Swedish Institute of Computer Science

SICS Technical Report T2013:03
ISSN 1100-3154

Ja-be-Ja: A Distributed Algorithm for Balanced Graph Partitioning

Authors:
Fatemeh Rahimian
Amir H. Payberah
Sarunas Girdzijauskas
Mark Jelasity
Seif Haridi

October 2012

Ja-be-Ja: A Distributed Algorithm for Balanced Graph Partitioning

Fatemeh Rahimian[1], Amir H. Payberah[1], Sarunas Girdzijauskas[2], Mark Jelasity[3], and Seif Haridi[2]

[1]Swedish Institute of Computer Science (SICS),
{fatemeh,amir}@sics.se
[2]KTH - Royal Institute of Technology,
{sarunasg,haridi}@kth.se
[3]Hungarian Academy of Sciences and University of Szeged,
jelasity@inf.u-szeged.hu

October 2012

Abstract

Balanced graph partitioning is a well known NP-complete problem with a wide range of applications. These applications include many large-scale distributed problems such as the optimal storage of large sets of graph-structured data over several hosts, or identifying clusters in on-line social networks. In such 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 distributed graph partitioning algorithm, called Ja-be-Ja[1]. The algorithm is massively parallel: each graph node is processed independently, and only the direct neighbors of the node, and a small subset of random nodes in the graph need to be known. 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 perform a thorough experimental analysis, which shows that the minimal edge-cut value 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.

Keywords. graph partitioning; distributed algorithm; load balancing;

[1]Ja-be-Ja means swap in Persian.

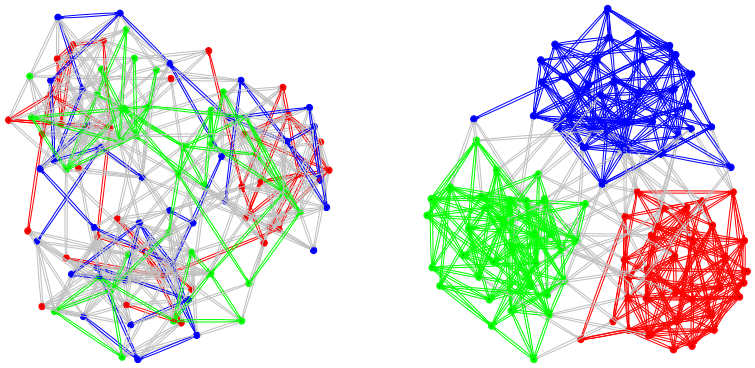
1 Introduction

Finding good partitions is a well-known and well-studied problem in graph theory [15]. The graph partitioning problem, sometimes referred to as the *min-cut problem*, is formulated as dividing a graph into a predefined number of components, such that the number of edges between different components is small. A variant of this problem is the *balanced* or uniform graph partitioning problem, where it is also important that the components hold an equal number of nodes. The examples of important applications include biological networks, circuit design, parallel programming, load balancing, graph databases and on-line social network analysis. The motivation for graph partitioning depends on the application. A good partitioning can be used to minimize communication cost, to balance load, or to identify densely connected clusters. For instance, several studies have been made to emphasize the importance of a good partitioning for reducing the communication cost in graph databases [3, 7, 29]. Figures 1(a) and 1(b) are examples of a poor and a good partitioning of a graph, respectively.

Here, we focus on graphs that are *extremely large-scale*. For example, every day petabytes of data are generated and processed in today's on-line social networking services, such as Facebook and Twitter. This data can be modeled as a graph, in which nodes represent users and edges represent the relationship between them. Similarly, search engines, like Google and Yahoo, manage very large amounts of data to capture and analyze the structure of the Internet. Likewise, this data can be modeled as a graph, with websites as nodes and the hyperlinks between them as edges.

The very large scale of the graphs we target poses a major challenge. Although a very large number of algorithms are known for graph partitioning [11, 17, 18, 19, 25, 26, 31, 32], 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, 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 nodes hosted on a single computer.

In this paper, we provide a distributed balanced graph partitioning algorithm, which does not require any global knowledge of the graph topology. That is, we do not have cheap access to the entire graph and we can process it only with partial information. Our solution, called Ja-be-Ja, is a decentralized local search/simulated annealing algorithm [1]. Each node of the graph is a processing unit, with local information about its neighboring nodes, and a small subset of random nodes in the graph, which it acquires by purely local interactions. Initially, every node selects a random partition, and over time nodes swap their partitions to improve a local utility value based on the number of neighbors they have in the same partition. Our algorithm is uniquely designed to deal with extremely large distributed graphs. The algorithm achieves this through its locality, simplicity and lack of synchronization requirements, which enables it to be adapted easily to graph processing frameworks such as Pregel [23] or GraphLab [22], or it can be applied on fully distributed graphs as well, where



(a) A poor partitioning of a graph. Nodes are partitioned randomly so there are many inter-partition links.

(b) A good partitioning of the same graph, where nodes that are highly connected are assigned to the same partition.

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

all network nodes represent a single graph vertex.

To evaluate Ja-be-Ja, we use multiple datasets of different characteristics, including a few synthetically generated graphs, some graphs that are well-known in the graph partitioning community [39], and some sampled graphs from Facebook [36] and Twitter [12]. We first investigate the impact of different heuristics on the resulting partitioning of the input graphs, and then compare Ja-be-Ja to Metis [17], 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, sometimes even better than, a centralized solution. In particular, for large graphs that represent the real-world social network structure in Facebook and Twitter, Ja-be-Ja outperforms Metis [17].

In the next section we define the exact problem that we are targeting, together with the boundary requirements of the potential applications. In Section 3 we study the related work of graph partitioning. Then, in Section 4 we explain Ja-be-Ja in detail, and evaluate it in Section 5. 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 formulate the optimization problem and describe our assumptions about the system we operate in.

2.1 Balanced k-way graph partitioning

We assume that we are given an undirected graph $G = (V, E)$, where V is the set of nodes (or vertices) and E is the set of edges. A k-way partitioning divides V into k subsets. Intuitively, in a good partitioning the number of edges that cross the boundaries of components is minimized. This is sometimes referred to as the *min-cut* problem in graph theory.

Balanced (or uniform) partitioning refers to the problem of partitioning the graph into equal-sized components, 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 partitioning can be given with the help of a partition function $\pi : V \to \{1, \ldots, k\}$ that assigns a *color* to each node. Instead of the notation $\pi(p)$, we will use the shorter π_p to refer to the color of node p. Nodes with the same color form a partition. Let us denote the set of neighbors of node p by N_p. We define $N_p(c)$ as the set of neighbors of p that have color c:

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

Finally, let the number of neighboring nodes of node p be x_p, and $x_p(c) = |N_p(c)|$ be the number of neighbors of p with color c.

We define the *energy* of the system as the number of edges between nodes with different colors (equivalent to edge-cut). Accordingly, the energy of a node is the number of its neighbors with a different color, and the energy of the graph is the sum of the energy of the nodes:

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

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

$$\pi^* = \arg\min_{\pi} E(G, \pi) \tag{3}$$

$$s.t. \quad |V(c_1)| = |V(c_2)|, \forall\ c_1, c_2 \in \{1, \ldots, k\} \tag{4}$$

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

2.2 Data distribution model

We assume the nodes of the graph are processed periodically and independently, where each node has access only to the state of its immediate neighbors. The nodes 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 [22] or Pregel [23], Google's distributed framework for processing very large graphs. It can also be used in peer-to-peer overlays, where each node is an independent computer. In both cases, no shared memory is required. Nodes 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 nodes. We call this the *one-host-multiple-nodes* model. Here, nodes on the same host can benefit from a shared memory on that host. For example, if a node exchanges some information with other nodes 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 environments, where each computer can emulate thousands of nodes at the same time.

3 Related Work

There exists some related work that address the k-way balanced graph partitioning problem, but in a centralized model. On the other hand, there are algorithms that have a distribution model similar to Ja-be-Ja, but do not compute a predefined number of balanced partitions. Here, we briefly overview some of these algorithms. To the best of our knowledge, Ja-be-Ja is the first algorithm that fills in the gap between these two sets of algorithms and can produce balanced partitions in a completely distributed model.

3.1 Balanced Graph Partitioning Algorithms.

Metis [17] is a widely known and successful algorithm based on *Multilevel Graph Partitioning (MGP)* [15]. MGP generally works in three phases. First, a sequence of smaller and smaller graphs are produced from the original graph, by iteratively contracting edges and unifying nodes. This is repeated until the number of nodes in the coarsened graph is small enough to perform an inexpensive partitioning. Second, the smallest graph is partitioned, and third, the partitions are propagated back through a sequence of un-contracting nodes and edges. Note, the best partition for the coarsened graph may not be optimal for the uncoarsened original graph, thus, the third phase also includes some local refinements to improve the cut size as the edges are un-contracted. Therefore, the MGP approach is usually coupled with other heuristics for local refinement. Metis combined several heuristics during coarsening, partitioning, and uncoarsening phases to improve the cut size. It also used a greedy refinement (GR) method, which was found to be significantly faster than the original MGP algorithm.

There are many other algorithms based on MGP. For example, Soper et al. [33] proposed the first algorithm that combined an evolutionary search technique with MPG. In this work, crossover and mutation operators are used to compute edge biases, which yield hints for the underlying multilevel graph partitioner. KaFFPa [32] is another MGP algorithm using local improvement algorithms that are based on flows and localized searches. Chardaire et al. [8] proposed a meta-heuristic which can be viewed as a genetic algorithm without

selection. Benlic et al. [5] provided a perturbation-based iterated tabu search procedure for partition refinement of each coarsened graph.

However, none of the above algorithms can run in parallel. In order to speedup the partitioning process for very large-scale graphs, designing algorithms that can be parallelized is inevitable. Many efforts in this direction have been thusly made. For instance, parMetis [18] is the parallel version of Metis, that has the fastest available parallel code. However, the speedup comes at the cost of lower quality partitions, as parMetis does not produce as good partitions as Metis does. KaFFPaE [31] is also a parallelized MGP algorithm, which produces even better partitions compared to its non-parallel ancestor KaFFPa [32]. Note, although these algorithms can produce the final partitioning faster, they require access to the entire graph at all times, which renders them very expensive for large graphs that can not fit into memory of a single computer.

3.2 Distributed Graph Partitioning Algorithms.

Apart from, Ja-be-Ja, there exist some other algorithms that operate based on partial information. The decentralized nature of these algorithms enables them to process very large graphs. For example, DiDiC [13] is a distributed diffusion-based algorithm that eliminates all the global operations for assigning nodes to partitions. Also, Cdc [30], which adopts some ideas from the diffusion-based models, is particularly designed for peer-to-peer networks. However, unlike Ja-be-Ja, these solutions may produce partitions of drastically different sizes. In fact, they tend to find good-shaped partitions rather than balanced ones, and therefore, the number and size of yielded partitions can not be controlled, as it depends on the topology of the input graph.

4 Solution

We propose Ja-be-Ja, a distributed heuristic algorithm for the balanced k-way graph partitioning problem.

4.1 The basic idea

Recall, that we defined the energy of the system as the number of edges between nodes with different colors, and the energy of a node is the number of its neighbors with a different color.

The basic idea is to initialize colors uniformly at random at each node, and then to apply heuristic local search to push the configuration towards lower energy states (min-cut). The local search operator is executed by all the graph nodes in parallel: each node attempts to change its color to the most dominant color among its neighbors. However, in order to preserve the size of the partitions, the nodes can not change their color independently. Instead, they only swap their color with one another. Each node iteratively selects another node

from either its neighbors or a random sample, and investigates the pair-wise benefit of a color exchange. If the color exchange results in a total lower energy then the two nodes swap their colors. Otherwise, they preserve their colors.

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 [34] as we describe below. Later, in the evaluation section (Section 5), we show the impact of this technique on the quality of the final partitioning.

Note that—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 so we cannot compute the minimum edge-cut in a reasonable time, even with a centralized solution and a complete knowledge of the graph. In Section 5.6, however, we compare our results with the best known partitioning so far.

4.2 Swapping: the local search operator

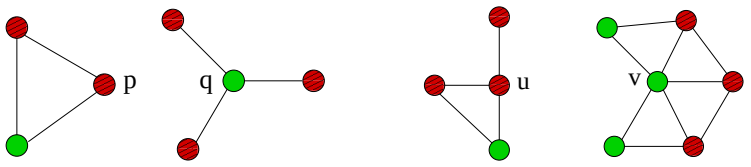
Let us elaborate on how the nodes select the neighbor to swap colors with. The first thing is to select a set of candidate nodes. We consider three possible ways of selecting this subset:

- Local (L): every node considers its directly connected nodes (neighbors) as candidates for color exchange.
- Random (R): every node selects a uniform random sample of the nodes in the graph. Note that there exist multiple techniques for taking a uniform sample of a given graph at a low cost [4, 10, 16, 24, 28, 37].
- Hybrid (H): in this policy first the neighbor nodes are selected (similar to the local policy). If this selection fails to improve the energy function, the node is given another chance for improvement, by letting it to select nodes from its random sample (similar to the random policy).

Now that we have established the methods to select candidates, we need to define how we select the swap partner. To decide if two nodes should swap their colors or not, we require: (i) a function to measure the pair-wise benefit of a color exchange, and (ii) some policy for escaping local optima.

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

$$x_p(\pi_q)^\alpha + x_q(\pi_p)^\alpha > x_p(\pi_p)^\alpha + x_q(\pi_q)^\alpha \tag{5}$$



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

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

Figure 2: Examples of two potential color exchanges.

where α is a parameter of the energy function. 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 nodes p and q in Figure 2(a) is accepted, as the nodes moves from a state with 1 and 0 neighbors of a similar color, to 1 and 3 such neighbors, respectively. However, nodes u and v in Figure 2(b), each in a state with 2 neighbors of a similar color, do not exchange their colors, if $\alpha = 1$ ($2 + 2 \not> 1 + 3$). If we set $\alpha > 1$, then nodes u and v will exchange their colors. Although, this exchange does not directly reduce the total edge-cut of the graph, it could be desirable, as it increases the probability of future color exchanges for the two green nodes, currently in the neighborhood of node v. In section 5 we tune parameter α to achieve the best results.

To avoid becoming stuck in a local optima, we use the well-known Simulated Annealing (SA) technique [34]. We introduce a *temperature* (T) and decrease it over time, similar to the cooling process [34]:

$$(x_p(\pi_q)^\alpha + x_q(\pi_p)^\alpha) \times T > x_p(\pi_p)^\alpha + x_q(\pi_q)^\alpha \tag{6}$$

As a result, in the beginning we might move in a direction which degrades the energy function, i.e., nodes exchange their color even if it increases the edge-cut. 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 SA process are (i) T_0, the initial temperature, which is greater than or equal one, and (ii) δ, the speed of the cool down process.

The temperature at round r is calculated by $T_r = T_{r-1} - \delta$, until it reaches and stays at value 1. From then on, only those exchanges that lead to some improvement will be accepted. We ran several experiments to tune these parameters, some of which will be presented in section 5.

We also use a multi-start search [34], by running the algorithm many times, starting from different initial states. Note, this technique is applied in a distributed model. More precisely, after each run, nodes use a gossip-based aggregation method [16] to calculate the edge-cut in the graph. If the new edge-cut is less than the previous one, they replace the best solution so far, by storing the smaller edge-cut value together with their local color for that particular cut.

Algorithm 1 Sample and Swap algorithm at node p

Require: Any node p in the graph has the following methods:

- $getNeighbors()$: returns p's neighbors.
- $getRandomSample()$: returns a uniform sample of all the nodes.
- T_0: the initial temperature.
- δ: the cool down speed.
- $T_r = T_0$ initially.

```
1:  procedure SampleAndSwap
2:      partner ← findPartner(p.getNeighbors(), T_r)
3:      if partner = null then
4:          partner ← findPartner(p.getRandomSample(), T_r)
5:      end if
6:      if partner ≠ null then
7:          handshake for color exchange between p and partner
8:      end if
9:      T_r ← T_r − δ
10:     if T_r < 1 then
11:         T_r ← 1
12:     end if
13: end procedure
```

4.3 Ja-be-Ja

Given an appropriate sampling policy and swapping technique, Ja-be-Ja is simply a combination of these two components. Algorithms 1 and 2 present the core of Ja-be-Ja. As shown in Algorithm 1, the hybrid node selection policy (Heuristic H) is used, which first tries the local (Algorithm 1, line 2) policy, and if it fails it follows the random (Algorithm 1, line 4) policy. Algorithms 2 shows how the partner is selected. In lines $5-11$ the two sides of Condition 5 are calculated. In line 13, these computed values are compared, while 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 for clarity. That is, the actual swap is done after the two nodes perform a handshake and agree on the swap. This is necessary, because the deciding node might have outdated information about the partner node, in which case, the calculations would be invalid. To avoid this, the deciding node sends a swap request to the partner node, along with its current color (π_p), partner's color ($\pi_{partner}$), the number of its neighbors with similar color to itself ($x_p(\pi_p)$), and the number of its neighbors with similar color to that of the partner node ($x_p(\pi_{partner})$). This is all the information that the partner node requires to verify the advantage of swap. If verification succeeds, the partner node sends an acknowledge (ACK) message back to the node and swap takes place. Otherwise, a negative acknowledgment message (NACK) is sent and the two nodes preserve their previous colors.

These sample and swap processes are periodically repeated by all the nodes, in parallel, and when no more swaps take place in the graph, the algorithm is converged. In principle, one needs a distributed protocol to detect convergence.

Algorithm 2 Find the best node as swap partner for node p

Require: Any node p in the graph has the following methods:

- $getDegree(c)$: returns the number of p's neighbors that have color c.

```
1:  function FINDPARTNER(Node[] nodes, float T_r)
2:      highest ← 0
3:      bestPartner ← null
4:      for q ∈ nodes do
5:          x_pp ← p.getDegree(p.color)
6:          x_qq ← q.getDegree(q.color)
7:          old ← x_pp^α + x_qq^α
8:
9:          x_pq ← p.getDegree(q.color)
10:         x_qp ← q.getDegree(p.color)
11:         new ← x_pq^α + x_qp^α
12:
13:         if (new × T_r > old) ∧ (new > higest) then
14:             bestPartnere ← q
15:             highest ← new
16:         end if
17:     end for
18:     return bestPartner
19: end function
```

However, in Section 5.5 we will show experimentally that the convergence time of the algorithm is independent of the graph type and size. Hence, in practice, no global information is required to detect the convergence; one can stop the protocol after a fixed number of steps, as this paramater is rather robust.

In the one-host-multiple-nodes model, the only change required to the core algorithm is to give preference to local host swaps in Algorithm 2. That is, if there are several nodes as potential partners for a swap, the seeking node selects the one that is located on the local host, if there is any. Note also, that in this model not each and every node 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 nodes.

We add a locality coefficient to the calculated benefit of a swap, i.e., γ (lines 14 and 15, Algorithm 3). Note that in the one-host-one-node model we have $\gamma = 1$. In Section 5.5, we study the trade-off between communication overhead and the edge-cut with and without considering the locality.

4.4 Generalizations of Ja-be-Ja

In this paper we discuss the case when the graph links are not weighted (they have a weight of one), and the partition sizes must be equal. However, these are not inherent constraints. Although we do not analyze these generalizations, we briefly describe how to deal with weighted graphs and arbitrary pre-defined partition sizes.

Weighted graphs. In real world applications links are often weighted. For example, in a graph database some operations are performed more frequently, thus, some links are accessed more often [9]. Hence, it makes sense to prioritize

Algorithm 3 Find the best node as swap partner in the one host-multiple nodes model

Require: Any node p in the graph has the following methods:

- $getDegree(c)$: returns the number of p's neighbors that have color c.
- $getHost()$: returns the host address of node p.

```
1:  function findPartner(Node[] nodes, float T_r)
2:      highest ← 0
3:      bestPartner ← null
4:      for q ∈ nodes do
5:          x_pp ← p.getDegree(p.color)
6:          x_qq ← q.getDegree(q.color)
7:          old ← x_pp^α + x_qq^α
8:
9:          x_pq ← p.getDegree(q.color)
10:         x_qp ← q.getDegree(p.color)
11:         new ← x_pq^α + x_qp^α
12:
13:         if p.getHost() = q.getHost() then
14:             old ← old × γ                          ▷ γ ≥ 1
15:             new ← new × γ
16:         end if
17:
18:         if (new × T_r > old) ∧ (new > higest) then
19:             bestPartnere ← q
20:             highest ← new
21:         end if
22:     end for
23:     return bestPartner
24: end function
```

such links when partitioning the graph. Ja-be-Ja can easily handle weighted links; we simply need to change the definition of $x_p(c)$. More precisely, instead of just counting the number of neighboring nodes with the same color, we sum the weights of these links:

$$x_p(c) = \sum_{q \in N_p(c)} w(p, q) \tag{7}$$

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

Arbitrary partition sizes. For example, assume we want to split the data over two machines that are not equally powerful. If the first machine has twice as many 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. For example, if nodes 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.

5 Experimental evaluation

We implemented Ja-be-Ja on PeerSim [27], a discrete event simulator for building P2P protocols. 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 [17], a well-known efficient centralized solution, and (ii) the best known available results from the Walshaw benchmark [39] for several graphs. Unless stated otherwise, we compute a 4-way partitioning of the input graph, with $T_0 = 2$, $\delta = 0.003$, $\alpha = 2$, and $\gamma = 1$.

5.1 Metrics

Although the most important metric for graph partitioning is edge-cut (or energy), there are a number of studies [14] that show that the edge-cut alone is not enough to measure the partitioning quality. Several metrics are, therefore, defined and used in the literature [25, 26], among which we selected the following ones in our evaluations:

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

The first two metrics are clear, but we need to specify the data migration metric in more detail. This metric makes sense only in the one-host-multiple-nodes model, where the intuition is that after finding a good partitioning, we want to re-arrange the graph nodes according to the partitioning that was found. Now, if we initialize all the graph nodes that are initially located at a given host with the same color, then an upper bound on the number of nodes that need to be migrated is given by the number of nodes that change color during the execution of the algorithm. Indeed, we use this latter definition as our data migration metric, and accordingly we do initialize the color of nodes with the same color on the same host as described above in the one-host-multiple-nodes scenario.

5.2 Datasets

We have used three types of graphs: (i) a set of synthetically generated graphs, (ii) several graphs from Walshaw archive [39], and (iii) sampled graphs from two well-known social networks, i.e., Twitter [12] and Facebook [36]. These graphs are listed in Table 1.

Table 1: Datasets

Dataset	|V|	|E|	Type	Reference
Synth-R	1000	4955	Synth.	-
Synth-C	1000	4912	Synth.	-
Synth-HC	1000	4889	Synth.	-
Synth-WS	1000	4147	Synth.	-
Synth-SF	1000	7936	Synth.	-
add20	2395	7462	Walshaw	[39]
data	2851	15093	Walshaw	[39]
3elt	4720	13722	Walshaw	[39]
4elt	15606	45878	Walshaw	[39]
vibrobox	12328	165250	Walshaw	[39]
Twitter	2731	164629	Social	[12]
Facebook	63731	817090	Social	[36]

5.2.1 Synthetic Graphs

We generated five different graphs synthetically. In all cases, the graph is undirected, all generated edges should be interpreted as bidirectional. There are no parallel edges either, if an edge is selected many times, still only one edge is generated.

- *Synth-R*: There are 1000 nodes in the graph, and each node selects 5 neighbors uniformly at random. Since each node might also be selected as neighbor by other nodes, the average node degree is around 10.
- *Synth-C*: Again, we have 1000 nodes in the graph, indexed from 1 to 1000. The graph is divided into four quarters, and every node in each quarter selects 5 neighbors in total, either from its own quarter with probability 0.75, or from other quarters with probability 0.25. As a result, four clusters are recognizable in the graph, though there are many links between the clusters.
- *Synth-HC*: This graph is similar to Synth-C, except that the probability that neighbors are selected from the same quarter is 0.95. Therefore, this graph is highly clustered, with very few links between clusters.
- *Synth-WS*: This is a graph based on Watts-Strogatz model [20], with 1000 nodes and average degree 8 per node. First, a lattice is constructed and then, some edges are rewired with probability 0.02.
- *Synth-SF*: This is an implementation of the Barabasi-Albert model [2] of growing scale free networks. There are 1000 nodes in the graph with an average degree 16.

5.2.2 The Walshaw Archive

The Walshaw archive [39] 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 1.

5.2.3 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 [12]. 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 [21] sampling nearly 10000 nodes by performing multiple breath first searches (BFS).

Initially, we randomly selected a number of nodes from the dataset. Then we added to this sample the nodes being followed by the selected nodes. Next, we extracted all the relations (following or being followed) between these nodes. Finally, we removed links to the nodes outside the sample. In order to ensure that this approach preserves the properties of the complete log, we took several samples, then compared the similarity of degree distribution of the samples and that of the full log. The results confirmed that the sampled graph has similar statistical properties to the full graph. Note that, although in Twitter the links are directed, we used an undirected version of the sampled graph for the purpose of our evaluations.

We also used a sample graph of Facebook, which is made available by Viswanath et. al. [36]. 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.

5.3 The impact of the sampling policies

In this section, we study the effect of different partner selection heuristics on the edge-cut. These heuristics were introduced in Section 4.2 and are denoted by L, R, and H. Here, we evaluated the one-node-one-host model, and to take uniform random samples of the graph we applied Newscast [16, 35] in our implementation. As shown in Table 2, 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 node, the edge-cut is reduced to 30% of the initial number for the Facebook graph. The

Table 2: The impact of different sampling heuristics on edge-cut.

Graph	initial	L	R	H
Synth-R	3730	2573	**2228**	2237
Synth-C	3670	1221	**910**	**910**
Synth-HC	3655	1528	**198**	**198**
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**

random selection policy, i.e., heuristic R, works even better than local (L) for all the graphs, as it is less likely to get stuck in a local optima. The best result for most graphs, however, is achieved with the combination of the two: the hybrid heuristic (H).

5.4 The impact of the swapping policies

In these experiments, we tune the parameters of the swapping policies, introduced in Section 4.2, by investigating their impact on the final edge-cut, as well as, the number of swaps. In Table 3 we observe how different values for α in the swapping condition (Condition 5) improve the edge-cut of the input graphs. We conclude that $\alpha = 2$ gives us a better result for a larger variety of the graphs. We also observe values greater than one have some negative impact on the final result.

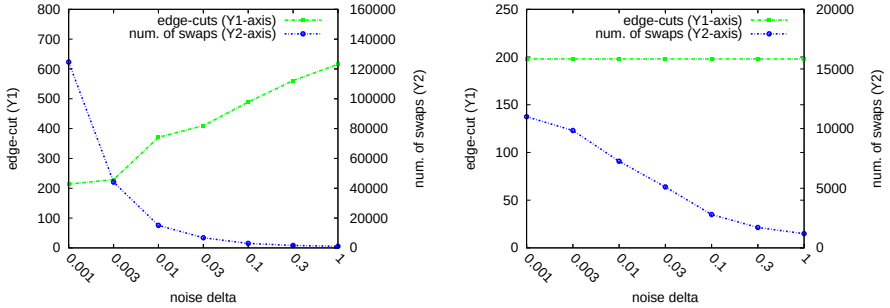
Table 4 lists the edge-cut improvement of Ja-be-Ja, with and without simulated annealing (SA). In the simulations without SA, we set $T_0 = 1$ to remove its effect on the energy function (Formula 6). Although the improvements with SA might be negligible for some graphs, for other graphs with various local optima, it can lead to much less edge-cut. We also ran several experiments to tune T_0, i.e., the initial temperature in the SA process. For the sake of space, we do not report all the results here. However, we concluded that $T_0 = 2$ gives us the best results. The other parameter of the simulated annealing technique is δ, the speed of the cool down process. We investigate the impact of δ on the edge-cut and the number of swaps. Figure 3 depicts the results with different values for δ. The higher δ is, we observe a higher edge-cut (Y1-axis) and a smaller number of swaps (Y2-axis). In other word, δ introduces a trade-off between the number of swaps and the quality of the partitioning (edge-cut). Note, a higher number of swaps means both a longer convergence time and more communication over-

Table 3: Tuning the energy function for a better edge-cut.

Graph	initial	$\alpha = 1$	$\alpha = 2$	$\alpha = 3$
Synth-R	3730	2262	2237	**2216**
Synth-C	3670	**910**	**910**	**910**
Synth-HC	3655	**198**	**198**	**198**
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

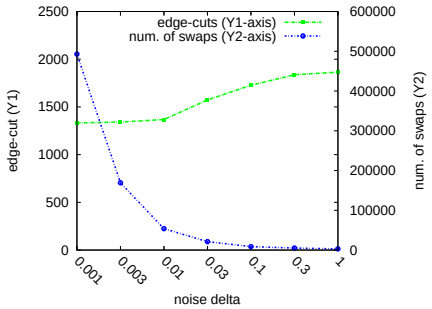
Table 4: The impact of simulated annealing on edge-cut.

Graph	initial	H	$H + SA$
Synth-R	3730	2295	**2237**
Synth-C	3670	**910**	**910**
Synth-HC	3655	**198**	**198**
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**

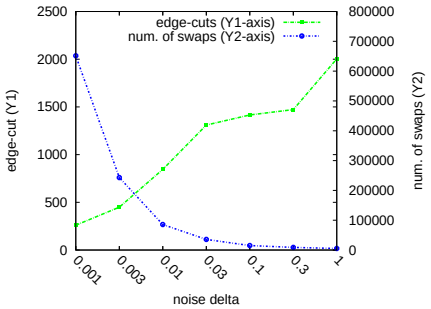


(a) Synth-WS graph.

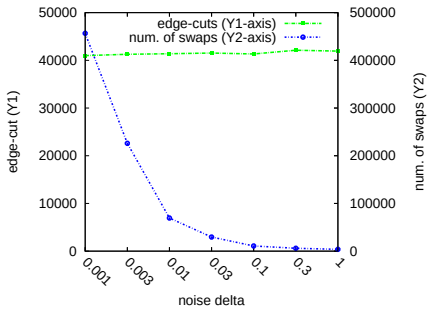
(b) Synth-HC graph.



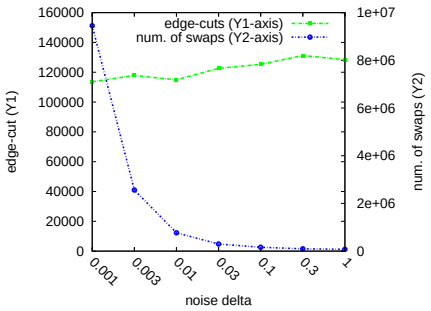
(c) add20 graph.



(d) 3elt graph.



(e) Twitter graph.



(f) Facebook graph.

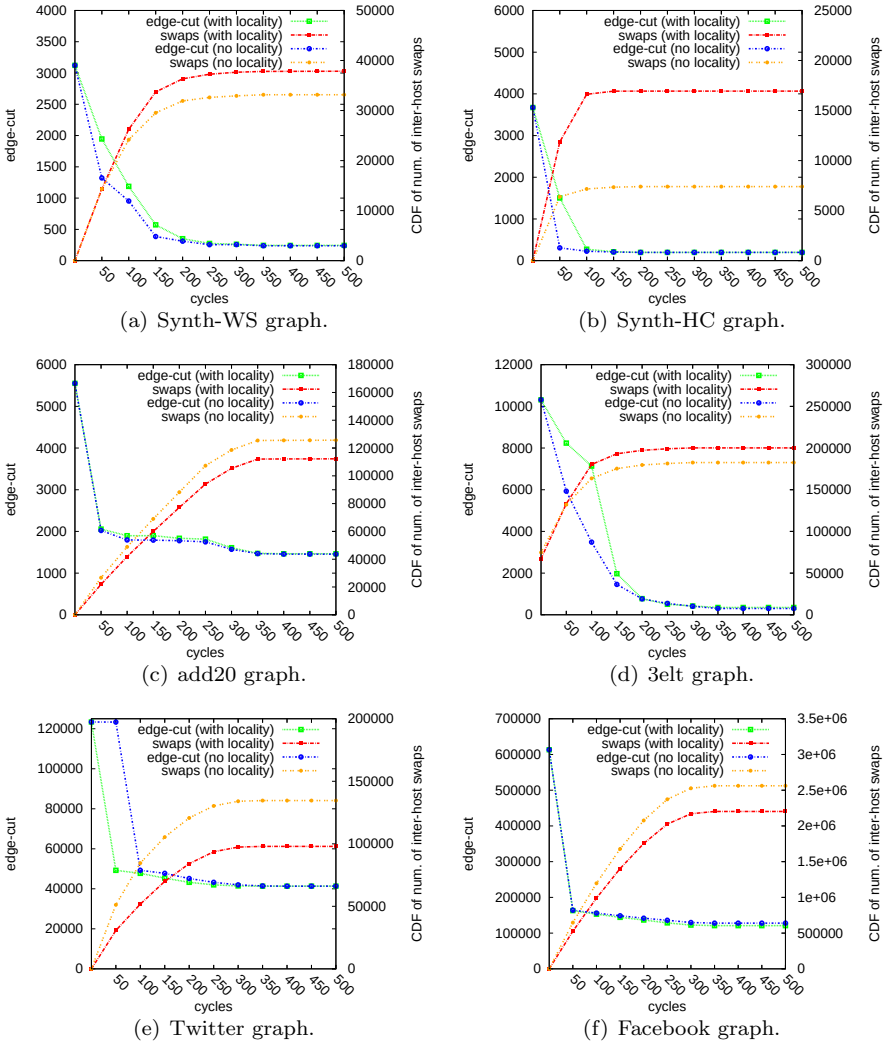
Figure 3: The number of swaps and edge-cut with different δ.

head. For example, if we choose $\alpha = 0.003$, it takes around 334 rounds for the temperature to go down from 2 to 1, and in just very few rounds after that, the algorithm converges.

5.5 Locality trade-offs

Here, we investigate the evolution of edge-cut, the number of swaps, and the number of migrations over time in one-host-multiple-nodes when the locality

is taken into acccount against when the locality in not an issue. Note that in this model, the swaps between nodes in one host are not counted. To promote local swaps, we set $\gamma = 1000$, meaning that if there are a number of potential swap partners for a node, those that are located on the local host are preferred. We assume there are four hosts in the systems, where each host gets a random



(a) Synth-WS graph.

(b) Synth-HC graph.

(c) add20 graph.

(d) 3elt graph.

(e) Twitter graph.

(f) Facebook graph.

Figure 4: The number of inter-partition swaps and edge-cut over time.

subset of nodes initially. They run the algorithm to find a better partitioning, by repeating the sample and swap steps periodically, until no more swaps occurs (convergence). As shown in Figure 4, in both models, the algorithm converges to

Table 5: The number of nodes that need to migrate.

graph	$\|V\|$	$\|mig\|$
Synth-HC	1000	734
Synth-WS	1000	720
add20	2395	1740
3elt	4720	3436
Twitter	2731	2000
Facebook	63731	47555

the final partitioning in round 350, that is, short after the temperature reaches one. It can also be observed that the convergence time is independent of the graph size and type, and only depends on the δ, the speed of cool down process. Although, we achieved much lower number of swaps in Twitter and Facebook graphs with higher δ, without a big negative effect on their edge-cut (Figures 3(e) and 3(f)), we have done these experiments with $\delta = 0.003$ for all the graphs.

Moreover, Figures 4(a), 4(b) and 4(d) suggest that if the local optima are far from a global optima for a graph, the local swaps are more likely to lead us towards a not good-enough local optima and get stuck there. However, if there are many possible quality partitionings with a similar edge-cut, as in social networks (Figures 4(e) and 4(f)), local swaps decrease the number of inter-partition swaps with almost no compromise for edge-cut.

Note, the actual data is not moved before the algorithm is converged to the final partitioning. In fact, during run-time, each node is only marked with a partition identifier (or a color) that suggests which partition the node should belong to. This decision may change several time before convergence, and that is why we do not want to migrate data between different hosts. When the algorithm converges, each data item is migrated from its initial partition to its final partition.

Table 5 shows the number of data items that need to be migrated after convergence of the algorithm. As expected, this number constitutes nearly 75% of the nodes for a 4-way partitioning. This is because each node 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 nodes stay in their initial partition and the remaining 75% have to migrate.

5.6 Comparison With the State-of-the-Art Graph Partitioners

In this section, we compare Ja-be-Ja to Metis [17] on all the input graphs. We also compare these results to the best known solutions for the graphs from the Walshaw benchmark [39]. Table 6 shows the edge-cut of the final 4-way partitioning. As shown, for some graphs, Metis produces better results, and for some other 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

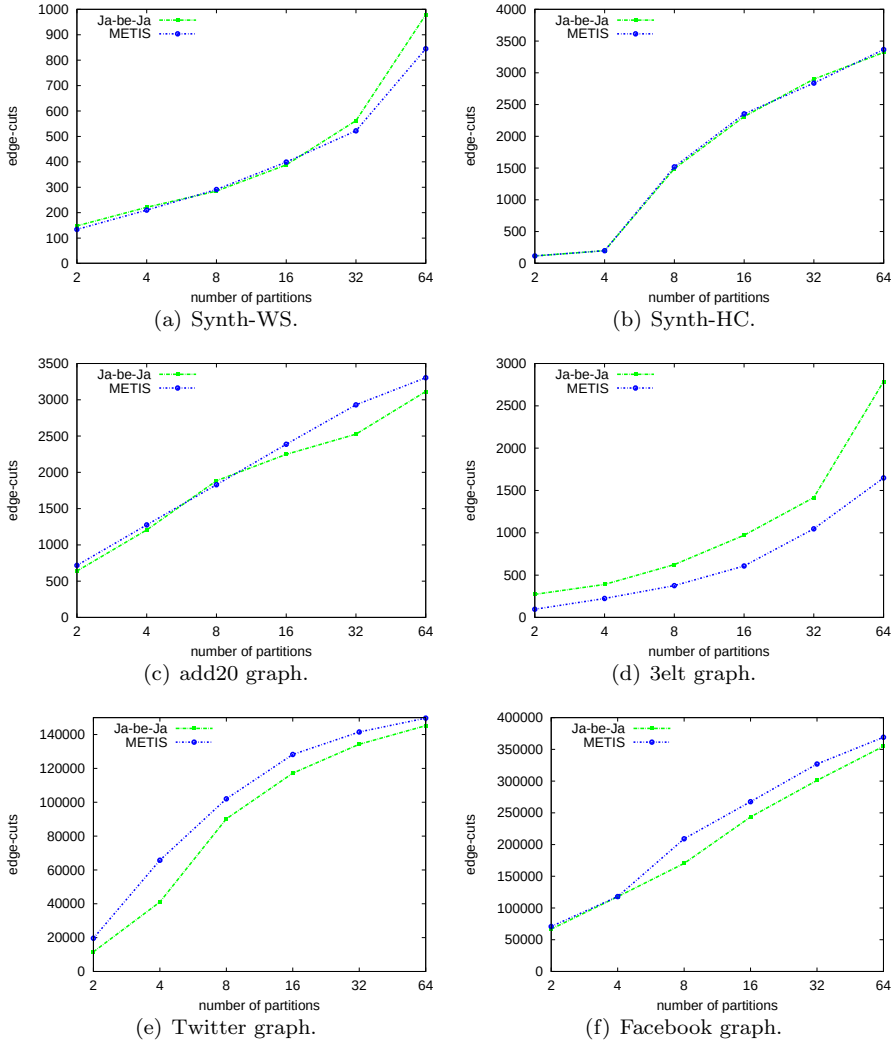


Figure 5: Ja-be-Ja vs. Metis scalability in different graphs.

practical when it comes to very large graphs. Moreover, for Synth-C and Synth-HC, where the optimum edge-cut is available, both Ja-be-Ja and Metis can find the optimal partitioning.

Next, we investigate the performance of the algorithms, in terms of edge-cut, when the number of required partitions grow. Figure 5 shows the resulting edge-cut of Ja-be-Ja versus Metis for 2-64 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 Synth-WS and 3elt), Ja-be-Ja finds a

Table 6: 4-way partitioning with Ja-be-Ja vs. Metis vs. the best known solution.

Graph	Ja-be-Ja	Metis	Best known edge-cut
Synth-R	**2237**	2274	-
Synth-C	**910**	**910**	910 (optimal)
Synth-HC	**198**	**198**	198 (optimal)
Synth-WS	221	**210**	-
Synth-SF	**4169**	4279	-
add20	**1206**	1276	1159 (Probe [8])
data	775	**452**	382 (Mma02 [6])
3elt	390	**224**	201 (Je [33])
4elt	1424	**374**	326 (Nw [38])
vibrobox	23174	**22526**	19098 (Mma02 [6])
Twitter	**41040**	65737	-
Facebook	**117844**	117996	-

better partitioning compared to Metis, when the number of partitions grows. In particular, Ja-be-Ja outperforms Metis in the social network graphs. For example, as shown in Figure 5(f) the edge-cut in Metis is nearly 20,000 more than Ja-be-Ja.

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 and is able to deal with big data, i.e., when the size of the input graph is beyond the resources of a single computer. 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

[1] E.H.L. Aarts and P.J.M. Laarhoven. Simulated annealing: an introduction. *Statistica Neerlandica*, 43(1):31–52, 1989.

[2] R. Albert and A.L. Barabási. Statistical mechanics of complex networks. *Reviews of modern physics*, 74(1):47, 2002.

[3] A. Averbuch and M. Neumann. Partitioning graph databases. Technical report, Royal Institute of Technology, Stockholm, Sweden, 2010.

[4] A. Awan, R.A. Ferreira, S. Jagannathan, and A. Grama. Distributed uniform sampling in unstructured peer-to-peer networks. In *Proc. of HICSS'06*, volume 9, pages 223c–223c. IEEE, 2006.

[5] U. Benlic and J.K. Hao. An effective multilevel tabu search approach for balanced graph partitioning. *Computers & Operations Research*, 38(7):1066–1075, 2011.

[6] U. Benlic and J.K. Hao. A multilevel memetic approach for improving graph k-partitions. *IEEE Tran. Evo. Comp.*, 15(5):624–642, 2011.

[7] P. Chairunnanda, S. Forsyth, and K. Daudjee. Graph data partition models for online social networks. In *Proc. of Hypertext'12*, pages 175–180. ACM, 2012.

[8] P. Chardaire, M. Barake, and G.P. McKeown. A probe-based heuristic for graph partitioning. *IEEE Tran. Comp.*, 56(12):1707–1720, 2007.

[9] D. Dominguez-Sal, P. Urbón-Bayes, A. Giménez-Vañó, S. Gómez-Villamor, N. Martínez-Bazán, and J. Larriba-Pey. Survey of graph database performance on the hpc scalable graph analysis benchmark. *Web-Age Inf. Manag.*, pages 37–48, 2010.

[10] J. Dowling and A.H. Payberah. Shuffling with a croupier: Nat-aware peer-sampling. In *Proc. of ICDCS'12*, pages 102–111. IEEE, 2012.

[11] A.J. Enright, S. Van Dongen, and C.A. Ouzounis. An efficient algorithm for large-scale detection of protein families. *Nucleic acids research*, 30(7):1575–1584, 2002.

[12] W. Galuba, K. Aberer, D. Chakraborty, Z. Despotovic, and W. Kellerer. Outtweeting the twitterers-predicting information cascades in microblogs. In *Proc. of WOSN'10*, pages 3–3. USENIX, 2010.

[13] J. Gehweiler and H. Meyerhenke. A distributed diffusive heuristic for clustering a virtual p2p supercomputer. In *Proc. of IPDPSW'10*, pages 1–8, 2010.

[14] B. Hendrickson. Graph partitioning and parallel solvers: Has the emperor no clothes? In *Proc. of IRREGULAR'98*, pages 218–225. Springer, 1998.

[15] B. Hendrickson and R. Leland. A multilevel algorithm for partitioning graphs. In *Proc. of CDROM'95*, page 28. ACM, 1995.

[16] M. Jelasity, A. Montresor, and O. Babaoglu. Gossip-based aggregation in large dynamic networks. *ACM Trans. Comp. Syst. (TOCS)*, 23(3):219–252, 2005.

[17] G. Karypis and V. Kumar. Parallel multilevel k-way partitioning scheme for irregular graphs. In *Proc. of Supercomputing'96*, pages 35–35. ACM, 1996.

[18] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Jour. Sci. Comp.*, 20(1):359, 1999.

[19] B.W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell System Technical Journal*, 49(2):291–307, 1970.

[20] J. Kleinberg. The small-world phenomenon: an algorithm perspective. In *Proc. of STOC'00*, pages 163–170. ACM, 2000.

[21] M. Kurant, A. Markopoulou, and P. Thiran. On the bias of bfs. *Arxiv preprint arXiv:1004.1729*, 2010.

[22] Y Low, J. Gonzalez, A. Kyrola, D. Bickson, C. Guestrin, and J.M. Hellerstein. Distributed graphlab: A framework for machine learning and data mining in the cloud. *Proc. VLDB Endow.*, 5(8):716–727, 2012.

[23] G. Malewicz, M.H. Austern, A.J.C. Bik, J.C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski. Pregel: a system for large-scale graph processing. In *Proc. of SIGMOD'10*, pages 135–146. ACM, 2010.

[24] L. Massoulié, E. Le Merrer, A.M. Kermarrec, and A. Ganesh. Peer counting and sampling in overlay networks: random walk methods. In *Proc. of PODC'06*, pages 123–132. ACM, 2006.

[25] H. Meyerhenke, B. Monien, and T. Sauerwald. A new diffusion-based multilevel algorithm for computing graph partitions of very high quality. In *Proc. of IPDPS'08*, pages 1–13. IEEE, 2008.

[26] H. Meyerhenke, B. Monien, and S. Schamberger. Graph partitioning and disturbed diffusion. *Parallel Computing*, 35(10-11):544–569, 2009.

[27] A. Montresor and M. Jelasity. Peersim: A scalable p2p simulator. In *Proc. of P2P'09*, pages 99–100. IEEE, 2009.

[28] A.H. Payberah, J. Dowling, and S. Haridi. Gozar: Nat-friendly peer sampling with one-hop distributed nat traversal. In *Proc. of DAIS'11*, pages 1–14. Springer, 2011.

[29] J.M. Pujol, V. Erramilli, G. Siganos, X. Yang, N. Laoutaris, P. Chhabra, and P. Rodriguez. The little engine (s) that could: Scaling online social networks. In *ACM SIGCOMM Comp. Comm.*, volume 40, pages 375–386. ACM, 2010.

[30] L. Ramaswamy, B. Gedik, and L. Liu. A distributed approach to node clustering in decentralized peer-to-peer networks. *IEEE Tran. Par. Dist. Sys.*, 16(9):814–829, 2005.

[31] P. Sanders and C. Schulz. Distributed evolutionary graph partitioning. *Arxiv preprint arXiv:1110.0477*, 2011.

[32] P. Sanders and C. Schulz. Engineering multilevel graph partitioning algorithms. *Algorithms*, pages 469–480, 2011.

[33] A.J. Soper, C. Walshaw, and M. Cross. A combined evolutionary search and multilevel optimisation approach to graph-partitioning. *Journal of Global Optimization*, 29(2):225–241, 2004.

[34] E.G. Talbi. *Metaheuristics: From design to implementation*. Wiley, 2009.

[35] N. Tölgyesi and M. Jelasity. Adaptive peer sampling with newscast. In *Proc. of Euro-Par'09*, pages 523–534. Springer-Verlag, 2009.

[36] B. Viswanath, A. Mislove, M. Cha, and K.P. Gummadi. On the evolution of user interaction in facebook. In *Proc. of WOSN'09*, pages 37–42. ACM, 2009.

[37] S. Voulgaris, D. Gavidia, and M. Van Steen. Cyclon: Inexpensive membership management for unstructured p2p overlays. *Jour. Net. Sys. Manag.*, 13(2):197–217, 2005.

[38] C. Walshaw. Focusware networks mno - a commercialised version of jostle: http://http://focusware.co.uk, Sep 2012.

[39] C. Walshaw. The graph partitioning archive: http://staffweb.cms.gre.ac.uk/˜wc06/partition, Aug 2012.