A Label-based Edge Partitioning for Multi-Layer Graphs

Camelia Constantin, Cédric Du Mouza, Yifan Li

A Label-based Edge Partitioning for Multi-Layer Graphs

Camelia Constantin
LIP6, Sorbonne Université, France
camelia.constantin@lip6.fr

Cedric du Mouza
CEDRIC, CNAM, France
dumouza@cnam.fr

Yifan Li
LIP6, Sorbonne Université, France
iamyifanli@gmail.com

Abstract

Social network systems rely on very large underlying graphs. Consequently, to achieve scalability, most data analytics and data mining algorithms are distributed and graphs are partitioned over a set of servers. In most real-world graphs, the edges and/or vertices have different semantics and queries largely consider this semantics. But while several works focus on efficient graph computations on these "multi-semantic" graphs, few ones are dedicated to their partitioning. In this work, we propose a novel approach to achieve edge partitioning for multi-layer graphs, which considers both structural and edge-types (labels) localities. Our experiments on real life datasets with benchmark graph applications confirm that the execution time and the inter-partition communication can be significantly reduced with our approach.

1. Introduction

Graph theory has been intensively studied in the past few centuries for solving real world problems in physics, biology, sociology and information systems. In particular, with the development of computer science, a number of data structures and algorithms have been developed to facilitate this calculation for a wide range of graph-related tasks from local telephone network design to components placement of an electronic circuit. However, the data stores created in last decades are becoming unexpectedly large especially in social media. For instance, Facebook has generated a massive social network with more than one billion users and hundreds of times more connections.

A solution to achieve scalability for analyzing and mining data is to partition the graph. In modern distributed computing systems, two main partition approaches for graph data are considered to scale to very large graphs: *vertex partition* and *edge partition*. Recently, [1] demonstrated that the vertex-cut method outperforms edge-cut for distributed computation over real world graphs, which are likely to have highly skew degree distribution. With respect to the workload balance status in a distributed computing context, it can improve the overall performance significantly.

Real life graph data often consists of different types of edges and/or vertices. Many graph computing tasks depend partly on this kind of information, like the recommendation calculation for given topics in social network, the community detection and analysis and the matching of regular expressions in labeled graph. However existing graph partitioning methods do not consider the heterogeneity structure of these graphs when allocating edges to the different servers. Naive approaches where the graph is split in several graphs, one per layer, lead either to vertices/edges redundancy or to unbalanced vertices/edges allocation. Additionally they require computations on different graphs so more communication and execution times. We believe that considering the edge/vertex heterogeneity when partitioning is essential to provide efficient computations by drastically reducing the communication between servers.

In this paper we consider a *Multi-layer Graph* model which assumes different types of connections, such as the Like/Follow/Friend interactions in social networks and/or the multi-labels on edge in user-interest graphs. Figure 1, represents an example of a 3-layer graph for a social network where we assume 3 topics: *sport*, *technology* and *news*. On the left-hand side, we have its representation as a single graph whose edges are multi-labeled. On the right-hand side we have its decomposition in three layers, *i.e.*, three graphs, each one corresponding to a distinct label. Consequently to determinate recommendations for Adam on the topic *news*, we can run the recommendation algorithm on the according layer and not the complete multi-labeled graph. So the main contribution of our paper is a new edge partition method which enhances distributed graph computation by considering simultaneously the graph's topology and its edge-semantics heterogeneity.

The paper is organized as follows: after an

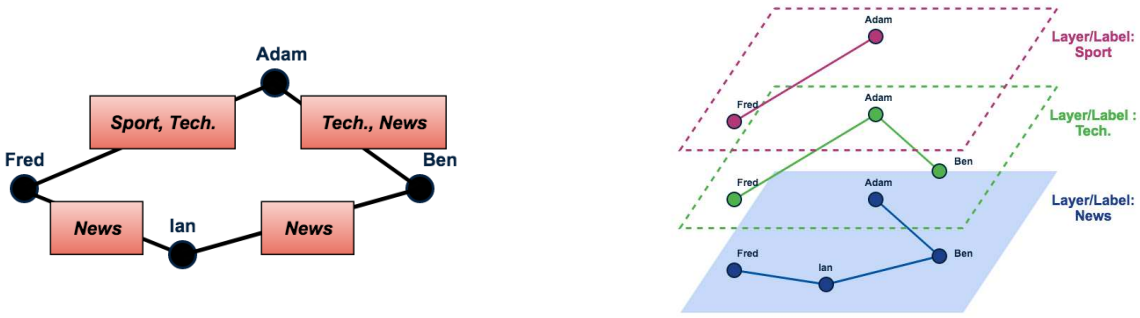


Figure 1. Multi-Layer Graph representations

introduction about multi-layer graph edge partitioning issue in Section 1, we present related work in Section 2. Then we introduce our block-based approach for graph partitioning using connectivity computation and block profiles in Section 3. We discuss about the seeds selection in Section 4 and block refinement and merging in Section 5. Experiments in Section 6 validate our approach. Finally Section 7 concludes the paper and gives perspectives.

2. Related work

Numerous applications rely on graph exploitation, like online social network advertisement, public transportation optimization, or biological network analysis. In research, many efforts have been done on graph mining algorithms such as clustering, recommendation computation or pattern matching, by measuring topological properties, analyzing attributed contents,. . .. Nowadays it becomes crucial to find solutions to deal with current large graphs for maintaining those achievements from existing works. The natural way to achieve scalability is the classical strategy ”divide and rule”, and to the best of our knowledge, the initial work on this topic was Bulk Synchronous Parallel(BSP) model[2]. Later, the authors of [3] proposed METIS and its parallel implementation ParMETIS, which are a set of serial programs which allows graph partitioning based on the multilevel recursive-bisection, multilevel k-way, and multi-constraint partitioning schemes. Giraph [4] is another system developed, w.r.t the distributed graph processing, to help for easily implementing their applications over large graph in a parallel computing framework. More precisely, this work relies on Pregel model [5] by Google which converts the graph computation procedure into a sequence of iterations, namely supersteps, separated by global synchronization points. During each superstep, there is a common user-defined function executed on every vertex and the messages will be sent/received by them to perform a specific graph algorithm, the processing will terminate when the graph reaches stable state which means no vertex is active. Almost at the same time, Low et al. proposed the GraphLab [6], an asynchronous shared-memory abstraction for distributed machine learning over graph data which is able to efficiently utilize memory of multiprocessors and get fast convergence for graph algorithms. All these proposals significantly improved the graph computations but the performance tends to degrade when the graph is particularly skewed, like the ones in social networks.

In [1], the new version of GraphLab, named PowerGraph, is proposed to overcome the limits of previous solutions for computing real life graphs which follow the highly skew degree distribution (power-law graphs). They introduced a new computation model named GAS that factors the vertex program along edges, to achieve the scalability of graph computation via partitioning graph data by edge instead of vertex. The graph processing module in Spark, GraphX[7], also employs a similar idea to GAS but provides more edge partitioning strategies to program developers.

In addition to their large scale, real graphs are also characterized by their heterogeneity. In other words, the vertices and edges in real world graphs usually have different types or attributes, which can largely impact the execution of graph algorithms deployed over them. While some recent works deal with heterogeneous network partitioning [8, 9] or clustering [10, 11], much additional efforts are still required to transfer vertex partitioning to edge partitioning, and to consider the relation between locality and label distribution has not been studied well. In [12], the authors present the $\mathcal{C}^3\mathcal{R}$ framework to detect user communities based on novel regularized spectral clustering approach that is able to perform an efficient partitioning of multi-layer user relations graph. But the time complexity of the framework, which is introduced by the graph Laplacians computation is a limit for real (very large) datasets.

Our approach is, to the best of our knowledge, the first one which achieves scalable edge partitioning for efficient querying on very large real multi-layer graphs.

3. Block Construction

In a *vertex-cut* partitioning, *i.e*, *edge* partitioning, a vertex which belongs to several edges might be duplicated across partitions. In this Section, after introducing some definitions, we present how to allocate the different edges to limit the communication between a vertex and its replicated when processing a computation over a multi-graph.

Multi-layer graph.

Definition 1 (Multi-layer graph) *A multi-layer graph is a triple $G(V, E, \Lambda)$ where $E \in V \times V$ denotes a set of edges, V a set of vertices and $\Lambda : E \rightarrow 2^{\mathcal{L}} \setminus \emptyset$ is the labeling function, with $\mathcal{L}$ the domain of the edge labels.*

In other word, a multi-layer graph is a graph whose edges are labeled with one or more labels. Conceptually a multi-layer graph may be considered as a set of graphs, one for each label, which share the same vertices. We define the restriction of a multi-layer graph G to a given label l as:

Definition 2 (Multi-layer graph restriction) *Let $G(V, E, \Lambda)$ be a multi-layer graph and a label $l \in dom(\Lambda)$. The restriction of G to l, denoted G_l, is the graph $G_l(V_l, E_l)$ where*

$$V_l \subseteq V \wedge E_l \subseteq E \wedge \forall e \in E_l, l \in \Lambda(e) \\ \wedge \forall e' \in E \setminus E_l, l \notin \Lambda(e')$$

So the graph G_l is the graph, possibly not-connected, which contains all edges from G labeled with l.

Block definition.

A block corresponds to a tightly knit cluster in graph, *e.g.* a community in social network. In our approach, we consider the block as a set of edges which are "close" one to another, and these blocks become the component units of each partition in computation, but also the allocation units for workload over machines. More formally, a block is defined as follows.

Definition 3 (Block) *Consider a graph $G(V, E)$, a set $\mathcal{S} \subseteq V$ of vertices called seeds, and an allocation function $alloc : E \times \mathcal{S} \rightarrow boolean$. A block b is a couple (s, E') where $s \in \mathcal{S}$ is a node called the* block seed *and $E' \subseteq E$ is a subset of edges such as $\forall e \in E'$, $alloc(e, s) = true$ and $\forall s' \in \mathcal{S} - \{s\}, alloc(e, s) = false$.*

The $alloc$ function allows to allocate an edge to the seed which is *"close"*, considering both the topology (*locality*) and the labels (*similarity*). Consequently a block groups a set of edges close to each other in the graph and which share some common topics.

To take into consideration the locality of edges we propose a *Connectivity Computation*.

3.1. Connectivity Computation

To preserve the locality inside a block, we adapt the *Inverse P-distance* to capture the connectivity, *i.e.*, closeness, from a seed to a given vertex. The idea is that the more paths there exist between two vertices and the shorter they are, the more connected (closer) these vertices are. We assume in the following that we have a directed graph but all the definitions work for a undirected graph with the associate semantics for paths and using the degree instead of the outdegree.

We compute the connectivity score $con_v^l(i, j)$ between vertex i and j for a given label l in multi-layer graph G:

$$con_v^l(i, j) = \sum_{p \in P_{ij}^l} S(p^l) \tag{1}$$

where P_{ij}^l is the set of paths between i and j in the multi-layer graph restriction G_l (*i.e.*, the set of l-labeled paths in the multi-layer graph G), and $S(p^l)$ is the inverse distance value calculated on path p^l, (v_0, v_1, ..., v_k) as:

$$S(p^l) = (1 - \alpha)^k \cdot \prod_{i=0}^{k-1} \frac{1}{outDeg^l(v_i)} \tag{2}$$

where $\alpha \in (0, 1)$ is the decay factor used in distance calculation, and $outDeg^l(v_i)$ is the number of out-edges with label l from vertex v_i.

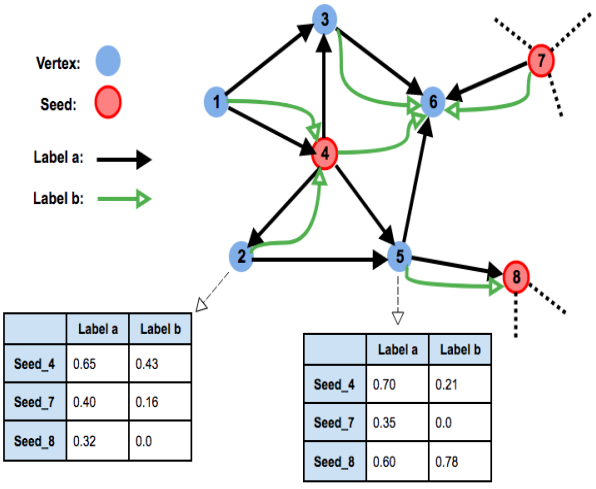
Based on our *vertex-to-vertex* connectivity score we define an *edge-to-vertex* connectivity score which captures how "close" an edge is from a given vertex.

Definition 4 (Edge Connectivity) *The connectivity score $con_e^l(k, e')$ between a vertex k and an edge $e' = (i, j)$ is:*

$$con_e^l(k, e') = \theta(con_v^l(k, i), con_v^l(k, j))$$

where θ is an aggregation function.

In the experiments we chose as the aggregate function the average function but any other aggregate function may be selected.



	Label a	Label b
Seed_4	0.65	0.43
Seed_7	0.40	0.16
Seed_8	0.32	0.0

	Label a	Label b
Seed_4	0.70	0.21
Seed_7	0.35	0.0
Seed_8	0.60	0.78

Figure 2. An example of connectivity scores calculation over multi-layer graph.

Example 1 *Figure 2 depicts a 2-layer graph with labels* a *and* b *and three seeds (vertices 4, 7 and 8). For the vertices 2 and 5 we compute the connectivity scores to the different seeds for the different labels. For instance, considering the label a, we compute on the restriction G_a the connectivity scores between the vertex 2 and the seeds 4, 7 and 8. Assume (al edges are not depicted) that we get respectively the connectivity scores $con_v^a(v_2, v_4) = 0.65$, $con_v^a(v_2, v_7) = 0.40$ and $con_v^a(v_2, v_8) = 0.32$. Similar computation for the vertex 5 on G_a gives respectively the scores 0.70, 0.35 and 0.60. Consider the aggregate $\theta = avg$, we obtain for the edge $e_{2,5}$ the edge connectivity scores $con_e^a(v_4, e_{2,5}) = 1.35/2$, $con_e^a(v_7, e_{2,5}) = 0.75/2$ and $con_e^a(v_8, e_{2,5}) = 0.92/2$. Observe that we have some vertex connectivity scores equal to zero. It means that there exists no path between the vertex and the corresponding seed.*

Since for each label, we have for each vertex (resp. edge) a score to each seed, we can adopt the following matrix representation.

Definition 5 (Connectivity Scores Matrices) *Consider the set $S = \{s_1, s_2, \ldots, s_\kappa\}$ of seeds and the set $L = \{l_1, l_2, \ldots, l_\lambda\}$ of labels. The vertex connectivity score matrix for a vertex k, denoted V_k, is a matrix with size $|S| \times |L|$ where*

$$\forall i \in [1..\kappa], \forall j \in [1..\lambda], V_k(i,j) = con_v^j(k, s_i).$$

From this definition we can deduce the definition of the edge connectivity score matrix E_ϵ for an edge $\epsilon = (k, k')$ as:

$$\forall i \in [1..\kappa], \forall j \in [1..\lambda], E_\epsilon(i,j) = \theta(V_k(i,j), V_{k'}(i,j))$$

where θ is an aggregation function.

Observe that the computation of these two matrices can be implemented by performing a *Breadth-First Search (BFS)* from each seed in S for each label in L, which can be easily deployed in *Pregel*-like model systems. This computation has a time complexity of $O(|S| \times |L| \times (r^q))$, where $|S|$ denotes the number of seeds, $|L|$ the number of labels and r the average degree of vertices, q corresponds to the depth of the BFS performed, generally a small value, *e.g.* 5 in our experiments. In the matrix E_ϵ each column corresponds to a label, and the highest score in a column points out the topologically closest seed for this edge in the corresponding graph restriction. However the candidate seed proposed for the edge allocation is likely to be different from one label to another. To determine the block to allocate the edge, we must consider all the labels in the same time. To achieve this, we first build a *block profile* for the different blocks.

Example 2 *Consider the example in Figure 2. The left table (resp. right table) corresponds to the matrix V_2 of vertex 2 (resp. V_5 of vertex 5) for the seed set $S = \{4, 7, 8\}$ and the label set $L = \{a, b\}$. According to Definition 4, with the choice of addition as aggregation function θ, we obtain the connectivity scores matrix E_ϵ from the edge $\epsilon = (2, 5)$ to every seed for each label in multi-layer graph:*

$$E_\epsilon = \begin{bmatrix} \mathbf{1.35} & 0.64 \\ 0.75 & 0.16 \\ 0.92 & \mathbf{0.78} \end{bmatrix}$$

We observe that for the label a (first column), the edge ϵ got the best score with seed S_4 while for the label b the best score is obtained with seed S_8.

3.2. Block Profiles

The block profile is a summarization of the interests of the users within a block. Different content-based profiles for communities are proposed in literature like in [13, 14]. We propose a simple block profile construction based on the number of occurrences of the different labels of the edges of the block, but more advanced strategies may be used.

So assume that for the set of edges E of a graph G we have a set $E' : \{E_1, E_2, ..., E_k\}$ of blocks, such that $\bigcup_{1 \leq i \leq k} E_i \subseteq E$ and $\forall i, \forall j, i \neq j \Rightarrow E_i \cap E_j = \emptyset$. Also assume the existence of the function $occ : 2^E \times \mathcal{L} \rightarrow N$ which returns the number of occurrences of a given label in a set of edges. We define the label score of a block as:

Definition 6 (label score of a block) *The label score for a block E and a label l is:*

$$score(E, l) = \frac{occ(E, l)}{\sum_{l_i \in L} occ(E, l_i)}$$

The block profile of a given block E consists of the set of the different label scores computed for each label $l \in L = \{l_1, l_2, \ldots, l_\lambda\}$. So we can represent the different block profiles for a set of blocks $E' : \{E_1, E_2, ..., E_k\}$ as a matrix P that we denote the *blocks profile matrix*:

$$P = (p_{ij})_{1 \leq i \leq \lambda, 1 \leq j \leq k} \; with \; p_{ij} = score(E_i, l_j)$$

Label correlation. The precedent definition assumes that the labels are independent one from another. In many applications, there exist however correlations between labels. For instance, we expect to group the edges (vertices) with labels *IT* and *Computer* together, or those with labels *Politics* and *Polls*, since they are more likely to be queried or processed concurrently in applications.

So we assume the existence of a correlation matrix of labels $C(L) \in [0, 1]^{|L| \times |L|}$, with $\forall i, \forall j, c_{ij} = c_{ji}$ is the correlation score for label l_i with l_j. Different correlation functions exist to build this matrix like Pearson, Kendall or Spearman, for instance. Observe that if the different labels are independent the matrix $C(L)$ is equals to identity matrix $I^{|L|^2}$.

Finally we adapt the definition of the block profile matrix to take into consideration the label correlation:

Definition 7 (blocks profile matrix) *Consider the set of blocks $E' : \{E_1, E_2, ..., E_k\}$ and the set of labels $L = \{l_1, l_2, \ldots, l_\lambda\}$, the blocks profile matrix P_{corr} is the matrix*

$$P_{corr} = P^T \times C(L)$$

For the sake of simplicity, we use the notation P instead of P_{corr} in the following.

3.3. Edge Allocation

Intuitively, starting a topic-based algorithm in a block with a small number of labels shared by its different edges will lead to less inter-block communication and consequently a faster execution time than a block with a large number of different labels with a uniform distribution. So our goal is to build such *topic focused* blocks, which represent basically blocks of users sharing similar interests, *i.e.*, a community. To achieve this, we allocate an edge based on topology and on the different block's profiles.

Proposition 1 (Edge allocation) *Consider an edge ϵ and blocks profile matrix P. ϵ is allocated to the seed s_i determined by:*

$$argmax_{x \in \{1, \ldots, |S|\}} \{a_x | a_x \in A = (E_\epsilon \odot P) \times 1^n\}$$

where $\odot$ denotes the entrywise product*, and 1^n is a vector with all values to* 1.

Example 3 *Assume in Figure 2 that labels a and b are independent, and the blocks matrix profile P for the blocks corresponding to the seeds 4, 7 and 8 is:*

$$\begin{bmatrix} 0.8 & 0.2 \\ 0.75 & 0.25 \\ 0.5 & 0.5 \end{bmatrix}$$

Then $A = (E_\epsilon \odot P) \times 1^n$ is:

$$A = \left(\begin{bmatrix} \mathbf{1.35} & 0.64 \\ 0.75 & 0.16 \\ 0.92 & \mathbf{0.78} \end{bmatrix} \odot \begin{bmatrix} 0.8 & 0.2 \\ 0.75 & 0.25 \\ 0.5 & 0.5 \end{bmatrix} \right) \cdot \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1.35 * 0.8 & 0.64 * 0.2 \\ 0.75 * 0.75 & 0.16 * 0.25 \\ 0.92 * 0.5 & 0.78 * 0.5 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1.208 \\ 0.6025 \\ 0.85 \end{bmatrix}$$

So from above result, edge $e_{2,5}$ should be allocated to the block associated to seed 4.

4. Seeds Selection

The choice of the seeds may largely impact the quality of the resulting partitioning since the edge allocation highly depends on the topological and thematic proximity of the seeds. While the seed selections is known as an NP-hard problem, we propose in the following a strategy with a linear complexity to select the seeds based on the topology and on the "profiles" of the seeds.

We first propose the *Variance of Edges*, $VOE(E')$, to measure how the labels from $\mathcal{L}$ are spread out over a set of edges $E' \subseteq E$. Intuitively, the more common edge labels and the fewer of them are involved, the higher VOE for E'. Formally, given a set of edges $E' \subseteq E$, and set of labels $L \in \mathcal{L}$, $VOE(E')$ is defined as the mean of the squared deviation from the mean of $label(E')$ for each label in L:

$$\frac{\sum_l^L (label^l(E') - \mu)^2}{|L|},$$

where $label^l(E')$ is number of edges having label l in edges E' and $\mu = \sum_l^L label^l(E')/|L|$. Based on this measure, we can now propose criteria to select the seeds for our block construction.

Definition 8 (Seed) *A vertex $s \in V$ belongs to the seed set S, which satisfies:*

$i)$ $\forall s' \in S, s \notin neighbor(s')$

$ii)$ $\forall v \in V \setminus S, \beta \cdot Deg^{\mathcal{L}}(s) + (1-\beta) \cdot VOE(adjacent(s)) \geq \beta \cdot Deg^{\mathcal{L}}(v) + (1-\beta) \cdot VOE(adjacent(v))$

where $adjacent(x)$ denotes the set of edges with x as vertex, $Deg^{\mathcal{L}}(x) = \sum_{l \in \mathcal{L}} label^l(adjacent(x))$ and β is an ad-hoc parameter in $[0, 1]$.

The first criterion avoids to select adjacent seeds which allows a better coverage of the large graph. Observe that we can decide to extend this criterion by requiring a distance of 2 or 3 between two seeds (but remember that recent results exhibit an average path length of 3.7 for Twitter [15]). The second criterion means that the seeds are selected among the most popular accounts (*i.e.*, accounts with the largest number of labels on adjacent edges) but also those which federates a community around them with common/similar interest (*i.e.*, labels), that will lead to a good block/partition construction. Indeed by selecting seeds with high degree and high VOE we expect to build dense blocks with a few number of distinct labels, leading to lower communication, thus processing time, when querying or mining the graph. Observe that the size of the seed set is a preset value, larger than the number of requested partitions.

We implemented the seed selection as a greedy algorithm presented in Algorithm 1. This algorithm has a linear complexity since it relies on a single scan of the list of vertices ordered by their degree. Consequently the number of vertices checked is comprised between $|S|$ and $|V|$, where $|S|$ denotes the size of the seed set.

The detailed procedure of selection is described in Algorithm 1. Example 4 illustrates the seed selection algorithm.

Example 4 *Suppose we have a 2-layer graph G', with labels $L = \{a, b\}$, and 3 candidate seeds, $\{c_1, c_2, c_3\}$, available. The statistics of labels on their adjacent edges are*
$(label^a(adjacent(c_1)), label^b(adjacent(c_1))) = (6, 4)$,
$(label^a(adjacent(c_2)), label^b(adjacent(c_2))) = (2, 6)$ *and*
$(label^a(adjacent(c_3)), label^b(adjacent(c_3))) = (4, 4)$.
Thus, we can calculate the $Deg^{\mathcal{L}}(c_i)$ of each candidate, represented here as a vector $dv = [6+4,\ 2+6,\ 4+4]^T = [10,\ 8,\ 8]^T$. We normalize this scores and get the vector $dv' = [10/26,\ 8/26,\ 8/26]^T$. At the same time, their VOEs can be figured out via formulas above. For instance, for c_1, its VOE is $((6-5)^2 + (4-5)^2)/2 = 1$. We compute similarly the values for c_2 and c_3 and find 4 and 0 respectively. We also represent them as a vector $ov = [1,\ 4,\ 0]$. After normalization we get the vector $ov' = [1/5,\ 4/5,\ 0/5]^T$. We assume in the following that the ad-hoc parameter β is set to 0.5. We get consequently the final suitability *score for each candidate, as $(1-0.5)dv' + 0.5ov' = [0.292,\ 0.554,\ 0.154]^T$. We conclude that candidate c_2 is most suitable for becoming a seed among these three candidates since it has the maximum value.*

Algorithm 1: Seeds selection algorithm

input : the graph $G(V, E)$, the set of labels L
output : a set of seeds S
parameter: the number of seeds m and the importance factor of VOE β

```
C ← ∅; //the seed-candidates
N ← ∅; //the neighbor vertices of existing candidates
while |C| < m do
    v = argmax_{x∈V\N} Deg(x); //the vertex with max "Degree"
    C+ = {v → (Deg(v), VOE(Adjacent(v)))} ;
    N+ = neighbor(v) ;
C' ← norm(C) ; //we normalize the Deg() and VOE() values for elements in C
foreach c ∈ C' do
    (deg, voe) = c.value() ;
    c.valueUpdate(deg * (1 − β) + voe * β) ;
S ← C'.maxByValue(m); //to select the candidates with first m max values
Return S;
```

5. Block Refinement and Merging

Our edge allocation algorithm ensures the construction of blocks which present a topic homogeneity but which may lead to a block size heterogeneity with small blocks or oppositely very large blocks. To make blocks fit the expected partition size we propose to proceed in two steps: first we split the oversized blocks into smaller ones and second we merge the different blocks to get the final partitioning.

For the first step, splitting a block into several sub-blocks (*i.e.*, re-allocating edges to different sub-blocks) in a random way would reduce the benefit of our block constructions where locality and topic similarity were considered. So we propose to iterate recursively the edge allocation algorithm by selecting $\lceil size(B)/max\_size\_block \rceil$ seeds within the oversized block B. The block construction process stops when all blocks have a size lower than the given parameter max_size_block.

Finally to get the final partitions and to respect their maximum size we perform block merging. Relying only

on block sizes to decide which blocks to merge (we call this strategy the *4/3 algorithm* in the following) would result in partitions composed by sub-blocks topologically and/or thematically distant. So we propose to exploit the block locality to decide the blocks to merge. To achieve this we consider the *block meta-graph*, a weighted directed graph G', where the vertex v'_i represents the block b_i and the weight w_{b_i,s_j} on edge $e'_{i,j}$ is a vector which is the sum of edge connectivity scores from every edge in b_i to seed s_j(of block b_j), in other words, $\oplus_{e \in b_i}(E^e_\epsilon)_j$, then we can merge the blocks according to Algorithm 2.

Algorithm 2: Meta-Block allocation algorithm

input : A set of blocks B of size n, a set of partitions P of size m, the labels L, the partition maximal size z
output : Each block is allocated to a $p_j \in P$

Initialization to avoid large blocks;
$B' \leftarrow \emptyset$;
foreach b_i *in the* B **do**
  $b_i.size \leftarrow \sum_l^L label^l(b_i)$
  if $b_i.size > z$ **then** $B' \leftarrow B' \cup split(b_i)$ **else** $B' \leftarrow B' \cup b_i$
while $B' \neq \emptyset$ **do**
  $p_i =$ `smallest(`P`)` //retrieve the smallest partition;
  $V = B'$`.multiple(`$P^L(p_i), w^T_{b_i,s_j}$`)` ; //P^L: blocks profile matrix
  $b \leftarrow$ `largest(`V`)` ; //to select the block with largest value in V
  $p_i =$ `merge(`p_i, b`)` ; //merge b with the smallest partition
  $B' = B' - \{b\}$;
Return P ;

Note that functions `split()` and `merge()` in Algorithm 2 correspond to the split and merge functions introduced earlier in this Section.

6. Experiments

To evaluate our partitioning method on multi-layer graphs, we consider three graph-basic fundamental operations used in graph analysis and mining: *Finding users with similar interest*, *Shortest Paths* and *Random Walk*, over our real-world datasets. We conduct the experiments on a Spark(1.6.1)[16] platform which is deployed on a 16 nodes cluster representing 100 cores. Observe that while the memory available allowed to store and handle our experimental datasets, we discard this centralized solution which was quickly outperformed by distributed solutions for our experiments. As the workload, *i.e.*, the number of edges, is balanced between partitions for each partitioning strategies used in experiments, we focus on the runtime of each method to evaluate their performance.

Table 1. Datasets

	Edges	Vertices	Layers
Twitter	85,062,587	2,141,325	17
Higgs	15,450,464	456,630	4

6.1. Settings

Datasets

Two distinct datasets whose main features are summarized in Table 1 are used in our experiments:

TWITTER: An excerpt of Twitter with around 2 million vertices and 85 million edges corresponding to the sole $follow$ links [17]. There exists a label l on an edge (u, v) to depict the interest of u for the posts (tweets) of v on topic l. 17 labels are extracted from a supervised classification model to tag the different edges: *leisure, technology, entertainment, labor, social, disaster, sport, life, environment, religion, law, politics, business, health, education, climate, war*.

HIGGS[18] dataset built from the graph derived from Twitter during the discussions and interactions on the news of discovery of elusive Higgs boson on 4th July 2012. For this dataset no topic is considered but we distinguished 4 different interactions between users extracted in this graph and used to tag the edges: *following, replying, mentioning and retweeting*.

These datasets are large enough to illustrate the benefit of our partitioning approach. To handle larger graphs one may add more cluster nodes.

Competitors

We compare our algorithm with the most popular edge-partitioning algorithms, *i.e.* the ones proposed in *GraphX* and in *PowerGraph*. More precisely we compare the following partitioning strategies:

- Random edges allocation methods [7], such as *RandomVertexCut, CanonicalRandomVertexCut, EdgePartition1D and EdgePartition2D*.
- Greedy method introduced in PowerGraph [1].
- Our block-based method, esp., the *Meta-Block* version has been employed in all following experiments as it always outperform *4/3 block allocation* method (greedy block allocation). We choose to initially build ten times more blocks than the expected number of partitions.

6.2. Finding user with similar interest

For the first experiment, we compare the impact of the partitioning strategies when performing the regular expression query $(v, l, *, l, x)$ which means we want to find all the vertices x which are connected to a given vertex by an edge with the same label l as v. This regular expression is a frequently-used query in graph mining operation [19] when we want to detect users x with similar interest l (whatever l may be) than user v. We run the different graph-partitioning algorithms to build partitionings with different sizes for each method. Then we conduct the regular expression query on these different partitionings and compare the resulting performances for 100,000 randomly selected vertices. Performance times correspond to total time (including pre-processing time).

In Figure 3, we observe that for the Twitter graph the communication cost linearly increases with the number of partitions for all partitionings. However our partitioning strategy largely reduces the execution time. For instance with 100 partitions, we obtain a gain of 0.84, 0.3 and 0.33 for the random, EdgePartition2D or PowerGraph partitionings respectively.

We perform similar experiments with Higgs graph and results are depicted in Figure 4. Our method also outperforms existing ones for this dataset with a similar gain which enlightens that our approach remains efficient for graphs with a small number of distinct labels. Indeed, our partitions present a topic-homogeneity, so few distinct labels inside a partition, whatever the number of existing labels is. In plus we group edges based on topological proximity. Consequently the search for this regular expression on the graph is mainly processed within each partition, with few communication between partitions, whatever the size of the graph and/or the number of label are.

6.3. Shortest Path

For this experiment we study the *shortest path* algorithm used in many applications such as community detection, influence or similarity computation, etc. Here, in our multi-graph assumption, we consider that a *shortest path (SP)* search corresponds to the following problem: for a graph G, given some landmarks $\Lambda \subseteq V$ and a specific label t, we want to find for every vertex $v \in V$ the shortest path to each reachable $\lambda \in \Lambda$ in the graph restriction G_t. The result of this search is a vector of shortest path values to each landmark for each vertex. Observe that since the algorithm can be implemented using recursive search of neighbors (BFS), the implementation can be naturally deployed in

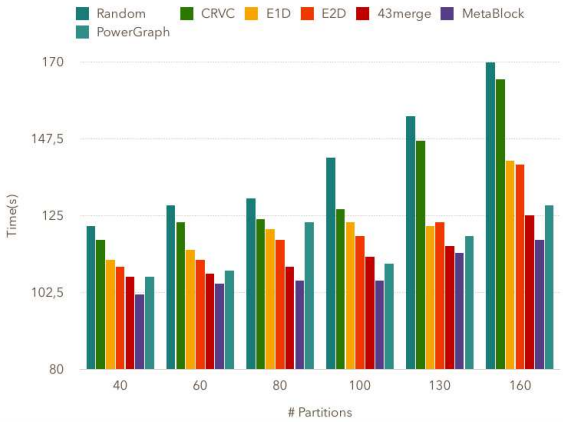


Figure 3. Regular expression query on Twitter graph

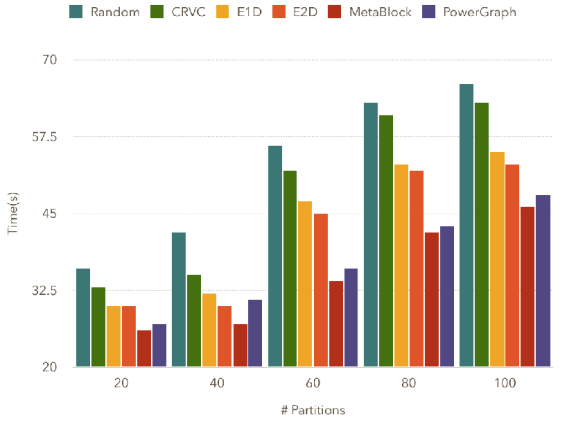


Figure 4. Regular expression query on Higgs graph

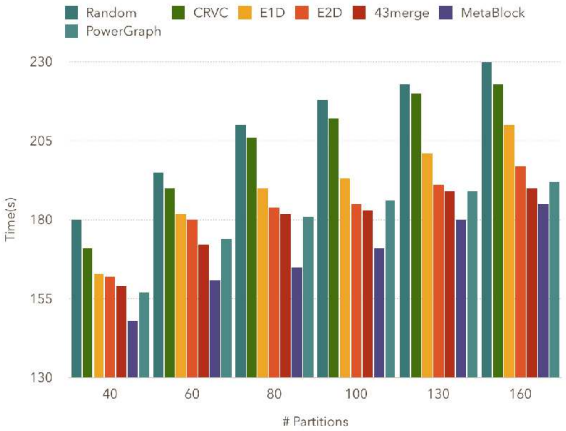


Figure 5. Shortest path on Twitter graph

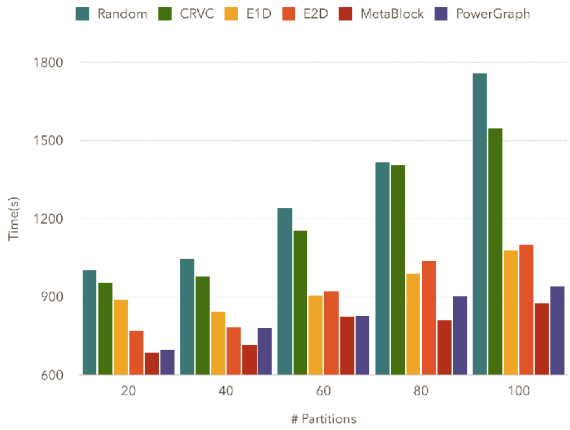


Figure 6. Shortest path on Higgs graph

Pregel-like systems, such as PowerGraph and GraphX. We construct 40-160 partitions for Twitter graph and we pick up a set of vertices randomly selected as landmarks (20 and 50 for respectively Twitter and Higgs datasets). The results are presented in Fig. 5. We notice that the runtime decreases from 10 to 60 percent, compared to other methods. The rationale is that our partitioning considers the connectivity and the labels for building blocks. So the resulting partitions usually present large subgraphs extracted from restriction graphs. Since the shortest paths are computed on the restricted graph of a chosen label, the computation is more likely to stay within a partition and thus reduces the communications between partitions. For the competitors the vertices which belong to the same restriction graph are allocated to much more partitions. So the vertex replication factor will be higher and more messages between partitions are required when performing a computation leading to higher runtimes. We make similar observations for the Higgs dataset (Fig. 6), so our partitioning strategy is also efficient for graphs with less labels.

6.4. Random Walk

Random-walks are the building blocks for numerous widespread algorithms in different areas such as recommendation, similarity computation, item ranking or knowledge inference. In this Section, we conduct an experiment to illustrate how random walk-based algorithms may benefit from our partitioning algorithm. Algorithms on labeled graphs generally consider edge label as independent during the computation. However, the layers or labels of multi-graph are not independent in a variety of applications [20]. So we propose two types of random walk implementations for our experiments:
1) *Independent-label Random Walks,* where the random walks are conducted over only edge with queried label λ, *i.e.*, the graph restriction G_λ.
2) *Correlated-label Random Walks,* where we rely on the *Correlation Matrix of Labels* L, $C(L)$, in Sec.3.3 into random walks execution. Thus the transitions for random walks consider all outgoing edges but with a weight which depends on the correlation between its label and the requested label. We choose here the Pearson correlation score, but other correlation scores may be investigate in the future. More precisely the transition probabilities are defined as:

$$P(v_i \to v_j) = \begin{cases} \frac{P(v_i \to v_j|\lambda, C(L))}{\sum_{e_{i,k} \in E} P(v_i \to v_k|\lambda, C(L))} & if\ e_{i,j} \in E, \\ 0 & otherwise. \end{cases}$$

where E is the edges of multi-layer graph G, λ is the label to query and $C(L)$ is the correlation matrix of labels L. In above formula, we propose to calculate the overall probabilities of each transition via linearly summarizing all conditional ones. For the conditional probability $P(v_p \to v_q|\lambda, C(L), e_{p,q} \in E)$, it can be estimated as:

$$P(v_p \to v_q|\lambda, C(L), e_{p,q} \in E) = \sum_{l \in Labs(e_{p,q})} C_{k,t}(L)$$

where $Labs(e_{p,q})$ is the set of labels for the edge $e_{p,q}$.

In Fig. 7, we see the benefit of our partitioning algorithm when performing independent-label random walks on Twitter dataset with a 30%-70% decrease for the runtime compared to executions on partitions produced by GraphX strategies, and around 5-10% decrease for GraphLab. The rationale is that with our partition building the random walk is likely to remain in the same partition since our partition algorithm considers graph distance and labels.

Fig. 8 presents the results for correlated-label random walks on Twitter dataset. We notice that our method, $MetaBlock\_hasCML$, always provides the best execution times. The gain is even more important compared to independant-label random walks. This is due to our building which groups edges considering the topics and their correlation.

7. Conclusion

In this paper, we present our approach for edge partitioning for multi-layer graphs which takes into account the locality of graph structure and the distribution of edge labels. We introduce new metrics, like Edge Connectivity Score and Blocks Profile Matrix, to determine the edge allocation. Our experiments

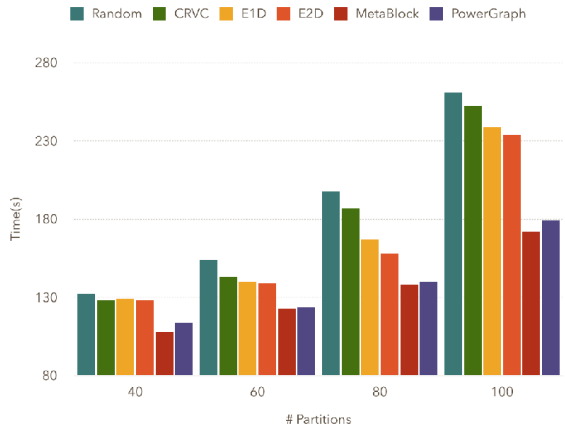


Figure 7. Independent-label random walks

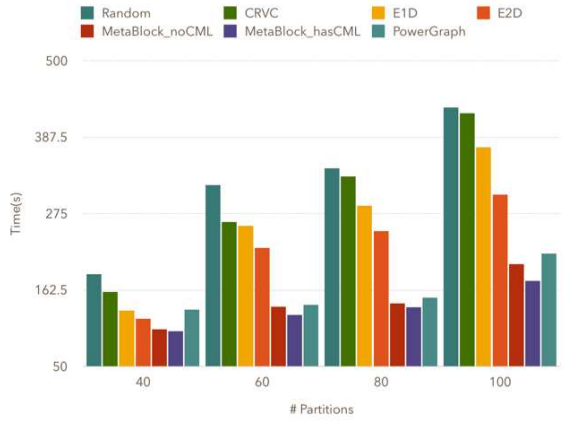


Figure 8. Correlated-label random walks

on real-world datasets illustrate that the partitioning provided by our proposal allows different classical graph algorithms for graph analysis or graph mining to be executed much faster than when using another graph partitioning strategy.

As future work, we first intend to extend our proposal to support more complex graph models, where for instance the vertices also have different types of labels. We also plan to investigate a self-adaptive mechanism to determine the different parameters of our algorithm such as the number of seeds. Finally we intend to investigate the other kinds of relevant/valuable graph operations that our approach may improve.

References

[1] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin, "PowerGraph: Distributed Graph-parallel Computation on Natural Graphs," in *OSDI*, pp. 17–30, 2012.

[2] L. G. Valiant, "A Bridging Model for Parallel Computation," *Commun. ACM*, vol. 33, no. 8, pp. 103–111, 1990.

[3] G. Karypis and V. Kumar, "A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs," *SIAM J. Scientific Computing*, vol. 20, no. 1, pp. 359–392, 1998.

[4] Apache, "Giraph." `http://giraph.apache.org`, 2012.

[5] G. Malewicz, M. H. Austern, A. J. C. Bik, J. C. Dehnert, I. Horn, N. Leiser, G. Czajkowski, and G. Inc, "Pregel: A System for Large-scale Graph Processing," in *SIGMOD*, pp. 135–146, 2010.

[6] Y. Low, D. Bickson, J. Gonzalez, C. Guestrin, A. Kyrola, and J. M. Hellerstein, "Distributed GraphLab: A Framework for Machine Learning and Data Mining in the Cloud," *Proc. VLDB Endow.*, vol. 5, no. 8, pp. 716–727, 2012.

[7] R. S. Xin, J. E. Gonzalez, M. J. Franklin, and I. Stoica, "GraphX: A Resilient Distributed Graph System on Spark," in *GRADES*, pp. 2:1–2:6, 2013.

[8] D. Alistarh, J. Iglesias, and M. Vojnovic, "Streaming Min-max Hypergraph Partitioning," in *NIPS*, pp. 1900–1908, 2015.

[9] K. Wei, R. Iyer, S. Wang, W. Bai, and J. Bilmes, "Mixed Robust/Average Submodular Partitioning: Fast Algorithms, Guarantees, and Applications," in *NIPS*, pp. 2233–2241, 2015.

[10] J. Chen, W. Dai, Y. Sun, and J. Dy, "Clustering and Ranking in Heterogeneous Information Networks via Gamma-Poisson Model," in *ICDM*, pp. 424–432, 2015.

[11] Y. Huang and X. Gao, "Clustering on Heterogeneous Networks," *Wiley Int. Rev. Data Min. and Knowl. Disc.*, vol. 4, no. 3, pp. 213–233, 2014.

[12] A. Farseev, I. Samborskii, A. Filchenkov, and T. Chua, "Cross-Domain Recommendation via Clustering on Multi-Layer Graphs," in *SIGIR*, pp. 195–204, 2017.

[13] H. Fani, F. Zarrinkalam, E. Bagheri, and W. Du, *Time-Sensitive Topic-Based Communities on Twitter*, pp. 192–204. Cham: Springer International Publishing, 2016.

[14] H. A. Abdelbary, A. M. ElKorany, and R. Bahgat, "Utilizing Deep Learning for Content-based Community Detection," in *2014 Science and Information Conference*, pp. 777–784, 2014.

[15] Q. Grossetti, C. Constantin, C. du Mouza, and N. Travers, "An Homophily-based Approach for Fast Post Recommendation in Microblogging Systems," in *EDBT*, (Wien, Austria), Mar. 2018.

[16] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauly, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing," in *NSDI*, pp. 15–28, 2012.

[17] C. Constantin, R. Dahimene, Q. Grossetti, and C. du Mouza, "Finding Users of Interest in Micro-blogging Systems," in *EDBT*, pp. 5–16, 2016.

[18] M. De Domenico, A. Lima, P. Mougel, and M. Musolesi, "The Anatomy of a Scientific Rumor," vol. 3, pp. 2980 EP –, 10 2013.

[19] P. Barceló, L. Libkin, and J. L. Reutter, "Querying Graph Patterns," in *PODS*, pp. 199–210, 2011.

[20] Y. Wang, X. Lin, and Q. Zhang, "Towards Metric Fusion on Multi-view Data: A Cross-view Based Graph Random Walk Approach," in *CIKM*, pp. 805–810, 2013.