Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Comparison of Clang Abstract Syntax Trees using String Kernels

HPCS 2018, Orleans, France

Raul Torres, Julian M. Kunkel, Manuel F. Dolz, Thomas Ludwig

Agenda

1. Motivation
2. Background
   - Intermediate representations
   - String kernels
3. Proposed solution
   - Creating strings from ASTs
   - Finding similarities with a novel string kernel
4. Evaluation
   - Experiment configuration
   - Blended spectrum kernel
   - Kast spectrum kernel
   - Kast1 spectrum kernel
5. Conclusions and future work

1. Motivation

Code similarity

- Computer programs exhibit similarities that can be detected before, during, and after execution time.

Code similarity

- Computer programs exhibit similarities that can be detected before, during, and after execution time.
- Programs that are similar tend to behave in similar manner too.

Code similarity

- Computer programs exhibit similarities that can be detected before, during, and after execution time.
- Programs that are similar tend to behave in similar manner too.
- Detecting program similarities may have the following applications:

- Computer programs exhibit similarities that can be detected before, during, and after execution time.
- Programs that are similar tend to behave in similar manner too.
- Detecting program similarities may have the following applications:
  - Analyze and improve the overall performance of a set of similar programs.

Code similarity

- Computer programs exhibit similarities that can be detected before, during, and after execution time.
- Programs that are similar tend to behave in similar manner too.
- Detecting program similarities may have the following applications:
  - Analyze and improve the overall performance of a set of similar programs.
  - Assist the programmer in finding code that is already implemented instead of coding from scratch.

Code similarity

- Computer programs exhibit similarities that can be detected before, during, and after execution time.
- Programs that are similar tend to behave in similar manner too.
- Detecting program similarities may have the following applications:
  - Analyze and improve the overall performance of a set of similar programs.
  - Assist the programmer in finding code that is already implemented instead of coding from scratch.
  - Find common mistakes at designing or writing programs: code smells.

- Computer programs exhibit similarities that can be detected before, during, and after execution time.
- Programs that are similar tend to behave in similar manner too.
- Detecting program similarities may have the following applications:
  - Analyze and improve the overall performance of a set of similar programs.
  - Assist the programmer in finding code that is already implemented instead of coding from scratch.
  - Find common mistakes at designing or writing programs: code smells.
  - Detect plagiarism.

The similarity between two programs can be determined by the amount of clones they share. There are four distinct types of them:

Code clones

The similarity between two programs can be determined by the amount of clones they share. There are four distinct types of them:

- *Type-1:* this type of clones stand for pieces of code containing differences in the layout, spaces and comments.

Code clones

The similarity between two programs can be determined by the amount of clones they share. There are four distinct types of them:

- *Type-1:* this type of clones stand for pieces of code containing differences in the layout, spaces and comments.
- *Type-2:* these codes also present differences in data types and identifiers.

Code clones

The similarity between two programs can be determined by the amount of clones they share. There are four distinct types of them:

- *Type-1:* this type of clones stand for pieces of code containing differences in the layout, spaces and comments.
- *Type-2:* these codes also present differences in data types and identifiers.
- *Type-3:* these clone types also include additions, modifications and deletions of lines of code.

Code clones

The similarity between two programs can be determined by the amount of clones they share. There are four distinct types of them:

- *Type-1:* this type of clones stand for pieces of code containing differences in the layout, spaces and comments.
- *Type-2:* these codes also present differences in data types and identifiers.
- *Type-3:* these clone types also include additions, modifications and deletions of lines of code.
- *Type-4:* represent codes that present different implementation but the same functionality.

Code clones

The similarity between two programs can be determined by the amount of clones they share. There are four distinct types of them:

- *Type-1:* this type of clones stand for pieces of code containing differences in the layout, spaces and comments.
- *Type-2:* these codes also present differences in data types and identifiers.
- *Type-3:* these clone types also include additions, modifications and deletions of lines of code.
- *Type-4:* represent codes that present different implementation but the same functionality.

Code clones

The similarity between two programs can be determined by the amount of clones they share. There are four distinct types of them:

- *Type-1:* this type of clones stand for pieces of code containing differences in the layout, spaces and comments.
- *Type-2:* these codes also present differences in data types and identifiers.
- *Type-3:* these clone types also include additions, modifications and deletions of lines of code.
- *Type-4:* represent codes that present different implementation but the same functionality.

What to compare? source code? intermediate representations? binary code? I/O access patterns?

2. Background

Generalities

- The IR of a program is the central data structure in a compiler. Around it, analysis, transformations and optimizations are performed.

- The IR of a program is the central data structure in a compiler. Around it, analysis, transformations and optimizations are performed.
- Broad categories:

- The IR of a program is the central data structure in a compiler. Around it, analysis, transformations and optimizations are performed.
- Broad categories:
  - *Graphical IRs*: This IR type stores the program information in a graph-like data structure.

Generalities

- The IR of a program is the central data structure in a compiler. Around it, analysis, transformations and optimizations are performed.
- Broad categories:
  - *Graphical IRs*: This IR type stores the program information in a graph-like data structure.
  - *Linear IRs*: This IR type makes use of simple linear sequences to store operations, similar to machine code.

- The IR of a program is the central data structure in a compiler. Around it, analysis, transformations and optimizations are performed.
- Broad categories:
  - *Graphical IRs*: This IR type stores the program information in a graph-like data structure.
  - *Linear IRs*: This IR type makes use of simple linear sequences to store operations, similar to machine code.
  - *Hybrid IRs*: Hybrid IRs combine elements of the previous two categories.

- The IR of a program is the central data structure in a compiler. Around it, analysis, transformations and optimizations are performed.
- Broad categories:
  - *Graphical IRs*: This IR type stores the program information in a graph-like data structure.
  - *Linear IRs*: This IR type makes use of simple linear sequences to store operations, similar to machine code.
  - *Hybrid IRs*: Hybrid IRs combine elements of the previous two categories.
- Complex compiler infrastructures might work with different interconnected IRs, some of them closer to the source code, others closer to the machine instruction level.

Abstract syntax trees (ASTs)

- ASTs are a graph-based intermediate representation.

Abstract syntax trees (ASTs)

- ASTs are a graph-based intermediate representation.
- They are defined as contractions of parse trees.

Abstract syntax trees (ASTs)

- ASTs are a graph-based intermediate representation.
- They are defined as contractions of parse trees.
- Most non-terminal symbols are ignored.

Abstract syntax trees (ASTs)

- ASTs are a graph-based intermediate representation.
- They are defined as contractions of parse trees.
- Most non-terminal symbols are ignored.
- The precedence and the meaning of the expressions are preserved.

Abstract syntax trees (ASTs)

- ASTs are a graph-based intermediate representation.
- They are defined as contractions of parse trees.
- Most non-terminal symbols are ignored.
- The precedence and the meaning of the expressions are preserved.
- Their level of abstraction is not far from the original source code.

Abstract syntax trees (ASTs)

- ASTs are a graph-based intermediate representation.
- They are defined as contractions of parse trees.
- Most non-terminal symbols are ignored.
- The precedence and the meaning of the expressions are preserved.
- Their level of abstraction is not far from the original source code.
- In this work we used the AST from Clang, a LLVM frontend for C/C++/Objective C programs.

Abstract syntax trees (ASTs)

- ASTs are a graph-based intermediate representation.
- They are defined as contractions of parse trees.
- Most non-terminal symbols are ignored.
- The precedence and the meaning of the expressions are preserved.
- Their level of abstraction is not far from the original source code.
- In this work we used the AST from Clang, a LLVM frontend for C/C++/Objective C programs.

Abstract syntax trees (ASTs)

- ASTs are a graph-based intermediate representation.
- They are defined as contractions of parse trees.
- Most non-terminal symbols are ignored.
- The precedence and the meaning of the expressions are preserved.
- Their level of abstraction is not far from the original source code.
- In this work we used the AST from Clang, a LLVM frontend for C/C++/Objective C programs.

How to compare these data structures? direct tree comparison? flatten into strings? extract attribute set?

- Strings are a common and useful form of representing data (e.g. DNA sequences).

Generalities

- Strings are a common and useful form of representing data (e.g. DNA sequences).
- String kernels can be intuitively understood as functions measuring the similarity of pairs of strings.

- Strings are a common and useful form of representing data (e.g. DNA sequences).
- String kernels can be intuitively understood as functions measuring the similarity of pairs of strings.
- The more similar two strings A and B are, the higher the value of a string kernel $K(A, B)$ will be.

- Strings are a common and useful form of representing data (e.g. DNA sequences).
- String kernels can be intuitively understood as functions measuring the similarity of pairs of strings.
- The more similar two strings A and B are, the higher the value of a string kernel $K(A, B)$ will be.
- In particular, string kernels check the number of shared substrings among a collection of strings.

Some kernel functions have been proposed:

Some kernel functions have been proposed:

- The bag-of-characters kernel:

Some kernel functions have been proposed:

- The bag-of-characters kernel:
  - Performs single-character matching.

Some kernel functions have been proposed:

- The bag-of-characters kernel:
  - Performs single-character matching.
- The bag-of-words kernel:

Some kernel functions have been proposed:

- The bag-of-characters kernel:
  - Performs single-character matching.
- The bag-of-words kernel:
  - Searches for shared words among strings.

Some kernel functions have been proposed:

- The bag-of-characters kernel:
  - Performs single-character matching.
- The bag-of-words kernel:
  - Searches for shared words among strings.
- The k-spectrum kernel:

Some kernel functions have been proposed:

- The bag-of-characters kernel:
  - Performs single-character matching.
- The bag-of-words kernel:
  - Searches for shared words among strings.
- The k-spectrum kernel:
  - Counts only sub-strings of length k.

Some kernel functions have been proposed:

- The bag-of-characters kernel:
  - Performs single-character matching.
- The bag-of-words kernel:
  - Searches for shared words among strings.
- The k-spectrum kernel:
  - Counts only sub-strings of length k.
- The k-blended spectrum kernel:

Some kernel functions have been proposed:

- The bag-of-characters kernel:
  - Performs single-character matching.
- The bag-of-words kernel:
  - Searches for shared words among strings.
- The *k*-spectrum kernel:
  - Counts only sub-strings of length k.
- The *k*-blended spectrum kernel:
  - It includes all strings whose length is minor than or equal to a given number k.

Some kernel functions have been proposed:

- The bag-of-characters kernel:
  - Performs single-character matching.
- The bag-of-words kernel:
  - Searches for shared words among strings.
- The k-spectrum kernel:
  - Counts only sub-strings of length k.
- The k-blended spectrum kernel:
  - It includes all strings whose length is minor than or equal to a given number k.

Some kernel functions have been proposed:

- The bag-of-characters kernel:
  - Performs single-character matching.
- The bag-of-words kernel:
  - Searches for shared words among strings.
- The k-spectrum kernel:
  - Counts only sub-strings of length k.
- The k-blended spectrum kernel:
  - It includes all strings whose length is minor than or equal to a given number k.

What is our contribution?

3. Proposed solution

Overview

1. Convert the trees into weighted strings.

Overview

1. Convert the trees into weighted strings.
2. Compress the strings in order to save some space.

Overview

1. Convert the trees into weighted strings.
2. Compress the strings in order to save some space.
3. Obtain a similarity matrix a.k.a kernel matrix using a novel string kernel function.

Overview

1. Convert the trees into weighted strings.
2. Compress the strings in order to save some space.
3. Obtain a similarity matrix a.k.a kernel matrix using a novel string kernel function.
4. Use a clustering algorithm to extract knowledge from the kernel matrix.

Overview

1. Convert the trees into weighted strings.
2. Compress the strings in order to save some space.
3. Obtain a similarity matrix a.k.a kernel matrix using a novel string kernel function.
4. Use a clustering algorithm to extract knowledge from the kernel matrix.

Overview

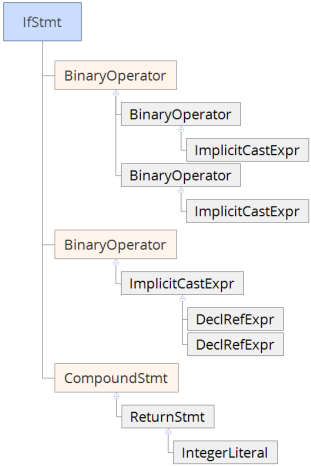
1. Convert the trees into weighted strings.
2. Compress the strings in order to save some space.
3. Obtain a similarity matrix a.k.a kernel matrix using a novel string kernel function.
4. Use a clustering algorithm to extract knowledge from the kernel matrix.

This work extends previous research from the authors, where they proposed a string kernel for the detection of patterns in I/O traces

- *“A novel string representation and kernel function for the comparison of I/O access patterns,” in Parallel Computing Technologies*.

From trees to strings

a) AST.



b) Extracted tokens.

Tokens	Repetitions
[IfStmt]	1
[BinaryOperator]	1
[BinaryOperator]	1
[ImplicitCastExpr]	1
[LEVEL_UP]	2
[BinaryOperator]	1
[ImplicitCastExpr]	1
[LEVEL_UP]	3
[BinaryOperator]	1
[ImplicitCastExpr]	1
[DeclRefExpr]	1
[LEVEL_UP]	1
[DeclRefExpr]	1
[LEVEL_UP]	3
[CompoundStmt]	1
[ReturnStmt]	1
[IntegerLiteral]	1
[LEVEL_UP]	4

Compression of the string

1. Similar consecutive tokens:

Compression of the string

1. Similar consecutive tokens:
   - $\texttt{[BinaryOp]}_1\,\texttt{[BinaryOp]}_1\,\texttt{[BinaryOp]}_1\,\texttt{[BinaryOp]}_1$

$$\downarrow$$

$$\texttt{[BinaryOperator]}_4$$

Compression of the string

1. Similar consecutive tokens:
   - $\texttt{[BinaryOp]}_1\,\texttt{[BinaryOp]}_1\,\texttt{[BinaryOp]}_1\,\texttt{[BinaryOp]}_1$

$$\downarrow$$

$$\texttt{[BinaryOperator]}_4$$

2. Delete specific tokens:

Compression of the string

1. Similar consecutive tokens:
   - $\texttt{[BinaryOp]}_1\texttt{[BinaryOp]}_1\texttt{[BinaryOp]}_1\texttt{[BinaryOp]}_1$
     $$\downarrow$$
     $$\texttt{[BinaryOperator]}_4$$
2. Delete specific tokens:
   - $\texttt{[CStyleCastExpr]}_1\texttt{[CallExpr]}_1\texttt{[ImplicitCastExpr]}_1$
     $$\texttt{[DeclRefExpr]}_1$$
     $$\downarrow$$
     $$\texttt{[DeclRefExpr]}_4$$

Compression of the string

1. Similar consecutive tokens:
   - $\texttt{[BinaryOp]}_1\texttt{[BinaryOp]}_1\texttt{[BinaryOp]}_1\texttt{[BinaryOp]}_1$

$$\downarrow$$

$$\texttt{[BinaryOperator]}_4$$

2. Delete specific tokens:
   - $\texttt{[CStyleCastExpr]}_1\texttt{[CallExpr]}_1\texttt{[ImplicitCastExpr]}_1$ $\texttt{[DeclRefExpr]}_1$

$$\downarrow$$

$$\texttt{[DeclRefExpr]}_4$$

3. Simplify declaration tokens:

Compression of the string

1. Similar consecutive tokens:
   - $\texttt{[BinaryOp]}_1\,\texttt{[BinaryOp]}_1\,\texttt{[BinaryOp]}_1\,\texttt{[BinaryOp]}_1$

     $\downarrow$

     $\texttt{[BinaryOperator]}_4$
2. Delete specific tokens:
   - $\texttt{[CStyleCastExpr]}_1\,\texttt{[CallExpr]}_1\,\texttt{[ImplicitCastExpr]}_1$
     $\texttt{[DeclRefExpr]}_1$

     $\downarrow$

     $\texttt{[DeclRefExpr]}_4$
3. Simplify declaration tokens:
   - $\texttt{[DeclStmt]}_1\,\texttt{[VarDecl]}_1\,\texttt{[DeclRefExpr]}_1\,\texttt{[LEVEL\_UP]}_4$

     $\downarrow$

     $\texttt{[DeclStmt]}_3\ \texttt{[LEVEL\_UP]}_4$

Compression of the string

1. Similar consecutive tokens:
   - [BinaryOp]$_1$ [BinaryOp]$_1$ [BinaryOp]$_1$ [BinaryOp]$_1$

     $\downarrow$

     [BinaryOperator]$_4$
2. Delete specific tokens:
   - [CStyleCastExpr]$_1$ [CallExpr]$_1$ [ImplicitCastExpr]$_1$ [DeclRefExpr]$_1$

     $\downarrow$

     [DeclRefExpr]$_4$
3. Simplify declaration tokens:
   - [DeclStmt]$_1$ [VarDecl]$_1$ [DeclRefExpr]$_1$ [LEVEL_UP]$_4$

     $\downarrow$

     [DeclStmt]$_3$ [LEVEL_UP]$_4$
4. Compress in pairs:

Compression of the string

1. Similar consecutive tokens:
   - $\texttt{[BinaryOp]}_1\texttt{[BinaryOp]}_1\texttt{[BinaryOp]}_1\texttt{[BinaryOp]}_1$

     $\downarrow$

     $\texttt{[BinaryOperator]}_4$
2. Delete specific tokens:
   - $\texttt{[CStyleCastExpr]}_1\texttt{[CallExpr]}_1\texttt{[ImplicitCastExpr]}_1$ $\texttt{[DeclRefExpr]}_1$

     $\downarrow$

     $\texttt{[DeclRefExpr]}_4$
3. Simplify declaration tokens:
   - $\texttt{[DeclStmt]}_1\texttt{[VarDecl]}_1\texttt{[DeclRefExpr]}_1\texttt{[LEVEL\_UP]}_4$

     $\downarrow$

     $\texttt{[DeclStmt]}_3$ $\texttt{[LEVEL\_UP]}_4$
4. Compress in pairs:
   - $\texttt{[IntegerLiteral]}_1\texttt{[LEVEL\_UP]}_5$ $\texttt{[IntegerLiteral]}_1\texttt{[LEVEL\_UP]}_2$

     $\downarrow$

     $\texttt{[IntegerLiteral]}_2\texttt{[LEVEL\_UP]}_7$

Having two strings A and B:

Having two strings A and B:

1. The algorithm requires a minimum weight or “cut weight” value as parameter.

Having two strings A and B:

1. The algorithm requires a minimum weight or “cut weight” value as parameter.
2. The aim is to find the longest matching substrings of A and B, whose weights are greater than or equal to the cut weight. They are called *valid matching substrings*.

Having two strings *A* and *B*:

1. The algorithm requires a minimum weight or “cut weight” value as parameter.
2. The aim is to find the longest matching substrings of *A* and *B*, whose weights are greater than or equal to the cut weight. They are called *valid matching substrings*.
3. A valid matching substring can appear more than once in each string.

Having two strings A and B:

1. The algorithm requires a minimum weight or “cut weight” value as parameter.
2. The aim is to find the longest matching substrings of A and B, whose weights are greater than or equal to the cut weight. They are called *valid matching substrings*.
3. A valid matching substring can appear more than once in each string.
4. A valid matching substring must not be a substring of another valid matching substring in at least one of the original strings.

KastX spectrum kernel family

An example with cut weight = 4

a) S_1 is the largest substring found on both examples.

A_{64} = a_3 b_2 c_4 d_2 e_1 f_5 g_1 h_1 i_1 j_2 k_1 l_3 m_1 n_2 f_3 g_1 h_2 o_1 p_1 q_1 r_2 s_1 t_5 u_9 b_7 c_2
19

17 18

B_{52} = v_2 a_5 b_1 c_1 d_3 e_1 f_4 g_1 h_1 w_2 x_2 y_1 a_1 b_2 c_6 d_1 e_3 f_1 g_1 h_3 z_1 b_5 c_1 f_1 g_1 h_1

KastX spectrum kernel family

An example with cut weight = 4

a) S_1 is the largest substring found on both examples.

$$A_{64} = \underbrace{a_3\ b_2\ c_4\ d_2\ e_1\ f_5\ g_1\ h_1}_{19}\ i_1\ j_2\ k_1\ l_3\ m_1\ n_2\ f_3\ g_1\ h_2\ o_1\ p_1\ q_1\ r_2\ s_1\ t_5\ u_9\ b_7\ c_2$$

$$B_{52} = v_2\ \overbrace{a_5\ b_1\ c_1\ d_3\ e_1\ f_4\ g_1\ h_1}^{17}\ w_2\ x_2\ y_1\ \overbrace{a_1\ b_2\ c_6\ d_1\ e_3\ f_1\ g_1\ h_3}^{18}\ z_1\ b_5\ c_1\ f_1\ g_1\ h_1$$

b) S_2 appears once as an independent case.

$$A_{64} = a_3\ b_2\ c_4\ d_2\ e_1\ \underbrace{f_5\ g_1\ h_1}_{7}\ i_1\ j_2\ k_1\ l_3\ m_1\ n_2\ \underbrace{f_3\ g_1\ h_2}_{6}\ o_1\ p_1\ q_1\ r_2\ s_1\ t_5\ u_9\ b_7\ c_2$$

$$B_{52} = v_2\ a_5\ b_1\ c_1\ d_3\ e_1\ \overbrace{f_4\ g_1\ h_1}^{6}\ w_2\ x_2\ y_1\ a_1\ b_2\ c_6\ d_1\ e_3\ \overbrace{f_1\ g_1\ h_3}^{5}\ z_1\ b_5\ c_1\ \overbrace{f_1\ g_1\ h_1}^{3\text{(ignored)}}$$

KastX spectrum kernel family

An example with cut weight = 4

a) S_1 is the largest substring found on both examples.

$A_{64} = a_3\ b_2\ c_4\ d_2\ e_1\ f_5\ g_1\ h_1\ i_1\ j_2\ k_1\ l_3\ m_1\ n_2\ f_3\ g_1\ h_2\ o_1\ p_1\ q_1\ r_2\ s_1\ t_5\ u_9\ b_7\ c_2$

19

17 18

$B_{52} = v_2\ a_5\ b_1\ c_1\ d_3\ e_1\ f_4\ g_1\ h_1\ w_2\ x_2\ y_1\ a_1\ b_2\ c_6\ d_1\ e_3\ f_1\ g_1\ h_3\ z_1\ b_5\ c_1\ f_1\ g_1\ h_1$

b) S_2 appears once as an independent case.

$A_{64} = a_3\ b_2\ c_4\ d_2\ e_1\ f_5\ g_1\ h_1\ i_1\ j_2\ k_1\ l_3\ m_1\ n_2\ f_3\ g_1\ h_2\ o_1\ p_1\ q_1\ r_2\ s_1\ t_5\ u_9\ b_7\ c_2$

7 6

6 5 3 (ignored)

$B_{52} = v_2\ a_5\ b_1\ c_1\ d_3\ e_1\ f_4\ g_1\ h_1\ w_2\ x_2\ y_1\ a_1\ b_2\ c_6\ d_1\ e_3\ f_1\ g_1\ h_3\ z_1\ b_5\ c_1\ f_1\ g_1\ h_1$

c) S_3 appears twice as an independent case.

$A_{64} = a_3\ b_2\ c_4\ d_2\ e_1\ f_5\ g_1\ h_1\ i_1\ j_2\ k_1\ l_3\ m_1\ n_2\ f_3\ g_1\ h_2\ o_1\ p_1\ q_1\ r_2\ s_1\ t_5\ u_9\ b_7\ c_2$

6 9

2 (ignored) 8 6

$B_{52} = v_2\ a_5\ b_1\ c_1\ d_3\ e_1\ f_4\ g_1\ h_1\ w_2\ x_2\ y_1\ a_1\ b_2\ c_6\ d_1\ e_3\ f_1\ g_1\ h_3\ z_1\ b_5\ c_1\ f_1\ g_1\ h_1$

The *kast1 spectrum kernel* has the following definition:

The *kast1 spectrum kernel* has the following definition:

- Each valid matching substring embeds a new feature for A and B.

The *kast1 spectrum kernel* has the following definition:

- Each valid matching substring embeds a new feature for A and B.
- The similarity value corresponds to the inner product of the new feature vectors of A and B.

The *kast1 spectrum kernel* has the following definition:

- Each valid matching substring embeds a new feature for A and B.
- The similarity value corresponds to the inner product of the new feature vectors of A and B.
- This kernel uses only the weight of the *independent* valid matching substrings.

The *kast1 spectrum kernel* has the following definition:

- Each valid matching substring embeds a new feature for A and B.
- The similarity value corresponds to the inner product of the new feature vectors of A and B.
- This kernel uses only the weight of the *independent* valid matching substrings.
- If the string does not present an independent occurrence of a particular valid matching substring, the feature value is set to 1, to avoid zero values when calculating the inner product.

Kast1 spectrum kernel: an example (I)

New feature vector for A

$$A_{64} = \underbrace{a_3\ b_2\ c_4\ d_2\ e_1\ f_5\ g_1\ h_1}_{19}\ i_1\ j_2\ k_1\ l_3\ m_1\ n_2\ f_3\ g_1\ h_2\ o_1\ p_1\ q_1\ r_2\ s_1\ t_5\ u_9\ b_7\ c_2$$

$$A_{64} = a_3\ b_2\ c_4\ d_2\ e_1\ \underbrace{f_5\ g_1\ h_1}_{7}\ i_1\ j_2\ k_1\ l_3\ m_1\ n_2\ \underbrace{f_3\ g_1\ h_2}_{6}\ o_1\ p_1\ q_1\ r_2\ s_1\ t_5\ u_9\ b_7\ c_2$$

$$A_{64} = a_3\ \underbrace{b_2\ c_4}_{6}\ d_2\ e_1\ f_5\ g_1\ h_1\ i_1\ j_2\ k_1\ l_3\ m_1\ n_2\ f_3\ g_1\ h_2\ o_1\ p_1\ q_1\ r_2\ s_1\ t_5\ u_9\ \underbrace{b_7\ c_2}_{9}$$

Kast1 spectrum kernel: an example (I)

New feature vector for A

$A_{64} = \underbrace{a_3\ b_2\ c_4\ d_2\ e_1\ f_5\ g_1\ h_1}_{19}\ i_1\ j_2\ k_1\ l_3\ m_1\ n_2\ f_3\ g_1\ h_2\ o_1\ p_1\ q_1\ r_2\ s_1\ t_5\ u_9\ b_7\ c_2$

$A_{64} = a_3\ b_2\ c_4\ d_2\ e_1\ \underbrace{f_5\ g_1\ h_1}_{7}\ i_1\ j_2\ k_1\ l_3\ m_1\ n_2\ \underbrace{f_3\ g_1\ h_2}_{6}\ o_1\ p_1\ q_1\ r_2\ s_1\ t_5\ u_9\ b_7\ c_2$

$A_{64} = a_3\ \underbrace{b_2\ c_4}_{6}\ d_2\ e_1\ f_5\ g_1\ h_1\ i_1\ j_2\ k_1\ l_3\ m_1\ n_2\ f_3\ g_1\ h_2\ o_1\ p_1\ q_1\ r_2\ s_1\ t_5\ u_9\ \underbrace{b_7\ c_2}_{9}$

$$weight\_k1_{w \geq 4}(S_1)_A = 19 \quad (1)$$

Kast1 spectrum kernel: an example (I)

New feature vector for A

$$A_{64} = \underbrace{a_3\ b_2\ c_4\ d_2\ e_1\ f_5\ g_1\ h_1}_{19}\ i_1\ j_2\ k_1\ l_3\ m_1\ n_2\ f_3\ g_1\ h_2\ o_1\ p_1\ q_1\ r_2\ s_1\ t_5\ u_9\ b_7\ c_2$$

$$A_{64} = a_3\ b_2\ c_4\ d_2\ e_1\ \underbrace{f_5\ g_1\ h_1}_{7}\ i_1\ j_2\ k_1\ l_3\ m_1\ n_2\ \underbrace{f_3\ g_1\ h_2}_{6}\ o_1\ p_1\ q_1\ r_2\ s_1\ t_5\ u_9\ b_7\ c_2$$

$$A_{64} = a_3\ \underbrace{b_2\ c_4}_{6}\ d_2\ e_1\ f_5\ g_1\ h_1\ i_1\ j_2\ k_1\ l_3\ m_1\ n_2\ f_3\ g_1\ h_2\ o_1\ p_1\ q_1\ r_2\ s_1\ t_5\ u_9\ \underbrace{b_7\ c_2}_{9}$$

$$weight\_k1_{w \geq 4}(S_1)_A = 19 \qquad (1)$$

$$weight\_k1_{w \geq 4}(S_2)_A = 6 \qquad (2)$$

Kast1 spectrum kernel: an example (I)

New feature vector for A

$$A_{64} = \underbrace{a_3\ b_2\ c_4\ d_2\ e_1\ f_5\ g_1\ h_1}_{19}\ i_1\ j_2\ k_1\ l_3\ m_1\ n_2\ f_3\ g_1\ h_2\ o_1\ p_1\ q_1\ r_2\ s_1\ t_5\ u_9\ b_7\ c_2$$

$$A_{64} = a_3\ b_2\ c_4\ d_2\ e_1\ \underbrace{f_5\ g_1\ h_1}_{7}\ i_1\ j_2\ k_1\ l_3\ m_1\ n_2\ \underbrace{f_3\ g_1\ h_2}_{6}\ o_1\ p_1\ q_1\ r_2\ s_1\ t_5\ u_9\ b_7\ c_2$$

$$A_{64} = a_3\ \underbrace{b_2\ c_4}_{6}\ d_2\ e_1\ f_5\ g_1\ h_1\ i_1\ j_2\ k_1\ l_3\ m_1\ n_2\ f_3\ g_1\ h_2\ o_1\ p_1\ q_1\ r_2\ s_1\ t_5\ u_9\ \underbrace{b_7\ c_2}_{9}$$

$$weight\_k1_{w\geq 4}(S_1)_A = 19 \tag{1}$$

$$weight\_k1_{w\geq 4}(S_2)_A = 6 \tag{2}$$

$$weight\_k1_{w\geq 4}(S_3)_A = 9 \tag{3}$$

Kast1 spectrum kernel: an example (I)

New feature vector for A

A_{64} = a_3 b_2 c_4 d_2 e_1 f_5 g_1 h_1 i_1 j_2 k_1 l_3 m_1 n_2 f_3 g_1 h_2 o_1 p_1 q_1 r_2 s_1 t_5 u_9 b_7 c_2

19

A_{64} = a_3 b_2 c_4 d_2 e_1 f_5 g_1 h_1 i_1 j_2 k_1 l_3 m_1 n_2 f_3 g_1 h_2 o_1 p_1 q_1 r_2 s_1 t_5 u_9 b_7 c_2

7 6

A_{64} = a_3 b_2 c_4 d_2 e_1 f_5 g_1 h_1 i_1 j_2 k_1 l_3 m_1 n_2 f_3 g_1 h_2 o_1 p_1 q_1 r_2 s_1 t_5 u_9 b_7 c_2

6 9

$$weight\_k1_{w\geq 4}(S_1)_A = 19 \tag{1}$$

$$weight\_k1_{w\geq 4}(S_2)_A = 6 \tag{2}$$

$$weight\_k1_{w\geq 4}(S_3)_A = 9 \tag{3}$$

$$f1_{w\geq 4}(A) = \{19, 6, 9\} \tag{4}$$

Kast1 spectrum kernel: an example (II)

New feature vector for B

$$\mathrm{B}_{52} = \mathrm{v}_2\ \overbrace{\mathrm{a}_5\ \mathrm{b}_1\ \mathrm{c}_1\ \mathrm{d}_3\ \mathrm{e}_1\ \mathrm{f}_4\ \mathrm{g}_1\ \mathrm{h}_1}^{17}\ \mathrm{w}_2\ \mathrm{x}_2\ \mathrm{y}_1\ \overbrace{\mathrm{a}_1\ \mathrm{b}_2\ \mathrm{c}_6\ \mathrm{d}_1\ \mathrm{e}_3\ \mathrm{f}_1\ \mathrm{g}_1\ \mathrm{h}_3}^{18}\ \mathrm{z}_1\ \mathrm{b}_5\ \mathrm{c}_1\ \mathrm{f}_1\ \mathrm{g}_1\ \mathrm{h}_1$$

$$\mathrm{B}_{52} = \mathrm{v}_2\ \mathrm{a}_5\ \mathrm{b}_1\ \mathrm{c}_1\ \mathrm{d}_3\ \mathrm{e}_1\ \overbrace{\mathrm{f}_4\ \mathrm{g}_1\ \mathrm{h}_1}^{6}\ \mathrm{w}_2\ \mathrm{x}_2\ \mathrm{y}_1\ \mathrm{a}_1\ \mathrm{b}_2\ \mathrm{c}_6\ \mathrm{d}_1\ \mathrm{e}_3\ \overbrace{\mathrm{f}_1\ \mathrm{g}_1\ \mathrm{h}_3}^{5}\ \mathrm{z}_1\ \mathrm{b}_5\ \mathrm{c}_1\ \overbrace{\mathrm{f}_1\ \mathrm{g}_1\ \mathrm{h}_1}^{3\text{(ignored)}}$$

$$\mathrm{B}_{52} = \mathrm{v}_2\ \mathrm{a}_5\ \overbrace{\mathrm{b}_1\ \mathrm{c}_1}^{2\text{(ignored)}}\ \mathrm{d}_3\ \mathrm{e}_1\ \mathrm{f}_4\ \mathrm{g}_1\ \mathrm{h}_1\ \mathrm{w}_2\ \mathrm{x}_2\ \mathrm{y}_1\ \mathrm{a}_1\ \overbrace{\mathrm{b}_2\ \mathrm{c}_6}^{8}\ \mathrm{d}_1\ \mathrm{e}_3\ \mathrm{f}_1\ \mathrm{g}_1\ \mathrm{h}_3\ \mathrm{z}_1\ \overbrace{\mathrm{b}_5\ \mathrm{c}_1}^{6}\ \mathrm{f}_1\ \mathrm{g}_1\ \mathrm{h}_1$$

Kast1 spectrum kernel: an example (II)

New feature vector for B

17 18

$B_{52} = v_2\ a_5\ b_1\ c_1\ d_3\ e_1\ f_4\ g_1\ h_1\ w_2\ x_2\ y_1\ a_1\ b_2\ c_6\ d_1\ e_3\ f_1\ g_1\ h_3\ z_1\ b_5\ c_1\ f_1\ g_1\ h_1$

6 5 3 (ignored)

$B_{52} = v_2\ a_5\ b_1\ c_1\ d_3\ e_1\ f_4\ g_1\ h_1\ w_2\ x_2\ y_1\ a_1\ b_2\ c_6\ d_1\ e_3\ f_1\ g_1\ h_3\ z_1\ b_5\ c_1\ f_1\ g_1\ h_1$

2 (ignored) 8 6

$B_{52} = v_2\ a_5\ b_1\ c_1\ d_3\ e_1\ f_4\ g_1\ h_1\ w_2\ x_2\ y_1\ a_1\ b_2\ c_6\ d_1\ e_3\ f_1\ g_1\ h_3\ z_1\ b_5\ c_1\ f_1\ g_1\ h_1$

$$weight\_k1_{w \geq 4}(S_1)_B = 17 + 18 = 35 \tag{5}$$

Kast1 spectrum kernel: an example (II)

New feature vector for B

$$B_{52} = v_2\ \overbrace{a_5\ b_1\ c_1\ d_3\ e_1\ f_4\ g_1\ h_1}^{17}\ w_2\ x_2\ y_1\ \overbrace{a_1\ b_2\ c_6\ d_1\ e_3\ f_1\ g_1\ h_3}^{18}\ z_1\ b_5\ c_1\ f_1\ g_1\ h_1$$

$$B_{52} = v_2\ a_5\ b_1\ c_1\ d_3\ e_1\ \overbrace{f_4\ g_1\ h_1}^{6}\ w_2\ x_2\ y_1\ a_1\ b_2\ c_6\ d_1\ e_3\ \overbrace{f_1\ g_1\ h_3}^{5}\ z_1\ b_5\ c_1\ \overbrace{f_1\ g_1\ h_1}^{3\,\text{(ignored)}}$$

$$B_{52} = v_2\ a_5\ \overbrace{b_1\ c_1}^{2\,\text{(ignored)}}\ d_3\ e_1\ f_4\ g_1\ h_1\ w_2\ x_2\ y_1\ a_1\ \overbrace{b_2\ c_6}^{8}\ d_1\ e_3\ f_1\ g_1\ h_3\ z_1\ \overbrace{b_5\ c_1}^{6}\ f_1\ g_1\ h_1$$

$$weight\_k1_{w \geq 4}(S_1)_B = 17 + 18 = 35 \qquad (5)$$

$$weight\_k1_{w \geq 4}(S_2)_B = 1 \qquad (6)$$

Kast1 spectrum kernel: an example (II)

New feature vector for B

$$B_{52} = v_2\ \overbrace{a_5\ b_1\ c_1\ d_3\ e_1\ f_4\ g_1\ h_1}^{17}\ w_2\ x_2\ y_1\ \overbrace{a_1\ b_2\ c_6\ d_1\ e_3\ f_1\ g_1\ h_3}^{18}\ z_1\ b_5\ c_1\ f_1\ g_1\ h_1$$

$$B_{52} = v_2\ a_5\ b_1\ c_1\ d_3\ e_1\ \overbrace{f_4\ g_1\ h_1}^{6}\ w_2\ x_2\ y_1\ a_1\ b_2\ c_6\ d_1\ e_3\ \overbrace{f_1\ g_1\ h_3}^{5}\ z_1\ b_5\ c_1\ \overbrace{f_1\ g_1\ h_1}^{3\text{(ignored)}}$$

$$B_{52} = v_2\ a_5\ \overbrace{b_1\ c_1}^{2\text{(ignored)}}\ d_3\ e_1\ f_4\ g_1\ h_1\ w_2\ x_2\ y_1\ a_1\ \overbrace{b_2\ c_6}^{8}\ d_1\ e_3\ f_1\ g_1\ h_3\ z_1\ \overbrace{b_5\ c_1}^{6}\ f_1\ g_1\ h_1$$

$$weight\_k1_{w \geq 4}(S_1)_B = 17 + 18 = 35 \tag{5}$$

$$weight\_k1_{w \geq 4}(S_2)_B = 1 \tag{6}$$

$$weight\_k1_{w \geq 4}(S_3)_B = 6 \tag{7}$$

Kast1 spectrum kernel: an example (II)

New feature vector for B

$$B_{52} = v_2\ \overbrace{a_5\ b_1\ c_1\ d_3\ e_1\ f_4\ g_1\ h_1}^{17}\ w_2\ x_2\ y_1\ \overbrace{a_1\ b_2\ c_6\ d_1\ e_3\ f_1\ g_1\ h_3}^{18}\ z_1\ b_5\ c_1\ f_1\ g_1\ h_1$$

$$B_{52} = v_2\ a_5\ b_1\ c_1\ d_3\ e_1\ \overbrace{f_4\ g_1\ h_1}^{6}\ w_2\ x_2\ y_1\ a_1\ b_2\ c_6\ d_1\ e_3\ \overbrace{f_1\ g_1\ h_3}^{5}\ z_1\ b_5\ c_1\ \overbrace{f_1\ g_1\ h_1}^{3\,\text{(ignored)}}$$

$$B_{52} = v_2\ a_5\ \overbrace{b_1\ c_1}^{2\,\text{(ignored)}}\ d_3\ e_1\ f_4\ g_1\ h_1\ w_2\ x_2\ y_1\ a_1\ \overbrace{b_2\ c_6}^{8}\ d_1\ e_3\ f_1\ g_1\ h_3\ z_1\ \overbrace{b_5\ c_1}^{6}\ f_1\ g_1\ h_1$$

$$weight\_k1_{w\geq 4}(S_1)_B = 17 + 18 = 35 \tag{5}$$

$$weight\_k1_{w\geq 4}(S_2)_B = 1 \tag{6}$$

$$weight\_k1_{w\geq 4}(S_3)_B = 6 \tag{7}$$

$$f1_{w\geq 4}(B) = \{35, 1, 6\} \tag{8}$$

Kast1 spectrum kernel: an example (III)

Similarity calculation

$$k1_{w \geq 4}(A, B) = \langle \{19, 6, 9\}, \{35, 1, 6\} \rangle = 725 \tag{9}$$

Kast1 spectrum kernel: an example (III)

Similarity calculation

$$k1_{w \geq 4}(A, B) = \langle \{19, 6, 9\}, \{35, 1, 6\} \rangle = 725 \tag{9}$$

$$\bar{k1}_{w \geq 4}(A, B) = \frac{725}{\sqrt{k1_{w \geq 4}(A, A) \times k1_{w \geq 4}(B, B)}} \tag{10}$$

Kast1 spectrum kernel: an example (III)

Similarity calculation

$$k1_{w \geq 4}(A, B) = \langle \{19, 6, 9\}, \{35, 1, 6\} \rangle = 725 \tag{9}$$

$$\bar{k}1_{w \geq 4}(A, B) = \frac{725}{\sqrt{k1_{w \geq 4}(A, A) \times k1_{w \geq 4}(B, B)}} \tag{10}$$

$$\bar{k}1_{w \geq 4}(A, B) = \frac{725}{64 \times 52} \tag{11}$$

Kast1 spectrum kernel: an example (III)

Similarity calculation

$$k1_{w\geq 4}(A,B) = \langle\{19,6,9\},\{35,1,6\}\rangle = 725 \tag{9}$$

$$\bar{k}1_{w\geq 4}(A,B) = \frac{725}{\sqrt{k1_{w\geq 4}(A,A) \times k1_{w\geq 4}(B,B)}} \tag{10}$$

$$\bar{k}1_{w\geq 4}(A,B) = \frac{725}{64 \times 52} \tag{11}$$

$$\bar{k}1_{w\geq 4}(A,B) = \frac{725}{3328} \tag{12}$$

Kast1 spectrum kernel: an example (III)

Similarity calculation

$$k1_{w\geq 4}(A,B) = \langle\{19,6,9\},\{35,1,6\}\rangle = 725 \tag{9}$$

$$\bar{k1}_{w\geq 4}(A,B) = \frac{725}{\sqrt{k1_{w\geq 4}(A,A) \times k1_{w\geq 4}(B,B)}} \tag{10}$$

$$\bar{k1}_{w\geq 4}(A,B) = \frac{725}{64 \times 52} \tag{11}$$

$$\bar{k1}_{w\geq 4}(A,B) = \frac{725}{3328} \tag{12}$$

$$\bar{k1}_{w\geq 4}(A,B) \approx 0.2178 \tag{13}$$

$$k1_{w\geq 4}(A,B) = \langle\{19,6,9\},\{35,1,6\}\rangle = 725 \tag{9}$$

$$\bar{k}1_{w\geq 4}(A,B) = \frac{725}{\sqrt{k1_{w\geq 4}(A,A) \times k1_{w\geq 4}(B,B)}} \tag{10}$$

$$\bar{k}1_{w\geq 4}(A,B) = \frac{725}{64 \times 52} \tag{11}$$

$$\bar{k}1_{w\geq 4}(A,B) = \frac{725}{3328} \tag{12}$$

$$\bar{k}1_{w\geq 4}(A,B) \approx 0.2178 \tag{13}$$

According to this kernel, Strings A and B are approximately 21.78% similar.

4. Evaluation

Code samples (I)

20 functions X 5 versions(Original, Type-1, Type-2, Type-3 and Type-4 clones) classified as follows:

Code samples (I)

20 functions X 5 versions(Original, Type-1, Type-2, Type-3 and Type-4 clones) classified as follows:

- (A) Matching functions.

Code samples (I)

20 functions X 5 versions(Original, Type-1, Type-2, Type-3 and Type-4 clones) classified as follows:

- (A) Matching functions.
  - K-spectrum kernel.

Code samples (I)

20 functions X 5 versions(Original, Type-1, Type-2, Type-3 and Type-4 clones) classified as follows:

- (A) Matching functions.
  - K-spectrum kernel.
  - Blended spectrum.

Code samples (I)

20 functions X 5 versions(Original, Type-1, Type-2, Type-3 and Type-4 clones) classified as follows:

- (A) Matching functions.
  - K-spectrum kernel.
  - Blended spectrum.
  - Bag-of-characters kernel.

Code samples (I)

20 functions X 5 versions(Original, Type-1, Type-2, Type-3 and Type-4 clones) classified as follows:

- (A) Matching functions.
  - K-spectrum kernel.
  - Blended spectrum.
  - Bag-of-characters kernel.
  - Bag-of-words kernel.

Code samples (I)

20 functions X 5 versions(Original, Type-1, Type-2, Type-3 and Type-4 clones) classified as follows:

- (A) Matching functions.
  - K-spectrum kernel.
  - Blended spectrum.
  - Bag-of-characters kernel.
  - Bag-of-words kernel.
  - Bag-of-sentences kernel.

Code samples (I)

20 functions X 5 versions(Original, Type-1, Type-2, Type-3 and Type-4 clones) classified as follows:

- (A) Matching functions.
  - K-spectrum kernel.
  - Blended spectrum.
  - Bag-of-characters kernel.
  - Bag-of-words kernel.
  - Bag-of-sentences kernel.
- (B) Sort functions.

Code samples (I)

20 functions X 5 versions(Original, Type-1, Type-2, Type-3 and Type-4 clones) classified as follows:

- (A) Matching functions.
  - K-spectrum kernel.
  - Blended spectrum.
  - Bag-of-characters kernel.
  - Bag-of-words kernel.
  - Bag-of-sentences kernel.
- (B) Sort functions.
  - Bubble sort.

Code samples (I)

20 functions X 5 versions(Original, Type-1, Type-2, Type-3 and Type-4 clones) classified as follows:

- (A) Matching functions.
  - K-spectrum kernel.
  - Blended spectrum.
  - Bag-of-characters kernel.
  - Bag-of-words kernel.
  - Bag-of-sentences kernel.
- (B) Sort functions.
  - Bubble sort.
  - Insert sort.

Code samples (I)

20 functions X 5 versions(Original, Type-1, Type-2, Type-3 and Type-4 clones) classified as follows:

- (A) Matching functions.
  - K-spectrum kernel.
  - Blended spectrum.
  - Bag-of-characters kernel.
  - Bag-of-words kernel.
  - Bag-of-sentences kernel.
- (B) Sort functions.
  - Bubble sort.
  - Insert sort.
  - Selection sort.

Code samples (I)

20 functions X 5 versions(Original, Type-1, Type-2, Type-3 and Type-4 clones) classified as follows:

- (A) Matching functions.
  - K-spectrum kernel.
  - Blended spectrum.
  - Bag-of-characters kernel.
  - Bag-of-words kernel.
  - Bag-of-sentences kernel.
- (B) Sort functions.
  - Bubble sort.
  - Insert sort.
  - Selection sort.
  - Heap sort.

Code samples (I)

20 functions X 5 versions(Original, Type-1, Type-2, Type-3 and Type-4 clones) classified as follows:

- (A) Matching functions.
  - K-spectrum kernel.
  - Blended spectrum.
  - Bag-of-characters kernel.
  - Bag-of-words kernel.
  - Bag-of-sentences kernel.
- (B) Sort functions.
  - Bubble sort.
  - Insert sort.
  - Selection sort.
  - Heap sort.
  - Merge sort.

Code samples (II)

- (C) 3D stencils.

Code samples (II)

- (C) 3D stencils.
  - Compact stencil.

Code samples (II)

- (C) 3D stencils.
  - Compact stencil.
  - Side stencil.

Code samples (II)

- (C) 3D stencils.
  - Compact stencil.
  - Side stencil.
  - Edge stencil.

Code samples (II)

- (C) 3D stencils.
  - Compact stencil.
  - Side stencil.
  - Edge stencil.
  - Vertex stencil.

Code samples (II)

- (C) 3D stencils.
  - Compact stencil.
  - Side stencil.
  - Edge stencil.
  - Vertex stencil.
  - Non-compact stencil 1 layer.

Code samples (II)

- (C) 3D stencils.
  - Compact stencil.
  - Side stencil.
  - Edge stencil.
  - Vertex stencil.
  - Non-compact stencil 1 layer.
- (D) 2D stencils.

Code samples (II)

- (C) 3D stencils.
  - Compact stencil.
  - Side stencil.
  - Edge stencil.
  - Vertex stencil.
  - Non-compact stencil 1 layer.
- (D) 2D stencils.
  - Compact stencil.

Code samples (II)

- (C) 3D stencils.
  - Compact stencil.
  - Side stencil.
  - Edge stencil.
  - Vertex stencil.
  - Non-compact stencil 1 layer.
- (D) 2D stencils.
  - Compact stencil.
  - Edge stencil.

Code samples (II)

- (C) 3D stencils.
  - Compact stencil.
  - Side stencil.
  - Edge stencil.
  - Vertex stencil.
  - Non-compact stencil 1 layer.
- (D) 2D stencils.
  - Compact stencil.
  - Edge stencil.
  - Vertex stencil.

Code samples (II)

- (C) 3D stencils.
  - Compact stencil.
  - Side stencil.
  - Edge stencil.
  - Vertex stencil.
  - Non-compact stencil 1 layer.
- (D) 2D stencils.
  - Compact stencil.
  - Edge stencil.
  - Vertex stencil.
  - Non-compact stencil 1 layer.

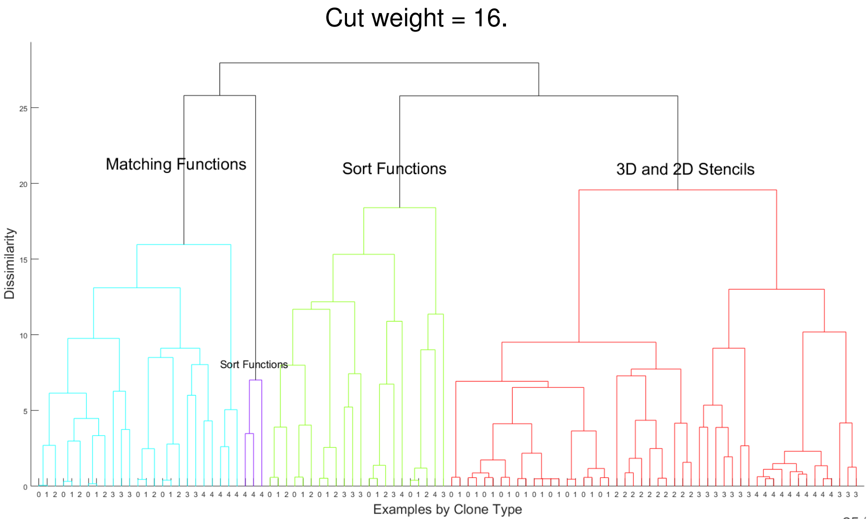
Code samples (II)

- (C) 3D stencils.
  - Compact stencil.
  - Side stencil.
  - Edge stencil.
  - Vertex stencil.
  - Non-compact stencil 1 layer.
- (D) 2D stencils.
  - Compact stencil.
  - Edge stencil.
  - Vertex stencil.
  - Non-compact stencil 1 layer.
  - Non-compact stencil 2 layers.

Other setups

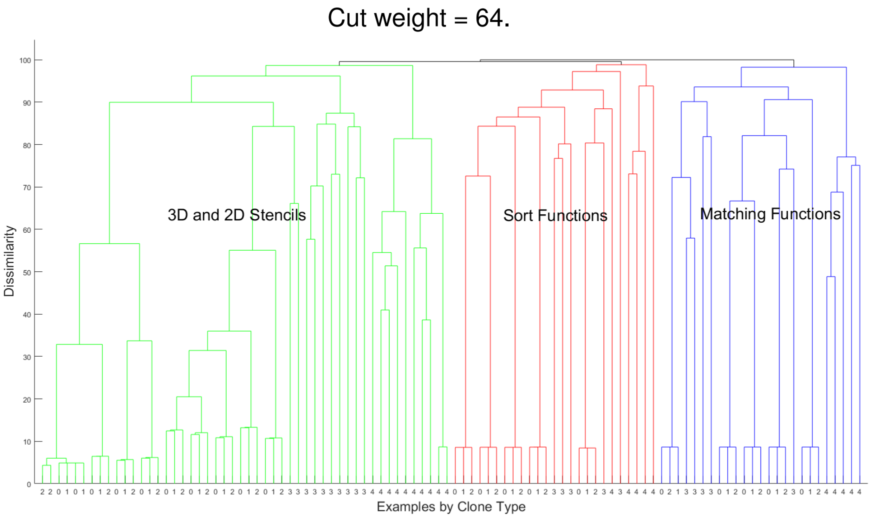
- The selected cut weight values were the following:
  - $\{2^0, 2^1, ..., 2^k\} : k = 9.$
- The clustering algorithm here used was:
  - Hierarchical Clustering.

Blended spectrum kernel



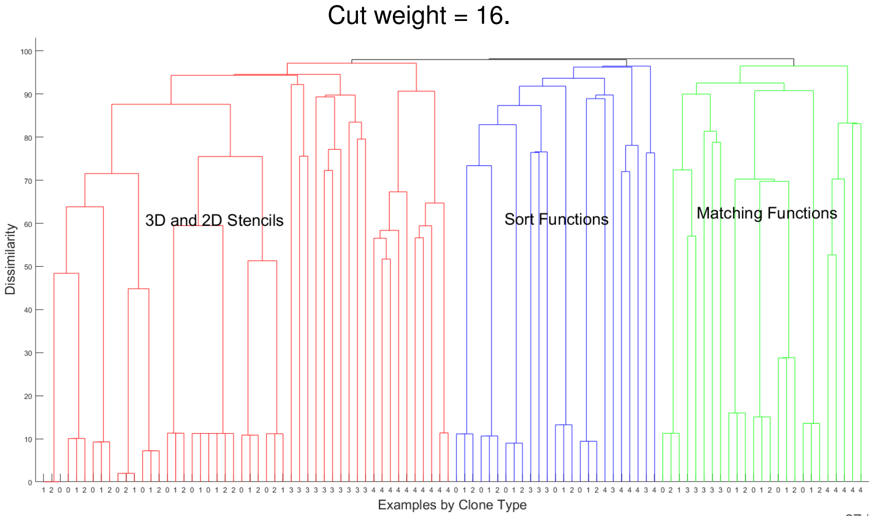
Baseline kernel 2 (previous work)

Kast spectrum kernel



Proposed kernel

Kast1 spectrum kernel



5. Conclusions and Future Work

Conclusions

- The proposed *kast1 spectrum kernel* and the *kast spectrum kernel* had similar clustering performance.

Conclusions

- The proposed *kast1 spectrum kernel* and the *kast spectrum kernel* had similar clustering performance.
- They showed a consistent formation of three clusters: matching functions, sorting functions and stencils (3D and 2D).

Conclusions

- The proposed *kast1 spectrum kernel* and the *kast spectrum kernel* had similar clustering performance.
- They showed a consistent formation of three clusters: matching functions, sorting functions and stencils (3D and 2D).
- They yielded better results than the *blended spectrum kernel* as the clustering showed no misplaced examples.

Conclusions

- The proposed *kast1 spectrum kernel* and the *kast spectrum kernel* had similar clustering performance.
- They showed a consistent formation of three clusters: matching functions, sorting functions and stencils (3D and 2D).
- They yielded better results than the *blended spectrum kernel* as the clustering showed no misplaced examples.
- This indicates that this novel comparison method can be promisingly utilized to find similarities in source code snippets.

Future work

- Automatic selection of the cut weight.

Future work

- Automatic selection of the cut weight.
- Analysis of the intra-cluster distances between clone types.

Future work

- Automatic selection of the cut weight.
- Analysis of the intra-cluster distances between clone types.
- Comparison against a tree kernel applied directly over the ASTs.

Future work

- Automatic selection of the cut weight.
- Analysis of the intra-cluster distances between clone types.
- Comparison against a tree kernel applied directly over the ASTs.
- Study the linear intermediate representation delivered by the LLVM Compiler Infrastructure.

Thanks!