SYS2VEC: System-to-Vector Latent Space Mappings

Theo Mahmut Bulut
Computer Science
University of Kaiserslautern
Kaiserslautern, Germany
Email: mahmutbulut0@gmail.com

Vasil L. Tenev
Embedded Systems Engineering
Fraunhofer IESE
Kaiserslautern, Germany
Email: vasil.tenev@iese.fraunhofer.de

Martin Becker
Embedded Systems Engineering
Fraunhofer IESE
Kaiserslautern, Germany
Email: martin.becker@iese.fraunhofer.de

***Abstract*—As a product line evolves, new members emerge and existing ones are maintained – more or less in sync with each other. In the context of long-living software and system product lines, the capability to predict evolution trends within the structure of the various assets is essential. It helps to understand the underlying dependencies between work items now and in the future and helps to make the product line architecture more robust against the predicted trends. With this, unnecessary erosion can be avoided and overall engineering efficiency can be increased. With the increasing complexity of today's systems, approaches that can identify and evaluate commonalities, variabilities, and interdependencies in a large number of complex product variants and versions are gaining importance. In order to increase efficiency, approaches that support an incremental analysis in the space and time dimension are desirable. A promising approach to this end is to map the versions of each variant to points in a vector space. Doing so, two challenges can be efficiently addressed: (i) the similarity measurement becomes the distance between vectors; and (ii) the estimation of evolutionary trends can be reduced to the well-known interpolation problem. In this paper, we present SYS2VEC – an approach for mapping product line variants into a latent vector space by means of machine learning techniques. With our approach, we are able to show an increase in accuracy by a factor of 4 and halve the execution time compared to similar machine-learning-based solutions.**

***Keywords*—System comparison; machine learning; graph similarity; product lines.**

I. Introduction

The rapid development in the embedded and IT world demands steady growth of customization flexibility for products and services. To satisfy this need, companies can do two things. On one side, new product variants can be forked and modified from existing ones or be derived using a strategic reuse approach, where a variation model exists [1]. On the other hand, existing product variants continue to develop in new versions. Such systems often evolve over long periods, during which they usually diverge from each other. Each new variant typically addresses new customer requirements, causing it to drift away from the product line' core. At the same time, variants can also grow closer together due to changes in the environment and the emergence of common architectural drivers.

Unfortunately, these tendencies are often not apparent, but it becomes essential to predict such evolution trends within the asset structure in the problem and solution space of a product line. It helps to understand the underlying dependencies between work items now and in the future and helps to make the product line architecture more robust against the predicted trends. With this, unnecessary erosion can be avoided and overall engineering efficiency can be increased. With the increasing complexity of today's systems, approaches that can identify and evaluate commonalities, variabilities, and interdependencies in a large number of complex product variants and versions are gaining importance. In the context of product line engineering, assets of the system architecture are considered. Requirements, architectural models, source code, configuration, and test artifacts could be subject to such analyses. In the following we focus on the analysis of architectural models, although the presented approach can be applied to other engineering artifacts as well.

To cope with the increasing complexity of today's systems, organizations are increasingly using model-based approaches in systems and software engineering. In these models, architecture, requirements, realization and quality assurance work items are tightly connected with each other. The models grow proportionally with the product line size and complexity. In principle, the models are highly interconnected graphs. Estimating the evolutionary trends for the architecture model aids in managing and planning of deviations that emerge between product variants and versions.

In order to increase efficiency of analyses in large-scale product lines, approaches that support an incremental analysis in the space and time dimension are desirable. A promising approach to this end is to map the versions of each variant to points in a vector space. Doing so, two challenges can be efficiently addressed: (i) the similarity measurement becomes the distance between vectors; and (ii) the estimation of evolutionary trends can be reduced to the well-known interpolation problem. This raises the question, of whether there is an efficient way to translate the structure of product line assets into a vector space.

In this paper, we present SYS2VEC – an approach for mapping system variants into a latent vector space by means of machine learning techniques. SYS2VEC approach is based on the Graph2Vec method, which uses Weisfeiler-Lehman hashing [2] to incorporate unified relabels for node features. With SYS2VEC, we are able to show an increase in accuracy by a factor of 4 and halve the execution time compared to similar machine-learning-based solutions. The paper provides the following *contributions*:

- it introduces an innovative approach to analyze system variants by representing them as graph-like structures in

a latent vector space,
- it discusses implementation aspects of the approach,
- presents validation results, and
- provides an overview on related machine learning approaches.

The remainder of the paper is structured as follows: Section II gives an overview on the approaches, techniques and tools related to this work. Section III discusses the main idea of using latent space representation for addressing the aforementioned challenges. In Section IV we present our approach and implementation in details. The validation with respect to our goals is shown in Section V. Section VI concludes the paper.

II. Related Work

In this section, we present the context in the conducted research work and provide an overview on machine-learning approaches, techniques and tools related to our approach.

1) Feature Extraction: In machine learning, feature extraction is a method to create input for ingestion by models. There are various ways to extract features from textual information. One of the most common methods is the Term Frequency-Inverse Document Frequency (TF-IDF) [3]. Some methods extract features to embedding space directly using Bag-of-words (BOW) methods with skip-gram models like WORD2VEC [4]. WORD2VEC extracts features to latent space straight from sentences. For feature extraction at the paragraph level with multiple sentences, DOC2VEC has been proposed. DOC2VEC has two approaches: one of them is Paragraph-Word Distributed Memory (PV-DM), and the other one is the Paragraph Vector-Distributed Bag Of Words (PV-DBOW) approach [5].

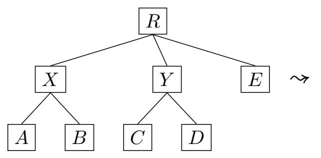
Since we do not need to incorporate individual word vectors from the fixed-length sliding window, we discard the PV-DM approach and use the PV-DBOW approach in our work. For our algorithm introduced in Section IV, vector output of WORD2VEC and DOC2VEC prevents preprocessing node attributes with vector input. Muhammad Isa et al. [6] have successfully applied TF-IDF for semantic feature extraction. They use frequency table output for graph representation learning. TF-IDF allows filtering of terms with low occurrence in the whole graph. Thus, a threshold can be used to filter the sparse matrix output of TF-IDF. In Sections IV, IV-B, and IV-C, we have detailed term and inverse document frequency related filtering to supply a better canonical input for the representation learning stages.

2) Unsupervised Machine Learning: Unsupervised machine learning methods learn the representations with minimal or no external intervention [7]–[10]. Unsupervised methods are broader in scope and fit the context of variant analysis better, as they incorporate hierarchies and node contents and features. One of the unsupervised approaches to feature learning over graph descriptions is OhmNet [11]. OhmNet originates from bioinformatics and generates multiple neural network layers for all protein sequences that are similar to each other. To this end, it trains a single machine learning model that maps the layers of protein structures to the corresponding tissue. In the context of variant analysis, each system variant corresponds to a different tissue. The training time depends on the number of system variants and increases linearly. Applied to our context, this method does not provide accurate prediction as it tends to converge to the global minima of all variants. Whereas in a real scenario, the variants may not even be related and should not even be considered and trained together.

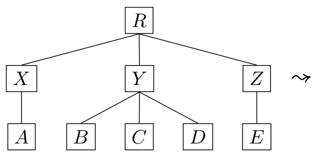
Non-Negative Matrix Factorization (NNMF) is one of the Blind Signal Separation (BSS) techniques widely used in unsupervised methods. It learns graph structures while incorporating both graph shape and node content into embedding space. One of the adapted NNMF method is FSCNMF (Fusing Structure and Content via Non-negative Matrix Factorization for Embedding Information Networks) [12], where the model uses attribute context of the nodes to learn node feature representation. This node based embedding generation method works with factorizing both adjacency and feature matrices. Node and feature embeddings are regularized together for create a representation in embedding space. It doesn't embed full graph shape, as it is supposed to be, since this method uses adjacency as main driver for embedding vector generation. The FSCNMF method outputs an N-dimensional representation, where N is the maximum nodes between training graphs. This output needs to be combined with manifold methods like t-distributed Stochastic Neighbor Embedding (t-SNE) [13] to produce a sensible output from the original output with dimensionality reduction techniques. When the methods are adopted in variant analysis context, multiple graphs of the same product line will fit the same model. That said, contextual inference will be high, but the graph shape will not be preserved very well.

Another approach for creating embedding for nodes, which utilizes a random-walk-based approach, is text-associated DeepWalk (TADW) [14]. TADW, like FSCNMF, factorizes textual node attribute matrix with actual representation matrix. Apart from FSCNMF, this method does random walks over the graph and creates matrices from the these walks. These walks are used as similarity measure for the graph's core representation. When adapted to variant analysis context, this method creates a single vector representation from both content and structure together. By default this method assumes that two graphs are different when they have different shapes. TADW doesn't apply to variant analysis domain very well, because content and structure might vary differently and if either one of them is different, they are assumed different by the learning method. Since this method also outputs an N-dimensional representation, it needs to be combined with manifold methods for more sensible outputs for the identification of formed clusters.

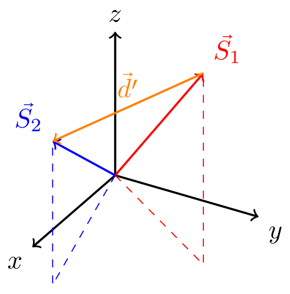
In addition to attribute-based graph embedding methods, whole graph embedding methods are most suitable to learn the representation of graphs [15]. They rely on various techniques and incorporate different partial information about the graphs. These methods can enrich an embedding graph structure by embedding edge features [16], by node features only [17], or by permutation preservation [18]. For large datasets, tracing



(a) Graph representation of system variant S_1.



(b) Graph representation of system variant S_2.



(c) Similarity measurement between S_1 and S_2.

Figure 1: Input system variants and output vector representations of example system S_1 and S_2.

based on Taylor series by eigenvalue decomposition is a hard task. Since first-order terms in Taylor expansions tend to approximate to a local maximum of the distribution, large graph eigenvalue growth rate cannot be estimated for large graphs [19]. These methods fallback is always hit for graphs, i. e., making the method's main optimization discarded for NetLSD.

The Weisfeiler-Lehman Kernel (WLK) can be used to incorporate graph topology information into the nodes and take the vertex neighborhood into consideration. WLK allows the quantization of neighborhood groups to the individual node attributes through hashing. Other than WLK, there are random walk kernels that do random walks with varying lengths over the graphs. DeepWalk is one of the methods that incorporate a random walk approach to learning the graph representation with unbiased, fixed-length, random walks starting from each node. According to Narayanan et al. [17], these methods tend to have low confidence values against the WLK based methods. Graph2Vec [2] is a method, which uses Weisfeiler-Lehman hashing to incorporate unified relabels for node features. Graph2Vec approach is well aligned with the variant problem, due to its graph representation learning. Furthermore, its base accuracy is good enough to deliver meaningful results out of non-linear substructures.

III. Latent Space Representation

Latent space is a multi-dimensional vector space, where representation vectors for variants are used to make correlation for similarity. The similarity between the variants is related to the angles between the vectors, their magnitude, and their orientation. For example, let S_1 and S_2 be two system variants (see Figures 1a and 1b). The similarity between them can be computed in three-dimensional space, as shown in Figure 1c. Here $\vec{S}_1$ is the vector representation of S_1, $\vec{S}_2$ represents S_2, and $\|d'\|$ corresponds to their similarity. In general, we work with finite-dimensional vectors in vector space, also known as finite-dimensional Hilbert space.

Distance measures between vectors enable computing similarities. We can compute the pairwise similarity between two variants in the vector space using the Euclidean norm. In Figure 1c, we can see that $\|d'\|$ is the Euclidean norm between two vector representations. For computing pairwise similarities, SYS2VEC relies on the Euclidean norm of vectors.

For having consistent vector representation for ingested systems coming out from SYS2VEC, variant hierarchies should have a consistent notation. Our algorithm uses PV-DBOW, which doesn't produce word vectors in a fixed window of elements like the other alternative model of DOC2VEC called Paragraph-Word Distributed Memory (PV-DM). But even though the order of the words in the graph's corpus is not essential for us. For this very reason, using a graph traversal notation is crucial to always get a consistent representation of the graph. When given a set of systems, our algorithm runs with Deep First Search (DFS) with vertex ordering notation of preordering. Since this approach can build a topological sort for nodes, it is a natural choice for the SYS2VEC. The underlying DOC2VEC [5] model works by doing subgraph sampling, and this approach can always yield the correct final form for any given subset of the graph.

IV. SYS2VEC Approach

SYS2VECs core processing pipeline and its processing stages can express a single variant with row vector representation. This Section discusses the algorithm's workflow and its stages in depth.

This section explains our main algorithm's stages which are:

- **System Variant Traverser.** Traversing systems structure and preparing a system variant as input for the normalization passes are explained in Section IV-A (cf. stage **1** in Figure 2).
- **Normalization Scheme.** In Section IV-B, we explain how system contents are normalized as input for the Weisfeiler-Lehman kernel and how its output is used by the representation learning (cf. stage **2** in Figure 2).
- **Representation Learning.** Section IV-C explains important arguments for SYS2VEC and its core representation learning model. In addition to that, we will discuss generated embedded space representations and their properties in Section IV-D (cf. stages **3** and **4** in Figure 2).
- **Latent Space Similarity.** Section III describes our approach of computing similarity based on the latent space representation (cf. stage **5** in Figure 2).
- **Method Optimization.** For getting most of the accuracy from our approach, we have developed model optimization. Section IV-E explains how SYS2VECs hyperparameter tuning approach works in detail. In addition to that, we explain how additional optimizations considered and incorporated into SYS2VEC.

A. System Variant Traverser

However the system represented digitally, we traverse through the system, and create variant trees. Our traversal algorithm (stage **1** in Figure 2) uses the Depth-First Search

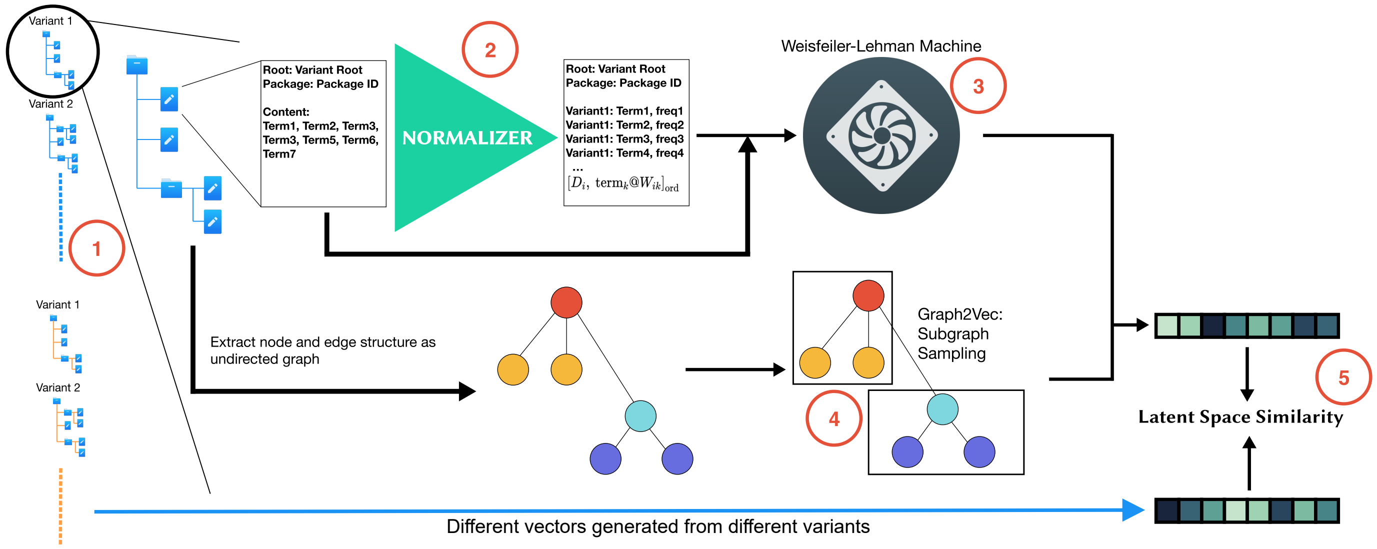


Figure 2: SYS2VEC, Similarity Seeking Variant Analysis Machine Learning Pipeline.

(DFS) to extract the hierarchical structure of the variants. DFS traversal gives us a consistent view between all graph variants and for their subsampling undergoing in DOC2VEC [5]. Since our data model is a generalized model for software variant graphs, our algorithm should also work with large graphs and datasets. Datasets containing elements over one billion nodes should be organized at scan time, which reveals the internal structure of graphs for achieving better results for machine learning models. Finding ordering with optimizing a cost function is solely an NP-hard problem. Since finding optimal ordering relies on vertex locality and can vary between graphs [20], we skip finding vertex ordering for each graph and utilizing DFS for all graphs.

B. Normalization Scheme

In order to do the graph core extraction using Weisfeiler-Lehman Kernel (WLK), [2], data of content filled elements should be normalized. In Section II-2, we explained the necessity of WLK pass for the extraction of graph topology. The content normalization scheme we have defined (stage **2** in Figure 2) will be used for creating Weisfeiler-Lehman hashable features after traversal (see Figure 3). Moreover, through a content-oriented normalization scheme, terms of low importance (aka lonely terms) can be optionally detected and eliminated.

Reduction parses all content definitions and then creates vocabulary with unique terms received after tokenization. Based on this vocabulary, first term frequencies and then graph-wide inverse document frequencies are generated. Learned frequencies produce document-term matrices based on single key-value tuples reduced from their sparse representations held in TF-IDF. By reduction, these corresponding tuple sets are added to ordered feature sets in all nodes of the graph. Normalized contents are generated with this method refills content filled elements to make graphs ready to be processed by WLK in the next stage.

```
1:  procedure ContentNorm($V, c$)  ▷ Where V - array of vertices' contents, c - count of all content filled elements in variants
2:      $X = makevectorizer()$  ▷ Create TF-IDF vectorizer
3:      $D = X(V)$  ▷ Process variants through vectorizer
4:      Let $L[1 \ldots c]$ be new array
5:      for $j = 1$ to $c$ do  ▷ Iterate over each content filled element
6:          $k = len(V[j])$  ▷ Get number of TF-IDF tuples
7:          $\mathbf{A}_j = create\_ordered\_set\_with\_size(k)$  ▷ Create ordered set for content filled element at index $j$
8:          for $b = 1$ to $k$ do
9:              $A_j.append(D[V[b]])$  ▷ Append TF-IDF pairs to ordered set
10:         end for
11:         $L[j] = A_j$  ▷ Assign content filled element's ordered set to the aggregation set
12:     end for
13:     return $L$
14: end procedure
```

Figure 3: Content Normalization.

After reduction, vertex contents are made into a TF-IDF document matrix. At this state, every vertex content state has filled key-value dictionary item. Before fed into Weisfeiler-Lehman Kernel, normalized inverted index pairs in ordered dictionaries can further be ranked. Moreover, if needed, lonely terms can be filtered by their threshold of occurrence. In addition to all the previous steps, preprocessing can be done [21]. In addition to that, the proposed normalization scheme can work with an ample amount of content (which is bounded by the underlying TF-IDF implementation). Since every normalization run is a one time pass for a single graph, most of the time is taken for building the vocabulary for graphs with a large degree. In [22], authors define ”large graphs” as graphs with hundreds to thousand nodes. Our experiments showed that SYS2VEC already works with approximately 2400 and more nodes. We detailed our experiments in Section V-A.

The researcher can configure the tokenization algorithm for the TF-IDF system. Standard structured formats like XML and JSON can easily be converted to dictionaries and then fed into dictionary-based tokenizer. Dictionary-based tokenization allows SYS2VEC to consume different system views like control flows, configuration files, structured binary representations, and various other formats by a simple adaptation of normalizer. For control flow graphs, one can use the original SYS2VEC repetitively for every node’s control flow. Since control flows are applying the rule of rooted graphs, they can be usable with SYS2VEC. When it comes to configuration files, text objects can be either translated into record formats like CSV or TSV then fed into the normalization scheme with a comma or tab-separated tokenization. When compared to control flows and file configurations, binary objects are needed to be either parsed into structured plain-text representations or encoded with binary-to-text encoding algorithms. The latter solution decreases TF-IDF confidence in normalization passes since binary data (when directly encoded into text) won’t give good term frequency but rather give inverse document frequency. For dealing with binary data, recent bioinformatics papers [23] [24] followed an approach where authors created weighted sparse binary matrices with a predefined aggregation window and then applied TF-IDF for normalization.

C. Representation Learning

Fundamentally, the algorithm creating latent space representation of the system variants should be task agnostic with a row vector output. Task agnosticism needed for whatever input features given, we should be able to produce a row vectors for the given system variants. Since we need to compare whole graphs, we take advantage of the paragraph vector-distributed bag of words (PV-DBOW) [5] representation used in Graph2Vec [17] . Since Graph2Vec is a transductive approach, intrinsically, SYS2VEC is transductive [25]. In SYS2VEC, ”transductive” means that vector representations are generated for a given fixed set of graphs to generate vector space lookup matrix.

Since our variant representations are rooted graphs, Graph2Vec does subgraph matching over the given set of system variants (stage **4** in Figure 2). During the normalization of the previous step, feature dictionaries were created for all nodes. These features are hashed for compression with their neighborhood similarities with Weisfeiler-Lehman kernel (stage **3** in Figure 2). Thus at each iteration phase, neighborhoods and features have been stored in nodes. Finally, graphs with aggregated data in their nodes dispensed and can be feed into DOC2VEC [5] to learn the representation with subgraph sampling followed by matrix factorization steps.

Representation learning for graph clustering doesn’t incorporate task-specific information from SYS2VECs reduction mechanism. This feature of Graph2Vec allows us to extend or manipulate feature dictionaries in the content nodes to represent relations. Moreover, the separation of SYS2VECs reduction mechanism from the representation learning model provides great flexibility to change the underlying model with other techniques in the field. Example relations like includes excludes at the package level, composition, and aggregation for classes can be appended to content dictionaries and preprocessed with the same workflow that we have described Section IV-B. Graph2Vec performs better generic representation when task-specific information doesn’t factorize together with the underlying PV-DBOW model [17]. Via underlying PV-DBOW, DOC2VEC skip-gram training learns the graph representation interpreted as document representation and yields a row vector.

Configuration of SYS2VEC relies on various parameters; these parameters are scattered through the pipeline components seen in Figure 2. Parameters below adjust representation learning from graphs in SYS2VEC. Variant traverser (in Figure 2 marked with **1**) receives these arguments:

Vertex ordering notation: Vertex ordering that nodes placed to form the graph after traversal. Ordering notation can change how row vector values are displaced through representation learning. Indirectly it changes vector orientation and magnitude in Hilbert space.

Tokenization scheme for creating the variant knowledge corpus: Tokenization procedure given to tokenizer for extracting sensible words as input for TF-IDF algorithm underneath. By default, SYS2VEC comes with word tokenization, which allows feature extraction by parsing two or more characters that assemble a word. Tokenization can be changed to adapt to structured data formats like XML, JSON, or blobs. SYS2VEC can create meaningful normalized features from existing variant data when tokenization is adapted. Adaptation can be made by changing the regular expression of tokenizer or supplying a custom function closure.

(Optional) Normalization stage bypass argument for experimenting with untouched vertex features: Optional flag for skipping normalization stages as will. By default, skipping normalization is disabled. If features are not meant to be preprocessed, TF-IDF reverse lookup can be skipped. In that case, Weisfeiler-Lehman will process all of the non-normalized feature set. Our experimentation showed that if the non-normalized feature set is in key-value format, this doesn’t change representation learning. Moreover, it doesn’t change the pairwise similarity computation. If nodes have heterogenous data scattered among them, it is better to use normalization passes to create homogenous representation for the graph.

Learning rate of the model: A factor that enables how much generalization should be memoized from the previous iteration of the training. Feature slices given to DOC2VEC should incorporate all details of normalized feature sets. For this reason, the learning rate during the training should be controlled for how much generalization memoized from a batch. A higher learning rate lowers the generalization, and a lower learning rate won’t incorporate most of the features given in a graph. Finding a good learning rate is an experimental discovery. A good learning rate gives a low loss value but won't prolong the model's training duration. For this exact reason, we have a learning rate as a parameter to optimize at the hyperparameter optimization stage in Section IV-E.

Output vector dimensions: Graph representation's row vector size, in other words, dimension output per variant. In Hilbert space, vector dimensions are essential to represent generalized notions of graphs. The row vector representations should have enough dimensions for variants to achieve better vector space similarity. Increasing dimensions can output better vectors for condensed features in content filled elements. When data is not enough, higher dimensions won't create a difference. Hyperparameter tuning is utilized to detect an optimized dimension count (as explained in Section IV-E).

Frequency threshold for features: Before running Weisfeiler-Lehman hashing, TF-IDF pairs can still have lonely terms that mightn't contribute to overall graph representation learning. This threshold behaves like a high-pass filter for TF-IDF frequencies in the ordered set. SYS2VEC comes with the tuning option of filtering terms with a low term occurrence frequency (see Section II-1). We have explained that option in Section IV-B as lonely term filtering. This option filters learned TF-IDF frequencies below a particular threshold value. SYS2VEC doesn't implement key-specific inverse transform for filtering by specific thresholds. Since filtering specific keys is an expert decision, and we don't want to incorporate another function to increase the time complexity of the processing. Not having additional key-specific filtering doesn't have downsides for SYS2VEC.

Epochs: Defines how many times full dataset batches are fed into the model. Almost all machine learning models have epochs to adjust how many times batches are fed as a complete cycle to a model. When epochs increased, subsampling for PV-DBOW will give different data attention and weights to update vector representations. Increasing epochs for large datasets enables producing more generalized vector representation. If epochs are kept less, the model can underfit and will not generalize enough to make meaningful vector space representations. Since accuracy determines the generalization of the model and generalization directly dependant on epochs, this value is also optimized by our hyperparameter optimization scheme (see Section IV-E).

D. Embedding Space Representation

After system traversing, normalization, and graph kernel extraction, learned graph representations need careful tuning from the machine learning model parameters' perspective. As explained in Section IV-C, subgraph representations are incorporated into embedding space with the influence of both SYS2VECs parameters and underlying Graph2Vec model parameters. The dimension for vector representation wasn't selected arbitrarily. Higher dimension count gives longer training times and sparse graph representations. Thus accuracy for the similarity correlation decreases due to heavy entropy interference and sparsity of generalized model parameters. Mentioned entropy interference prevents convergence of the model for very high output dimensions. In [26], authors noting having more parameters to generalize will increase the error rate of the model inference. When dimensionality increased parallel to model parameters, the model's accuracy will first seem to improve but then decrease drastically. This phenomenon is called Hughes phenomena [27]. In [28], empirical observations showed that after 1000 dimensions, convergence to at most 50% probability of error is relatively slow.

In contrast to that, giving a low dimension to the SYS2VEC prevents capturing all given system variants' features. Similarity metrics don't diverge when no attention is given to the system variants specific properties. For this reason, output embedding dimensions should carefully be picked between multiple systems. Selecting the optimum dimension can be automatized and optimized by various parameter optimization methods, which are explained in section IV-E.

Representation learned from graphs projected from latent space to embedding space and generates row vectors for every system variant individually. SYS2VEC heavily relies on PV-DBOWs embedding projection. Embedding representations for SYS2VEC is unchanged Graph2Vec embeddings. After pairwise similarity computations finished, dimensionality reductions can use embeddings to visualize vectors' density in vector space.

Apart from the embedding generation, there are various improvements suggested for PV-DBOW. PV-DBOWs embedding projection uses dot product similarity for word-paragraph correlation. In SYS2VEC, the mentioned projection corresponds to a random subsample of a graph against the whole graph. For this reason, representations are heavily influenced by the distribution of subsamples over a graph. Though there are suggestions to improve PV-DBOW for better embedding generation for randomly distributed subsamples [29], we didn't consider subsample randomness as a problem in a diverse set of systems and their graphs since they are not relevant to our approach.

E. Method Optimization

Representation vector generation after model training gives output vectors for every variant. Since SYS2VEC is taking parameters mentioned in Section IV-C, accuracy against the validation dataset varies with different model parameters for both SYS2VEC and underlying Graph2Vec implementation.

Successful representation vector and similarity matrix generation don't mean that our system is perfectly working with its full potential. For getting better accuracy from the model, hyperparameter optimization methods exist. For this purpose, we have included an automatic hyperparameter tuning method inside SYS2VEC. Our autotuning optimization strategy is using the Optuna framework, which is a cutting-edge black-box optimization toolbox for machine learning models [30]. We have defined Optuna's optimization goal for its' black-box approach as the maximization of the accuracy scalar defined by pairwise similarity of sampled variants throughout trials of representation learning with different parameter spaces, which the framework has estimated throughout the hyperparameter optimization.

SYS2VECs hyperparameter tuning configures the parameter spaces listed below for maximizing the accuracy with given categorical and range-based parameter space. Parameter spaces are manually selected and tweaked based on the knowledge about our problem. First, we have tested minimum and maximum values of parameters with our approach, then we set them as a parameter range:

- **Weisfeiler-Lehman iterations over the system variants.** Integer parameter space from 2 to 10.
- **Output dimensions.** Categorical parameter space with powers of 2 as in 128, 256, 512, 1024, 2048, 4096.
- **Epochs** Categorical parameter space with multiples of 10 as in 10, 20, 30, 40, 50.
- **Learning rate** Uniform distribution between 0.005 and 1.0.

The aforementioned parameters and their parameter space are explored and tested for the convergence to the global maximum during the hyperparameter optimization. After the hyperparameter optimization trials finish, the framework outputs the best parameter selection and accuracy received with these parameters. If received accuracy is better than our method's initial accuracy, we are accepting this parameter set as the new basis for SYS2VEC. We have observed that after optimizations successfully finished, we see nearly 10% accuracy improvement against the validation dataset.

A tree-structured Parzen estimator [31] does parameter sampling for the hyperparameter optimization. SYS2VEC comes with hardcoded 100 trials for optimization passes, which is seen enough during our optimization trials. Since sampling and objective function seeks the global maximum throughout the optimization, we haven't defined any early termination criteria when no accuracy improvement has been seen during these trials.

With hyperparameter optimization, accuracy levels rise to 64%. When our optimization for hyperparameters runs, it lays out better quality than the out-of-the-box method. These results can be compared with the accuracy of the graph2vec model [17] using the MUTAG dataset. MUTAG is a data set containing graphs of 188 chemical compounds labeled with respect to their mutagenic effect on bacteria. MUTAG dataset can be processed by SYS2VEC since our method's core for learning graph representations is the same as Graph2Vec. Generated representations will be the same as processing data through the Graph2Vec method.

SYS2VEC consists of various runtime optimizations in its stages, which makes representation generation and similarity computation faster compared to unoptimized interpreted code:

Embedding generation: The DOC2VEC implementation we are using has C extensions for parallelizing the computation and working faster with native code. In our experiments, precompiled C implementation brought nearly sixty times faster epochs than pure Python implementation [32].

Similarity calculations: SYS2VECs vector space similarity calculation is using LLVM's vectorization backend for faster similarity computation through the Numba just-in-time (JIT) compiler for Python [33]. Vectorization enables us to use SIMD instructions dedicated to vector processing. Normally this optimization is only possible with a backend dedicated to analyzing loop specialization. LLVM's JIT compilation engine gives code auto-vectorization and enables faster similarity computation for our case [34]. We can compute similarities for the given variant pairs faster with single instruction multiple data (SIMD) instructions. The mentioned JIT compiler only compiles pairwise similarity distance methods for variants to enable compiled bytecode reuse across the whole variant ranges during similarity computation. This compilation happens once at the very beginning when vector space similarity calculation starts; after that, it is cached to reuse across all given systems [33].

Model and vectorizer caching: TF-IDF vectorizer used in the normalization scheme can save the vocabulary and don't need to rebuild between the runs of the SYS2VEC. SYS2VEC can freeze models and load and save dictionaries and embeddings. These actions allow bypassing the traversing stage if needed and enable faster experimentation over the variants. Moreover, it allows running inference tasks on SYS2VEC. These optimizations are incorporated for improving the performance for processing high volume variant data in SYS2VEC.

V. Validation

In this section, we validate the SYS2VEC method and compare it with other approaches in the field, following different approaches to solve the similarity computation for system variants.

The main validation obstacle for the research is having a broader set of graph data to test full accuracy against the approximation-based methods in the field. Graph datasets used for validation in research either lack graph counts (cf. Zitnik et al. [11], where authors use a dataset with 219 graphs describing cellular systems) or the amount of nodes per graph is way less (cf. Bai et al. [7], where authors have at most 100 nodes). Nevertheless, initial observations showed that SYS2VEC performs with good accuracy for the graph-based similarity analysis.

Our method uses the transductive approach of Graph2Vec [17], but it is flexible enough to use another representation learning method. Stages of SYS2VEC can be independently used to feed graph data back into the deep learning techniques like Graph Matching Networks (GMN) [35] [36] for representation learning. In addition to that, our approach's accuracy baseline without hyperparameter tuning is 55%. With hyperparameter tuning, it goes beyond 65%. Additionally, SYS2VEC is flexible for adaptation with various model bases as mentioned in Section IV-C.

A. Correctness

As shown in Table I, SYS2VECs dataset contains five systems by the same tier one supplier with variant count varying from 16 to 91 variants for each system. Each variant corresponds to a software-hardware management system for

TABLE I: SUMMARY OF THE DATASET USED FOR THE EXPERIMENTATION.

System Name	Variant Count	min. Nodes	max. Nodes
System Z	63	1864	3462
System Y	23	952	2458
System X	16	1079	3267
System W	91	1209	2406
System V	51	229	3397

TABLE II: RELATIVE ACCURACY AND ERROR BOUND OF SYS2VEC COMPARED TO OTHER METHODS.

Algorithm	**Accuracy (avg. ± std.)**	**δ - Mean Error Bound**	**Total Runtime**	**Algorithm Type**
Multiple Alignment	87.2 ± (0.2)	∽ 12.8	<1 hour	Deterministic
OhmNet	16.1 ± (0.9)	∽ 80.0	∽ 3 hours	ML based
SYS2VEC	65.4 ± (2.3)	∽ 42.0	<30 minutes	ML based

an automotive facility. More concretely, we analyzed the configuration data structured in hierarchical sets, where each node in the set represents a pair of a configuration item and the associated value. Our approach is validated against the similarity computation method, called Multiple Alignment, developed by Tenev [37] and additionally described by Duszynski [38].

In Table II, we have compared our method with relative success rate against similarity heuristic of multiple alignment method in [37]. The relative accuracy of SYS2VEC against the deterministic Multiple Alignment method [37] is lower than expected. But SYS2VEC can be used as internal heuristics to guide Greedy algorithms to converge to optimal alignment solution as an alternative to the proposed modified Center-Star heuristic [37]. With an error rate of likely 40 percent against the approach mentioned in [37], [39], SYS2VEC can also be used to get an overview of which variants are highly similar and what is possible evolutionary tree can be consolidated from them.

Our next aim to improve SYS2VEC is to incorporate accuracy improvements. The main breakthrough of SYS2VEC is operating on graphs without extra micro-tuning for their content and structure. SYS2VEC creates embedding, which is reusable across various vector space algorithms. Moreover, our method is flexible enough to substitute the embedding generation model with different algorithms. This substitution will be towards improving the accuracy of the representation learning by experimenting with deep neural networks.

B. Comparison to Other Techniques

Experiments ran with various methods showed that our method's transductive approach exploits the representation learning and similarity calculation well over the large graphs. As shown with the dataset specifications in Table I, our method handed out better accuracy than other machine learning methods, which can be observed from Table II. In addition to that, total runtime where the sum of training and inference runtime is way less compared even to deterministic methods. Moreover, since it doesn't need Graph Edit Distance (GED), or Tree Edit Distance (TED) as a supervisory signal for representation learning, it's time complexity is lower than other methods. As shown in Table II, Zhang-Sasha [9] TED approximations are taking ample amount of total runtime on very large graphs. For similarity calculation, we aim for an interactive analysis approach; for this reason, the Zhang-Sasha-based SimGNN method is not suitable with a runtime of 2 days. This timing behavior is too long, and it is not useful for practical use case scenarios. In addition to that, GED calculation is NP-hard [40]. For this reason, we have also experimented with approximate GED methods with error bound [41]. But these methods time complexity increases exponentially with respect to the graph's node count, like in the Zhang-Sasha case in Table II. Above a certain degree of the graphs, machine learning methods execution is dedicated mostly to the edit distance calculation.

Experiments showed that SYS2VEC has an acceptable error rate against the baseline of the greedy alignment in [37]. Unlike edit distance-based approaches, our approach can work with a vast amount of data. Also, SYS2VEC doesn't need to learn for correspondence of certain systems into vector representations like OhmNet. Since our systems are separated, and the learning correspondence doesn't bring value for generalizing representation learning, the OhmNet [11] method does not add extra features to the whole-graph embedding generation procedure.

Apart from Machine Learning methods we have experimented, Clustal [42] has been used on our dataset. Initial experiments showed that Hidden Markov Model (HMM) states are way larger than bioinformatics problems. Multiple sequence alignment tools like Clustal takes node coloring into consideration, which corresponds to nucleobases. For system variant analysis case, this can grow based on the features generated from the normalization scheme, and later combined for neighborhood feature incorporation using Weisfeiler-Lehman hashing. Thus, for large graphs, node coloring can have way more elements than nucleobases. While bioinformatics problems have well-known nucleobases (AGCT) to accept as node coloring, they are less than our dataset's distinct variant count. Thus, HMM-based tools are not working with an arbitrary number of node coloring for fixed ordered graphs.

Nevertheless, we have experimented with this approach by changing the Clustal Omega code to accept UTF-8 table characters to see how it will be done in practice. But this gave us a lot of states which took a long time to process. So we have to terminate our experiment because of this method's time complexity. Even with modified Clustal code, we haven't successfully made it work with systems with 91 variants.

VI. CONCLUSION AND FUTURE WORK

In this paper, we introduce SYS2VEC, a new approach that analyzes system variants by combining a novel normalization scheme with unsupervised machine learning for fast graph comparison. Moreover, our method improves the time performance in comparison to other machine-learning approaches and can easily be applied to existing system

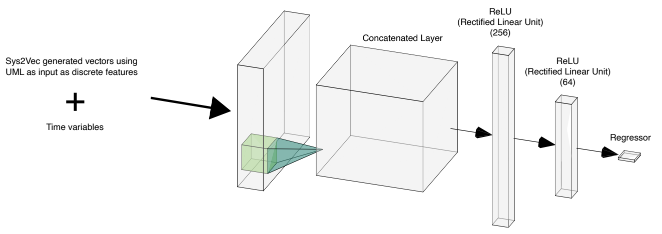

Sys2Vec generated vectors using UML as input as discrete features
+
Time variables
Concatenated Layer
ReLU (Rectified Linear Unit) (256)
ReLU (Rectified Linear Unit) (64)
Regressor

Figure 4: Example evolutionary tree deep learning methodolody using SYS2VEC.

variants. SYS2VEC is exploiting the stochastic nature of the machine learning models to decrease the time taken for pairwise similarity comparison using row vectors in the finite-dimensional vector space. Additionally, our approach enables inference tasks like clustering and trend estimation using supervised methods for variant analysis.

In addition to creating an unsupervised graph comparison method for variant analysis, we have explored various existing machine learning models (e. g., SimGNN [7], FSCNMF [12], OhmNet [11], TADW [14]) for either learning graph structures in a supervised manner to create a heuristic for alignment methods, or learning whole-graph representations to utilize similarity metrics to assemble a comparison heuristic for known graphs.

A. *Future Work*

SYS2VEC can be used to predict evolution trends with supervised learning methods like LSTM. Supervised methods can be trained with time-series data using our vector representation per time slice to predict the evolution of software and system products. The inference of evolution trends will help to consolidate product lines and will make variant aggregation easier. For this, first using Unified Modeling Language (UML) as variant content input for the SYS2VEC for generating similarity, and then using these similarities and time information to building evolution steps with UML evolutionary notation [43] is possible. As shown in Figure 4, utilizing deep learning and selecting time as a continuous feature in addition to discrete embeddings generated by SYS2VEC is possible to model evolutionary trees.

Our future work will include exploring other similarity measurement methods for variant analysis in SYS2VEC. These include, but are not limited to, Jensen-Shannon distance, first or second Kulczynski coefficients, and Hellinger distance. These methods are promising, since better similarity results can be achieved with them for document learning and classification tasks compared to Euclidean and Manhattan distances [44].

In future research, good exploration can be on Graph Matching Networks [35] as a supervised message passing neural network (MPNN) for embedding generation stage to improve the method's accuracy. Moreover, one of our future explorations will be on unsupervised learning methods like embedding propagation [45]. For improving ordering passes in the normalization scheme, experimenting on token position preserving advanced embedding techniques like Google BERT [46] is another way to improve embedding structure for SYS2VEC. In the long term, we focus on developing time-series-based evolution-prediction (as suggested in Figure 4) with the help of SYS2VEC vector space mapping output, auto-tuning for learning, and the normalization scheme improvements. These improvements will enable aggregating DTIs and allow experimenting on product line consolidation with machine learning methods.

References

[1] D. Faust and C. Verhoef, "Software product line migration and deployment," *Software: Practice and Experience*, vol. 33, no. 10, pp. 933–955, 2003.

[2] N. Shervashidze, P. Schweitzer, E. J. Van Leeuwen, K. Mehlhorn, and K. M. Borgwardt, "Weisfeiler-Lehman graph kernels," *Journal of Machine Learning Research*, vol. 12, pp. 2539–2561, 2011.

[3] Q. Liu, J. Wang, D. Zhang, Y. Yang, and N. Wang, "Text features extraction based on tf-idf associating semantic," *2018 IEEE 4th International Conference on Computer and Communications, ICCC 2018*, pp. 2338–2343, 2018.

[4] T. Demeester, I. Sutskever, K. Chen, J. Dean, and G. Corado, "Distributed Representations of Words and Phrases and their Compositionality," *EMNLP 2016 - Conference on Empirical Methods in Natural Language Processing, Proceedings*, pp. 1389–1399, 2016.

[5] Q. V. Le and T. Mikolov, "Distributed Representations of Sentences and Documents," *31st International Conference on Machine Learning, ICML 2014*, vol. 4, pp. 2931–2939, may 2014.

[6] S. Muhammad Isa, R. Suwandi, and Y. Pricilia Andrean, "Optimizing the Hyperparameter of Feature Extraction and Machine Learning Classification Algorithms," Bina Nusantara University Jakarta, Tech. Rep. 3, 2019.

[7] Y. Bai, H. Ding, S. Bian, T. Chen, Y. Sun, and W. Wang, "SimGNN: A Neural Network Approach to Fast Graph Similarity Computation," *WSDM 2019*, 2020.

[8] C.-L. Lin, "Hardness of approximating graph transformation problem," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 1994, vol. 834 LNCS, pp. 74–82.

[9] K. Zhang and D. Shasha, "Simple Fast Algorithms for the Editing Distance between Trees and Related Problems," *SIAM Journal on Computing*, vol. 18, no. 6, pp. 1245–1262, dec 1989.

[10] E. D. Demaine, S. Mozes, B. Rossman, and O. Weimann, "An optimal decomposition algorithm for tree edit distance," *ACM Transactions on Algorithms*, vol. 6, no. 1, pp. 1–19, dec 2009.

[11] M. Zitnik and J. Leskovec, "Predicting multicellular function through multi-layer tissue networks," *Bioinformatics*, vol. 33, no. 14, pp. i190–i198, jul 2017.

[12] S. Bandyopadhyay, H. Kara, A. Kannan, and M. N. Murty, "FSCNMF: Fusing Structure and Content via Non-negative Matrix Factorization for Embedding Information Networks," *CoRR*, apr 2018.

[13] L. Van Der Maaten and G. Hinton, "Visualizing data using t-SNE," *Journal of Machine Learning Research*, vol. 9, pp. 2579–2625, 2008.

[14] C. Yang, Z. Liu, D. Zhao, M. Sun, and E. Y. Chang, "Network representation learning with rich text information," Tsinghua University and HTC Beijing Advanced Technology and Research Center, Tech. Rep., 2015.

[15] H. Cai, V. W. Zheng, and K. C. C. Chang, "A Comprehensive Survey of Graph Embedding: Problems, Techniques, and Applications," *IEEE Transactions on Knowledge and Data Engineering*, vol. 30, no. 9, pp. 1616–1637, 2018.

[16] H. Chen and H. Koga, "GL2vec: Graph Embedding Enriched by Line Graphs with Edge Features," in *Neural Information Processing - 26th International Conference, ICONIP 2019, Sydney, NSW, Australia, December 12-15, 2019, Proceedings, Part III*, ser. Lecture Notes in Computer Science, T. Gedeon, K. W. Wong, and M. Lee, Eds. Cham: Springer International Publishing, 2019, vol. 11955, pp. 3–14.

[17] A. Narayanan, M. Chandramohan, R. Venkatesan, L. Chen, Y. Liu, and S. Jaiswal, "graph2vec: Learning Distributed Representations of Graphs," *28th Modern Artificial Intelligence and Cognitive Science Conference, MAICS 2017*, pp. 189–190, jul 2017.

[18] A. Tsitsulin, D. Mottin, P. Karras, A. Bronstein, and E. Müller, “NetLSD: Hearing the Shape of a Graph,” in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. New York, NY, USA: ACM, jul 2018, pp. 2347–2356.

[19] C. Moler and C. Van Loan, “Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later,” *SIAM Review*, vol. 45, no. 1, pp. 3–49, 2003.

[20] K. Zhao, Y. Rong, J. X. Yu, J. Huang, and H. Zhang, “Graph Ordering: Towards the Optimal by Learning,” *CoRR*, jan 2020.

[21] T. N. Phan, J. Küng, and T. K. Dang, “An Efficient Similarity Search in Large Data Collections with MapReduce,” in *Future Data and Security Engineering - First International Conference, FDSE 2014, Ho Chi Minh City, Vietnam, November 19-21, 2014, Proceedings*, ser. Lecture Notes in Computer Science, T. K. Dang, R. Wagner, E. Neuhold, M. Takizawa, J. Küng, and N. Thoai, Eds. Cham: Springer International Publishing, 2014, vol. 8860, no. May 2019, pp. 44–57.

[22] Y. Tian and J. M. Patel, “TALE: A tool for approximate large graph matching,” EECS Department, University of Michigan, Tech. Rep., 2008.

[23] D. A. Cusanovich, A. J. Hill, D. Aghamirzaie, R. M. Daza, H. A. Pliner, J. B. Berletch, G. N. Filippova, X. Huang, L. Christiansen, W. S. DeWitt, C. Lee, S. G. Regalado, D. F. Read, F. J. Steemers, C. M. Disteche, C. Trapnell, and J. Shendure, “A Single-Cell Atlas of In Vivo Mammalian Chromatin Accessibility,” *Cell*, vol. 174, no. 5, pp. 1309–1324.e18, 2018.

[24] M. Moussa and I. I. Măndoiu, “Single cell RNA-seq data clustering using TF-IDF based methods,” *BMC Genomics*, vol. 19, no. Suppl 6, 2018.

[25] M. Grohe, “Word2vec, node2vec, graph2vec, X2vec: Towards a Theory of Vector Embeddings of Structured Data,” *Proceedings of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pp. 1–16, 2020.

[26] M. R. B. Clarke, R. O. Duda, and P. E. Hart, “Pattern Classification and Scene Analysis.” *Journal of the Royal Statistical Society. Series A (General)*, vol. 137, no. 3, p. 442, 1974.

[27] G. F. Hughes, “On the Mean Accuracy of Statistical Pattern Recognizers,” *IEEE Transactions on Information Theory*, vol. 14, no. 1, pp. 55–63, 1968.

[28] G. V. Trunk, “A Problem of Dimensionality: A Simple Example,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-1, no. 3, pp. 306–307, jul 1979.

[29] J. H. Lau and T. Baldwin, “An Empirical Evaluation of doc2vec with Practical Insights into Document Embedding Generation,” in *Proceedings of the 1st Workshop on Representation Learning for NLP*. Stroudsburg, PA, USA: Association for Computational Linguistics, 2016, pp. 78–86.

[30] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, “Optuna: A Next-generation Hyperparameter Optimization Framework,” *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 2623–2631, jul 2019.

[31] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, “Algorithms for hyper-parameter optimization,” in *Proceedings of the 24th International Conference on Neural Information Processing Systems*, ser. NIPS’11. Red Hook, NY, USA: Curran Associates Inc., 2011, p. 2546–2554.

[36] X. Ling, L. Wu, S. Wang, T. Ma, F. Xu, A. X. Liu, C. Wu, and S. Ji, “Multilevel Graph Matching Networks for Deep Graph Similarity Learning,” *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–15, jul 2020.

[32] R. Řehůřek and P. Sojka, “Software Framework for Topic Modelling with Large Corpora,” in *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*. Valletta, Malta: ELRA, May 2010, pp. 45–50, %urlhttp://is.muni.cz/publication/884893/en.

[33] S. K. Lam, A. Pitrou, and S. Seibert, “Numba: A LLVM-based Python JIT Compiler,” in *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC - LLVM ’15*. New York, New York, USA: ACM Press, 2015, pp. 1–6.

[34] A. H. Ashouri, W. Killian, J. Cavazos, G. Palermo, and C. Silvano, “A Survey on Compiler Autotuning using Machine Learning,” *ACM Computing Surveys*, vol. 51, no. 5, pp. 1–42, jan 2019.

[35] Y. Li, C. Gu, T. Dullien, O. Vinyals, and P. Kohli, “Graph Matching Networks for Learning the Similarity of Graph Structured Objects,” DeepMind, Google, Tech. Rep., 2019.

[37] V. Tenev, “Directed coloured multigraph alignments for variant analysis of software systems,” Bachelor Thesis, University of Kaiserslautern, Germany, 2011.

[38] S. Duszynski, *Analyzing similarity of cloned software variants using hierarchical set models*, ser. PhD theses in experimental software engineering. Stuttgart: Fraunhofer Verlag, 2015, vol. 51.

[39] D. Gusfield, “Efficient methods for multiple sequence alignment with guaranteed error bounds,” *Bulletin of Mathematical Biology*, vol. 55, no. 1, pp. 141–154, jan 1993.

[40] L. Chang, X. Feng, X. Lin, L. Qin, W. Zhang, and D. Ouyang, “Speeding Up GED verification for graph similarity search,” *Proceedings - International Conference on Data Engineering*, vol. 2020-April, pp. 793–804, 2020.

[41] Z. Abu-Aisheh, R. Raveaux, J. Y. Ramel, and P. Martineau, “An exact graph edit distance algorithm for solving pattern recognition problems,” *ICPRAM 2015 - 4th International Conference on Pattern Recognition Applications and Methods, Proceedings*, vol. 1, pp. 271–278, 2015.

[42] F. Sievers, A. Wilm, D. Dineen, T. J. Gibson, K. Karplus, W. Li, R. Lopez, H. McWilliam, M. Remmert, J. Söding, J. D. Thompson, and D. G. Higgins, “Fast, scalable generation of high-quality protein multiple sequence alignments using Clustal Omega,” *Molecular Systems Biology*, vol. 7, no. 539, 2011.

[43] R. France and J. Bieman, “Multi-view software evolution: a UML-based framework for evolving object-oriented software,” in *Proceedings IEEE International Conference on Software Maintenance. ICSM 2001*, no. Icsm. IEEE Comput. Soc, 2001, pp. 386–395.

[44] K. Rieck KONRADRIECK and F. Pavel Laskov PAVELLASKOV, “Linear-Time Computation of Similarity Measures for Sequential Data,” *Journal of Machine Learning Research*, vol. 9, pp. 23–48, 2008.

[45] A. Garcia-Duran and M. Niepert, “Learning Graph Representations with Embedding Propagation,” *Advances in Neural Information Processing Systems*, vol. 2017-Decem, no. Nips, pp. 5120–5131, oct 2017.

[46] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding,” *NAACL HLT 2019 - 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies - Proceedings of the Conference*, vol. 1, no. Mlm, pp. 4171–4186, oct 2018.