A Distributed GPU-based Correlation Clustering Algorithm for Large-scale Signed Social Networks

Mario Levorato[1], **Lúcia Drummond**[1], **Rosa Figueiredo**[2], **Yuri Frota**[1]

[1]Instituto de Computação
Universidade Federal Fluminense (UFF) – Niterói, RJ – Brasil

[2]Laboratoire d'Informatique d'Avignon
Université d'Avignon – Avignon, France

{mlevorato,lucia,yuri}@ic.uff.br, rosa.figueiredo@univ-avignon.fr

Abstract. *When applied to signed networks, the Correlation Clustering (CC) problem consists of an important tool to study how balanced a social group behaves and if this group might evolve to a possible balanced state. Solving such combinatorial optimization problem is a challenging task, which heavily relies on heuristic procedures, one of the few solution methods capable of analyzing large network instances. In this work, we present a novel approach to solve the CC problem on large-scale signed networks. A distributed GPU-powered version of the ILS metaheuristic, which benefits from data parallelism, has been developed. This technique provides good quality clustering results when compared to non-distributed methods. Experiments were conducted on both synthetic and real datasets. The proposed algorithm achieved improved solution values when compared to the existing parallel solution method. In particular, one of the largest graphs we have considered in our experiments contains 1 million nodes and 8 million edges – such graph can be clustered in two hours using our algorithm. The new method can process networks for which there is no efficient solution using the existing algorithms found in the literature.*

1. Introduction

A group of individuals can often be viewed as social networks, where vertices represent individuals and edges their relations. Relationships in social networks can have more than two status like presence or absence of a trust/friendship between two individuals. There may be negative links like distrust or dislike. Signed networks are used for this purpose and, within these relationships, communities are formed.

Community detection and clustering on signed networks can provide significant insights into understanding group interactions in social systems and further deducing how a social system evolves and if (or when) the system will reach balanced or relative stable status. Knowledge about communities also allows us to better understand and analyze behaviors of users inside communities, as well as their formation and disintegration.

All these aspects are part of an important theory called social balance or structural balance, originated from early studies of [Heider 1946] and later expanded by [Cartwright and Harary 1956] and [Davis 1967]. The basic ideas underlying structural balance are commonly represented with the aphorisms: "my friend's friend is my friend, my friend's enemy is my enemy, my enemy's friend is my enemy, my enemy's enemy is my friend [Aronson and Cope 1968, Schwartz 2010]. Structural balance theory affirms that human societies tend to avoid tension and conflictual relations [Facchetti et al. 2011, Srinivasan 2011]. In a signed graph which represents a social network, this translates into a level of balance greater than expected, if compared to a random signed graph of equivalent size [Facchetti et al. 2011], and such measure can help us understand interesting social phenomena, like alliances and disputes among parties

or nations [Macon et al. 2012, Doreian and Mrvar 2015] and the social situation where polarization is frequent, like national elections [Srinivasan 2011].

We study the Correlation Clustering (CC) problem applied to measuring structural balance in signed social networks. The CC problem can be stated as: given n objects where certain pairs of objects are labeled as similar and other pairs as dissimilar, find a clustering which maximizes the number of similar pairs within clusters, plus the number of dissimilar pairs between clusters. In fact, the CC problem is not only useful in social network analysis, but also in other research areas: efficient document classification [Bansal et al. 2002], detection of embedded matrix structures [Gülpinar et al. 2004], biological systems [DasGupta et al. 2007], grouping of genes [Bhattacharya and De 2008], community structure [Macon et al. 2012], portfolio analysis in risk management [Huffner et al. 2009] and image segmentation [Kim et al. 2014].

Even though large-scale unsigned social networks (like the ones built from Facebook and Twitter) have been extensively studied [Duch and Arenas 2005, Newman 2006, Brandes et al. 2008], only a few datasets of relatively large signed social networks were released [Kunegis 2013, Leskovec and Krevl 2014]. While these graphs tend to follow a power-law distribution, denser graphs can be obtained by generating edges with similarity measures between all pairs of elements, based on datasets of ratings or voting information, like Movielens [GroupLens 2017] and the European Parliament [Mendonça et al. 2015]. Though these networks may contain less vertices than typical world-scale social networks, they can be harder to analyze, for their high edge density.

Measuring structural balance in large signed networks is complex and time consuming. As other real-world optimization problems, the CC problem is NP-hard [Bansal et al. 2002] and the number of possible clustering configurations is exponential. To tackle this challenge, we first designed parallel master-slave algorithms based on Greedy Randomized Adaptive Search Procedure (GRASP) [Feo and Resende 1995] and Iterated Local Search (ILS) [Lourenço et al. 2003] metaheuristics for the CC problem [Drummond et al. 2013, Levorato et al. 2015b], which outperformed the previous solution methods with similar or improved solution quality. We then developed a parallel local search procedure for the CC problem, accelerated by General Purpose Graphics Processing Units (GPGPUs) [Levorato et al. 2015a]. When dense graphs need to be processed, this procedure reduces the complexity of the local search step of both GRASP and ILS, by analyzing several neighborhood movements in parallel. Recently, in [Levorato et al. 2017], the parallel ILS algorithm for the CC problem was used to cluster the large real-world social networks provided by [Leskovec and Krevl 2014], with up to 10^5 vertices and 10^6 edges.

The need to efficiently process larger and denser networks has motivated us to develop a new fully distributed algorithm, based on heterogeneous computing, using multiple processor types (CPUs and GPUs), task parallelism and distributed data parallelism at the same time. However, clustering signed graphs which are distributed over several compute nodes consists of a challenging subject, since the local search procedures and heuristics in the literature are inherently sequential, as they need to be aware of the whole graph at each step. Also, communication between machines can quickly become a performance bottleneck in many graph applications. We applied dynamic vertex repartitioning during the execution of the distributed algorithm, which demands extra communication required for reassigning data between compute nodes. Besides the local search in the optimization algorithm, dynamic repartitioning is also used as an adaptive load balancing technique, so as to avoid overload of a specific processing node.

In this paper, we propose a distributed optimization algorithm for solving the CC problem on large-scale signed graphs, whose processing would be impossible on a single machine. To the best of our knowledge, since this is the first work on distributed metaheuristics applied to a clustering problem, we do not compare to other distributed techniques as there is no existing comparable technique. As seen in the next section, the nearest works are based on a variation of the CC problem on complete graphs, whose input data and results are not directly comparable. We conduct evaluations on synthetic datasets and real networks. The solution values of the distributed CC algorithm were then compared with the same base metaheuristic (ILS-CC) described in [Levorato et al. 2017], running as a parallel program. Results show the effectiveness of the proposed method.

2. Related work

Correlation clustering was formalized for the first time by [Bansal et al. 2002]. In the general case, the problem of maximizing agreements (minimizing disagreements) is NP-hard and APX-hard (hard to approximate within an arbitrarily small constant) [Bansal et al. 2002, Charikar et al. 2003]. Two variations of the CC problem are known. It can be computed on complete graphs (i.e. all edges are present and all weights are ± 1), or on general graphs (arbitrary edge weights) – the problem studied in this work. Since these variations of the problem require distinct types of input graphs, their solutions cannot be directly compared.

When applied to complete unweighted graphs, the CC problem can be solved via approximation algorithms. [Chierichetti et al. 2014] proposed a parallel 3-approximation algorithm to the optimal CC on complete graphs, which can be implemented in a distributed framework such as MapReduce and scales to huge datasets like Twitter (41M nodes, 2.5B positive edges and 2.9M maximum degree). [Pan et al. 2015] developed parallel CC algorithms with 3-approximation factor, that, although not distributed, can scale to billion-edge graphs, returning a valid solution in less than five seconds.

In this work, we consider the CC problem on general signed graphs. In this case, Integer Linear Programming (ILP) can be used to solve the CC problem optimally, but only when the number of data points is small. For its complexity, the only available solutions for large instances are either heuristic or approximate. The best known approximation ratio for the CC problem is $O(logn)$. [Bansal et al. 2002] proposed two approximation algorithms: one to maximize "agreements" (the number of positive within clusters and negative edges between clusters) and another to minimize disagreements. An approximation algorithm based on rounding a linear program is provided by [Demaine et al. 2006]. [Ailon et al. 2008] proposed a constant factor 2.5 approximation for disagreement minimization (the best known factor so far). Greedy neighborhood-based heuristics for the problem were proposed by [Elsner and Schudy 2009] and [Wang and Li 2013], while in [Yang et al. 2007], the CC problem is known as *community mining* and an agent-based heuristic called FEC is proposed to its solution. A genetic algorithm applied to document clustering has also used the CC problem as objective function [Zhang et al. 2008].

After developing parallel GRASP [Drummond et al. 2013] and ILS [Levorato et al. 2015b] metaheuristics for the CC problem, in [Levorato et al. 2017], we presented a thorough analysis of the sequential and parallel ILS algorithms, in comparison with the aforementioned CC solution approaches proposed in the literature. When a direct comparison was possible, the results evidentiated the superiority of the ILS-CC algorithm, which presented similar or improved solution quality.

3. A Distributed algorithm for solving the Correlation Clustering problem

When it comes to signed social networks, the instances generated from Wikipedia, Slashdot and Epinions websites, available in [Leskovec and Krevl 2014], have thousands of nodes and in some cases almost a million relationships[1]. In order to scale the ILS procedure to solve larger network instances, we extended the existing ILS procedure for the CC problem [Levorato et al. 2015b] to explore the natural data parallelism present in clustering problems: the graph can be split into smaller (non-overlapping) subgraphs and ILS can be used to obtain a clustering solution for each subgraph. The basic idea is to use distributed computing so that each node runs the optimization algorithm over a subset of vertices and then combine all partial solutions (and their respective clustering) into a global solution for the whole network.

3.1. Preliminaries

Let $G = (V, E)$ be an undirected signed graph where V is the set of n vertices and E is the set of edges, where each edge has weight $w_e \geq 0$. Let $s(e)$ denote the label ($\langle - \rangle$, $\langle + \rangle$) of the edge e. Let E^- and E^+ denote, respectively, the set of negative and positive edges in G. Note that the terminology "positive" and "negative" refers to the edge label and not the weight; edge weights are always nonnegative regardless of the label.

For a vertex set $S \subseteq V$, let $E[S] = \{(i, j) \in E \mid i, j \in S\}$ denote the *subset of edges induced by* S. For two vertex sets $S, W \subseteq V$, let $E[S : W] = \{(i, j) \in E \mid (i \in S, j \in W) \vee (i \in W, j \in S)\}$ denote the *subset of edges that connect vertices from the clusters* S *and* W. Given a clustering $\mathcal{C} = \{S_1, S_2, \ldots, S_k\}$, for $1 \leq i, j \leq k$, let

$$\Omega^+(S_i, S_j) = \sum_{e \in E^+ \cap E[S_i:S_j]} w_e \text{ and } \Omega^-(S_i, S_j) = \sum_{e \in E^- \cap E[S_i:S_j]} w_e.$$

We call an edge $e = (u, v)$ a positive mistake if $s(e) = \langle + \rangle$ and $e \in E[S_i : S_j] : S_i, S_j \in \mathcal{C}, i \neq j$. We call an edge $e = (u, v)$ a negative mistake if $s(e) = \langle - \rangle$ and $e \in E[S_i : S_i]$ for some $S_i \in \mathcal{C}$. The number of mistakes or *imbalance* of a clustering $[I(\mathcal{C})]$, is given by the sum of positive and negative mistakes or, in other words, the weighted sum of unrelated pairs that are clustered together, in addition to the weighted sum of related pairs that are separate.

$$I(\mathcal{C}) = \sum_{1 \leq i \leq k} \Omega^-(S_i, S_i) + \sum_{1 \leq i < j \leq k} \Omega^+(S_i, S_j). \quad (1)$$

That being said, a formal definition to the CC problem can be provided.

Problem 3.1 (CC problem) *Let $G = (V, E)$ be a signed graph and w_e be a nonnegative edge weight associated with each edge $e \in E$. The correlation clustering problem is the problem of finding a clustering $\mathcal{C}$ of V such that the imbalance $I(\mathcal{C})$ is minimized.*

A *partition* of V is a division of V into non-overlapping and non-empty subsets and a graph $G' = (V', E')$ is called a *subgraph* of a graph $G = (V, E)$ if $V' \subseteq V$ and $E' \subseteq E$. G' is called an *induced subgraph*, or the *subgraph induced* by V', if E' consists of all edges of G spanned by V'.

Now suppose the set V of vertices of G is *partitioned* into q subsets, one for each process running in parallel, such that $\Phi = \{V_1, V_2, \ldots, V_q\}$ denotes a *vertex-process par-*

[1]Epinions signed social network has 131828 vertices and 711210 edges.

tition of V. This way, $\forall i \in \{1, \ldots, q\}$, each process q_i will be in charge of manipulating a subgraph G'_i, induced by V_i.

3.2. Distributed GPU-powered Iterated Local Search (ILS) algorithm for the CC Problem

The Iterated Local Search [Lourenço et al. 2003] (ILS) is a metaheuristic that explores a sequence of solutions created by perturbations of the current best solution and then refines these solutions to their local optima using an embedded heuristic. According to our tests, within a 2-hour time limit, the multistart ILS procedure ($ILSMultiStartCC$) from [Levorato et al. 2015b], embedded in our distributed algorithm, is capable of efficiently solving the CC problem on real-world networks of size up to $n = 80,000$ and edge density $d \leq 0.02\%$. To process larger graphs, we opted to develop a distributed algorithm that splits large-scale network instances into smaller subgraphs, delegating the solution of the CC problem on each subgraph to a specific process. Therefore, all processes execute the same code (the $ILSMultiStartCC$ procedure) over a different subgraph.

Message passing was used for communication among processes. When q processors are used, a master process reads the problem data, divides the graph into smaller parts and passes them to the remaining $q - 1$ processes. Each process executes a copy of ILS over a specific subgraph it was assigned to, according to the vertex partitioning (each vertex subset goes to a process running on a separate CPU). Finally, the master process is in charge of gathering each search result produced in parallel and merging them to a global solution for the whole graph.

A second parallelization strategy used in this algorithm was applied in the local search phase (inside the $ILSMultiStartCC$ procedure), using parallelism to evaluate hundreds of vertex movements at once with the help of General Purpose Graphics Processing Units (GPGPUs). It consists of a parallel local search procedure, known as CUDA-VND [Levorato et al. 2015a], which outperforms the previous local search procedure (which used sequential code) in execution time, presenting similar solution quality.

We also applied distributed data parallelism to our algorithm by the use of special data structures from the Boost Parallel Graph Library [Gregor and Lumsdaine 2005]. Distributing graph vertices and edges between several compute nodes allowed the processing of extremely large graphs which would not fit into a single machine. This allows the algorithm to efficiently scale to graphs in the order of 10^6 nodes and number of edges above 10^9, without the limitations of a single shared memory address space.

The distributed algorithm includes an initial distribution of vertices among processes (Algorithm 1) and an optimization algorithm executed in a distributed fashion with a load balance mechanism, which also includes termination detection (Algorithms 2 and 3).

3.2.1. Initial distribution

The signed graph is split according to the number of vertices $n = |V|$ and the number of processes running in parallel (q parameter). Each process will be initially in charge of finding a solution for the subgraph induced by the set V_i, where $|V_i| = |V| / q$.

The $SplitGraphBetweenProcesses$ procedure (Algorithm 1) generates an initial vertex-process partition Φ, comprised of q subsets, from V_1 to V_q, and delegates to each process q_j the task of finding the solution of the CC problem on the corresponding subgraph induced by V_j. Let R be the set of vertices not yet associated to any process. For each process q_j, the procedure initially adds to the subset V_j the vertex $v \in R$ with the smallest negative-edge sum considering the residual subgraph induced by set R (line 4).

Then, at each step, the vertex v_{max} is added to the referred partition. This vertex, which has not been previously inserted in V_j, is chosen so that it presents the maximum cardinality of positive edges between itself and V_j (line 6). In other words, this criterion tends to minimize the number of mistakes of the initial partition, since vertices with higher affinity (i.e. more positive edges) will be included in the same subgraph (same process). At the end, each subset V_j in the initial partition Φ is forwarded to the corresponding process (line 10).

Algorithm 1: $SplitGraphBetweenProcesses$ (master procedure)

Variables: $G = (V, E)$ and number of processes q

1 $R \leftarrow V$;
2 $\overline{\mathcal{C}} \leftarrow \emptyset, \Phi \leftarrow \emptyset$;
3 **for** *each process* q_j *such that* $1 \leq j \leq q$ **do**
4 $\quad V_j \leftarrow \{\arg\min\{\Omega^-(R, \{v\}) \mid v \in R\}\}$;
5 $\quad$ **while** $\left(|V_j| < \frac{|V|}{q}\right)$ **do**
6 $\quad\quad v_{max} \leftarrow \arg\max\{\Omega^+(V_j, \{v\}) \mid v \in (R \setminus V_j)\}$;
7 $\quad\quad V_j \leftarrow V_j \cup \{v_{max}\}$;
8 $\quad R \leftarrow R \setminus V_j$;
9 $\quad \Phi \leftarrow \Phi \cup V_j$;
10 $\quad$ **send message** $(\langle InitialDistribution \rangle, V_j)$ to process q_j;

In a second version of the initial distribution, Algorithm 1 was replaced by a uniformly random distribution of vertices between the processes, which scales to huge graph instances. We applied such distribution approach only when the graph was too big for a single machine's memory. As seen in the next section, experiments performed with the most dense graph instances confirmed the solution quality of our method.

3.2.2. Distributed optimization and load balance

Algorithm 2: $DistributedILS$ (master procedure)

Variables: $G = (V, E)$, vertex-process partition Φ, clustering $\mathcal{C}$, number of processes q

1 $r \leftarrow 1$, $\mathcal{C} \leftarrow \emptyset$, $I(\mathcal{C}) \leftarrow \infty$;
2 **while** $r \leq 4$ and time limit not exceeded **do**
3 $\quad \overline{\Phi} \leftarrow \Phi$;
4 $\quad$ `UpdatePartition` ($\overline{\Phi}$, *MovementType(r)*);
5 $\quad$ **for** *each process* q_j *where* $1 \leq j \leq q$ **do**
6 $\quad\quad$ **send message** $(\langle MovementType(r) \rangle, \overline{\Phi})$ to process q_j;
7 $\quad \overline{\mathcal{C}} \leftarrow \emptyset$;
8 $\quad$ **for** *each process* q_j *where* $1 \leq j \leq q$ **do**
9 $\quad\quad$ **receive message** $(\langle ILSResult \rangle, \mathcal{C}_j)$ from process q_j;
10 $\quad\quad \overline{\mathcal{C}} \leftarrow \overline{\mathcal{C}} \cup \mathcal{C}_j$;
11 $\quad$ **if** $I(\overline{\mathcal{C}}) < I(\mathcal{C})$ **then**
12 $\quad\quad \mathcal{C} \leftarrow \overline{\mathcal{C}}$, $\Phi \leftarrow \overline{\Phi}, r \leftarrow 1$;
13 $\quad$ **else**
14 $\quad\quad r \leftarrow r + 1$; `// Go to the next VND neighborhood`
15 **return** $\Phi, \mathcal{C}$;

After the initial distribution of vertices, the master executes $DistributedILS$ (Algorithm 2), which is based on the Variable Neighborhood Descent (VND) [Mladenović and Hansen 1997] procedure. It is in charge of exploring four different distributed neighborhood structures (vertex/cluster movements between process partitions), listed in Figure 1. The master generates new vertex partitionings Φ, distributed among the processes, allowing that a better global solution be reached if and only if an improvement of imbalance for the whole graph is obtained. The master procedure

sends, to each worker process q_j, the requested neighborhood movement type and the new vertex-process partition $\overline{\Phi}$. After receiving the processing results, the individual CC solution of each modified subgraph ($\mathcal{C}_j$) is then merged into a global solution $\overline{\mathcal{C}}$ for the whole graph (lines 9-10) and the corresponding global imbalance $I(\overline{\mathcal{C}})$ is tested for improvement (line 11). If this is the case, the new incumbent is updated and r is returned to its initial value (line 12). Otherwise, the next neighborhood is considered (line 14). The algorithm halts when no better clustering solution is found in the most distant neighborhood of the current best solution ($\mathcal{C}$) or if the time limit is exceeded.

Algorithm 3 summarizes the tasks performed by each worker process. The initial distribution of vertices is received in the beginning of the program. The main loop is then responsible for processing the messages received from the master process, which contain the neighborhood movement requested. A load balancing procedure (line 4) tries to rebalance the number of vertices in each subgraph, to prevent a process from being overloaded. This first step is fast and computationally inexpensive, since moving clusters of the current solution between processes causes no change in the CC solution value. Afterwards, the neighborhood movement (requested by the master process) is performed (line 5), as depicted in Figure 1. Then, each worker process runs the $ILSMultiStartCC$ procedure on its modified subgraph $G'(V_j)$ (line 7). Finally, on the next line, the local ILS clustering result $\mathcal{C}_j$ is sent back to the master process q_m.

Algorithm 3: $DistributedILS$ (worker procedure)

```
   Variables: G = (V, E), clustering C_j, number of processes q
1  receive message (⟨InitialDistribution⟩, V_j) from process q_m;
2  while not terminate do
3      receive message (⟨MovementType⟩, Φ) from process q_m;
4      Φ̄ ← LoadBalance (Φ,q);
5      V_j ← Repartition (⟨MovementType⟩, Φ̄);
6      G' ← subgraph induced by V_j ∈ Φ̄;
7      C_j ← ILSMultiStartCC (G');
8      send message (⟨ILSResult⟩, C_j) to process q_m;
9      if termination detected then
10         terminate ← true;
```

Remark that the objective function calculation also benefits from parallelism. For every change in the clustering, a reduction operation is in charge of receiving the local solution value of each subgraph (lines 9-10 in Algorithm 2), from every participating process. Based on these values, the change in global solution is then computed according to the new CC clustering $\mathcal{C}$, as well as the vertex-process partition Φ and the edges that connect vertices from different processes.

The most complex neighborhood structure is $MoveQuasiClique$ (MovementType(4) in Figure 1). The algorithm identifies vertex-overloaded processes, i.e. processes with more than (n/q) vertices. Then, for each process that falls into this category, the procedure tries to identify and move a subset of vertices $\mathbb{K}$ (here defined as a *quasi-clique* – an almost complete subgraph [Brunato et al. 2007]) from an existing cluster to a new cluster in another process. A *quasi-clique* $\mathbb{K}$ exhibits high affinity, that is, high density of internal positive edges and, at the same time, low density of internal negative edges, and therefore constitutes a good subset of vertices to be moved.

3.2.3. Termination detection

The master process finishes its work and terminates when no additional improvement can be found based on the current best solution or if the time limit is exceeded. In other words, the procedure in Algorithm 2 returns when no better clustering solution

is found in the most distant neighborhood of the current solution $\mathcal{C}$. This propagates a termination message to all worker processes as well.

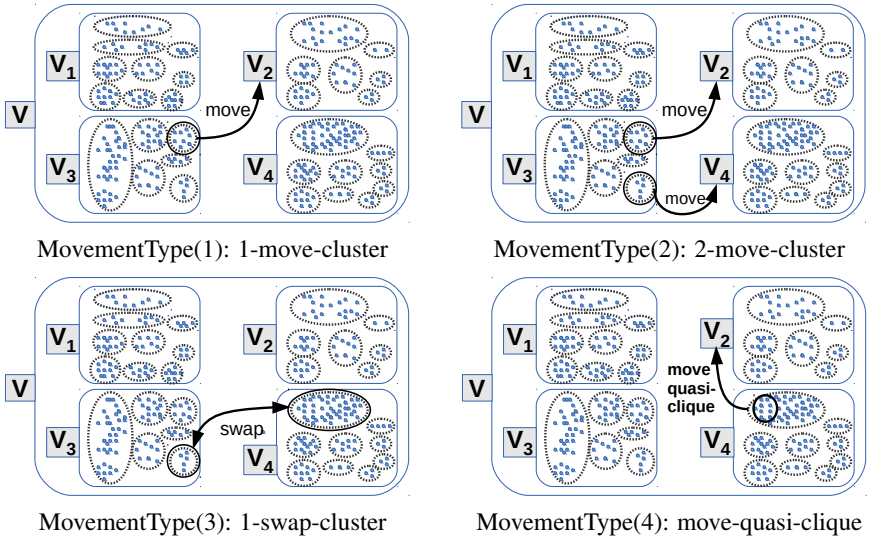

Figure 1. Distributed ILS neighborhood structures. A cluster movement between processes involves at most three modified subgraphs, thus requiring three processes to execute the local ILS procedure (e.g. *2-move-cluster* moves two clusters – and corresponding vertices – from one process subgraph to two different processes). The remaining (idle) processes can be then used to compute other neighborhood movements in parallel.

4. Experiments

The goal of these experiments is to assess the performance of the $DistributedILS$ algorithm using different sets of signed graph instances, when compared to our previous best solution technique, the Independent Parallel $ILS - CC$ algorithm ($ParILS$) [Levorato et al. 2017], which from now on will be called *baseline*. The local search procedure used by both algorithms (*baseline* and $DistributedILS$) runs on the GPU [Levorato et al. 2015a].

4.1. Computational environment

The algorithms described in the previous section were implemented in ANSI C++ (GCC v4.4.7-11), MPI (OpenMPI v1.2.8.4) for message passing and Boost Parallel BGL (v1.61.0) for graph data parallelism. All experiments were performed (with exclusive access) on the SDumont cluster [LNCC 2017] with 198 nodes, each one with two Intel Xeon E5-2695v2 Ivy Bridge @2.4GHz processors (12 cores each) and 64GB of RAM under RedHat Linux 6.4. Each node is equipped with 2 NVIDIA Tesla K40 GPUs containing 12GB of RAM and 2880 CUDA cores. CUDA code was written in "C for CUDA V6.5" [NVIDIA 2014]. The presented results were obtained from 50 independent runs.

4.2. Test problems

Computational experiments were undertaken on (i) a set of 2 social networks from the literature, (ii) a set of 3 network instances generated from MovieLens movie ratings website, and (iii) a set of 6 random signed networks with a predefined community structure. We will briefly describe these instances[2].

(i) This set of instances is composed by two signed networks widely used in the related literature: the first representing the large scale social network of technology-related news website Slashdot [Leskovec et al. 2010, Facchetti et al. 2011], and

[2]All instances are available in `http://www.ic.uff.br/~yuri/CCinst-large.html`.

the second one contains the Epinions [Epinions 1999] signed network, an on-line review website which allows users to either like or dislike other people's reviews. Since both networks are originally signed digraphs, they were converted to undirected graphs.

(ii) We proposed three new signed social networks based on movie ratings given by each user from the MovieLens Website. The dataset used was MovieLens 20M (20 million ratings and 465,000 tag applications applied to 27,000 movies by 138,000 users; Released on 4/2015) [GroupLens 2017]. The signed graphs were generated according to the following algorithm. Let the movie rating be an integer between 1 and 5. For each pair of users, we identify the list of movies both users have rated. Based on this list, we totalize the number of similar ratings each user gave to the same movie. A movie rating from users a and b is similar if both users gave a rating of 1 or 2 (bad movie), or if both users rated the movie with 3 (regular movie), or if both users rated the movie with 4 or 5 (good movie). We then normalize the ratings by subtracting from each user rating the its own average rating, in order to prevent problems with the difference of scaling between users. Based on this data, a user-user matrix $D = \{d_{uv} : u, v \in V\}$ of cosine distance similarities is then calculated. Each value is in the range of [-1,1], where -1 represents perfect disagreement and 1 means perfect agreement between users. If the absolute value of d_{uv} is greater than a given threshold mw, we add a positive or negative edge between users u and v, according to the sign of d_{uv}.

(iii) We generated six large random signed networks with a predefined community structure, according to [Yang et al. 2007]. The random signed network is defined as $SG(c, n, k, p_{in}, p_-, p_+)$, where c is the number of communities in the network, n is the number of nodes in each community, k is the degree of each node, p_{in} is the probability of each node connecting other nodes in the same community, p_- denotes the probability of negative links appearing within communities, and p_+ denotes that of positive links appearing between communities. Each generated instance has a different value for the parameter c. The other parameters were fixed to $n = 4096$, $k = 16$, $p_{in} = 0.8$, $p_- = 0.8$ and $p_+ = 0.6$.

Instance		$\|V\|$	$\|E\|$	**Instance**		$\|V\|$	$\|E\|$
Slashdot		82144	500481	Random	$c = 8$	32768	262144
Epinions		131828	711210		$c = 16$	65536	524284
MovieLens	$mw = 0.90$	138494	371118		$c = 32$	131072	1048576
	$mw = 0.85$	138494	3451683		$c = 64$	262144	2097152
	$mw = 0.80$	138494	10211818		$c = 128$	524288	4194304
					$c = 256$	1048576	8388608

Table 1. Dimensions of each signed graph instance $|V|$: number of vertices, $|E|$: number of edges.

4.3. Obtained results

Baseline was configured to use 10 processes running in parallel, equally dividing the number of multistart iterations of the original $ILSMultiStartCC$ procedure. In $DistributedILS$, the runtime configuration used 8 processes for instances with the number of vertices $|V| \leq 5 \times 10^5$, and 16 processes for larger graphs. The evaluation of the solution values was carried out by means of an ANOVA analysis [Montgomery 2005] at 99.9% confidence level. In the presented results, since the p-value is is always less than the significance level of 0.001, we can reject the null hypothesis and conclude that the means are significantly different. After running both algorithms for 2 hours, $DistributedILS$ has enhanced the solution quality on Epinions instance (Figure 2-b) with an average improvement of 9.9%. On the other hand, when solving the Slashdot instance (Figure 2-a),

the *baseline* achieved the best solution value, but with a slight average gap (smaller than 0.8%) when compared to $DistributedILS$.

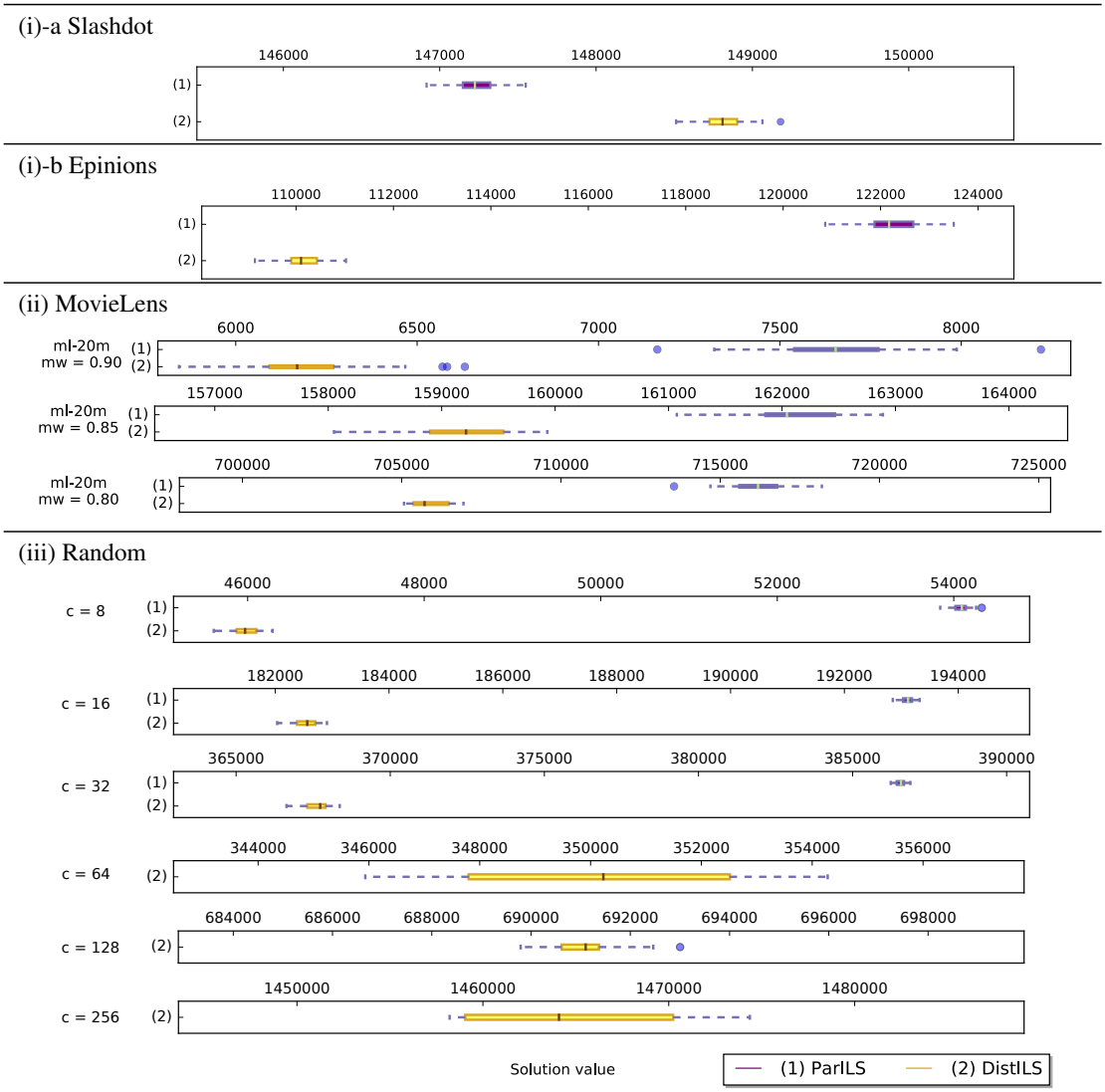

Figure 2. $Baseline$ **(ParILS) and** $DistributedILS$ **(DistILS) CC results for literature instances (i), for Movielens network instances (ii), and for random networks with predefined community structure (iii). The boxplot shows the distribution of the solution values obtained by each algorithm after running for** 2 **hours, considering** 50 **independent executions.**

Regarding the solution of the MovieLens instances in (ii), as shown on Figure 2, an analysis of the gap in solution values between $DistributedILS$ and *baseline* also indicates the superiority of the proposed method. For the $mw = 0.90$ instance, the solution values were 19.23% smaller than those obtained by *baseline*. When solving larger instances with $mw = 0.85$ and $mw = 0.80$, $DistributedILS$ presented solutions with average improvements of 1.83% and 1.38%. Finally, when solving the random instances with predefined community structure in (iii), the proposed algorithm returned an improved solution on test instances $c = 8$, $c = 16$ and $c = 32$ (Figure 2-(iii)). The average gaps in solution values were -14.96%, -5.47% and -4.87%, respectively. The *baseline* method was not able to process the remaining instances ($c \geq 64$), either for not providing an initial solution within the $2h$ time limit or due to lack of memory. Therefore, in these cases, we only display the solution values obtained by $DistributedILS$.

5. Concluding remarks

We developed the first distributed algorithm for efficiently solving the CC problem, based on a heterogeneous computing platform. It iteratively repartitions the graph between processes, invoking the ILS metaheuristic locally on each node and merging individual results into a global solution. Experiments were conducted on both synthetic and real datasets. On the synthetic dataset our approach is able to scale to 1,048,576 nodes and 8,388,608 edges. The proposed algorithm can provide efficient solutions to the CC problem when traditional metaheuristics fail due to the need to be aware of the entire graph, relying on a single memory space.

Many future research topics could be built upon this framework, including a fully distributed heuristic for building an initial global solution. Larger signed graph instances (both in the number of vertices and edges) may be generated as well.

Acknowledgement

The authors acknowledge the National Laboratory for Scientific Computing (LNCC/MCTI, Brazil) for providing HPC resources of the SDumont supercomputer, which have contributed to the research results reported within this paper. URL: `http://sdumont.lncc.br` We also thank the National Council for Scientific and Technological Development (CNPq) Cnpq-Universal project number 443883/2014-9.

References

Ailon, N., Charikar, M., and Newman, A. (2008). Aggregating inconsistent information: ranking and clustering. *Journal of the ACM (JACM)*, 55(5):23.

Aronson, E. and Cope, V. (1968). My enemy's enemy is my friend. *Journal of personality and social psychology*, 8(1p1):8.

Bansal, N., Blum, A., and Chawla, S. (2002). Correlation clustering. In *The 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002. Proceedings.*, pages 238–250, Vancouver, Canada. IEEE.

Bhattacharya, A. and De, R. K. (2008). Divisive correlation clustering algorithm (dcca) for grouping of genes: detecting varying patterns in expression profiles. *bioinformatics*, 24(11):1359–1366.

Brandes, U., Delling, D., Gaertler, M., Gorke, R., Hoefer, M., Nikoloski, Z., and Wagner, D. (2008). On modularity clustering. *Knowledge and Data Engineering, IEEE Transactions on*, 20(2):172–188.

Brunato, M., Hoos, H. H., and Battiti, R. (2007). On effectively finding maximal quasi-cliques in graphs. In *International conference on learning and intelligent optimization*, pages 41–55. Springer.

Cartwright, D. and Harary, F. (1956). Structural balance: A generalization of heider's theory. *Psychological Review*, 63(5):277–293.

Charikar, M., Guruswami, V., and Wirth, A. (2003). Clustering with qualitative information. In *Foundations of Computer Science, 2003. Proceedings. 44th Annual IEEE Symposium on*, pages 524–533. IEEE.

Chierichetti, F., Dalvi, N., and Kumar, R. (2014). Correlation clustering in mapreduce. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM.

DasGupta, B., Encisob, G. A., Sontag, E., and Zhanga, Y. (2007). Algorithmic and complexity results for decompositions of biological networks into monotone subsystems. *BioSystems*, 90(1):161–178.

Davis, J. (1967). Clustering and structural balance in graphs. *Human Relations*, 20(2):181–187.

Demaine, E. D., Emanuel, D., Fiat, A., and Immorlica, N. (2006). Correlation clustering in general weighted graphs. *Theoretical Computer Science*, 361(2):172–187.

Doreian, P. and Mrvar, A. (2015). Structural balance and signed international relations. *Journal of Social Structure*, 16:1.

Drummond, L., Figueiredo, R., Frota, Y., and Levorato, M. (2013). Efficient solution of the correlation clustering problem: An application to structural balance. In *Lecture Notes in Computer Science*, pages 674–683. Springer Nature.

Duch, J. and Arenas, A. (2005). Community detection in complex networks using extremal optimization. *Physical review E*, 72(2):027104.

Elsner, M. and Schudy, W. (2009). Bounding and comparing methods for correlation clustering beyond ilp. In *Proceedings of the Workshop on Integer Linear Programming for Natural Langauge Processing*, ILP '09, pages 19–27, Stroudsburg, PA, USA.

Epinions (1999). Website. URL http://www.epinions.com. Accessed on March 2015.

Facchetti, G., Iacono, G., and Altafini, C. (2011). Computing global structural balance in large-scale signed social networks. In *Proceedings of the National Academy of Sciences of the United States of America*,

volume 108, pages 20953–20958.

Feo, T. A. and Resende, M. G. (1995). Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6(2):109–133.

Gregor, D. and Lumsdaine, A. (2005). The parallel bgl: A generic library for distributed graph computations. *Parallel Object-Oriented Scientific Computing (POOSC)*, 2:1–18.

GroupLens (2017). Movielens dataset collection. `https://grouplens.org/datasets/movielens`.

Gülpinar, N., Gutin, G., Mitra, G., and Zverovitch, A. (2004). Extracting pure network submatrices in linear programs using signed graphs. *Discrete Applied Mathematics*, 137:359–372.

Heider, F. (1946). Attitudes and cognitive organization. *Journal of Psychology*, 21(1):107–112.

Huffner, F., Betzler, N., and Niedermeier, R. (2009). Separator-based data reduction for signed graph balancing. *Journal of Combinatorial Optimization*, 20(4):335–360.

Kim, S., Yoo, C. D., Nowozin, S., and Kohli, P. (2014). Image segmentation UsingHigher-order correlation clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(9):1761–1774.

Kunegis, J. (2013). Konect: the koblenz network collection. In *Proceedings of the 22nd International Conference on World Wide Web*, pages 1343–1350. ACM.

Leskovec, J., Huttenlocher, D., and Kleinberg, J. (2010). Signed networks in social media. In *Proceedings of the 28th international conference on Human factors in computing systems - CHI '10*, pages 1361–1370. Association for Computing Machinery (ACM).

Leskovec, J. and Krevl, A. (2014). SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`.

Levorato, M., Drummond, L., Frota, Y., and Figueiredo, R. (2015a). A GPU-accelerated local search algorithm for the Correlation Clustering problem. In *Proceedings of the XLVII Brazilian Symposium on Operations Research*, Porto de Galinhas, PE, Brazil.

Levorato, M., Drummond, L., Frota, Y., and Figueiredo, R. (2015b). An ils algorithm to evaluate structural balance in signed social networks. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, pages 1117–1122.

Levorato, M., Figueiredo, R., Frota, Y., and Drummond, L. (2017). Evaluating balancing on social networks through the efficient solution of correlation clustering problems. *EURO Journal on Computational Optimization*, pages 1–32.

LNCC (2017). Santos dumont supercomputer. http://sdumont.lncc.br.

Lourenço, H. R., Martin, O. C., and Sttzle, T. (2003). Iterated local search. In *Handbook of Metaheuristics*, pages 320–353. Springer Nature.

Macon, K., Mucha, P., and Porter, M. (2012). Community structure in the united nations general assembly. *Physica A: Statistical Mechanics and its Applications*, 391(1-2):343–361.

Mendonça, I., Figueiredo, R., Labatut, V., and Michelon, P. (2015). Relevance of negative links in graph partitioning: A case study using votes from the european parliament. In *2015 Second European Network Intelligence Conference*, pages 122–129. IEEE.

Mladenović, N. and Hansen, P. (1997). Variable neighborhood search. *Computers & Operations Research*, 24(11):1097–1100.

Montgomery, D. (2005). Design and analysis of experiments (6th ed) john wiley and sons. *New York, NY*.

Newman, M. E. (2006). Modularity and community structure in networks. *Proceedings of the National Academy of Sciences*, 103(23):8577–8582.

NVIDIA, C. (2014). Toolkit v6. 5. *URL http://docs.nvidia.com/cuda/cuda-c-programming-guide*.

Pan, X., Papailiopoulos, D., Oymak, S., Recht, B., Ramchandran, K., and Jordan, M. I. (2015). Parallel correlation clustering on big graphs. In *Advances in Neural Information Processing Systems*, pages 82–90.

Schwartz, T. (2010). The friend of my enemy is my enemy, the enemy of my enemy is my friend: Axioms for structural balance and bi-polarity. *Mathematical Social Sciences*, 60(1):39–45.

Srinivasan, A. (2011). Local balancing influences global structure in social networks. In *Proceedings of the National Academy of Sciences of the United States of America*, volume 108, pages 1751–1752.

Wang, N. and Li, J. (2013). Restoring: A greedy heuristic approach based on neighborhood for correlation clustering. In *Advanced Data Mining and Applications*, pages 348–359. Springer.

Yang, B., Cheung, W., and Liu, J. (2007). Community mining from signed social networks. *IEEE Transactions on Knowledge and Data Engineering*, 19(10):1333–1348.

Zhang, Z., Cheng, H., Chen, W., Zhang, S., and Fang, Q. (2008). Correlation clustering based on genetic algorithm for documents clustering. In *2008 IEEE Congress on Evolutionary Computation*, pages 3193–3198.