メソッド呼び出しに基づくJavaプログラムの類似度計算と可視化

東京情報大学　総合情報学部
宇田川 佳久

1

目　次

2

1. 研究の背景と概要

- ICTによる社会・生産の改革が進められている *(DX)*。
- プログラミング技法は複雑さを増し，ソフトウェア開発の効率化やエンジニアの育成にも改革が求められている *(EX)*。
- インターネットに公開されているサンプルプログラムを参考にする方法がある。
- サンプルプログラムの違いや類似性は，プログラムの中身を目視で確認しなければならならず，この作業がプログラミング学習の制約となっていた。
- 類似するプログラムを検索することで，プログラミング開発や学習を支援することを研究テーマとした。
- 本研究の目的は、類似するプログラムの検索方法と表示方法を実証することである。

3

従来の研究との違い

(1)メソッド単位の類似度をAPIの種類と引用回数に基づいて算出する。

(2)メソッド単位の類似度をグラフで表示し，関連性を直感的に把握できる機能を提供している。

(3)頻出するAPIの組合わせを算出する機能により，サンプルプログラムが提示しているAPIの組合わせを頻度順に提示できる。

2021/5/21 4

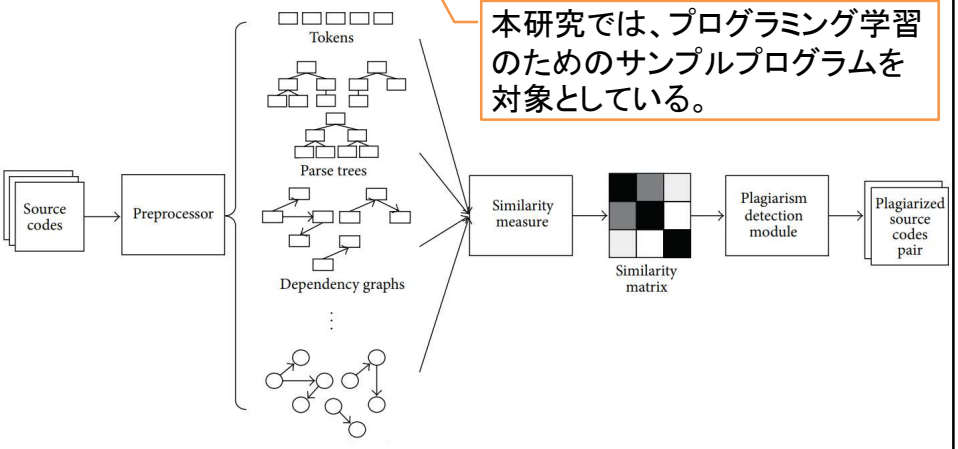
関連研究

1. F. Zhang, Guofan Li, Cong Liu, and Qian Song, "Flowchart-Based Cross-Language Source Code Similarity Detection," Hindawi Scientific Programming Vol. 2020, Article ID 8835310, pp.1-15.
2. R. Pelánekr, T. Effenberger, M. Vanek, V. Sassmann, and D. Gmiterko, "Measuring Item Similarity in Introductory Programming," Proceedings of the Fifth Annual ACM Conference on Learning at Scale, June 2018, pp.1-4.
3. C. Ragkhitwetsagul, "Measuring Code Similarity in Large-scaled Code Corpora," IEEE International Conference on Software Maintenance and Evolution (ICSME), Oct. 2016.
4. H-J. Song, S-B. Park, and S. Y. Park, "Computation of Program Source Code Similarity by Composition of Parse Tree and Call Graph," Mathematical Problems in Engineering Vol. 2015, Article ID 429807, pp.1-12.
5. I. Keivanloo, J. Rilling, and Y. Zou, "Spotting working code examples," Proceedings of the 36th International Conference on Software Engineering, May 2014, pp.664–675.
6. F. A. Shamsi and A. Elnagar, "An Intelligent Assessment Tool for Students' Java Submissions in Introductory Programming Courses," Journal of Intelligent Learning Systems and Applications, Apr. 2012, pp.59-69.

5

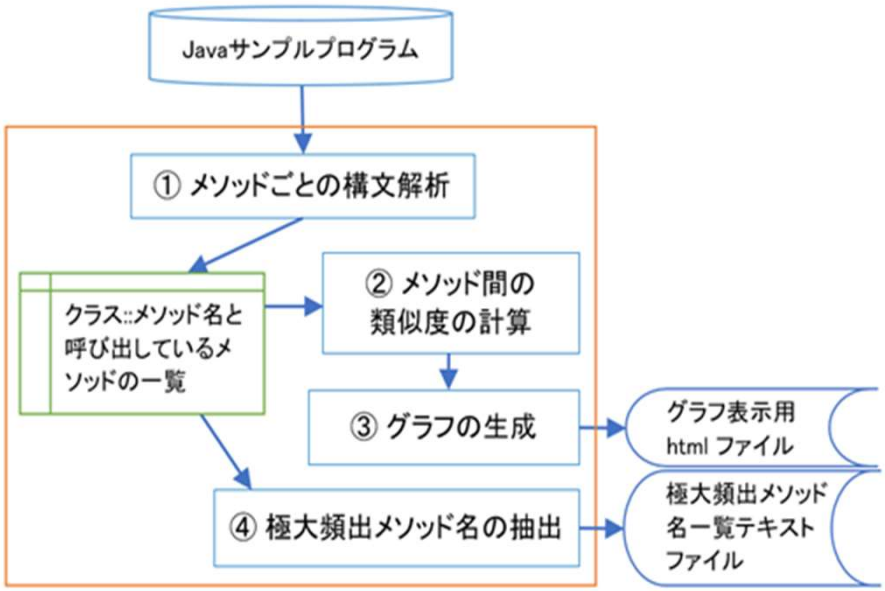
2015年 H-Je Song氏らは[4]，プログラムのローカルな制御構造を循環複雑度で指標化し，プログラムの全体構造をメソッド呼び出しで表すことを提案した。これらの2種類の指標を使うことで，従来手法よりも高い精度で類似プログラムを検出できることを実験により示した。プログラムの盗用検知が目的。

本研究では、プログラミング学習のためのサンプルプログラムを対象としている。



Function call graph

2. 開発した機能と処理流れ



3. メソッドの構文解析と類似度の計算

- Javaプログラムの構文解析では，JDT（Java Development Tools）-Core の Scanner クラスを使用した．

```
public class LongestCommonSubsequence_2 {

    int lcs( char[] X, char[] Y, int m, int n ) {
      int L[][] = new int[m+1][n+1];

      for (int i=0; i<=m; i++) {
        for (int j=0; j<=n; j++) {
          if (i == 0 || j == 0)
              L[i][j] = 0;
          else if (X[i-1] == Y[j-1])
              L[i][j] = L[i-1][j-1] + 1;
          else
              L[i][j] = max(L[i-1][j], L[i][j-1]);
        }
      }
    return L[m][n];
    }

    int max(int a, int b) {
      return (a > b)? a : b;
    }
```

1	1	.	17	36	new
2	2	++	18	40	static
3	4	+	19	42	1
4	5	-	20	47	
5	6	[	21	49	{
6	12	<=	22	57	public
7	15	>	23	63	:
8	19	==	24	67	]
9	22	n	25	68	class
10	23	(	26	73	=
11	25	;	27	82	for
12	26	)	28	83	if
13	29	?	29	84	return
14	31	||	30	105	char
15	32		31	109	int
16	33	}	32	112	void
			33	114	else

2021/5/21

３．メソッドの構文解析と類似度の計算

- Javaプログラムを構成するトークンを100種類以上に分類する機能を提供している．また，コメントを除外するため，実行文の解析を効率的に行える．
- 一行に複数のメソッドがある場合にも対応している。
- println() や printStackTrace() など，共通的に使用されるメソッドは，除外する機能を作り込んだ．

```
private static void timeMeasure(String MID) {
    try {
        System.out.println(MID);
        Method method = Class.forName("Sample1607").getMethod(MID);
        long startTime = System.currentTimeMillis();
        method.invoke(null); // メソッドの実行。
        long endTime = System.currentTimeMillis();
        System.out.println(MID+" ReadTime : "+(endTime - startTime)+" msec");
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```


- 任意個のメソッドとメソッド呼び出しを管理するために、Mapの ArrayList を使っている。
- グラフ表示には、Vis.js を使っている。

```
int cnt= 0;
ArrayList< Map<String, Integer> >  MCall;
for( Javaソースファイル) {
    if(クラス名::メソッド宣言){
        cnt++;
        MCall.add(cnt, Map(呼び出されているメソッド名, 呼出し数) );
}
Num= Mcall.size();      //メソッド宣言の数;
GR= ∅;
for( j=0; j < Num-1; j++ ) { // メソッド間の類似度を計算する
    for( k=j+1; k < Num; k++ ) {
        //  MCallのメソッド名と呼出し数を使ってCos類似度を計算する。
        Sim[j][k]= CosSim( j, k );
        if( Sim[j][k] > 閾値 )
            // ノードj, kをアークSim[j][k]で結合したネットワークグラフの生成
            GR= GR ∪ Gen_Graph(j, k);
    }
}
```

総当たりで類似度を計算する。

閾値以上ならグラフ表示する。

メソッド抽出処理の概要（事例）

```
1  package Sample_1;
2  import java.io.*;
3  import java.lang.reflect.Method;
4
5  class Sample1607 {
6      static String FName = "c:\\temp\\Apriori\\MD_Struct.txt";
7      public static void main(String args[]) {
8          timeMeasure("ReadByte");
9          timeMeasure("ReadByteBuffered");
10         timeMeasure("ReadText");
11         timeMeasure("ReadTextBuffered");
12     }
13     public static void ReadByte() throws IOException {
14         FileInputStream fis = new FileInputStream(FName);
15         int code;
16         while ((code = fis.read()) != -1) {
17             Integer.toHexString(code);
18         }
19         fis.close();
20     }
           中略
39     public static void ReadTextBuffered() throws IOException {
40         BufferedReader br = new BufferedReader(new FileReader(FName));
41         int data;
42         while ((data = br.read()) != -1) {
43             Character.toChars(data);
44         }
45         br.close();
46     }
47     private static void timeMeasure(String MID) {
48         try {
49             System.out.println(MID);
50             Method method = Class.forName("Sample1607").getMethod(MID);
51             long startTime = System.currentTimeMillis();
52             method.invoke(null); // メソッドの実行。
53             long endTime = System.currentTimeMillis();
54             System.out.println(MID+" ReadTime : "+(endTime - startTime)+"
55         } catch (Exception e) {
56             e.printStackTrace();
```

2021/5/21

```
Sample1607::main(String[])
  timeMeasure,4
Sample1607::ReadByte()
  FileInputStream,1
  close,1
  read,1
  toHexString,1
Sample1607::ReadByteBuffered
  BufferedInputStream,1
  FileInputStream,1
  close,1
  read,1
  toHexString,1
Sample1607::ReadText()
  FileInputStream,1
  InputStreamReader,1
  close,1
  read,1
  toChars,1
Sample1607::ReadTextBuffered
  BufferedReader,1
  FileReader,1
  close,1
  read,1
  toChars,1
Sample1607::timeMeasure(Stri
  currentTimeMillis,2
  forName,1
  getMethod,1
  invoke,1
```

類似度の計算

- 呼び出しているメソッド名と個数で、Wordのベクトルを表現する。
- Wordのベクトル間の Cos類似度を計算する。
- 呼び出している個数を反映した類似度を計算できる。
- なお、Wordの発生頻度は考慮していない（tf-idfによる補正はしていない）。

$$\mathbf{a}=(a_1, a_2,\ldots, a_n),\ \mathbf{b}=(b_1, b_2,\ldots, b_n)$$

$$cos(a,b) = \frac{\sum_{i=1}^{n} a_i b_i}{\sqrt{\sum_{i=1}^{n} a_i^2}\sqrt{\sum_{i=1}^{n} b_i^2}}$$

- 処理対象とするメソッドを総当たりでCos類似度を計算する。
- 自身の類似度（常に1）は計算していない。

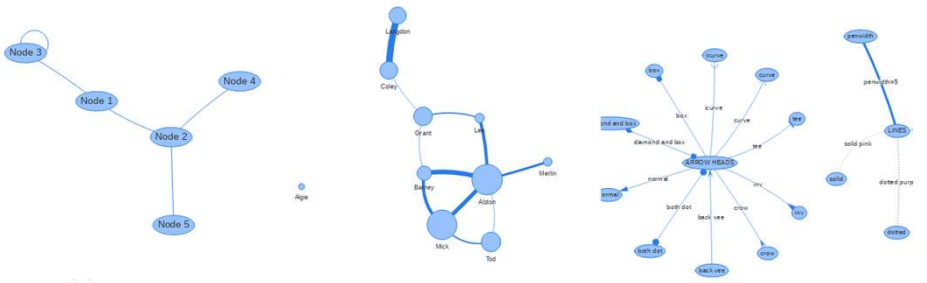
```
== Gen_Lsit_of_Called_Methods ===
0/1=0.146   0/2=0.408   0/3=0.365   0/4=0.612   0/5=0.236   0/6=0.408   0/7=0   0/8=0.408   0/9=0.365
1/2=0   1/3=0   1/4=0   1/5=0   1/6=0   1/7=0   1/8=0   1/9=0   1/10=0   1/11=0   1/12=0   1/13=0   1/1
2/3=0.894   2/4=0.75   2/5=0.577   2/6=0.5   2/7=0   2/8=1   2/9=0.894   2/10=0.671   2/11=0.447   2/12
3/4=0.671   3/5=0.516   3/6=0.447   3/7=0   3/8=0.894   3/9=1   3/10=0.6   3/11=0.4   3/12=0   3/13=0.2
4/5=0.577   4/6=0.5   4/7=0   4/8=0.75   4/9=0.671   4/10=0.894   4/11=0.447   4/12=0   4/13=0.289   4/
5/6=0.866   5/7=0   5/8=0.577   5/9=0.516   5/10=0.516   5/11=0.775   5/12=0   5/13=0.333   5/14=0.289
6/7=0   6/8=0.5   6/9=0.447   6/10=0.447   6/11=0.894   6/12=0   6/13=0.289   6/14=0.25   6/15=0.25   6
7/8=0   7/9=0   7/10=0   7/11=0   7/12=0   7/13=0   7/14=0   7/15=0   7/16=0   7/17=0   7/18=0   7/19=0
8/9=0.894   8/10=0.671   8/11=0.447   8/12=0   8/13=0.289   8/14=0.25   8/15=0.25   8/16=0.189   8/17=0
9/10=0.6   9/11=0.4   9/12=0   9/13=0.258   9/14=0.224   9/15=0.224   9/16=0.169   9/17=0.2   9/18=0
10/11=0.6   10/12=0   10/13=0.258   10/14=0.224   10/15=0.224   10/16=0.169   10/17=0.2   10/18=0   10/
11/12=0   11/13=0.258   11/14=0.224   11/15=0.224   11/16=0.169   11/17=0.2   11/18=0   11/19=0.258   1
12/13=0   12/14=0   12/15=0   12/16=0   12/17=0   12/18=0   12/19=0   12/20=0   12/21=0   12/22=0   12/
13/14=0.866   13/15=0.866   13/16=0.655   13/17=0.516   13/18=0   13/19=1   13/20=0.866   13/21=0.667
14/15=0.75   14/16=0.567   14/17=0.447   14/18=0   14/19=0.866   14/20=1   14/21=0.577   14/22=0.5   14
15/16=0.567   15/17=0.447   15/18=0   15/19=0.866   15/20=0.75   15/21=0.577   15/22=0.5   15/23=0
16/17=0.676   16/18=0   16/19=0.655   16/20=0.567   16/21=0.873   16/22=0.756   16/23=0
17/18=0   17/19=0.516   17/20=0.447   17/21=0.775   17/22=0.894   17/23=0
18/19=0   18/20=0   18/21=0   18/22=0   18/23=0
19/20=0.866   19/21=0.667   19/22=0.577   19/23=0
20/21=0.577   20/22=0.5   20/23=0
21/22=0.866   21/23=0
22/23=0
```

2021/5/21 13

4．類似したメソッド集合の視覚的表示

4.1 Vis.jsについて

- vis.js という可視化ライブラリに含まれるグラフ可視化ライブラリである。Apache License 2.0 / MIT Licenseであり無料で使える。
- Vis Networkは様々なグラフを描画することができる。
- JavaScriptで書かれているため、ブラウザで表示でき、インタラクティブな操作も可能である。



4.2 Vis.js ネットワークグラフの生成

```
<!doctype html>
<html>
<head>
<title>Java Program Network</title>
<style type="text/css">
#mynetwork {
    width: 1200px;
    height: 800px;
    border: 1px solid lightgray;
}
</style>
<script type="text/javascript" src="../vis.js"></script>
<link href="../vis-network.css" type="text/css" />
<script type="text/javascript">
function draw() {
    <nodes の定義>
    <edges の定義>
    var container = document.getElementById
    var data = {
    nodes: nodes,
    edges: edges
};
var options = {};
var network = new vis.Network(container, data, options);
}
</script>
</head>
<body onload="draw()">
<div id="mynetwork"></div>
</body>
</html>
```

Vis.js をインストールしたパスに変更する。

表示したい Node edge を定められたフォーマットで書き込む。

15

変数nodesとedgesの仕様

```
var nodes = [
{id:<番号>, label: '<ラベル名>', shape: '<形状>'},
. . . ];
var edges = [
{from: <番号>, to: <番号>, color: '<色>',
  width:<幅>, length:<長さ>, label:'<類似度>'},
... ];
```

2021/5/21

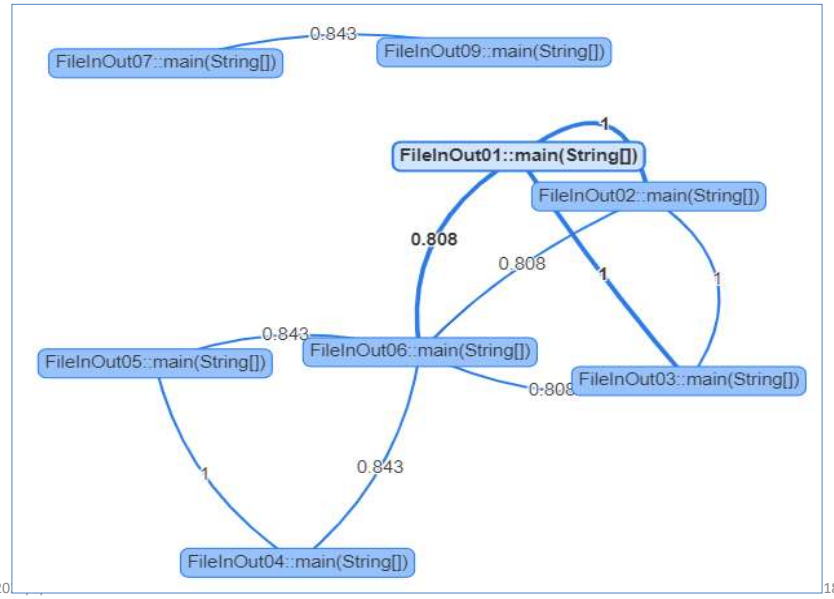
16

変数nodesとedgesの生成例

```
var nodes = [
{id:1, label: 'FileInOut01::main(String[])', shape: 'box'},
{id:2, label: 'FileInOut02::main(String[])', shape: 'box'},
{id:3, label: 'FileInOut03::main(String[])', shape: 'box'},
{id:4, label: 'FileInOut06::main(String[])', shape: 'box'},
{id:5, label: 'FileInOut04::main(String[])', shape: 'box'},
{id:6, label: 'FileInOut05::main(String[])', shape: 'box'},
{id:7, label: 'FileInOut07::main(String[])', shape: 'box'},
{id:8, label: 'FileInOut09::main(String[])', shape: 'box'}
];
var edges = [
{from: 1, to: 2, color: 'red', width: 2, length: 200, label:'1'},
{from: 1, to: 3, color: 'red', width: 2, length: 200, label:'1'},
{from: 1, to: 4, color: 'red', width: 2, length: 200, label:'0.808'},
{from: 2, to: 3, color: 'red', width: 2, length: 200, label:'1'},
{from: 2, to: 4, color: 'red', width: 2, length: 200, label:'0.808'},
{from: 3, to: 4, color: 'red', width: 2, length: 200, label:'0.808'},
{from: 5, to: 6, color: 'red', width: 2, length: 200, label:'1'},
{from: 5, to: 4, color: 'red', width: 2, length: 200, label:'0.843'},
{from: 6, to: 4, color: 'red', width: 2, length: 200, label:'0.843'},
{from: 7, to: 8, color: 'red', width: 2, length: 200, label:'0.843'}
];
```

2021/5/21 17

Vis.jsによるグラフの表示例



20 18

5. 頻出するメソッド集合の抽出

5.1 Aprioriアルゴリズムと極大頻出集合

- 頻出するデータを見つけ出す問題は，大量のデータから有用なパターンを見つけるための基本手法。
- Aprioriアルゴリズムは、典型的なもの。
- 頻出データは大量に検出されることが知られており，その中から有意な頻出系列を検出する技法として、極大頻出集合がある。
- 本研究では、呼び出されているメソッド名の極大頻出集合を計算することで、プログラミングのテーマを見つけることを試みた。

極大頻出集合

timeMeasure
FileInputStream close read toHexString
BufferedInputStream FileInputStream close read toHexString
FileInputStream InputStreamReader close read toChars
BufferedReader FileReader close read toChars
currentTimeMillis forName getMethod invoke

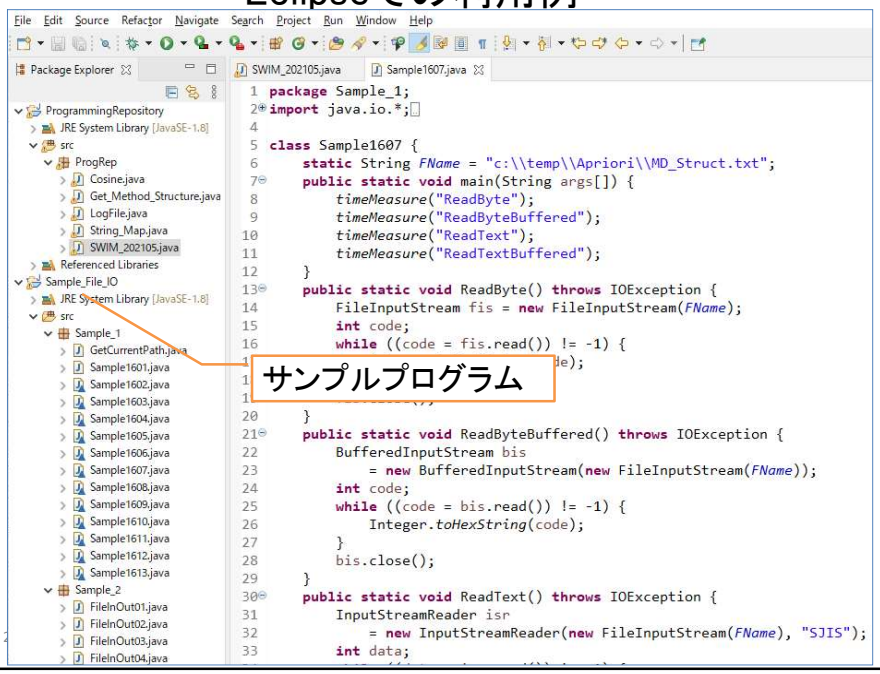
```
1  close (66.6667)
2  read close (66.6667)
3  read (66.6667)
4  FileInputStream read close (50)
5  FileInputStream read (50)
6  FileInputStream close (50)
7  FileInputStream (50)
8  toHexString FileInputStream read close (33.3333)
9  toHexString FileInputStream read (33.3333)
10 toHexString FileInputStream close (33.3333)
11 toHexString FileInputStream (33.3333)
12 toHexString read close (33.3333)
13 toHexString read (33.3333)
14 toHexString close (33.3333)
15 toHexString (33.3333)
16 toChars close read (33.3333)
17 toChars close (33.3333)
18 toChars read (33.3333)
19 toChars (33.3333)
```

支持度30% 以上で抽出した
極大頻出アイテム集合（データ）

```
1 toHexString FileInputStream read close (33.3333)
2 toChars close read (33.3333)
```

支持度30% 以上で抽出した
頻出アイテム集合（データ）

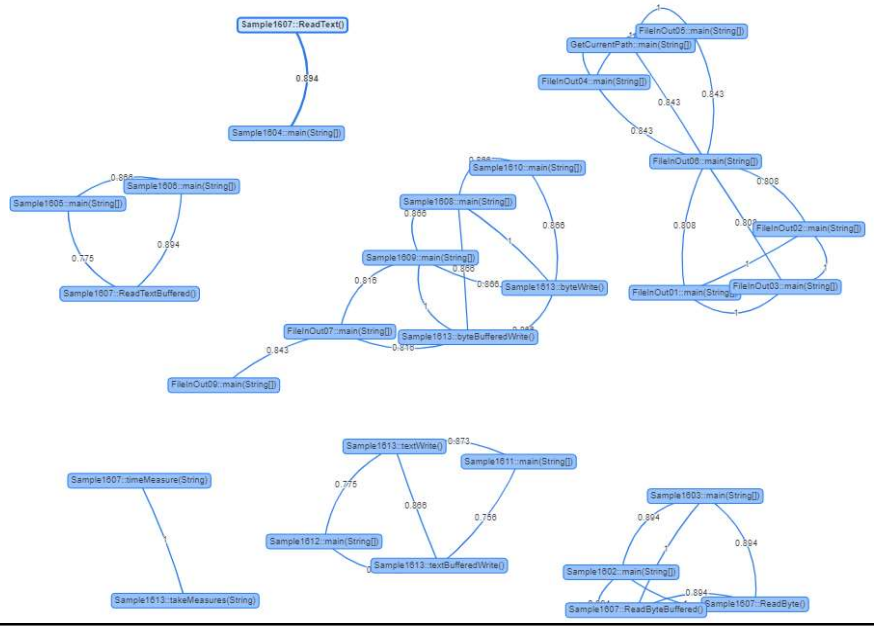
Eclipseでの利用例




```
1  package Sample_1;
2  import java.io.*;
4
5  class Sample1607 {
6      static String FName = "c:\\temp\\Apriori\\MD_Struct.txt";
7      public static void main(String args[]) {
8          timeMeasure("ReadByte");
9          timeMeasure("ReadByteBuffered");
10         timeMeasure("ReadText");
11         timeMeasure("ReadTextBuffered");
12     }
13     public static void ReadByte() throws IOException {
14         FileInputStream fis = new FileInputStream(FName);
15         int code;
16         while ((code = fis.read()) != -1) {
17             de);
18
19     }
20     }
21     public static void ReadByteBuffered() throws IOException {
22         BufferedInputStream bis
23             = new BufferedInputStream(new FileInputStream(FName));
24         int code;
25         while ((code = bis.read()) != -1) {
26             Integer.toHexString(code);
27         }
28         bis.close();
29     }
30     public static void ReadText() throws IOException {
31         InputStreamReader isr
32             = new InputStreamReader(new FileInputStream(FName), "SJIS");
33         int data;
```

File IOに関するサンプルプログラムの分析例

（類似度75%以上を抽出しグラフ表示したもの）



6. おわりに

1. サンプルプログラムの利用を推進するための類似検索機能とグラフ表示方法を提案した。
2. 加えて、頻出するメソッドを抽出することで，優先的に学習すべきメソッド集合を導出することを試みた。
3. プログラムの構造解析により、呼び出しているメソッド名と回数を抽出し、Cos類似度でメソッド同士の類似度を計算した。

- 今後は、実用的なサンプルプログラムを使った実験を行う。
- また、教育現場への適用を計画している。

23