Applications of Graph Integration to Function Comparison and Malware Classification

Michael Slawinski, Staff Data Scientist
Andy Wortman, Research Engineer Associate Principal

Oct 25, 2019

Cylance Inc.

Agenda

1. Overview of our Vectorization Method
2. The .NET Framework and Common Language Runtime (CLR)
3. Decompilation
4. Graph Integration
5. Results

Overview of our Vectorization Method

Summary

Summary

Overall Goal

Construct a vectorization method to be leveraged by a classifier on .NET files

Summary

Overall Goal

Construct a vectorization method to be leveraged by a classifier on .NET files

Our Vectorization Method - an overview

Summary

Overall Goal

Construct a vectorization method to be leveraged by a classifier on .NET files

Our Vectorization Method - an overview

1. Decompile(file) $\longrightarrow \{G\}$

Summary

Overall Goal

Construct a vectorization method to be leveraged by a classifier on .NET files

Our Vectorization Method - an overview

1. Decompile(file) $\longrightarrow \{G\}$
2. Define a set of functions $\{f : \mathrm{Vert}(G) \longrightarrow \mathbb{R}\}$ applicable to every possible G

Summary

Overall Goal

Construct a vectorization method to be leveraged by a classifier on .NET files

Our Vectorization Method - an overview

1. Decompile(file) $\longrightarrow \{G\}$
2. Define a set of functions $\{f : \mathrm{Vert}(G) \longrightarrow \mathbb{R}\}$ applicable to every possible G
3. Compute antiderivatives of functions defined in previous step

Summary

Overall Goal

Construct a vectorization method to be leveraged by a classifier on .NET files

Our Vectorization Method - an overview

1. Decompile(file) $\longrightarrow \{G\}$
2. Define a set of functions $\{f : \mathrm{Vert}(G) \longrightarrow \mathbb{R}\}$ applicable to every possible G
3. Compute antiderivatives of functions defined in previous step
4. Compute component-wise mean/std of antiderivatives across all G resulting from decompilation of the given file

The .NET Framework and Common Language Runtime (CLR)

.NET Framework - two main components

.NET Framework - two main components

Framework Class Library (FCL)

- user interface
- data access
- database connectivity
- cryptography
- web application development

.NET Framework - two main components

Framework Class Library (FCL)

- user interface
- data access
- database connectivity
- cryptography
- web application development

Common Language Runtime (CLR)

.NET Framework - two main components

Framework Class Library (FCL)

- user interface
- data access
- database connectivity
- cryptography
- web application development

Common Language Runtime (CLR)

- is an application virtual machine which provides

.NET Framework - two main components

Framework Class Library (FCL)

- user interface
- data access
- database connectivity
- cryptography
- web application development

Common Language Runtime (CLR)

- is an application virtual machine which provides
  - security, memory management, exception handling

.NET Framework - two main components

Framework Class Library (FCL)

- user interface
- data access
- database connectivity
- cryptography
- web application development

Common Language Runtime (CLR)

- is an application virtual machine which provides
    - security, memory management, exception handling
- compilation of high-level .NET code results in an Intermediate Language Binary

.NET Framework - two main components

Framework Class Library (FCL)

- user interface
- data access
- database connectivity
- cryptography
- web application development

Common Language Runtime (CLR)

- is an application virtual machine which provides
  - security, memory management, exception handling
- compilation of high-level .NET code results in an Intermediate Language Binary
- the CLR JITs the code from IL to machine code run on the cpu

Decompilation

Decompilation

Decompilation

Definition

Decompilation is a program transformation by which compiled code is transformed into a high-level human-readable form.

Decompilation

Definition

Decompilation is a program transformation by which compiled code is transformed into a high-level human-readable form.

Definition

An *Abstract Syntax Tree* is a tree representation of the abstract syntactic structure of the source code, where each node denotes a construct occurring in the source code.

Decompilation

Definition

Decompilation is a program transformation by which compiled code is transformed into a high-level human-readable form.

Definition

An *Abstract Syntax Tree* is a tree representation of the abstract syntactic structure of the source code, where each node denotes a construct occurring in the source code.

Program control flow is understood by studying the structure of two types of control flow graphs resulting from decompilation.

Decompilation

Definition

Decompilation is a program transformation by which compiled code is transformed into a high-level human-readable form.

Definition

An *Abstract Syntax Tree* is a tree representation of the abstract syntactic structure of the source code, where each node denotes a construct occurring in the source code.

Program control flow is understood by studying the structure of two types of control flow graphs resulting from decompilation.

- the function call graph describes the calling structure of the functions (subroutines) constituting the overall program

Decompilation

Definition

Decompilation is a program transformation by which compiled code is transformed into a high-level human-readable form.

Definition

An *Abstract Syntax Tree* is a tree representation of the abstract syntactic structure of the source code, where each node denotes a construct occurring in the source code.

Program control flow is understood by studying the structure of two types of control flow graphs resulting from decompilation.

- the function call graph describes the calling structure of the functions (subroutines) constituting the overall program
- Shortsighted Data Flow Graphs (SDFG) - each obtained by merging all paths through the AST corresponding to a constituent function

Abstract Syntax Trees

Abstract Syntax Trees

Example: Arithmetic Expressions

Abstract Syntax Trees

Example: Arithmetic Expressions

Consider the following BinaryOp expression:

$$5 * 3 + (4 + 2 \ \% \ 2 * 8)$$

Abstract Syntax Trees

Example: Arithmetic Expressions

Consider the following BinaryOp expression:

$$5 * 3 + (4 + 2 \ \% \ 2 * 8)$$

The semantic structure of this expression can be distilled by considering the following binary tree:

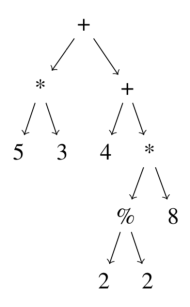
Abstract Syntax Trees

Example: Arithmetic Expressions

Consider the following BinaryOp expression:

$$5 * 3 + (4 + 2 \ \% \ 2 * 8)$$

The semantic structure of this expression can be distilled by considering the following binary tree:

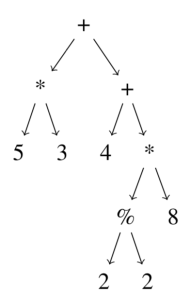


Example: Arithmetic Expressions

Consider the following BinaryOp expression:

$$5 * 3 + (4 + 2 \ \% \ 2 * 8)$$

The semantic structure of this expression can be distilled by considering the following binary tree:



Distilled semantic structure = order of operations

CLR AST Dictionary - CLR-Specific or C#-specific AST Members in Blue

CLR AST Dictionary - CLR-Specific or C#-specific AST Members in Blue

Control Flow

- if - reference the conditional and execute accordingly
- break - immediately exit the enclosing loop
- CLRWhile - infinite loop

CLR AST Dictionary - CLR-Specific or C#-specific AST Members in Blue

Control Flow

- if - reference the conditional and execute accordingly
- break - immediately exit the enclosing loop
- CLRWhile - infinite loop

Expressions

Code that when evaluated **does** yield a value. Valid in places such as tests, for loops, conditionals, or as the right-hand side of assignments.

- BinaryOp - expression computed from two operands and some operator
- Call - function call, including list of args pass to the function

CLR AST Dictionary - CLR-Specific or C#-specific AST Members in Blue

Control Flow

- if - reference the conditional and execute accordingly
- break - immediately exit the enclosing loop
- CLRWhile - infinite loop

Expressions

Code that when evaluated **does** yield a value. Valid in places such as tests, for loops, conditionals, or as the right-hand side of assignments.

- BinaryOp - expression computed from two operands and some operator
- Call - function call, including list of args pass to the function

Statements

Code that when evaluated **does not** yield a value. E.g., a statement cannot be on the right-hand side of an assignment.

- Assignment - storage of rh variable to the location yielded by lh variable
- CLRVariableWithInitializer - declaration and subsequent initialization

CLR-Specific or C#-specific AST Members

Construction of Shortsighted Data Flow Graph

Construction of Shortsighted Data Flow Graph

Small code block resulting in a nonlinear SDFG.

Construction of Shortsighted Data Flow Graph

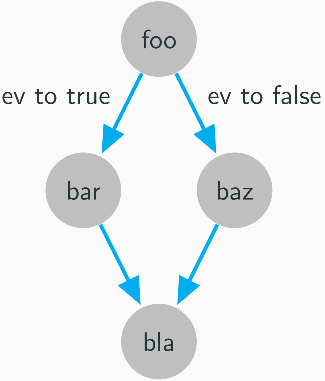
Small code block resulting in a nonlinear SDFG.

```
if foo() {
        bar();
}
else {
        baz();
}
bla();
```

Construction of Shortsighted Data Flow Graph

Small code block resulting in a nonlinear SDFG.

```
if foo() {
        bar();
}
else {
        baz();
}
bla();
```



Functions on SDFG Graphs - Motivation

Functions on SDFG Graphs - Motivation

We often study an object X by studying a set of functions defined on X

$$X^* := \{f : X \longrightarrow \mathbb{R}\}$$

Functions on SDFG Graphs - Motivation

We often study an object X by studying a set of functions defined on X

$$X^* := \{f : X \longrightarrow \mathbb{R}\}$$

Example 1

Consider the case of a distribution $\mathcal{D}$ on a sample space Ω defined by the measure μ. We might choose to study $\mathcal{D}$ by studying

$$f_n : \mathcal{D} \mapsto \int_\Omega x^n d\mu(x)$$

Functions on SDFG Graphs - Motivation

We often study an object X by studying a set of functions defined on X

$$X^* := \{f : X \longrightarrow \mathbb{R}\}$$

Example 1

Consider the case of a distribution $\mathcal{D}$ on a sample space Ω defined by the measure μ. We might choose to study $\mathcal{D}$ by studying

$$f_n : \mathcal{D} \mapsto \int_\Omega x^n d\mu(x)$$

Example 2

Consider the set of invertible $n \times n$ matrices $GL_n(\mathbb{F})$ on some field $\mathbb{F}$. We might choose to study $GL_n(\mathbb{F})$ by studying

$$\mathrm{tr}, \det : GL_n(\mathbb{F}) \longrightarrow \mathbb{R}$$

Functions on SDFG Graphs

Functions on SDFG Graphs

Let G be a SDFG graph resulting from traversing a given AST corresponding to some source code function.

Functions on SDFG Graphs

Let G be a SDFG graph resulting from traversing a given AST corresponding to some source code function.

Example

Functions on SDFG Graphs

Let G be a SDFG graph resulting from traversing a given AST corresponding to some source code function.

Example

Define

$$\mathsf{NumPass2Call} : \mathrm{Vert}(G) \longrightarrow \mathbb{R}$$

Functions on SDFG Graphs

Let G be a SDFG graph resulting from traversing a given AST corresponding to some source code function.

Example

Define

$$\mathsf{NumPass2Call} : \mathrm{Vert}(G) \longrightarrow \mathbb{R}$$

by

$$v \mapsto \#\mathsf{args}_v$$

Functions on SDFG Graphs

Let G be a SDFG graph resulting from traversing a given AST corresponding to some source code function.

Example

Define

$$\mathsf{NumPass2Call} : \mathrm{Vert}(G) \longrightarrow \mathbb{R}$$

by

$$v \mapsto \#\mathsf{args}_v$$

where $\#\mathsf{args}_v$ is the number of arguments passed to the function called at v.

Functions on SDFG Graphs

Let G be a SDFG graph resulting from traversing a given AST corresponding to some source code function.

Example

Define

$$\text{NumPass2Call} : \mathrm{Vert}(G) \longrightarrow \mathbb{R}$$

by

$$v \mapsto \#\text{args}_v$$

where $\#\text{args}_v$ is the number of arguments passed to the function called at v.

Other Examples:

Functions on SDFG Graphs

Let G be a SDFG graph resulting from traversing a given AST corresponding to some source code function.

Example

Define

$$\text{NumPass2Call} : \mathrm{Vert}(G) \longrightarrow \mathbb{R}$$

by

$$v \mapsto \#\text{args}_v$$

where $\#\text{args}_v$ is the number of arguments passed to the function called at v.

Other Examples:

1. $\text{BinaryOp} : v \mapsto \eta(\text{whichOpCode}_v)$

for some string-to-float hash function η.

Functions on SDFG Graphs

Let G be a SDFG graph resulting from traversing a given AST corresponding to some source code function.

Example

Define

$$\text{NumPass2Call} : \mathrm{Vert}(G) \longrightarrow \mathbb{R}$$

by

$$v \mapsto \#\text{args}_v$$

where $\#\text{args}_v$ is the number of arguments passed to the function called at v.

Other Examples:

1. $\text{BinaryOp} : v \mapsto \eta(\text{whichOpCode}_v)$
2. $\text{CLRClassRef} : v \mapsto \eta(\text{ReferencedClass}_v)$

for some string-to-float hash function η.

Graph Antiderivative - Ingredients

Graph Antiderivative - Ingredients

In order to define an integral of a function

$$f : \mathrm{Vert}(G) \longrightarrow \mathbb{R}$$

for G a directed graph, we must define a measure μ on $\mathrm{Vert}(G)$ in such a way that f is measurable.

Graph Antiderivative - Ingredients

In order to define an integral of a function

$$f : \mathrm{Vert}(G) \longrightarrow \mathbb{R}$$

for G a directed graph, we must define a measure μ on $\mathrm{Vert}(G)$ in such a way that f is measurable.

We do this by imposing a Markov chain structure on G and taking μ to be the PageRank measure

$$\mathbb{P} : \mathrm{Vert}(G) \longrightarrow [0, 1]$$

$$v \mapsto \mathrm{PageRank}(G)_v$$

Graph Antiderivative - Ingredients

In order to define an integral of a function

$$f : \mathrm{Vert}(G) \longrightarrow \mathbb{R}$$

for G a directed graph, we must define a measure μ on $\mathrm{Vert}(G)$ in such a way that f is measurable.

We do this by imposing a Markov chain structure on G and taking μ to be the PageRank measure

$$\mathbb{P} : \mathrm{Vert}(G) \longrightarrow [0, 1]$$

$$v \mapsto \mathrm{PageRank}(G)_v$$

where the PageRank vector is taken to be the steady-state probability distribution over the nodes resulting from the long-run behavior of the random-walk Markov Chain.

Graph Integration

Markov Chains and the PageRank Vector

Markov Chains and the PageRank Vector

Definition

A discrete-time *Markov chain* is a sequence of random variables $X_1, X_2, \ldots$ such that

$$P(X_{n+1} = x | X_1 = x_1, \ldots, X_n = x_n) = P(X_{n+1} = x | X_n = x_n)$$

Markov Chains and the PageRank Vector

Definition

A discrete-time *Markov chain* is a sequence of random variables $X_1, X_2, \ldots$ such that

$$P(X_{n+1} = x | X_1 = x_1, \ldots, X_n = x_n) = P(X_{n+1} = x | X_n = x_n)$$

Given G, order the vertices $\{v_i\}$ of the graph G and define the $n \times n$ probability transition matrix T by

$$t_{ij} = \begin{cases} 1/|v_i^{\mathrm{out}}| & \textit{if } (v_i, v_j) \in \mathrm{Edges}(G) \\ 0 & \text{otherwise} \end{cases}$$

where v_i^{out} is the set of edges emanating from vertex v_i and $n = |\mathrm{Vert}(G)|$.

Markov Chains and the PageRank Vector

Definition

A discrete-time *Markov chain* is a sequence of random variables $X_1, X_2, \ldots$ such that

$$P(X_{n+1} = x | X_1 = x_1, \ldots, X_n = x_n) = P(X_{n+1} = x | X_n = x_n)$$

Given G, order the vertices $\{v_i\}$ of the graph G and define the $n \times n$ probability transition matrix T by

$$t_{ij} = \begin{cases} 1/|v_i^{\mathrm{out}}| & \textit{if } (v_i, v_j) \in \mathrm{Edges}(G) \\ 0 & \text{otherwise} \end{cases}$$

where v_i^{out} is the set of edges emanating from vertex v_i and $n = |\mathrm{Vert}(G)|$.

To ensure the irreducibility of our transition matrix, we smooth T to

Markov Chains and the PageRank Vector

Definition

A discrete-time *Markov chain* is a sequence of random variables $X_1, X_2, \ldots$ such that

$$P(X_{n+1} = x | X_1 = x_1, \ldots, X_n = x_n) = P(X_{n+1} = x | X_n = x_n)$$

Given G, order the vertices $\{v_i\}$ of the graph G and define the $n \times n$ probability transition matrix T by

$$t_{ij} = \begin{cases} 1/|v_i^{\mathrm{out}}| & \textit{if } (v_i, v_j) \in \mathrm{Edges}(G) \\ 0 & \text{otherwise} \end{cases}$$

where v_i^{out} is the set of edges emanating from vertex v_i and $n = |\mathrm{Vert}(G)|$.

To ensure the irreducibility of our transition matrix, we smooth T to

$$M = (1 - p)T + pB \qquad \text{(Perron-Frobenius)}$$

Markov Chains and the PageRank Vector

Definition

A discrete-time *Markov chain* is a sequence of random variables $X_1, X_2, \ldots$ such that

$$P(X_{n+1} = x | X_1 = x_1, \ldots, X_n = x_n) = P(X_{n+1} = x | X_n = x_n)$$

Given G, order the vertices $\{v_i\}$ of the graph G and define the $n \times n$ probability transition matrix T by

$$t_{ij} = \begin{cases} 1/|v_i^{\mathrm{out}}| & \text{if } (v_i, v_j) \in \mathrm{Edges}(G) \\ 0 & \text{otherwise} \end{cases}$$

where v_i^{out} is the set of edges emanating from vertex v_i and $n = |\mathrm{Vert}(G)|$.

To ensure the irreducibility of our transition matrix, we smooth T to

$$M = (1-p)T + pB \qquad \text{(Perron-Frobenius)}$$

where

$$B = \frac{1}{n}\begin{bmatrix} 1 & 1 & \ldots \\ \vdots & \ddots & \\ 1 & & 1 \end{bmatrix}$$

PageRank Measure

PageRank Measure

The PageRank vector $\mathbb{P}$ is given by the left eigenvector of M and corresponds to

PageRank Measure

The PageRank vector $\mathbb{P}$ is given by the left eigenvector of M and corresponds to

$$\lim_{n \to \infty} M^n \frac{1}{|\mathrm{Vert}(G)|} \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}$$

PageRank Measure

The PageRank vector $\mathbb{P}$ is given by the left eigenvector of M and corresponds to

$$\lim_{n\to\infty} M^n \frac{1}{|\mathrm{Vert}(G)|} \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}$$

and can usually be adequately approximated with $n = 10$.

PageRank Measure

The PageRank vector $\mathbb{P}$ is given by the left eigenvector of M and corresponds to

$$\lim_{n \to \infty} M^n \frac{1}{|\mathrm{Vert}(G)|} \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}$$

and can usually be adequately approximated with $n = 10$.

The corresponding Markov chain is defined by

PageRank Measure

The PageRank vector $\mathbb{P}$ is given by the left eigenvector of M and corresponds to

$$\lim_{n \to \infty} M^n \frac{1}{|\mathrm{Vert}(G)|} \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}$$

and can usually be adequately approximated with $n = 10$.

The corresponding Markov chain is defined by

$$P(X_t = v_i | X_{t-1} = v_j) = (1-p)t_{ij} + p\frac{1}{n}$$

PageRank Measure

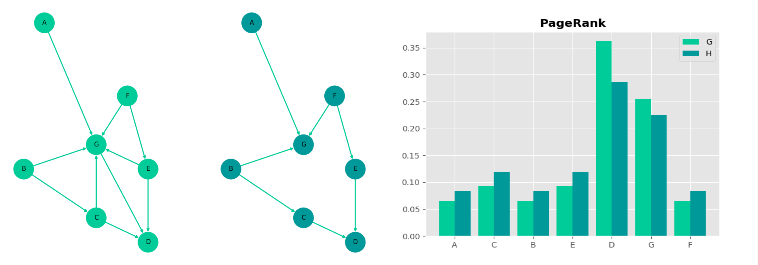
The PageRank vector $\mathbb{P}$ is given by the left eigenvector of M and corresponds to

$$\lim_{n\to\infty} M^n \frac{1}{|\mathrm{Vert}(G)|} \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}$$

and can usually be adequately approximated with $n = 10$.

The corresponding Markov chain is defined by

$$P(X_t = v_i | X_{t-1} = v_j) = (1-p)t_{ij} + p\frac{1}{n}$$



Construction of the Graph Integral

Construction of the Graph Integral

Consider a function $f : \mathrm{Vert}(G) \longrightarrow \mathbb{R}$ for G a finite directed graph.

Construction of the Graph Integral

Consider a function $f : \mathrm{Vert}(G) \longrightarrow \mathbb{R}$ for G a finite directed graph.

Let $\mathbb{P} = \{p_v\}$ be the PageRank measure on $\mathrm{Vert}(G)$. We can then define a measure ν_f on $\mathrm{Vert}(G)$ by

Construction of the Graph Integral

Consider a function $f : \mathrm{Vert}(G) \longrightarrow \mathbb{R}$ for G a finite directed graph.

Let $\mathbb{P} = \{p_v\}$ be the PageRank measure on $\mathrm{Vert}(G)$. We can then define a measure ν_f on $\mathrm{Vert}(G)$ by

$$\nu_f(S) = \int_S f \, d\mathbb{P}$$

Construction of the Graph Integral

Consider a function $f : \mathrm{Vert}(G) \longrightarrow \mathbb{R}$ for G a finite directed graph.

Let $\mathbb{P} = \{p_v\}$ be the PageRank measure on $\mathrm{Vert}(G)$. We can then define a measure ν_f on $\mathrm{Vert}(G)$ by

$$\begin{aligned}\nu_f(S) &= \int_S f \, d\mathbb{P} \\ &= \sum_{\alpha_j \in \mathrm{image}(f)} \alpha_j \mathbb{P}(f^{-1}(\alpha_j) \cap S)\end{aligned}$$

Construction of the Graph Integral

Consider a function $f : \mathrm{Vert}(G) \longrightarrow \mathbb{R}$ for G a finite directed graph.

Let $\mathbb{P} = \{p_v\}$ be the PageRank measure on $\mathrm{Vert}(G)$. We can then define a measure ν_f on $\mathrm{Vert}(G)$ by

$$
\begin{aligned}
\nu_f(S) &= \int_S f \, d\mathbb{P} \\
&= \sum_{\alpha_j \in \mathrm{image}(f)} \alpha_j \mathbb{P}(f^{-1}(\alpha_j) \cap S) \\
&= \sum_{v \in S} f(v) p_v
\end{aligned}
$$

Construction of the Graph Integral

Consider a function $f : \mathrm{Vert}(G) \longrightarrow \mathbb{R}$ for G a finite directed graph.

Let $\mathbb{P} = \{p_v\}$ be the PageRank measure on $\mathrm{Vert}(G)$. We can then define a measure ν_f on $\mathrm{Vert}(G)$ by

$$\begin{aligned}\nu_f(S) &= \int_S f\, d\mathbb{P} \\ &= \sum_{\alpha_j \in \mathrm{image}(f)} \alpha_j \mathbb{P}(f^{-1}(\alpha_j) \cap S) \\ &= \sum_{v \in S} f(v) p_v\end{aligned}$$

Let $\mathcal{P}$ be a partition of $[0,1]$ and let $G_q = \{v | p_v \leq q\}$.

Construction of the Graph Integral

Consider a function $f : \mathrm{Vert}(G) \longrightarrow \mathbb{R}$ for G a finite directed graph.

Let $\mathbb{P} = \{p_v\}$ be the PageRank measure on $\mathrm{Vert}(G)$. We can then define a measure ν_f on $\mathrm{Vert}(G)$ by

$$\begin{aligned} \nu_f(S) &= \int_S f \, d\mathbb{P} \\ &= \sum_{\alpha_j \in \mathrm{image}(f)} \alpha_j \mathbb{P}(f^{-1}(\alpha_j) \cap S) \\ &= \sum_{v \in S} f(v) p_v \end{aligned}$$

Let $\mathcal{P}$ be a partition of $[0, 1]$ and let $G_q = \{v | p_v \leq q\}$.

The filtration

$$G_{q_1} \subseteq G_{q_2} \subseteq \cdots \subseteq G_{q_{|\mathcal{P}|}} = \mathrm{Vert}(G)$$

Construction of the Graph Integral

Consider a function $f : \mathrm{Vert}(G) \longrightarrow \mathbb{R}$ for G a finite directed graph.

Let $\mathbb{P} = \{p_v\}$ be the PageRank measure on $\mathrm{Vert}(G)$. We can then define a measure ν_f on $\mathrm{Vert}(G)$ by

$$\begin{aligned} \nu_f(S) &= \int_S f \, d\mathbb{P} \\ &= \sum_{\alpha_j \in \mathrm{image}(f)} \alpha_j \mathbb{P}(f^{-1}(\alpha_j) \cap S) \\ &= \sum_{v \in S} f(v) p_v \end{aligned}$$

Let $\mathcal{P}$ be a partition of $[0, 1]$ and let $G_q = \{v | p_v \leq q\}$.

The filtration

$$G_{q_1} \subseteq G_{q_2} \subseteq \cdots \subseteq G_{q_{|\mathcal{P}|}} = \mathrm{Vert}(G)$$

allows us to define our graph antiderivative $F_{f,G}$ of f by

Construction of the Graph Integral

Consider a function $f : \mathrm{Vert}(G) \longrightarrow \mathbb{R}$ for G a finite directed graph.

Let $\mathbb{P} = \{p_v\}$ be the PageRank measure on $\mathrm{Vert}(G)$. We can then define a measure ν_f on $\mathrm{Vert}(G)$ by

$$\begin{aligned}\nu_f(S) &= \int_S f \, d\mathbb{P} \\ &= \sum_{\alpha_j \in \mathrm{image}(f)} \alpha_j \mathbb{P}(f^{-1}(\alpha_j) \cap S) \\ &= \sum_{v \in S} f(v) p_v\end{aligned}$$

Let $\mathcal{P}$ be a partition of $[0, 1]$ and let $G_q = \{v | p_v \leq q\}$.

The filtration

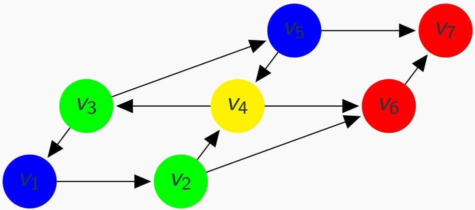
$$G_{q_1} \subseteq G_{q_2} \subseteq \cdots \subseteq G_{q_{|\mathcal{P}|}} = \mathrm{Vert}(G)$$

allows us to define our graph antiderivative $F_{f,G}$ of f by

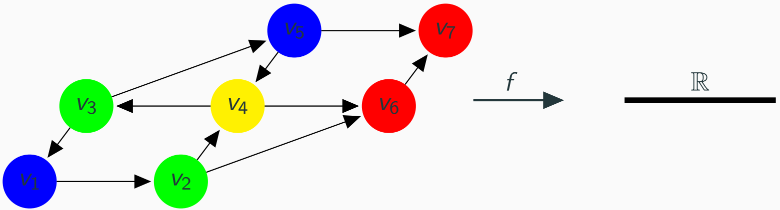
$$\begin{aligned}F_{f,G} &:= (\nu_f(G_{q_1}), \nu_f(G_{q_2}), \ldots, \nu_f(G_{q_{|\mathcal{P}|}})) \\ &= (\mathbb{E}[f|_{G_{q_1}}], \mathbb{E}[f|_{G_{q_2}}], \ldots, \mathbb{E}[f|_{G_{|\mathcal{P}|}}])\end{aligned}$$

The Graph Antiderivative Visualized

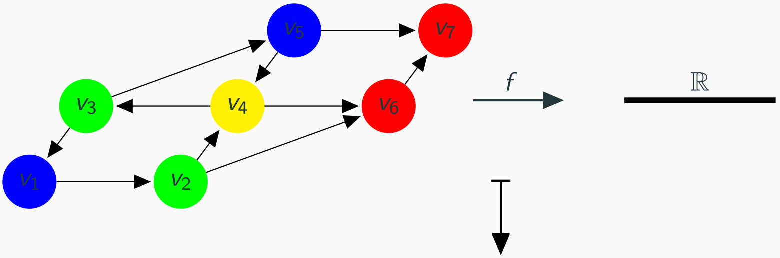
The Graph Antiderivative Visualized



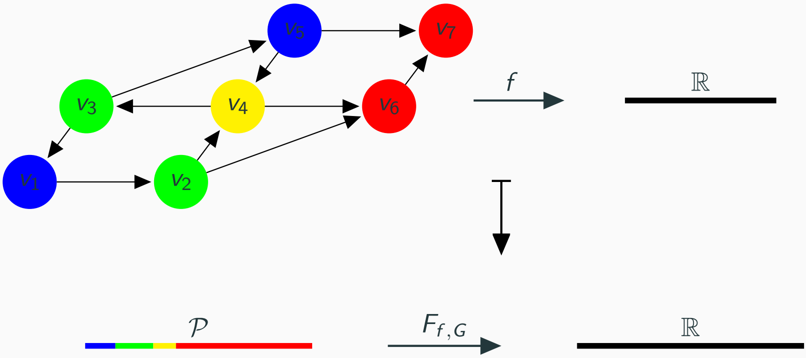
The Graph Antiderivative Visualized



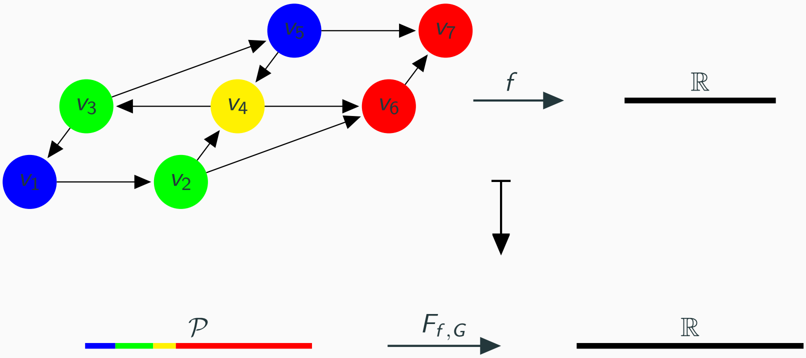
The Graph Antiderivative Visualized



The Graph Antiderivative Visualized



The Graph Antiderivative Visualized



Antiderivative

$$\Gamma \times \mathrm{Fun}(\bigsqcup_{\Gamma} \mathrm{Vert}(G), \mathbb{R}) \longrightarrow \mathrm{Fun}(\mathcal{P}, \mathbb{R})$$

$$(G, f) \mapsto (F_{f,G} : q \mapsto \mathbb{E}[f|_{G_q}]),$$

Graph Integration: Example

Graph Integration: Example

Consider a SDFG G given by:

$$\mathrm{Edge}(G) = \{(v_1, v_2), (v_1, v_3), (v_2, v_4), (v_3, v_4)\}$$

$$\text{PageRank}(G) = \langle p_{v_1} = 0.10, p_{v_2} = 0.15, p_{v_3} = 0.25, p_{v_4} = 0.50\rangle$$

Graph Integration: Example

Consider a SDFG G given by:

$$\mathrm{Edge}(G) = \{(v_1, v_2), (v_1, v_3), (v_2, v_4), (v_3, v_4)\}$$

$$\text{PageRank}(G) = \langle p_{v_1} = 0.10, p_{v_2} = 0.15, p_{v_3} = 0.25, p_{v_4} = 0.50\rangle$$

Assume the nodes $v_1, v_4 \in \mathrm{Vert}(G)$ both correspond to function calls $\phi_{v_i}(args_{v_i})$, where $args_{v_i}$ represent the set of arguments passed to ϕ_{v_i}.

Graph Integration: Example

Consider a SDFG G given by:

$$\mathrm{Edge}(G) = \{(v_1, v_2), (v_1, v_3), (v_2, v_4), (v_3, v_4)\}$$

$$\text{PageRank}(G) = \langle p_{v_1} = 0.10, p_{v_2} = 0.15, p_{v_3} = 0.25, p_{v_4} = 0.50\rangle$$

Assume the nodes $v_1, v_4 \in \mathrm{Vert}(G)$ both correspond to function calls $\phi_{v_i}(args_{v_i})$, where $args_{v_i}$ represent the set of arguments passed to ϕ_{v_i}. Define

$$\text{NumPass2Call} : \text{Vert}(G) \longrightarrow \mathbb{R}$$

Graph Integration: Example

Consider a SDFG G given by:

$$\mathrm{Edge}(G) = \{(v_1, v_2), (v_1, v_3), (v_2, v_4), (v_3, v_4)\}$$

$$\mathsf{PageRank}(G) = \langle p_{v_1} = 0.10, p_{v_2} = 0.15, p_{v_3} = 0.25, p_{v_4} = 0.50\rangle$$

Assume the nodes $v_1, v_4 \in \mathrm{Vert}(G)$ both correspond to function calls $\phi_{v_i}(args_{v_i})$, where $args_{v_i}$ represent the set of arguments passed to ϕ_{v_i}. Define

$$\mathsf{NumPass2Call} : \mathsf{Vert}(G) \longrightarrow \mathbb{R}$$

by

$$v_i \mapsto \begin{cases} \#\mathsf{args}_{v_i} & \text{if } i \in \{1, 4\} \\ 0 & \text{otherwise} \end{cases}$$

Graph Integration: Example

Consider a SDFG G given by:

$$\mathrm{Edge}(G) = \{(v_1, v_2), (v_1, v_3), (v_2, v_4), (v_3, v_4)\}$$

$$\mathsf{PageRank}(G) = \langle p_{v_1} = 0.10, p_{v_2} = 0.15, p_{v_3} = 0.25, p_{v_4} = 0.50\rangle$$

Assume the nodes $v_1, v_4 \in \mathrm{Vert}(G)$ both correspond to function calls $\phi_{v_i}(args_{v_i})$, where $args_{v_i}$ represent the set of arguments passed to ϕ_{v_i}. Define

$$\mathsf{NumPass2Call} : \mathsf{Vert}(G) \longrightarrow \mathbb{R}$$

by

$$v_i \mapsto \begin{cases} \#\mathsf{args}_{v_i} & \text{if } i \in \{1, 4\} \\ 0 & \text{otherwise} \end{cases}$$

Let $\mathcal{P} = (0.05, 0.12, 0.95)$.

Graph Integration: Example

Consider a SDFG G given by:

$$\mathrm{Edge}(G) = \{(v_1, v_2), (v_1, v_3), (v_2, v_4), (v_3, v_4)\}$$

$$\mathsf{PageRank}(G) = \langle p_{v_1} = 0.10, p_{v_2} = 0.15, p_{v_3} = 0.25, p_{v_4} = 0.50\rangle$$

Assume the nodes $v_1, v_4 \in \mathrm{Vert}(G)$ both correspond to function calls $\phi_{v_i}(args_{v_i})$, where $args_{v_i}$ represent the set of arguments passed to ϕ_{v_i}. Define

$$\mathsf{NumPass2Call} : \mathsf{Vert}(G) \longrightarrow \mathbb{R}$$

by

$$v_i \mapsto \begin{cases} \#\mathsf{args}_{v_i} & \text{if } i \in \{1, 4\} \\ 0 & \text{otherwise} \end{cases}$$

Let $\mathcal{P} = (0.05, 0.12, 0.95)$.

Then $F_{\mathrm{NumPass2Call},G} : \mathcal{P} \longrightarrow \mathbb{R}$ takes the form

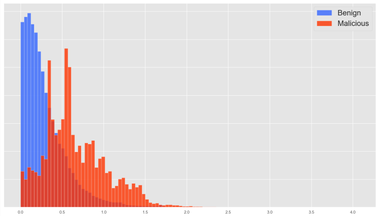
$$\begin{pmatrix} 0.05 \\ 0.12 \\ 0.95 \end{pmatrix} \mapsto \begin{pmatrix} 0 \\ 0.1 * \#\mathsf{args}_{v_1} \\ 0.1 * \#\mathsf{args}_{v_1} + 0.5 * \#\mathsf{args}_{v_4} \end{pmatrix}$$

Results

Vectorization Efficacy

Vectorization Efficacy

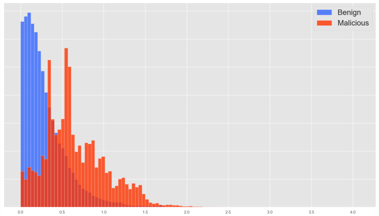
$\int_0^{0.6} \mathrm{ClassRefname} : v \mapsto \eta(\mathrm{name}(v))d\mathbb{P}$



Name of referenced class at v

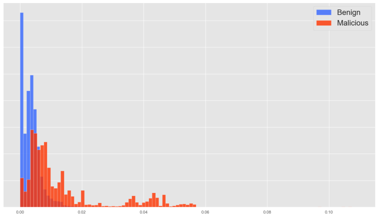
Vectorization Efficacy

$\int_0^{0.6} \text{ClassRefname} : v \mapsto \eta(\text{name}(v))d\mathbb{P}$



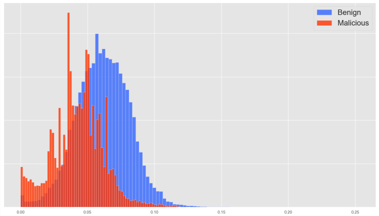
Name of referenced class at v

$\int_0^{0.4} \text{CLRLiteral} : v \mapsto \eta(\text{type}(v))d\mathbb{P}$



Type of literal occurring at v

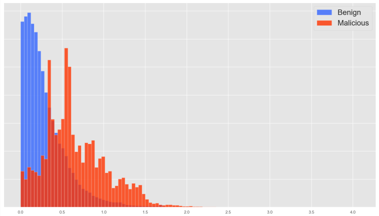
$\int_0^{0.95} \text{ArgRefType} : v \mapsto \eta(\text{type}(v))d\mathbb{P}$



Type of argument referenced at v

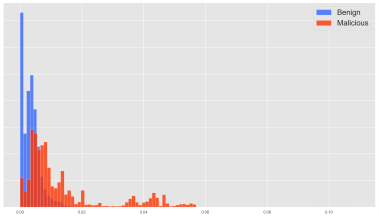
Vectorization Efficacy

$\int_0^{0.6} \text{ClassRefname} : v \mapsto \eta(\text{name}(v))d\mathbb{P}$



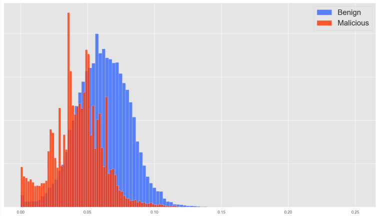
Name of referenced class at v

$\int_0^{0.4} \text{CLRLiteral} : v \mapsto \eta(\text{type}(v))d\mathbb{P}$



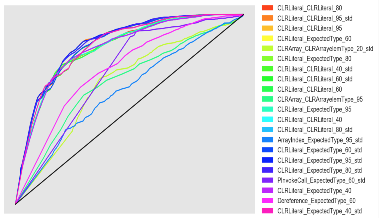
Type of literal occurring at v

$\int_0^{0.95} \text{ArgRefType} : v \mapsto \eta(\text{type}(v))d\mathbb{P}$



Type of argument referenced at v

Top Features by AUC



Value/type of literal expression at v

Model Results - Random Forest

Model Results - Random Forest

Table 1: Graph Antiderivative-based vectorization

Class	Precision	Recall	F1-score	Support
Benign	97.88%	99.37%	98.62%	696827
Malware	98.94%	96.47%	97.69%	424420
avg/total	98.28%	98.27%	98.27%	1121247
False Positive Rate	1.10%			
False Negative Rate	1.72%			

Model Results - Random Forest

Table 1: Graph Antiderivative-based vectorization

Class	Precision	Recall	F1-score	Support
Benign	97.88%	99.37%	98.62%	696827
Malware	98.94%	96.47%	97.69%	424420
avg/total	98.28%	98.27%	98.27%	1121247
False Positive Rate	1.10%			
False Negative Rate	1.72%			

Table 2: Text-only vectorization

Class	Precision	Recall	F1-score	Support
Benign	90.61%	87.04%	88.79%	696827
Malware	87.80%	91.18%	89.46%	424420
avg/total	89.19%	89.13%	89.13%	1121247
False Positive Rate	8.79%			
False Negative Rate	12.96%			

Questions?