

Scalable Adaptive Label Propagation in Grappa

Golnoosh Farnadi^{*‡}, Zeinab Mahdavi[†], Ivan Keller[‡], Jacob Nelson[§], Ankur Teredesai[†],
Marie-Francine Moens[‡], and Martine De Cock[†]

^{*}Department of Applied Mathematics, Computer Science and Statistics, Ghent University, Belgium

[†]Center for Data Science, University of Washington Tacoma, US

[‡]Department of Computer Science, KU Leuven, Belgium

[§]Department of Computer Science and Engineering, University of Washington, US

Abstract—Nodes of a social graph often represent entities with specific labels, denoting properties such as age-group or gender. Design of algorithms to assign labels to unlabeled nodes by leveraging node-proximity and a-priori labels of seed nodes is of significant interest. A semi-supervised approach to solve this problem is termed “LPA-Label Propagation Algorithm” where labels of a subset of nodes are iteratively propagated through the network to infer yet unknown node labels. While LPA for node labelling is extremely fast and simple, it works well only with an assumption of node-homophily – connected nodes are connected because they must deserve a similar label – which can often be a misnomer. In this paper we propose a novel algorithm “Adaptive Label Propagation” that dynamically adapts to the underlying characteristics of homophily, heterophily, or otherwise, of the connections of the network, and applies suitable label propagation strategies accordingly. Moreover, our adaptive label propagation approach is scalable as demonstrated by its implementation in Grappa, a distributed shared-memory system. Our experiments on social graphs from Facebook, YouTube, Live Journal, Orkut and Netlog demonstrate that our approach not only improves the labelling accuracy but also computes results for millions of users within a few seconds.

I. INTRODUCTION

Label propagation is a technique in which the labels of a portion of the nodes of a graph are propagated iteratively to assign labels to the unknown nodes [12]. It is a fast and simple approach for node labelling that is popular for community detection in social networks [10]. Our interest in label propagation is related but slightly different, namely user modelling in social networks. Automatically inferring attributes of users on social media platforms has gained a lot of interest both from academia and industry. Examples of recent efforts include contests¹ in which participants are challenged to infer age, gender and personality traits of Facebook, Twitter and YouTube users from their status updates, tweets or videoblogs. The datasets used in these contests are small (100s to 1000s of users) and do not contain information about the social network structure, i.e. about the edges in the social graph. The work presented in this paper on the other hand is precisely about leveraging knowledge of friendship links in a social network to infer missing user attributes, and doing so in a manner that scales out to graphs with millions of users. To

this end, we propose a scalable adaptive version of Zhu and Ghahramani’s original label propagation algorithm (LPA) [12] for node labelling in social graphs.

In the original LPA, the labels of a portion of nodes are iteratively propagated through the network to fill in the labels of the unlabelled nodes. While LPA for node labelling is extremely fast and simple, it works well only with an assumption of node-homophily – connected nodes are connected because they must deserve a similar label – which can often be a misnomer. For example, users in a social dating network are mostly connected with their opposite sex. In this paper we propose a novel “Adaptive Label Propagation” algorithm (Adaptive-LPA) that dynamically adapts to the underlying characteristics of homophily, heterophily, or otherwise, of the connections of the network, and applies suitable label propagation strategies accordingly.

Another limitation of the original LPA algorithm is scalability. Social network applications are characterized by large-scale computations. A variety of frameworks has been proposed to ease this process. Buzun et al. [1] summarize the known implementations of community detection algorithms including label propagation algorithms in different big data infrastructures, such as LPA using Hadoop MapReduce [8], SLPA using MPI [11], BMLPA [9], and COPRA [2], among many others. In most of the mentioned frameworks, graph computation is still a challenging task [5].

In this paper, we implement our proposed Adaptive-LPA algorithm in *Grappa* [7], a modern runtime system that provides a shared memory abstraction for clusters. Grappa was developed to resolve deficiencies of earlier Distributed Shared Memory (DSM) systems through a combination of efficient threading, communication, and synchronization primitives. In addition to its low-level shared-memory API, Grappa also implements variants of many common programming abstractions for big data, include MapReduce and a GraphLab-like vertex-centric model [6]. Grappa’s efficient low-level primitives allow it to combine multiple abstractions in the same program, and in many cases perform better than the original, native implementations of these abstractions [7]. Since a vertex-centric abstraction is a good fit for the graph structure of social networks, we use Grappa’s vertex-centric API to implement our proposed Adaptive-LPA algorithm.

The remainder of this paper is structured as follows: In Section II, we present the formulation of our proposed approach,

¹ WCPR2013, <http://mypersonality.org/wiki/doku.php?id=wcpr13>; WCPR2014, <https://sites.google.com/site/wcprst/home/wcpr14>; PAN2015, <http://www.uni-weimar.de/medien/webis/events/pan-15>, accessed on Jul 11, 2015

i.e. the Adaptive-LPA algorithm. We provide the details of our Grappa implementation in Section III. We analyze the behaviour of Adaptive-LPA on several datasets based on the accuracy of prediction and scalability in Section IV. Finally, we conclude and provide the future direction of this work in Section V.

II. ADAPTIVE LABEL PROPAGATION ALGORITHM

The label propagation algorithm (LPA) was introduced in 2002 by Zhu and Ghahramani [12]. LPA propagates the labels of a subset of nodes through the network iteratively to infer the unknown node labels. Let $\mathcal{X}_L = (x_1, \dots, x_l)$ be l seed nodes, and $\mathcal{Y}_L = (y_1, \dots, y_l)$ their corresponding known labels. Likewise, let $\mathcal{X}_U = (x_{l+1}, \dots, x_{l+u})$ be u unlabeled nodes and $\mathcal{Y}_U = (y_{l+1}, \dots, y_{l+u})$ their respective unknown labels. Typically $l \ll u$. Suppose the labels y_i belong to a discrete finite set of n classes denoted $\mathcal{C} = \{c_1, c_2, \dots, c_n\}$.

Consider the undirected social graph $G = (V, E)$ where V is the set of individuals (i.e., $V = X_L \cup X_U$) and E is the set of their social links. We set the weight equal to 1 whenever a friendship link is present in the social network, and 0 otherwise. Thus, the weights are evenly set to 1, $\forall (v_i, v_j) \in E$. Let T be the $(l+u) \times (l+u)$ column normalized adjacency matrix (matrix of the weights) that defines the *transitions*. Thus, the transitions are of the form:

$$T_{ij} = \begin{cases} \frac{1}{|N(v_j)|}, & \text{if } (v_i, v_j) \in E \\ 0, & \text{otherwise} \end{cases}$$

where $N(v_j)$ denotes the set of neighbors of v_j . In the above expression, the value of the transition (normalized weight) between v_j and each of its neighbors is related to the number of neighbors v_j has. The more neighbors v_j has, the smaller the T_{ij} values of the edges adjacent to v_j are, i.e. the less each of v_j 's neighbors will contribute individually to inferring the label of v_j . Each node is assigned a ‘‘soft’’ label that can be interpreted as a probability distribution over the label values: let Y be the $(l+u) \times n$ matrix where:

$$Y_{ik} = P(y_i = c_k) \quad \forall i \in \{1, \dots, l+u\}, \forall c_k \in \mathcal{C}$$

represents the probability that node x_i has label $c_k \in \mathcal{C}$. LPA iteratively updates the soft labels of the unlabeled nodes as follows:

- 1) **Initialization.** Soft labels of the seed nodes are clamped to their given label ($P(y = c) = 1$ if $y = c$ and 0 otherwise):

$$Y_{ik}^0 = P(y_i = c_k) \quad \forall i \in \{1, \dots, l\}, \forall c_k \in \mathcal{C}$$

And soft labels of the unlabeled nodes (i.e., $Y_{ik}^0, \forall i \in \{l+1, \dots, l+u\}, \forall c_k \in \mathcal{C}$) are unknown.

- 2) **Upgrade soft labels.** Each unlabeled node x_i updates its soft label by the weighted sum of the soft labels of all its labeled neighbors (i.e., neighbors with known label including seed nodes and nodes that their soft labels

in the previous iteration are updated) according to the weights T_{ij} :

$$\tilde{Y}_{ik}^{t+1} = \sum_{j=1}^{l+u} T_{ij} \tilde{Y}_{jk}^t \quad \forall i \in \{l+1, \dots, l+u\}, \forall c_k \in \mathcal{C}$$

or equivalently with matrix formulation: $\tilde{Y}^{t+1} = T Y^t$, where t denotes the iteration number. The row-normalization on \tilde{Y} to keep the probability distribution interpretation is done by:

$$Y_{ik}^{t+1} = \frac{\tilde{Y}_{ik}^{t+1}}{\sum_{c_{k'} \in \mathcal{C}} \tilde{Y}_{ik'}^{t+1}} \quad \forall i \in \{1, \dots, l+u\}, \forall c_k \in \mathcal{C}$$

- 3) **Repeat step 2 until convergence.** All nodes repeatedly propagate their soft label at each iteration. The seed nodes maintain their original label throughout, while the other nodes keep updating their soft label until convergence (i.e., until all nodes receive soft labels). The proof of the convergence of the algorithm is discussed in [12]; it does not depend on the number of classes.

On convergence, hard labels are assigned to all unlabeled nodes according to their soft labels using the most probable label:

$$\hat{y}_i = \arg \max_{c_k \in \mathcal{C}} Y_{ik}^\infty$$

where Y^∞ represents the matrix Y on convergence.

LPA relies on an iterative spread of label information from nodes to their neighbors. In the form presented above, LPA for node labelling is effective when users' labels are similar to their neighbors' label, i.e. when there is an homophily relation among the users w.r.t. their labels. Not all network structures and labels adhere to this however. For example, in a dating network, neighboring nodes typically have opposite genders as opposed to the same gender. It is not difficult to imagine ways in which to incorporate this prior knowledge into the propagation strategy, namely ‘‘propagate the opposite of the label’’. The Adaptive-LPA algorithms that we propose goes yet one step further by dynamically adjusting the propagation strategy by computing the probability of belonging to each label for a node given the labels of its known neighbors. This means that no prior knowledge of the form ‘‘this is a dating network’’ is required. Instead, Adaptive-LPA derives this knowledge automatically by inspecting the seed nodes, and then turns it into an appropriate propagation strategy to label all nodes.

We introduce the neighbor's information in the LPA mechanism using a conditional probability matrix (CP) of size $n \times n$ as:

$$CP = \begin{bmatrix} p_{c_1|c_1} & p_{c_1|c_2} & \dots & p_{c_1|c_n} \\ p_{c_2|c_1} & p_{c_2|c_2} & \dots & p_{c_2|c_n} \\ \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots \\ p_{c_n|c_1} & p_{c_n|c_2} & \dots & p_{c_n|c_n} \end{bmatrix}$$

where $p_{c_k|c_{k'}}$ denotes the probability that a node belongs to class c_k given a neighbor from class $c_{k'}$. We estimate this probability using the seed nodes as follows:

$$p_{c_k|c_{k'}} = \frac{\sum_{(v_i, v_j) \in E'} \min(Y_{ik}^0, Y_{jk'}^0)}{\sum_{(v_i, v_j) \in E'} Y_{jk'}^0}$$

where E' is the set of edges for which both nodes belong to the seed nodes. Using the conditional probability matrix, we change the propagation step of LPA to propagate Y as follows:

$$\tilde{Y}_{ik}^{t+1} = \sum_{j \in N(x_i)} \sum_{c_{k'} \in \mathcal{C}} \frac{CP_{kk'}}{|N(x_j)|} Y_{jk'}^t$$

$$\forall i \in \{l+1, \dots, l+u\}, \forall c_k \in \mathcal{C}$$

III. GRAPPA IMPLEMENTATION

Grappa exploits concurrency in big data applications to cover network access latencies, enabling users to program a cluster as if it were a single, large shared memory machine. It does so using three main components: a *user-level multitasking system* supporting thousands of concurrent threads per core in the system, a *communication layer* that combines independent messages with common destinations to increase network efficiency, and a *distributed shared memory (DSM)* layer that provides access to data anywhere in the system. Remote memory is accessed using *delegate operations*, which execute a small piece of code at the core where a value is stored; this allows efficient fine-grained memory operations, including both simple reads and writes and more complex synchronization operations. This allows Grappa's DSM to provide the same memory consistency model as is used in C/C++. Grappa is implemented as a C++ user library using MPI [3] for communication. For further details, see [7].

We first implement the standard approach to LPA using Grappa's vertex-centric API. This implements a subset of the "graph-parallel" techniques developed in the GraphLab project [6] to support iterative, vertex-centric computation over sparse, natural graphs. The key idea in GraphLab is the "Gather, Apply, Scatter" (GAS) model used to describe the computation performed at each vertex in the graph using data from neighboring vertices. For more explanation and implementation details of this approach, see [6].

The Adaptive-LPA algorithm that we propose goes yet one step further. Fig. 1 shows the core of the Grappa implementation of our proposed Adaptive-LPA approach, including the following main components:

Gather stores the number of times a label occurs among the known neighbors of a user in an array. The size of this array is set to the number of classes $|\mathcal{C}|$. *LP-Delta* is the array that stores the probability of label occurrences in each class for each node. We define the sum operator ($+=$) to gather the LP-Delta arrays of the neighbors and then normalize all values. To calculate the probabilities that come from the neighbor, the *CP* matrix is used and the transition is applied, where the

```
AdaptiveLabelPropagation(Vertex v) {
    do_scatter = false;
}
bool gather_edges(const Vertex v) const { return true; }

Gather gather(const Vertex v, Edges e) const {
    CHECK(v->label < number_of_classes);
    return Lp_Delta(v->label);
}

void apply(Vertex v, const Gather total) {
    double maxCount = 0.0;
    int maxLabel = v->label;
    for (int i = 0; i < number_of_classes; i++) {
        if (total.deltaProb[i] > maxCount) {
            maxCount = total.deltaProb[i];
            maxLabel = i;
        }
    }
    if (maxLabel == Unknown) {
        do_scatter = false;
        v->activate();
    } else if (maxLabel != v->label) {
        v->label = maxLabel;
        for (int i = 0; i < number_of_classes; i++) {
            deltaProb[i] = total.deltaProb[i];
        }
        do_scatter = true;
    } else {
        do_scatter = false;
    }
}
bool scatter_edges(const Vertex v) const { return do_scatter; }

Gather scatter(const Edges e, Vertex target) const {
    CHECK(target.nadj > 0) << "The network size must be greater than 0";
    Lp_Delta delta;
    for (int i = 0; i < number_of_classes; ++i) {
        for (int j = 0; j < number_of_classes; ++j) {
            if (deltaProb[i] == Unknown)
                delta.deltaProb[i] = Unknown;
            else
                delta.deltaProb[i] += symmetric_prob.probs[i][j] * deltaProb[i] / target.nadj;
        }
    }
    delta.normalize();
    return delta;
};
```

Fig. 1: Snapshot of the Adaptive-LPA implementation in Grappa which illustrates how labels are gathered and scattered iteratively using the GraphLab's GAS engine.

result is divided to the size of the neighbor's network (*nadj*). For simple propagation, *LP-Delta* is just computed based on the probability of label occurrences from the neighbors (i.e., soft labels), and based on the transition, it is divided by the neighbor's network size. We use the *symmetric* allocation to define the *CP* matrix and seeds which allocates space for a copy of the *CP* matrix and a copy of the seeds on every core in the system (e.g., *symmetric_prob.probs* is the *CP* matrix).

Apply finds the most probable label between neighbors. If all the neighbors are unknown, we skip the node and revisit it in the next iteration. This continues until we have enough information (at least one labeled neighbor) about this node. Once we find a label, the scatter flag is set to true.

Scatter If the scatter flag is true, label updates are passed to neighbors. And, the scatter flag is true when we find a label for a user. Then by using the *do-scatter* flag, we recognize the nodes which get a label and then we distribute the changed labels to their neighbors.

IV. EXPERIMENTAL RESULTS

In this section we conduct two sets of experiments. The first set is to measure the accuracy of our proposed *Adaptive-LPA* approach for node labelling. To this end we use a dataset with 3 million users from the social network Netlog. In the second set of experiments we focus on the scalability of our LPA and Adaptive-LPA implementations in Grappa. For these experiments we use six large scale datasets from social

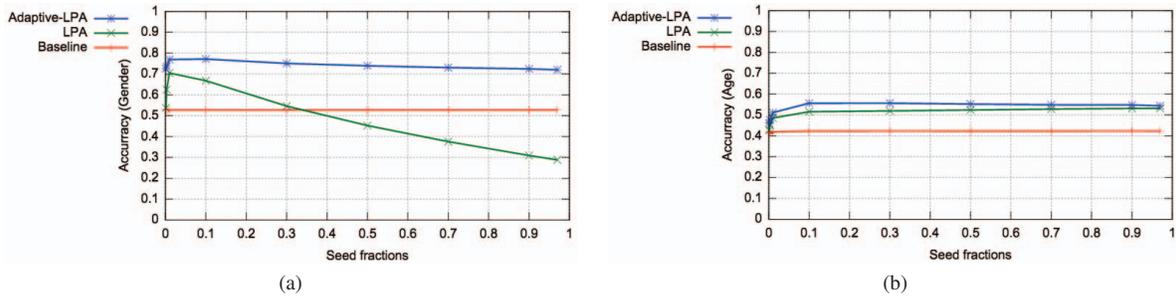


Fig. 2: Comparing accuracy of (a) gender prediction and (b) age prediction between the majority baseline (red line), the LPA (green line), and the Adaptive-LPA (blue line) approaches.

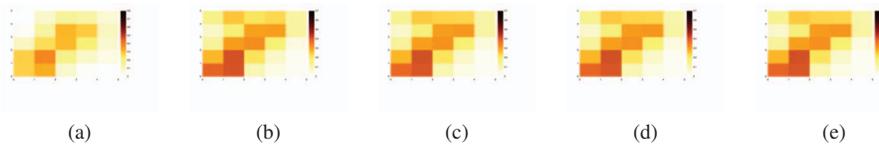


Fig. 3: Heatmap using a 6×6 CP matrix of age groups, with five seed fractions. For each fraction, the CP is averaged over the CPs of 10 randomly selected fractions of the same size (a) 0.1% of the data, (b) 1% of the the data, (c) 10% of the data, (d) 50% of the data, and (e) 97% of the data. The age group increases from the bottom to the top and from the left side to the right side of the matrix.

networks, namely² Facebook, YouTube, Netlog, LiveJournal (two samples), and Orkut.

A. Node labelling dataset

Netlog, currently known better under the name of its successor Twoo, is a large social networking site with over 150 million users. We crawled a Netlog dataset starting from a random user, following the friendship links in a breadth-first way, and collecting publicly available information of the users. We ran the crawler for a few weeks in May 2014 and collected age, gender and the friend lists of 3,359,775 users in 1059 connected components. For this study, we chose the giant connected component of this set of graphs. It consists of 3,351,975 users and 8,029,423 friendship links. Table I presents some relevant statistics of this dataset³.

TABLE I: General statistics on the sample Netlog dataset.

Network	
number of nodes (users)	3,351,975
number of edges (friendship links)	8,029,423
degree distribution power law exponent	2.25
clustering coefficient	0.11%
min degree: 1, max degree : 2510, mean degree : 4.8	
Gender	
# of females	1,585,080 (47.3%)
# of males	1,766,895 (52.7%)
Age (years)	
min age: 1, max age : 114, mean age : 29.4	

²Social media sites: <http://www.facebook.com>, <https://www.youtube.com/>, <http://en.netlog.com>, <http://www.livejournal.com/>, <https://orkut.google.com>

³The sample is available at: <http://www.cwi.ugent.be/NetlogDataSet.html>

On Netlog, gender takes two possible values: “male” or “female”. Our sample has 52.7% of males and 47.3% of females. Netlog is considered to target young adults by providing services such as game playing, chatting, video sharing and blog posting. Our sample reflects this behavior as the age distribution presents a high peak around 20 years (Fig. 4). The mode of the age is 23 years old, the median is 25 years old and the mean age is 29.4 years old. 70% of the users in our sample are between 17 and 30 years. Around 2.5% of the users are under 18 and only 0.05% are under 15 while 1.30% declare ages higher than 73 years. The minimum and maximum ages, respectively 1 and 114, are most probably fake values. Note that in this study, our analysis merely relies on the declared age and we will not discuss its relation with the true user age.

B. Age and gender inference with LPA and Adaptive-LPA

Fig. 2 represents the results of applying LPA and Adaptive-LPA for the gender and age prediction tasks. For age prediction, we categorized the ages of user in six buckets: $[0, 17)$, $[17, 24)$, $[24, 31)$, $[31, 46)$, $[46, 73)$, $[73, 114]$, each corresponding to a class label for age. In both plots, the red line is the majority baseline, the green line shows the accuracy of using LPA and the blue line corresponds to the accuracy of our proposed Adaptive-LPA. The majority baseline predicts for each data point the frequent label across the seed nodes (e.g., if 60% of the seeds are female, then it predicts that all the unlabeled users are female users). All the results are averaged over 10 randomly selected seed sets based on 10 different seed fractions: 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 0.5, 0.7, 0.9, 0.97.

Regarding gender prediction, the LPA approach outperforms the majority baseline when the size of the set of seed nodes is less than 35% of the data. As the size of the set of seed nodes increases, the accuracy decreases and becomes worse than the baseline. However, by adapting to the relations between users with the CP matrix, the Adaptive-LPA approach outperforms both the majority baseline and the LPA approach significantly (by performing paired t-test with $p\text{-value}=6.679 \cdot 10^{-10}$ and $p\text{-value}=0.0008151$, respectively).

To gain insight into the cause of the differences in accuracy of the LPA and the Adaptive-LPA algorithms for the task of gender inference, we look at the neighbors' statistics in this sample. Not surprisingly, male and female users tend to have different preferences regarding the gender of their friends. Netlog users show a particularly sharp behavior in this respect. As Fig. 5 shows, a large fraction of users choose to connect with the opposite gender. This is particularly true for female users, where 73% have only males in their neighborhood while 58% of male users have only female connections. Fig. 5 clearly indicates that users have a strong tendency to connect with the opposite gender in Netlog. This property is called *gender heterophily* and it derives from a particular usage of this social networking site, namely for flirting and dating purposes. Hence it is no surprise that LPA, which assumes label homophily, performs poorly on Netlog data.

The Adaptive-LPA algorithm on the other hand adapts to the behavior of users. This approach not only gives better accuracy in predicting the labels but can also detect the dynamics of the network without having any prior knowledge. The importance of this is illustrated by the fact that Netlog was originally designed as a general purpose social network where users were expected to form homophily relations, but it organically became a dating platform. In other words, the prior knowledge about the expected kind of relationships among users became obsolete as the network grew.

Regarding age prediction, both the LPA and the Adaptive-LPA approaches outperform the majority baseline (using paired t-test with $p\text{-value}=0.000133$ and $p\text{-value}=1.126 \cdot 10^{-5}$, respectively), however the Adaptive-LPA approach performs slightly better than the LPA approach. To explain this behavior, for each user's age in the range from one to 114 years (i.e.,

min and max age in the sample), we computed the average distribution of the neighbor ages. The result is visualized in Fig. 4 where only the mean and quartiles are displayed for each distribution. As shown in the figure, users aged from 17 to 72 have neighbors whose median age is almost linearly dependent on their age. This concerns the vast majority of 97.9% of the users. From 17 to around 30 years (almost 70% of users) we observe *age homophily*: users mostly connect with others having roughly the same age. As their age increases, users tend to uniformly connect with younger users. This behavior is consistent with what we can expect from a dating service. From age 73 there is a clear change in the connection behavior, where users connect with much younger users and the median of the age of their neighbors drops suddenly from 45 (for 72-year-old users) to 25 years (for 73-year-old users). This pronounced shift could indicate the start of dissimulated ages. Indeed, it is likely that users who declare to be aged from 73 to 114 are not providing their true age. However, they represent only 1.30% of the users in the whole data set. Interestingly, the distribution of the age of neighbors for these users is similar to the one for users between 13 and 16. The LPA approach proves to work with homophily relations and this is the main reason that the LPA approach significantly outperforms the majority baseline for the task of age prediction. Regarding the Adaptive-LPA results, using the CP matrix of seeds, allows us to detect the same trend as seen in Fig. 4.

Fig. 3 presents a heatmap of the 6×6 CP matrix of age, for the selected five seed fractions. Obviously, having more users as seeds results in a more accurate picture of the whole dataset, however even with 0.01% of the data, we can still observe the age homophily of the users in the whole dataset. There is a clear shift to the left for three age buckets (i.e., [17, 24), [24, 31), and [31, 73)) which shows their tendency of having similar or younger friends, while for younger users in age [0, 17), users are more likely to have similar or older friends, and finally for the age bucket [73, 114) that we assume are mostly fake profiles, we can clearly see the tendency of having friends in age bucket [17, 24) and less interest to have friends in their own age group. Our Adaptive-LPA approach detects that the vast majority of users prefers to connect to their similar age group, while at the same time acknowledging the particular, non-mainstream behavior of minority groups.

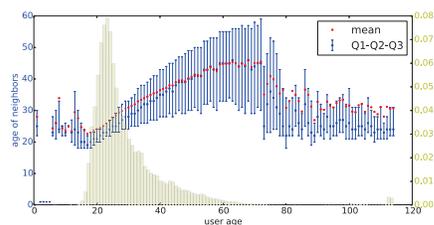


Fig. 4: Age of neighbors depending on the user age in Netlog. Red dots indicate the mean of neighbor ages, blue dots indicate the median and the vertical blue segments are delimited by the first and third quartiles. The user age distribution is shown as the pale yellow overlaid histogram.

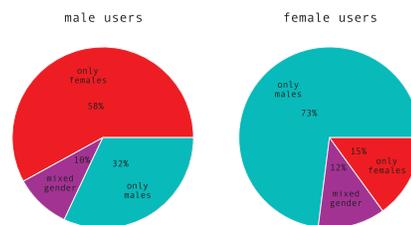


Fig. 5: Fraction of neighbor's gender for both female and male users in Netlog, where red indicates having only female friends, green represents having only male friends and purple is for having mixed gender.

The Adaptive-LPA outperforms the LPA approach significantly, using paired t-test with $p\text{-value} = 2.98 \cdot 10^{-5}$, however there are two main reasons why the difference between the accuracy results of these two approaches is not remarkable. First, only 1.3% of the users in the dataset does not follow the age homophily, which can only be detected by the Adaptive-LPA approach, and second, for users in the age groups $[0, 17)$ and $[17, 24)$, which are the vast majority of the data, there is a very strong age homophily tendency to connect with users of their own age group, which can be modelled with both approaches (i.e., the LPA approach as well as the Adaptive-LPA approach).

Finally, for both age and gender prediction tasks, the best results are achieved by the Adaptive-LPA approach, when using 10% of the data as seeds as presented in Table II.

TABLE II: Comparing accuracy of age and gender prediction. Values in bold are statistically significant with a rejection threshold of 0.001 using a paired t-test w.r.t. the majority baseline and values with a * are statistically significant with a rejection threshold of 0.001 using a paired t-test w.r.t. the LPA approach. All results are averaged over 10-fold cross validation using 10% of the data as seeds.

	Majority baseline	LPA	Adaptive-LPA
Accuracy (gender)	0.53	0.67	0.77*
Accuracy (age)	0.42	0.52	0.56*

C. Scalability datasets

To measure the scalability of our implementations of LPA and Adaptive-LPA in Grappa, we tested our algorithms on various real-world datasets provided by the Stanford Large Network Dataset Collection, also known as the SNAP datasets [4]. The size of the social network datasets that we selected vary from 4K users to 5M users and from 88k friendship relations to 117M relations. We obtained two different samples from LiveJournal. The second sample (i.e., LiveJournal[2]) includes 4,847,571 nodes with 68,993,773 edges. However, since the graph was not fully connected, we extracted the giant connected component (GCC) from this sample. The statistical details of all datasets that we use in this study are presented in Table III.

TABLE III: General statistics on the SNAP datasets.

Data Source	Nodes	Edges	mean degree
Facebook	4,039	88,234	43.7
YouTube	1,134,890	2,987,624	5.2
LiveJournal[1]	3,997,962	34,681,189	17.3
LiveJournal[2]-GCC	4,843,953	68,983,820	28.5
Orkut	3,072,441	117,185,083	76.3

Note that all the datasets in Table III are undirected graphs, therefore to make the input graph for LPA and Adaptive-LPA, we produce edges for both directions. By producing edges, we increase the size of our graphs up to 2 times of the number of edges mentioned in the table (for example, the Orkut graph includes more than 230M edges). To run our Grappa implementation, we used the Sampa group cluster at

the University of Washington. It is a cluster of dual 6-core, 2.66 GHz Intel Xeon X5650 processors with 24 GB of memory per node, connected with a 40Gb Mellanox ConnectX-2 InfiniBand network.

For all the experiments except the experiment using different cores, we use 8 total processor cores using 2 nodes with 4 cores per node. Note that the main difference between the Adaptive-LPA approach and the LPA approach is the pre-processing calculation of the CP matrix which happens before starting the iterations. Hence, it would not affect the performance much. Thus, to save space, we only present the scalability results of the LPA approach which is equivalent to Adaptive-LPA approach without the CP calculation.

D. Scalability Performance

The performance of the Grappa implementation of the LPA approach using the six real-world social network datasets is presented in Fig. 6. Plot (a) presents the scaling results of running the LPA approach using all six datasets. For all the datasets that we use in the experiment of Fig. 6, Plot (a), we set the seed fraction to 10% of the whole dataset and perform node labelling based on randomly generating 2 classes. For all the experiments, we use 2 nodes and we change the number of cores from 1 to 8 which gives us 2 to 16 total processor cores, respectively. Adding more cores reduces the processing time; this behavior is most noticeable for the largest graph (i.e., Orkut with 230M edges) where the processing time decreases from 262 seconds to 93 seconds by increasing the number of processor cores from 2 to 16, respectively. The sizes of the graphs are too small to have enough parallelism to scale further on this hardware.

According to Fig. 6, Plot (b), the LPA approach scales linearly with the size of the graph, thus more users and more connections between users would increase the processing time of the LPA approach. Interestingly, using the Grappa implementation, we label all nodes in a social graph of more than 3 million users and 16M directed friendship relations (i.e., the Netlog sample using 16 processor cores) in only 8 seconds.

To measure the performance of the algorithm with differently sized seed fractions (see Fig. 6, Plot (c)), we apply the implementation of the LPA approach in Grappa on the Netlog dataset for age prediction with 6 labels using 10 different fraction sizes. Obviously, having less data (i.e., a smaller seed size), increases the processing time as the LPA approach requires more iterations for convergence. Hence, as we expect, the performance of the LPA approach linearly gets faster by increasing the seed size.

We also investigate the effects of adding more labels for node labeling. We randomly generate labels for the Netlog dataset from 2 labels up to 1000 labels, and as shown in Fig. 6, Plot (d), expanding the size of the label set increases the processing time of the node labelling approach, however it scales linearly with the size of label set. The average amount of time that we need to label the Netlog sample with more than 3 million users and 16 million undirected connections with 8 total processor cores for 2 classes is 10 seconds while for 1000

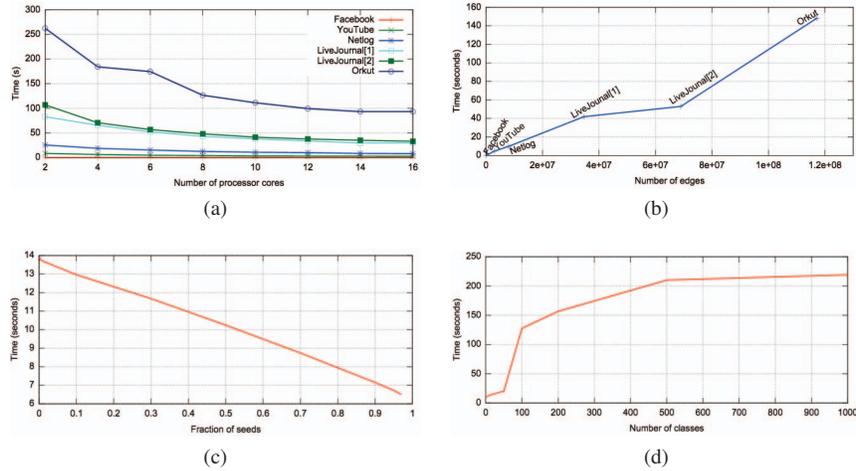


Fig. 6: Scalability analysis of the Grappa implementation using various (a) processor cores, (b) graph sizes, (c) seed fractions and (d) classes.

classes it is only 219 seconds. Interestingly, for any number of labels, LPA for Netlog converges after 6-8 iterations.

V. CONCLUSION

In this study, we have introduced the “Adaptive Label Propagation Algorithm” (Adaptive-LPA), a novel label propagation algorithm for node labelling in social networks. Adaptive-LPA detects the relations between users according to their known labels and propagates labels from labelled nodes to unlabelled ones. We have shown how the Adaptive-LPA approach can lead to more accurate results in node labelling in practice by applying it for the tasks of age and gender predictions for more than 3 million users in Netlog. Since the computation tasks in large graphs such as real social graphs are expensive, we have implemented our proposed Adaptive-LPA along with the LPA algorithm in a scalable framework, called Grappa. We have used the GraphLab-like vertex-centric “Gather-Apply-Scatter” API in Grappa to implement both algorithms. We have presented the effects of the graph size, number of labels and size of labelled data on the processing time of the LPA algorithm. To measure the performance, we have used six datasets, namely, Facebook, YouTube, Netlog, LiveJournal (2 samples) and Orkut. Our results indicate that not only our Adaptive-LPA algorithm gets better prediction accuracy in practice, it can scale linearly for large graphs with millions of edges in a few seconds.

As a next step, we want to compare the performance of the Adaptive-LPA with the available LPA implementations in big data infrastructures such as the LPA implementation in MapReduce. Furthermore, besides age and gender prediction, many other node with more complex relations, such as friends’ personality in a social network, can benefit from the Adaptive-LPA approach. Exploring the effects of using Adaptive-LPA for other node labelling applications remains for our future work.

ACKNOWLEDGMENTS

This work was funded in part by the SBO-program of the Flemish Agency for Innovation by Science and Technology (IWT-SBO-Nr. 110067).

REFERENCES

- [1] Nazar Buzun, Anton Korshunov, Valeriy Avanesov, Ilya Filonenko, Ilya Kozlov, Denis Turdakov, and Hangkyu Kim. Egolp: Fast and distributed community detection in billion-node social networks. In *Data Mining Workshop (ICDMW), 2014 IEEE International Conference on*, pages 533–540. IEEE, 2014.
- [2] Steve Gregory. Finding overlapping communities in networks by label propagation. *New Journal of Physics*, 12(10):103018, 2010.
- [3] William Gropp, Ewing Lusk, and Anthony Skjellum. *Using MPI: portable parallel programming with the message-passing interface*, volume 1. MIT press, 1999.
- [4] Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. <http://snap.stanford.edu/data>, June 2014.
- [5] Jure Leskovec, Kevin J Lang, Anirban Dasgupta, and Michael W Mahoney. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. *Internet Mathematics*, 6(1):29–123, 2009.
- [6] Yucheng Low, Danny Bickson, Joseph Gonzalez, Carlos Guestrin, Aapo Kyrola, and Joseph M Hellerstein. Distributed graphlab: a framework for machine learning and data mining in the cloud. *Proceedings of the VLDB Endowment*, 5(8):716–727, 2012.
- [7] Jacob Nelson, Brandon Holt, Brandon Myers, Preston Briggs, Luis Ceze, Simon Kahan, and Mark Oskin. Latency-tolerant software distributed shared memory. In *2015 USENIX Annual Technical Conference (USENIX ATC 15)*.
- [8] Usha Nandini Raghavan, Réka Albert, and Soundar Kumara. Near linear time algorithm to detect community structures in large-scale networks. *Physical Review E*, 76(3):036106, 2007.
- [9] Zhi-Hao Wu, You-Fang Lin, Steve Gregory, Huai-Yu Wan, and Sheng-Feng Tian. Balanced multi-label propagation for overlapping community detection in social networks. *Journal of Computer Science and Technology*, 27(3):468–479, 2012.
- [10] Jierui Xie and Boleslaw K Szymanski. Community detection using a neighborhood strength driven label propagation algorithm. In *Network Science Workshop (NSW), 2011 IEEE*, pages 188–195. IEEE, 2011.
- [11] Jierui Xie, Boleslaw K Szymanski, and Xiaoming Liu. Slpa: Uncovering overlapping communities in social networks via a speaker-listener interaction dynamic process. In *Proceedings of the 2011 IEEE 11th International Conference on Data Mining Workshops (ICDMW)*, pages 344–349. IEEE, 2011.
- [12] Xiaojin Zhu and Zoubin Ghahramani. Learning from labeled and unlabeled data with label propagation. Technical report, Technical Report CMU-CALD-02-107, Carnegie Mellon University, 2002.