# ENCOPLOT: Pairwise Sequence Matching in Linear Time Applied to Plagiarism Detection

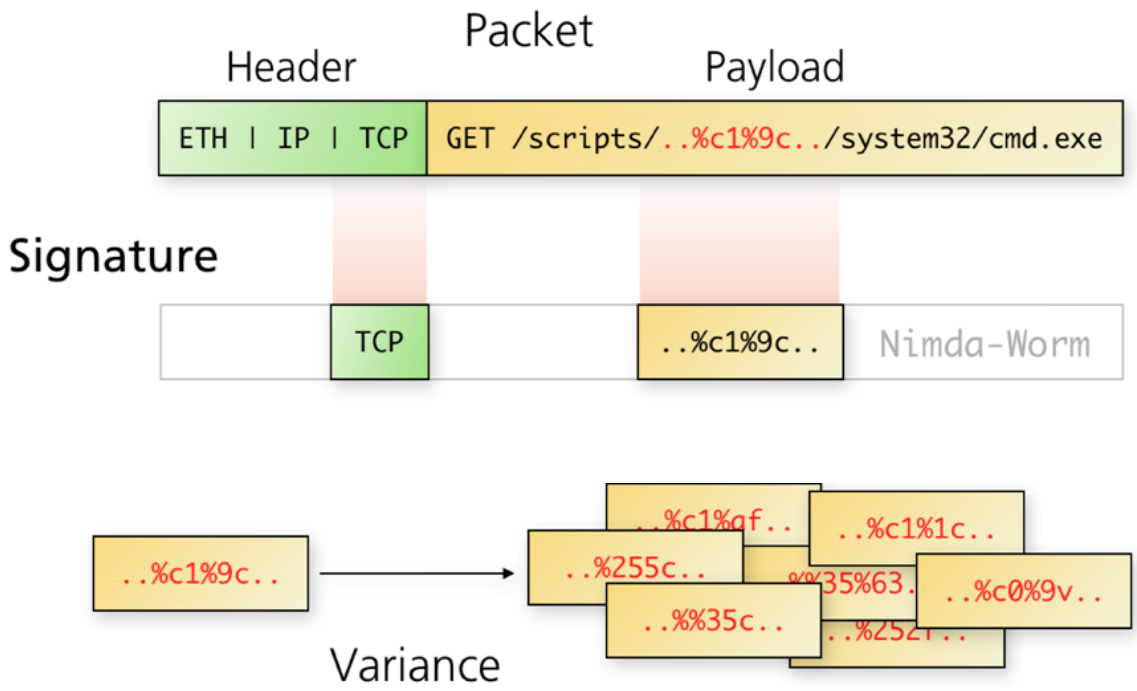
Cristian Grozea, **Christian Gehl**, Marius N. Popescu*
christian.gehl@first.fraunhofer.de

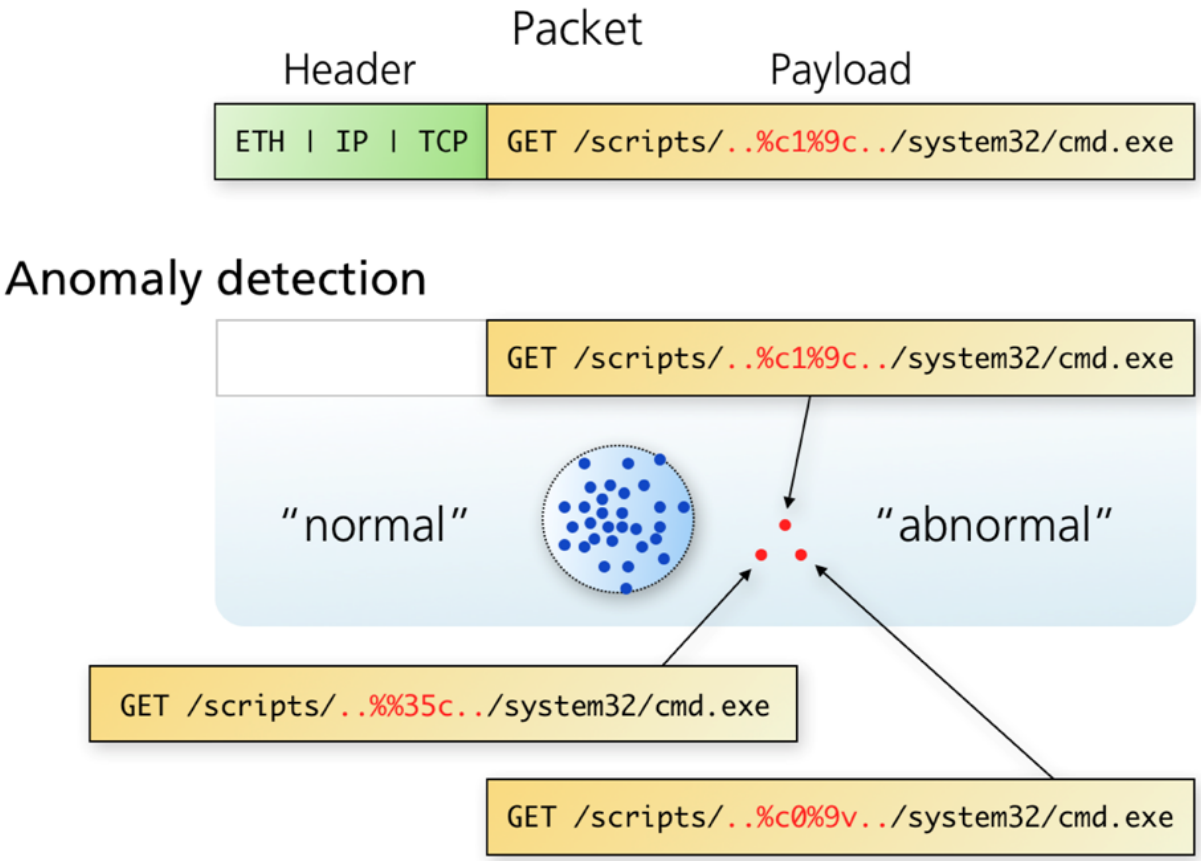
Fraunhofer Institute FIRST (IDA)
Project ReMIND

September 10, 2009

University of Bucharest*

# Contents


Packet
Header
Payload
ETH | IP | TCP
GET /scripts/..%c1%9c../system32/cmd.exe
Signature
TCP
..%c1%9c..
Nimda-Worm
..%c1%9c..
..%c1%af..
..%c1%1c..
..%255c..
..%c0%9v..
..%%35c..
Variance




- *libmindy*: extraction and embedding of n-gram (character,words) and pairwise similarity measures (distances, kernel functions)

# What is and what is not plagiarism

- Copying of text - unless it's quoting - is plagiarism.
  Easy to detect
  - can be detected at the text level
- Copying ideas is also plagiarism.
  Not so easy to detect
  - can be seen at semantic level
- Self-plagiarism: Copying text from your own previous papers.
  Unclear
  - it is not considered plagiarism by some

# 1st International Competition on Plagiarism Detection

- Training dataset, plagiarism annotated
- Test dataset, unannotated, used for evaluation
- each 7000 source documents and 7000 suspicious documents
- Automatic plagiarism and obfuscation: reorder paragraphs, change and insert or delete words
- Two tasks: internal plagiarism (spot passages that are not matching the context e.g. in style), external plagiarism (find the source in a given list and indicate what passages are copied from where)

# Two types

- Based on indexing (hashing)

## Features

Indexing a collection allows for fast retrieval of the matching documents for a query.
The size of the document set that the collection was created from is not so much a factor.
But, queries are rather inflexible (exact matching is easiest to have).

- Pairwise comparison

## Features

The time to check against N possible source documents is O(N).
Best flexibility in matching. This is what we used.

# Two types

- Based on indexing (hashing)

## Features

Indexing a collection allows for fast retrieval of the matching documents for a query.
The size of the document set that the collection was created from is not so much a factor.
But, queries are rather inflexible (exact matching is easiest to have).

- Pairwise comparison

## Features

The time to check against N possible source documents is O(N).
Best flexibility in matching. This is what we used.

# Possibly the best plagiarism detection

Many ways to see copying/plagiarism between two texts:

- common substrings
- redundancy
- common information
- deficiency of the novel information

# N-Gram Coincidence Plot

## Algorithm

Input: Sequences A and B to compare

Output: list (x,y) of positions in A, respectively B, where there is exactly the same N-gram

Steps

1. Extract the N-grams from A and B
2. Sort these two lists of N-grams
3. Compare these lists in a modified mergesort algorithm. Whenever the two smallest N-grams are the equal, output the position in A and the one in B.

# N-Gram Coincidence Plot

## Algorithm

Input: Sequences A and B to compare

Output: list (x,y) of positions in A, respectively B, where there is exactly the same N-gram

### Steps

1. Extract the N-grams from A and B
2. Sort these two lists of N-grams
3. Compare these lists in a modified mergesort algorithm. Whenever the two smallest N-grams are the equal, output the position in A and the one in B.

# N-Gram Coincidence Plot

## Algorithm

Input: Sequences A and B to compare
Output: list (x,y) of positions in A, respectively B, where there is exactly the same N-gram
Steps
1. Extract the N-grams from A and B
2. Sort these two lists of N-grams
3. Compare these lists in a modified mergesort algorithm. Whenever the two smallest N-grams are the equal, output the position in A and the one in B.

# N-Gram Coincidence Plot

## Algorithm

Input: Sequences A and B to compare
Output: list (x,y) of positions in A, respectively B, where there is exactly the same N-gram
Steps
1. Extract the N-grams from A and B
2. Sort these two lists of N-grams
3. Compare these lists in a modified mergesort algorithm. Whenever the two smallest N-grams are the equal, output the position in A and the one in B.

# N-Gram Coincidence Plot

## Algorithm

Input: Sequences A and B to compare
Output: list (x,y) of positions in A, respectively B, where there is exactly the same N-gram
Steps
1. Extract the N-grams from A and B
2. Sort these two lists of N-grams
3. Compare these lists in a modified mergesort algorithm.

Whenever the two smallest N-grams are the equal, output the position in A and the one in B.

# Small example

A=abcabd
B=xabdy

N=2

| Encoplot pairs | Dotplot pairs |
|---|---|
| 1 2 ab | 1 2 ab |
| | 4 2 ab |
| 5 4 bd | 5 4 bd |

N=3

| Encoplot pairs | Dotplot pairs |
|---|---|
| 4 2 abd | 4 2 abd |

# Small example

A=abcabd
B=xabdy

N=2

| Encoplot pairs | Dotplot pairs |
|---|---|
| 1 2 ab | 1 2 ab |
| | 4 2 ab |
| 5 4 bd | 5 4 bd |

N=3

| Encoplot pairs | Dotplot pairs |
|---|---|
| 4 2 abd | 4 2 abd |

# Small example

A=abcabd
B=xabdy

N=2

| Encoplot pairs | Dotplot pairs |
|---|---|
| 1 2 ab | 1 2 ab |
| | 4 2 ab |
| 5 4 bd | 5 4 bd |

N=3

| Encoplot pairs | Dotplot pairs |
|---|---|
| 4 2 abd | 4 2 abd |

# Encoplot Features

- Guaranteed linear time (Dotplot is quadratic).
- Field-agnostic, possible to use in computational biology as well, for example.
- Extremely fast highly optimized implementation available (for N up to 16, on 64 bit CPUs).

# Encoplot vs Dotplot Analysis

**Question: what is the price paid for speed?**

Encoplot matches the first N-gram in text $A$ with the first identical N-gram in the text $B$, the second occurence with the second occurence and so on.

Encoplot may break sequences on N-grams that are duplicated in one of the texts. A sequence too fragmented may no longer lead to the recognition of a suspicious match.

Being duplicated means their informational content is reduced (e.g. typical formulations such as "despite this, we are").

Only the parts that are rather unique in each of the text are guaranteed to be put in correspondence. Hopefully these correspond to high information substrings, "signatures" that really identify the text.

# Encoplot vs Dotplot Analysis

**Question: what is the price paid for speed?**

Encoplot matches the first N-gram in text $A$ with the first identical N-gram in the text $B$, the second occurence with the second occurence and so on.

Encoplot may break sequences on N-grams that are duplicated in one of the texts. A sequence too fragmented may no longer lead to the recognition of a suspicious match.

Being duplicated means their informational content is reduced (e.g. typical formulations such as "despite this, we are").

Only the parts that are rather unique in each of the text are guaranteed to be put in correspondence. Hopefully these correspond to high information substrings, "signatures" that really identify the text.

# Encoplot vs Dotplot Analysis

**Question: what is the price paid for speed?**

Encoplot matches the first N-gram in text $A$ with the first identical N-gram in the text $B$, the second occurence with the second occurence and so on.

Encoplot may break sequences on N-grams that are duplicated in one of the texts. A sequence too fragmented may no longer lead to the recognition of a suspicious match.

Being duplicated means their informational content is reduced (e.g. typical formulations such as "despite this, we are").

Only the parts that are rather unique in each of the text are guaranteed to be put in correspondence. Hopefully these correspond to high information substrings, "signatures" that really identify the text.

# Encoplot vs Dotplot Analysis

**Question: what is the price paid for speed?**

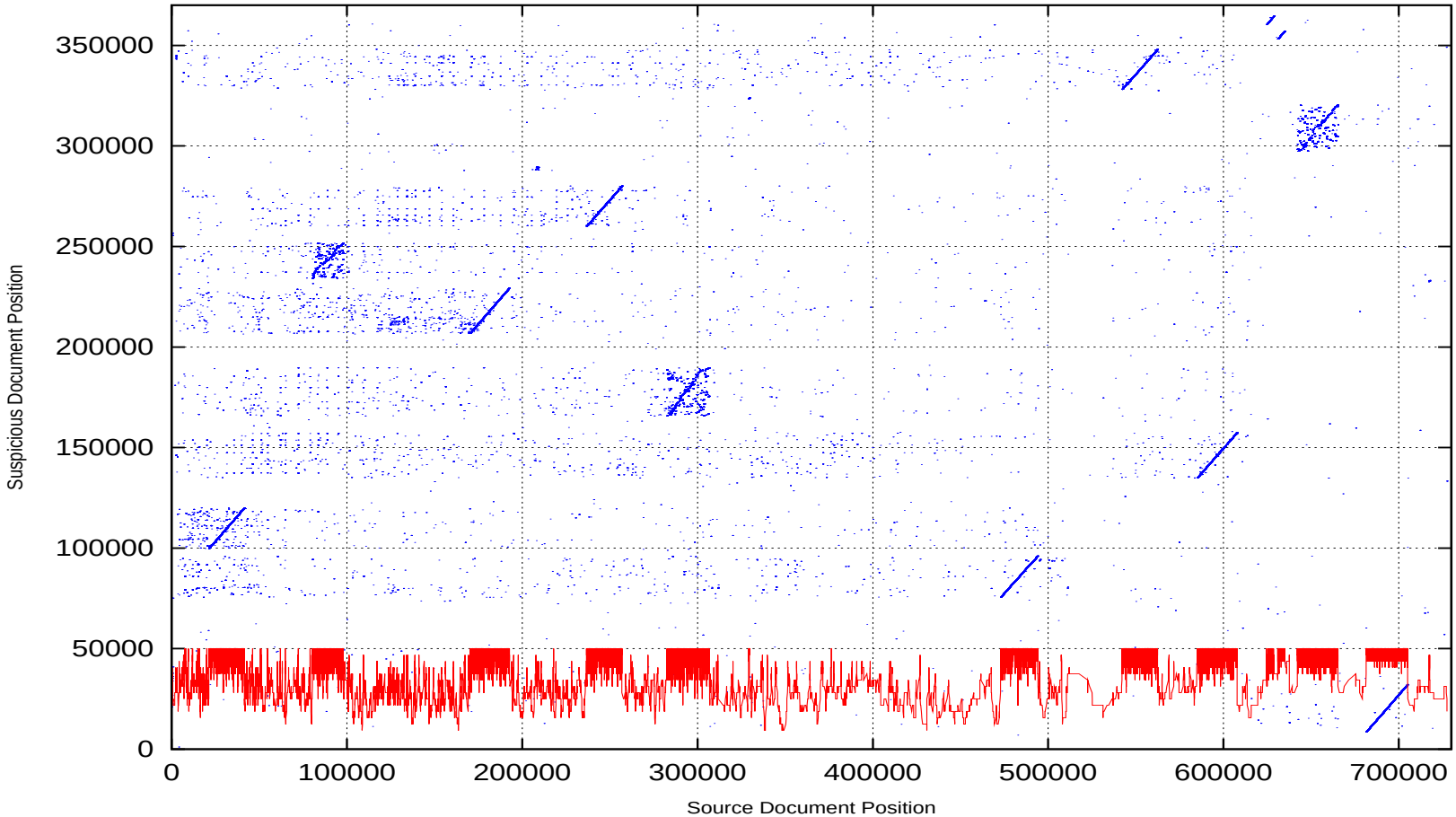
Encoplot matches the first N-gram in text $A$ with the first identical N-gram in the text $B$, the second occurence with the second occurence and so on.

Encoplot may break sequences on N-grams that are duplicated in one of the texts. A sequence too fragmented may no longer lead to the recognition of a suspicious match.

Being duplicated means their informational content is reduced (e.g. typical formulations such as "despite this, we are").

Only the parts that are rather unique in each of the text are guaranteed to be put in correspondence. Hopefully these correspond to high information substrings, "signatures" that really identify the text.

# Encoplot: source 3094 vs suspicious 9



# Challenge approach

No stemming (looked like it brings only 1% improval of the performance).
Used 16-grams, character based, as opposed to word based - good for avoiding to treat common formulations as significant.

- Computation of a kernel matrix (49 million pairs) using a linear kernel over binary representation of 16-grams (ignoring frequency in document), normalized.
- Selection of the pruning: best worked by ranking using the kernel of the suspicious documents for each source document.
- Kept 50 "most suspicious" for each source.

# Challenge approach

No stemming (looked like it brings only 1% improval of the performance).
Used 16-grams, character based, as opposed to word based - good for avoiding to treat common formulations as significant.

- Computation of a kernel matrix (49 million pairs) using a linear kernel over binary representation of 16-grams (ignoring frequency in document), normalized.
- Selection of the pruning: best worked by ranking using the kernel of the suspicious documents for each source document.
- Kept 50 "most suspicious" for each source.

# Challenge approach

No stemming (looked like it brings only 1% improval of the performance).
Used 16-grams, character based, as opposed to word based - good for avoiding to treat common formulations as significant.

- Computation of a kernel matrix (49 million pairs) using a linear kernel over binary representation of 16-grams (ignoring frequency in document), normalized.
- Selection of the pruning: best worked by ranking using the kernel of the suspicious documents for each source document.
- Kept 50 “most suspicious” for each source.

# Challenge approach – continued

- For each (source, suspicious) pair in the about 350,000 kept, compute the encoplot and apply a heuristic to isolate the clusters (diagonals), in linear time.
- Filter once more the list of detections, in order to only keep the very convincing matches (long, still holding after whitespace elimination, high matching score). This increases the precision (less false positives) with the price of decreasing the recall (more false negatives).

# Challenge approach – continued

- For each (source, suspicious) pair in the about 350,000 kept, compute the encoplot and apply a heuristic to isolate the clusters (diagonals), in linear time.
- Filter once more the list of detections, in order to only keep the very convincing matches (long, still holding after whitespace elimination, high matching score). This increases the precision (less false positives) with the price of decreasing the recall (more false negatives).

# Performance

- *libmindy* able to compute kernel matrix with 49 millions elements in 12 hours on an 8-core machine.
- encoplot + heuristic clustering of dots able to do detailed analysis (passages matching) for 350000 document pairs in less than 8 hours.

# Performance

- *libmindy* able to compute kernel matrix with 49 millions elements in 12 hours on an 8-core machine.
- encoplot + heuristic clustering of dots able to do detailed analysis (passages matching) for 350000 document pairs in less than 8 hours.

# Results

| Rank | Overall score | F-measure | Precision | Recall | Granularity | Participant |
|---|---|---|---|---|---|---|
| 1 | 0.6957 | 0.6976 | 0.7418 | 0.6585 | 1.0027 | C. Grozea<br>Fraunhofer FIRST, Germany |
| 2 | 0.6093 | 0.6192 | 0.5573 | 0.6967 | 1.0164 | J. Kasprzak, M. Brandejs, and M. Kipa<br>Masaryk University, Czech Republic |
| 3 | 0.6041 | 0.6491 | 0.6727 | 0.6272 | 1.0745 | C. Basile(a), D. Benedetto(b), E. Caglioti<br>(a)Universit di Bologna and (b)Universit L |

Thank you! (time to take questions)