Fast Approximate Matching of Programs for Protecting Libre/Open Source Software by Using Spatial Indexes

Arnoldo Müller   Takeshi Shinohara

Department of Artificial Intelligence
Kyushu Institute of Technology (Iizuka, Japan)

SCAM 2007

Libre/ Open Source Software

Protect Libre Software!

- FLOSS (Free/Libre Open Source Software).
- Copy-left. (GPL)
- Licensing Violations.
  - FSF and GPL-violations.
  - Using “strings” (binutils).
- Objective: Binary Program Matching.
  - Different Compilers/Obfuscators/Strings.
  - Return the top *n* most similar programs.
  - Detect License Violations.

Libre/ Open Source Software

Protect Libre Software!

- FLOSS (Free/Libre Open Source Software).
- Copy-left. (GPL)
- Licensing Violations.
  - FSF and GPL-violations.
  - Using “strings” (binutils).
- Objective: Binary Program Matching.
  - Different Compilers/Obfuscators/Strings.
  - Return the top *n* most similar programs.
  - Detect License Violations.

Libre/ Open Source Software

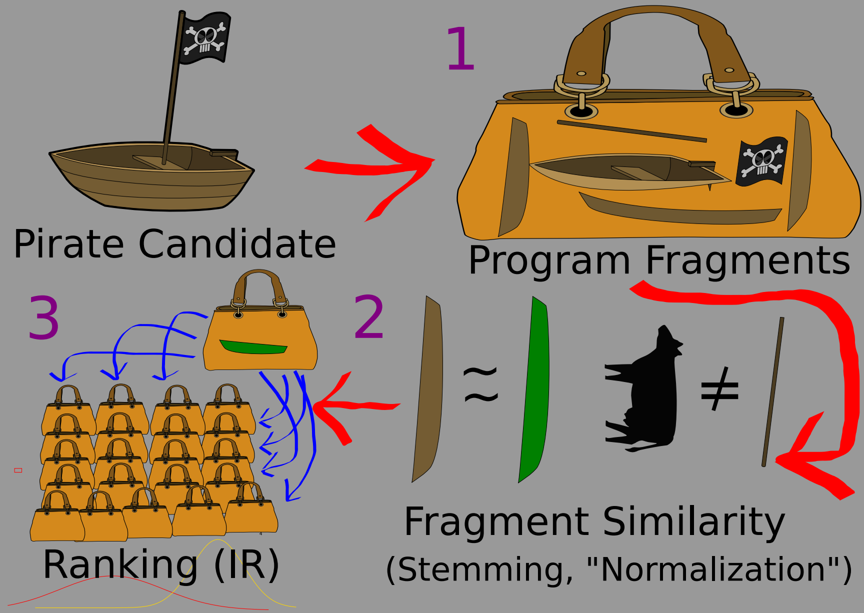
Protect Libre Software!

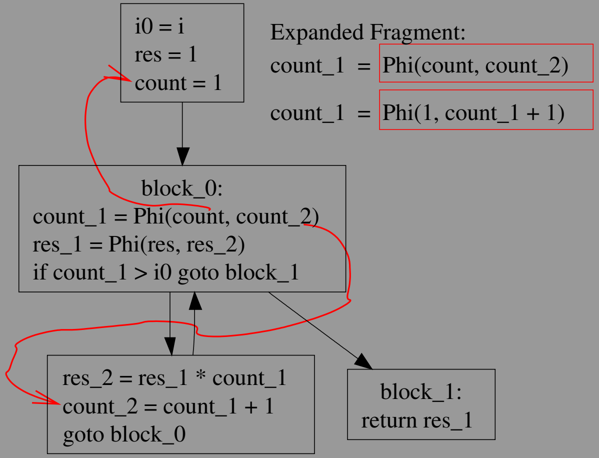
- FLOSS (Free/Libre Open Source Software).
- Copy-left. (GPL)
- Licensing Violations.
  - FSF and GPL-violations.
  - Using "strings" (binutils).
- Objective: Binary Program Matching.
  - Different Compilers/Obfuscators/Strings.
  - Return the top *n* most similar programs.
  - Detect License Violations.

Libre/ Open Source Software

Protect Libre Software!

- FLOSS (Free/Libre Open Source Software).
- Copy-left. (GPL)
- Licensing Violations.
  - FSF and GPL-violations.
  - Using “strings” (binutils).
- Objective: Binary Program Matching.
  - Different Compilers/Obfuscators/Strings.
  - Return the top *n* most similar programs.
  - Detect License Violations.


1
Pirate Candidate
Program Fragments
2
≈
≠
Fragment Similarity
(Stemming, "Normalization")
3
Ranking (IR)



- Output: machine instruction trees.
- Matched with a distance function *d*.
- Ignore strings, function names.

Fast Matching With Spatial Indexes

Select pivots, create a tuple based on *d*.

- Similarity search:
  - M-tree & friends: slow for *d*.
  - Vectors: B-tree (1 dim) or Spatial Index.
- Trees cannot be indexed directly.
- SMAP can be used to index them:
  - Select *i* pivots $p_1 \ldots p_i$ from the database.
  - Create a tuple $(d(o, p_1), \ldots, d(o, p_i))$ for each object *o*.
  - Calculate lower bound with L_∞ distance.
- This tuple can be stored in the Spatial Index!

Fast Matching With Spatial Indexes

Select pivots, create a tuple based on *d*.

- Similarity search:
  - M-tree & friends: slow for *d*.
  - Vectors: B-tree (1 dim) or Spatial Index.
- Trees cannot be indexed directly.
- SMAP can be used to index them:
  - Select *i* pivots $p_1 \ldots p_i$ from the database.
  - Create a tuple $(d(o, p_1), \ldots, d(o, p_i))$ for each object *o*.
  - Calculate lower bound with L_∞ distance.
- This tuple can be stored in the Spatial Index!

Fast Matching With Spatial Indexes

Select pivots, create a tuple based on *d*.

- Similarity search:
  - M-tree & friends: slow for *d*.
  - Vectors: B-tree (1 dim) or Spatial Index.
- Trees cannot be indexed directly.
- SMAP can be used to index them:
  - Select i pivots $p_1 \ldots p_i$ from the database.
  - Create a tuple $(d(o, p_1), \ldots, d(o, p_i))$ for each object o.
  - Calculate lower bound with L_∞ distance.
- This tuple can be stored in the Spatial Index!

Fast Matching With Spatial Indexes

Select pivots, create a tuple based on *d*.

- Similarity search:
  - M-tree & friends: slow for *d*.
  - Vectors: B-tree (1 dim) or Spatial Index.
- Trees cannot be indexed directly.
- SMAP can be used to index them:
  - Select *i* pivots $p_1 \ldots p_i$ from the database.
  - Create a tuple $(d(o, p_1), \ldots, d(o, p_i))$ for each object *o*.
  - Calculate lower bound with L_∞ distance.
- This tuple can be stored in the Spatial Index!

Fast Matching With Spatial Indexes

Select pivots, create a tuple based on *d*.

- Similarity search:
  - M-tree & friends: slow for *d*.
  - Vectors: B-tree (1 dim) or Spatial Index.
- Trees cannot be indexed directly.
- SMAP can be used to index them:
  - Select *i* pivots $p_1 \ldots p_i$ from the database.
  - Create a tuple $(d(o, p_1), \ldots, d(o, p_i))$ for each object *o*.
  - Calculate lower bound with L_∞ distance.
- This tuple can be stored in the Spatial Index!

Synopsis of the Paper

Tackle performance and precision issues

- Matching one program was slow (3 days):
  - 10 hours (7x) (Tree parsing, *d*).
  - 6 min (682x) (Spatial Index + SMAP).
- Previous ranking technique was not precise:
  - 22% of the time correct.
  - Improved to 96% (IR).
- Experiments:
  - DB:1670 programs.
  - Whole program matching. Max. obfuscation.
  - Query sets: (Default 1290 100%) (ZKM 290 100%) (Sandmark 280 96%)

Synopsis of the Paper

Tackle performance and precision issues

- Matching one program was slow (3 days):
  - 10 hours (7x) (Tree parsing, *d*).
  - 6 min (682x) (Spatial Index + SMAP).
- Previous ranking technique was not precise:
  - 22% of the time correct.
  - Improved to 96% (IR).
- Experiments:
  - DB:1670 programs.
  - Whole program matching. Max. obfuscation.
  - Query sets: (Default 1290 100%) (ZKM 290 100%) (Sandmark 280 96%)

Synopsis of the Paper

Tackle performance and precision issues

- Matching one program was slow (3 days):
  - 10 hours (7x) (Tree parsing, *d*).
  - 6 min (682x) (Spatial Index + SMAP).
- Previous ranking technique was not precise:
  - 22% of the time correct.
  - Improved to 96% (IR).
- Experiments:
  - DB:1670 programs.
  - Whole program matching. Max. obfuscation.
  - Query sets: (Default 1290 100%) (ZKM 290 100%) (Sandmark 280 96%)

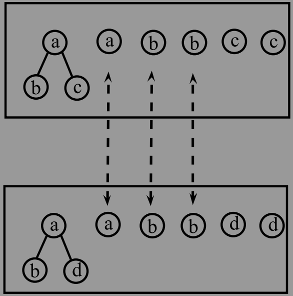
Thank you!
Contact Information

- Arnoldo Müller
  - arnoldoMuller@gmail.com
  - arnoldo@daisy.ai.kyutech.ac.jp
- For a tree (and anything else) matcher:
  - http://obsearch.berlios.de/

Distance of Fragments

Example (Trees $a(b, c)$ and $a(b, d)$)

Multi-set:



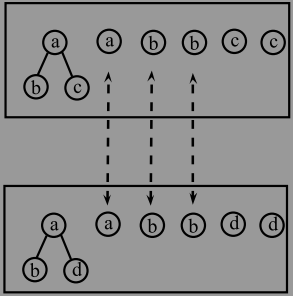
$$d(a(b,c), a(b,d)) =$$

$$\frac{(6+6)-(2\times 3)}{2} = 3$$

Distance of Fragments

Example (Trees $a(b, c)$ and $a(b, d)$)

Multi-set:



$$d(a(b, c), a(b, d)) =$$

$$\frac{(6+6)-(2\times 3)}{2} = 3$$

Matching Techniques Employed

Spatial Indexes

- Spatial Indexes work in Euclidean spaces.
- Trees cannot be indexed directly.
- SMAP can be used to index them:
  - Select i pivots $p_1 \ldots p_i$ from the database.
  - Create a tuple $(d(o, p_1), \ldots, d(o, p_i))$ for each object o.
- This tuple can be indexed in the spatial index!

Matching Techniques Employed

Spatial Indexes

- Spatial Indexes work in Euclidean spaces.
- Trees cannot be indexed directly.
- SMAP can be used to index them:
  - Select i pivots $p_1 \ldots p_i$ from the database.
  - Create a tuple $(d(o, p_1), \ldots, d(o, p_i))$ for each object o.
- This tuple can be indexed in the spatial index!

Matching Techniques Employed

Spatial Indexes

- Spatial Indexes work in Euclidean spaces.
- Trees cannot be indexed directly.
- SMAP can be used to index them:
  - Select i pivots $p_1 \ldots p_i$ from the database.
  - Create a tuple $(d(o, p_1), \ldots, d(o, p_i))$ for each object o.
- This tuple can be indexed in the spatial index!

Matching Techniques Employed

Sequential Search

- Tree creation is a heavy task.
  - Parse a string into a tree.
- Load all the trees into memory? No!
- Match the database against the query.
- Fragments of size differing in more than r ignored.

Matching Techniques Employed

Sequential Search

- Tree creation is a heavy task.
  - Parse a string into a tree.
- Load all the trees into memory? No!
- Match the database against the query.
- Fragments of size differing in more than r ignored.

Program Ranking

Naive calculation replaced by information retrieval ranking

- For an application *a* and a query *q*:
- Naive calculation: $\frac{|q \cap a|}{|a|}$ (NR).
  - "Rareness" of fragments not taken into account.
  - Distribution of the fragments in *q* and *a* ignored.
- Information retrieval techniques (IR).
  - Consider all this and more.
  - Employed Lucene (open source information retrieval software).

Program Ranking

Naive calculation replaced by information retrieval ranking

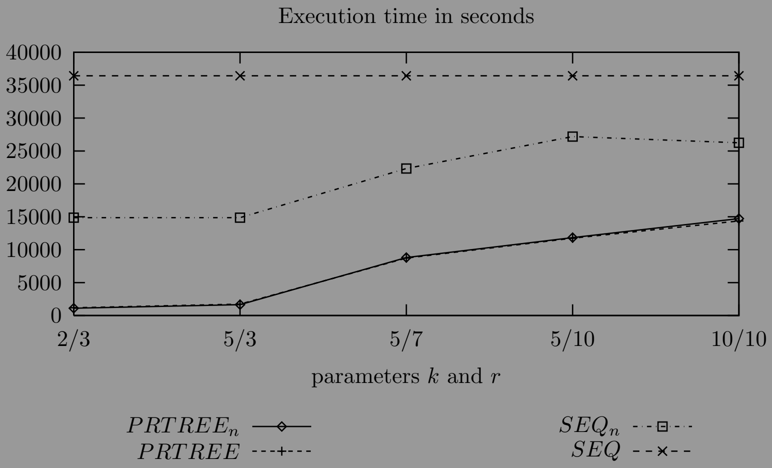
- For an application *a* and a query *q*:
- Naive calculation: $\frac{|q \cap a|}{|a|}$ (NR).
  - “Rareness” of fragments not taken into account.
  - Distribution of the fragments in *q* and *a* ignored.
- Information retrieval techniques (IR).
  - Consider all this and more.
  - Employed Lucene (open source information retrieval software).

Performance

- Database size: 340000 fragments (30 MB).
- Query size: 1641 fragments (100kb).
- Prototype written in C++.
- *PRTREE*: Spatial Index.
- *SEQ*: Sequential search.
- k : Retrieve closest k elements from DB.
- r : For query q retrieve only if $d(q, j) \leq r$ where $j \in DB$.

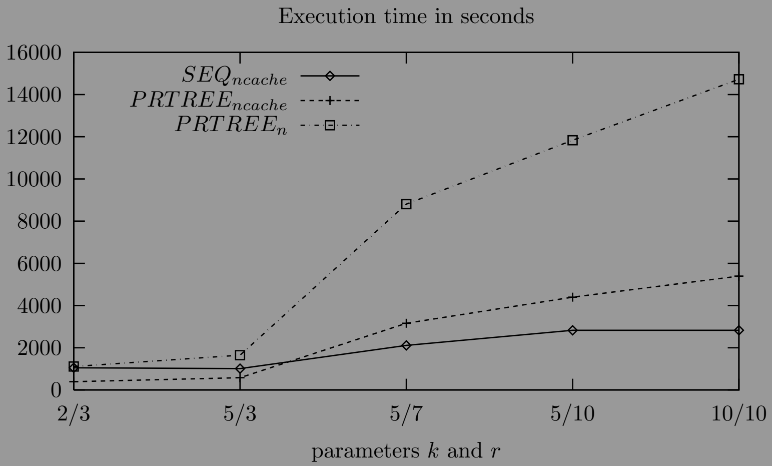
Pruning by tree size

Improvement in sequential search.



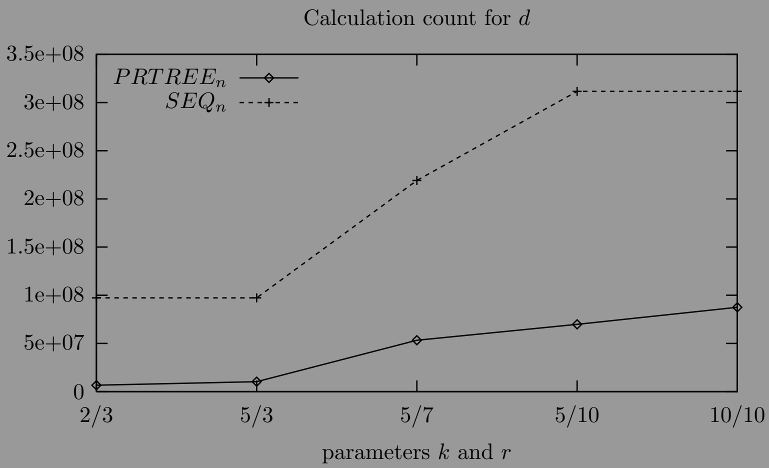
Using cache

Using cache improves considerably performance



Distance Computations

Distance computations are greatly reduced by *PRTREE*



Triangle Inequality

Some notes:

- Exploit this:
  - $|d(x,y) - d(x,z)| \leq d(y,z)$
- L_∞ for 2 vectors p and q:
  - $L_\infty = \max_i(|p_i - q_i|)$

Preliminaries

- Programs downloaded from different sources.
- Query sets constructed:
    - *A*: byte-code as it was indexed.
    - *B*: Zelix Klass Master 4.5.
    - *C*: Sandmark 3.4.

Set	Transformation	# of Programs
A	default	1293
B	Zelix	290
C	Sandmark	281

Overall Results for IR

- %X: accum. % of identifications for set X.
- $m(X)$: number of matches found for set X.

n	%A	$m(A)$	%B	$m(B)$	%C	$m(C)$
1	97.3	1259	96.8	281	87.5	246
2	98.8	19	99.6	8	90.7	9
3	99.3	7	100	1	92.1	4
4	99.8	6	–	–	93.5	4
5	99.9	1	–	–	94.6	3
6	99.9	0	–	–	94.6	0
7	99.9	0	–	–	95.0	1
8	99.9	0	–	–	95.3	1
9	100	1	–	–	95.7	1
10	–	–	–	–	96.0	1

Overall Results for NR

- $\%X$: accum. % of identifications for set X.
- $m(X)$: number of matches found for set X.

n	$\%A$	$m(A)$	$\%B$	$m(B)$	$\%C$	$m(C)$
1	18.2	236	4.4	13	9.6	27
2	33.2	194	15.8	33	12.8	9
3	49.1	206	25.5	28	14.2	4
4	59.0	127	33.4	23	16.0	5
5	65.1	80	40.0	19	17.7	5
6	69.9	61	45.5	16	18.1	1
7	73.7	50	51.0	16	19.5	4
8	77.4	48	54.1	9	20.6	3
9	80.6	41	58.2	12	21.3	2
10	83.6	38	62.4	12	22.7	4

License Violation Detection Example

Embedded open source can be detected

- Query: “ccmtools”
- Returned:
  - antlr-2.7.6-1jpp.noarch
  - antlr-2.7.6-1jpp.noarch.rpm.jpackage
  - antlr
  - ccmtools
- “antlr” is actually embedded in ccmtools

Next Steps!

- Expression normalization.
- Normalization Learning?
- Syntactically close but semantically different fragments.
- Other fragment extraction approaches.
- Detection of false negatives must be implemented.

Summary

- A very simple and new technique has been proposed.
  - Fragment + Tree-distance + Ranking.
- Performance was substantially improved
  - Use of Spatial Indexes + SMAP.
- Reliability improved
  - By using information retrieval techniques.
- Possible applications:
  - Low-level duplicated functionality detection.

Destroying our Technique

Ways of attacking our method

- Disable the extraction of SSA:
  - Dynamic fragment extraction.
- Transform the Fragments:
  - Modify assignment expressions.
  - Many fragments must be changed/added (IR).
  - Some fragments cause more damage than others.
    - Depends on IR equations (private).
    - Depends on Database (frequency, private).

Destroying our Technique

Ways of attacking our method

- Disable the extraction of SSA:
  - Dynamic fragment extraction.
- Transform the Fragments:
  - Modify assignment expressions.
  - Many fragments must be changed/added (IR).
  - Some fragments cause more damage than others.
    - Depends on IR equations (private).
    - Depends on Database (frequency, private).

Destroying our Technique

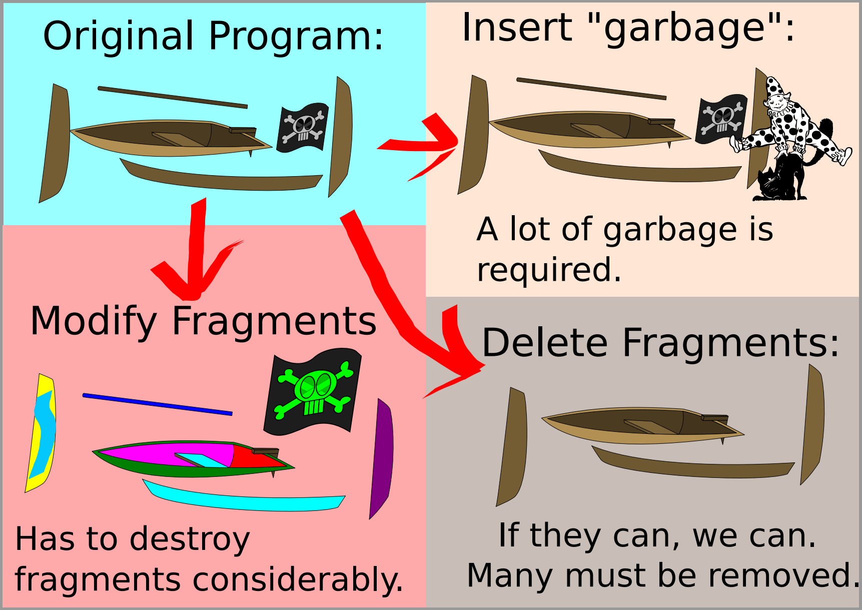
Ways of attacking our method

- Disable the extraction of SSA:
  - Dynamic fragment extraction.
- Transform the Fragments:
  - Modify assignment expressions.
  - Many fragments must be changed/added (IR).
  - Some fragments cause more damage than others.
    - Depends on IR equations (private).
    - Depends on Database (frequency, private).

Destroying our Technique

Ways of attacking our method

- Disable the extraction of SSA:
  - Dynamic fragment extraction.
- Transform the Fragments:
  - Modify assignment expressions.
  - Many fragments must be changed/added (IR).
  - Some fragments cause more damage than others.
    - Depends on IR equations (private).
    - Depends on Database (frequency, private).


Original Program:
Insert "garbage":
A lot of garbage is required.
Modify Fragments
Has to destroy fragments considerably.
Delete Fragments:
If they can, we can.
Many must be removed.

Modifying fragments

Make $d > r$

- Fragments are trees.
- We use a range r to accept 2 fragments as similar:
  - $d(a, b) \leq r$ for a, b fragments.
- Change fragments so that $d(a, b) > r$.
  - Insert r nodes (easier).
  - Delete r nodes (if they can, we can).
  - Change r nodes for others (normalization).

Destroying our technique (Summary)

Many fragments must be added/modified/deleted. (IR compensates)

- Fragment insertion:
  - New instructions must be added.
  - Many new fragments are required.
- Fragment deletion:
  - If they can, we can (static analysis).
- Fragment modification:
  - Insertion: requires *r* insertions.
    - Program can grow very much.
  - Deletion: requires *r* deletions.
    - if they can we can.
  - Replacement: term re-writing.

Replacement

For Strings is already hard

- abcdef
- ahhhef (for $r = 3$)
- When alphabet (instructions for fragments) = 30:
  - $p(30, 3) = 24360$
  - $p(30, 7) = 1.026e + 10$
- For trees the possible permutations get bigger!
- Architectures with more instructions!

Replacement

For Strings is already hard

- abcdef
- ahhhef (for $r = 3$)
- When alphabet (instructions for fragments) = 30:
  - $p(30, 3) = 24360$
  - $p(30, 7) = 1.026e + 10$
- For trees the possible permutations get bigger!
- Architectures with more instructions!