Simulated Annealing for Edge Partitioning

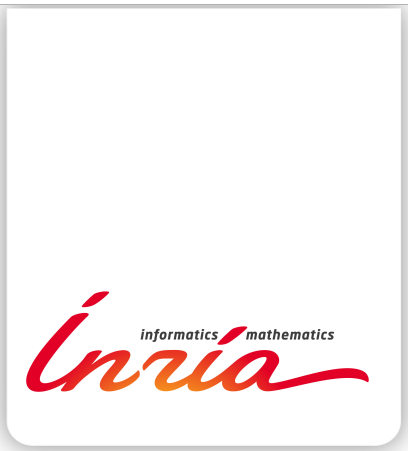
Hlib Mykhailenko, Giovanni Neglia, Fabrice Huet

Simulated Annealing for Edge Partitioning

Hlib Mykhailenko, Giovanni Neglia,
Fabrice Huet

RESEARCH REPORT
N° 9019
January 2017
Project-Team NEO

ISSN 0249-6399 ISRN INRIA/RR--9019--FR+ENG



Simulated Annealing for Edge Partitioning

Hlib Mykhailenko*, Giovanni Neglia†,
Fabrice Huet‡

Project-Team NEO

Research Report n° 9019 — January 2017 — 18 pages

Abstract: In distributed graph computation, graph partitioning is an important preliminary step, because the computation time can significantly depend on how the graph has been split among the different executors. In this paper, we propose a framework for distributed edge partitioning based on simulated annealing. The framework can be used to optimize a large family of partitioning metrics. We provide sufficient conditions for convergence to the optimum as well as discuss which metrics can be efficiently optimized in a distributed way. We implemented our partitioners in Apache GraphX and performed a preliminary comparison with *JA-BE-JA-VC*, a state-of-the-art partitioner that inspired our approach. We show that our approach can provide improvements, but further research is required to identify suitable metrics to optimize as well as to design a more efficient exploration phase for our algorithm without sacrificing convergence properties.

Key-words: graph partitioning, edge partitioning, simulated annealing, Spark, GraphX

* Université Côte d'Azur, Inria, France. Email: `hlib.mykhailenko@inria.fr`
† Université Côte d'Azur, Inria, France. Email: `giovanni.neglia@inria.fr`
‡ Université Côte d'Azur, CNRS, I3S, France. Email: `fabrice.huet@unice.fr`

RESEARCH CENTRE
SOPHIA ANTIPOLIS – MÉDITERRANÉE

2004 route des Lucioles - BP 93
06902 Sophia Antipolis Cedex

Recuit Simulé pour Partitionnement des Arêtes

Résumé : Dans le calcul de graphe distribué, le partitionnement de graphe est une étape préliminaire importante, car le temps de calcul peut dépendre significativement de comment le graphe a été divisé entre les différents exécuteurs. Dans cet article, nous proposons un framework pour le partage des arêtes distribué basé sur le recuit simulé. Le framework peut être utilisé pour optimiser une grande famille de métriques de partitionnement. Nous fournissons des conditions suffisantes pour la convergence vers l'optimum et nous discutons des métriques qui peuvent être efficacement optimisées de manière distribuée. Nous avons implémenté notre partitionneur dans Apache GraphX et effectué une comparaison préliminaire avec *JA-BE-JA-VC*, partitionneur à l'état de l'art qui a inspiré notre approche. Nous montrons que notre approche peut apporter des améliorations, mais de nouvelles recherches sont nécessaires pour identifier des métriques appropriées pour optimiser ainsi que pour concevoir une phase d'exploration plus efficace pour notre algorithme sans sacrifier les propriétés de convergence.

Mots-clés : partitionnement de graphe, partitionnement des arêtes, recuit simulé, Spark, GraphX

1 Introduction

Analyzing large graphs is a space intensive operation which usually cannot be performed on a single machine. Hence, there has been a lot of work dedicated to designing programming models and building distributed middleware to perform such a computation on a set of machines, often called slaves or executors. Usually, the first step consists in partitioning the graph and distributing it over the set of executors. That way, each executor processes its own part of the graph locally, and then, it shares it with other executors. A good partitioning reduces graph processing time achieving computational balance and limited communication requirements, respectively if the partitions have roughly the same size and have low intersection.

There are two approaches to partition the graph - vertex and edge partitioning - depending if vertices or edges are assigned to the partition. Recently, it has been advocated [6] [4] that edge partitioning is more effective for power-law graphs and some recent distributed computation frameworks like GraphX [10] and PowerGraph [6] rely indeed on it. Edge partitioning is also referred to as vertex-cut partitioning because a vertex is "cut," if its edges are assigned to different partitions. These partitions will in general need to communicate during the graph processing phase to synchronize their computation. For this reason the number of vertices cut, and the number of their pieces are two common indicators of the partition quality in terms of communication requirements.

A new edge partitioning algorithm, called *JA-BE-JA-VC*, has been proposed in [9] and shown to significantly outperform existing algorithms. It iteratively improves on an initial (arbitrary) edge partition assignment, by allowing two edges to swap their assignment if this seems to be beneficial to reduce the number of cuts of the corresponding vertices. In order to avoid to get stuck at local minima, *JA-BE-JA-VC* borrows from simulated annealing (*SA*) the idea to permit apparently detrimentals swaps at early stages of the partitioning.

In this paper, we develop this initial inspiration and propose graph partitioners based on *SA* for Spark. To this purpose, we first reverse engineer *JA-BE-JA-VC* to show which metric it is targeting. Second, we propose a general SA framework that can optimize a large spectrum of objective functions, and for which convergence results can be proven. A naive implementation of this approach (as well as of *JA-BE-JA-VC*) would require a significant number of costly synchronization operations during the partitioning. Then, a third contribution of this paper, is to explain how these algorithms can be efficiently implemented in a distributed architecture as Spark. As a proof of concept, we perform some preliminary experiments considering a different objective function that takes into account both communication cost and computational balance. We show that this objective function may obtain even better partitions than *JA-BE-JA-VC*, but this does not happen consistently and further investigation is required to choose the edges to swap.

The rest of the paper is organized as follows. We first start by introducing the notation and tools used in this work (Section 2). Next we reverse-engineer *JA-BE-JA-VC* algorithm in Section 3. Then we present the *SA* framework

for edge partitioning in Section 4. We compare *JA-BE-JA-VC* and the *SA* framework in Section 5. Finally, we conclude in Section 6.

2 Background and notation

Apache Spark [12] is a large-scale distributed framework for data processing. It is built on the notion of Resilient Distributed Dataset (RDD) [11]. RDD is an immutable lazy-evaluated distributed collection with predefined functions which can be applied to them. Apache GraphX [10] is a widely-used Spark's API for graph processing. GraphX programs are executed as Apache Spark Jobs. In this framework, graphs are composed of two RDDs: a vertex RDD and an edge RDD and partitions are built using a vertex-cut partitioner. In this paper, we propose new edge partitioners for GraphX, in the sense that i) GraphX is used first to partition the graph, ii) the graph is later loaded in GraphX according to this partition in order to be processed.

Consider an undirected graph $G = (V, E)$ with V and E representing the set of vertices and edges respectively. The graph will be partitioned in N distinct partitions, that we identify with N different colors in the set C. Let $E(c)$ denote the set of edges with color $c \in C$, then $E = \bigcup_{c \in C} E(c)$. Given a vertex $v \in V$, its degree is denoted by d_v and the number of its edges with color c is denoted by $n_v(c)$.

The quality of a partition can be evaluated using multiple metrics [9, 7, 8] . We introduce four of them which will be used in the remaining of the paper.

`Balance` (denoted as BAL) is the ratio between the maximum and the average number of edges in the partitions:

$$\text{BAL} = \frac{\max_{c=1,\dots N}(|E(c)|)}{|E|/N}. \tag{1}$$

`STD` is the normalized standard deviation of partition sizes (in terms of number of edges):

$$\text{STD} = \sqrt{\sum_{c \in C} \left(\frac{|E(c)|}{|E|/N} - 1 \right)^2 \frac{1}{N}}. \tag{2}$$

`Vertex-cut` (denoted as VC) represents the number of pieces in which the vertices were cut:

$$\text{VC} = |V| - \sum_{v \in V} \sum_{c \in C} \mathbb{1}(n_v(c) = d_v). \tag{3}$$

`Communication cost` (denoted as CC) is the total number of vertices in partitioned graph excluding those vertices that were not cut:

$$\mathrm{CC} = \sum_{v \in V} \sum_{c \in C} \mathbb{1}(0 < n_v(c) < d_v). \tag{4}$$

The first two metrics quantify the different number of edges across partitions and indicate how balanced the partitioning is. The other two metrics are related to the vertices which are cut and will cause communication among executors during the computation phase.

JA-BE-JA-VC [9] is a recently proposed edge partitioner. Given an initial color assignment to edges (e.g. a random one), the algorithm iteratively selects two vertices u and u'. For each of this two vertices, it then selects an edge among those whose color is less represented in their neighborhood. For example, considering u, it will select an edge of color $\hat{c} \in \operatorname{argmin} n_u(c)$. Let us denote these two edges as (u, v) and (u', v') with color respectively c and c'. The algorithm always swaps the colors of the two edges if

$$\begin{aligned} g(u, v, c) + g(u', v', c') < g(u, v, c') + g(u', v', c) \\ + \frac{1}{d_u} + \frac{1}{d_v} + \frac{1}{d_{u'}} + \frac{1}{d_{v'}}, \end{aligned} \tag{5}$$

where $g(u, v, c) \triangleq \frac{n_u(c)}{d_u} + \frac{n_v(c)}{d_v}$. The more links of color c the two nodes u and v have, the larger $g(u, v, c)$ is, and then the two nodes are potentially cut in a smaller number of pieces. In particular, if all edges of u and v have color c, $g(u, v, c)$ attains its maximum value equal to 2. If we consider that $g(u, v, c)$ is a measure of the quality of the current assignment, (5) compares the current assignment with an assignment were colors are swapped.[1] While we have provided an interpretation of $g(u, v, c)$, one may wonder which objective function (if any) *JA-BE-JA-VC* is optimizing by swapping color according to the criterium in (5) and if it is related to one of the usual partitioning quality metrics like VC or CC defined above. In Sec. 3 we provide an answer to such questions.

The authors of [9] state that, in order to avoid to get stuck in local optima (of this still unknown objective function), one can introduce the possibility to accept changes that do not satisfy (5), especially during the first iterations. To this purpose, inspired by simulated annealing (*SA*) [5, Chapter 7] they introduce a positive parameter T (the temperature) and change condition (5) as follows:

$$\begin{aligned} g(u, v, c) + g(u', v', c) <(1 + T)\Big[g(u, v, c') + g(u', v', c) \\ + \frac{1}{d_u} + \frac{1}{d_v} + \frac{1}{d_{u'}} + \frac{1}{d_{v'}}\Big], \end{aligned} \tag{6}$$

where T decreases linearly from some initial value to zero.

[1] The additional terms on the right hand side correspond to the fact that there is one link more of color c' (resp. c) for u and v (resp. c')

JA-BE-JA-VC is presented in [9] as a distributed algorithm, because an edge swap requires only information available to the nodes involved, i.e. u, v, u' and v'. We observe that this property is not necessarily helpful for performing the partitioning operation on distributed computation frameworks like Spark, because this information is not necessarily local to the executor that processes the two edges. Indeed, while the executor may have access to both the edges (u, v) and (u', v'), it may not know the current value of edges of a given color that each vertex has (e.g. it may not know $n_u(c)$), because these other edges might be assigned to other partitioners. Moreover, the color of these remote edges might have their color changed concurrently. Hence, expensive communication exchange among executors could be required to implement *JA-BE-JA-VC* in Spark. In Sec. 4 we discuss how to modify *JA-BE-JA-VC* and our algorithm in order to prevent this problem.

3 Reverse Engineering *JA-BE-JA-VC*

The first contribution of this paper is to identify the global objective function *JA-BE-JA-VC* is optimizing when links are swapped according to (5). Let m_c be the initial number of edges with color c. Condition (5) corresponds to greedily minimizing the function

$$\mathcal{E}_{comm} = \frac{1}{2|E|(1-\frac{1}{N})}\left(2|E| - \sum_{v\in V}\sum_{c\in C}\frac{n_v(c)^2}{d_v}\right), \tag{7}$$

under the constraint that at each step $|E(c)| = m_c$ for any color c. The constraint is easy to understand, because *JA-BE-JA-VC* simply swaps colors of two edges so the number of edges of a given color is always equal to the initial value. It is easy to show that a swap makes $\mathcal{E}_{comm}$ decrease if and only if condition (5) is satisfied:

$$\begin{gathered}
\Delta\mathcal{E}_{comm} < 0 \\
\hat{n}_u(c) = n_u(c) - 1 \\
\hat{n}_u(c') = n_u(c') + 1 \\
\hat{n}_v(c) = n_v(c) - 1 \\
\hat{n}_v(c') = n_v(c') + 1 \\
\hat{n}_{u'}(c) = n_{u'}(c) + 1 \\
\hat{n}_{u'}(c') = n_{u'}(c') - 1 \\
\hat{n}_{v'}(c) = n_{v'}(c) + 1 \\
\hat{n}_{v'}(c') = n_{v'}(c') - 1 \\
\widehat{\mathcal{E}_{comm}} - \mathcal{E}_{comm} < 0
\end{gathered}$$

$$-\frac{\hat{n}_u(c)^2}{d_u}-\frac{\hat{n}_v(c)^2}{d_v}-\frac{\hat{n}_u(c')^2}{d_u}-\frac{\hat{n}_v(c')^2}{d_v}-\frac{\hat{n}_{u'}(c)^2}{d_{u'}}-\frac{\hat{n}_{v'}(c)^2}{d_{v'}}-\frac{\hat{n}_{u'}(c')^2}{d_{u'}}-\frac{\hat{n}_{v'}(c')^2}{d_{u'}}+\frac{n_u(c)^2}{d_u}$$
$$+\frac{n_v(c)^2}{d_v}+\frac{n_u(c')^2}{d_u}+\frac{n_v(c')^2}{d_v}+\frac{n_{u'}(c)^2}{d_{u'}}+\frac{n_{v'}(c)^2}{d_{v'}}+\frac{n_{u'}(c')^2}{d_{u'}}+\frac{n_{v'}(c')^2}{d_{v'}}<0$$

$$\frac{-\hat{n}_u(c)^2-\hat{n}_u(c')^2+n_u(c)^2+n_u(c')^2}{d_u}+\frac{-\hat{n}_v(c)^2-\hat{n}_v(c')^2+n_v(c)^2+n_v(c')^2}{d_v}$$
$$+\frac{-\hat{n}_{u'}(c)^2-\hat{n}_{u'}(c')^2+n_{u'}(c)^2+n_{u'}(c')^2}{d_{u'}}+\frac{-\hat{n}_{v'}(c)^2-\hat{n}_{v'}(c')^2+n_{v'}(c)^2+n_{v'}(c')^2}{d_{v'}}<0$$

$$\frac{n_u(c)^2-(n_u(c)-1)^2+n_u(c')^2-(n_u(c')+1)^2}{d_u}$$
$$+\frac{n_v(c)^2-(n_v(c)-1)^2+n_v(c')^2-(n_v(c')+1)^2}{d_v}$$
$$+\frac{n_{u'}(c)^2-(n_{u'}(c)+1)^2+n_{u'}(c')^2-(n_{u'}(c')-1)^2}{d_{u'}}$$
$$+\frac{n_{v'}(c)^2-(n_{v'}(c)+1)^2+n_{v'}(c')^2-(n_{v'}(c')-1)^2}{d_{v'}}<0$$

$$\frac{2n_u(c)-2n_u(c')-2}{d_u}+\frac{2n_v(c)-2n_v(c')-2}{d_v}$$
$$+\frac{2n_{u'}(c')-2n_{u'}(c)-2}{d_{u'}}+\frac{2n_{v'}(c')-2n_{v'}(c)-2}{d_{v'}}<0$$

$$2(\frac{n_u(c)}{d_u}+\frac{n_v(c)}{d_v}-\frac{n_u(c')}{d_u}-\frac{n_v(c')}{d_v}$$
$$+\frac{n_{u'}(c')}{d_{u'}}+\frac{n_{v'}(c')}{d_{v'}}-\frac{n_{u'}(c)}{d_{u'}}-\frac{n_{v'}(c)}{d_{v'}}-\frac{1}{d_u}-\frac{1}{d_v}-\frac{1}{d_{u'}}-\frac{1}{d_{v'}})<0$$

$$2\left(g(u,v,c) - g(u,v,c') + g(u',v',c') - g(u',v',c) - \frac{1}{d_u} - \frac{1}{d_v} - \frac{1}{d_{u'}} - \frac{1}{d_{v'}}\right) < 0$$
$$g(u,v,c) + g(u',v',c') < g(u,v,c') + g(u',v',c) + \frac{1}{d_u} + \frac{1}{d_v} + \frac{1}{d_{u'}} + \frac{1}{d_{v'}},$$

where $\widehat{\mathcal{E}_{comm}}$ denotes energy value after swap is occurred.

We could equivalently, and more succinctly, state that *JA-BE-JA-VC* is maximizing $\sum_{v \in V} \sum_{c \in C} \frac{n_v(c)^2}{d_v}$. The advantage of (7) is that $\mathcal{E}_{comm}$ belongs to $[0, 1]$ and *SA* algorithms are usually presented as minimizing an energy function.

Because of the above considerations, *JA-BE-JA-VC* can be thought as a heuristic to solve the following problem:

$$\underset{\mathbf{c} \in C^{|E|}}{\text{minimize}} \ \mathcal{E}_{comm}, \ \text{subject to } E(c) = m_c$$

where $\mathbf{c}$ denotes the vectors of colors chosen for all the edges in the network.

While the greedy rule (5) would lead, in general, to a local minimum of $\mathcal{E}_{comm}$, one may wonder if rule (6), together with the specific criterium to choose the edges to swap in *JA-BE-JA-VC* and to change the temperature, can lead to a solution of the problem stated above. Unfortunately, it is possible to show that this is not the case in general.

A second remark is that (7) appears to be a quite arbitrary metric, and is not directly related to metrics like VC or CC. Our experiments show indeed that $\mathcal{E}_{comm}$ can decrease while both VC and CC increases, even if the evaluation in [9] indicates that this is not usually the case.

In the next section we address these two remarks.

4 A general *SA* framework for edge partitioning

Let us consider a general optimization problem

$$\begin{aligned} \underset{\mathbf{c} \in C^{|E|}}{\text{minimize}} \quad & \mathcal{E}(\mathbf{c}) \\ \text{subject to} \quad & \mathbf{c} \in D, \end{aligned}$$

where D is a generic set of constraints. We can solve this problem with *SA* as follows.

- given the current solution $\mathbf{c}$, select a possible alternative $\mathbf{c}' \in D$ with probability $q_{\mathbf{c},\mathbf{c}'}$
- if $\mathcal{E}(\mathbf{c}') \leq \mathcal{E}(\mathbf{c})$ accept the change, otherwise accept it with probability $\exp\left(\frac{\mathcal{E}(\mathbf{c}) - \mathcal{E}(\mathbf{c}')}{T}\right) < 1$.

If the selection probabilities $q_{\mathbf{c},\mathbf{c}'}$ are symmetric ($q_{\mathbf{c},\mathbf{c}'} = q_{\mathbf{c}',\mathbf{c}}$) and if the temperature decreases as $T_0 / \log(1 + k)$, where $k \geq 0$ is the iteration number and

the initial temperature T_0 is large enough, then this algorithm is guaranteed to converge (with probability 1) to the optimal solution of the above problem.[2]

This algorithm is very general and can be applied to any energy function $\mathcal{E}$ including the metrics described in Section 2. A practical limit is that the algorithm may not be easy to distribute for a generic function $\mathcal{E}(\mathbf{c})$, because of its dependency on the whole vector $\mathbf{c}$. Nevertheless, we can observe that a *SA* algorithm needs to evaluate only the energy differences $\mathcal{E}(\mathbf{c}') - \mathcal{E}(\mathbf{c})$. Then, as far as such differences depend only on a few elements of the vectors $\mathbf{c}$ and $\mathbf{c}'$, the algorithm has still a possibility to be implemented in a distributed way.

If the function $\mathcal{E}$ can be expressed as sum of potentials of the cliques of order not larger than r,[3] evaluating the energy difference requires only to evaluate the value of the potentials for the corresponding cliques (see [5, Chapter 7]). The energy function $\mathcal{E}_{comm}$ considered by *JA-BE-JA-VC* falls in this category and in fact at each step the energy difference between two states requires to count only the edges of those colors that u, v, u' and v' have (5). But, as we said, our framework is more general and can accommodate any function that can be expressed as sum of clique potentials. For example in what follows we consider the following function

$$\mathcal{E} = \mathcal{E}_{comm} + \alpha \mathcal{E}_{bal} \tag{8}$$

$$\mathcal{E}_{bal} = \frac{1}{|E|^2(1-\frac{1}{N})^2} \sum_{c \in C} \left(|E(c)| - \frac{|E|}{N} \right)^2, \tag{9}$$

that allows to trade off the communication requirements associated to a partition, captured by $\mathcal{E}_{comm}$, and the computational balance, captured by $\mathcal{E}_{bal}$.[4] The term $\mathcal{E}_{bal}$ indeed range from 0 for a perfectly balanced partition to 1 for a partition where all the edges have been assigned the same color. The parameter $\alpha > 0$ allows the user to tune the relative importance of the two terms.

The function $\mathcal{E}$ can be optimized according to the general framework we described above as follows. We select an edge uniformly at random (say it (u, v) with color c) from E and decide probabilistically if we want to swap its color with another edge ((u', v') with color c') or change the color c to another color c'' without affecting other edges. Both the edge (u', v') and the color c'' are selected uniformly at random from the corresponding sets. For a color swapping operation the difference of energy is equal to

$$\Delta\mathcal{E} = \frac{1}{|E|(1-\frac{1}{N})}(g(u,v,c) - g(u,v,c') + g(u',v',c') - g(u',v',c) - \frac{1}{d_u} - \frac{1}{d_v} - \frac{1}{d_{u'}} - \frac{1}{d_{u'}}), \tag{10}$$

[2] *JA-BE-JA-VC* does not satisfy any of these conditions, and then it is not guaranteed to converge to the optimal solution.

[3] Cliques of order 1 are nodes, cliques of order 2 are edges, etc..

[4] In [8] we have shown that linear combinations of similar metrics can be good predictors for the final computation time.

similarly to the condition (5) for *JA-BE-JA-VC*. For a simple color change operation, the change of energy is:

$$\Delta\mathcal{E} = \frac{1}{|E|(1-\frac{1}{N})}\left(g(u,v,c) - g(u,v,c'') - \frac{1}{d_u} - \frac{1}{d_v}\right) + \alpha\frac{2}{|E|^2(1-\frac{1}{N})^2}\left(n_u(c'') + n_v(c'') - n_u(c) - n_v(c) + 1\right). \quad (11)$$

In both cases, only information about the nodes involved and their neighborhood is required as it was the case for *JA-BE-JA-VC*. At the same time, the same difficulty noted in Sec. 2 holds. In an edge-centric distributed framework, in general the edges for a node are processed by different executors. A naive implementation of our *SA* algorithm of *JA-BE-JA-VC* would require each executor to propagate color updates (e.g. the new values of $n_u(c)$, $n_v(c)$, etc.) to other executors at each iteration leading to an unacceptable partitioning time.

In order to prevent this problem, we implemented the following distributed version of the algorithm. First, edges are randomly distributed among the executors, and each of them executes the general *SA* algorithm described above on the local set of edges for L iterations. No communication can take place among executors during this phase. After this phase is finished, communication is allowed, the correct values are computed and all the edges are again distributed at random among the executors. This reshuffle guarantees that any pair of edges has a probability to be considered for color swapping. We observe that during a local phase, an executor may compute erroneously the energy differences, if other executors are changing the color of local edges involving the same nodes. While the algorithm is intrinsically robust to such "errors," they can still prevent the convergence to the global optimum if they happen too often. It is then important to limit the number L of iterations during the local phase so that such errors do not happen to often.

To calculate L we considered following situation. When we change a color of an edge in partition j (P_j), in average there are $2\bar{d}$ (where $\bar{d}$ is the average degree in the graph) edges attached to this edge. Hence, the probability that all these $2\bar{d}$ edges will not situate in P_i is the following:

$$\left(1 - \frac{1}{N}\right)^{2\bar{d}}$$

Then, the probability that at least one edge from these $2\bar{d}$ edges will be in P_i is the following:

$$1 - \left(1 - \frac{1}{N}\right)^{2\bar{d}}$$

That way, maximum number of vertices in P_i affected by performing swaps in all other $N-1$ partitions is the following:

$$4\bar{d}(N-1)\left(1-\left(1-\frac{1}{N}\right)^{2\bar{d}}\right) \tag{12}$$

In case we perform L swaps during *local step*, then in P_i we have two sets of vertices: s_{direct}, $s_{indirect}$. s_{direct} consists of vertices that are directly affected by the L swaps performed in current partition, $|s_{direct}| <= 4L$. $s_{indirect}$ is the set of vertices that are affected by the swaps in other partitions; its size is at most the value in (12).

Hence, we need to be sure that all these two sets (s_{direct}, $s_{indirect}$) are significantly smaller than $\frac{|V|}{N}$:

$$max\left(4L, 4L\bar{d}(N-1)\left(1-\left(1-\frac{1}{N}\right)^{2\bar{d}}\right)\right) < 4L\left(1+4\bar{d}Ne^{-2\bar{d}}\right) << \frac{|V|}{N} \tag{13}$$

$$L << \frac{|V|}{4N\left(1+4\bar{d}Ne^{-2\bar{d}}\right)} \tag{14}$$

Given the same number of potential changes considered, this implementation requires L times less synchronization phases among the executors. Due to the large time required for the synchronization phase, one can expect the total partitioning time to be reduced by roughly the same factor. Our experiments show that this is the case. In the setting described in the following section, with $L = 200$, there is no difference between the final partitions produced by the two implementations, but the partitioning time for the naive one is 100 times larger.

The detailed steps of our algorithm are described in Alg. 1.

5 Evaluation

We present here some preliminary experimental results.

All our experiments were performed on a cluster of 2 nodes (1 master and 1 slave) with dual-Xeon E5-2680 v2 @2.80GHz with 192GB RAM and 10 cores (20 threads). We used Spark version 1.4.0 and Spark standalone cluster as a resource manager. We configured five Spark's properties as following:

- *spark.executor.memory* 20g
- *spark.driver.memory* 40g
- *spark.cores.max* 10
- *spark.local.dir* "/tmp"
- *spark.cleaner.ttl* 20

Algorithm 1 Implementation of *SA* for GraphX

```
1: procedure SimulatedAnnealing
2:     G ← randomlyAssignColors(G)
3:     round ← 0
4:     T ← T_init
5:     while T > 0 do
6:         G ← partitionRandomly(G)
7:         G ← propagateValues(G)
8:         for i < L do                          ▷ Executes locally on each partition
9:             if tossCoin(2/3) == "head" then   ▷ swapping with probability 2/3...
10:                e ← randomEdge(partition)
11:                e' ← randomEdge(partition)
12:                Δℰ ← computeDelta(e, e')
13:                if Δℰ < 0 then
14:                    swapColors(e, e')
15:                else
16:                    swapColorsWithProb(e, e', e^{-Δℰ/T})
17:                end if
18:            else                                ▷ changing...
19:                e ← randomEdge(partition)
20:                c' ← anotherColor(c)
21:                Δℰ ← computeDelta(e, c')
22:                if Δℰ < 0 then
23:                    changeColor(e, c')
24:                else
25:                    changeColorWithProb(e, c', e^{-Δℰ/T})
26:                end if
27:            end if
28:            i++
29:        end for
30:        round++
31:        T ← T_init − round ∗ ΔT
32:    end while
33:    G = partitionBasedOnColor(G)
34: end procedure
```

Since GraphX does not have a built-in function which provides values for the metrics introduced in 2, we wrote code which extracts these metric information [1].

We used two undirected graphs: the *email-Enron* graph (36,692 vertices/ 367,662 edges) and the *com-Amazon* graph (334,863 vertices/ 925,863 edges) provided by SNAP project [3].

We implemented *JA-BE-JA-VC* and *SA* (Alg. 1) for GraphX. In both cases we considered the distributed operation described at the end of the previous section: $L(= 200)$ steps are performed locally at each executor before performing a synchronization phase. For a fair comparison between the two algorithms, L corresponds in both cases to the number of alternative configurations considered (i.e. those for which the energy variation is computed). The source code is available online [2].

Each experiment consists of the following steps: i) launch the computational cluster; ii) load the given graph; iii) *color* the graph according to random partitioner (*CanonicalRandomVertexCut*); iv) *re-color* with *JA-BE-JA-VC* or *SA*; v) compute the partition metrics.

While *SA* is guaranteed to converge if T decreases as the inverse of the logarithm of the number of iterations, the convergence would be too slow for practical purposes. For this reason and to permit a simpler comparison of *SA* and *JA-BE-JA-VC*, in both cases the initial temperature decreases linearly from T_0 till 0 in 100 or 1000 iterations.

As we said our *SA* partitioner can be used to optimize different functions. We start by comparing *JA-BE-JA-VC* and *SA* when they have the same target $\mathcal{E}_{comm}$ in (7). When the objective function is the same, the two main differences between the algorithms are i) the way to choose the edges to swap, and ii) the rule to accept a change. In particular, *JA-BE-JA-VC* chooses edges whose color is the rarest in a neighborhood, while *SA* selects them uniformly at random. *JA-BE-JA-VC* then decides to swap or not the colors according to the deterministic rule 6, while *SA* adopts the probabilistic rule (see Section 4)

The corresponding results are in Tables 1, 2, row I and II, for different values of the initial temperature for *email-Enron*. Both the algorithms reduce the value of $\mathcal{E}_{comm}$, but the decrease is larger of *JA-BE-JA-VC*. This is essentially due to the fact that *JA-BE-JA-VC* performs more swaps, although the number of pairs of links to swap considered is the same (equal to L times the number of iterations). For example for the initial temperature $5 * 10^{-11}$ *SA* swaps the color of 8808 edges, while *JA-BE-JA-VC* swaps the color of 37057 edges. In fact, *SA* random edge selection leads to consider a large number of pairs that is not useful to swap, while *JA-BE-JA-VC* only considers candidates that are more likely to be advantageous to swap. In this sense *JA-BE-JA-VC* is greedier than *SA*: *JA-BE-JA-VC* exploits more, while *SA* explore more. Adopting the same choice for *SA* would lead to lose the condition $q_{\mathbf{c},\mathbf{c}'} = q_{\mathbf{c}',\mathbf{c}}$, that is required to guarantee convergence to the global minimum of the function. We plan to investigate in the future how to bias the selection process so that edges whose color is less represented are more likely to be selected, but still the condition $q_{\mathbf{c},\mathbf{c}'} = q_{\mathbf{c}',\mathbf{c}}$ is satisfied.

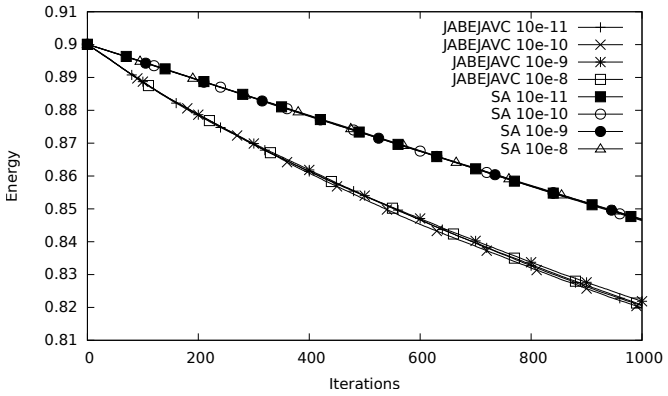


Figure 1: *Energy* value for *JA-BE-JA-VC* and *SA* using *email-Enron* graph (1000 iterations were performed)

Although *SA* seems to be less effective to reduce the energy, the results in Tables 1, 2 also shows that the function $\mathcal{E}_{comm}$ is not a necessarily a good proxy for the usual partition metrics like VC or CC. If fact, we see that for 100 iterations *SA* slightly outperforms *JA-BE-JA-VC* both in terms of VC and CC for the initial temperature values, even if its final energy is always larger. For 1000 iterations *JA-BE-JA-VC* performs better in terms of VC and worse in terms of CC. What is even more striking is that the increase in the number of iterations leads to a decrease of the energy value (as expected), but not necessarily to a decrease of VC and CC! These results suggest that more meaningful energy functions should probably be considered and our framework has the advantage to work for a large family of different objectives.

We now move to compare *JA-BE-JA-VC* with *SA* when the function $\mathcal{E} = \mathcal{E}_{comm} + \alpha\mathcal{E}_{bal}$ is considered. Figure 1 shows the evolution of the energy after each local phase and then after $L = 200$ local iterations.[5] While the two energies have different expression, the initial energy values are indistinguishable, because the starting point is an almost balanced partition and then $\mathcal{E}_{bal} << \mathcal{E}_{comm}$ and $\mathcal{E} \approx \mathcal{E}_{bal}$. As in the previous case, *JA-BE-JA-VC* appears to be greedier in reducing the energy function. Tables 1, 2 show that this does not lead necessarily to a better partition. Indeed, after 100 iterations, *SA* minimizing $\mathcal{E}$ provides the best partitions in terms of VC, CC and BAL, while STD is slightly worse. After 1000 iterations, *SA* has further improved VC and CC at the expenses of the balance metrics like BAL and STD.

Figures 2 and 3 show similar results comparing the final metrics VC and CC for an even larger range of initial temperatures.

While for *email-Enron* we have shown how *SA* can improve communication metrics at the expenses of balance metrics, we show that the opposite result

[5]Note that after a given iteration the energy can increase both for *JA-BE-JA-VC* and *SA*, but the figure shows that every L iterations, the energy always decrease.

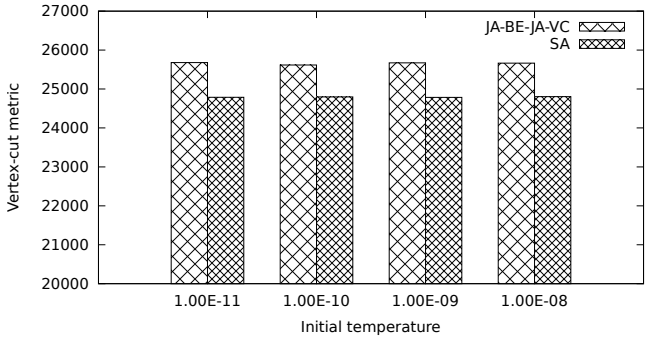


Figure 2: Vertex-cut metric for *JA-BE-JA-VC* and *SA* using *email-Enron* graph (1000 iterations were performed)

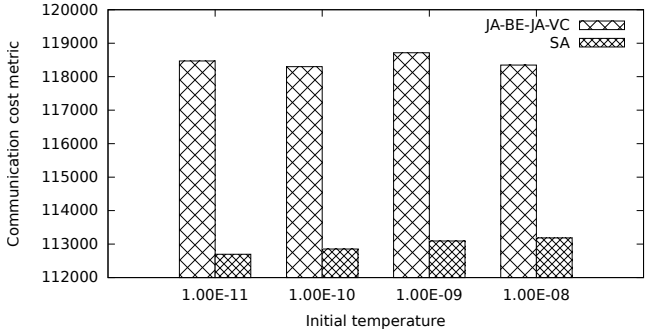


Figure 3: *Communication cost* metric for *JA-BE-JA-VC* and *SA* using *email-Enron* graph (1000 iterations were performed)

can be obtained on a different graph (*com-Amazon*) in Figs. 4 and 5. Moreover, for a given graph, it is possible to tune the relative importance of the different metrics by varying the parameter α.

6 Conclusions

In this paper, we have proposed a framework for distributed edge partitioning based on simulated annealing and originally inspired by *JA-BE-JA-VC*. Our framework is more general because it can be used to optimize a large family of partitioning metrics and convergence is guaranteed as long as the temperature is decreased slowly enough. We have discussed how our approach can be implemented in distributed computation frameworks like GraphX. Our preliminary results have revealed potential improvements in comparison to *JA-BE-JA-VC*, but have also shown that further research is required to identify energy functions that are good indicators of the quality of a partition. Moreover, the experiments have shown that a completely unbiased edge selection procedure (where any pair

Table 1: Shows final partitioning metrics obtained by partitioners (I - *JA-BE-JA-VC* partitioner; II - *SA* using only E_{comm} as energy function, III - *SA* using $E_{comm} + 0.5E_{bal}$ as energy function). Temperature decreases linearly from T_0 till 0.0 by 100 iterations.

	T_0	Final $\mathcal{E}$	Vertex-cut	Comm. cost	Balance	STD
I	5.00E-11	0.888788	25091	122477	1.0091	0.0055
	1.00E-10	0.888638	25096	122572	1.0091	0.0055
	5.00E-10	0.888634	25085	122350	1.0091	0.0055
II	5.00E-11	0.900026	25046	120083	1.0091	0.0055
	1.00E-10	0.900027	25046	120087	1.0091	0.0055
	5.00E-10	0.900006	25046	120080	1.0091	0.0055
III	5.00E-11	0.894742	25044	120768	1.0088	0.0058
	1.00E-10	0.894692	25043	120751	1.0079	0.0058
	5.00E-10	0.894617	25043	120782	1.0084	0.0061

Table 2: Shows final partitioning metrics obtained by partitioners (I - *JA-BE-JA-VC* partitioner; II - *SA* using only E_{comm} as energy function, III - *SA* using $E_{comm} + 0.5E_{bal}$ as energy function). Temperature decreases linearly from T_0 till 0.0 by 1000 iterations.

	T_0	Final $\mathcal{E}$	Vertex-cut	Comm. cost	Balance	STD
I	5.00E-11	0.8197	25685	118453	1.0091	0.0055
	1.00E-10	0.8202	25656	118588	1.0091	0.0055
	5.00E-10	0.8193	25603	118455	1.0091	0.0055
II	5.00E-11	0.8989	25048	120648	1.0091	0.0055
	1.00E-10	0.8990	25049	120634	1.0091	0.0055
	5.00E-10	0.8989	25046	120664	1.0091	0.0055
III	5.00E-11	0.8466	24795	113008	1.0144	0.0114
	1.00E-10	0.8467	24794	112951	1.0178	0.0099
	5.00E-10	0.8466	24771	112946	1.0332	0.0211

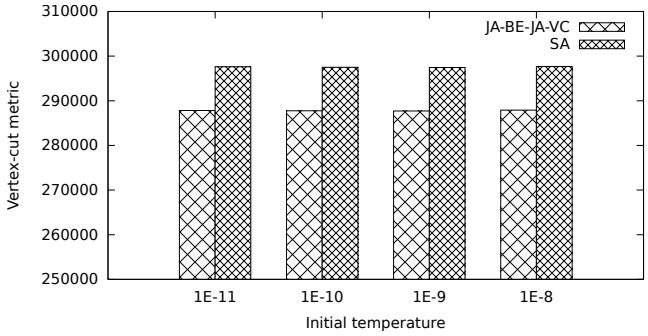


Figure 4: *Vertex-cut* metric value for *JA-BE-JA-VC* and *SA* using *com-Amazon* graph (1000 iterations were performed)

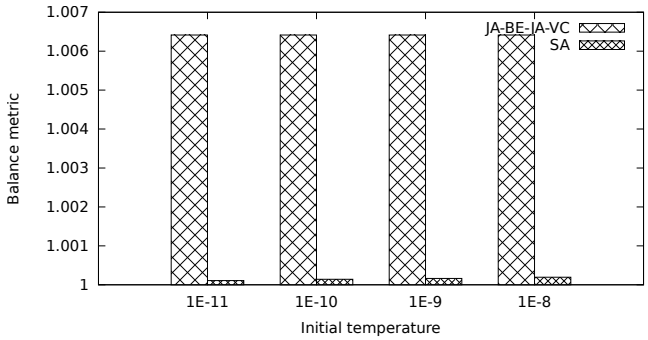


Figure 5: *Balance* metric value for *JA-BE-JA-VC* and *SA* using *com-Amazon* graph (1000 iterations were performed)

is equally likely to be drawn) can be very inefficient and we plan to investigate how a more intelligent exploration phase can be introduced without losing the theoretical guarantees about convergence.

References

[1] Code to gather metric information. `https://github.com/Mykhailenko/scala-graphx-tw-g5k/`

[2] Source code of JA-BE-JA-VC and SA. `https://bitbucket.org/hlibmykhailenko/jabejavc/`

[3] Stanford Large Network Dataset Collection. `https://snap.stanford.edu/data/`

[4] Bourse, F., Lelarge, M., Vojnovic, M.: Balanced graph edge partition. In: Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 1456–1465. ACM (2014)

[5] Brémaud, P.: Markov chains: Gibbs fields, Monte Carlo simulation, and queues, vol. 31. Springer Science & Business Media (2013)

[6] Gonzalez, J.E., Low, Y., Gu, H., Bickson, D., Guestrin, C.: Powergraph: Distributed graph-parallel computation on natural graphs. In: Presented as part of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12). pp. 17–30 (2012)

[7] Guerrieri, A., Montresor, A.: DFEP: Distributed funding-based edge partitioning. In: European Conference on Parallel Processing. pp. 346–358. Springer (2015)

[8] Mykhailenko, H., Neglia, G., Huet, F.: Which Metrics for Vertex-Cut Partitioning? In: Proceedings of the 11th International Conference for Internet Technology and Secured Transactions (2016)

[9] Rahimian, F., Payberah, A.H., Girdzijauskas, S., Haridi, S.: Distributed Vertex-Cut Partitioning. In: 4th International Conference on Distributed Applications and Interoperable Systems (DAIS). vol. LNCS-8460, pp. 186–200. Berlin, Germany (2014)

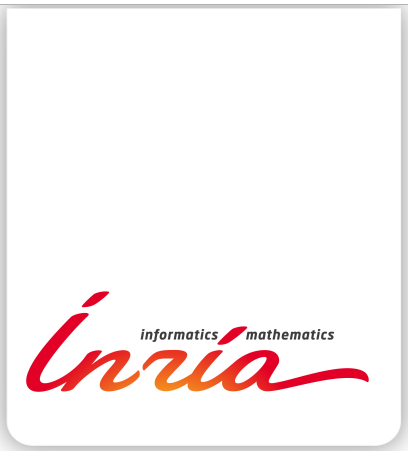
[10] Xin, R.S., Gonzalez, J.E., Franklin, M.J., Stoica, I.: Graphx: A resilient distributed graph system on spark. In: First International Workshop on Graph Data Management Experiences and Systems. p. 2. ACM (2013)

[11] Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., Franklin, M.J., Shenker, S., Stoica, I.: Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In: Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation (2012)

[12] Zaharia, M., Chowdhury, M., Franklin, M.J., Shenker, S., Stoica, I.: Spark: cluster computing with working sets. HotCloud 10, 10–10 (2010)

7 Acknowledgment

This work was partly funded by the French Government (National Research Agency, ANR) through the "Investments for the Future" Program reference #ANR-11-LABX-0031- 01.



RESEARCH CENTRE
SOPHIA ANTIPOLIS – MÉDITERRANÉE

2004 route des Lucioles - BP 93
06902 Sophia Antipolis Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399