An Extensive Study on the Binary Diffing Problem

12012109 安钧文

12010336 黄慧惠

Supervisor：Prof. Yuqun Zhang

Content

- **Background**
- Related Work
- Research Questions
- Proposed Approach

Binary Diffing

- Input two binary files
- Output matches
  - Added/Deleted/Matched
- Granularity
  - Function/Basic Block/Instruction

Motivation

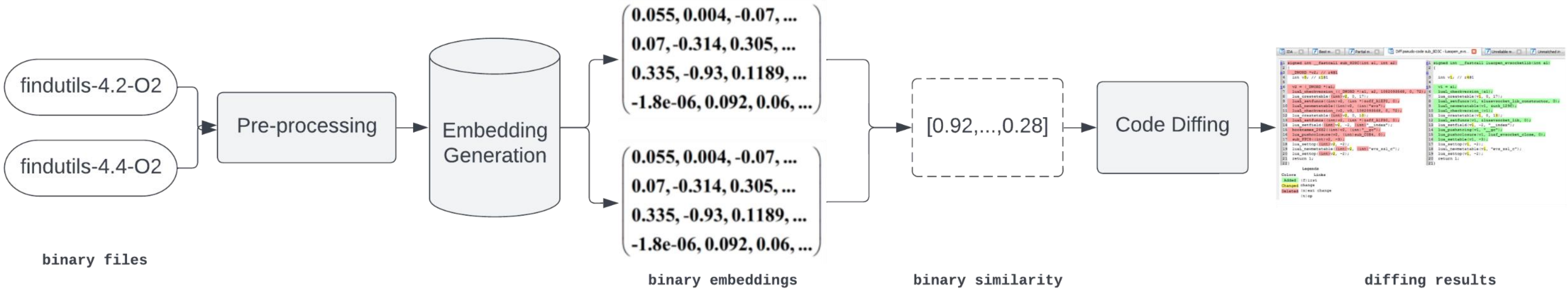
- **Efficient patching**: Reduce the size of the patch and make it faster to download and install.
- **Malware analysis**: Compare different versions of software, analyze changes and identify vulnerabilities.
- **Version control**

Traditional Approach

- Direct matching on call graphs and control-flow graphs
- Focuses on **syntax** of instructions

Learning-based Approach

- Pre-processing:
  - disassembly binary code, tokenize
- Embedding generation
- **Semantic** similarity calculation
- Code diffing
  - graph matching, network alignment



Dilemmas

SUSTech Southern University of Science and Technology

Code structure

- change the order of code execution
- change the size of code
- introduce new instructions

Time complexity: computationally expensive, especially for large files or complex software.

Content

- Background
- **Related Work**
- Research Questions
- Proposed Approach

Binary Similarity Detection

SUSTech Southern University of Science and Technology

Natural Language Processing based

- BinShot (ACSAC’ 22)
- jTrans (ISSTA’ 22)
- PalmTree (CCS’ 21)
- InnerEye (NDSS’ 19)
- Asm2Vec (S&P’ 19)

Combined

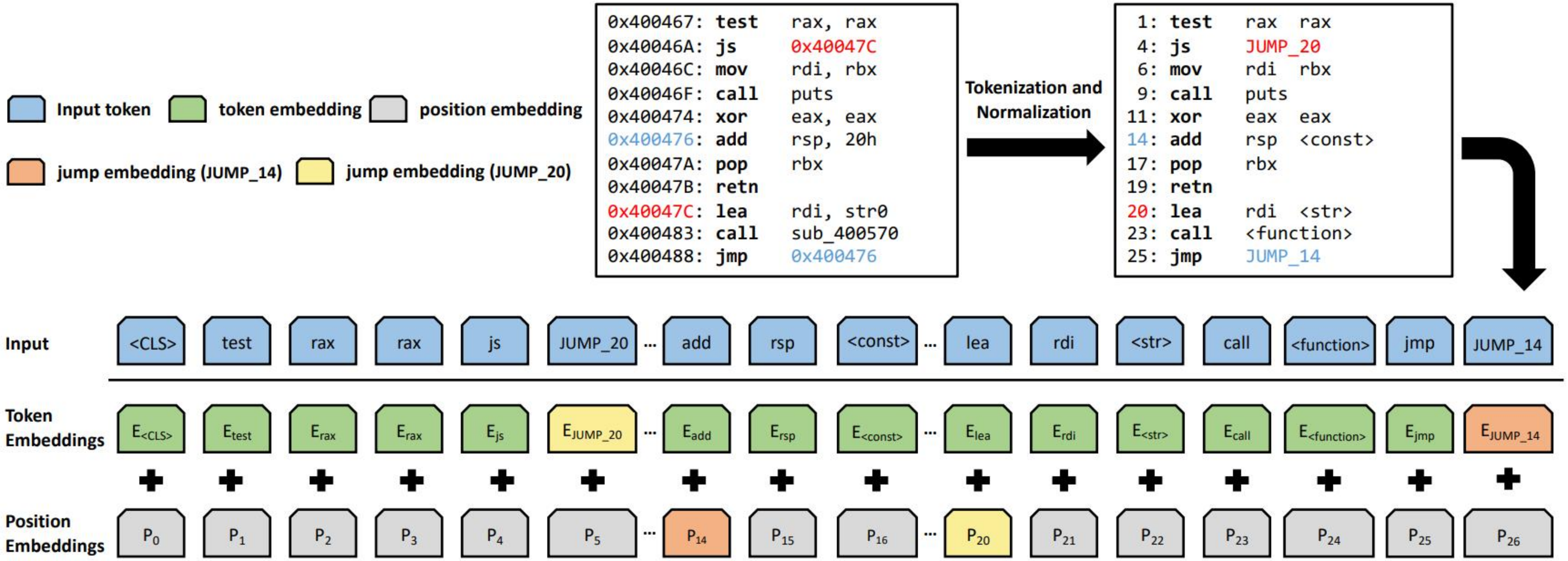
- OrderMatters (AAAI’ 20)
- DeepBinDiff (NDSS’ 20)

Graph Representation Learning based

- XBA (ISSTA’ 22)
- Asteria (DSN’ 21)
- GMN (ICML’ 19)
- VulSeeker (ASE’ 18)
- Gemini (CCS’ 17)

jTrans - BERT (ISSTA' 22)

- Model assembly instructions as natural language tokens, but with considerations of code semantic



H. Wang, W. Qu, G. Katz, W. Zhu, Z. Gao, H. Qiu, J. Zhuge, and C. Zhang, "jtrans: jump-aware transformer for binary code similarity detection,"

DeepBinDiff – Graph Embedding

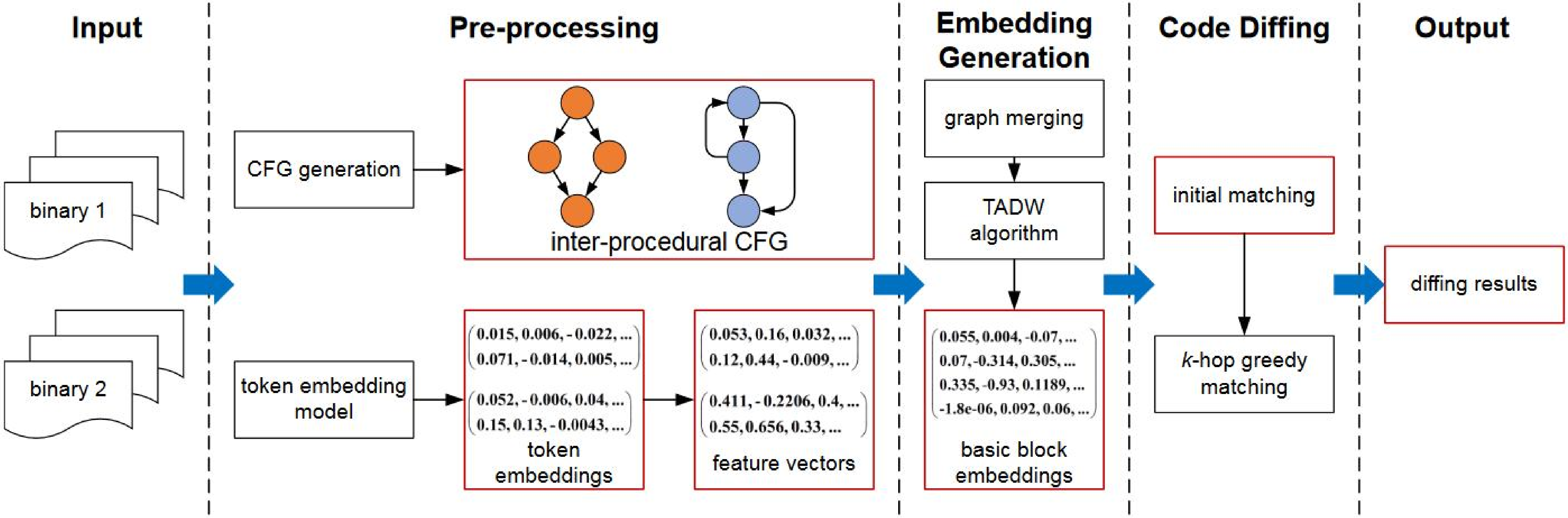


Fig. 1: Overview of DeepBinDiff.

 Yue Duan, Xuezixiang Li, Jinghan Wang, and Heng Yin. DeepBinDiff:Learning Program-Wide Code Representations for Binary Diffing.

Diffing Approaches

SUSTech Southern University of Science and Technology

- Diffing tools
  - Diaphora
  - BinDiff

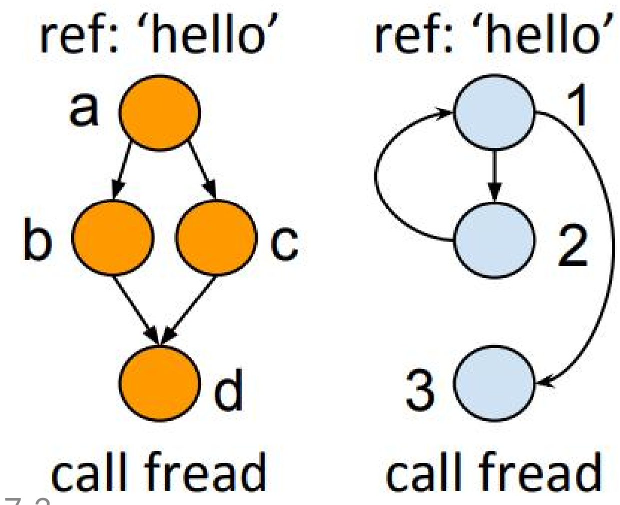
Maximum Weight Matching (MVM)	Maximum Common Edge Subgraph (MCS)	Network Alignment Problem (NAP)
• Hungarian Algorithm	• DeepBinDiff (NDSS' 20) • BinDiff	• IsoRank • QBinDiff (ASE' 21)

$$\alpha \mathbf{x}^T Q_1 \mathbf{x} + (1-\alpha)\mathbf{x}^T Q_2 \mathbf{x}.$$

Farbie Rosse, Elie Mengin.
Binary Diffing as a Network Alignment Problem via Belief Propagation

DeepBinDiff - Code Diffing

- Expand neighboring edges
- Match the most similar pairs
- Implicitly solving maximum common subgraph problem



Algorithm 1 k-Hop Greedy Matching Algorithm

$Set_{virtualnodes} \leftarrow$ {virtual nodes from merged graphs}
$Set_{initial} \leftarrow$ ComputeInitialSet($Set_{virtualnodes}$)
$Set_{matched} \leftarrow Set_{initial}$; $Set_{currPairs} \leftarrow Set_{initial}$

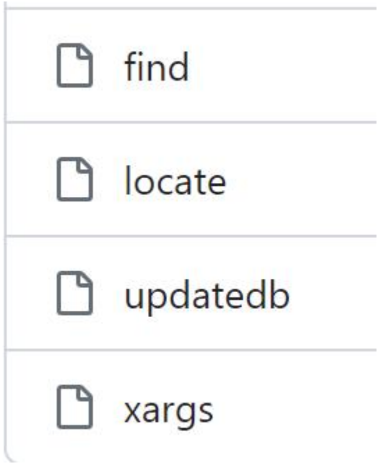
while $Set_{currPairs}$!= empty **do**
  $(node_1, node_2) \leftarrow Set_{currPairs}.pop()$
  $nb_{node_1} \leftarrow$ GetKHopNeighbors($node_1$)
  $nb_{node_2} \leftarrow$ GetKHopNeighbors($node_2$)
  $newPair \leftarrow$ FindMaxUnmatched(nb_{node_1}, nb_{node_2})
  if $newPair$!= null **then**
    $Set_{matched} \leftarrow Set_{matched} \cup newPair$
    $Set_{currPairs} \leftarrow Set_{currPairs} \cup newPair$
  end if
end while
$Set_{unreached} \leftarrow$ {basic blocks that are not yet matched}
$\{Set_m, Set_i, Set_d\} \leftarrow$ LinearAssign($Set_{unreached}$)
$Set_{matched} \leftarrow Set_{matched} \cup Set_m$
output $Set_{matched}, Set_i, Set_d$ as the diffing result

Content

- Background
- Related Work
- **Research Questions**
- Proposed Approach

Dataset

- Findutils
  - 4 binary files
  - 4 versions (2005-2022 span of 17 years)
  - Compiled with GCC v5.4, Ubuntu 18.04, O1 optimization
  - Stripped symbol table



Ground truth

- Extracted from source code, manually verified
- Mapped to function address pairs
- Focus on soundness rather than completeness
  - E.g. (00003F70, 00006320)

Experiment Setup

- Dataset
  - Findutils
- Decompilation
  - IDA pro v7.6
- Function similarity measurement
  - jTrans (ISSTA' 22)

Metrics

- Precision – exclude unknown matches
- Recall
- F1

$$Precision = \frac{||M_c||}{||M_c|| + ||M - M_u - M_c||}$$

$$Recall = \frac{||M_c||}{||G||}$$

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

$||M_c||$: True Positive

$||M - M_u - M_c||$: False Positive

$||G||$: Ground truth

Research Question 1

- **How do different diffing algorithms with the same similarity results perform?**
- Evaluation Target:
  - Hungarian Algorithm (MWM)
  - K-hop matching (MCS)
  - IsoRank (NAP)

Results

Table 1: Result of different diffing algorithms

Version	Precision			Recall			F1-score		
	Hungarian	K-hop	IsoRank	Hungarian	K-hop	IsoRank	Hungarian	K-hop	IsoRank
v4.2.33 - v4.9.0	0.894	0.909	**0.931**	0.747	**0.759**	0.684	0.814	**0.827**	0.788
v4.4.1 - v4.9.0	0.876	**0.893**	0.818	**0.795**	0.662	0.583	**0.833**	0.760	0.679
v4.6.0 - v4.9.0	**0.959**	0.942	0.884	**0.897**	0.784	0.690	**0.927**	0.856	0.775

Research Question 2

- **What are the external factors that affect diffing performance?**
  - Function Inlining
  - External function references
  - Stripped vs. Unstripped

External Function References

- Same intruction-jmp
- Same embedding content
- Could lead to random matching
- Lower presicion

.fopen

```
; Attributes: thunk

; FILE *fopen(const char *filename, const char *modes)
_fopen proc near
jmp     cs:fopen_ptr
_fopen endp
```

.error

```
; Attributes: thunk

; void error(int status, int errnum, const char *format, ...)
_error proc near
jmp     cs:error_ptr
_error endp
```

```
.strpbrk - .tolower
.__freading - .pthread_mutex_unlock
.fchdir - .iswalnum
.gnu_dev_major - .wcrtomb
.realloc - .fflush
.fdopen - .getmntent
.setlocale - .setmntent
.poll - .nl_langinfo
.strftime - .endpwent
.memmove - .sscanf
.error - .mktime
.memrchr - .strpbrk
.waitpid - .__freading
.open - .fchdir
.access - .gnu_dev_major
.fseeko - .fdopen
.fopen - .setlocale
.sysconf - .timegm
.strtoumax - .poll
```

Stripped Symbol Table

Table 2: K-hop matching performance of symbol table stripping

Version	Unstripped			Stripped		
	Precision	Recall	F1-score	Precision	Recall	F1-score
v4.2.33 - v4.9.0	0.923	0.732	0.807	0.909	0.759	0.829
v4.4.1 - v4.9.0	0.901	0.739	0.772	0.893	0.662	0.760
v4.6.0 - v4.9.0	0.946	0.810	0.844	0.942	0.784	0.856

Table 3: Diaphora performance of symbol table stripping

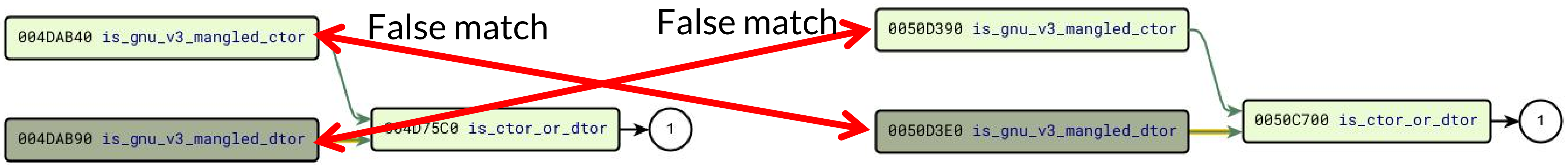
Version	Unstripped			Stripped		
	Precision	Recall	F1-score	Precision	Recall	F1-score
v4.2.33 - v4.9.0	0.955	0.928	0.941	0.947	0.455	0.615
v4.4.1 - v4.9.0	0.958	0.931	0.944	0.940	0.523	0.672
4v.6.0 - v4.9.0	0.958	0.930	0.943	0.980	0.661	0.791

Research Question 3

- **What is the cause of false positive results?**
- Evaluation Target:
  - Hungarian Algorithm (MWM)
  - K-hop matching (MCS)
  - IsoRank (NAP)

Hungarian Algorithm

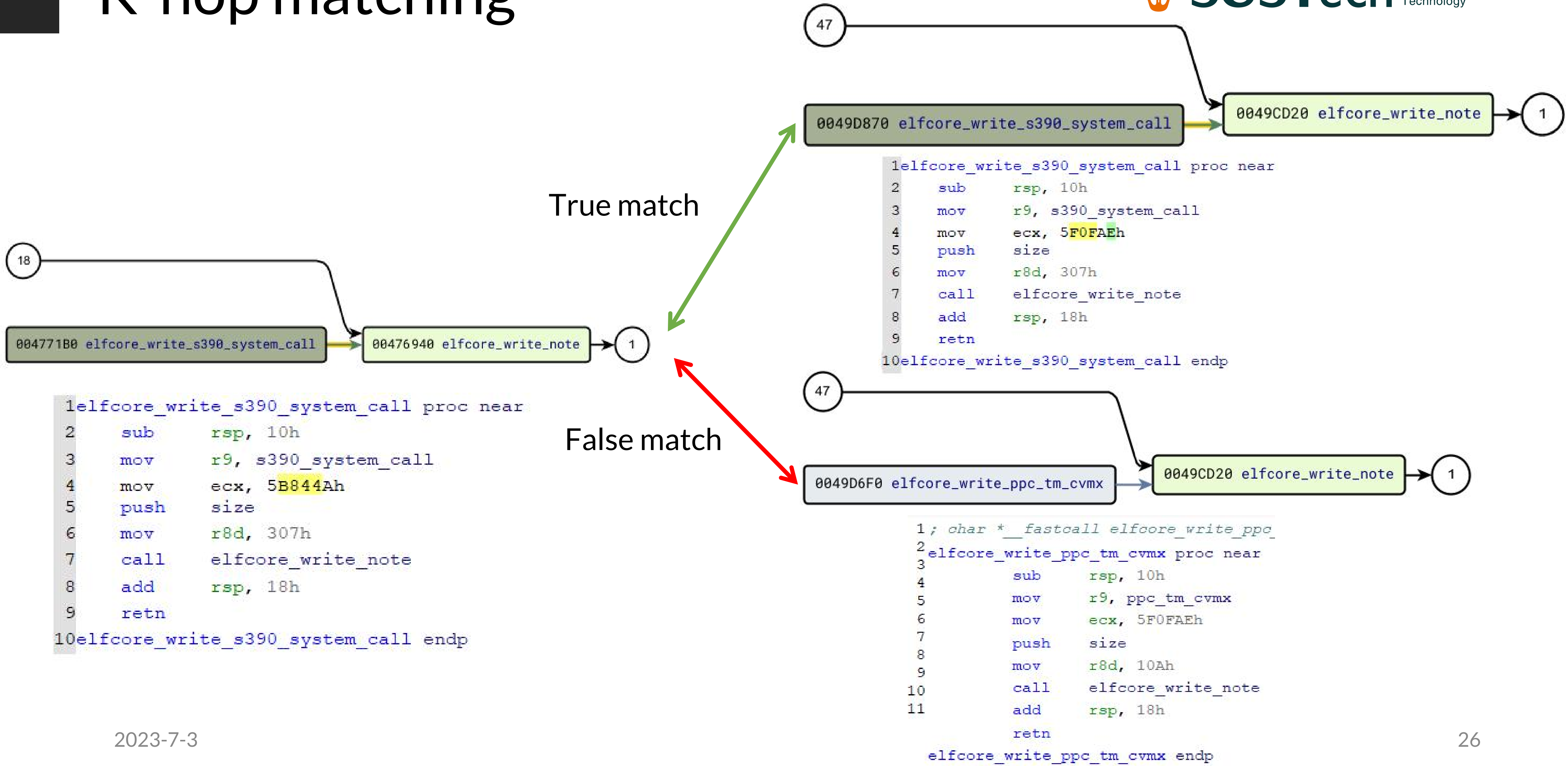
Wrongly cross-match



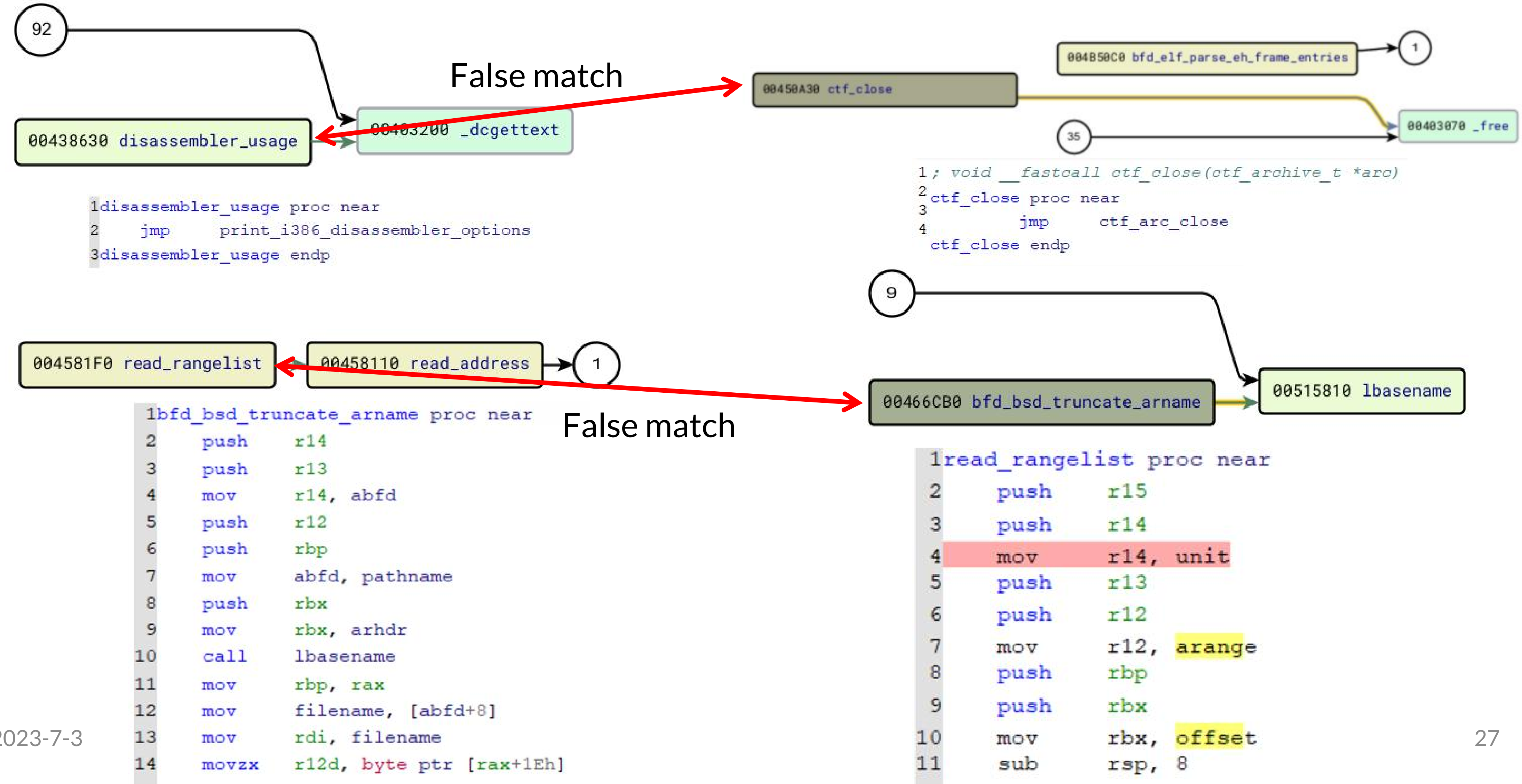

```
is_gnu_v3_mangled_ctor proc near
    sub     rsp, 18h
    lea     rdx, [rsp+4]
    mov     rsi, rsp
    mov     rax, fs:28h
    mov     [rsp+8], rax
    xor     eax, eax
    call    is_ctor_or_dtor
    xor     edx, edx
    test    eax, eax
    cmovnz  edx, [rsp]
    mov     rcx, [rsp+8]
    xor     rcx, fs:28h
    jnz     short loc_4DAB80
loc_4dab79:
    mov     eax, edx
    add     rsp, 18h
    retn
loc_4dab80:
    call    ___stack_chk_fail
is_gnu_v3_mangled_ctor endp
```

```
is_gnu_v3_mangled_dtor proc near
    sub     rsp, 18h
    lea     rdx, [rsp+4]
    mov     rsi, rsp
    mov     rax, fs:28h
    mov     [rsp+8], rax
    xor     eax, eax
    call    is_ctor_or_dtor
    xor     edx, edx
    test    eax, eax
    cmovnz  edx, [rsp+4]
    mov     rcx, [rsp+8]
    xor     rcx, fs:28h
    jnz     short loc_50D421
loc_50d41a:
    mov     eax, edx
    add     rsp, 18h
    retn
loc_50d421:
    call    ___stack_chk_fail
is_gnu_v3_mangled_dtor endp
```

K-hop matching



IsoRank



92
00438630 disassembler_usage
00403200 _dcgettext
False match
00450A30 ctf_close
004B50C0 bfd_elf_parse_eh_frame_entries
1
35
00403070 _free
disassembler_usage proc near
jmp print_i386_disassembler_options
disassembler_usage endp
; void __fastcall ctf_close(ctf_archive_t *arc)
ctf_close proc near
jmp ctf_arc_close
ctf_close endp
9
004581F0 read_rangelist
00458110 read_address
1
00466CB0 bfd_bsd_truncate_arname
00515810 lbasename
False match
bfd_bsd_truncate_arname proc near
push r14
push r13
mov r14, abfd
push r12
push rbp
mov abfd, pathname
push rbx
mov rbx, arhdr
call lbasename
mov rbp, rax
mov filename, [abfd+8]
mov rdi, filename
movzx r12d, byte ptr [rax+1Eh]
read_rangelist proc near
push r15
push r14
mov r14, unit
push r13
push r12
mov r12, arange
push rbp
push rbx
mov rbx, offset
sub rsp, 8

Content

- Background
- Related Work
- Research Questions
- **Proposed Approach**

Approach

SUSTech Southern University of Science and Technology

- **Intuition**:
  - Across version changes, major functions are preserved and easily distinguishable
  - Code changes tend to happen around major functions (e.g., refactoring)
  - Instead of aligning from external function references (as in DeepBindiff), start directly inside the call graph

Approach

- **Seed-and-Extend**
- Combine idea of both Hungarian algorithm and K-hop matching
- Leverage both node and network information
- Use heuristic to approximate global network alignment
  - Utilize as much starting points as possible

Approach

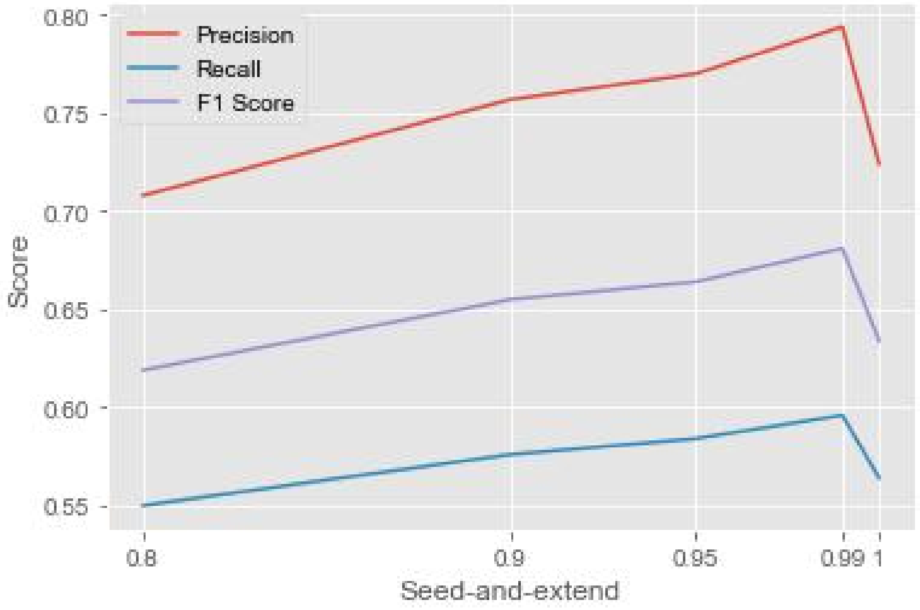
- **Step 1** – Find initial alignment based on similarity threshold θ
  - match distinguishable major functions
- **Step 2 –** K-hop matching on initial alignments
  - match “hard” cases with graph’s assistance
- K = 2 due to dense call graph

Algorithm 1 Seed-and-Extend

procedure $\textsc{SeedAndExtend}(G_1, G_2, \theta, k)$
    $initialAlignment \leftarrow \emptyset$
    for $n_1 \in G_1$ **do**
        $initialPair \leftarrow \text{findMaxSim}(G_2, \theta)$
        if $initialPair! = null$ **then**
            $initialAlignment.\text{add}(initialPair)$
        end if
    end for
    $currentAligned, resultAligned \leftarrow initialAlignment$
    while $currentAligned$ is not empty **do**
        $i, j \leftarrow currentAligned.\text{pop}()$
        $neighbors_i \leftarrow \text{findTwoHopNeighbors}(i)$
        $neighbors_j \leftarrow \text{findTwoHopNeighbors}(j)$
        $maxPair \leftarrow \text{findMaxUnmatched}(neighbors_i, neighbors_j)$
        $resultAlignment.\text{add}(maxPair)$
    end while
    return $resultAligned$
end procedure

Evaluation

- **How to determine the threshold?**
- Use objdump as evaluation binary
  - Call graph is more complex
  - Function number is bigger
- Experimented on various θ value



Evaluation

- **Performance of Seed-and-Extend**
  - Competent performance on both software
  - Better performance on bigger scale binary

Binary	Version	Precision				Recall				F1-score			
		Hungarian	K-hop	IsoRank	S-E	Hungarian	K-hop	IsoRank	S-E	Hungarian	K-hop	IsoRank	S-E
findutils	v4.2.33 - v4.9.0	0.894	0.909	**0.931**	0.882	0.747	**0.759**	0.684	**0.759**	0.814	**0.827**	0.788	0.816
	v4.4.1 - v4.9.0	0.876	0.893	0.818	**0.896**	**0.795**	0.662	0.583	**0.795**	0.833	0.760	0.679	**0.842**
	v4.6.0 - v4.9.0	**0.959**	0.942	0.884	0.941	**0.897**	0.784	0.690	0.874	**0.927**	0.856	0.775	0.906
objdump	v2.25 - v2.40	0.690	0.719	0.857	**0.794**	0.544	0.521	0.504	**0.596**	0.608	0.623	0.635	**0.681**

Evaluation

- **What are possible drawbacks?**
- Heavily depends on similarity results
- Unable to match similar functions with identical call graph

References

[1] Zeping Yu, Rui Cao, Qiyi Tang, Sen Nie, Junzhou ouang, and Shi Wu. Order Matters: Semantic-Aware Neural Networks for Binary Code Similarity Detection. Proceedings of the AAAI Conference on Artificial Intelligence, 34(01):1145–1152, April 2020.
[2] H. Wang, W. Qu, G. Katz, W. Zhu, Z. Gao, H. Qiu, J. Zhuge, and C. Zhang, "jtrans: jump-aware transformer for binary code similarity detection," in Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis, 2022, pp. 1–13
[3] Yue Duan, Xuezixiang Li, Jinghan Wang, and Heng Yin. DeepBinDiff: Learning Program-Wide Code Representations for Binary Diffing. In Proceedings of Network and Distributed System Security Symposium, 2020.
[4] E. Mengin and F. Rossi, "Binary Diffing as a Network Alignment Problem via Belief Propagation," 2021 36th IEEE/ACM International Conference on Auto- mated Software Engineering (ASE), Melbourne, Australia, 2021, pp. 967-978, doi: 10.1109/ASE51524.2021.9678782.
[5] F. Zuo, X. Li, P. Young, L. Luo, Q. Zeng, and Z. Zhang, "Neural Machine Translation Inspired Binary Code Similarity Comparison beyond Function Pairs," in Proceedings 2019 Network and Distributed System Security Symposium. Reston, VA: Internet Society, 2019
[6] F. Zuo, X. Li, P. Young, L. Luo, Q. Zeng, and Z. Zhang, "Neural Machine Translation Inspired Binary Code Similarity Comparison beyond Function Pairs," in Proceedings 2019 Network and Distributed System Security Symposium. Reston, VA: Internet Society, 2019.
[7] Harold W. Kuhn, "The Hungarian Method for the assignment problem", Naval Re-
search Logistics Quarterly, 2: 83–97, 1955. Kuhn's original publication.