Mining Patterns in Source Code using Tree Mining Algorithms

Hoang Son Pham[1], Siegfried Nijssen[1], Kim Mens[1],
Dario Di Nucci[2], Tim Molderz[2], Coen De Roover[2],
Johan Fabry[3], and Vadim Zaytsev[3]

[1] ICTEAM, UCLouvain, Belgium
[2] Software Languages Lab, VUB, Belgium
[3] Raincode Labs, Belgium

AIA meeting, 8th May, 2019

Outline

- Introduction
- Frequent tree mining
- Problem statement
- Constraint-based maximal tree mining
- Experimental results
- Conclusion and future work

Introduction

- Maintain and develop big software projects are complicated tasks.
- Software engineers need intelligent support systems
- Many design and code conventions get encoded in source code:
  - Coding idioms
  - Usage protocols
- Software engineers can obtain valuable information from these regularities

Introduction

- Examples of patterns in source code:

  1. Coding idioms:

  - Programming language structures, i.e., for-statement

```
for (Figure figure : selectedFigures) {…}
```

  - Code repeated among different classes

```
protected void updateEnabledState() {
  if (getView() != null) {
    setEnabled(...);
  } else {
    setEnabled(false);
  }
}
```

Introduction

- Examples of patterns in source code:

2. Usage protocols

- Methods always appear paired:

```
f.fileOpen();
   ...
f.fileClose();
```

```
figure.willChange();
   …
figure.changed();
```

Introduction

- Examples of patterns in source code:

2. Usage protocols

- A set of methods implemented on several classes

```
UndoableEdit edit = new AbstractUndoableEdit() {
    @Override
    public String getPresentationName() {
        ...
    }
    @Override
    public void undo() {
        super.undo();
        Iterator<Object> iRestore = restoreData.iterator();
        ...
    }
    @Override
    public void redo() {
        super.redo();
    ...
    }
};
```

Introduction

- Applications:
  - to help software engineers to understand, analyze, maintain and improve the system
  - to build code recommendation system or code completion tool

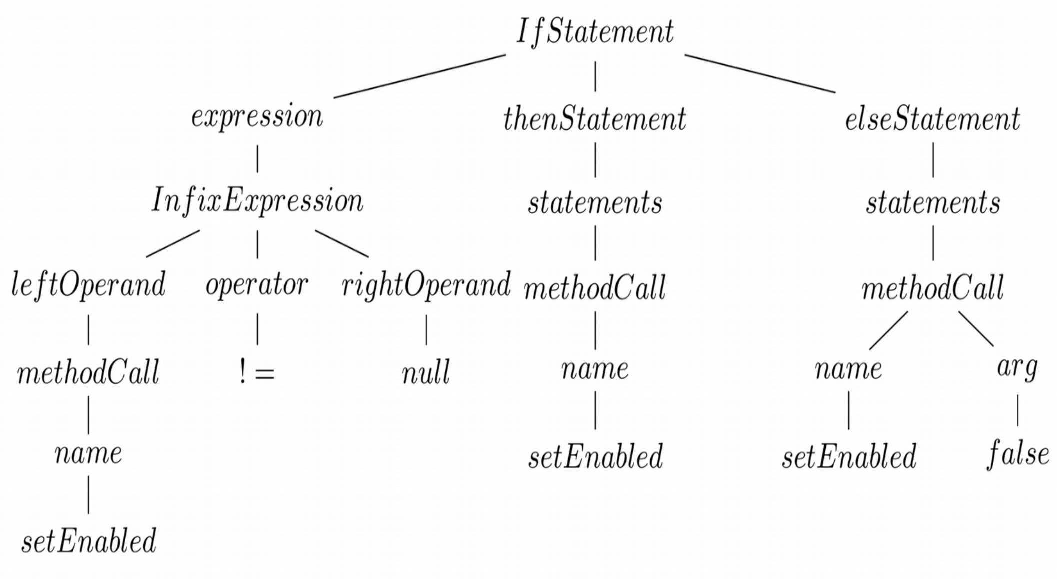
Introduction

- Source code are represented by Abstract Syntax Trees (ASTs).
- Source code patterns correspond to fragments of trees that occur frequently
- **Discover source code patterns by using frequent tree mining algorithms**

Source code pattern

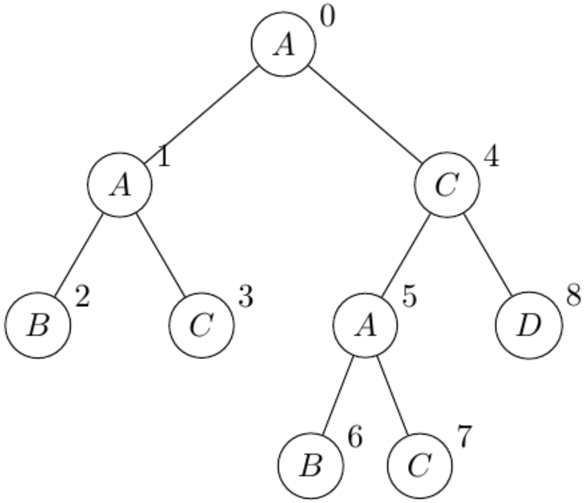
```
protected void updateEnabledState() {
    if (getView() != null) {
        setEnabled(...);
    } else {
        setEnabled(false);
    }
}
```

Tree representation

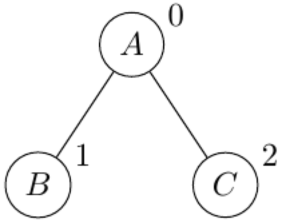


Frequent tree mining

- Tree data: labeled, ordered, rooted trees
- Matching ordered tree: induced subtree
- Support of subtree: the number of trees in a database in which the subtree occurs
- Frequent tree mining: discover all frequent subtrees in a given tree data.



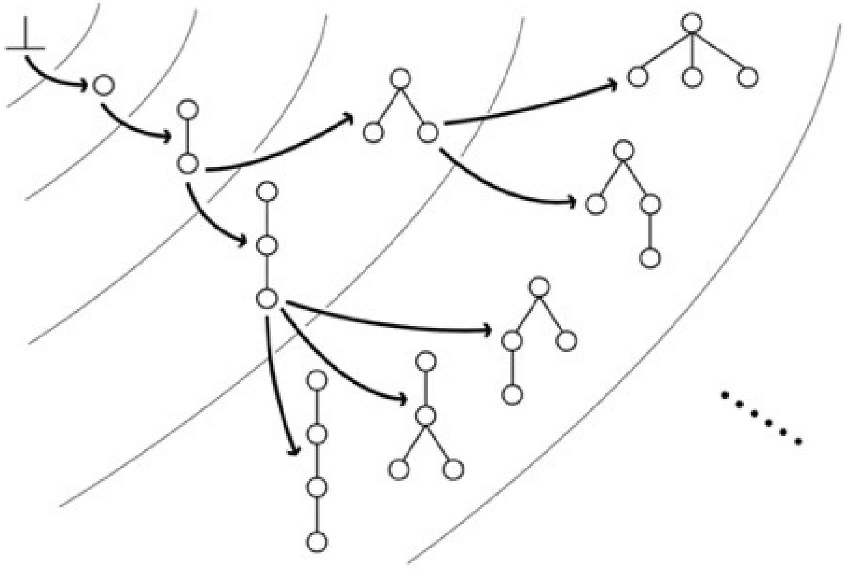
Tree data



Induced subtree

Frequent tree mining

- FREQT
  - Search patterns: depth-first search
  - Grow patterns: rightmost path extension
  - Prune: minimum support



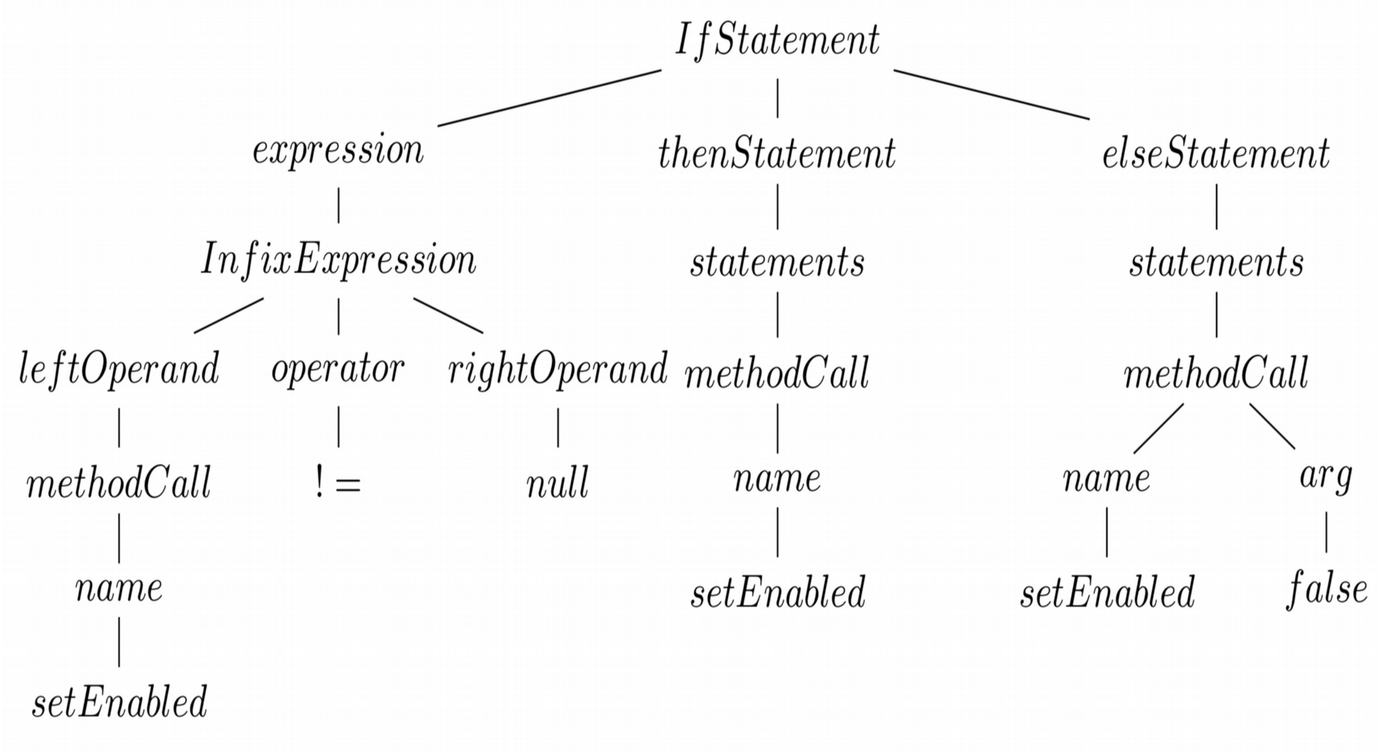
Problem:

- not scale for source code mining

- produces a large amount of useless patterns

Problem statement

- Interesting patterns in source code:
  - Reflect structure of programming language
  - The total number of nodes/leafs in a pattern is sufficiently large



Problem statement

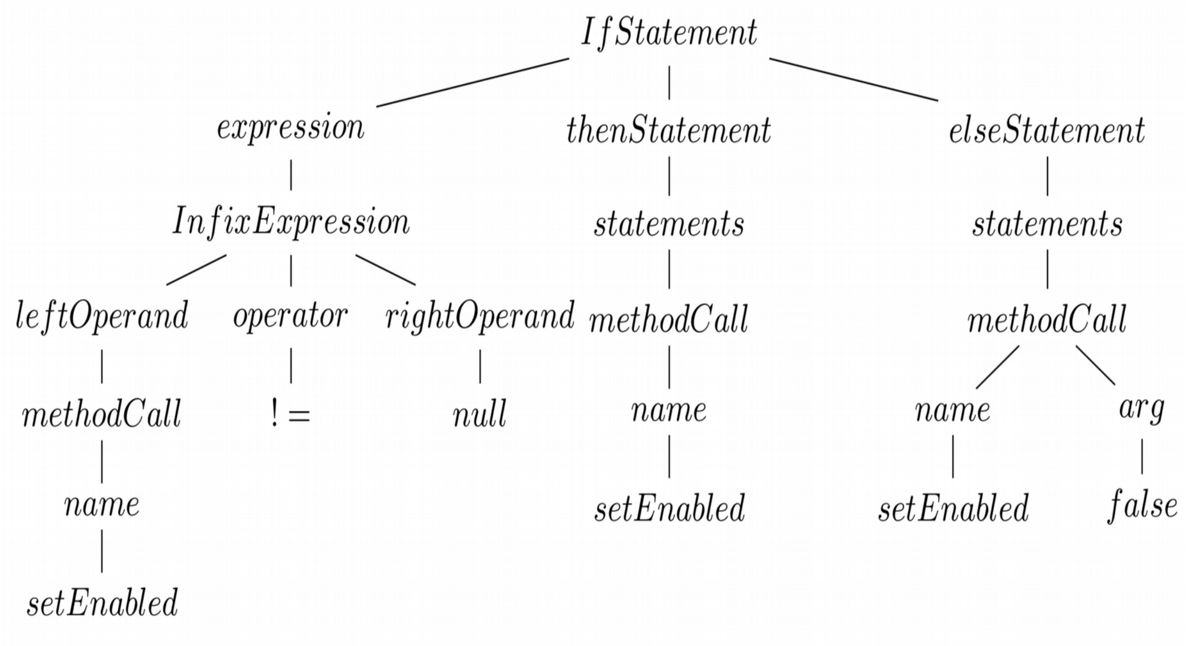
- Combine constraint-based data mining and maximal frequent tree mining algorithm to
  - find patterns with size as large as possible
  - reduce number of output patterns
  - reduce execution time

Constraint-based maximal tree mining

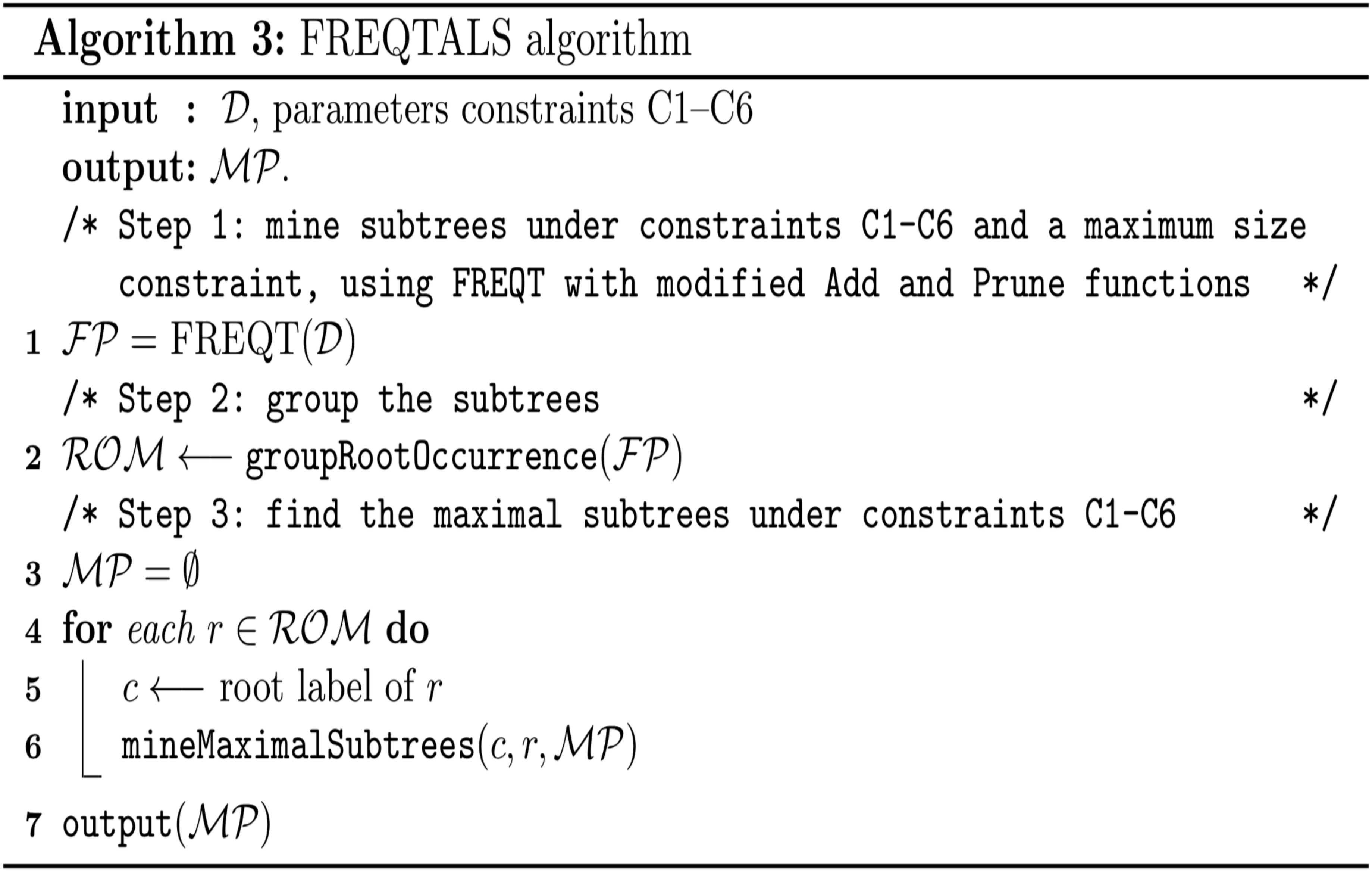
- 1: Minimum size constraints
  - C1: the number of leafs in a pattern is sufficiently large
  - C2: the total number of nodes in a pattern is sufficiently large.
- 2: Constraints on labels
  - C3: limit the set of labels allowed to occur in the root of patterns;
  - C4: provide labels forbidden from occurring in the pattern;
  - C5: limit the number of siblings in a pattern that can have the same label

Constraint-based maximal tree mining

- 3: Constraint on leafs
  - C6: All leaf nodes in a pattern must have a label that is included in leafs of data.
- 4: Obligatory children
  - Given a node, some of its children can be mandatory
  - Avoid to produce patterns that missing mandatory labels



FREQTALS algorithm

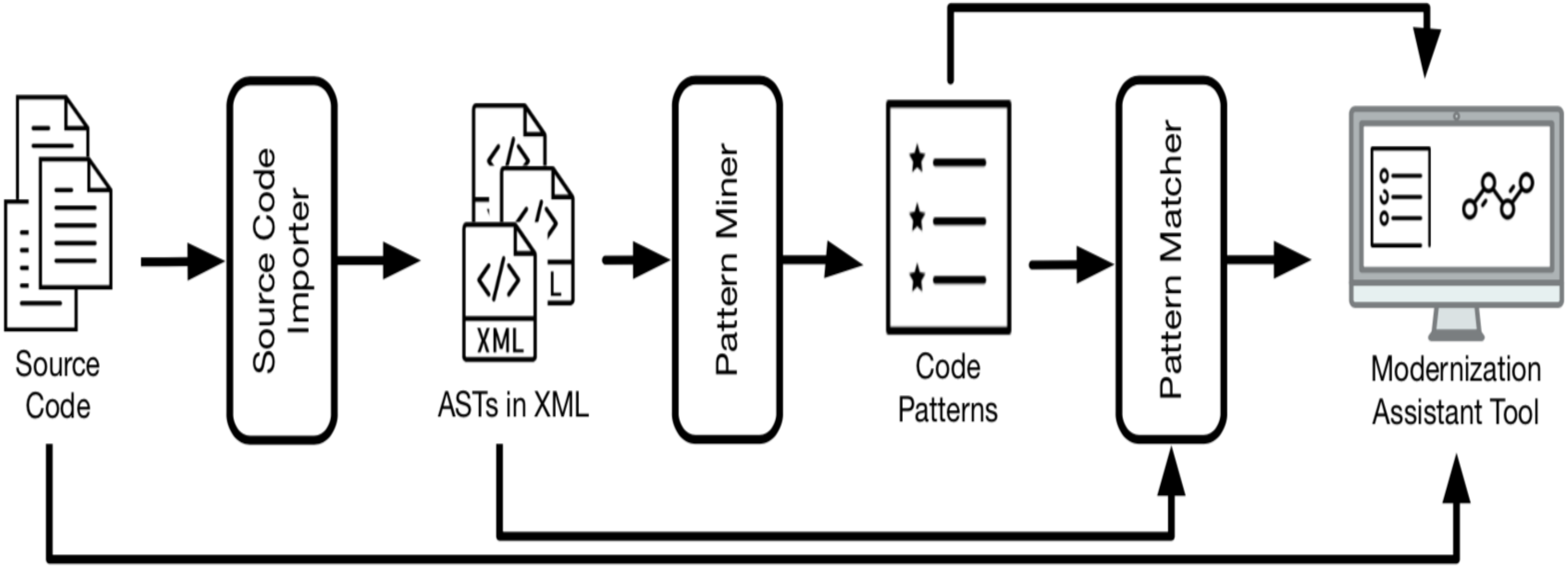


Algorithm 3: FREQTALS algorithm

```
input  : $\mathcal{D}$, parameters constraints C1–C6
output: $\mathcal{MP}$.
/* Step 1: mine subtrees under constraints C1-C6 and a maximum size
   constraint, using FREQT with modified Add and Prune functions   */
$\mathcal{FP}$ = FREQT($\mathcal{D}$)
/* Step 2: group the subtrees                                     */
$\mathcal{ROM} \longleftarrow$ groupRootOccurrence($\mathcal{FP}$)
/* Step 3: find the maximal subtrees under constraints C1-C6      */
$\mathcal{MP} = \emptyset$
for each $r \in \mathcal{ROM}$ do
    $c \longleftarrow$ root label of $r$
    mineMaximalSubtrees($c, r, \mathcal{MP}$)
output($\mathcal{MP}$)
```

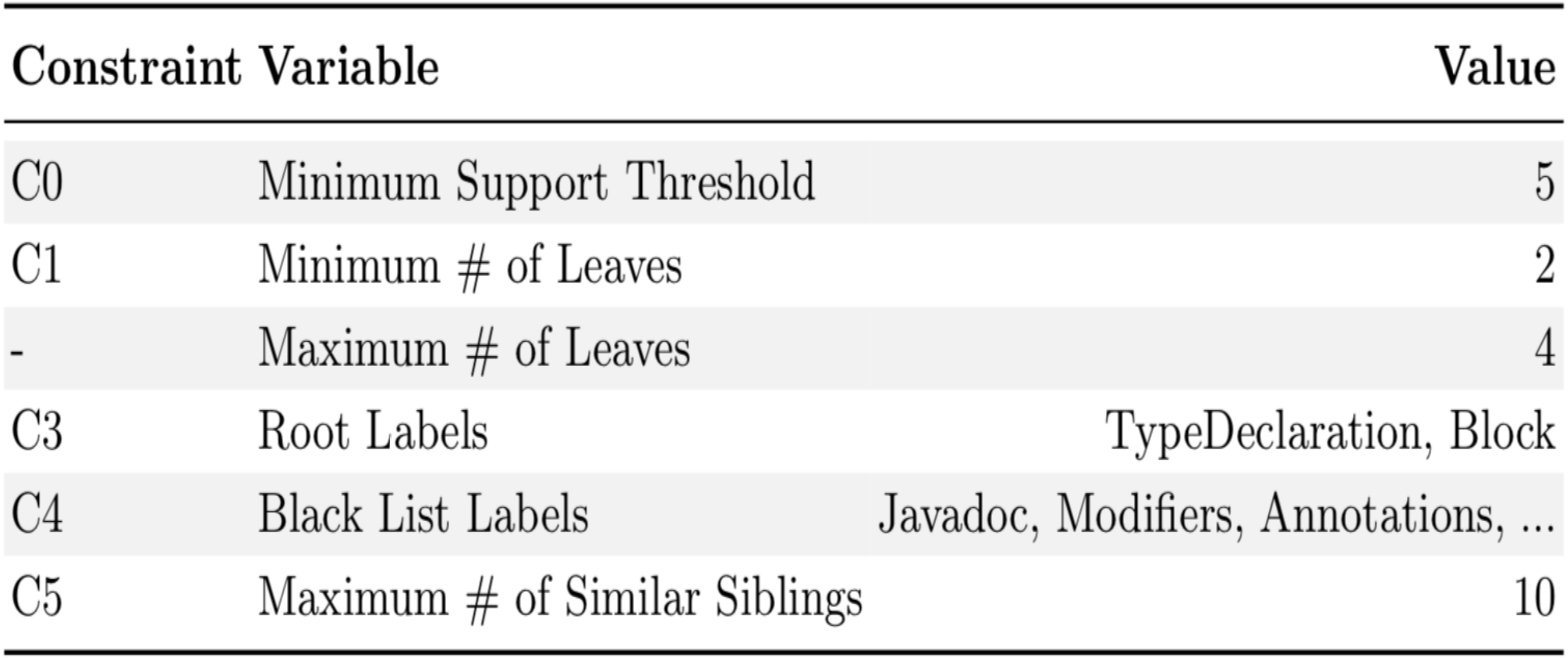
Experiments

- The modular language-parametric framework for mining code regularities



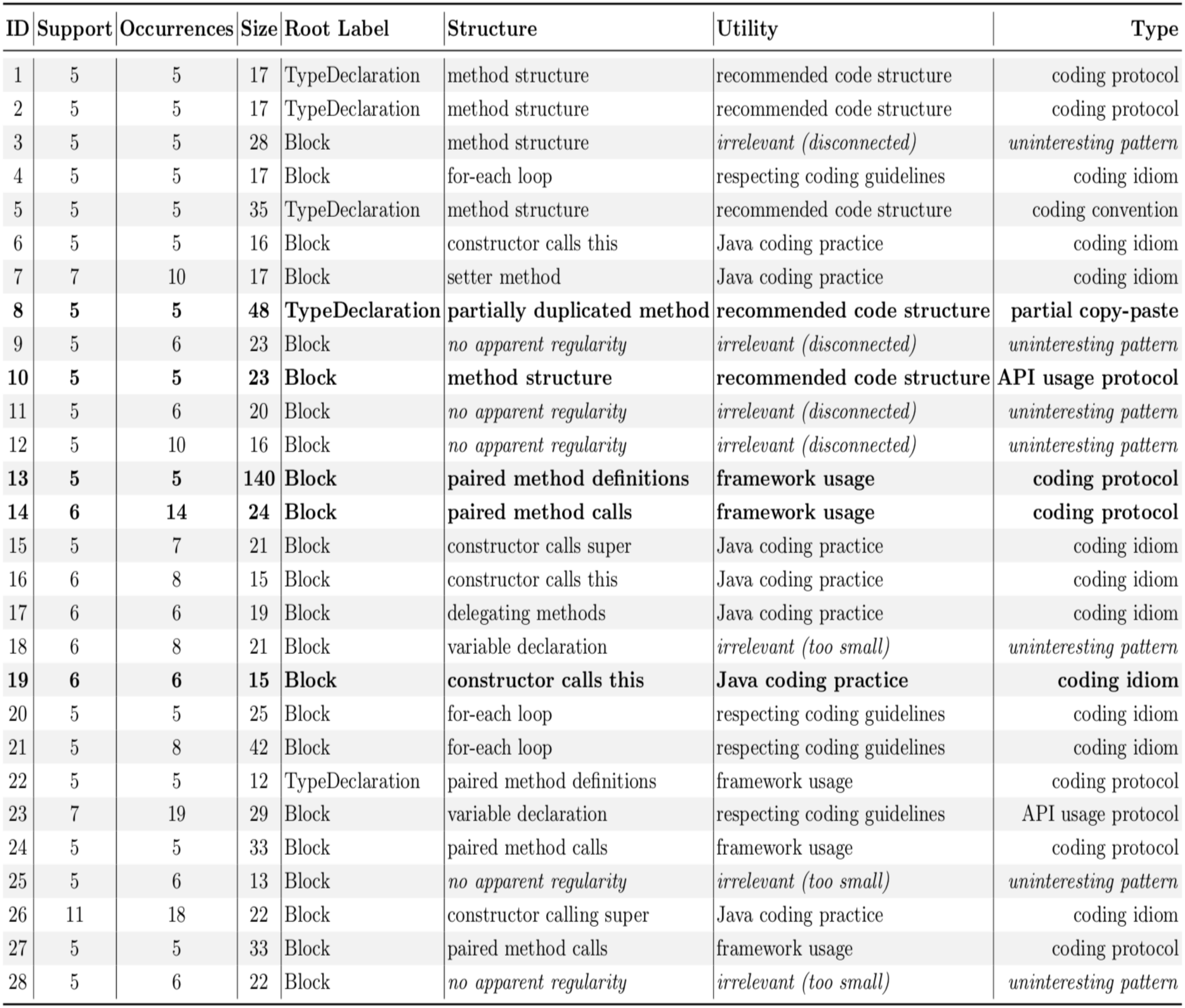
Experiments

- Dataset: Java source code (Jhotdraw project)
- Configurations:



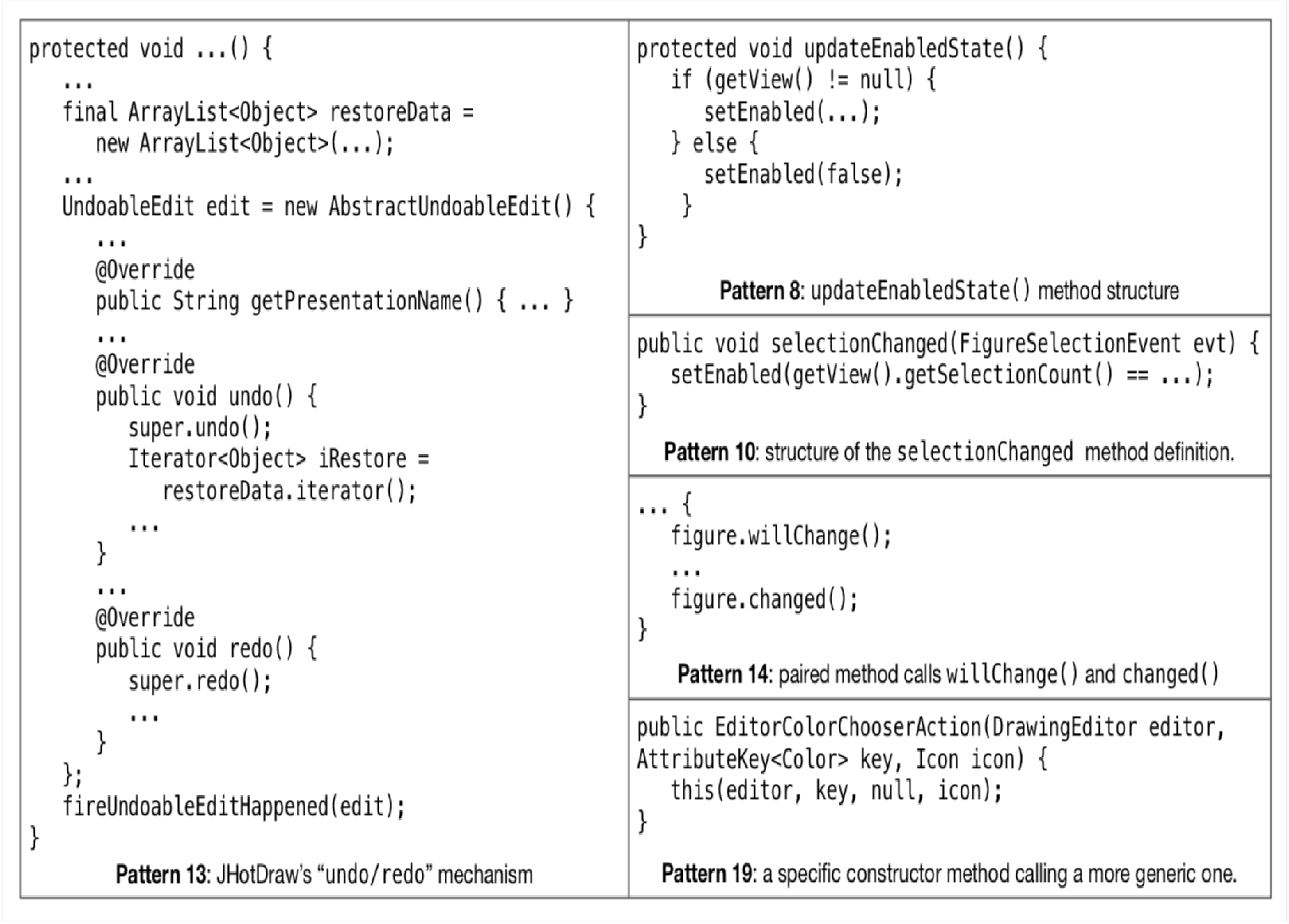
Constraint	Variable	Value
C0	Minimum Support Threshold	5
C1	Minimum # of Leaves	2
-	Maximum # of Leaves	4
C3	Root Labels	TypeDeclaration, Block
C4	Black List Labels	Javadoc, Modifiers, Annotations, ...
C5	Maximum # of Similar Siblings	10

Qualitative Analysis



ID	Support	Occurrences	Size	Root Label	Structure	Utility	Type
1	5	5	17	TypeDeclaration	method structure	recommended code structure	coding protocol
2	5	5	17	TypeDeclaration	method structure	recommended code structure	coding protocol
3	5	5	28	Block	method structure	*irrelevant (disconnected)*	*uninteresting pattern*
4	5	5	17	Block	for-each loop	respecting coding guidelines	coding idiom
5	5	5	35	TypeDeclaration	method structure	recommended code structure	coding convention
6	5	5	16	Block	constructor calls this	Java coding practice	coding idiom
7	7	10	17	Block	setter method	Java coding practice	coding idiom
8	**5**	**5**	**48**	**TypeDeclaration**	**partially duplicated method**	**recommended code structure**	**partial copy-paste**
9	5	6	23	Block	*no apparent regularity*	*irrelevant (disconnected)*	*uninteresting pattern*
10	**5**	**5**	**23**	**Block**	**method structure**	**recommended code structure**	**API usage protocol**
11	5	6	20	Block	*no apparent regularity*	*irrelevant (disconnected)*	*uninteresting pattern*
12	5	10	16	Block	*no apparent regularity*	*irrelevant (disconnected)*	*uninteresting pattern*
13	**5**	**5**	**140**	**Block**	**paired method definitions**	**framework usage**	**coding protocol**
14	**6**	**14**	**24**	**Block**	**paired method calls**	**framework usage**	**coding protocol**
15	5	7	21	Block	constructor calls super	Java coding practice	coding idiom
16	6	8	15	Block	constructor calls this	Java coding practice	coding idiom
17	6	6	19	Block	delegating methods	Java coding practice	coding idiom
18	6	8	21	Block	variable declaration	*irrelevant (too small)*	*uninteresting pattern*
19	**6**	**6**	**15**	**Block**	**constructor calls this**	**Java coding practice**	**coding idiom**
20	5	5	25	Block	for-each loop	respecting coding guidelines	coding idiom
21	5	8	42	Block	for-each loop	respecting coding guidelines	coding idiom
22	5	5	12	TypeDeclaration	paired method definitions	framework usage	coding protocol
23	7	19	29	Block	variable declaration	respecting coding guidelines	API usage protocol
24	5	5	33	Block	paired method calls	framework usage	coding protocol
25	5	6	13	Block	*no apparent regularity*	*irrelevant (too small)*	*uninteresting pattern*
26	11	18	22	Block	constructor calling super	Java coding practice	coding idiom
27	5	5	33	Block	paired method calls	framework usage	coding protocol
28	5	6	22	Block	*no apparent regularity*	*irrelevant (too small)*	*uninteresting pattern*

Qualitative Analysis



```
protected void ...() {
   ...
   final ArrayList<Object> restoreData =
      new ArrayList<Object>(...);
   ...
   UndoableEdit edit = new AbstractUndoableEdit() {
      ...
      @Override
      public String getPresentationName() { ... }
      ...
      @Override
      public void undo() {
         super.undo();
         Iterator<Object> iRestore =
            restoreData.iterator();
         ...
      }
      ...
      @Override
      public void redo() {
         super.redo();
         ...
      }
   };
   fireUndoableEditHappened(edit);
}
```

Pattern 13: JHotDraw's “undo/redo” mechanism

```
protected void updateEnabledState() {
   if (getView() != null) {
      setEnabled(...);
   } else {
      setEnabled(false);
    }
}
```

Pattern 8: updateEnabledState() method structure

```
public void selectionChanged(FigureSelectionEvent evt) {
   setEnabled(getView().getSelectionCount() == ...);
}
```

Pattern 10: structure of the selectionChanged method definition.

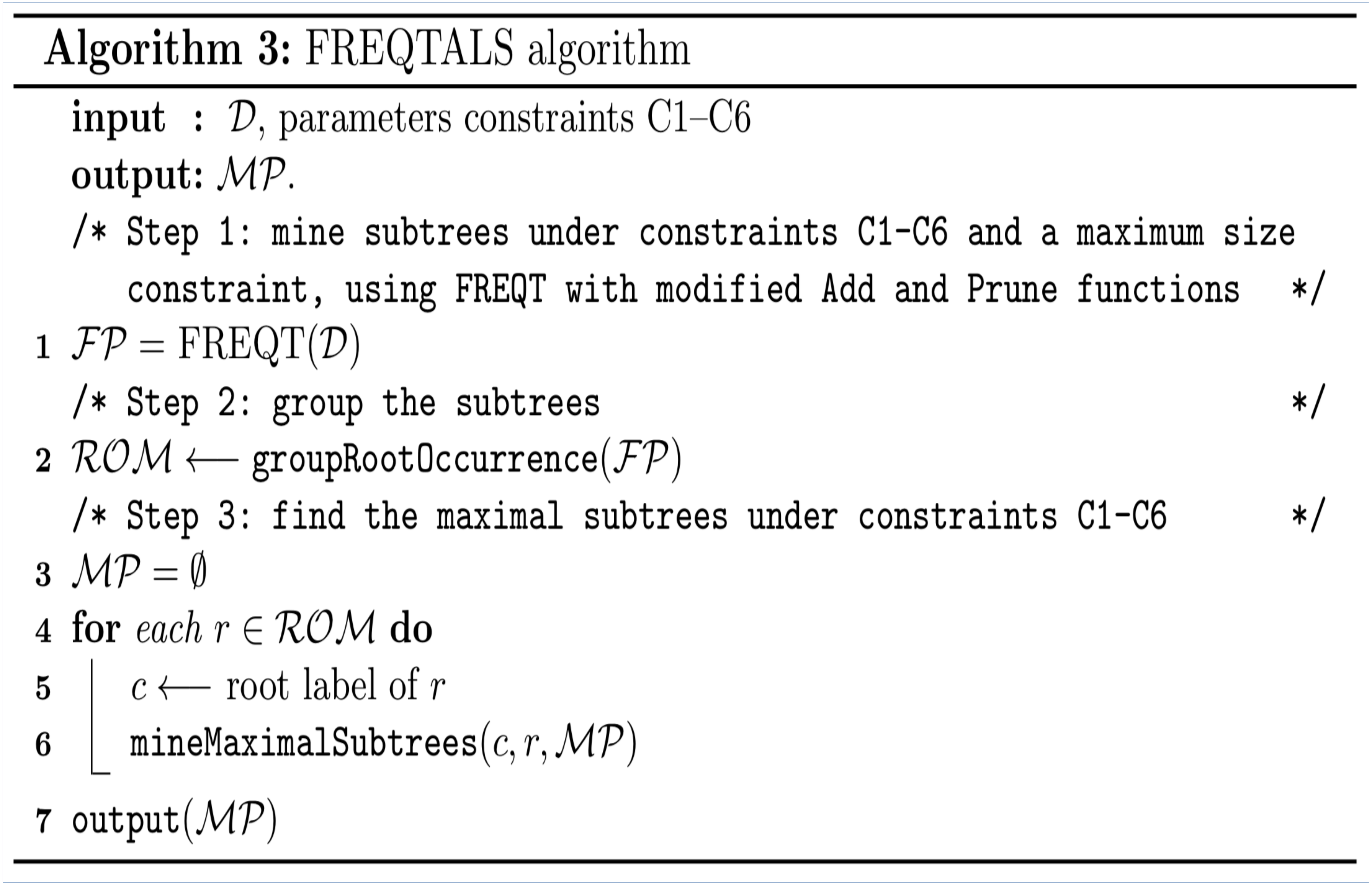
```
... {
   figure.willChange();
   ...
   figure.changed();
}
```

Pattern 14: paired method calls willChange() and changed()

```
public EditorColorChooserAction(DrawingEditor editor,
AttributeKey<Color> key, Icon icon) {
   this(editor, key, null, icon);
}
```

Pattern 19: a specific constructor method calling a more generic one.

Comparison between Original FREQT and FREQTALS



Algorithm 3: FREQTALS algorithm

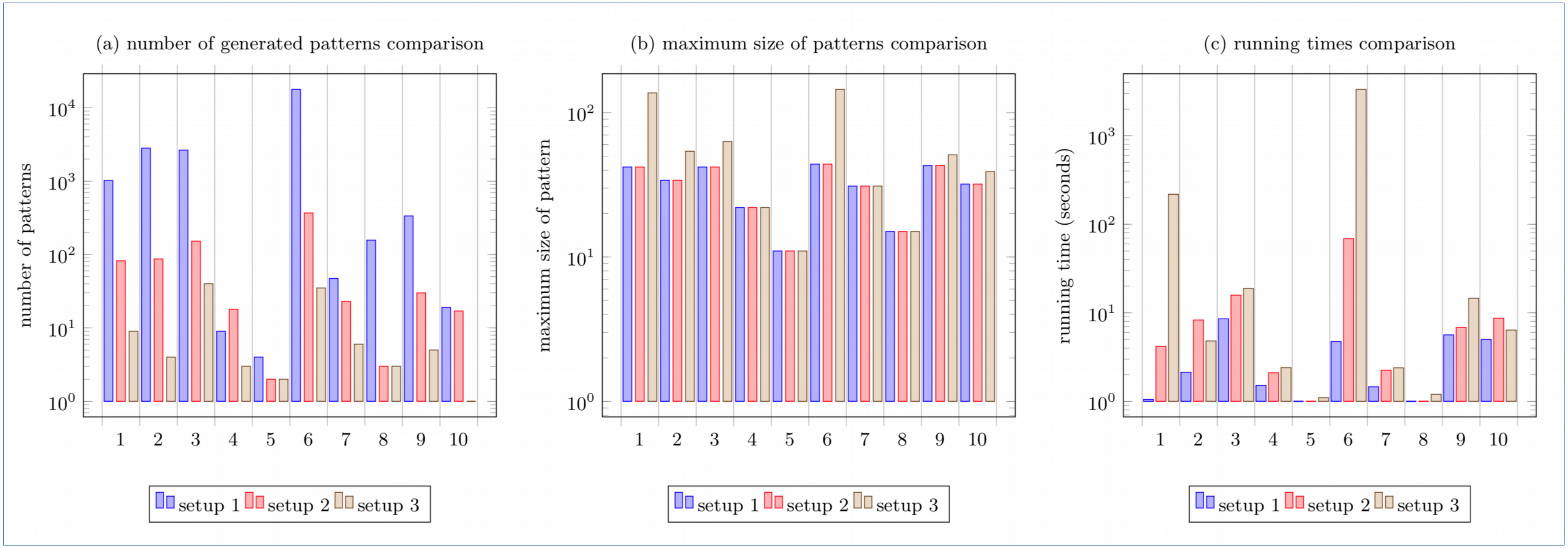
```
input  : $\mathcal{D}$, parameters constraints C1–C6
output: $\mathcal{MP}$.
/* Step 1: mine subtrees under constraints C1-C6 and a maximum size
   constraint, using FREQT with modified Add and Prune functions   */
$\mathcal{FP} = \mathrm{FREQT}(\mathcal{D})$
/* Step 2: group the subtrees                                     */
$\mathcal{ROM} \longleftarrow$ groupRootOccurrence($\mathcal{FP}$)
/* Step 3: find the maximal subtrees under constraints C1-C6      */
$\mathcal{MP} = \emptyset$
for each $r \in \mathcal{ROM}$ do
    $c \longleftarrow$ root label of $r$
    mineMaximalSubtrees($c, r, \mathcal{MP}$)
output($\mathcal{MP}$)
```

Comparison between Original FREQT and FREQTALS

Algorithm	**Frequent Patterns**	**Running time**
FREQTALS with extensions	1,280	2.21 seconds
Original FREQT with *minsup* constraint	42,430	Timeout (5 minutes)

Evaluate the extensions of FREQTALS

- Setup 1: applies only step 1, with a maximal size constraint
- Setup 2: filters maximal patterns from result of step 1
- Setup 3: applies all steps to find maximal patterns **without** maximal size constraint



Conclusion and future work

- Contributions of FREQTALS algorithm:
  - the discovered patterns highlight relevant code regularities;
  - the patterns found are significantly larger;
  - the execution time and amount of output patterns are significantly smaller for the same *minsup*;
  - the configurations are easily adapted for other programming languages.

Conclusion and future work

- Future work:
  - Employ FREQTALS on legacy programming languages;
  - Explore more details of the quality of patterns;
  - Improve the scalability;
  - Define guidelines to setup configurations.

Thank you very much!