A Distributed Algorithm For Large-Scale Graph Partitioning

Fatemeh Rahimian, KTH - Royal Institute of Technology and Swedish Institute of Computer Science (SICS), Sweden
Amir H. Payberah, Swedish Institute of Computer Science (SICS), Sweden
Sarunas Girdzijauskas, KTH - Royal Institute of Technology, Sweden
Mark Jelasity, MTA SZTE Research Group on AI, Szeged, Hungary
Seif Haridi, KTH - Royal Institute of Technology and Swedish Institute of Computer Science (SICS), Sweden

Balanced graph partitioning is an NP-complete problem with a wide range of applications. These applications include many large-scale distributed problems including the optimal storage of large sets of graph-structured data over several hosts. However, in very large-scale distributed scenarios, state-of-the-art algorithms are not directly applicable, because they typically involve frequent global operations over the entire graph. In this paper, we propose a fully distributed algorithm, called JA-BE-JA, that uses local search and simulated annealing techniques for two types of graph partitioning: *edge-cut* partitioning, and *vertex-cut* partitioning. The algorithm is massively parallel: there is no central coordination, each vertex is processed independently, and only the direct neighbors of a vertex, and a small subset of random vertices in the graph need to be known locally. Strict synchronization is not required. These features allow JA-BE-JA to be easily adapted to any distributed graph-processing system from data centers to fully distributed networks. We show that the minimal edge-cut value empirically achieved by JA-BE-JA is comparable to state-of-the-art centralized algorithms such as METIS. In particular, on large social networks JA-BE-JA outperforms METIS. We also show that JA-BE-JA computes very low vertex-cuts, which are proved significantly more effective than edge-cuts for processing most real-world graphs.

Categories and Subject Descriptors: C.2.0 [**Computer-Communication Networks**]: Network Protocols

General Terms: Design, Algorithms, Performance

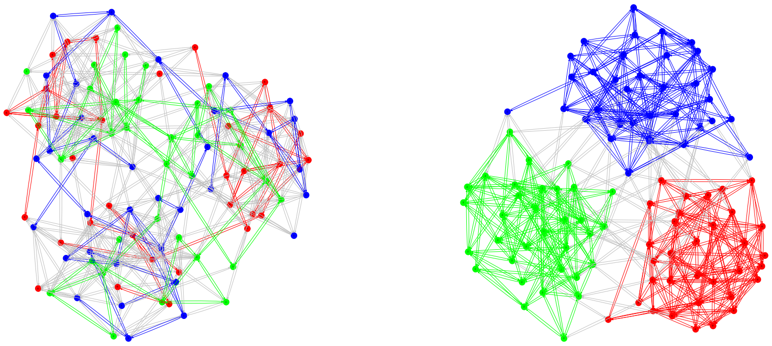
Additional Key Words and Phrases: graph partitioning, edge-cut partitioning, vertex-cut partitioning, distributed algorithm, load balancing, simulated annealing

ACM Reference Format:
Fatemeh Rahimian, Amir H. Payberah, Sarunas Girdzijauskas, Mark Jelasity, and Seif Haridi, 2014, A Distributed Algorithm For Large-Scale Graph Partitioning *ACM Trans. Autonom. Adapt. Syst.* 10, 2, Article 12 (June 2015), 24 pages.
DOI:http://dx.doi.org/10.1145/2714568

1. INTRODUCTION

A wide variety of real-world data can be naturally described as graphs. Take for instance communication networks, social networks, or biological networks. With the ever increasing size of such networks, it is crucial to exploit the natural connectedness of their data in order to store and process them efficiently. Hence, we are now observing an upsurge in the development of distributed and parallel graph processing tools and techniques. Since the size of the graphs (in terms of both vertices and edges) can grow very large, sometimes we have to partition them into multiple smaller clusters that can be processed efficiently in parallel. Unlike the conventional parallel data processing, parallel graph processing requires each vertex or edge to be processed in the context of its neighborhood. Therefore, it is important to maintain the locality of information while partitioning the graph across multiple (virtual) machines. It is also important to produce equal-size partitions that distribute the computational load evenly between clusters.

Finding good partitions is a well-known and well-studied problem in graph theory [Hendrickson and Leland 1995]. In its classical form, graph partitioning usually refers to *edge-cut* partitioning, that is, to divide vertices of a graph into disjoint clus-



(a) A poor edge-cut partitioning. Vertices are assigned to partitions at random, thus, there are many inter-partition links.

(b) A good edge-cut of the same graph, where vertices that are highly connected are assigned to the same partition.

Fig. 1: Illustration of graph partitioning. The color of each vertex represents the partition it belongs to. The colored links are connections between two vertices in the same partition. The gray links are inter-partition connections.

ters of nearly equal size, while the number of edges that span separated clusters is minimal. Figures 1(a) and 1(b) are examples of a poor and a good edge-cut partitioning of a graph, respectively. Note, if each partition in this graph represent, for instance, a server that stores and maintains data of the vertices it holds, then the inter-partition links are translated into communication overhead between the servers, which should be kept as small as possible. While a good edge-cut partitioning can reduce such communication overheads and also balance the number of vertices in each partition, there are some studies [Abou-Rjeili and Karypis 2006; Lang 2004; Leskovec et al. 2009] that show tools that utilize edge-cut partitioning do not achieve good performance on real-world graphs (which are mostly power-law graphs). This is mainly due to an unbalanced number of edges in each cluster combined with the fact that the complexity of most graph computations is influenced by the order of edges.

In contrast, both theory [Albert et al. 2000] and practice [Gonzalez et al. 2012; Xin et al. 2013] prove that power-law graphs (e.g., social networks or collaboration networks) can be efficiently processed in parallel if *vertex-cuts* are used. As opposed to edge-cut partitioning, a vertex-cut partitioning divides edges of a graph into equal size clusters. The vertices that hold the endpoints of an edge are also placed in the same cluster as the edge itself. However, the vertices are not unique across clusters and might have to be *replicated* (cut), due to the distribution of their edges across different clusters. A good vertex-cut is one that requires a minimum number of replicas. Figure 2 illustrates the difference between these two types of partitioning.

In this paper, we focus on processing extremely large-scale graphs, e.g., user relationship and interaction graphs from online social networking services such as Facebook or Twitter, resulting in graphs with billions of vertices and hundreds of billions of edges. The very large scale of the graphs we target poses a major challenge. Although numerous algorithms are known for graph partitioning [Enright et al. 2002; Karypis and Kumar 1999a; 1998; Kernighan and Lin 1970; Meyerhenke et al. 2008; Meyerhenke et al. 2009; Sanders and Schulz 2012; 2011], including parallel ones, most of the techniques involved assume a form of cheap random access to the entire graph. In contrast to this, large scale graphs do not fit into the main memory of a single computer, in fact,

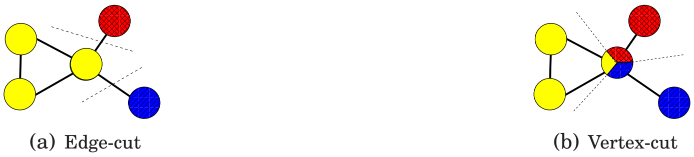


Fig. 2: Partitioning a graph into three clusters

they often do not fit on a single local file system either. Worse still, the graph can be fully distributed as well, with only very few vertices hosted on a single computer.

We provide a distributed balanced graph partitioning algorithm, called JA-BE-JA, both for edge-cut and vertex-cut partitioning. Choosing between edge-cut and vertex-partitioning depends on the application, and JA-BE-JA, to the best of our knowledge, is the only algorithm that can be applied in both models. JA-BE-JA is a decentralized local search algorithm and it does not require any global knowledge of the graph topology. That is, we do not have cheap access to the entire graph and we have to process it with only partial information. Each vertex of the graph is a processing unit, with local information about its neighboring vertices, and a small subset of random vertices in the graph, which it acquires by purely local interactions. Initially, every vertex/edge is assigned to a random partition, and over time vertices communicate and improve upon the initial assignment.

Our algorithm is uniquely designed to partition extremely large graphs. The algorithm achieves this through its locality, simplicity and lack of synchronization requirements, which enables it to be easily adapted to graph processing frameworks such as Pregel [Malewicz et al. 2010] or GraphLab [Low et al. 2012]. Furthermore, JA-BE-JA can be applied on fully distributed graphs, where each network node represents a single graph vertex.

To evaluate JA-BE-JA for edge-cut partitioning, we use multiple datasets of different characteristics, including a few synthetically generated graphs, some graphs that are well-known in the graph partitioning community [Walshaw 2012b], and some sampled graphs from Facebook [Viswanath et al. 2009] and Twitter [Galuba et al. 2010]. We first investigate the impact of different heuristics on the resulting partitioning of the input graphs, and then compare JA-BE-JA to METIS [Karypis and Kumar 1999a], a well-known centralized solution. We show that, although JA-BE-JA does not have cheap random access to the graph data, it can work as good as, and sometimes even better than, a centralized solution. In particular, for large graphs that represent real-world social network structures, such as Facebook and Twitter, JA-BE-JA outperforms METIS [Karypis and Kumar 1999a].

For vertex-cut partitioning, we will compare our solution with [Guerrieri and Montresor 2014], and show that JA-BE-JA not only guarantees to keep the size of the partitions balanced, but also produces a better vertex-cut.

In the next section we define the exact problems that we are targeting, together with the boundary requirements of the potential applications. Then, in Section 3 we explain JA-BE-JA in detail, and evaluate it in Section 4. In Section 5 we study the related work of graph partitioning. Finally, in Section 6 we conclude the work.

2. PROBLEM STATEMENT

The problem that we address in this paper is *distributed balanced k-way* graph partitioning. In this section we define two variations of this problem, namely *edge-cut* and *vertex-cut* partitioning. We also formulate the optimization problem and describe our assumptions about the system we operate in.

2.1. Balanced Edge-cut Partitioning

Given an undirected graph $G = (V, E)$, where V is the set of vertices and E is the set of edges, a k-way edge-cut partitioning divides V into k subsets. Intuitively, in a good partitioning the number of edges that cross the boundaries of components is minimized. *Balanced* (uniform) partitioning refers to the problem of partitioning a graph into equal-sized components with respect to the number of vertices in each component. The equal size constraint can be softened by requiring that the partition sizes differ only by a factor of a small ϵ.

A k-way edge-cut partitioning can be given with the help of a partition function $\pi : V \rightarrow \{1, \ldots, k\}$ that assigns a *color* to each vertex. Hence, $\pi(p)$, or π_p for short, refers to the color of vertex p. Vertices with the same color form a partition. We denote the set of neighbors of vertex p by N_p, and define $N_p(c)$ as the set of neighbors of p that have color c:

$$N_p(c) = \{q \in N_p : \pi_q = c\} \tag{1}$$

The number of neighbors of vertex p is denoted by $d_p = |N_p|$, and $d_p(c) = |N_p(c)|$ is the number of neighbors of p with color c. We define the *energy* of the system as the number of edges between vertices with different colors (equivalent to edge-cut). Accordingly, the energy of a vertex is the number of its neighbors with a different color, and the energy of the graph is the sum of the energy of the vertices:

$$E(G, \pi) = \frac{1}{2} \sum_{p \in V} \left(d_p - d_p(\pi_p)\right), \tag{2}$$

where we divide the sum by two since the sum counts each edge twice. Now we can formulate the balanced optimization problem: find the optimal partitioning π^* such that

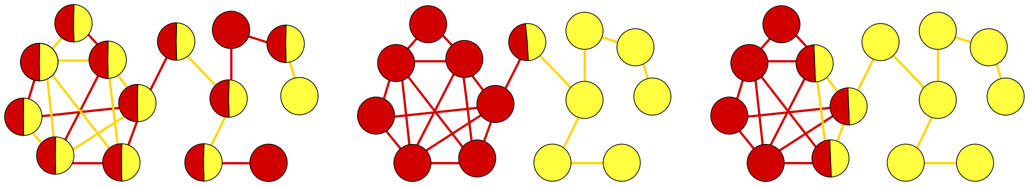
$$\begin{gathered} \pi^* = \arg\min_\pi E(G, \pi) \\ s.t. \;\; |V(c_1)| = |V(c_2)|, \forall\, c_1, c_2 \in \{1, \ldots, k\} \end{gathered} \tag{3}$$

where $V(c)$ is the set of vertices with color c.

2.2. Balanced Vertex-cut Partitioning

Given an undirected graph $G = (V, E)$, where V is the set of vertices and E is the set of edges, a k-way balanced vertex-cut partitioning divides the set of edges E into k subsets of equal size. Each partition also has a subset of vertices that hold at least one of the edges in that partition. However, vertices are not unique across partitions, that is, some vertices have to be replicated in more than one partition, due to the distribution of their edges across several partitions. A good vertex-cut partitioning strives to minimize the number of replicated vertices. Figure 3 shows a graph with three different vertex-cut partitionings. The graph edges are partitioned into two clusters. Two colors, yellow and red, are representing these two partitions. Vertices that have edges of one color only, are also colored accordingly, and the vertices that have to be replicated are cut. A very naïve solution is to randomly assign edges to partitions. As shown in Figure 3(a), in a random assignment, nearly all the vertices have edges of different colors, thus, they have to be replicated in both partitions. Figure 3(b) illustrates what happens if we use an edge-cut partitioner, and then randomly assign the cut edges to one of the partitions. As shown, the vertex-cut improves significantly. However, the number of edges in the partitions is very unbalanced. What we desire is depicted in Figure 3(c), where the the number of replicated vertices is kept as low as possible, while the size of the partitions, with respect to the number of edges, is balanced.

A k-way balanced vertex-cut partitioning can be given with the help of a partition function $\pi : E \rightarrow \{1, \ldots, k\}$ that assigns a *color* to each edge. Hence, $\pi(e)$, or π_e for



(a) A random vertex-cut. Edges are evenly distributed, but nearly all the vertices have to be replicated.

(b) Partitioning based on edge-cut and then assigning the cut edges randomly to one partition. Only one vertex is replicated, but edges are not evenly distributed.

(c) A good vertex-cut partitioning. Three vertices are replicated and the number of edges in the two partitions is balanced.

Fig. 3: Vertex-cut partitioning into two clusters. The color of each edge/vertex represents the partition it belongs to. The cut vertices belong to both partitions.

short, refers to the color of edge e. Edges with the same color form a partition. We denote the set of edges that are connected (or incident) to vertex p by E_p. Accordingly, $E_p(c)$ indicates the subset of edges incident with p that have color c:

$$E_p(c) = \{e \in E_p : \pi_e = c\} \tag{4}$$

We refer to $|E_p(c)|$ as the *cardinality* of color c at vertex p. Then, the *energy* of a vertex p, denoted by $\gamma(p, \pi)$, is defined as the number of different colors assigned to the edges incident with p, i.e., the number of colors with $|E_p(c)|$ greater than zero.

$$\gamma(p, \pi) = \sum_{|E_p(c)|>0} 1, \forall\, c \in \{1, \dots, k\} \tag{5}$$

In other words, the energy of a vertex is equivalent to the number of required replicas for that vertex (i.e., the number of times the vertex has to be cut). The energy of the graph is then the sum of the energy of all its vertices:

$$\Gamma(G, \pi) = \sum_{p \in V} \gamma(p, \pi) \tag{6}$$

Now we can formulate an optimization problem as follows: find the optimal partitioning π^* such that:

$$\begin{gathered} \pi^* = \arg\min\nolimits_\pi \Gamma(G, \pi) \\ s.t.\ \ |E(c_1)| = |E(c_2)|, \forall\, c_1, c_2 \in \{1, \dots, k\} \end{gathered} \tag{7}$$

where $|E(c)|$ is the number of edges with color c.

2.3. Data Distribution Model

We assume that the vertices of the graph are processed periodically and asynchronously, where each vertex only has access to the state of its immediate neighbors and a small set of random vertices in the graph. The vertices could be placed either on an independent host each, or processed in separate threads in a distributed framework. This model, which we refer to as the *one-host-one-node* model, is appropriate for frameworks like GraphLab [Low et al. 2012] or Pregel [Malewicz et al. 2010], Google's distributed framework for processing very large graphs. It can also be used in peer-to-peer overlays, where each vertex is an independent computer. In both cases, no shared memory is required. Vertices communicate only through messages over edges of the graph, and each message adds to the communication overhead.

The algorithm can take advantage of the case when a computer hosts more than one graph vertex. We call this the *one-host-multiple-nodes* model. Here, vertices on the same host can benefit from a shared memory on that host. For example, if a vertex exchanges some information with other vertices on the same host, the communication cost is negligible. However, information exchange across hosts is costly and constitutes the main body of the communication overhead. This model is interesting for data centers or cloud computing environments, where each computer can emulate thousands of vertices at the same time.

3. SOLUTION

We propose JA-BE-JA[1], a distributed heuristic algorithm for the balanced k-way graph partitioning problem. We use different colors to identify distinct partitions. The colors are assigned to either vertices or edges, for edge-cut and vertex-cut partitioning, respectively. We use the term color exchange in both cases, which means the exchange of colors between vertices for edge-cut partitioning, and between edges for vertex-cut partitioning. However, in both cases it is always the vertices that act as processing units and run the algorithm, and edges are only treated as passive elements.

3.1. The Basic Idea

The basic idea is to assign colors uniformly at random and then to apply a local search to push the configuration towards lower energy states (min-cut).

The local search operator is executed by all the graph vertices in parallel: each vertex attempts to change either its own color (in edge-cut partitioning), or the color of one of the edges that is connected to it (in vertex-cut partitioning), to the most dominant color in the neighborhood. However, in order to preserve the size of the partitions, the colors cannot change independently. Instead, colors can only be *swapped*. Each vertex iteratively selects another vertex among either its *neighbors* or a *random sample*, and investigates the pair-wise utility of a color exchange. If the color exchange decreases the energy, then the two vertices proceed with the color exchange. Otherwise, they preserve their colors.

To implement this idea, JA-BE-JA combines two main components: (i) a *sampling component* that enables a vertex to choose other vertices for color exchange, and (ii) a *swapping component* that indicates if the color swap should happen. The sampling component is identical in both edge-cut and vertex-cut partitioning, whereas the swapping components are different due to inherent differences in their objective functions. We will explain these two components in the following sections.

Before delving into the details, however, it is important to note that when applying local search, the key problem is to ensure that the algorithm does not get stuck in a local optimum. For this purpose, we employ the *simulated annealing* technique [Van Laarhoven and Aarts 1987; Talbi 2009] as we will describe below. Later, in the evaluation section (Section 4), we show the impact of this technique on the quality of the final partitioning.

Note, since no color is added to/removed from the graph, the distribution of colors is preserved during the course of optimization. Hence, if the initial random coloring of the graph is uniform, we will have balanced partitions at each step. We stress that this is a heuristic algorithm, so it cannot be proven (or, in fact, expected) that the globally minimal energy value is achieved. Exact algorithms are not feasible since the problem is NP-complete [Andreev and Räcke 2004], so the min-cut cannot be computed in a reasonable time, even with a centralized solution and a complete knowledge of

[1] JA-BE-JA means “swap” in Persian.

the graph. In Section 4.1.5, however, we compare our results with the best known partitioning solutions over a number of benchmark problem instances.

3.2. Sampling Component

In both edge-cut partitioning and vertex-cut partitioning, a vertex should first select a set of candidate vertices for potential color exchanges. We consider three possible ways of selecting the candidate set:

- *Local* (L): every vertex considers its directly connected vertices (*neighbors*) as candidates for color exchange.
- *Random* (R): every vertex selects a uniform random sample of the vertices in the graph. Note that there exist multiple techniques for taking a uniform sample of a given graph at a low cost [Awan et al. 2006; Dowling and Payberah 2012; Jelasity et al. 2005; Massoulié et al. 2006; Payberah et al. 2011; Voulgaris et al. 2005].
- *Hybrid* (H): in this policy first the immediate neighbor vertices are selected (i.e., the local policy). If this selection fails to improve the pair-wise utility, the vertex is given another chance for improvement, by letting it select vertices from its random sample (i.e., the random policy).

We will show later in Section 4.1.2 that the hybrid policy performs better than the other two in most cases, thus, it is considered as the prime policy in the sampling component.

3.3. Swapping Component: Edge-cut Partitioning

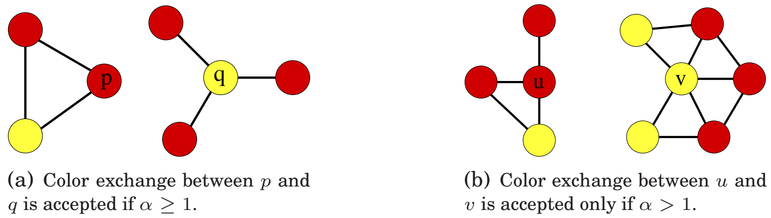
After finding a set of candidates for a color exchange, a vertex selects the best one from the set as the swap *partner*. To decide if two vertices should exchange their colors, we require: (i) a function to measure the pairwise utility of a color exchange, and (ii) a policy for escaping local optima. The utility function should be such that it reduces the energy of the graph, thus, it is different for edge-cut and vertex-cut partitioning. In both cases, however, the vertex that maximizes the utility function is selected from the candidate set.

In order to minimize the edge-cut of the partitioning, we try to maximize $d_p(\pi_p)$ for all vertices p in the graph, which only requires local information at each vertex. Two vertices p and q with colors π_p and π_q, respectively, exchange their colors only if this exchange decreases their energy (increases the number of neighbors with a similar color to that of the vertex):

$$d_p(\pi_q)^\alpha + d_q(\pi_p)^\alpha > d_p(\pi_p)^\alpha + d_q(\pi_q)^\alpha \tag{8}$$

where α is a parameter of the energy function, which takes on real values greater than or equal to 1. If $\alpha = 1$, a color exchange is accepted if it increases the total number of edges with the same color at two ends. For example, color exchange for vertices p and q in Figure 4(a) is accepted, as the vertices change from a state with 1 and 0 neighbors of a similar color, to 1 and 3 such neighbors, respectively. However, vertices u and v in Figure 4(b), each in a state with 2 neighbors of a similar color, do not exchange their colors, if $\alpha = 1$, because $1 + 3 \not> 2 + 2$. However, if $\alpha > 1$, then vertices u and v will exchange their colors. Although this exchange does not directly reduce the total size of the edge-cut of the graph, it increases the probability of future color exchanges for the two yellow vertices, currently in the neighborhood of vertex v. In section 4 we evaluate the effect of the parameter α. Based on this relation, we can define the utility of the swap as:

$$U = [d_p(\pi_q)^\alpha + d_q(\pi_p)^\alpha] - [d_p(\pi_p)^\alpha + d_q(\pi_q)^\alpha] \tag{9}$$



(a) Color exchange between p and q is accepted if $\alpha \geq 1$.

(b) Color exchange between u and v is accepted only if $\alpha > 1$.

Fig. 4: Examples of two potential color exchanges.

The swap will take place if the utility is greater than zero. In Section 4.1.3 we will discuss about the appropriate value for α.

To avoid becoming stuck in a local optimum, we use the well-known simulated annealing technique [Van Laarhoven and Aarts 1987; Talbi 2009]. We introduce a *temperature* ($T \in [1, T_0]$), which starts at T_0 and is decreased over time, similar to the cooling process in [Van Laarhoven and Aarts 1987; Talbi 2009]. The updated utility functions becomes:

$$U = [d_p(\pi_q)^\alpha + d_q(\pi_p)^\alpha] \times T_r - [d_p(\pi_p)^\alpha + d_q(\pi_q)^\alpha] \quad (10)$$

As a result, in the beginning we might move in a direction that degrades the energy function, i.e., vertices exchange their color even if the edge-cut is increased. Over time, however, we take more conservative steps and do not allow those exchanges that result in a higher edge-cut. The two parameters of the simulated annealing process are (i) T_0, the initial temperature, which is greater than or equal to one, and (ii) δ, that determines the speed of the cooling process. The temperature in round r is calculated as $T_r = max\{1, T_{r-1} - \delta\}$. When the temperature reaches the lower bound 1, it is not decreased anymore. From then on, the decision procedure falls back on using Equation (8). Algorithms 1, 2, and 3 show the edge-cut partitioning process.

3.4. Swapping Component: Vertex-cut Partitioning

The swapping component in vertex-cut partitioning and edge-cut partitioning are similar, but their difference is in the utility function calculation. The main idea of this heuristic is to check whether exchanging the color of two edges decreases the energy of their incident vertices or not. If it does, the two edges swap their colors, otherwise they keep them.

To every edge e (with two endpoints p and q) we assign a value υ, with respect to color c, that indicates the relative number of neighboring edges of e with color c. That is:

$$\upsilon(e,\ c) = \begin{cases} \frac{|E_p(c)|-1}{|E_p|} + \frac{|E_q(c)|-1}{|E_q|} & if\ c = \pi_e \\ \frac{|E_p(c)|}{|E_p|} + \frac{|E_q(c)|}{|E_q|} & otherwise \end{cases} \quad (11)$$

Note that in the first case, $E_p(c)$ and $E_q(c)$ include edge e, and that is why we need to decrement them by one.

First of all, a vertex selects one of its edges for color exchange. A naïve policy for edge selection is random selection, but, as explained in [Rahimian et al. 2014], this policy will not lead our local search in the right direction. Therefore, we consider a more effective policy, i.e., greedy policy for edge selection. With this policy, a vertex selects one of its edges, e.g., e, which has a color with the minimum cardinality:

$$e \in E_p(c^*), \quad c^* = \arg\min_c |E_p(c)| \quad (12)$$

Algorithm 1 Edge/Vertex Cut

```
procedure $cut(graph, policy)$
    // policy identifies the partitioning algorithm, i.g., edge-cut or vertex-cut;
    $bestPartner \leftarrow getBestPartner($self$.getNeighbours(), policy, T_r);$
    if $(bestPartner = null)$ then
        $bestPartner \leftarrow getBestPartner(graph.getRandomVertices(), policy, T_r);$
    if $(bestPartner \neq null)$ then
        if $(policy = EdgeCut)$ then
            $swapVertexColor($self$, bestPartner);$
        else
            $swapEdgeColor($self$, bestPartner);$
    $T_r \leftarrow T_r - \delta;$
    if $(T_r < 1)$ then
        $T_r \leftarrow 1;$
```

Algorithm 2 Select Best Partner

```
procedure $getBestPartner(candidates, policy, T_r)$
    $highestUtility \leftarrow 0;$
    $bestPartner \leftarrow null;$
    forall $(partner \in candidates)$ do
        if $(policy = EdgeCut)$ then
            $utility \leftarrow swapVertexUtility($self$, partner, T_r);$
        else
            $utility \leftarrow swapEdgeUtility($self$, partner, T_r);$
        if $((utility > 0) \wedge (utility > higestUtility))$ then
            $bestPartner \leftarrow partner;$
            $highestUtility \leftarrow utility;$
    return $bestPartner;$
```

Algorithm 3 Calculate Vertex Utility

```
procedure $swapVertexUtility(vertex1, vertex2, T_r)$
    $c1 \leftarrow vertex1.getColor();$
    $c2 \leftarrow vertex2.getColor();$
    $u1c1 \leftarrow getVertexValue(vertex1, c1);$
    $u2c2 \leftarrow getVertexValue(vertex2, c2);$
    $u1c2 \leftarrow getVertexValue(vertex1, c2);$
    $u2c1 \leftarrow getVertexValue(vertex2, c1);$
    return $((u1c2^{\alpha} + u2c1^{\alpha}) \times T_r) - (u1c1^{\alpha} + u2c2^{\alpha});$
```

Next, the objective is to maximize the overall value of edges during the color exchange process. More precisely, vertex p exchanges the color of its edge e with the color of another edge e' owned by node p', if and only if:

$$\upsilon(e, c') + \upsilon(e', c) > \upsilon(e, c) + \upsilon(e', c'), \tag{13}$$

where $c = \pi_e$ and $c' = \pi'_e$. Accordingly we can define the utility function as:

$$U = [\upsilon(e, c') + \upsilon(e', c)] - [\upsilon(e, c) + \upsilon(e', c')]. \tag{14}$$

Similar to edge-cut partitioning, we use the simulated annealing technique [Talbi 2009] to prevent getting stuck in a local optimum. Therefore, as shown in Algorithm 4 the updated utility function becomes:

$$U = [\upsilon(e, c') + \upsilon(e', c)] \times T_r - [\upsilon(e, c) + \upsilon(e', c')] \tag{15}$$

3.5. Ja-Be-Ja

Algorithm 1 presents the core of JA-BE-JA, which is run periodically by all vertices of a graph. As it shows, we use the hybrid heuristic for vertex selection, which first tries

Algorithm 4 Calculate Edge Utility

```
procedure $swapEdgeUtility(vertex1, vertex2, T_r)$
    $c1 \leftarrow vertex1.getColorWithMinCardinality();$
    $c2 \leftarrow vertex2.getColorWithMinCardinality();$
    $edg1 \leftarrow vertex1.getEdges(c1).getOneRandom();$
    $edg2 \leftarrow vertex2.getEdges(c2).getOneRandom();$
    $u1c1 \leftarrow getEdgeValue(edge1.src, edge1.dest, c1);$
    $u2c2 \leftarrow getEdgeValue(edge2.src, edge2.dest, c2);$
    $u1c2 \leftarrow getEdgeValue(edge1.src, edge1.dest, c2);$
    $u2c1 \leftarrow getEdgeValue(edge2.src, edge2.dest, c1);$
    return $((u1c2 + u2c1) \times T_r) - (u1c1 + u2c2);$
```

the local policy, and if it fails it follows the random policy. Algorithms 3 and 4 show how we calculate the two sides of Equations (10) and (15) for edge-cut and vertex-cut partitioning, respectively. Also, the current temperature, T_r, biases the comparison towards selecting new states (in the initial rounds).

Note that the actual swapping operation is implemented as an optimistic transaction, the details of which are not included in the algorithm listing to avoid distraction from the core algorithm. The actual swap is done after the two vertices perform a handshake and agree on the swap. This is necessary, because the deciding vertex might have outdated information about the partner vertex. During the handshake, the initiating vertex sends a swap request to the partner vertex, along with all the information that the partner vertex needs to verify the swap utility. For example, in case of edge-cut partitioning this information includes the current color (π_p), the partner's color ($\pi_{partner}$), the number of neighbors with the same color ($d_p(\pi_p)$), and the number of neighbors with the color of the partner vertex ($d_p(\pi_{partner})$). If the verification succeeds, the partner vertex replies with an acknowledgment (ACK) message and the swap takes place. Otherwise, a negative acknowledgment message (NACK) is sent and the existing color of the two vertices or edges will be preserved. These sample and swap processes are periodically repeated by all the vertices, in parallel, and when no more swaps take place in the graph, the algorithm has converged.

We also use a multi-start search [Talbi 2009], by running the algorithm many times, starting from different initial states. Note that this technique is applied in a distributed way. More precisely, after each run, vertices use a gossip-based aggregation method [Jelasity et al. 2005] to calculate the edge-cut (vertex-cut) in the graph. If the new edge-cut (vertex-cut) is smaller than the previous one, they update the best solution found so far by storing the new edge-cut (vertex-cut) value together with the current local color.

In the one-host-multiple-nodes model, the only change required is to give preference to local host swaps when selecting the best partner, i.e., in Algorithm 2. That is, if there are several vertices as potential partners for a swap, the vertex selects the one that is located on the local host, if there is such a candidate. Note that in this model not each and every vertex requires to maintain a random view for itself. Instead, the host can maintain a large enough sample of the graph to be used as a source of samples for all hosted vertices. In Section 4.1.4, we study the trade-off between communication overhead and the edge-cut with and without considering the locality.

3.6. Generalizations of Ja-Be-Ja

So far, we have discussed the case when the graph edges are not weighted and the partition sizes are equal. However, JA-BE-JA is not limited to these cases. In this section, we briefly describe how it can deal with weighted graphs and produce arbitrary pre-defined partition sizes.

Weighted graphs. In real world applications, vertices and/or edges are often weighted. For example, in a graph database some operations are performed more frequently, thus, some edges are accessed more often [Dominguez-Sal et al. 2010]. In order to prioritize such edges for edge-cut partitioning of a graph, we change the definition of d_p, such that, instead of just counting the number of neighboring vertices with the same color, we sum the weights of these edges:

$$d_p(c) = \sum_{q \in N_p(c)} w(p, q) \tag{16}$$

where $w(p, q)$ is the weight of the edge between p and q.

A similar approach can be taken, in order to enable vertex-cut partitioning for graphs with weighted vertices.

Arbitrary partition sizes. Assume we want to split the data over two machines that are not equally powerful. If the first machine has twice as much resources than the second one, we need a 2-way partitioning with one component being twice as large as the other. To do that, we can initialize the graph partitioning with a biased distribution of colors. For example, for edge-cut partitioning, if vertices initially choose randomly between two partitions c_1 and c_2, such that c_1 is twice as likely to be chosen, then the final partitioning will have a partition c_1, which is twice as big. This is true for any distribution of interest, as JA-BE-JA is guaranteed to preserve the initial distribution of colors. Likewise, for vertex-cut partitioning, any given distribution for the edge colors can be used for initialization, and from then on this distribution will remain an invariant.

4. EXPERIMENTAL EVALUATION

We have implemented JA-BE-JA on PEERSIM [Montresor and Jelasity 2009], a discrete event simulator for building P2P protocols. We used multiple graphs of different nature and size for evaluating JA-BE-JA. In particular, we have considered four types of graphs: (i) two synthetically generated graphs, (ii) several graphs from the Walshaw archive [Walshaw 2012b], (iii) sampled graphs from two well-known social networks: Twitter [Galuba et al. 2010] and Facebook [Viswanath et al. 2009], and (iv) two collaboration networks from the Stanford snap dataset [Leskovec 2011]. These graphs and some of their properties are listed in Table I.

Synthetic Graphs. We generated two different graphs synthetically. The first one is based on the Watts-Strogatz model [Watts and Strogatz 1998], with 1000 vertices and average degree 8 per vertex. First, a lattice is constructed and then some edges are rewired with probability 0.02. We refer to this graph as *Synth-WS*. The second graph, *Synth-SF*, is an implementation of the Barabási-Albert model [Albert and Barabási 2002] of growing scale-free networks. This graph also includes 1000 vertices with an average degree of 16. Both graphs are undirected and there are no parallel edges either.

The Walshaw Archive. The Walshaw archive [Walshaw 2012b] consists of the best partitioning found to date for a set of graphs, and reports the partitioning algorithms that achieved those best results. This archive, which has been active since the year 2000, includes the results from most of the major graph partitioning software packages, and is kept updated regularly by receiving new results from the researchers in this field. For our experiments, we have chosen graphs add20, data, 3elt, 4elt, and vibrobox, which are the small and medium size graphs in the archive, listed in Table I.

Table I: Datasets.

Dataset	|V|	|E|	Type	Power-law	Reference
Synth-WS	1000	4147	Synth.	No	-
Synth-SF	1000	7936	Synth.	No	-
add20	2395	7462	Walshaw	No	[Walshaw 2012b]
data	2851	15093	Walshaw	No	[Walshaw 2012b]
3elt	4720	13722	Walshaw	No	[Walshaw 2012b]
4elt	15606	45878	Walshaw	No	[Walshaw 2012b]
vibrobox	12328	165250	Walshaw	No	[Walshaw 2012b]
Twitter	2731	164629	Social	Yes	[Galuba et al. 2010]
Facebook	63731	817090	Social	Yes	[Viswanath et al. 2009]
Astroph	17903	196972	Collaboration	Yes	[Leskovec 2011]
Email-Enron	36692	367662	Collaboration	Yes	[Leskovec 2011]

The Social Network Graphs. Since social network graphs are one of the main targets of our partitioning algorithm, we investigate the performance of JA-BE-JA on two sampled datasets, which represent the social network graphs of Twitter and Facebook. We sampled our Twitter graph from the follower network of 2.4 million Twitter users [Galuba et al. 2010]. There are several known approaches for producing an unbiased sample of a very large social network, such that the sample has similar graph properties to those of the original graph. We used an approach discussed in [Kurant et al. 2010] sampling nearly 10000 vertices by performing multiple breadth first searches (BFS). We also used a sample graph of Facebook, which is made available by Viswanath et al. [Viswanath et al. 2009]. This data is collected by crawling the New Orleans regional network during December 29th, 2008 and January 3rd, 2009, and includes those users who had a publicly accessible profile in the network. The data, however, is anonymized.

The Collaboration Network Graphs. Two graphs among our input dataset fall into this category: Astroph and Email-Enron. We have selected these two graphs only for the sake of comparison with the state-of-the-art work on vertex-cut partitioning [Guerrieri and Montresor 2014]. Therefore, the result for edge-cut partitioning of these graphs is not reported.

We organize the rest of this section in two main parts, one for edge-cut partitioning and the other for vertex-cut partitioning.

4.1. Edge-cut Partitioning

First, we investigate the impact of different heuristics and parameters on different types of graphs. Then, we conduct an extensive experimental evaluation to compare the performance of JA-BE-JA to (i) METIS [Karypis and Kumar 1999a], a well-known efficient centralized solution, and (ii) the best known available results from the Walshaw benchmark [Walshaw 2012b] for several graphs. Unless stated otherwise, we compute a 4-way partitioning of the input graph with initial temperature $T_0 = 2$, the temperature is reduced by $\delta = 0.003$ in each step until it reaches value 1. However, the algorithm will continue to run until there are no changes/swaps anymore. The parameter α is set to 2. In Section 4.1.3 we will show how we have come to select these values for these parameters.

4.1.1. Metrics. Although the most important metric for edge-cut graph partitioning is the size of the edge-cut (or energy), there are a number of studies [Hendrickson 1998] that show that this metric alone is not enough to measure the partitioning quality. Several metrics are, therefore, defined and used in the literature [Meyerhenke et al. 2008; Meyerhenke et al. 2009], among which we selected the following ones in our evaluations:

— *edge-cut*: the number of inter-partition edges, as given in Formula 2, i.e., $E(G, \pi)$.
— *swaps*: the number of swaps that take place *between different hosts* during run-time (that is, swaps between graph vertices stored on the same host are not counted).
— *data migration*: the number of vertices that need to be migrated from their initial partition to their final partition.

While the size of the edge-cut is a quality metric for partitioning, the number of swaps defines the cost of the algorithm. Moreover, the data migration metric makes sense only in the one-host-multiple-nodes model, where some graph vertices have to migrate from one physical machine to another after finding the final partitioning. If the graph vertices that are initially located at a given host get the same initial color, then this metric is given by the number of vertices that end up with a different color by the time the algorithm has converged.

In all the experiments that follow, we have executed the algorithm 10 times for each graph. The only exception is the Facebook graph, for which we only ran the experiments 3 times. The value reported for the edge-cut in all the tables and plots is the minimum edge-cut among different runs. For the number of swaps and migrations, we report the ones associated with the reported (minimum) edge-cut. Note that in all cases, we could also report the average and the standard deviation or variance of the edge-cut across different runs. But we observed a similar trend for the average and minimum when it came to tuning the parameters and drawing conclusions. Therefore, to be consistent with the related work, and at the same time not to overwhelm readers with numbers, we report the minimum edge-cut only. For the sake of completion however, we report the average edge-cut and the standard deviation of different runs in Section 4.1.5, where we compare JA-BE-JA with the state-of-the-art.

4.1.2. The impact of the sampling policies. In this section, we study the effect of different sampling heuristics on the edge-cut. These heuristics were introduced in Section 3.2 and are denoted by L, R, and H. Here, we evaluate the one-node-one-host model, and to take uniform random samples of the graph we applied Newscast [Jelasity et al. 2005; Tölgyesi and Jelasity 2009] in our implementation. As shown in Table II, all heuristics significantly reduce the initial edge-cut that belongs to a random partitioning. Even with heuristic L, which only requires the information about direct neighbors of each vertex, the edge-cut is reduced to 30% of the initial number for the Facebook graph. The random selection policy (R) works even better than local (L) for all the graphs, as it is less likely to get stuck in a local optimum. The best result for most graphs, however, is achieved with the combination of L and R: the hybrid heuristic (H).

4.1.3. The impact of the swapping policies. In these experiments, we study the effect of the parameters that define the swapping policies, introduced in Section 3.3. We investigate the impact of the parameters on the final edge-cut, as well as on the number of swaps.

Table III contains the edge-cut values achieved with different values of α, a parameter of the swapping condition in Equation (8). The setting $\alpha = 2$ gives the best result for most of the graphs. Previously, we explained that it is good to use an α greater than 1, because it encourages swaps that do not change the edge-cut, but only change the distribution of color around the two negotiating vertices, in favor of vertices with a higher degree. For example, instead of having two vertices with 3 neighbors of similar color each, we prefer to have one vertex with 5 similar neighbors and another one with 1 similar neighbor. The sum will still be the same, i.e., 6. However, if α is set to a

Table II: The minimum edge-cut achieved with different sampling heuristics. $\alpha = 2$ and simulated annealing is used.

Graph	initial	*Local*	*Random*	*Hybrid*
Synth-WS	3127	1051	600	**221**
Synth-SF	5934	4571	**4151**	4169
add20	5601	3241	1446	**1206**
data	11326	3975	1583	**775**
3elt	10315	4292	1815	**390**
4elt	34418	14304	6315	**1424**
vibrobox	123931	42914	**22865**	23174
Twitter	123683	45568	41079	**41040**
Facebook	612585	181661	119551	**117844**

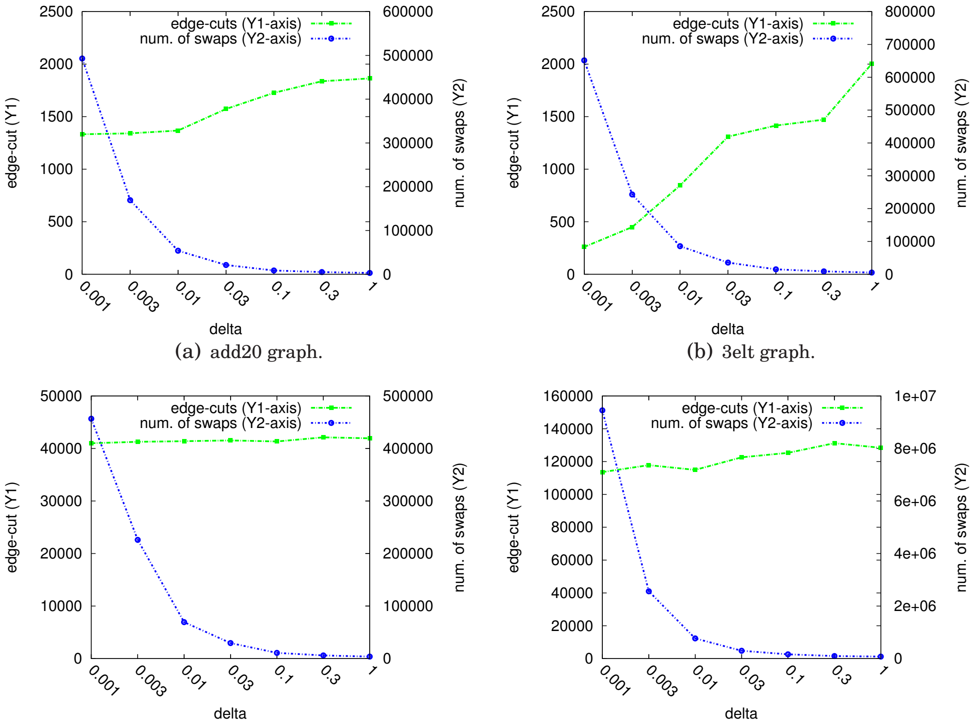
Table III: The minimum edge-cut achieved with different values for α. Hybrid sampling and simulated annealing are used.

Graph	initial	$\alpha = 1$	$\alpha = 2$	$\alpha = 3$
Synth-WS	3127	265	**221**	290
Synth-SF	5934	4190	**4169**	4215
add20	5601	**1206**	**1206**	1420
data	11326	**618**	775	1241
3elt	10315	601	**390**	1106
4elt	34418	1473	**1424**	2704
vibrobox	123931	23802	**23174**	25602
Twitter	123683	**40775**	41040	41247
Facebook	612585	124328	**117844**	133920

Table IV: The minimum edge-cut achieved with and without simulated annealing, while $\alpha = 2$ and Hybrid sampling is used.

Graph	initial	without simulated annealing	with simulated annealing
Synth-WS	3127	503	**221**
Synth-SF	5934	4258	**4169**
add20	5601	1600	**1206**
data	11326	1375	**775**
3elt	10315	1635	**390**
4elt	34418	6240	**1424**
vibrobox	123931	26870	**23174**
Twitter	123683	41087	**41040**
Facebook	612585	152670	**117844**

high value, then it can also encourage swaps that decrease this sum, thus, increase the edge-cut. For example, with $\alpha = 3$ we could end up in a state where the two vertices have 4 and 1 similar neighbors, respectively; because $4^3 + 1^3 > 3^3 + 3^3$, while the initial state with 3 and 3 similar neighbors was a better state. The higher α gets, the more likely such wrong swaps will take place. This is confirmed in the experiments that we conducted. As shown in Table III, even with $\alpha = 3$ vertices seem to overestimate the value of some swaps and end up in an inferior state. Note, this effect could happen even in case of $\alpha = 2$, but the number of wrong decision will be far lower than the good decisions. Therefore, we use $\alpha = 2$ in the rest of our experiments.



(a) add20 graph.

(b) 3elt graph.

(c) Twitter graph.

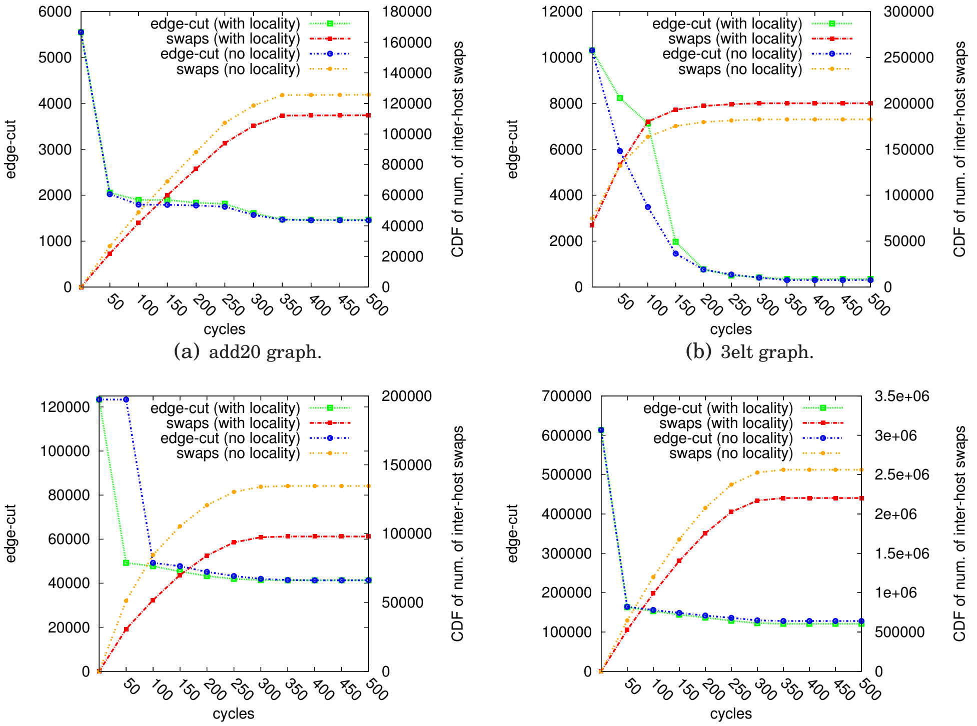
(d) Facebook graph.

Fig. 5: The number of swaps and edge-cut over δ.

Table IV lists the edge-cut with and without simulated annealing (SA). In the simulations without SA, we set $T_0 = 1$, which is the lowest allowed temperature in our case (see Equation (10)). Although the improvements due to SA might be minor for some graphs, for other graphs with various local optima SA can lead to a much smaller edge-cut. We also ran several experiments to investigate the effect of T_0 and observed that $T_0 = 2$ gives the best results in most cases.

The other parameter of the simulated annealing technique is δ, the speed of the cooling process. We investigate the impact of δ on the edge-cut and on the number of swaps. Figure 5 shows the results as a function of different values for δ. The higher δ is, the higher the edge-cut is (Y1-axis) and the smaller the number of swaps is (Y2-axis). In other words, δ represents a trade-off between the number of swaps and the quality of the partitioning (edge-cut). Note that a higher number of swaps means both a longer convergence time and more communication overhead. For example, for $\delta = 0.003$, it takes around 334 rounds for the temperature to decrease from 2 to 1, and in just very few rounds after reaching the temperature of 1, the algorithm converges. Interestingly, the social network graphs are very robust to δ in terms of the edge-cut value, so in the case of highly clustered graphs the best choice seems to be a relatively fast cooling schedule.

4.1.4. Locality. Here, we investigate the evolution of the edge-cut, the number of swaps, and the number of migrations over time, assuming the one-host-multiple-nodes model. Recall, that swaps between vertices within the same host are not counted. We assume there are four hosts in the systems, where each host gets a random subset



(a) add20 graph.

(b) 3elt graph.

(c) Twitter graph.

(d) Facebook graph.

Fig. 6: Evolution of edge-cut and the number of swaps over time.

of vertices initially. They run the algorithm to find a better partitioning by repeating the sample and swap steps periodically, until no more swaps occur (convergence). As shown in Figure 6, in both models, the algorithm converges to the final partitioning in round 350, that is, shortly after the temperature reaches 1. We also observe that the convergence time is mainly dependent on the parameters of the simulated annealing process, and so it can be controlled by the initial temperature T_0 and the cooling schedule parameter δ.

Although (as we have seen) we can achieve a much lower number of swaps in Twitter and Facebook graphs with higher values of δ without sacrificing the solution quality (Figures 5(c) and 5(d)), we have performed these experiments with the same setting of $\delta = 0.003$ for all the graphs. As shown in Figure 6(b), locally biased swapping results in relatively more inter-host swaps over the 3elt graph. Fortunately, in the rest of the graphs—that include the practically interesting social network samples as well—we can see the opposite (and more favorable) trend, namely that JA-BE-JA achieves the same edge-cut with much fewer inter-host swaps. We speculate that this is due to the fact that in the latter group of graphs there are various partitionings of the graph with a similar edge-cut value, thus, local swaps will be more likely to be good enough.

When the goal is to re-arrange the graph, data is not actually moved before the algorithm has converged to the final partitioning. Instead, on a given host, all vertices are initialized with the same color. During run-time, only the color labels are exchanged. The color of a vertex may change several times before convergence. When the algo-

Table V: The number and the fraction of vertices that need to migrate.

graph	$\|V\|$	$\|mig\|$	fraction
Synth-WS	1000	720	72%
add20	2395	1740	72.6%
3elt	4720	3436	72.7%
Twitter	2731	2000	73%
Facebook	63731	47555	74.6%

Table VI: The average and minimum edge-cut achieved by JA-BE-JA vs. METIS vs. the best known edge-cut.

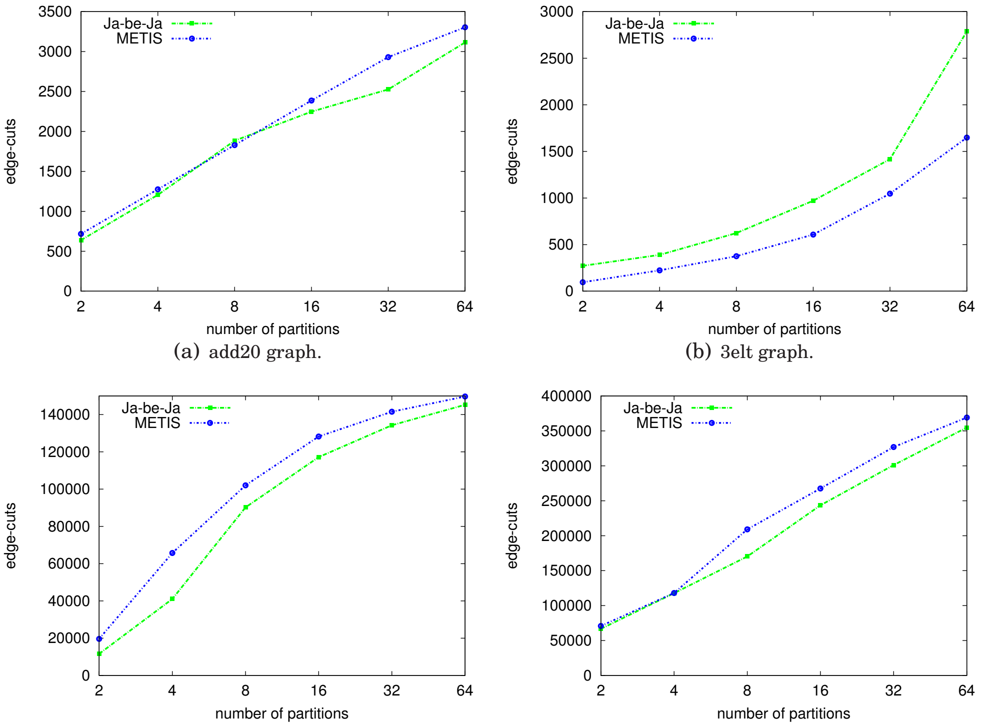
Graph	JA-BE-JA AVG (STD)	JA-BE-JA MIN	METIS	Best known edge-cut
Synth-WS	264 (27)	221	**210**	-
Synth-SF	4183 (13)	**4169**	4279	-
add20	1376(114)	**1206**	1276	1159 ([Chardaire et al. 2007])
data	974 (84)	775	**452**	382 ([Benlic and Hao 2011b])
3elt	516 (87)	390	**224**	201 ([Soper et al. 2004])
4elt	1690 (133)	1424	**374**	326 ([Walshaw 2012a])
vibrobox	24501 (767)	23174	**22526**	19098 ([Benlic and Hao 2011b])
Twitter	41251 (186)	**41040**	65737	-
Facebook	125395 (7124)	**117844**	117996	-

rithm converges, each data item (vertex) is migrated from its initial partition to its final partition indicated by its color.

Note that migration could be optimized given the final partitioning, but we simply assume that vertices with a color different from the original color will migrate. Table V shows the number of data items that need to be migrated after the convergence of the algorithm. As expected, this number constitutes nearly 75% of the vertices for a 4-way partitioning. This is because each vertex initially selects one out of four partitions uniformly at random, and the probability that it is not moved to a different partition is only 25%. Equivalently, 25% of the vertices stay in their initial partition and the remaining 75% have to migrate.

4.1.5. Comparison With the State-of-the-Art. In this section, we compare JA-BE-JA to METIS [Karypis and Kumar 1999a] on all the input graphs. We also compare these results to the best known solutions for the graphs from the Walshaw benchmark [Walshaw 2012b]. Table VI shows the edge-cut produced for the 4-way partitioning of the input graphs. In this table, we have also reported the average edge-cut achieved by JA-BE-JA, as well as the standard deviation across multiple runs. However, to be fair, we always compare the minimum edge-cut achieved by any of the solutions against each other. As shown, for some graphs, METIS produces better results, and for some others JA-BE-JA works better. However, the advantage of JA-BE-JA is that it does not require all the graph data at once, and therefore, it is more practical when processing very large graphs.

Next, we investigate the performance of the algorithms, in terms of edge-cut, when the number of the required partitions grows. Figure 7 shows the resulting edge-cut of JA-BE-JA versus METIS for 2 to 64 partitions. Naturally, when there are more partitions in the graph, the edge-cut will also grow. However, as shown in most of the graphs (except for 3elt), JA-BE-JA finds a better partitioning compared to METIS, when the



(a) add20 graph.

(b) 3elt graph.

(c) Twitter graph.

(d) Facebook graph.

Fig. 7: The minimum edge-cut achieved with JA-BE-JA vs. METIS for various number of partitions (k)

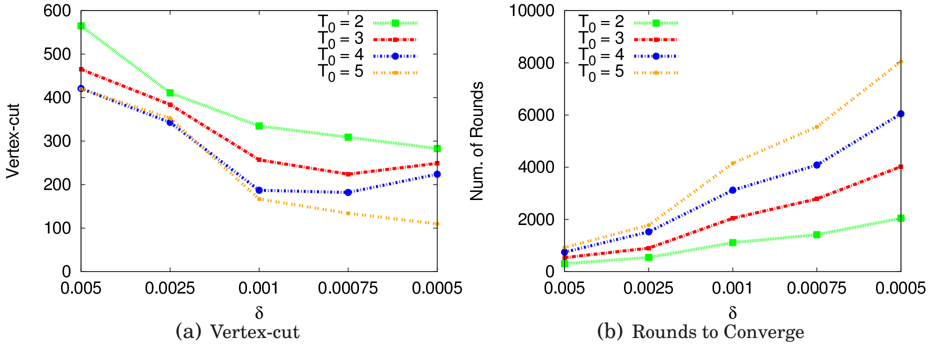
number of partitions grows. In particular, JA-BE-JA outperforms METIS in the social network graphs. For example, as shown in Figure 7(d) the edge-cut in METIS is nearly 20K more than JA-BE-JA. Note that unlike METIS, JA-BE-JA does not make use of any global information or operation over the entire graph.

4.2. Vertex-cut partitioning

In this section, we first introduce the metrics that we used for evaluating our solution. Then, we study the impact of our simulated annealing parameters on the partitioning quality. Next, we show how different policies, introduced in Section 3.4, perform. We also measure the performance of these policies in scale, and compare them to two state of the art solutions.

4.2.1. Metrics. We measure the following metrics to evaluate the quality of the vertex-cut partitioning:

— *Vertex-cut*: this metric counts the number of times that graph vertices have to be cut. That is, a vertex with one cut has replicas in two partitions, and a vertex with two cuts is replicated over three partitions. This is an important metric for when we want to put the partitioned graph in use (e.g., let's assume we want to compute Page Rank algorithm on an already partitioned graph). If a graph vertex is replicated over several partitions, every computation that involves a modification to that vertex, should be propagated to all the other replicas of that vertex, for the sake of



(a) Vertex-cut (b) Rounds to Converge

Fig. 8: Tuning simulated annealing parameters on data graph from the Walshaw archive (K=2)

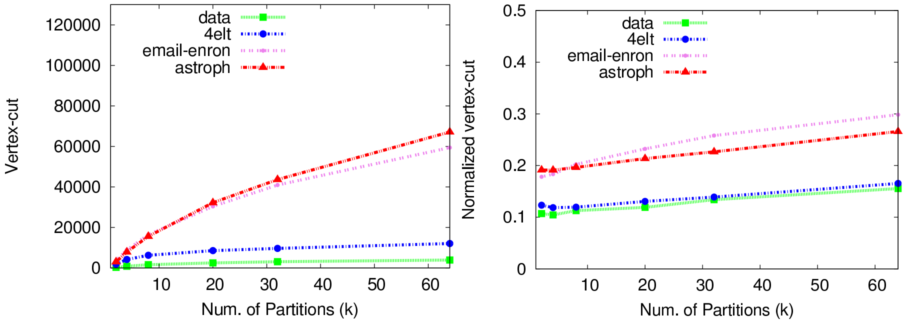
consistency. Therefore, vertex-cut directly affects the communication cost imposed by the partitioned graph.

- *Normalized vertex-cut*: this metric calculates the vertex-cut of the final partitioning relative to the random partitioning, thus, it shows to what extent the algorithm can reduce the vertex-cut.
- *Standard deviation of partition sizes*: this metric measures the Standard Deviation (STD) of normalized size of the partitions. More precisely, we first measure the size of the partitions, in terms of the number of edges, relative to the average (expected) size. In a perfect balanced partitioning the normalized size should be 1. We then calculate how much the normalized size deviates from 1.

Each experiment is repeated 3 times per graph, and for the sake of consistency we report the values associated with the minimum vertex-cut achieved among the 3 runs. Note that the trends and conclusions would remain the same should we had used the average values instead.

4.2.2. Tuning the parameters. We conducted several experiments to tune the two parameters of the simulated annealing, namely T_0 and δ. For these experiments we selected the Data graph (Table I) and $k = 2$. As shown in Figure 8(a), the vertex-cut decreases when T_0 increases. However, Figure 8(b) illustrates that this improvement is achieved in a higher number of rounds, that is, a bigger T_0 delays the convergence time. Similarly, a smaller δ results in a better vertex-cut, at the cost of more rounds. In other words, T_0 and δ are parameters of a trade-off between vertex-cut and the convergence time and can be tuned based on the priorities of the applications (see Section 4.1.3 for a similar argument). Moreover, we found that for a larger k, it is better to choose a smaller δ, because when the number of partitions increases, the solution space expands and it is more likely for the algorithm to get stuck in local optima. Unless otherwise mentioned, in the rest of our experiments, we use $\delta = 0.0005$ for $k = 32$ and $k = 64$, and $\delta = 0.001$ for other values of k.

4.2.3. Performance. Figure 9(a) depicts how the vertex-cut changes for various numbers of partitions. To better understand this result, we also report the vertex-cut of JA-BE-JA relative to that of a random partitioning in Figure 9(b). As shown, JA-BE-JA reduces the vertex-cut to nearly 10-15% for Data and 4elt graphs, and to 20-30% for our power-law graphs. Note, in general the vertex-cut is expected to increase with higher number of partitions. If we only had one partition, there would be no vertex-cut. As soon as we have more than one partition, the vertices on the border of those partitions have to be cut. The more the partitions, the bigger the bordering region gets,



(a) Vertex-cuts.

(b) Normalized vertex-cut

Fig. 9: The improvements for different number of partitions.

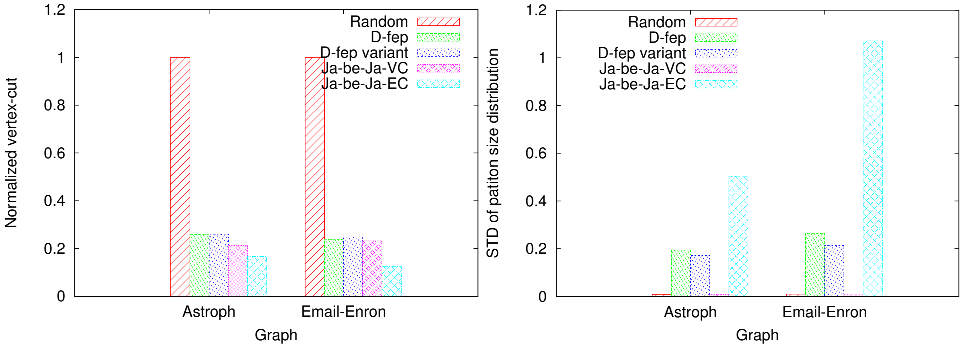
thus, the more vertices are cut. In the extreme case, where every edge has a distinct color, all the vertices have to be cut over and over (depending on their degree), and no improvement would be possible compared to a random partitioning.

4.2.4. Comparisons to the state of the art. In this section, to distinguish JA-BE-JA for edge-cut partitioning and vertex-cut partitioning, we call the former one JA-BE-JA-EC, and the latter one JA-BE-JA-VC. We compare JA-BE-JA-VC to JA-BE-JA-EC, and also to a vertex-cut partitioner by Guerrieri et al. [Guerrieri and Montresor 2014], which employs one of the two policies, namely *D-fep* or *D-fep Variant*. We also show how it would be to employ an edge-cut partitioner, e.g., JA-BE-JA-EC, to partition the graph, and then, assign the cut edges randomly to one of the partitions, where their endpoints belong to. This is similar to the example in Figure 3(b). This experiment is performed on Astroph and Email-Enron graphs with $k = 20$. To make the comparisons easier, instead of reporting the raw numbers for vertex-cut, we report the normalized vertex-cut, that is the vertex-cut relative to that of a random partitioning. As shown in Figure 10(a), JA-BE-JA-EC produces the minimum vertex-cut. However, Figure 10(b) shows that the partition sizes are very unbalanced. Note, JA-BE-JA-EC balances the number of vertices across partitions, and here we are measuring the partition size in terms of the number of edges. That is why JA-BE-JA-EC deviates from balanced partition sizes. The vertex-cuts of D-fep and its variant are more than JA-BE-JA-EC, but their partition sizes are much more balanced. JA-BE-JA-VC has a better vertex cut than D-fep and its variant, while the partition sizes are nearly equal.

As explained in Section 4.2.2, the convergence time of JA-BE-JA-VC is independent of the graph size and is mainly affected by the parameters of the simulated annealing process. While this is true for JA-BE-JA-VC, [Guerrieri and Montresor 2014] shows that both D-fep and its variant converge in only very few rounds and produce very good vertex-cuts for graphs Astroph and Email-Enron. However, as depicted in Figure 10(b) these algorithms do not maintain the balance of the partition sizes. In fact, without proper coordination, the standard deviation of the partition size distribution could grow to prohibitively large levels. JA-BE-JA-VC, however, maintains the initial distribution of edge colors, and can even be used to produce partitions of any desired size distribution, with a better vertex-cut. This comes, however, at the cost of a longer running time.

5. RELATED WORK

In this section we study some of the existing work on both edge-cut and vertex-cut partitioning.



(a) Normalized vertex-cuts.

(b) Standard deviation of partition sizes

Fig. 10: Comparisons (k=20)

5.1. Edge-cut Partitioning

A significant number of algorithms exist for edge-cut partitioning [Baños et al. 2003; Bui and Moon 1996; Hendrickson and Leland 1995; Karypis and Kumar 1998; 1999b; Walshaw and Cross 2000; Sanders and Schulz 2011]. These algorithms can be classified into two main categories: (i) centralized algorithms, which assume cheap random access to the entire graph, and (ii) distributed algorithms.

A common approach in the centralized edge-cut partitioning is to use Multilevel Graph Partitioning (MGP) [Hendrickson and Leland 1995]. METIS [Karypis and Kumar 1998] is a well-known algorithm based on MGP that combines several heuristics during its coarsening, partitioning, and un-coarsening phases to improve the cut size. KAFFPA [Sanders and Schulz 2011] is another MGP algorithm that uses local improvement algorithms based on flows and localized searches. There exist also other works that combined different meta-heuristics with MPG, e.g., Soper et al. [Soper et al. 2004] and Chardaire et al. [Chardaire et al. 2007] used a Genetic Algorithm (GA) with MPG, and Benlic et al. [Benlic and Hao 2011a] utilized Tabu search.

Parallelization is a technique used by some systems to speed up the partitioning process. For example, PARMETIS [Karypis and Kumar 1999b] is the parallel version of METIS, KAFFPAE [Sanders and Schulz 2012] is a parallelized version of its ancestor KAFFPA [Sanders and Schulz 2011], and [Talbi and Bessiere 1991] is a parallel graph partitioning technique based on parallel GA [Luque and Alba 2011].

Although the above algorithms are fast and produce good min-cuts, they require access to the entire graph at all times, which is not feasible for large graphs. Ja-be-Ja [Rahimian et al. 2013] is a recent algorithm, which is fully distributed and uses local search and simulated annealing techniques [Talbi 2009] for graph partitioning. In this algorithm each vertex is processed independently, and only the direct neighbors of the vertex, and a small subset of random vertices in the graph need to be known locally. DIDIC [Gehweiler and Meyerhenke 2010] and CDC [Ramaswamy et al. 2005] are two other distributed algorithms for graph partitioning, which eliminate global operations for assigning vertices to partitions. However, DIDIC does not guarantee to produce equal-size partitions. Moreover, while it can enforce an upper bound on the number of created partitions, it does not have control over the exact number of partitions [Averbuch and Neumann 2013].

5.2. Vertex-cut Partitioning

While there exist numerous solutions for edge-cut partitioning, very little effort has been made for vertex-cut partitioning. SBV-Cut [Kim and Candan 2012] is one of the few algorithms for vertex-cut partitioning. First, a set of balanced vertices are identified for bisecting a directed graph. Then, the graph is further partitioned by a recursive application of structurally-balanced cuts to obtain a hierarchical partitioning of the graph.

PowerGraph [Gonzalez et al. 2012] is a distributed graph processing framework that uses vertex-cuts to evenly assign edges of a graph to multiple machines, such that the number of machines spanned by each vertex is small. PowerGraph reduces the communication overhead and imposes a balanced computation load on the machines. GraphX [Xin et al. 2013] is another graph processing system on Spark [Zaharia et al. 2010; Zaharia et al. 2012] that uses a vertex-cut partitioning.

DFEP [Guerrieri and Montresor 2014] is the most recent distributed vertex-cut partitioning algorithm. It works based on a market model, where the partitions are buyers of vertices with their funding. Initially, all partitions are given the same amount of funding. Then, in each round, a partition p tries to buy edges that are neighbors of the already taken edges by p, and an edge will be sold to the highest offer. There exists a coordinator in the system that monitors the size of each partition and sends additional units of funding to the partitions, inversely proportional to the size of each partition.

6. CONCLUSION

We provided an algorithm that, to the best of our knowledge, is the first distributed algorithm for balanced graph partitioning that does not require any global knowledge. To compute the partitioning, nodes of the graph require only some local information and perform only local operations. Therefore, the entire graph does not need to be loaded into memory, and the algorithm can run in parallel on as many computers as available. We showed that our algorithm can achieve a quality partitioning as good as a centralized algorithm. We also studied the trade-off between the quality of the partitioning versus the cost of it, in terms of the number of swaps during the run-time of the algorithm.

REFERENCES

Amine Abou-Rjeili and George Karypis. 2006. Multilevel algorithms for partitioning power-law graphs. In *Proceedings of IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 10–pp.

Réka Albert and Albert-László Barabási. 2002. Statistical mechanics of complex networks. *Reviews of modern physics* 74, 1 (2002), 47.

Réka Albert, Hawoong Jeong, and Albert-László Barabási. 2000. Error and attack tolerance of complex networks. *Nature* 406, 6794 (2000), 378–382.

Konstantin Andreev and Harald Räcke. 2004. Balanced Graph Partitioning. In *Proceedings of ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*. ACM, 120–124.

Alex Averbuch and Martin Neumann. 2013. Partitioning Graph Databases-A Quantitative Evaluation. *arXiv preprint arXiv:1301.5121* (2013).

Asad Awan, Ronaldo A Ferreira, Suresh Jagannathan, and Ananth Grama. 2006. Distributed uniform sampling in unstructured peer-to-peer networks. In *Proceedings of Hawaii International Conference on System Sciences (HICSS)*, Vol. 9. IEEE, 223c–223c.

Raul Baños, Consolación Gil, Julio Ortega, and Francisco G Montoya. 2003. Multilevel heuristic algorithm for graph partitioning. In *Proceedings of Applications of Evolutionary Computing*. Springer, 143–153.

Una Benlic and Jin-Kao Hao. 2011a. An effective multilevel tabu search approach for balanced graph partitioning. *Computers & Operations Research* 38, 7 (2011), 1066–1075.

Una Benlic and Jin-Kao Hao. 2011b. A multilevel memetic approach for improving graph k-partitions. *IEEE Transactions on Evolutionary Computation (TEC)* 15, 5 (2011), 624–642.

Thang Nguyen Bui and Byung Ro Moon. 1996. Genetic algorithm and graph partitioning. *IEEE Transactions on Computers (TC)* 45, 7 (1996), 841–855.

Pierre Chardaire, Musbah Barake, and Geoff P McKeown. 2007. A probe-based heuristic for graph partitioning. *IEEE Trans. Comput.* 56, 12 (2007), 1707–1720.

David Dominguez-Sal, P Urbón-Bayes, Aleix Giménez-Vañó, Sergio Gómez-Villamor, Norbert Martínez-Bazán, and Josep-Lluis Larriba-Pey. 2010. Survey of graph database performance on the HPC scalable graph analysis benchmark. (2010), 37–48.

Jim Dowling and Amir H Payberah. 2012. Shuffling with a croupier: Nat-aware peer-sampling. In *Proceedings of IEEE International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 102–111.

Anton J Enright, Stijn Van Dongen, and Christos A Ouzounis. 2002. An efficient algorithm for large-scale detection of protein families. *Nucleic acids research* 30, 7 (2002), 1575–1584.

Wojciech Galuba, Karl Aberer, Dipanjan Chakraborty, Zoran Despotovic, and Wolfgang Kellerer. 2010. Outtweeting the twitterers-predicting information cascades in microblogs. In *Proceedings of Workshop on Online Social Networks (WOSN)*. USENIX Association, 3–3.

Joachim Gehweiler and Henning Meyerhenke. 2010. A distributed diffusive heuristic for clustering a virtual P2P supercomputer. In *Proceedings of IEEE International Parallel & Distributed Processing Symposium Workshops and Phd Forum (IPDPSW)*. IEEE, 1–8.

Joseph E Gonzalez, Yucheng Low, Haijie Gu, Danny Bickson, and Carlos Guestrin. 2012. PowerGraph: Distributed Graph-Parallel Computation on Natural Graphs.. In *Proceedings of USENIX Symposium on Operating System Design and Implementation (OSDI)*, Vol. 12. USENIX, 2.

Alessio Guerrieri and Alberto Montresor. 2014. Distributed Edge Partitioning for Graph Processing. *arXiv preprint arXiv:1403.6270* (2014).

Bruce Hendrickson. 1998. Graph partitioning and parallel solvers: Has the emperor no clothes?. In *Proceedings of Solving Irregularly Structured Problems in Parallel*. Springer, 218–225.

Bruce Hendrickson and Robert W Leland. 1995. A Multi-Level Algorithm For Partitioning Graphs. *SC* 95 (1995), 28.

Márk Jelasity, Alberto Montresor, and Ozalp Babaoglu. 2005. Gossip-based aggregation in large dynamic networks. *ACM Transactions on Computer Systems (TOCS)* 23, 3 (2005), 219–252.

George Karypis and Vipin Kumar. 1998. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on scientific Computing* 20, 1 (1998), 359–392.

George Karypis and Vipin Kumar. 1999a. Parallel multilevel series k-way partitioning scheme for irregular graphs. *SIAM Rev.* 41, 2 (1999), 278–300.

George Karypis and Vipin Kumar. 1999b. Parallel multilevel series k-way partitioning scheme for irregular graphs. *SIAM Rev.* 41, 2 (1999), 278–300.

Brian W Kernighan and Shen Lin. 1970. An efficient heuristic procedure for partitioning graphs. *Bell system technical journal* 49, 2 (1970), 291–307.

Mijung Kim and K Selçuk Candan. 2012. SBV-Cut: Vertex-cut based graph partitioning using structural balance vertices. *Data & Knowledge Engineering* 72 (2012), 285–303.

Maciej Kurant, Athina Markopoulou, and Patrick Thiran. 2010. On the bias of BFS (breadth first search). In *Proceedings of International Teletraffic Congress (ITC)*. IEEE, 1–8.

Kevin Lang. 2004. Finding good nearly balanced cuts in power law graphs. *Tech. Rep. YRL-2004-036, Yahoo! Research Labs* (2004).

Jure Leskovec. 2011. Stanford large network dataset collection. *URL http://snap. stanford. edu/data/index. html* (2011).

Jure Leskovec, Kevin J Lang, Anirban Dasgupta, and Michael W Mahoney. 2009. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. *Internet Mathematics* 6, 1 (2009), 29–123.

Yucheng Low, Danny Bickson, Joseph Gonzalez, Carlos Guestrin, Aapo Kyrola, and Joseph M Hellerstein. 2012. Distributed GraphLab: a framework for machine learning and data mining in the cloud. *Proceedings of International Conference on Very Large Databases (VLDB)* 5, 8 (2012), 716–727.

Gabriel Luque and Enrique Alba. 2011. *Parallel Genetic Algorithms: Theory and Real World Applications*. Vol. 367. Springer.

Grzegorz Malewicz, Matthew H Austern, Aart JC Bik, James C Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. 2010. Pregel: a system for large-scale graph processing. In *Proceedings of ACM Special Interest Group on Management Of Data (SIGMOD)*. ACM, 135–146.

Laurent Massoulié, Erwan Le Merrer, Anne-Marie Kermarrec, and Ayalvadi Ganesh. 2006. Peer counting and sampling in overlay networks: random walk methods. In *Proceedings of ACM Symposium on Principles of Distributed Computing (PODC)*. ACM, 123–132.

Henning Meyerhenke, Burkhard Monien, and Thomas Sauerwald. 2008. A new diffusion-based multilevel algorithm for computing graph partitions of very high quality. In *Proceedings of IEEE International Symposium on Parallel and Distributed (IPDPS)*. IEEE, 1–13.

Henning Meyerhenke, Burkhard Monien, and Stefan Schamberger. 2009. Graph partitioning and disturbed diffusion. *Parallel Comput.* 35, 10 (2009), 544–569.

Alberto Montresor and Márk Jelasity. 2009. PeerSim: A scalable P2P simulator. In *Proceedings of IEEE International Conference on Peer-to-Peer Computing (P2P)*. IEEE, 99–100.

Amir H Payberah, Jim Dowling, and Seif Haridi. 2011. Gozar: Nat-friendly peer sampling with one-hop distributed nat traversal. In *Proceedings of IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS)*. Springer, 1–14.

Fatemeh Rahimian, Amir H. Payberah, Sarunas Girdzijauskas, and Seif Haridi. 2014. Distributed Vertex-Cut Partitioning. In *Proceedings of IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS)*. Springer.

Fatemeh Rahimian, Amir H Payberah, Sarunas Girdzijauskas, Mark Jelasity, and Seif Haridi. 2013. Jabe-Ja: A distributed algorithm for balanced graph partitioning. In *Proceedings of IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*. IEEE, 51–60.

Lakshmish Ramaswamy, Bugra Gedik, and Ling Liu. 2005. A distributed approach to node clustering in decentralized peer-to-peer networks. *IEEE Transactions on Parallel and Distributed Systems (TPDS)* 16, 9 (2005), 814–829.

Peter Sanders and Christian Schulz. 2011. Engineering multilevel graph partitioning algorithms. In *Algorithms–ESA 2011*. Springer, 469–480.

Peter Sanders and Christian Schulz. 2012. Distributed Evolutionary Graph Partitioning. In *Proceedings of ALENEX*. SIAM, 16–29.

Alan J Soper, Chris Walshaw, and Mark Cross. 2004. A combined evolutionary search and multilevel optimisation approach to graph-partitioning. *Journal of Global Optimization* 29, 2 (2004), 225–241.

El-Ghazali Talbi. 2009. *Metaheuristics: from design to implementation*. Vol. 74. John Wiley & Sons.

E-G Talbi and Pierre Bessiere. 1991. A parallel genetic algorithm for the graph partitioning problem. In *Proceedings of ACM International Conference on Supercomputing (ICS)*. ACM, 312–320.

Norbert Tölgyesi and Márk Jelasity. 2009. Adaptive peer sampling with newscast. In *Proceedings of International Conference on Parallel Processing (Euro-Par)*. Springer, 523–534.

Peter JM Van Laarhoven and Emile HL Aarts. 1987. *Simulated annealing*. Springer.

Bimal Viswanath, Alan Mislove, Meeyoung Cha, and Krishna P Gummadi. 2009. On the evolution of user interaction in facebook. In *Proceedings of ACM Workshop on Online Social Networks (WOSN)*. ACM, 37–42.

Spyros Voulgaris, Daniela Gavidia, and Maarten Van Steen. 2005. Cyclon: Inexpensive membership management for unstructured p2p overlays. *Journal of Network and Systems Management* 13, 2 (2005), 197–217.

C. Walshaw. 2012a. FocusWare NetWorks MNO - a commercialised version of JOSTLE: http://http://focusware.co.uk. (Sep 2012).

C. Walshaw. 2012b. The graph partitioning archive: http://staffweb.cms.gre.ac.uk/˜wc06/partition. (Aug 2012).

Chris Walshaw and Mark Cross. 2000. Mesh partitioning: a multilevel balancing and refinement algorithm. *SIAM Journal on Scientific Computing* 22, 1 (2000), 63–80.

Duncan J Watts and Steven H Strogatz. 1998. Collective dynamics of small-world networks. *Nature* 393, 6684 (1998), 440–442.

Reynold S Xin, Joseph E Gonzalez, Michael J Franklin, and Ion Stoica. 2013. Graphx: A resilient distributed graph system on spark. In *Proceedings of International Workshop on Graph Data Management Experiences and Systems (GRADES)*. ACM, 2.

Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J Franklin, Scott Shenker, and Ion Stoica. 2012. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of USENIX Conference on Networked Systems Design and Implementation (NSDI)*. USENIX, 2–2.

Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. 2010. Spark: cluster computing with working sets. In *Proceedings of USENIX Workshop on Hot Topics in Cloud Computing (HotCloud)*. USENIX, 10–10.

Received February 2014; revised ; accepted