2013 IEEE 7th International Conference on Self-Adaptive and Self-Organizing Systems

JA-BE-JA: A Distributed Algorithm for Balanced Graph Partitioning

Fatemeh Rahimian†, Amir H. Payberah†, Sarunas Girdzijauskas†, Mark Jelasity‡, Seif Haridi†
†*KTH - Royal Institute of Technology and Swedish Institute of Computer Science (SICS), Sweden*
{*rahimian, payberah, sarunasg, haridi*}*@kth.se*
‡*Hungarian Academy of Sciences and University of Szeged, Hungary, jelasity@inf.u-szeged.hu*

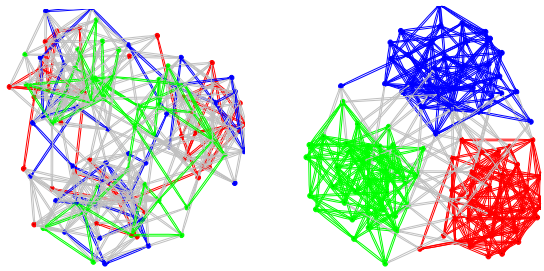
***Abstract*—Balanced graph partitioning is a well known NP-complete problem with a wide range of applications. These applications include many large-scale distributed problems including the optimal storage of large sets of graph-structured data over several hosts—a key problem in today's Cloud infrastructure. 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 graph partitioning. The algorithm is massively parallel: there is no central coordination, each 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 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 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, which makes JA-BE-JA—a bottom-up, self-organizing algorithm—a highly competitive practical solution for graph partitioning.**

***Keywords*-graph partitioning; distributed algorithm; load balancing; simulated annealing;**

I. INTRODUCTION

Every day, petabytes of data are generated and processed in on-line social networking services. Some of this data can be modeled as a graph, in which nodes represent users and edges represent the relationship between them. Similarly, search engines 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. One important problem related to graph-structured data processing is partitioning: extremely large scale graphs must be distributed to hosts in such a way, that most of the adjacent edges are stored on the same host [1].

Finding good partitions is a well-known and well-studied problem in graph theory [2]. 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. Figures 1(a) and 1(b) are examples of a poor and a good partitioning of a graph, respectively.



(a) A poor graph partitioning. 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.

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 nodes and hundreds of billions of 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 [3], [4], [5], [6], [7], [8], [9], [10], 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.

We provide a distributed balanced graph partitioning algorithm, which does not require any global knowledge

978-0-7695-5129-6/13 $26.00 © 2013 IEEE
DOI 10.1109/SASO.2013.13

of the graph topology. That is, we do not have cheap access to the entire graph and we have to process it only with partial information. Our solution, called JA-BE-JA, is a decentralized local search algorithm. 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 increase the number of neighbors they have in the same partition as themselves.

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 [11] or GraphLab [1]. 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, we use multiple datasets of different characteristics, including a few synthetically generated graphs, some graphs that are well-known in the graph partitioning community [12], and some sampled graphs from Facebook [13] and Twitter [14]. We first investigate the impact of different heuristics on the resulting partitioning of the input graphs, and then compare JA-BE-JA to METIS [4], 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 [4].

In the next section we define the exact problem that we are targeting, together with the boundary requirements of the potential applications. In Section III we study the related work of graph partitioning. Then, in Section IV we explain JA-BE-JA in detail, and evaluate it in Section V. Finally, in Section VI we conclude the work.

II. 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.

A. Balanced k-way graph partitioning

We are given an undirected graph $G = (V, E)$, where V is the set of nodes (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 referred to as the *min-cut* problem in graph theory. Balanced (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 $\pi : V \rightarrow \{1, \ldots, k\}$ that assigns a *color* to each node. Hence, $\pi(p)$, or π_p for short, refers to the color of node p. Nodes with the same color form a partition. We denote the set of neighbors of node p by N_p, and define $N_p(c)$ as the set of neighbors of p that have color c:

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

The number of neighbors of node p is denoted by d_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 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:

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

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

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

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

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

B. Data distribution model

We assume that the nodes of the graph are processed periodically and asynchronously, where each node only has access to the state of its immediate neighbors and a small set of random nodes in the graph. 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 [1] or Pregel [11], 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 node. 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.

III. RELATED WORK

A. Graph Partitioning

There exist quite a few works that address the k-way balanced graph partitioning problem in a centralized model. Also, there are partitioning algorithms that have a distributed model similar to that of 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.

B. Balanced Graph Partitioning Algorithms

METIS [4] is a widely known and successful algorithm based on *Multilevel Graph Partitioning (MGP)* [2]. MGP generally works in three phases: (i) a sequence of 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, (ii) the smallest graph is partitioned, and (iii) the partitions are propagated back through a sequence of un-contracting nodes and edges.

Note that 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, e.g., Kernighan-Lin (KL) algorithm [15]. METIS combines several heuristics during its coarsening, partitioning, and uncoarsening phases to improve the cut size. It also uses a greedy refinement 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. [16] proposed an algorithm that combined a Genetic Algorithm (GA) technique with MPG. In [16] crossover and mutation operators are used to compute edge biases, which yield hints for the underlying multilevel graph partitioner. Chardaire et al. [17] also proposed a meta-heuristic, which can be viewed as a GA without selection. Benlic et al. [18] provided a perturbation-based iterated tabu search procedure for partition refinement of each coarsened graph. KAFFPA [10] is another MGP algorithm using local improvement algorithms that are based on flows and localized searches.

In order to speedup the partitioning process for very large-scale graphs, designing algorithms that can be parallelized is inevitable. PARMETIS [5] is the parallel version of METIS that improves the partitioning time, but at the cost of lower quality partitions. KAFFPAE [9] is also a parallelized MGP algorithm, which produces even better partitions compared to its non-parallel ancestor KAFFPA [10]. Moreover, Talbi et al. proposed a parallel graph partitioning technique in [19] based on parallel GA [20]. 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 the memory of a single computer.

C. 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 [21] is a distributed diffusion-based algorithm that eliminates all the global operations for assigning nodes to partitions. Also, CDC [22], 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. We initially carried out experiments with DIDIC for our problem, however had to abandon it since we observed that it tends 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.

IV. SOLUTION

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

A. The basic idea

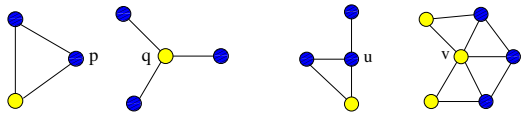
Recall, that we defined the energy of the system as the number of edges between nodes with different colors, and the energy of a node is the number of its neighbors with a different color. The basic idea is to initialize colors uniformly at random, and then to apply heuristic local search to push the 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 cannot change their color independently. Instead, they only *swap*[1] their color with one another. Each node iteratively selects another node from 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 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 [23] as we describe below. Later, in the evaluation section (Section V), 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 V-F, however, we compare our results with the best known partitioning over a number of benchmark problem instances.

[1] JA-BE-JA means swap in Persian.



(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.

B. Swapping: the local search operator

Firstly, a node selects a set of candidate nodes for swapping. We consider three possible ways of selecting the candidate set:

- *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 [24], [25], [26], [27], [28], [29].
- *Hybrid* (H): in this policy first the immediate neighbor nodes are selected (i.e., the local policy). If this selection fails to improve the pair-wise utility, the node is given another chance for improvement, by letting it to select nodes from its random sample (i.e., the random policy).

Secondly, a node needs to define how to select the swap *partner*. The partner of a node p is the node that p chooses among its candidates to exchange its color with. To decide if two nodes should swap their colors, we require: (i) a function to measure the pair-wise utility of a color exchange, and (ii) a policy for escaping local optima.

In order to minimize the edge-cut of the partitioning, we try to maximize $d_p(\pi_p)$ 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 number of neighbors with a similar color to that of the node):

$$d_p(\pi_q)^\alpha + d_q(\pi_p)^\alpha > d_p(\pi_p)^\alpha + d_q(\pi_q)^\alpha \qquad (5)$$

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 change 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 $\alpha = 1$, because $2 + 2 \not> 1 + 3$. However, if $\alpha > 1$, then nodes u and v will exchange their colors. Although, this exchange does not directly reduce the total edge-cut of the graph, it increases the probability of future color exchanges for the two yellow nodes, currently in the neighborhood of node v. In section V we evaluate the effect of the parameter α.

To avoid becoming stuck in a local optimum, we use the well-known Simulated Annealing (SA) technique [23]. We introduce a *temperature* (T) and decrease it over time, similar to the cooling process in [23]. The updated decision criterion becomes

$$(d_p(\pi_q)^\alpha + d_q(\pi_p)^\alpha) \times T > d_p(\pi_p)^\alpha + d_q(\pi_q)^\alpha. \qquad (6)$$

As a result, in the beginning we might move in a direction that degrades the energy function, i.e., nodes 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 SA 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 $T_r = T_{r-1} - \delta$. When the temperature reaches the lower bound 1, it is not decreased anymore. From then on, the decision procedure falls back on using equation (5).

We also use a multi-start search [23], 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, nodes use a gossip-based aggregation method [26] to calculate the edge-cut in the graph. If the new edge-cut is smaller than the previous one, they update the best solution found so far by storing the new edge-cut value together with the current local color.

C. Ja-Be-Ja

JA-BE-JA combines the two aforementioned components: the sampling policy and swapping technique. Algorithm IV-B presents the core of JA-BE-JA. As shown in method `SampleAndSwap`, we use the hybrid heuristic for node selection, which first tries the local policy (line 3), and if it fails it follows the random policy (line 5). Method `FindPartner` shows how the partner is selected. We calculate the two sides of equation (5) in lines $20 - 25$, and in line 26, we compare these computed values. Here, 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 nodes perform a handshake and agree on the swap. This is necessary, because the deciding node might have outdated information about the partner node. During the handshake, the initiating node sends a swap request to the partner node, along with all the information that the partner node needs to verify the swap utility: 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 node ($d_p(\pi_{partner})$). If the verification succeeds, the partner node replies with an acknowledgment (ACK) message and the 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

Algorithm 1 JA-BE-JA Algorithm.

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

- $getNeighbors()$: returns p's neighbors.
- $getSample()$: returns a uniform sample of all the nodes.
- $getDegree(c)$: returns the number of p's neighbors that have color c.

```
1: //Sample and Swap algorithm at node p
2: procedure SAMPLEANDSWAP
3:     partner ← FindPartner(p.getNeighbors(), T_r)
4:     if partner = null then
5:         partner ← FindPartner(p.getSample(), T_r)
6:     end if
7:     if partner ≠ null then
8:         color exchange handshake between p and partner
9:     end if
10:    T_r ← T_r − δ
11:    if T_r < 1 then
12:        T_r ← 1
13:    end if
14: end procedure

15: //Find the best node as swap partner for node p
16: function FINDPARTNER(Node[] nodes, float T_r)
17:    highest ← 0
18:    bestPartner ← null
19:    for q ∈ nodes do
20:        d_pp ← p.getDegree(p.color)
21:        d_qq ← q.getDegree(q.color)
22:        old ← d_pp^α + d_qq^α
23:        d_pq ← p.getDegree(q.color)
24:        d_qp ← q.getDegree(p.color)
25:        new ← d_pq^α + d_qp^α
26:        if (new × T_r > old) ∧ (new > higest) then
27:            bestPartnere ← q
28:            highest ← new
29:        end if
30:    end for
31:    return bestPartner
32: end function
```

parallel, and when no more swaps take place in the graph, the algorithm has converged.

In the one-host-multiple-nodes model, the only change required to the core algorithm is to give preference to local host swaps. That is, if there are several nodes as potential partners for a swap, the node 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 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. In Section V-E, we study the trade-off between communication overhead and the edge-cut with and without considering the locality.

D. Generalizations of Ja-Be-Ja

So far, we have discussed the case when the graph links 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 links are often weighted. For example, in a graph database some operations are performed more frequently, thus, some links are accessed more often [30]. In order to prioritize such links when partitioning the graph, we change the definition of d_p, such that, instead of just counting the number of neighboring nodes with the same color, we sum the weights of these links:

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

where $w(p,q)$ is 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.

V. EXPERIMENTAL EVALUATION

We implemented JA-BE-JA on PEERSIM [31], 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 [4], a well-known efficient centralized solution, and (ii) the best known available results from the Walshaw benchmark [12] 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, and parameter α is set to 2.

A. Metrics

Although the most important metric for graph partitioning is edge-cut (or energy), there are a number of studies [32] 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 [7], [8], 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 nodes stored on the same host are not counted).

Table I
DATASETS

Dataset	|V|	|E|	Type	Reference
Synth-WS	1000	4147	Synth.	-
Synth-SF	1000	7936	Synth.	-
add20	2395	7462	Walshaw	[12]
data	2851	15093	Walshaw	[12]
3elt	4720	13722	Walshaw	[12]
4elt	15606	45878	Walshaw	[12]
vibrobox	12328	165250	Walshaw	[12]
Twitter	2731	164629	Social	[14]
Facebook	63731	817090	Social	[13]

- data migration: the number of nodes that need to be migrated from their initial partition to their final partition.

While 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 nodes have to migrate from one physical machine to another after finding the final partitioning. If the graph nodes that are initially located at a given host get the same initial color, then this metric is given by the number of nodes that end up with a different color by the time the algorithm has converged.

B. Datasets

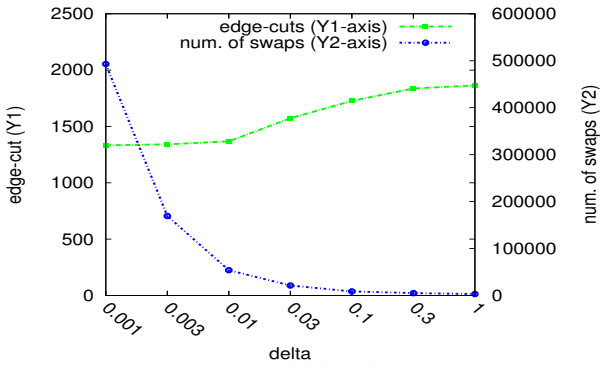
We have used three types of graphs: (i) two synthetically generated graphs, (ii) several graphs from Walshaw archive [12], and (iii) sampled graphs from two well-known social networks: Twitter [14] and Facebook [13]. These graphs are listed in Table I.

Synthetic Graphs.: We generated two different graphs synthetically. The first one is based on the Watts-Strogatz model [33], with 1000 nodes and average degree 8 per node. 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 Barabasi-Albert model [34] of growing scale free networks. This graph also includes 1000 nodes with an average degree of 16. Both graphs are undirected and there are no parallel edges either.

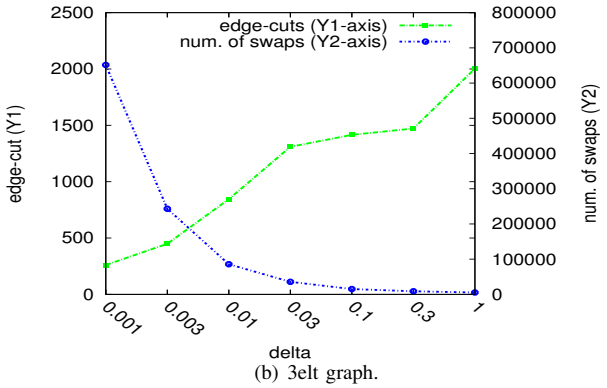
The Walshaw Archive.: The Walshaw archive [12] consists of the best partitioning found to date for a set of graphs, and reports the partitioning algorithms that achieved those best results. This archive, which has been active since the year 2000, includes the results from most of the major graph partitioning 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.

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.

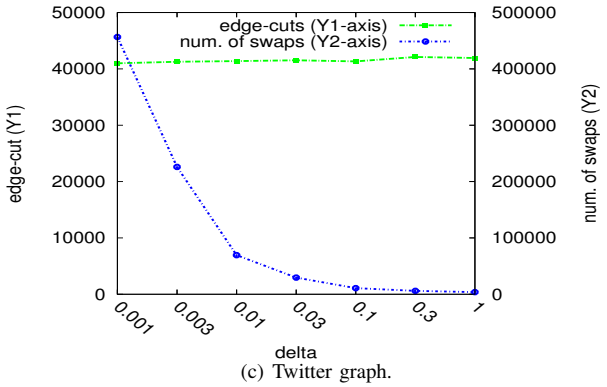
We sampled our Twitter graph from the follower network of 2.4 million Twitter users [14]. There are several known



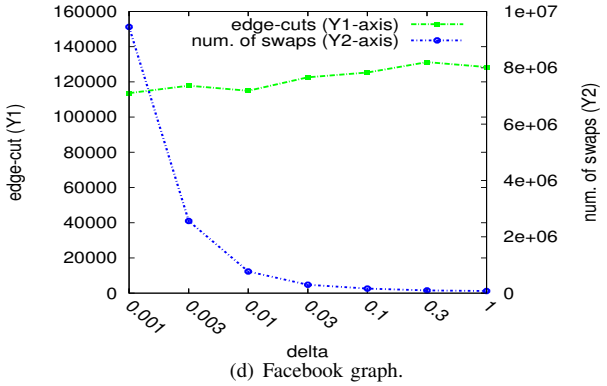
(a) add20 graph.



(b) 3elt graph.



(c) Twitter graph.



(d) Facebook graph.

Figure 3. The number of swaps and edge-cut over δ.

Table II
DIFFERENT SAMPLING HEURISTICS, WITH $\alpha = 2$ AND SA.

Graph	initial	L	R	H
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**

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 [35] sampling nearly 10000 nodes by performing multiple breadth first searches (BFS). We also used a sample graph of Facebook, which is made available by Viswanath et. al. [13]. 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.

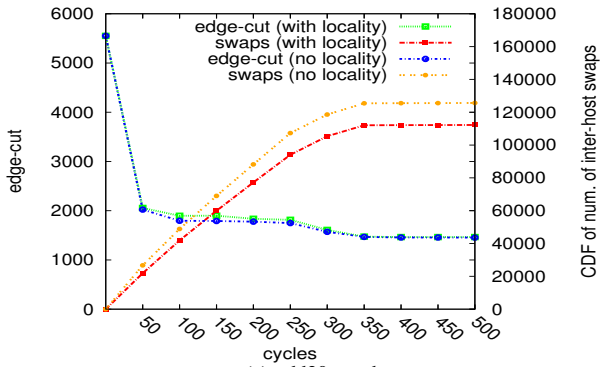
C. The impact of the sampling policies

In this section, we study the effect of different sampling heuristics on the edge-cut. These heuristics were introduced in Section IV-B 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 [26], [36] 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 node, the edge-cut is reduced to 30% of the initial number for the Facebook graph. The 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 optimum. The best result for most graphs, however, is achieved with the combination of L and R: the hybrid heuristic (H).

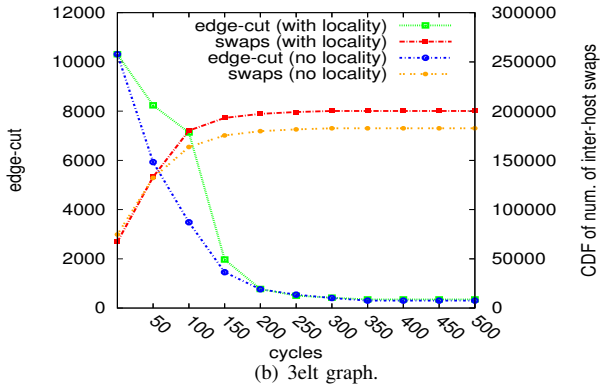
D. The impact of the swapping policies

In these experiments, we study the effect of the parameters that define the swapping policies, introduced in Section IV-B. 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 (5). The setting $\alpha = 2$ gives the best result for most of the graphs. In Section IV-B we explained why α is better to be greater than 1. In this experiment, we observe that if α is set too high, nodes might overestimate the value of a swap and end up in an inferior state.

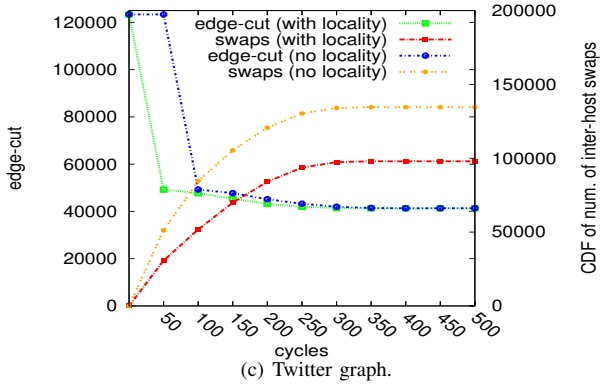
Table IV lists the edge-cut with and without simulated annealing (SA). In the simulations without SA, we set $T_0 = 1$, which is the lowest allowed temperature in our case (see equation (6)). Although the improvements due to SA might be minor for some graphs, for other graphs with various



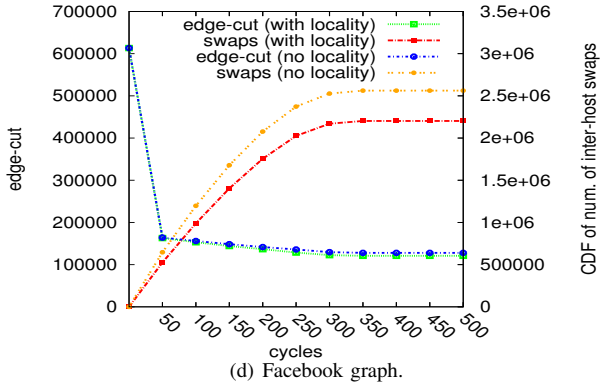
(a) add20 graph.



(b) 3elt graph.



(c) Twitter graph.



(d) Facebook graph.

Figure 4. Evolution of edge-cut and swaps over time.

Table III
TUNING α, WITH HYBRID SAMPLING AND SA.

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

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. These experiments are not reported due to lack of space.

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 3 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.

E. 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 nodes within the same host are not counted. We assume there are four hosts in the systems, where each host gets a random 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 occur (convergence). As shown in Figure 4, 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 3(c) and 3(d)), we have performed these experiments with the same setting of $\delta = 0.003$ for all the graphs. As shown in Figure 4(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 different partitionings of the graph with a similar edge-cut value, thus, local swaps will be more likely to be good enough.

Table IV
THE IMPACT OF SA ON THE EDGE-CUT ($\alpha = 2$).

Graph	initial	H	$H + SA$
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**

Table V
THE NUMBER OF NODES THAT NEED TO MIGRATE.

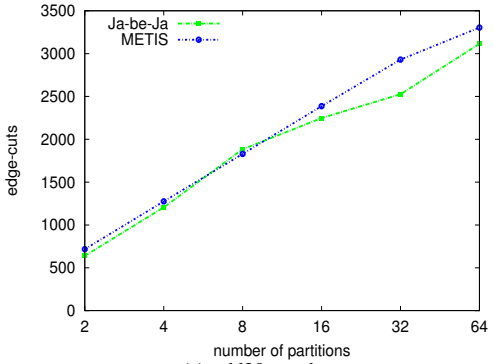
graph	$\|V\|$	$\|mig\|$
Synth-WS	1000	720
add20	2395	1740
3elt	4720	3436
Twitter	2731	2000
Facebook	63731	47555

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 nodes are initialized with the same color. During run-time, only the color labels are exchanged. The color of a node may change several times before convergence. When the algorithm converges, each data item (node) is migrated from its initial partition to its final partition indicated by its color.

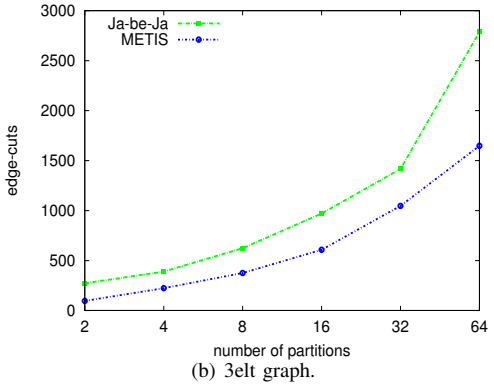
Note that migration could be optimized given the final partitioning, but we simply assume that nodes with a color different from the original color will migrate. Table V shows the number of data items that need to be migrated after the convergence of the algorithm. As expected, this number constitutes nearly 75% of the 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.

F. Comparison With the State-of-the-Art

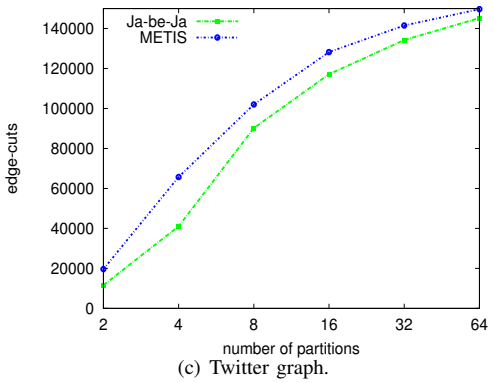
In this section, we compare JA-BE-JA to METIS [4] on all the input graphs. We also compare these results to the best known solutions for the graphs from the Walshaw benchmark [12]. Table VI shows the edge-cut of the final 4-way partitioning. As shown, for some graphs, METIS produces better results, and 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.



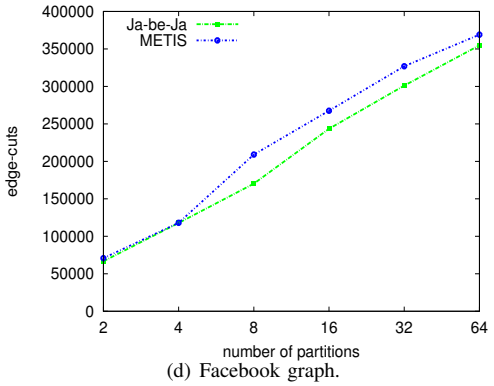
(a) add20 graph.



(b) 3elt graph.



(c) Twitter graph.



(d) Facebook graph.

Figure 5. JA-BE-JA vs. METIS scalability in different graphs.

Table VI
JA-BE-JA VS. METIS VS. THE BEST KNOWN EDGE-CUT.

Graph	JA-BE-JA	METIS	Best known edge-cut
Synth-WS	221	**210**	-
Synth-SF	**4169**	4279	-
add20	**1206**	1276	1159 (PROBE [17])
data	775	**452**	382 (MMA02 [37])
3elt	390	**224**	201 (JE [16])
4elt	1424	**374**	326 (NW [38])
vibrobox	23174	**22526**	19098 (MMA02 [37])
Twitter	**41040**	65737	-
Facebook	**117844**	117996	-

Next, we investigate the performance of the algorithms, in terms of edge-cut, when the number of the required partitions grows. Figure 5 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 number of partitions grows. In particular, JA-BE-JA outperforms METIS in the social network graphs. For example, as shown in Figure 5(d) the edge-cut in METIS is nearly 20,000 more than JA-BE-JA. Note, unlike METIS, JA-BE-JA does not make use of any global information or operation over the entire graph.

VI. 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.

ACKNOWLEDGMENT

M. Jelasity was supported by the Bolyai Scholarship of the Hungarian Academy of Sciences. This work was partially supported by the European Union and the European Social Fund through project FuturICT.hu (grant no.: TAMOP-4.2.2.C-11/1/KONV-2012-0013).

REFERENCES

[1] Y. Low, J. Gonzalez, A. Kyrola, D. Bickson, C. Guestrin, and J. Hellerstein, "Distributed graphlab: A framework for machine learning and data mining in the cloud," *Proc. VLDB Endow.*, vol. 5, no. 8, pp. 716–727, 2012.

[2] B. Hendrickson and R. Leland, "A multilevel algorithm for partitioning graphs," in *Proc. of CDROM'95*. ACM, 1995, p. 28.

[3] A. Enright, S. Van Dongen, and C. Ouzounis, "An efficient algorithm for large-scale detection of protein families," *Nucleic acids research*, vol. 30, no. 7, pp. 1575–1584, 2002.

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

[5] ——, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM Jour. Sci. Comp.*, vol. 20, no. 1, p. 359, 1999.

[6] B. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *Bell System Technical Journal*, vol. 49, no. 2, pp. 291–307, 1970.

[7] 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*. IEEE, 2008, pp. 1–13.

[8] H. Meyerhenke, B. Monien, and S. Schamberger, "Graph partitioning and disturbed diffusion," *Parallel Computing*, vol. 35, no. 10-11, pp. 544–569, 2009.

[9] P. Sanders and C. Schulz, "Distributed Evolutionary Graph Partitioning," in *ALENEX*, 2012, pp. 16–29.

[10] ——, "Engineering Multilevel Graph Partitioning Algorithms," in *Proceedings of the 19th European Symposium on Algorithms*, ser. LNCS, vol. 6942, 2011, pp. 469–480.

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

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

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

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

[15] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *The Bell system technical journal*, vol. 49, no. 1, pp. 291–307, 1970.

[16] A. Soper, C. Walshaw, and M. Cross, "A combined evolutionary search and multilevel optimisation approach to graph-partitioning," *Journal of Global Optimization*, vol. 29, no. 2, pp. 225–241, 2004.

[17] P. Chardaire, M. Barake, and G. McKeown, "A probe-based heuristic for graph partitioning," *IEEE Tran. Comp.*, vol. 56, no. 12, pp. 1707–1720, 2007.

[18] U. Benlic and J. Hao, "An effective multilevel tabu search approach for balanced graph partitioning," *Computers & Operations Research*, vol. 38, no. 7, pp. 1066–1075, 2011.

[19] E. Talbi and P. Bessiere, "A parallel genetic algorithm for the graph partitioning problem," in *Proc. of ICS'91*. ACM, 1991, pp. 312–320.

[20] G. Luque and E. Alba, *Parallel Genetic Algorithms: Theory and Real World Applications*. Springer, 2011, vol. 367.

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

[22] L. Ramaswamy, B. Gedik, and L. Liu, "A distributed approach to node clustering in decentralized peer-to-peer networks," *IEEE Tran. Par. Dist. Sys.*, vol. 16, no. 9, pp. 814–829, 2005.

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

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

[25] J. Dowling and A. Payberah, "Shuffling with a croupier: Nat-aware peer-sampling," in *Proc. of ICDCS'12*. IEEE, 2012, pp. 102–111.

[26] M. Jelasity, A. Montresor, and O. Babaoglu, "Gossip-based aggregation in large dynamic networks," *ACM Trans. Comp. Syst. (TOCS)*, vol. 23, no. 3, pp. 219–252, 2005.

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

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

[29] S. Voulgaris, D. Gavidia, and M. Van Steen, "Cyclon: Inexpensive membership management for unstructured p2p overlays," *Jour. Net. Sys. Manag.*, vol. 13, no. 2, pp. 197–217, 2005.

[30] 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.*, pp. 37–48, 2010.

[31] A. Montresor and M. Jelasity, "Peersim: A scalable p2p simulator," in *Proc. of P2P'09*. IEEE, 2009, pp. 99–100.

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

[33] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'small-world' networks," *Nature*, vol. 393, pp. 440–442, 1998.

[34] R. Albert and A. Barabási, "Statistical mechanics of complex networks," *Reviews of modern physics*, vol. 74, no. 1, p. 47, 2002.

[35] M. Kurant, A. Markopoulou, and P. Thiran, "On the bias of bfs (breadth first search)," in *Teletraffic Congress (ITC), 2010 22nd International*. IEEE, 2010, pp. 1–8.

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

[37] U. Benlic and J. Hao, "A multilevel memetic approach for improving graph k-partitions," *IEEE Tran. Evo. Comp.*, vol. 15, no. 5, pp. 624–642, 2011.

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