This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI

10.1109/ACCESS.2020.3011909, IEEE Access

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2017.DOI

**Code Characterization with Graph Convolutions and Capsule Networks**

**POORNIMA HARIDAS1, GOPINATH CHENNUPATI2, NANDAKISHORE SANTHI2, PHILLIP ROMERO2 and STEPHAN EIDENBENZ2**

1 Courant Institute of Mathematical Sciences, New York University, New York City, NY, USA (e-mail: poornima.haridas@cs.nyu.edu)

* Los Alamos National Laboratory, Los Alamos, NM, USA (e-mail: gchennupati@lanl.gov, prr@lanl.gov, nsanthi@lanl.gov, eidenben@lanl.gov)

Corresponding author(s): Gopinath Chennupati (gchennupati@lanl.gov).

This work was supported by the U.S. Department of Energy National Nuclear Security Administration under Contract No. DE-AC52-06NA25396. This research used resources provided by the Los Alamos National Laboratory Institutional Computing Program, which is supported by the U.S. Department of Energy National Nuclear Security Administration under Contract No. 89233218CNA000001.

 **ABSTRACT** We propose SiCaGCN, a learning system to predict the similarity of a given software code to aset of codes that are permitted to run on a computational resource, such as a supercomputer or a cloud server. This code characterization allows us to detect abusive codes. Our system relies on a structural analysis of the control-flow graph of the software codes and two different graph similarity measures: Graph Edit Distance (GED) and a singular values based metric. SiCaGCN combines elements of Graph Convolutional Neural Networks (GCN), Capsule networks, attention mechanism, and neural tensor networks. Our experimental results include a study of the trade-offs between the two similarity metrics and two variations of our learning networks, with and without the use of capsules. Our main findings are that the use of capsules reduces mean square error significantly for both similarity metrics. Use of capsules reduces the runtime to calculate the GED while increases the runtime of singular values calculation.

 **INDEX TERMS** Capsule Networks, Control Flow Graph, GCN, Similarity, Eigen Values

**I. INTRODUCTION**

N the era of exascale computing code characterization Iis extremely important for super-computing centers and cloud vendors. Especially, for the institutions such as the US DOE national labs that run scientific codes (such as hydrody-namics simulations, molecular dynamics, etc). In some cases, these codes appear similar to the allowed scientific codes while abusive algorithms (for example, Bitcoin mining) run underneath, misusing the compute resources. In fact, one of the illicit crypto-currency mining is the use of computers connected to internet in order to run the mining codes. The first analysis appeared in 2014 [1], that analyzed thousands of public transactions in order to detect the bitcoin mining. Since 2014, these Recently, these threats have increased significantly, a detailed survey of crypto-mining is in [2]. To highlight the seriousness of the problem, recently in May 2020, the highly secure European supercomputers have been attacked by cryptocurrency miners 1.

In addressing the cryptocurrency mining problem, we pro-

* https://www.bbc.com/news/technology-52709660

pose a deep learning solution that can in the first place detect the occurrence of such a threat. Our approach, SiCaGCN, calculates the similarity of a pair of programs. SiCaGCN models a function to map a pair of programs, represented as control flow graph (CFG) of basic blocks. The model minimizes the difference between the predicted similarity scores and the ground truth, where each training data point is a pair of CFG of basic blocks and the true similarity score.

In code characterization, several approaches [3], [4], [5],

1. characterize the code search. These approaches work best when the code base in question is well-documented and comments are indicative of the actual intent of the code. At-tempts in literature employ control flow graphs for software defects [7] and malware detection [8]. Our, representation of code as a control flow graph of basic blocks is different. This represents the program at a logical level as basic blocks. Thereby, the system is safe from being tricked by obfuscating variables and comments.

Since, we represent the programs as graphs, we employ two commonly known graph similarity/distance metrics: Graph Edit Distance (GED) [9] and maximum common

|  |  |
| --- | --- |
| VOLUME 4, 2016 | 1 |

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3011909, IEEE Access



Poornima et al.: Preparation of Papers for IEEE Access

subgraph (MCS) [10]. Computing the GED/MCS is N P - complete [10], [11]. Moreover, the state-of-the-art algorithms fail to calculate GED within a reasonable amount of time [12] for graphs with more than 16 nodes. In our paper, we predict the following two similarity metrics.

1. GED based similarity metric [13], compute intensive.
2. Singular values similarity metric, easy to compute.

The SiCaGCN uses Graph Convolutional Neural Networks (GCN) [14] where convolutions are embedded in Graph Neural Networks (GNN) [15] from a spectral or spatial perspective as in [16]. In SiCaGCN, each node is learned to be an embedding vector. In literature, Zhang et al. [17] applied element-wise max-pooling to the node embeddings. Verma et al. [18] generated graph embeddings by computing the element-wise co-variance of all nodes. Bai et al. [13] gen-erated graph embeddings via an attention based mechanism from pooled GCN node embeddings. Li et al. [19] proposed the graph matching networks. These attempts capture node features in the form of a scalar, which is insufficient to ac-count for all the node/graph properties. To build high-quality graph embeddings, the properties of nodes with respect to the graph along with the structures around a node play an important role. We use these graph embeddings to calculate similarity metrics between graphs and characterize the codes. We adopt an approach inspired from CapsNet [20], [21] for better embeddings. We extend the scalar features into vectors to form graph capsules that preserve the information of graph properties more efficiently [22]. Finally, we use attention to weigh the node capsules with respect to their importance in the overall graph and the similarity metric.

We evaluate SiCaGCN on a custom dataset of C/C++ codes. The results indicate that our method is better in many aspects compared to the state-of-the-art technique. Moreover, we apply our approach to find the abusive use of compute resources, where we successfully detect the applications that run bitcoin mining algorithms on DOE resources.

Our contributions are as follows: we introduce a new GCN architecture that captures different latent properties of pro-grams using capsules; we prepare a new code dataset com-posed of various C/C++ programs; third, we apply SiCaGCN to the code dataset to produce a similarity metric between a pair of control-flow graphs of basic blocks that can be extended to other graphical datasets as well; finally, we compare the two similarity metrics to observe which helps to differentiate codes efficiently.

The rest of the paper is organized as follows: section II de-scribes the existing work; section III presents the SiCaGCN architecture; section IV explains the experimental results and section V concludes and recommends future directions.

**II. BACKGROUND**

We describe various concepts used in the paper.

1. Graph

A weighted directed graph can be defined as G = (V; X; A), where V = fv1; v2; : : : ; vN g is the set of

nodes, N is the number of nodes and X 2 RN d represents the features of each node and d is the number of feature channels. A 2 f0; 1gN N is the adjacency matrix where Aij = 1 if there is an edge from vi to vj and, Aij = 0 otherwise.

1. Graph Edit Distance

Our first similarity metric to test SiCaGCN and distin-guish abusive codes is, Graph Edit Distance(GED) [9]. GED between two graphs Gi and Gj is defined as the number of edit operations in the optimal alignments that transform Gi into Gj. The edit operations on the graph are: insertion or deletion of a vertex/edge, relabel a vertex, where the edit cost is 1 per operation.

1. Singular Values

The singular values of a matrix A of dimension M N can be defined as the square roots of the eigenvalues of the N×N matrix XT X. If X is a real symmetric matrix with non-negative eigenvalues, then the singular values will be the same as the eigenvalues of X.

1. Graph Convolution Network

We make use of GCN [14], a popular GNN architecture in our network. The convolution operation is applied to each node and the neighbors at every layer of GCN. The embedding for each node is then computed through an activation function. This can be summarized as follows:

|  |  |
| --- | --- |
| Zl+1 = f(T; Zl; W l) | (1) |

where Z 2 RN d represents the node features at layer l, d is the number of feature channels, Z0 = X; W l 2 Rd d0 is a channel filter that is a trainable weight matrix.

f is a nonlinear activation function, T 2 RN N is the information transform matrix which is calculated from A to guide the information flowing between nodes. To generate graph capsules, we take node features extracted from different layers of the GCN. We do this because at each layer l, a GCN extracts features of a node taking into consideration neighbors within l steps of the node as demonstrated in [14], which gives more detailed features for each node.

1. Attention Mechanism

Attention mechanism is popular in NLP [23], [24] and computer vision [25], [26] domains to capture the rele-vant parts of the input with respect to the task at hand. This mechanism defines and normalizes an attention measure to identify the relevance of each part of an input and applies the normalized attention to the input and weighs the features. In SiCaGCN, we use two types of attentions: make model size invariant by scaling the node capsule; and weight the graph capsules based on the specific similarity metric.

1. Neural Tensor Network (NTN)

The concept of a Neural Tensor Network to explore the relationship between two entities is first explored in [27] for NLP tasks. They show that the reasoning can be improved when entities are represented as an average of their constituting word vectors instead of individual

2 VOLUME 4, 2016

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3011909, IEEE Access



Poornima et al.: Preparation of Papers for IEEE Access

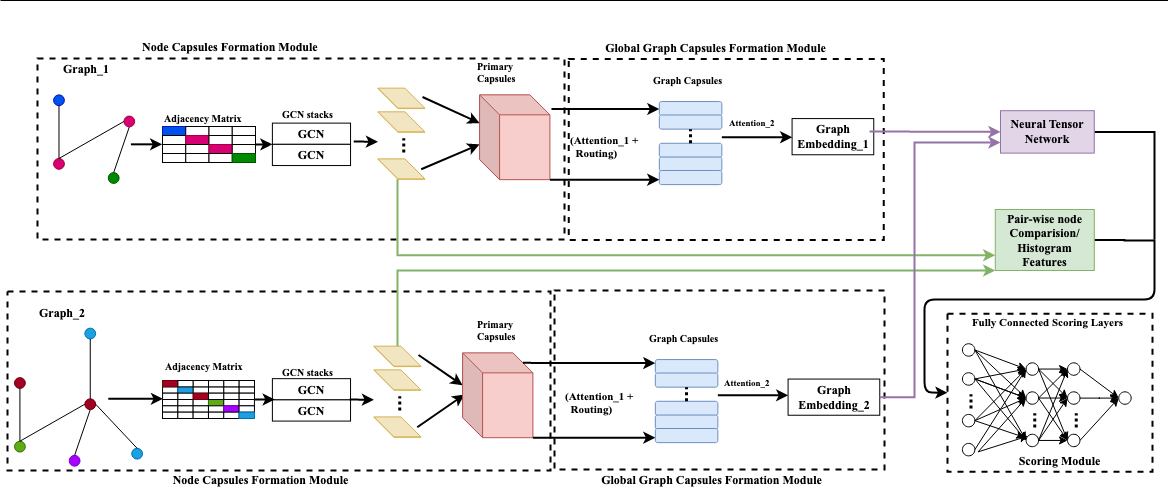


FIGURE 1: SiCaGCN Architecture contains four modules – 1) Node Capsule Formation accepts the graph as input and outputs node embeddings, 2) Global Graph Capsules Formation generates graph capsules with final graph embeddings, 3) Neural Tensor Network models the relation between two graphs and 4) Scoring module outputs the similarity metric. The input to SiCaGCN is a pair of graphs for two different programs and the output is a predicted similarity metric.

atomic units or with a single entity vector. We use the NTN concept and extend it to model a relation between the graph capsules of a pair of graphs.

* 1. Capsule Network

The concept of capsules was introduced in [20] to detect the presence of features, the orientation, position of a feature. The concept was developed further in [21] to extract features of an image based on the CNN model. In [21] the length of capsules reflects the probability of the presence of different features and the direction of capsules reflects the detailed properties of the fea-tures. Information between the layers is transmitted via a dynamic routing mechanism. Capsules are the main motivation behind our approach. We combine the con-cepts of a capsule with that of GNN to get better graph embeddings, in turn, get a similarity metric.

* 1. CFG of Basic Blocks

The control flow graph consists of vertices representing the basic blocks (a sequence of instructions without branching) of the code. In fact, the control flow graph of basic blocks are obtained from the compiled binary code without actually using the source code.

1. **THE SICAGCN ARCHITECTURE**

We describe SiCaGCN, an end-to-end neural network that learns a function to map a pair of graphs to the predefined similarity score. FIGURE 1 is the overview of the proposed architecture. SiCaGCN consists of four main modules:

1. Node Capsules Formation Module: In this module we apply, GCN to extract local node features as embed-

dings. These embeddings are used to build primary node capsules and obtain fine-grained node-level features.

* 1. Global Graph Capsules Formation Module: The node capsules undergo through a fused mechanism of a cou-ple of attentions and dynamic routing to generate multi-ple capsules for graphs and a final graph embedding.
  2. Neural Tensor Network (NTN) Module: Given the weighted graph-level capsules of two graphs, NTN is used to model the relation between the two graphs.
  3. Scoring Module: Appends the fine-grained node-level features with the output of the NTN and regresses to obtain the similarity metric for the given pair of graphs.

1. **NODE CAPSULES FORMATION MODULE**

The basic node features are extracted with GCNs, a GNN architecture that was developed by [14]. To obtain multi-scale node features, we extract features from different layers which are represented as capsules. This can be expressed as:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Zjl+1 = f(Xi | 1 | | 1 | | ZilWijl)] | (2) |  |
| D |  | AD |  |  |
| 2 | 2 |  |
|  | e | | ee | |  |  |  |

where Wijl 2 Rd d0 is the trainable weights matrix. It serves as the channel filters from the i-th channel at the l-th layer to

the j-th channel at the (l+1)-th layer. The activation function we consider is f ( ) = tanh( ) and Zl+1 2 RN d0, Z0 = X, Ae = A + I1 and De = PAeij. We consider features of sub-components with different sizes, node features of all GNN layers are used to generate the high-level capsules.

We model a pair-wise node comparison method to supple-ment the graph-level capsules with fine-grained node-level

|  |  |
| --- | --- |
| VOLUME 4, 2016 | 3 |

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3011909, IEEE Access



Poornima et al.: Preparation of Papers for IEEE Access

information. This is to ensure that we capture all node-level information and feature distribution, which will be lost when a graph-capsule is formed. To capture the information, we do the following: Consider graphs Gi and Gj with nodes Ni and Nj. If the graphs are of different sizes, we pad the smaller graph with fake nodes. NiNj number of pairwise embeddings are obtained by S = (UiUjT ), where Zi 2 RUi d and Uj 2 RNj d are the node capsules of Gi and Gj, respectively. The sigmoid function is applied to ensure the similarities scores are in the range of (0; 1). Histogram features of S are extracted as hist(S) 2 Rb, where b is the number of bins in the histogram, to ensure that our model is invariant to graph representation and node ordering.

1. **GLOBAL GRAPH CAPSULES FORMATION MODULE**

This module formulates the graph capsules as following:

* 1. Attention 1 – Primary node capsules are scaled using the first attention mechanism, Fattn1( ) that ensures the graph-capsule formed is graph size (number of primary capsules) invariant and focuses on important parts of the graph. Fattn1( ) is a two-layer fully connected neural

network . The number of input units of Fattn1( ) is

P

d Call where Call = l Cl and the number of output units equals Call. Node-based normalization is applied to generate attention value in each channel and then the original node capsules are scaled, which is written as:

Fatt1(sn)i

scaled(s(n; i)) = P e s(n;i) (3)

n(Fatt1(sen)i)

where sn 2 R1 Calld is obtained by concatenating all capsules of the node n. s(n;i) 2 R1 d represents the i-th capsule of the node n and Fatt1(sen) 2 R1 Call is the generated attention value. The result of this module is of the form S 2 RN Call d.

1. Calculate Votes – This sub-module inspired by [28] out-puts a tensor V 2 RN Call P d where Call denotes the number of channels and P denotes the defined number of graph capsules.
2. Dynamic Routing Mechanism – High-level graph cap-sules are computed with [28], which is based on votes produced in the previous step.
3. Attention 2 – To generate one meaningful embedding per graph from all the capsules obtained, we intro-duce a mechanism to weigh the capsules relevant to the global context of the graph. We use another at-tention mechanism for the same. Consider the input node capsules to be U 2 RN d, where the n-th row, un 2 Rd is the capsule n. First, a global graph con-text c 2 Rd is computed, which is the mean of all node capsules followed by a nonlinear transformation:

c = tanh(( 1 PN un)W2), where W2 2 Rd d is

N n=1

a learnable weight matrix. The context c provides the global structural information of the graph that is adapted to the given similarity metric, via learning the weight matrix. Based on c, we compute one attention weight for each capsule. To make capsule n, attention aware of

the global context, the inner product between c and its capsule embedding is taken. This gives higher weights to capsules similar to the global context. A sigmoid

|  |  |  |  |
| --- | --- | --- | --- |
| function (x) = | 1 | is applied to the result to |  |
| 1+exp(x) |  |

ensure that the attention weights are in the range (0; 1).

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| The final graph | embedding, h | | | | 2 R | d is the weighted sum | | |  |
|  |  |  | N |  |  |  |  |
| of node capsules, h = | | |  | n=1 anun. Therefore, the final | | | | |  |
| attention module can | | be summarized as follows: | | | | | |  |  |
|  | P |  |  |  |  |  |  |
|  |  |  | N | |  |  |  |  |  |
|  |  |  | X | |  |  |  | (4) |  |
|  | h = | |  | (uT c)u | | | n |  |
|  |  |  |  |  | n |  |  |  |

n=1

**C. NEURAL TENSOR NETWORK (NTN) MODULE**

This module shows the relation between the two attention weighed graph capsules. Taking the inner product of the capsules results in weak interaction between the two capsules as shown in [27]. Therefore, we use NTN, as demonstrated in [27] to model the relation between two graph-level capsules,

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| g(hi; hj) = f(hi | | W3 | hj + V | hj | + b3) (5) | | |  |
|  | T | [1:K] |  | hi |  |  |  |  |
| [1:K] | 2 Rd d K is a weight tensor, | | | | | hi | denotes |  |
| where W3 | hj |  |

concatenation, V 2 RK 2d is a weight vector, (b3 2 RK ) is a bias vector, f( ) is an activation function and K is a hyper-parameter to control the number of similarity scores that can be produced by the model for each graph-capsule pair.

**D. SCORING MODULE**

The histogram feature vector from node capsules formation module is normalized and concatenated with the graph-level interaction scores from the NTN module and fed to the fully connected layers to reduce to a final similarity score, s0ij 2 R for the graph pair. This is compared against the ground-truth similarity score using the mean squared error loss function:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 |  | X | |  |  |  |
| M SEloss = |  |  | (s0 | s(G ; G ))2 | (6) |  |
|  |  |  |
|  | jDj | | 2 | ij | i j |  |  |
|  | (i;j) | D |  |  |  |

* is the set of training graph pairs, and s(Gi; Gj) is the ground-truth similarity metric between graphs Gi and Gj.

**IV. EXPERIMENTS**

While SiCaGCN can in principle be used for a large set of graph similarity learning problems, we conducted exper-iments on using the SiCaGCN approach for code charac-terization, which we describe in more detail. We compare our approach against SimGNN, trained for 5000 epochs for GED similarity metric and for around 1000 epochs for SV similarity metric. The learning rate for our network is 1e-4 with an Adam optimizer (weight decay of 5 104 ) and 16 histogram bins. SiCaGCN runsfor 500 epochs for GED similarity metric and for 1000 epochs for SV metric.

**A. SIMGNN**

Similarity computation via graph neural networks (SimGNN) [13] computes the similarity between a pair of graphs using

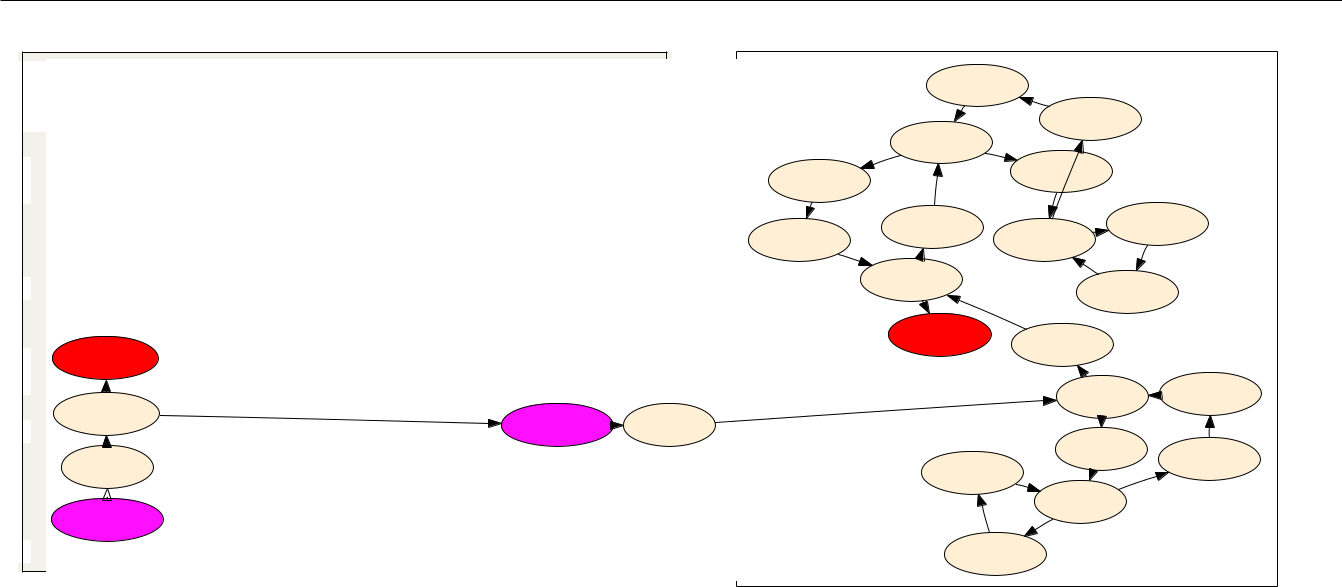
4 VOLUME 4, 2016

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3011909, IEEE Access



Poornima et al.: Preparation of Papers for IEEE Access



|  |  |
| --- | --- |
| 1 | **#define N 256;** |
| 2 | **//Matrix multiplication kernel** |
| 3 | **float\*\* r8\_ijk(float\*\* a, float\*\* b, float\*\* c) {** |

* **int i, j, k;**

5 **//Initialization**

6 **for (i = 0; i < N; i++) //i loop**

* **for (j = 0; j < N; j++) { //j loop**

8 **a[i][j] = 0.0;**

* **b[i][j] = c[i][j] = 1.0;**

10 **}**

11 **for (i=0; i<N; i++) //i loop**

1. **for (j=0; j<N; j++) //j loop**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 13 | [BB %0.2] |  |  |  |  |  |  |
| **for (k=0; k<N; k++) //k loop** | |  |  |  |  |  |
| 14 | 1 **a[i][k] = a[i][k] + b[i][j]** **\*** **c[j][k];** | | |  |  |  |  |
| 15 | **return a;** | 1 |  | 1 |  |  |  |
| 16 | [BB %0.1] | [/ENTRY/] | [BB | %0] |  |
| **} //end of r8\_ijk()** |  |  |  |
|  | 1 |  |  |  |  |  |  |

17 **int** **main() {**

[BB %0]

18 **float A[N][N], B[N][N], C[N][N];**

1

19 **A = r8\_ijk(A,B,C);**

20 [/ENTRY/]

**return 0;**

21**} //end of main()**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | [BB %90] | | 1 |  |  |  |
|  |  |  |  |  |  |  |
|  |  | 1 |  | [BB %89] |  |  |  |
|  | 0.04 | [BB %45] | 0.95 |  |  |  |  |
|  |  |  |  |  |  |
| [BB %93] |  | 1 |  | [BB %48] |  |  |  |
|  |  | 0.04 |  |  |  |
|  |  |  |  |  |  |  |
| 1 |  |  |  | 1 |  |  |  |
|  | [BB %44] |  | [BB %52] | | |  |
| [BB %94] |  |  |  |
|  | 0.95 | [BB %49] 0.95 | |  |  |  |
| 1 |  |  | 1 | 1 |  |  |
| [BB %41] | |  |  |  |
|  |  | [BB %86] | |  |  |
|  |  | 0.04 | 1 |  |  |
|  |  |  |  |  |  |
|  |  | [BB %97] |  | [BB %40] |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  | 0.04 |  |  |  |
|  | 1 |  |  | [BB %4] | 1 | [BB %37] |  |
|  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  | 0.95 |  | 1 |  |
|  |  |  |  | [BB %7] |  | [BB %36] |  |
|  |  | [BB %33] | |  |  |  |
|  |  | 1 | 0.04 |  |  |
|  |  |  | 1 |  |  |
|  |  |  | [BB %8] |  |  |
|  |  |  | 1 |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  | 0.95 |  |  |  |
|  |  | [BB %11] | |  |  |  |  |

FIGURE 2: Matrix multiplication as an input code C example, only relevant function is shown.

1 **; <label>:44** **;preds = %3**

* **%7 = load i32, i32\* %i, align 4**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 3 | **%8** | **=** | **mul nsw i32 2,** | **%7** |
| 4 | **%9** | **=** | **load i32, i32\*** | **%i, align 4** |

* **%10 = sext i32 %9 to i64**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 6 |  | **%11** | **=** | **load i32\*, i32\*\* %1, align 8** |  |  |
| 7 |  | **%12** | **= getelementptr inbounds i32, i32\* %11, i64** | |  |  |
| 8 |  |  | **%10** | |  |  |
|  | **store** | | **i32 %8, i32\* %12, align 4** |  |  |

* **br label %13**

FIGURE 4: Annotated BasicBlock from LLVM IR

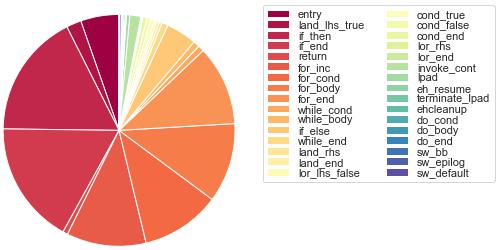


FIGURE 5: Distribution of different basic block (node) labels

GCN. SimGNN encodes each node of a graph into a vector representing the surrounding structural properties. SimGNN consists of two stages, in the first stage, the node level embeddings are passed through attention mechanism then the entire graph level embeddings are passed through context-aware attentions, these embeddings are passed through neural tensor network. In the second stage, the node-level embed-dings are used to calculate the pair-wise node comparison. The output of the two stages are combined before making the final prediction of the similarity score. We use the same hyper-parameters that are used in the original SimGNN [13].

FIGURE 3: CFG of the basic blocks in r8\_ijk() with branching probabilities and basic block labels.

That are, 3 GCN layers, ReLU activation function, Adam optimizer and a learning rate of 1e-4.

**B. DATASET AND PRE-PROCESSING**

We developed a custom dataset with 77 standard C/C++ codes, which comprise of standard benchmark codes from [29], mini-apps (jacobi, conjugate gradient, etc) and linear algebraic kernels (matrix multiplication, etc). For ex-ample, FIGURE 2 shows the kernels of matrix multiplication code in C, while FIGURE 3 shows the corresponding control-flow graph of basic blocks with the branch probabilities across multiple basic blocks. For a detailed explanation of calculating these branch probabilities, please refer to [30]. The graph of basic block is prepared from the intermediate representation (IR) of the original code. FIGURE 4 shows an example IR. Similarly, this analysis can be extended to large scale distributed applications [31], [32] that run on GPUs and compute clusters.

Each of these source codes (Si) in our dataset are compiled to produce a control flow graph, which are transformed into adjacency matrices of the control-flow graph of the basic blocks of the code in the following manner:

1. Si is compiled using the C/C++/Objective-C compiler clang (and flang for Fortran), to get an architecture independent LLVM IR (see FIGURE 4).
2. We parse the LLVM IR to produce a logical control flow graph of the basic blocks of the code. A basic-block is a straight-line code with a single entry and a single exit; a standard concept in compiler design (see FIGURE 3).
3. The adjacency matrix of the control flow graph and labels of the nodes are then used as input to the network. Labels of the nodes are control-flow structures like the beginning, body and end of for loops, if-else statements, etc. FIGURE 5 shows different node labels and their

|  |  |
| --- | --- |
| VOLUME 4, 2016 | 5 |

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3011909, IEEE Access



Poornima et al.: Preparation of Papers for IEEE Access

distribution in our dataset.

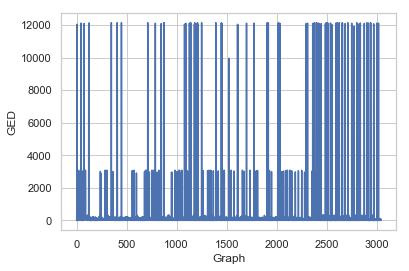


FIGURE 6: GED of all graph pairs

Node degrees are used as attributes as we have not in-cluded other node-wise data flow properties. Each graph is paired with every other graph. Here, we use both node labels and degree as features for SiCaGCN; we note that other feature sets are possible, in particular the basic block control flow structure can be easily learned from the actual binary of the codes as well, we just used the described LLVM-based approach for convenience from an existing tool chain (see [33] for more details). We then calculate two similarity metrics for each pair of graphs Gi and Gj.

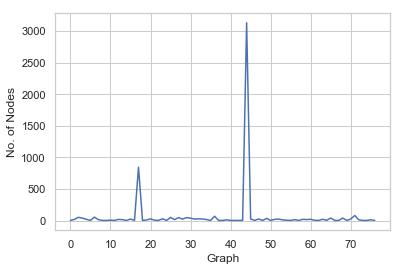
1. GED – For each graph pair, we calculate the GED based ground-truth similarity metric as follows:
   1. The graph edit distance is calculated as GED(Gi; Gj) based on the approximation algorithm in [34].
   2. GED is then normalized, normGED(Gi; Gj) =

GED(Gi;Gj )

(jGij+jGj j)=2

c) Finally an exponential function (x) = ex is used to transform the normGED into a similarity score in the range of (0,1].

FIGURE 7: Distribution of the number of nodes per graph



1. Time to generate the ground-truth to train the model for each similarity metric. For GED, we used the method in [34], available as a python library.
2. Wall time required to train the model and get the MSE below the baseline MSE. Baseline MSE is defined as:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| MSE = | 1 | N (yi |  | )2 | (7) |  |
| y |  |

X

N

i=1

1. Mean Squared Error (MSE) - average squared difference between the computed similarities (predicted) and the ground-truth similarities (target) for each model.
2. We report the Spearman’s Rank Order Correlation ( ) and the Pearson’s Correlation ( ) of the predicted values with respect to the ground truth.

TABLE 1: Time to generate ground-truth data and the base-line MSE for both the similarity metrics.

|  |  |  |
| --- | --- | --- |
| Similarity Metric | Time (minutes) | Baseline MSE |
|  |  |  |
| SV | 16 | 0.0497 |
| GED | 929 | 5.7881 |

Note that higher the similarity score, the more similar the given pair of graphs. With 77 codes we have 3003 pairs. A 80-20 train-test split was taken.

* 1. Singular values (SV ) – For each graph-pair, we calcu-late the singular values based ground-truth as follows:
     1. We take the adjacency matrix Ai and Aj of each graph and update the diagonal entries with the out degree of the node.
     2. The singular values of Ai and Aj are calculated as SVi and SVj. SVi and SVj are expanded to the size of the largest graph in the dataset by repeating the singular values in order.
     3. The Pearson Correlation between SVi and SVj is calculated and normalized to a range of [0, 1].

1. **EVALUATION METRICS**

The metrics used to evaluate the models and similarity are:

1) Time

FIGURE 7 shows the distribution of nodes in ground truth dataset. TABLE 1 shows the wall time to produce the ground-truth and the baseline MSE for both the similarity metrics. Clearly, SV is faster to produce the ground truth.

TABLE 2: MSE comparison

|  |  |  |
| --- | --- | --- |
| Model | MSE | Similarity metric |
|  |  |  |
| SimGNN | 11.7444 | GED |
| SiCaGCN | 4.743 | GED |
| SimGNN | 0.043 | SV |
| SiCaGCN | 0.032 | SV |

**D. RESULTS**

We analyze the predictive power of our model with that of the SimGNN. The results of the best method are shown in bold. TABLE 2 compares SimGNN and SiCaGCN in predicting

6 VOLUME 4, 2016

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3011909, IEEE Access



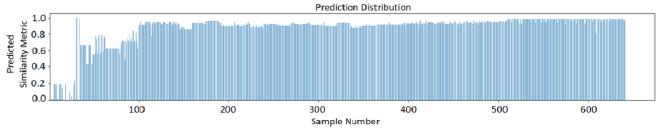
Poornima et al.: Preparation of Papers for IEEE Access

the MSE using both the similarity metrics. The results indi-cate that SiCaGCN is better compared to SimGNN for bot hthe similarity metrics.

TABLE 3: SimGNN vs SiCaGCN for correlation and runtime

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Metric | Model | MSE |  |  | Time(min) |
|  |  |  |  |  |  |
| GED | SimGNN | 11.744 | 0.939 | 0.955 | 1350 |
| GED | SiCaGCN | 4.743 | 0.906 | 0.804 | 350 |
| SV | SimGNN | 0.042 | 0.947 | 0.762 | 150 |
| SV | SiCaGCN | 0.032 | 0.958 | 0.95 | 715 |

Similarly, we analyze the Spearman and Pearson correla-tion coefficients for both the models of GED and SV metrics. TABLE 3 compares the correlations and the runtimes of both the models for both the metrics. The best results are in bold for the corresponding metric. For GED, SiCaGCN is best in terms of MSE, whereas SimGNN predictions have better correlations while the SicaGCN is better in terms of the runtime. For SV, SiCaGCN performs better in predicting the MSE with better correlations, while SimGNN is better in terms of runtime. Overall SiCaGCN achieves best MSE for both the metrics and comparable values for other metrics if not better. To trade-off between the two models in terms of runtime, although SimGNN is better for SV, we prefer to spend extra time to get better results with SiCaGCN.



* + 1. Predictions of SimGNN
  1. Predictions of SiCaGCN

1. Ground truth target similarity values

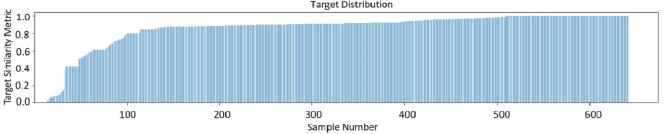
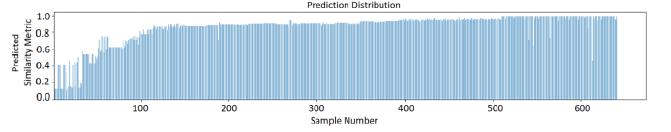
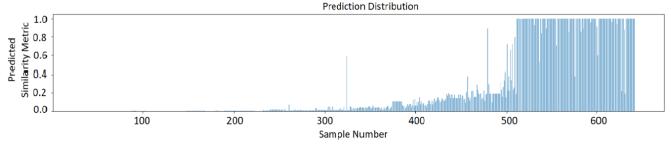
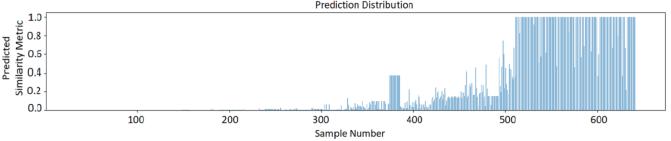


FIGURE 8: Target vs Predicted SV for both the models.

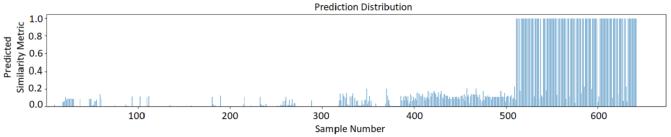
The evaluation metrics in TABLE 3 for the models for SV clearly show that our model out performs the SimGNN model, however SiCaGCN takes more time to get to a lower MSE value. On the other hand, FIGURE 8 compares the predicted SV (for both the models, see FIGURE 8a–8b) with that of the ground truth (FIGURE 8c). The results indicate that SimGNN fails to reach the baseline MSE, therefore, SiCaGCN is the best model.

FIGURE 9 shows the predicted GED values for both the models with different epochs. SimGNN, despite a significant increase in the number of epochs, fails to converge to a

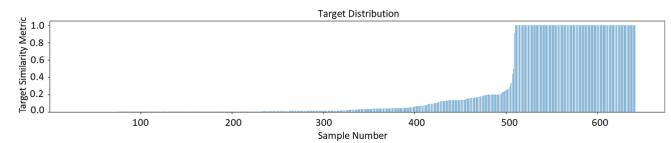
(a) Predictions of SimGNN(at 500 epochs)



(b) Predictions of SimGNN(at 5000 epochs)



(c) Predictions of SiCaGCN(at 500 epochs)



(d) Ground-Truth Target similarity values

FIGURE 9: Target vs Predicted GED for both the models

better MSE compared to the baseline. Moreover, the 500 epochs of SiCaGCN takes 350 minutes to reach convergence while that of SimGNN with 5000 epochs takes 1350 minutes without actually converging. Therefore, SiCaGCN is a better approach both in terms of time and producing reasonably good solutions.

TABLE 4: Results for Code Pairing with B for SiCaGCN

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Model | Code Pairs | Metric | Ground-Truth | | | Predicted | |  |
|  |  |  |  |  |  |  |  |  |
| SiCaGCN | C1, B | GED | 0:451 | 10 | 3 | 0:180 10 | 3 |  |
| 3 |  |  |
| SimGNN | C1, B | GED | 0:451 | 10 |  | 0.0 |  |  |
| SiCaGCN | C1, B | SV | 0.493 | |  | 0.426 |  |  |
| SimGNN | C1, B | SV | 0.493 | |  | 0.324 |  |  |

**E. CASE STUDY : BITCOIN MINER**

To test the practical use of our approach in identifying the abusive codes from non-abusive codes. We paired a bitcoin mining code with all the codes in our dataset. We pre-dicted both the similarity metrics using both SimGNN and SiCaGCN. We calculate the ground-truth for all the pairs in order to ensure that a bitcoin mining code is significantly different from scientific codes.

TABLE 4 summarizes the findings for the code pair of Floyd-Warshall (C1) and bitcoin mining (B). SiCaGCN performs better and has more robust predictions compared to SimGNN. SiCaGCN produces accurate similarity values while SimGNN produces a similarity value of 0:0 for all the pairings with B for GED metric and that of SV are close to 0 irrespective of the ground truth. Therefore, SiCaGCN is capable of identifying the abusive codes that exploit the

|  |  |
| --- | --- |
| VOLUME 4, 2016 | 7 |

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3011909, IEEE Access



Poornima et al.: Preparation of Papers for IEEE Access

compute resources from various institutions, which promises greater practical utility in real world.

**V. CONCLUSION AND FUTURE WORK**

We proposed the SiCaGCN system that predicts graph similarity metrics and presented a study of how to apply SiCaGCN to the problem of code characterization. We find that SiCaGCN with the advantage of capsules, produces better graph embeddings and therefore optimizes faster. For future work, we plan to study SiCaGCN in more detail for code characterization with the aim of understanding the sensitivity of SiCaGCN on the chosen feature sets; more-over, we plan to apply the SiCaGCN approach to other graph similarity domains, in particular for the social network graphs. Moreover, we further want to employ unsupervised methods such as the community detection strategies [35] in order to find the clusters of programs and separate the programs that are abusive. Another interesting unsupervised direction will be to incorporate large scale nonnegative low rank representations [36] of programs graphs in order to detect the outliers, application of heuristics [37] in order to automatically generate the deep network architectures for abusive program detection.

**ACKNOWLEDGEMENTS**

This work was supported by the U.S. Department of Energy National Nuclear Security Administration under Contract No. DE-AC52-06NA25396. This research used resources provided by the Los Alamos National Laboratory Institu-tional Computing Program, which is supported by the U.S. Department of Energy National Nuclear Security Adminis-tration under Contract No. 89233218CNA000001.

**REFERENCES**

1. D. Y. Huang, H. Dharmdasani, S. Meiklejohn, V. Dave, C. Grier, D. Mc-Coy, S. Savage, N. Weaver, A. C. Snoeren, and K. Levchenko, “Botcoin: Monetizing stolen cycles.” in Proceedings ofthe Network and Distributed System Security Symposium (NDSS), 2014, pp. 1–16.
2. S. Pastrana and G. Suarez-Tangil, “A first look at the crypto-mining malware ecosystem: A decade of unrestricted wealth,” in Proceedings of the Internet Measurement Conference, 2019, pp. 73–86.
3. M. Kimmig, M. Monperrus, and M. Mezini, “Querying source code with natural language,” in 2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011). IEEE, 2011, pp. 376–379.
4. M. Allamanis and C. Sutton, “Mining source code repositories at massive scale using language modeling,” in Proceedings of the 10th Working Conference on Mining Software Repositories. IEEE Press, 2013, pp. 207–216.
5. V. Saini, F. Farmahinifarahani, Y. Lu, P. Baldi, and C. V. Lopes, “Oreo: Detection of clones in the twilight zone,” in Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, 2018, pp. 354– 365.
6. S. Sachdev, H. Li, S. Luan, S. Kim, K. Sen, and S. Chandra, “Retrieval on source code: a neural code search,” in Proceedings of the 2nd ACM SIG-PLAN International Workshop on Machine Learning and Programming Languages. ACM, 2018, pp. 31–41.
7. A. V. Phan, M. Le Nguyen, and L. T. Bui, “Convolutional neural networks over control flow graphs for software defect prediction,” in 2017 IEEE 29th International Conference on Tools with Artificial Intelligence (ICTAI). IEEE, 2017, pp. 45–52.
8. J. Yan, G. Yan, and D. Jin, “Classifying malware represented as control flow graphs using deep graph convolutional neural network,” in 2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN). IEEE, 2019, pp. 52–63.
9. H. Bunke, “What is the distance between graphs,” Bulletin of the EATCS, vol. 20, pp. 35–39, 1983.
10. H. Bunke and K. Shearer, “A graph distance metric based on the maximal common subgraph,” Pattern recognition letters, vol. 19, no. 3-4, pp. 255– 259, 1998.
11. Z. Zeng, A. K. Tung, J. Wang, J. Feng, and L. Zhou, “Comparing stars: On approximating graph edit distance,” Proceedings of the VLDB Endowment, vol. 2, no. 1, pp. 25–36, 2009.
12. D. B. Blumenthal and J. Gamper, “On the exact computation of the graph edit distance,” Pattern Recognition Letters, 2018.
13. Y. Bai, H. Ding, S. Bian, T. Chen, Y. Sun, and W. Wang, “Simgnn: A neural network approach to fast graph similarity computation,” in Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining. ACM, 2019, pp. 384–392.
14. T. N. Kipf and M. Welling, “Semi-supervised classification with graph convolutional networks,” arXiv preprint arXiv:1609.02907, 2016.
15. F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, “The graph neural network model,” IEEE Transactions on Neural Net-works, vol. 20, no. 1, pp. 61–80, 2008.
16. J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, “Spectral networks and locally connected networks on graphs,” arXiv preprint arXiv:1312.6203, 2013.
17. M. Zhang, Z. Cui, M. Neumann, and Y. Chen, “An end-to-end deep learning architecture for graph classification,” in Thirty-Second AAAI Conference on Artificial Intelligence, 2018.
18. S. Verma and Z.-L. Zhang, “Graph capsule convolutional neural net-works,” arXiv preprint arXiv:1805.08090, 2018.
19. Y. Li, C. Gu, T. Dullien, O. Vinyals, and P. Kohli, “Graph matching networks for learning the similarity of graph structured objects,” arXiv preprint arXiv:1904.12787, 2019.
20. G. E. Hinton, A. Krizhevsky, and S. D. Wang, “Transforming auto-encoders,” in International Conference on Artificial Neural Networks. Springer, 2011, pp. 44–51.
21. G. E. Hinton, S. Sabour, and N. Frosst, “Matrix capsules with em routing,” 2018.
22. Z. Xinyi and L. Chen, “Capsule graph neural network,” 2018.
23. Y. Cui, Z. Chen, S. Wei, S. Wang, T. Liu, and G. Hu, “Attention-over-attention neural networks for reading comprehension,” arXiv preprint arXiv:1607.04423, 2016.
24. J. Lu, J. Yang, D. Batra, and D. Parikh, “Hierarchical question-image co-attention for visual question answering,” in Advances In Neural Informa-tion Processing Systems, 2016, pp. 289–297.
25. P. Anderson, X. He, C. Buehler, D. Teney, M. Johnson, S. Gould, and L. Zhang, “Bottom-up and top-down attention for image captioning and visual question answering,” in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 6077–6086.
26. Z. Yang, X. He, J. Gao, L. Deng, and A. Smola, “Stacked attention networks for image question answering,” in Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 21–29.
27. R. Socher, D. Chen, C. D. Manning, and A. Ng, “Reasoning with neural tensor networks for knowledge base completion,” in Advances in neural information processing systems, 2013, pp. 926–934.
28. S. Sabour, N. Frosst, and G. E. Hinton, “Dynamic routing between capsules,” in Advances in neural information processing systems, 2017, pp. 3856–3866.
29. T. Helmuth and L. Spector, “General program synthesis benchmark suite,” in Proceedings of the 2015 Annual Conference on Genetic and Evolution-ary Computation. ACM, 2015, pp. 1039–1046.
30. G. Chennupati, N. Santhi, S. Eidenbenz, and S. Thulasidasan, “An analyt-ical memory hierarchy model for performance prediction,” in 2017 Winter Simulation Conference (WSC). IEEE, 2017, pp. 908–919.
31. M. A. Obaida, J. Liu, G. Chennupati, N. Santhi, and S. Eidenbenz, “Par-allel application performance prediction using analysis based models and hpc simulations,” in Proceedings of the 2018 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation, 2018, pp. 49–59.
32. Y. Arafa, A.-H. A. Badawy, G. Chennupati, N. Santhi, and S. Eidenbenz, “Ppt-gpu: Scalable gpu performance modeling,” IEEE Computer Archi-tecture Letters, vol. 18, no. 1, pp. 55–58, 2019.
33. G. Chennupati, N. Santhi, and S. Eidenbenz, “Scalable performance pre-diction of codes with memory hierarchy and pipelines,” in Proceedings of

8 VOLUME 4, 2016

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3011909, IEEE Access



Poornima et al.: Preparation of Papers for IEEE Access

the 2019 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation. ACM, 2019, pp. 13–24.

1. A. Fischer, K. Riesen, and H. Bunke, “Improved quadratic time approxi-mation of graph edit distance by combining hausdorff matching and greedy assignment,” Pattern Recognition Letters, vol. 87, pp. 55–62, 2017.
2. H.-J. Li, Z. Bu, Z. Wang, J. Cao, and Y. Shi, “Enhance the performance of network computation by a tunable weighting strategy,” IEEE Transactions on Emerging Topics in Computational Intelligence, vol. 2, no. 3, pp. 214– 223, 2018.
3. G. Chennupati, R. Vangara, E. Skau, H. Djidjev, and B. Alexandrov, “Distributed non-negative matrix factorization with determination of the number of latent features,” The Journal of Supercomputing, vol. 76, no. 9, pp. 1–31, 2020. [Online]. Available: https://doi.org/10.1007/s11227-020-03181-6
4. G. Chennupati, R. M. A. Azad, and C. Ryan, “Performance optimization of multi-core grammatical evolution generated parallel recursive programs,” in Proceedings of the 2015 Annual Conference on Genetic and Evolution-ary Computation, 2015, pp. 1007–1014.

POORNIMA HARIDAS is a Computer Science graduate student at Courant Institute of Technol-ogy, New York University. Her areas of research interests include Deep Learning applications in Computer Vision, NLP and Reinforcement Learn-ing.



GOPINATH CHENNUPATI is a Computer Sci-entist in the Information Sciences (CCS-3) group at Los Alamos National Laboratory (LANL). Gopinath has obtained his PhD from University of Limerick, Ireland. Gopinath works on high per-formance computing (HPC), performance model-ing, natural language processing (NLP), deep/ma-chine learning, high performance linear algebra, matrix/tensor factorization and parallel program-ming, etc.



NANDAKISHORE SANTHI is a Computer Sci-entist with the Information Sciences Group (CCS-



1. at Los Alamos National Laboratory. He holds a PhD in Electrical and Computer Engineering from the University of California San Diego. His areas of research interests include parallel discrete event simulation, performance modeling of HPC sys-tems, applied mathematics, communication sys-tems and computer architectures.

PHIL ROMERO is a Computer Scientist with the High Performance Computing Environments Group (HPC-ENV) at Los Alamos National Lab-oratory. He obtained an MS in Mechanical En-gineering and an MBA from Stanford Univer-sity. His research interests include machine learn-ing, deep learning, data science and visualization, quantum computing, performance modeling, and simulation.



STEPHAN EIDENBENZ is a Computer Scien-tist with the Information Sciences Group (CCS-



1. at Los Alamos National Laboratory. Prior to that he was a Director of the Information Science and Technology (ISTI) institute at Los Alamos National Laboratory. He obtained a PhD from the Swiss Federal Institute of Technology, Zurich (ETHZ) in Computer Science. His research in-terests include cyber security, computational co-design, communication networks, scalable model-

ing and simulation, and theoretical computer science.



|  |  |
| --- | --- |
| VOLUME 4, 2016 | 9 |

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/.