Deepdense: Enabling Node Embedding to Dense Subgraph Mining

Walid Megherbi, Mohammed Haddad, Hamida Seba

DeepDense: Enabling Node Embedding to Dense Subgraph Mining

This is a preprint: the final paper is published in Expert Systems with Applications
https://www.sciencedirect.com/science/article/abs/pii/S0957417423023187

Walid Megherbi[a], Mohammed Haddad[a], Hamida Seba[a]

[a] *Univ Lyon, UCBL, CNRS, INSA Lyon, LIRIS, UMR5205, F-69622 Villeurbanne, France.*

Abstract

Dense subgraphs convey important information and insights about a graph structure. This explains why dense subgraph mining is a problem of key interest that arises in several tasks and applications such as graph visualization, graph summarization, graph clustering, and complex network analysis. It is a hard problem that has been intensively addressed by the data mining community.
In this paper, we propose a deep learning approach that enumerates all occurrences of dense subgraphs in a graph without any constraints or limitations on their size. More precisely, we enrich exiting structural node embedding with extra information, computed on node neighborhoods, to effectively capture their belonging to dense subgraphs. We evaluate our approach on several datasets to attest its efficiency on two main applications: graph summarization and graph clustering.

1. Introduction

Dense subgraph search is a fundamental problem for graph mining and network analysis [1, 2]. It arises in many applications related to complex network analysis such as community detection in social networks, large graph summarization, network visualization, biological network analysis, etc. Given a graph G with n nodes and m edges, the problem of discovering dense subgraphs in G can be defined as the problem of finding the subgraphs of G that have a remarkable density independently from, or relatively to, the other subgraphs of G. Typical dense subgraphs are cliques and quasi-cliques but the problem has several variants according to the definition of the density of a subgraph and also according to the application in hand. Most of these problems are known to be hard in general [3] and have been subject to several studies. Most existing works focus on the problem of enumerating specific kind of dense subgraphs such as cliques, quasi-cliques or bipartites with several approaches ranging from exact algorithms [4], approximations [5], and more recently machine learning [6].
In this paper, we focus on enumerating four specific kind of dense subgraphs: cliques, quasi-cliques, bipartites, and $k-$stars.

Email addresses: walid.megherbi@univ-lyon1.fr (Walid Megherbi), mohammed.haddad@univ-lyon1.fr (Mohammed Haddad), hamida.seba@univ-lyon1.fr (Hamida Seba)

For this, we propose to learn a graph representation, i.e., an node embedding, to detect dense subgraphs. To our knowledge, it is a novel approach as no existing node embedding has been proposed for this task.

There are few methods that use deep learning to detect particular substructures in a graph such as the one described in [6]. However, this approach is designed to detect repeating small motifs and consequently does not handle the problem of detecting all dense subgraphs. Our methodology is also completely different as we will show in the following sections. To motivate our contribution and show its usefulness, we rely on an incremental methodology where we first use existing node embedding solutions and evaluate their ability to detect dense substructures. Then, we propose a new node embedding that incorporates an additional dimension that gives more features to the node vectors in order to capture the belonging of nodes to dense substructures.
To show the effectiveness of our approach, we carry-out several experiments considering various datasets. We also, evaluate our approach on two applications that rely on dense subgraph mining: graph summarization and graph clustering. Our results attest the efficiency of the proposed approach.

The remainder of the paper is organized as follows: Section 2 defines formally the problem of dense subgraph mining and gives the notation used throughout the paper. Section 3 describes related work to motivate this contribution. In Section 4 we test if existing node embedding methods are able to mine dense subgraphs. Then, in Section 5, we describe our approach and methodology. In Section 6, we present the experiments we undertook to evaluate our approach. Section 7 concludes the paper.

2. Preliminaries

A graph $G(V, E)$ is defined as a structure made up of a set of *nodes* or *vertices* V and a set of *edges* E. Each edge links two nodes of the graph, which are not necessarily distinct. When an edge exists between two vertices u and v, these vertices are said to be adjacent or neighbors. The set of all neighbors of a vertex $u \in V$ is denoted by $N(u)$. The degree of a node is the number of its neighbors, i.e., $d(u) = |N(u)|$. A directed graph is a graph with directed edges and a weighted graph is a graph with weights on the edges.
A subgraph G' is a graph contained in another graph G, i.e., the set of vertices of the subgraph G' is a subset of the set of vertices of G and the set of edges of G' is a subset of the set of edges of G. More formally, we say that a graph $G' = (V', E')$ is a subgraph of $G = (V, E)$ if $V' \subseteq V$ and $E' \subseteq E$.

A dense subgraph is a maximal subgraph which has a remarkable density. A dense subgraph of G is maximal if it is not itself a subgraph of another subgraph of G that has the same density property. Several definitions of density are considered in the literature [2]. The density of a subgraph can be defined based either on the degree of the nodes of the subgraph, or the degree of the whole subgraph. In the latter, the most common definition for an unweighted graph is given by $\frac{|E'|}{|V'|}$, where $|E'|$ is the number of edges of the subgraph G' and $|V'|$ is its number of nodes.

The problem of searching or enumerating dense subgraphs, is a hot research topic. It fact, dense subgraphs give insights into the structure of a graph and help analyze complex networks.

The main dense subgraphs for unweighted graphs are cliques and quasi-cliques, $k-$stars and bipartite graphs defined as follows:

- A clique is a complete graph, i.e., a graph in which any pair of disjoint nodes (v_i,v_j) is connected by an edge e_{ij}.
- A quasi-clique generalizes the notion of clique. Given a constant $\gamma \in [0, 1]$, a graph $G = (V, E)$ is a γ-quasi-clique, if for all $v_i \in G$, $d(v_i) \geq \gamma(|V| - 1)$. For example, if $\gamma = 0.5$ then every node of the quasi-clique is connected to at least half of all other nodes in the quasi-clique.
- A $k-$star is a structure composed of a central node c and a set of k neighboring nodes $\{v_1, v_2, ..., v_k\}$ called spokes with the constraint that no edges exist between the spokes. These structures become interesting, i.e., storing the list of nodes takes less spece than storing the edges of the star, when $k >= 3$.
- A graph is said to be *bipartite* if its set of vertices can be divided into two disjoint subsets V_1 and V_2 such that each edge has one end in the set V_1 and the other in the set V_2. When all the vertices of V_1 are connected to all vertices of V_2, it is called a complete bipartite graph.

In this paper, we propose to deal with dense subgraph enumeration using deep learning, i.e., we learn a latent node representation that is capable of predicting if the node belongs to a particular type of dense subgraphs or not. Our goal is to significantly reduce the time complexity of this task. In fact, neural networks are known for their ability to make temporal complexity constant once trained which is interesting for dense subgraph search because the algorithms can be computationally intensive.
To do so, we follow an incremental methodology by first surveying existing latent node representations and their suitability for dense subgraph mining. Then, we introduce a new node representation enriched with information capturing the structural properties that allow to know if a node belongs to a dense subgraph or not.

So, we address this problem within two steps:

- In the first step, we deal with a simple classification problem that aims to distinguish the nodes that belong to a particular kind of dense subgraphs from those that do not. We achieve this classification for each desired type of dense subgraphs, i.e., clique, $k-$star, ..., etc.

  Problem: Given a graph $G = (V, E)$, how to distinguish the subset of nodes $S \subseteq V$ belonging to a particular type t of dense subgraphs from the subset of nodes which do not belong to this particular kind of dense subgraphs.

  Thus, the problem of, whether a node belongs to a particular kind of dense subgraphs or not, is defined as a multi-class classification as follows:

$$\forall v_i \in V \ \text{class}(v_i) = \begin{cases} 1, & \text{if } v_i \text{ belongs to a dense} \\ & \text{subgraph of type } t \\ 0, & \text{otherwise} \end{cases}$$

  This step is the main contribution of this paper.

- In the second step, we have a list of subsets of nodes S_1, S_2, $\cdots$, S_k where each subset S_t $1 \leq t \leq k$ belongs to a distinct type t of dense subgraphs. The problem is then to construct the dense subgraphs of type t formed by the nodes in each S_t. This step is easy to achieve in

practice even if the corresponding problem is hard because we know that all the nodes, in a given set, belong to a given kind of subgraphs. For $k-$stars, high degree nodes are considered as centers and low degree nodes are considered as spokes. For cliques and complete bipartites, we rely on the practical efficiency of Bron and Kerbosch algorithm [4] guided by the degree of the nodes. For quasi-cliques, the heuristic of Sanei-Mehri *et al.* [5] is fast in this particular case.

3. Related work

Dense subgraph mining (DSGM) is a fundamental task encountered in many real-word applications related to graph and network analysis. This motivated several research work especially in the graph mining and graph algorithms communities. A detailed survey of existing approaches that address this problem and its multiple variants can be found in [2]. The problem has been studied in various aspects (enumeration, top-k, etc.), with exact algorithms as well as approximations, we review here the main variants and approaches to solve them. Several works focus on enumerating the main dense subgraphs such as cliques. Particularly, maximal cliques enumeration has been shown to be hard but efficient implementation of exact solutions have been proposed in [7] even if the worst-case complexity remains exponential $O(3^{n/3})$.

Quasi-clique enumeration is also well known to be difficult, as the associated decision problem is also NP-hard and have already been studied a lot trough several aspects: exhaustive enumeration [8], top-k enumeration [9], etc. The densest subgraph problem, that aims to find a subgraph with maximum average degree in a weighted graph, has also received a lot of interest and can be solved in polynomial time, when the weights are not negative, by solving a maximum flow problem and even in linear time with a $1/2-$approximation [10].

DSGM is also tightly related to community detection where several algorithms are proposed [11]. Here a dense subgraph has also a constraint on the density of its links with the nodes that are outside the subgraph. Among existing solutions, we can cite Louvain [12], which detects communities by optimizing modularity, defined as the fraction of edges of a graph G that are within the same community minus the expected value of the fraction with randomly placed edges [13]. Louvain puts in the same community two neighbors if this produces the maximum modularity. Metis [14] is also a community detection approach that first coarsens the input graph by iteratively grouping nodes into supernodes in a way that preserves edge cuts. The main drawback of this algorithm is that it needs the user to specify the number of dense components the graph should be broken into.

Several works also consider dense subgraphs as seeds for clustering methods [15]. To the best of our knowledge, there are no approaches based on learning for dense subgraph mining.

4. Are Existing Node Embedding Able to Mine Dense Subgraphs?

In this section, we focus on the first step of our framework which is a classification task, that given a type t of dense subgraphs, will classify the nodes of the input graph as belonging or not to a dense subgraph of this type. To do so, we propose to train a neuronal network to achieve this classification. For this, we first need a vector representation for each node that contains sufficient information about the node. Such node representation is called a node embedding. A good embedding should capture as much relevant information as possible about the graph topology, node-to-node relationship, subgraphs, *etc.* Some node embedding methods focus solely on capturing structural properties of the nodes but others include also attributes. As belonging to a structural

dense subgraph, such as a clique or a bipartite, is a pure structural property of nodes, we rely on three main node embedding methods known to capture this kind of properties hoping that the information contained in the vectors computed by them is sufficient to recognize the nodes belonging to a dense subgraph. These methods are:

- DeepWalk [16]: This is one of the first approaches proposed to compute node embedding. From each node, random walks are launched. Every walk is considered as a short sentence in a special language and a node is a word is this language. Then, Deepwalk applies the skip-gram model [17] to generates embedding for nodes that maximize their co-occurrence in the same walks by analogy to words in sentences.

- Node2vec [18]: This approach extends DeepWalk by adding a breadth component into the walks. In fact, a naive random walk explores the graph in depth first. Node2vec introduces two hyperparameters p and q respectively defined as weights in the probability for going either depth first or breadth first in the next step of the walk.

- Line [19]: Contrary to the first two embeddings, Line does not rely on random walks. It rather defines a proximity relationship between nodes. The first-order proximity represents the local proximity and can be described as the weight of the edge connecting two nodes. The second-order proximity represents how similar are two nodes based on the proportion of common neighbors. Line preserves both first and second order proximities and scales well with large networks.

To evaluate the ability of these embedding to detect dense subgraphs, we first construct a dataset to train a neural network on this task. So, we generate a set of synthetic graphs containing specific dense subgraphs of various sizes. To generate a graph containing dense subgraphs, we first generate a random graph $G = (V, E)$ using Erds-Rnyi graph model. Then, we construct a number of dense subgraphs of various sizes. Finally we add the dense subgraphs to G by removing as many vertices from G as the number of vertices in all the generated dense subgraphs, together with their edges, and replacing them by the vertices and edges of the dense subgraphs. These vertices are then linked to the rest of the graph by adding random edges.

Once the graphs are generated, we divide them into training, validation, and test subsets.

We construct node embedding for the three subsets using the embedding methods described in the beginning of the section. Then, we train a neural network to recognize each type of dense subgraph separately.

We used a shallow neural network with two hidden layers and the sigmoid activation function. The imputs are the embedding of the nodes and the output is a probability used to make predictions about the class, i.e. the type of dense structure, of the input node.

To train this classifier, we use the mean squared error (MSE) loss function with stochastic gradient descent to adjust the parameters of the model. However, when testing these neural networks, we found that they have not learn to properly classify nodes and assign them the class to which they belong, as shown in Fig. 1, with a test data accuracy decreasing while the training data accuracy increases. We recall that at this point, we try to recognize one type of dense subgraph par neural network. So, for each node, there are two classes: 1 if the node belongs to the dense subgraph and 0 otherwise. To overcome this negative results, we first tried to avoid any ovefitting problem by re-balancing the data so that there are as many nodes belonging to class 1 as to class 0, but the problem persisted. This led to the conclusion that the embedding vectors given as input to the

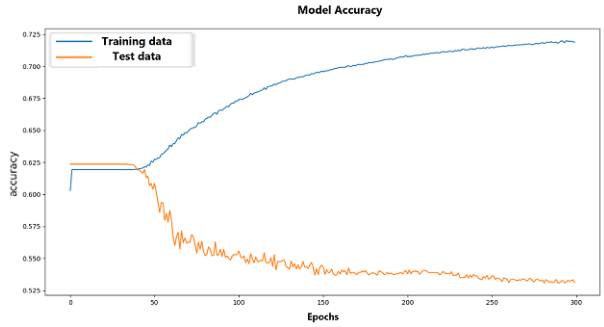

Figure 1: Performance of the neural network with existing embedding methods.

neural network do not contain enough information to recognize whether a node belongs to a dense subgraph or not.

5. A New Embedding for Dense Subgraph Mining

In this section, we address the problem of how to capture or characterize the belonging of a node to a dense subgraph. It is clear from the previous section that existing node embedding do not contain such information. We argue that this is mainly due to the fact that many graph embedding approaches tend to learn more easily high-quality embedding vectors for high-degree nodes while not ensuring good quality embedding vectors for low-degree nodes. In fact, such embedding vectors are often sub-optimal when confronted to limited structural connectivity [20]. To address this issue, We propose to collect this information independently from the embedding and then concatenate it to the embedding as an extra dimension. For this, we propose to compute for each node a value that reflects its connectivity compared to the rest of the graph, i.e., how the node reflects the density of the structure to which it belongs. So, we define a node property, we call *density revealing degree* (dd), aiming at giving more insight and enriching the obtained embedding vectors with information about the relative structural connectivity in the graph.

Definition 1. *Let $G = (V, E)$ be a graph. The density revealing degree of a node $u \in V$ is given by:*

$$dd(u) = \frac{|V|\, d(u)}{\sum_{v \in V} d(v)} \tag{1}$$

where $d(u)$ is the degree of u in G.

Let d^* be the average degree of graph G. One may observe that :

$$dd(u) \; is \begin{cases} < 1 & d(u) < d^* \\ \approx 1 & d(u) \approx d^* \\ > 1 & d(u) > d^* \end{cases}$$

As $dd(u)$ characterizes a node with a global view on how the graph is connected, it will help with the additional feature contained in the embedding to determine if a node belongs to a dense

subgraph. Our main claim is that by enriching the existing node embedding, with the *dd* values we introduced, we will allow the neural network to learn to distinguish nodes belonging to dense subgraphs from nodes that do not belong to them. To verify our claim, we first evaluate by experiment the ability of *dd* values to recognize nodes that belong to dense subgraphs. Then, in the next section, we evaluate this approach on other applications related to dense subgraph mining. To evaluate the ability of *dd* values to capture enough information on the belonging of each node to a dense subgraph, we trained a neural network on this task using four types of node embedding:

- A node embedding consisting only of the *dd* value of the node.
- A node embedding consisting of two parts: an embedding obtained by an existing method (such as node2vec, or Line), and the *dd* value of the node.
- An embedding obtained by an existing method, and augmented with the degree of the node.
- An embedding obtained by an existing method, and augmented with the average degree in the graph.

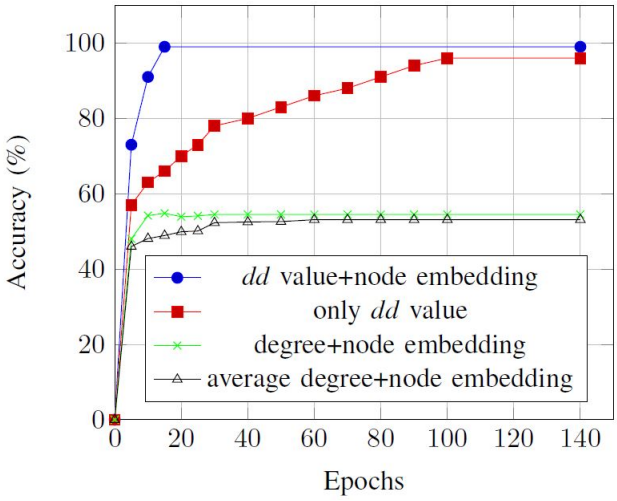

Figure 2: Performance of *dd* values with and without node embedding.

The results of our experiments depicted in Fig. 2 show clearly that the *dd* values are essential to classify the nodes according to their membership to specific dense subgraphs. Using the degree or the average degree, instead, is not really helpful. In fact, the neural network achieved 96% accuracy after 100 epochs with *dd* values as embedding, and 99% accuracy in only 15 epochs with embedding augmented with *dd* values. These results prove that although the existing embeddings alone do not contain enough information to enable a neural network to recognize the membership of nodes to dense subgraphs, they nevertheless contain important information that improve the accuracy as well as the speed of convergence of the neural network when using the proposed *dd* values. The figure also shows that the neural network does not learn to classify the nodes in the two cases where the inputs are embedding augmented with degrees and embedding augmented with the average degree in the graph, giving an accuracy close to 53%.

6. Evaluation

We show the effectiveness of our dense subgraph mining approach, called DeepDense, on two main applications: graph summarization and clustering. For this we use synthetic graphs as well as several real world graphs summarized in Table 1.

Table 1: Datasets.

Graphs	#nodes	#edges
rt-higgs	425 008	732 827
Dc3	116 835	488 953
Enron	79 870	288 364
As-oregon	10 900	31 180
rt-lolgop	9 765	10 075
rt-http	8 917	10 314
ego-facebook	2 888	3 000
football	115	613
les miserables	77	254
Dolphins	61	159
karate	34	77

6.1. Evaluation on graph Summarization

For several graph summarization and compression approaches, finding dense subgraphs is a key step in building a small and compact representation of a graph. A dense subgraph is generally easy to compress. For example, to store a clique, we only need to store the set of its vertices. A quasi-clique can also be represented by the set of its vertices augmented by a list of correcting edges, i.e., the set of edges that we had to add to the quasi-clique to obtain a clique, and that do not exist in the input graph. In this approach, a graph can be represented by its set of dense subgraphs and the set of all correcting edges that have been added to store dense subgraphs as a sets of vertices. These correcting edges are called errors. The cost of compressing the graph is computed by the cost of representing the considered dense subgraphs plus the cost of the errors. From an adjacency matrix point of view, the graph is represented by a matrix S of perfect dense subgraphs (cliques, complete bipartites, $k-$stars) and an error matrix E, i.e., $G = S + E$. The challenge is then how to select the best dense subgraphs that minimize the size of this representation. The exiting approaches differ by the type of dense subgraphs they consider and also by how they find these dense subgraphs [21], [22],[23] but almost all of them use Minimum Description Length (MDL) [24] to select the best dense subgraphs to consider. To choose this best subgraphs, the methods compute the compression cost obtained if a given structure is selected in S. If by considering the structure, no gain is observed in the final compressed graph representation, this structure is not retained.

We propose in the following, to use DeepDense to generate the dense structures within this summarization framework. We compare the obtained results with three other techniques used for this purpose in [21], namely: Louvain [12], Metis [14], described in Section 3, and Slashburn [25], a node reordering algorithm originally developed for graph compression. Slashburn finds dense components in a graph by first removing high centrality nodes from the graph, and then rearranging the nodes such that high degree nodes are assigned the lowest IDs in the adjacency matrix and nodes in disconnected components get the highest IDs. These reordering puts together the nodes belonging to the same dense subgraphs. However,

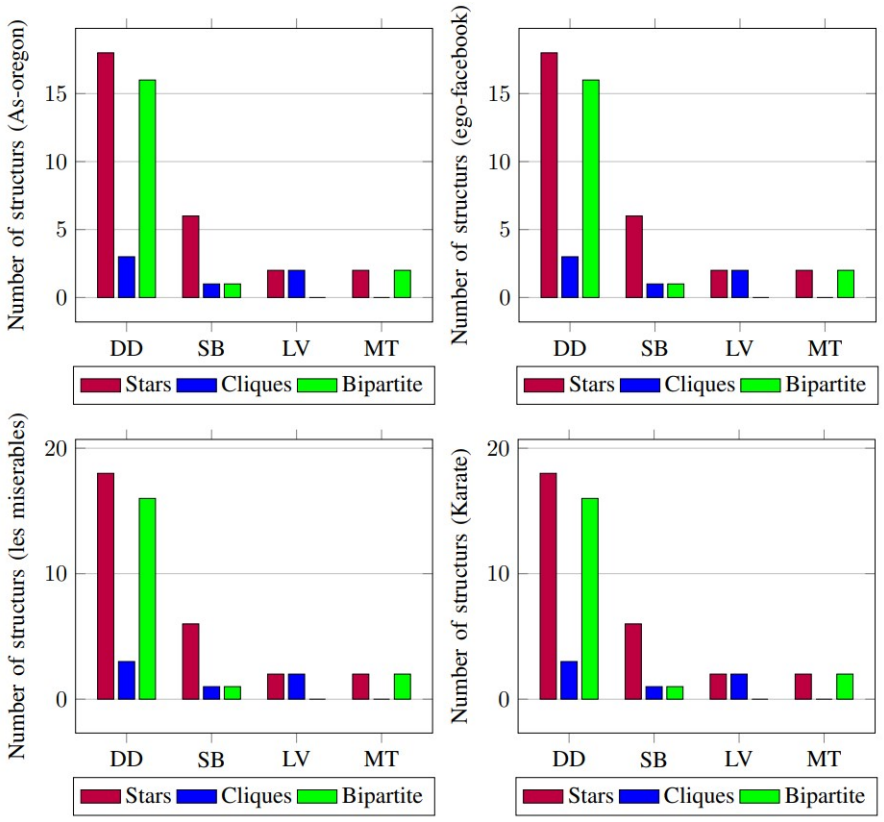

Figure 3: Number of patterns detected by DD: DeepDense, SB: SlashBurn, LV: Louvain, MT: Metis.

these algorithms return a set of structures without identifying the type of each structure: a quasi-clique, a $k-$star, a bipartite, etc. Consequently, another step is needed and consists to determine first what is the best type to use for the considered dense subgraph, i.e., is it interesting to store it as a clique, a complete bipartite or a $k-$star by testing all possible cases with MDL. Given a structure s, the algorithm calculates its cost if it is considered as a clique, then its cost as a $k-$star, and so on. Then, it stores the structure with the dense subgraph type that returns the best cost. The advantage of DeepDense is that each discovered dense subgraph is labeled with a unique type.

We consider three main metrics in our comparison: precision, compression rate and runtime.

- **Precision**: this metric measures how well these algorithms identify the dense subgraphs of an input graph. For this, we computed the number of structures of each kind found by each algorithm on four datasets: As-oregon, ego-facebook, Karate and Les miserables. Fig. 3 depicts the obtained results. It clearly shows that DeepDense manages to detect many more structures than any other method, regardless of the type of the dense subgraphs. Also, the subgraphs found by DeepDense are labeled with the correct type (clique, $k-$star, etc.) and do not require additional processing to determine how to store them in the compression step. This is not the case for the other methods.

Table 2: Compression rate.

Graph	Size (bits)	Compression rate[1]				
		SB	DD	DD*	LV	MT
rt-higgs	13 601 966	93%	**70%**	73%	100%	100%
Dc3	7 437 752	80%	**63%**	65%	100%	100%
Enron	4 292 728	74%	**72%**	77%	100%	100%
As-oregon	546 127	69%	**67%**	**67%**	95%	96%
rt-lolgop	137 557	37%	18%	**15%**	41%	100%
rt-http	137 767	70%	**66%**	70%	100%	100%
ego-facebook	35 472	38%	**35%**	**35%**	44%	92%

(1)The compression rate is the ratio of the size (in bits) of the summary, i.e., $S + E$, on the size of the original graph. SB: SlashBurn; DD: DeepDense ; DD*: DeepDense (using only $k-$stars); LV: LOUVAIN; MT: METIS

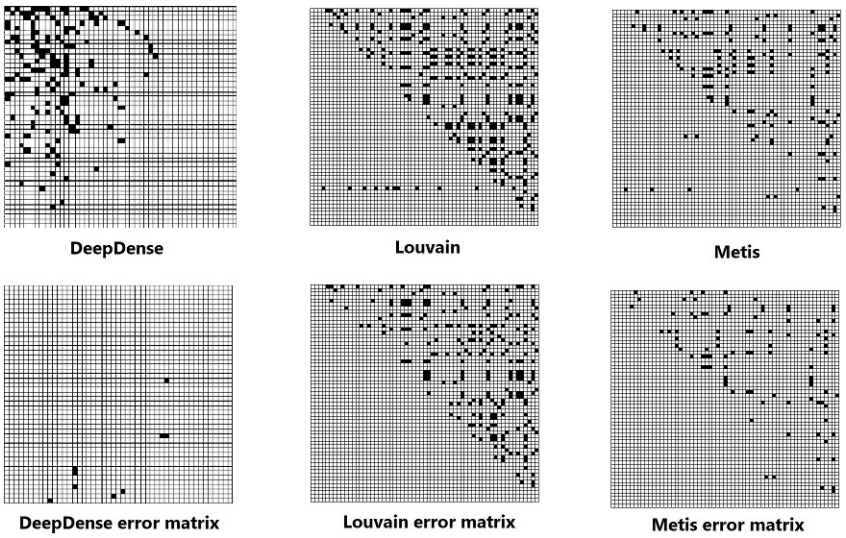

Figure 4: Structure and error matrices, of the Dolphins graph, obtained by DeepDense, Louvain and Metis.

- **Compression Rate**: the compression rate measures how well the considered dense subgraphs compress the graph. It is the ratio between the size in bits of $S + E$ and the size of the original graph G. Table 2 presents the compression rate obtained with the four algorithms on the considered datasets. We can clearly see that DeepDense outperforms all the other methods. This can be explained by the precision of DeepDense. In fact DeepDense produce more structures and by the way offer more choices to pick the best structures that minimize $S + E$. Fig. 4 shows this directly on the adjacency matrix where we can see that both the matrix S of structures and the error matrix E of DeepDense are the most optimized. It is also worth noting that DeepDense has the advantage of allowing us to choose the kind of dense graphs we want to consider for a compression as all the structures it finds are labeled. For example, we remarked that we can obtain better compression rate while considering only $k-$stars or only quasi-cliques for certain graphs of our dataset. We denoted this by DD* in Table 2 that shows these results.

- Runtime: Table 3 presents the runtime achieved by each algorithm to find the dense subgraphs for all the datasets. We can see that DeepDense performs well compared to the other methods even if it is less performant than Metis on large graphs, the latter having rather low complexity of $O(|E|K)$ where K is the number of clusters/partitions that needs to be given as a parameter. However, this is not a scalability problem for our method because the neural network detects the patterns in constant time. However, when the graph is larger than the size of the Neural network, we treat it as batches which allow us to deal with even larger graphs easily.

Table 3: Runtime (seconds).

Graphs	SB	DD	LV	MT
rt-higgs	1387.14s	63.387s	141.62s	**50.184s**
Dc3	431.58s	114.83s	119.43s	**39.947s**
Enron	277.937s	99.36s	108.118s	**27.677s**
As-oregon	36.971s	**4.893s**	39.694s	14.718s
rt-lolgop	13.635s	**3.05s**	47.765s	10.884s
rt-http	14.80s	**3.018s**	41.252s	19.721s
ego-facebook	4.972s	**2.827s**	28.092s	8.680s

SB: SlashBurn; LV: LOUVAIN; MT: METIS; DD: DeepDense

6.2. Evaluation on Graph Clustering and Community Detection

A cluster or community in a graph is a connected dense subgraph. Cluster detection approaches can be classified into several categories such as cut-based methods, stochastic block modeling and random walk-based methods like Walktrap [26] and SynWalk [27].

Consider a clustering of the nodes of a graph into a set of k non-empty sets C_i, $1 \leq i \leq k$. If the clusters are disjoint, they are called non-overlapping clusters. A clustering of $G = (V, E)$ infers a mapping function $m : V \rightarrow \{1, \ldots, k\}$, where each node of the graph is associated to the index of its cluster, i.e., for a node x, if $x \in C_i$ then $m(x) = i$.
In the following, we compare DeepDense with the state-of-the-art community detection methods: Louvain [12], Walktrap [26], SynWalk [27] and Infomap [28].

In order to evaluate the results of these clustering methods, we use datasets with ground-truth on existing communities as done in the state of the art comparisons [29, 30, 31]. We first begin with synthetic graphs containing this ground truth, namely the LFR benchmark [32] which is one of the most used dataset in this domain. The LFR benchmark allows to generate graphs according to the mixing parameter μ of each node [32] which is the ratio between its number of edges with nodes outside the cluster and its degree. For an undirected and unweighted graph and a candidate cluster C, the mixing parameter of a node x is defined as:

$$\mu(x) = \frac{d(x)^{out}}{d(x)} \tag{2}$$

where $d(x)^{out}$ is the number of links between x and nodes outside its cluster $C_{m(x)}$ and $d(x)$ is the degree of x. A cluster C_i is said to be a strong cluster [33] if for all its nodes $\mu(x) < 0.5$.
The metric often used to measure the quality of the clusters obtained in the LFR benchmarks is the Adjusted Mutual Information (AMI) [34]. This metric evaluates and compares the clusters obtained

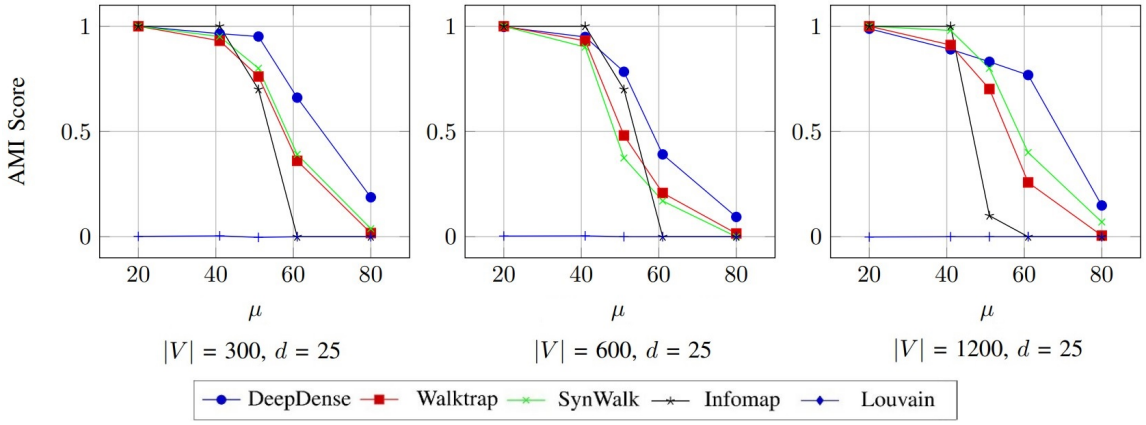

Figure 5: Performance on clustering on the LFR dataset.

Table 4: Modularity values and AMI score for real graphs.

Graphs	Modularity					AMI score				
	DD	**SW**	**IM**	**WT**	**LV**	**DD**	**SW**	**IM**	**WT**	**LV**
Football	0.55	0.55	0.57	**0.61**	**0.61**	**0.045**	0.027	0.022	0.019	0.007
Dolphins	0.46	0.47	0.42	**0.51**	**0.51**	**0.042**	0.019	0.016	0.016	0.009
Karate	0.30	0.29	0.27	0.40	**0.41**	**0.391**	0.228	0.243	0.182	0.082

DD: DeepDense, SW: Synwalk, IM: Infomap, WT: Walktrap, LV: Louvain

by tested algorithms with ground-truth. An AMI value close to 1 indicates a strong similarity with the ground-truth while a value around 0 reflects low similarity.

Fig. 5 shows the variation of the AMI score in function of the mixing parameter μ for the five algorithms. Overall, DeepDense outperforms all the other methods in terms of AMI on sufficiently dense networks or networks with mixing parameters $\mu > 0.6$. Also, compared to the other methods, the results show that DeepDense never reaches a zero AMI, even for $\mu = 0.8$, its values are between 0.08 and 0.2.

We also evaluate the quality of the obtained clustering on real graphs. For this, we use a dedicated dataset available in [35] which contains ground truth on the available communities. For this, we use two metrics: the AMI score and the modularity [13]. Recall that modularity represents the fraction of edges of a graph G that are within the same cluster minus the expected value of the fraction with randomly placed edges. Table 4 presents our results. It shows clearly that for all graphs, Deepdense achieves the best AMI score. For modularity, we can see that Louvain which is itself based on modularity optimization gives the best results followed by Walktrap. DeepDense, Infomap and SynWalk have similar results which are inferior to Louvain or Walktrap. However, it has been shown that modularity optimization may fail to identify *good modules* or communities in real world graphs [36].

7. Conclusion

Dense subgraph mining is a key problem that arises in several tasks and applications such as graph clustering, graph summarization, fraud detection, and complex network analysis.
In this paper, we proposed DeepDense, a deep learning approach that enumerates the dense subgraphs in a graph without any constraint or limitation on their size. Our approach consists in enriching existing structural node embedding with an additional computed value that captures the belonging of nodes to dense components in the graph, leading to a more meaningful node embedding. Our experiments on several real and synthetic datasets show the effectiveness of DeepDense on two main applications, namely graph summarization and graph clustering. We have confidence that our approach works as well on directed graphs as on undirected graphs but more experiments are needed to attest this. Furthermore, the *dd* value formula can be easily adapted to deal with weighted graphs.
Beyond dense subgraph mining, this work shows that learned representations can be mixed up with engineered features for better accuracy and convergence.

Acknowledgment

The source code and data are available here https://gitlab.liris.cnrs.fr/hseba/deepdense/

This work is supported by the French National Research Agency (ANR) under grant ANR-20-CE39-0008.

[1] V. E. Lee, N. Ruan, R. Jin, C. Aggarwal, A Survey of Algorithms for Dense Subgraph Discovery, Springer US, Boston, MA, 2010, pp. 303–336. doi:10.1007/978-1-4419-6045-0_10.

[2] A. Gionis, C. E. Tsourakakis, Dense subgraph discovery: Kdd 2015 tutorial, in: 21th ACM SIGKDD, 2015, p. 23132314. doi:10.1145/2783258.2789987.

[3] J. Hu, R. Cheng, K. C.-C. Chang, A. Sankar, Y. Fang, B. Y. Lam, Discovering maximal motif cliques in large heterogeneous information networks, in: 2019 IEEE 35th International Conference on Data Engineering (ICDE), IEEE, 2019, pp. 746–757.

[4] C. Bron, J. Kerbosch, Algorithm 457: finding all cliques of an undirected graph, Communications of the ACM 16 (9) (1973) 575–577. doi:10.1145/362342.362367.

[5] S.-V. Sanei-Mehri, A. Das, H. Hashemi, S. Tirthapura, Mining largest maximal quasi-cliques, ACM Trans. Knowl. Discov. Data 15 (5). doi:10.1145/3446637.

[6] R. Ying, A. Z. Wang, J. You, J. Leskovec, Frequent subgraph mining by walking in order embedding space, in: GRL+, ICML workshops, 2020.
URL https://snap.stanford.edu/frequent-subgraph-mining/

[7] Y. Jin, B. Xiong, K. He, Y. Zhou, Y. Zhou, On fast enumeration of maximal cliques in large graphs, Expert Systems with Applications 187 (2022) 115915.

[8] G. Liu, L. Wong, Effective pruning techniques for mining quasi-cliques, in: W. Daelemans, B. Goethals, K. Morik (Eds.), Machine Learning and Knowledge Discovery in Databases, 2008, pp. 33–49.

[9] S.-V. Sanei-Mehri, A. Das, S. Tirthapura, Enumerating top-k quasi-cliques, in: IEEE International Conference on Big Data, 2018, pp. 1107–1112. doi:10.1109/BigData.2018.8622352.

[10] M. Charikar, Greedy approximation algorithms for finding dense components in a graph, in: Approximation Algorithms for Combinatorial Optimization, Springer, 2000, pp. 84–95.

[11] J. Leskovec, K. J. Lang, M. Mahoney, Empirical comparison of algorithms for network community detection, in: 19th International Conference on World Wide Web, WWW '10, ACM, New York, NY, USA, 2010, p. 631640. doi:10.1145/1772690.1772755.

[12] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, E. Lefebvre, Fast unfolding of communities in large networks, Journal of statistical mechanics: theory and experiment 2008 (10) (2008) P10008.

[13] M. E. Newman, Modularity and community structure in networks, Proceedings of the national academy of sciences 103 (23) (2006) 8577–8582.

[14] G. Karypis, R. Aggarwal, V. Kumar, S. Shekhar, Multilevel hypergraph partitioning: Applications in vlsi domain, IEEE Transactions on Very Large Scale Integration (VLSI) Systems 7 (1) (1999) 69–79.

[15] A. Keszler, T. Szirnyi, Z. Tuza, Dense subgraph mining with a mixed graph model, Pattern Recognition Letters 34 (11) (2013) 1252–1262. doi:https://doi.org/10.1016/j.patrec.2013.03.035.

[16] B. Perozzi, R. Al-Rfou, S. Skiena, Deepwalk: Online learning of social representations, in: ACM SIGKDD international conference on Knowledge discovery and data mining, 2014, pp. 701–710.

[17] T. Mikolov, K. Chen, G. Corrado, J. Dean, Efficient estimation of word representations in vector space, arXiv preprint arXiv:1301.3781.

[18] A. Grover, J. Leskovec, node2vec: Scalable feature learning for networks, in: ACM SIGKDD international conference on Knowledge discovery and data mining, 2016, pp. 855–864.

[19] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, Q. Mei, Line: Large-scale information network embedding, in: 24th international conference on world wide web, 2015, pp. 1067–1077.

[20] Z. Liu, W. Zhang, Y. Fang, X. Zhang, S. C. Hoi, Towards locality-aware meta-learning of tail node embeddings on networks, in: ACM International Conference on Information & Knowledge Management, 2020, pp. 975–984.

[21] Y. Liu, T. Safavi, N. Shah, D. Koutra, Reducing large graphs to small supergraphs: a unified approach, Social Network Analysis and Mining 8 (1) (2018) 1–18.

[22] S. Navlakha, R. Rastogi, N. Shrivastava, Graph summarization with bounded error, in: ACM SIGMOD international conference on Management of data, 2008, pp. 419–432.

[23] D. Chakrabarti, Autopart: Parameter-free graph partitioning and outlier detection, in: European conference on principles of data mining and knowledge discovery, Springer, 2004, pp. 112–124.

[24] J. Rissanen, Modeling by shortest data description, Automatica 14 (5) (1978) 465–471.

[25] U. Kang, C. Faloutsos, Beyond'caveman communities': Hubs and spokes for graph compression and mining, in: 2011 IEEE 11th international conference on data mining, IEEE, 2011, pp. 300–309.

[26] P. Pons, M. Latapy, Computing communities in large networks using random walks, in: International symposium on computer and information sciences, Springer, 2005, pp. 284–293.

[27] C. Toth, D. Helic, B. C. Geiger, Synwalk: community detection via random walk modelling, Data Mining and Knowledge Discovery (2022) 1–42.

[28] M. Rosvall, D. Axelsson, C. T. Bergstrom, The map equation, The European Physical Journal Special Topics 178 (1) (2009) 13–23.

[29] S. Fortunato, D. Hric, Community detection in networks: A user guide, Physics reports 659 (2016) 1–44.

[30] G. K. Orman, V. Labatut, A comparison of community detection algorithms on artificial networks, in: International conference on discovery science, Springer, 2009, pp. 242–256.

[31] Z. Yang, R. Algesheimer, C. J. Tessone, A comparative analysis of community detection algorithms on artificial networks, Scientific reports 6 (1) (2016) 1–18.

[32] A. Lancichinetti, S. Fortunato, Benchmarks for testing community detection algorithms on directed and weighted graphs with overlapping communities, Physical Review E 80 (1) (2009) 016118.

[33] F. Radicchi, C. Castellano, F. Cecconi, V. Loreto, D. Parisi, Defining and identifying communities in networks, National academy of sciences 101 (9) (2004) 2658–2663.

[34] N. X. Vinh, J. Epps, J. Bailey, Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance, Machine Learning Research 11 (2010) 2837–2854.

[35] `https://github.com/altsoph/community_loglike.git`.

[36] S. Fortunato, M. Barthélemy, Resolution limit in community detection, Proceedings of the National Academy of Sciences USA 104 (1) (2007) 36–41, 17190818[pmid]. `doi:10.1073/pnas.0605965104`.