケースに加えて偶数を入力するテストケースが必要である．テストケース 2 では，偶数の入力によって `false` の分岐を網羅できるため，両方の網羅基準で `isOdd()` を完全に網羅できる．

(注4)：本研究で使用したテスト生成ツールである EvoSuite（2.4 節で説明）では，命令網羅の代わりに行網羅を使用する [2]．EvoSuite はテスト生成時にバイトコード表現を参照するため，命令網羅では網羅対象を正確に計測できない．なぜならば，バイトコード命令と実際のソースコード上の命令が直接対応しない場合があるためである．

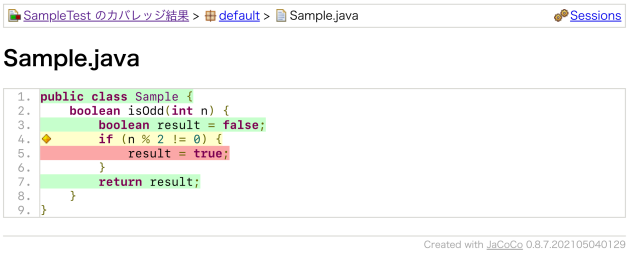

図 2　JaCoCo のカバレッジレポート出力例

2.3　網羅率計測ツール

網羅率計測ツールとは，単体テストにおけるテストスイートの網羅率および網羅箇所を計測し可視化するツールである．Java 言語向けのツールとしては JaCoCo(注5)や Clover(注6)がある．

網羅率計測ツールが出力するレポートの例を示す．図 2 は，図 1(a) の `Sample` クラスに対し図 1(b) のテストスイートを実行し，その結果を JaCoCo で出力した際に得られるカバレッジレポートである．行や分岐がテストスイートによって網羅された場合は緑で，分岐の一部だけが網羅された場合は黄で，網羅できなかった場合は赤でハイライトされる．このように，テストスイートによる網羅の可否が行単位で可視化される．そのため，網羅率計測ツールは，網羅できなかったコードを特定したりどのようなテストケースが不足しているかを確認したりするのに役立つ．

2.4　テスト生成ツール

一般に，単体テストのテストスイートを作成する作業は多大な労力を要する．そこで，単体テストのテストスイートを自動的に生成する研究が行われている [1]．その成果として，EvoSuite [2] や Randoop [3]，SUSHI [4] といったテスト生成ツールが公開され

表 1　網羅基準と網羅に必要なテストスイートの関係

テストスイート		`isOdd()` を完全に網羅できるか	
番号	入力	行網羅	分岐網羅
1	奇数	Yes	No
2	奇数 偶数	Yes	Yes

(注5)：`https://www.jacoco.org/jacoco/`
(注6)：`https://openclover.org/`

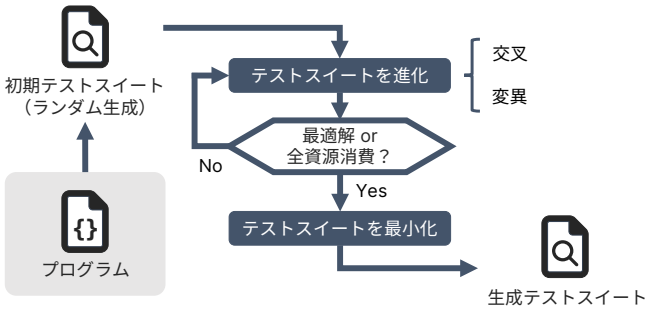

図 3　テストスイート生成における EvoSuite の動作

ている．テスト生成ツールは，入力したプログラムを最大限網羅するようなテストスイートを自動生成する．テスト生成ツールがテストスイートを生成する方針はツールによってさまざまである．例えば，EvoSuite では遺伝的アルゴリズムを使用する．一方で，Randoop ではランダムテストを使用する．また，SUSHI では遺伝的アルゴリズムに加えてシンボリック実行を使用する．

本研究で使用した EvoSuite について説明する．EvoSuite は，遺伝的アルゴリズムをベースに，ハイブリッド検索 [5]，動的シンボリック実行 [6]，テスト容易性変換 [7] といった技術を利用して網羅率の高い JUnit テストスイートを生成するツールである．EvoSuite は，さまざまな Java 向けのテスト生成ツールの中でも特に網羅率の高いテストスイートを生成できる [8] 点が特徴である．

EvoSuite がテストスイートを生成する際の動作アルゴリズム [9] を説明する．アルゴリズムの概略を図 3 に示す．EvoSuite では，遺伝的アルゴリズムにおける個体をテストスイートに含まれるテストケースとして表現する．まず，入力したプログラムをもとにランダムなテストケースを生成し，それらの集合を初期テストスイートとする．次に，このテストスイートを初期世代として，ある世代が最適解に達するか，割り当てられた計算資源を使い果たすまでテストスイートを進化させ続ける．進化過程では，親世代の個体に対し交叉や変異を施した個体が子孫の個体となる．交叉は，子孫の個体の誕生時に命令の一部を入れ替える操作を指す．変異は，子孫の個体の誕生時に既存の命令を変更あるいは削除するか，新たな命令を追加する操作を指す．最終的に得られたテストスイートの各テストケースに対し不要な命令を削除してテストスイートを最小化する．

3. Motivating Example

テスト生成ツールは，入力したプログラムに対し，そのプログラムを完全に網羅したテストスイートを生成できない場合がある．この問題は，テスト生成ツールの生成アルゴリズムやテスト生成時のパラメータ設定，あるいは入力プログラムの構造など，さまざまな要因に左右される．本研究では，それらの要因の 1 つである入力プログラムの構造に着目しこの問題の原因を調査する．

この問題を示す例として，EvoSuite が生成したテストスイートで網羅されないコードを含むメソッドの例を図 4[(注7)] に示す．図 4 に含まれるハイライトは，3 行目の条件文 `val == null` の `false` 分岐および 5 行目以降の命令を網羅できなかったことを表している．

(注7)：本稿では，生成テストスイートによって一部だけ実行されたコードを 黄色 で，実行されなかったコードを 橙色 で表す．

```
1  boolean getBooleanProperty(String prop, boolean
       defaultVal) {
2      String val = System.getProperty(prop);
3      if ( val == null )
4          return defaultVal;
5      if ( val.equalsIgnoreCase("true") ) {
6          return true ;
7      } else {
8          return false ;
9      }
10 }
```

図 4　生成テストスイートで網羅できないコードを持つメソッドの例

テスト生成ツールが生成したテストスイートによってこれらのコードが実行されなかった原因について考察する．5 行目以降が実行されない理由は，3 行目の条件文 `val == null` の `false` 分岐が実行できていないためである．そこで，この条件文に登場する変数 `val` に着目すると，この変数は 2 行目の `System.getProperty(prop)` の返り値である．つまり，3 行目の条件が実行されない理由は，`System.getProperty(prop)` が `null` 以外の値を返さないためである．`System.getProperty()` メソッドは，実際のシステムプロパティ名（`"user.dir"`など）を入力するとそのプロパティの値を返し，それ以外の文字列を入力すると `null` を返すという仕様である．

以上より，テスト生成ツールが生成したテストスイートによって実行できなかった原因は，`System.getProperty()` メソッドを使用したために，EvoSuite が入力に実際のシステムプロパティ名を持つテストケースを生成できなかったからと結論づけられる．

4. 調 査 準 備

調査にあたって実施した準備について説明する．まず，複数の Java メソッドで構成されるデータセット [10] に対し EvoSuite でテストスイートを生成する．次に，生成テストスイートで網羅できないコードを持つメソッドを調査対象のメソッドとして抽出する．

4.1 データセット

本研究では，調査対象として 768 件の Java メソッドで構成されるデータセット [10] を使用した．このデータセットに含まれるメソッドはすべて以下のような特徴を持つ．

- メソッドの処理が外部の変数やメソッドに依存しない
- 1 つ以上の引数を持ち，処理終了時に何らかの値を返す
- Java 8 までの言語仕様で記述される
- `java.lang` および `java.util` のみ使用する

4.2 網 羅 基 準

EvoSuite では，テストスイート生成時の網羅目標としてさまざまな網羅基準を設定している．そのため，どの網羅基準に着目するかによって調査結果が大きく異なると考えられる．そこで，本研究では，以下の理由により行網羅と分岐網羅に着目する．

- 代表的な網羅基準である
- JaCoCo による網羅率の計測と網羅箇所の可視化が可能

4.3 手　　順

調査準備の手順を以下に示す．

1. データセットの各メソッド単体で構成されるクラスを作成
2. 各クラスに対し EvoSuite でテストスイートを生成
3. JaCoCo を用いてテストスイートの網羅率を計測
4. 行網羅または分岐網羅が 1.0 未満のメソッドを抽出

手順 1 では，データセットに含まれる各メソッドに対し，そのメソッド単体で構成されるクラスを作成した．このとき，すべてのクラスをコンパイルできるように，クラスファイルの先頭に `import java.lang.*;` と `import java.util.*;` を挿入した．

4.4 結　　果

データセットに含まれる 768 件のメソッドのうち，約 9.5 %に相当する 73 件のメソッドを抽出した．なお，成功テストを生成できなかったメソッドや，JaCoCo のバグにより網羅されていないと誤判定されたメソッドは除外した．

5. 調　　査

生成テストスイートで網羅できないコードを持つメソッドに対し，そのコードが網羅されなかった原因を調査した．その後，網羅できない原因の共通点を考察しパターン化した．

5.1 調 査 方 法

4. 節で抽出したメソッドに対し，JaCoCo で出力したレポートを参照して生成テストスイートが網羅できなかったコードを特定した．次に，そのコードの処理やメソッドのプログラム構造などを踏まえながら，そのコードを網羅できなかった原因を考察した．そして，各メソッドを調査して得られた結果をもとに，網羅できない原因に見られる共通点を考察しパターン化した．

5.2 結　　果

生成テストスイートが何らかのコードを網羅できなかった原因として，「特定の値を要求」「特定の型を要求」「実行不可能コード」「マルチスレッド処理」 の 4 つのパターンが見られた．各パターンとそのパターンに該当したメソッドの件数を表 2 に示す．網羅できないコードを含む 73 件のメソッドのうち，全体の約 79% に相当する 58 件が「特定の値を要求」 のパターンに分類された．残りの 15 件のメソッドは，「実行不可能コード」 「特定の型を要求」「マルチスレッド処理」 の 3 パターンに分類された．

なお，これらのパターンは生成テストスイートによって網羅できない必要条件であるが十分条件ではない．すなわち，これらのパターンに該当する任意のメソッドが網羅できないコードを持つわけではない．

5.2.1 パターン「特定の値を要求」

パターン「特定の値を要求」とは，生成テストスイートで網羅できなかったコードを網羅するために，引数に何らかの条件を満たした値を必要とする場合を指す．このパターンに該当するメソッドで見られた，引数にそのような値を要求する処理の例を表 3 に示す．

表 2　生成テストスイートで網羅できない原因のパターン

パターン	件数
「特定の値を要求」	58
「特定の型を要求」	6
「実行不可能コード」	6
「マルチスレッド処理」	3

```
1 Float parseFloat(String value, float defaultVal) {
2     try {
3         return Float.parseFloat(value);
4     } catch (NumberFormatException e) {
5         ...
6     }
7 }
```

図 5　パターン「特定の値を要求」に該当するメソッドの例

```
1 int uncheckedIntCast(Object x) {
2     if (x instanceof Number)
3         return ((Number) x).intValue();
4     return ((Character) x).charValue();
5 }
```

図 6　パターン「特定の型を要求」 に該当するメソッドの例

このパターンに該当するメソッドの例を図 5 に示す．このメソッドは，`float` 型の数値を表す文字列（`"10.0f"`など）を `float` 型の数値に変換するメソッドである．このメソッドでは，3 行目の `Float.parseFloat()` の正常系処理が生成テストスイートによって網羅されていない．よって，この処理を網羅するためには，`Float.parseFloat()` に対して数値を表す文字列を入力するテストケース，すなわち，数値を表す文字列を引数に取るテストケースが必要である．

5.2.2 パターン「特定の型を要求」

パターン「特定の型を要求」 とは，生成テストスイートで網羅できなかったコードを網羅するために，引数の型のクラスが持つ子クラスの型を要求する処理を含む場合を指す．

このパターンに該当するメソッドの例を図 6 に示す．このメソッドは，`Object` 型の引数を `int` 型にキャストしその値を返すメソッドである．このメソッドでは，2 行目の条件の `false` 分岐，およびその分岐先の命令（4 行目）が網羅されていない．3 行目の分岐では，引数の型が `Number` 型かどうかを判定している．よって，4 行目を網羅するためには，`Number` 型以外の変数を引数に入力するテストケースが必要である．

5.2.3 パターン「実行不可能コード」

パターン「実行不可能コード」 とは，引数にどのような値を与えても論理的に実行不可能なコードを含むため，生成テストスイートによる網羅ができない場合を指す．

このパターンに該当するメソッドの例を図 7 に示す．このメソッドは，文字列 `str` を文字 `ch` で分割し文字列のリストとして返すメソッドである．変数 `ix` は，3 行目において値 0 で初期化されており，`ix` の値が変化しうる 4〜9 行目の処理においても，その値が 0 から減少するような処理は存在しない．よって，10 行目の `if` 文の `false` 分岐は，引数がどのような値であっても実行不可能である．

5.2.4 パターン「マルチスレッド処理」

パターン「マルチスレッド処理」 とは，生成テストスイートで網羅できなかったコードを網羅するために，マルチスレッドを利用したテストケースが必要な場合を指す．EvoSuite は，マルチスレッ

```
1  List<String> splitToList0(String str, char ch) {
2      List<String> result = new ArrayList<>();
3      int ix = 0, len = str.length();
4      for (int i = 0; i < len; i++) {
5          if (str.charAt(i) == ch) {
6              result.add(str.substring(ix, i));
7              ix = i + 1;
8          }
9      }
10     if ( ix >= 0 ) {
11         result.add(str.substring(ix));
12     }
13     return result;
14 }
```

図 7　パターン「実行不可能コード」 に該当するメソッドの例

```
1  Integer apply(Integer i) {
2      try {
3          Thread.sleep(1);
4      } catch ( InterruptedException e ) {
5          e.printStackTrace() ;
6      }
7      return i;
8  }
```

図 8　パターン「マルチスレッド処理」 に該当するメソッドの例

ドを利用したテストスイートの生成に対応していない [9].

このパターンに該当するメソッドの例を図 8 に示す．このメソッドは，現在実行中のスレッドを休止させるメソッドである．このメソッドでは，3 行目の `Thread.sleep()` から発生する検査例外 `InterruptedException` の捕捉（4 行目）と捕捉時の処理（5 行目）が生成テストスイートによって実行されていない．例外 `InterruptedException` は，スレッドが待ち状態，休止状態，占有されている状態のいずれかにおいて，スレッドに割り込みが発生した際に投げられる例外である．この例外を網羅するためには，マルチスレッド処理を利用して `InterruptedException` を意図的に発生させるテストケースが必要である．

6. 考　　察

調査の結果得られた 4 つのパターンについて，網羅できない原因を解決する方法が存在するかどうかを，EvoSuite の実装と入力プログラムの構造という 2 つの観点から考察する．

6.1　パターン「特定の値を要求」 ／「特定の型を要求」

EvoSuite が「特定の値を要求」 や「特定の型を要求」 のパターンに該当する処理を網羅できなかったのは，網羅に必要な条件を満たす値を入力するテストケースが生成されないためであった．

```
// 「特定の値を要求」の場合
if (引数 == 条件を満たす定数) {}

// 「特定の値を要求」の場合
if (引数 instanceof 条件を満たす型) {}
```

図 9　挿入するダミー分岐

EvoSuite は，テストスイート生成時に Java バイトコードに静的に埋め込まれたプリミティブ型および文字列の定数を参照する性質がある [2]．そこで，この性質を利用して，EvoSuite が網羅に必要な条件を満たす定数を入力するテストケースを生成するように誘導できる．

まず，EvoSuite の実装を改善する観点から考察する．この観点からは，テストスイート生成時に利用者が指定した定数や型を何らかの形で参照できる機能を EvoSuite に追加する方法が考えられる．例えば，利用者が EvoSuite に参照させたい定数や型を EvoSuite 実行時の引数として渡す機能が考えられる．

次に，入力プログラムの構造を改善する観点から考察する．この観点からは，条件を満たす定数や型を入力メソッドに何らかの形で埋め込む方法が考えられる．そのような方法の例として，図 9 に示すようなダミー分岐を挿入する方法がある．この方法の適用例を図 5 のメソッドを使用して説明する．3 行目の `Float.parseFloat()` を網羅できない原因は，`Float.parseFloat()` の入力として数値を表す文字列が要求されるためであった．そこで，図 10 のように，引数 `value` が数値を表す文字列の 1 つである`"10.0f"`と一致するかどうかを判定するダミー分岐を挿入する．

この方法の妥当性を検証する．まず，「特定の値を要求」 や「特定の型を要求」 のパターンに該当するメソッドに対しこの方法を適用した．その後，再度 EvoSuite でテストスイートを生成し，指定した定数を引数に入力するテストケースが生成されたか確認した．その結果，網羅に必要な条件を持つ引数が 1 つだけ存在するメソッドについては，そのようなテストケースが生成されていた．一方で，以下の 3 つのすべての特徴を持つメソッドでは，そのようなテストケースが生成されていなかった．

1. 複数の引数を持つ
2. それぞれの引数が網羅に必要な条件を持つ
3. ある引数の条件が他の引数の条件に影響する

よって，この方法では，条件を満たした複数の引数を同時に入力するテストケースを生成するように誘導できないと結論づけられる．

6.2　パターン「実行不可能コード」

定義上，実行不可能コードはどのようなテストケースを用意しても網羅できない．そのため，テストケースの追加や変更による解決

表 3　網羅に特定の条件を満たす値が必要な処理の例

網羅に特定の条件を満たす値が必要な処理	引数に要求される条件の例	条件を満たす引数の例
システムのプロパティを取得	`System.getProperty()` の正常系入力	`"user.dir"`
システムの環境変数を取得	`System.getenv()` の正常系入力	`"PATH"`
数値を表す文字列を数値に変換	`Float.parseFloat()` の正常系入力	`"10.0f"`
文字の種類（大文字や小文字等）を専用のメソッドで判定	`Character.isLowerCase()` の正常系入力	`'a'`
文字コードの大小関係で文字を比較	Unicode で `U+0x80` 以上 `U+0x800` 未満の文字	`'Ã'`
キャストでオーバーフローを判定	`long` 型では表現できるが `int` 型ではオーバーフローする整数	`2147483648L`

```
1    Float parseFloat(String value, float defaultVal) {
2  +     if (value == "10.0f") {}
3        try { return Float.parseFloat(value);
4        } catch (NumberFormatException e) {
5            ...
6        }
7    }
```

図 10　ダミー分岐の挿入例

は不可能である．代わりに，実行不可能コードを網羅対象から除外したり，テストスイート生成時の網羅優先度を下げたりする方法が考えられる．

EvoSuite では，ある網羅基準における個々の網羅目標を最適化する代わりに，その網羅基準全体を最適化するアプローチを採用している [9]．これにより，実行不可能コードにおける網羅優先度の低下を実現している．

また，実行不可能コードの一部は統合開発環境の機能などを利用して検出できる．この機能を活用して，プログラマがテストスイート生成前に実行不可能コードを可能な限り削除すれば，実行不可能コードの一部を網羅対象から除外できる．例えば図 7 のメソッドであれば，この方法は 10 行目の `if` 文を削除することに相当する．

6.3　パターン「マルチスレッド処理」

EvoSuite は，マルチスレッド処理を利用したテストスイートの生成に対応していない [9]．そのため，入力プログラムを改善する観点からの解決は不可能である．一方で，EvoSuite を発展させてマルチスレッド処理に対応したテストスイートを生成する研究 [11] が行われている．

7.　妥当性の脅威

本研究では，4.1 節で述べたような特徴を持つメソッドを対象に調査した．しかし，実社会における Java プロジェクトでは，外部の変数やメソッドなどに依存したり，外部パッケージを使用したりするのが一般的である．よって，本研究とは異なるデータセットや Java プロジェクト等を対象に調査を実施した場合，異なる結果を得る可能性がある．

本研究では，網羅基準として行網羅と分岐網羅に着目して調査した．しかし，EvoSuite は例外網羅や出力網羅といったさまざまな基準を網羅目標に設定する [12]．そのため，行網羅や分岐網羅以外に対して調査を実施した場合，異なる結果を得る可能性がある．

また，2.4 節で述べたように，テスト生成ツールにはさまざまな種類が存在し，ツールによってテストの生成方針が異なる．そのため，EvoSuite 以外のツールを使用して調査を実施した場合，異なる結果を得る可能性がある．

8.　おわりに

本研究では，テスト生成ツールに入力するプログラムの構造が生成テストスイートの網羅率に与える影響について調査した．まず，Java メソッドに対し EvoSuite でテストスイートを生成し，行網羅と分岐網羅の少なくとも一方が 1.0 未満のメソッドを抽出した．次に，生成テストスイートが網羅できないコードに対しそのコードを網羅できない原因を調査した．その結果，網羅できない原因として「特定の値を要求」「特定の型を要求」「実行不可能コード」「マルチスレッド処理」の 4 つのパターンを発見した．また，これらのパターンに対し，網羅できない原因を解決する方法が存在するか考察した．その結果，「特定の値を要求」「特定の型を要求」「実行不可能コード」のパターンに対して解決方法を提案した．

今後の課題としては，本研究で使用したデータセットとは異なるデータセットに対して調査を実施する点が挙げられる．また，EvoSuite 以外のテスト生成ツールを用いた際にどのような結果が得られるかを検証する点も今後の課題である．

謝辞　本研究は JSPS 科研費（JP20H04166，JP21K18302，JP21K11820，JP21H04877，JP22H03567，JP22K11985）の助成を得て行われた．

文　献

[1] P. McMinn, “Search-based software test data generation: a survey,” Software Testing, Verification and Reliability, pp.105–156, 2004.
[2] G. Fraser and A. Arcuri, “Evosuite: automatic test suite generation for object-oriented software,” Proc. European Conference on Foundations of Software Engineering, pp.416–419, 2011.
[3] C. Pacheco and M.D. Ernst, “Randoop: feedback-directed random testing for Java,” Proc. Object-Oriented Programming, Systems, Languages and Applications, pp.815–816, 2007.
[4] P. Braione, G. Denaro, A. Mattavelli, and M. Pezzè, “SUSHI: A test generator for programs with complex structured inputs,” Proc. International Conference on Software Engineering, pp.21–24, 2018.
[5] M. Harman and P. McMinn, “A theoretical and empirical study of search-based testing: Local, global, and hybrid search,” IEEE Transactions on Software Engineering, pp.226–247, 2010.
[6] P. Godefroid, N. Klarlund, and K. Sen, “DART: Directed automated random testing,” Proc. ACM SIGPLAN Conference on Programming Language Design and Implementation, p.213 – 223, 2005.
[7] M. Harman, L. Hu, R. Hierons, J. Wegener, H. Sthamer, A. Baresel, and M. Roper, “Testability transformation,” IEEE Transactions on Software Engineering, pp.3–16, 2004.
[8] S. Vogl, S. Schweikl, G. Fraser, A. Arcuri, J. Campos, and A. Panichella, “Evosuite at the sbst 2021 tool competition,” Proc. International Workshop on Search-Based Software Testing, pp.28–29, 2021.
[9] G. Fraser and A. Arcuri, “Whole test suite generation,” IEEE Transactions on Software Engineering, pp.276–291, 2013.
[10] Y. Higo, S. Matsumoto, S. Kusumoto, and K. Yasuda, “Constructing dataset of functionally equivalent java methods,” Proc. International Conference on Mining Software Repositories, pp.682–686, 2022.
[11] S. Steenbuck and G. Fraser, “Generating unit tests for concurrent classes,” Proc. International Conference on Software Testing, Verification and Validation, pp.144–153, 2013.
[12] J.M. Rojas, J. Campos, M.V.G. Fraser, and A. Arcuri, “Combining multiple coverage criteria in search-based unit test generation,” Search-Based Software Engineering, pp.93–108, Springer, 2015.