Distributed Social Graph Embedding*

Anne-Marie Kermarrec
INRIA Rennes - Bretagne Atlantique
Anne-Marie.Kermarrec@inria.fr

Vincent Leroy
Yahoo! Research
Barcelona, Spain
leroy@yahoo-inc.com

Gilles Trédan
Technische Universität Berlin
Deutsche Telekom Laboratories
gilles@net.t-labs.tu-berlin.de

ABSTRACT

Distributed recommender systems are becoming increasingly important for they address both scalability and the Big Brother syndrome. Link prediction is one of the core mechanism in recommender systems and relies on extracting some notion of proximity between entities in a graph. Applied to social networks, defining a proximity metric between users enable to predict potential relevant future relationships. In this paper, we propose SoCS (Social Coordinate Systems), a fully distributed algorithm that embeds any social graph in an Euclidean space, which can easily be used to implement link prediction.

To the best of our knowledge, SoCS is the first system explicitly relying on graph embedding. Inspired by recent works on non-isomorphic embeddings, the SoCS embedding preserves the community structure of the original graph, while being easy to decentralize. Nodes thus get assigned coordinates that reflect their social position. We show through experiments on real and synthetic data sets that these coordinates can be exploited for efficient link prediction.

Categories and Subject Descriptors

C.2.4 [**Distributed Systems**]: Distributed applications

General Terms

Design, Experimentation, Performance

Keywords

peer-to-peer, link prediction, embedding

1. INTRODUCTION

Peer-to-peer (P2P) networks are an appealing platform for social networks for they offer opportunities to increase the privacy of the users. In this paper, we address the problem of the distributed computation of *link prediction* [14]. Link prediction is a very important functionality in social network, as it assists users in finding new contacts in the social graph. Decentralized link prediction systems are appealing for two main reasons. First, many link prediction algorithms involves heavy computations and do not scale well with the size of the graph. As social networks now involve millions of users, a distributed approach enables each user to do his share of the computation, hence addressing the scalability problem. Second, some users are relunctant to let private companies handle their private data [5].

Building a distributed link prediction system is a challenging task. It requires to extract a notion of *proximity* between users in the whole social graph. The intuition is that only the *closest* entities to a given user guide his recommendations. Various data sources can be leveraged to establish this proximity, such as users' browsing/downloading/tagging behaviors. These approaches are however often application-dependent. Instead, we adopt a generic approach and solely exploit the social network of each user: only the relationships between friends are used to infer a social proximity measure. Although this measure could be exploited in multiple ways, a straightforward application is to suggest new friends to a user. This problem is typically referred to as *link prediction* [14]. In this paper, we address the challenge of decentralizing the proximity computation.

Most of the social graphs reflecting real life relationships exhibit a *community structure*. In other words, these graphs are often composed of smalls and internally well-connected subgraphs (the communities). *Bridges*, or *long links*, sparsely connect these communities. A user in a social network is typically connected to some communities such as her professional environment, her university friends, her childhood friends, etc. Connections between these communities may exist and the user precisely acts as a bridge between them. Extracting the community structure from a graph is crucial to guide a recommender system to predict links within the community. This extraction is however not straightforward and is a computationally expensive task that is hardly compatible with dynamic graphs. Moreover, the traditional definition of community, still under discussion, is fairly rigid: a vertex either belongs to a community or does not. This does not fit the current Internet setting. Overlapping communities have only been recently considered (see *e.g.* [17]). In this paper, we substitute the (tree-shaped) traditional community structure by a notion of community space, that we call *social* space. In this space, communities are no longer strictly defined as sets, but vertices from the same community belong to the same area. This definition allows vertices in the social space to be in the core or at the border of a community, as well as in multiple communities.

*This work is supported by the ERC Starting Grant, under the project GOSSPLE, number 204742.

CIKM'11, October 24–28, 2011, Glasgow, Scotland, UK.

In this paper, we introduce a novel decentralized algorithm based on (non-isometric) force based graph embedding to assist a link prediction system. More specifically, we propose SoCS (Social Coordinate Systems), a fully distributed graph embedding algorithm that embeds a social graph into a metric space. The embedding achieved by SoCS preserves the community structure of the input graph thus enabling to easily predict links. SoCS predicts links between vertices that turn out to be close in the social space but are not connected in the input graph. SoCS is a very simple algorithm, less than 20 lines of code. SoCS is based on a gossip protocol and does not require any vertex to have a global knowledge of the system.

As we show in the following, force based graph embedding is a natural candidate to distribution. Thus, our approach provides a sound basis for distributed link prediction systems. SoCS can be parameterized with any force-based model and we explore in this paper the application of two well-known energy models to generate a social graph embedding. We provide empirical evidence that the state-of-the-art force model as presented by theoreticians (the LinLog model [15]) is not necessarily the model providing the best results in a practical setting. Additionally, we obtain the surprising result that distributed versions of our protocol, that solely rely on local knowledge, often provide a better accuracy than their centralized counterparts. This demonstrates a connection with local non-linear dimensionality reduction algorithms. We evaluate SoCS in the context of a terrorist collaboration network, a co-authorship dataset, and a synthetic small-world Kleinberg topologies [11].

Our results show that SoCS is able to achieve social embedding: links removed from social networks are accurately predicted, and SoCS enables to clearly distinguish between short and long-range links in a small-world network. We also observe that SoCS is efficient and highly resilient to dynamics. Thus, SoCS is a perfect candidate to implement a contact recommender system in a P2P social network. The rest of the paper is organized as follows. In Section 2, we present background on graph embedding and force-based layout. We present SoCS in Section 3 and provide some experimental results in Section 4. Finally, we review related work in Section 5 and conclude in Section 6.

2. BACKGROUND

In this section, we present the link prediction problem and provide the background on graph embedding required to understand our decentralized protocol. We also introduce the two force models that we consider in this work.

2.1 Link Prediction

The link prediction problem, introduced in [14], consists in predicting the links that are likely to appear in a social network. Most of the existing approaches rely only on the structure of the graph to determine a similarity between users, and rank their predictions according to this similarity. Section 4, we evaluate SoCS against the three following algorithms. The graph shortest path (SP) and the number of common neighbors (CNSP) are used in [14]. In the case of CNSP, we first compute the shortest path between nodes, and then compare the number of common neighbors if the shortest path's length is 2. The hierarchical random graph (HRG) model, presented in [2], relies on the probability that two given nodes belong to the same community to predict links.

2.2 Graph Embedding

Consider $G(V, E)$, an undirected graph of $n = |V|$ nodes representing a distributed system. Let $\mathcal{P}$ be the *host* space and let $\dim \mathcal{P} = d$. A graph embedding is a *mapping* of the graph vertices to positions in the host space. In other words, each vertex gets assigned to a point in the host space.

Generally, the objective of graph embeddings is to preserve the original graph distances in the host space such that the graph shortest path distance between two nodes is equal to the distance between their respective images in the host space. If this property holds for any two nodes, the embedding is said *isometric*. Yet, our goal to achieve a social embedding is to precisely apply some distortion to extract the community structure of a graph so that edges within communities are shorter than those between communities. Therefore, in SoCS, we use low-dimensional host spaces ($d \ll n$): our goal is not to minimize the distortion, since an isometric embedding would not respect the community structure.

2.3 Force-based graph embedding

In this paper, we consider force-based embeddings (FBE), introduced by Eades [4]. Several variations have been proposed since. Intuitively, it is possible to explain the algorithm using a physical analogy: edges represent springs and nodes represent electrically equally charged particles. Edges (springs) attract the vertices they link, whereas vertices (particles) repulse each other. This is the physical system simulated by FBE algorithms. The embedding is achieved once the system reaches an equilibrium.

FBEs are iterative algorithms and rely on two kinds of forces that define the attractions and repulsions that each node is subject to in the host space. Initially, each node is placed at random in the host space. At each iteration of the algorithm, the forces are applied to the nodes. *Attractive* forces are always applied to a node by its *graph neighbors* whereas *repulsive* forces are applied by *all* nodes. Figure 1 illustrates this principle: node a is attracted by b, c and e, and repulsed by b, c, d, e and f. We denote by V_i the graph neighbors of the node i. Let P_i be the image (*i.e.* the position) of node i in the host space. Let $||\overrightarrow{P_iP_j}||$ be the distance between P_i and P_j, and let $\overrightarrow{e_{ij}}$ be the direction from P_i to P_j: $\overrightarrow{e_{ij}} = \frac{\overrightarrow{P_iP_j}}{||\overrightarrow{P_iP_j}||}$.

Following Noack's formalism [16], we define forces as proportional to some power of the distance between two images: attraction force $\overrightarrow{A_i}$ and repulsion force $\overrightarrow{R_i}$ applied to a node $i \in V$ are respectively defined as

$$\overrightarrow{A_i} = \sum_{j \in V_i} ||\overrightarrow{P_iP_j}||^{fa}.\overrightarrow{e_{ij}}, \qquad \overrightarrow{R_i} = -\sum_{j \in V} ||\overrightarrow{P_iP_j}||^{fr}.\overrightarrow{e_{ij}}.$$

Thus each couple (fa, fr), as respective parameters of the attraction and the repulsion forces, defines a new force model.

In this paper, we study two common FBE models. The first one is Noack's *LinLog* [15] defined by $(fa = 0, fr = -1)$, that is considered as the best force model to preserve the community structure of a graph. The second one is the carbon copy of physical Hooke's spring attraction and Coulomb's electrostatic repulsion forces (such as presented in [19]). We denote *HC* this model hereafter, defined by $(fa = 1, fr = -2)$.

3. SOCIAL COORDINATE SYSTEM (SOCS)

In this section, we describe how SoCS performs a distributed embedding of a social graph. For the sake of clarity, we hereafter add the adjective *social* to every element referring to the SoCS host space. For instance the host space of the embedding is designated as the *social space* and the distance in that space is designated as the *social distance*. Nodes that have close *social positions* are social neighbors. At this point it is particularly important to distinguish between the graph neighbors of a node (the friends of this node) and the social neighbors of this node, which are not necessarily the same. Indeed, SoCS will recommend to a node its closest social neighbors that are not already graph neighbors.

In a nutshell, each node that runs a SoCS instance (*i.e.* each graph node) regularly updates its position in the social space. Each node first gathers the positions of its graph and social neighbors. Then, using these positions, each node computes the forces that are applied to it, and derives its updated social position. The rest of the protocol is a gossip protocol that provides each node with a list of its social neighbors. This list of social neighbors is then used to compute new positions.

3.1 System model

SoCS is a peer-to-peer (P2P) algorithm that embeds a social graph and operates on a P2P overlay network. We consider a system of n nodes and assume a one-to-one mapping between the nodes of the social graph and the machines connected to the P2P network, hence we refer as node both to the machine connected to the network and to the logical entity in the social graph. The social graph is an input in SoCS, so each node is aware of its graph neighbors. In a dynamic social graph, the set of graph neighbors may vary during the execution. To join an existing SoCS P2P network, a node contacts its graph neighbors to establish connections. Traditionally, in FBEs, each node is assigned a random initial position. In SoCS, when a node first joins an existing network, its position is initialized at the barycenter of its graph neighbors.

3.2 Rationale

Let us consider the example graph on Figure 1. Assume that this graph represents *friendship* relations between people. Nodes d and f are both two hops away from a. From the shortest path perspective, they thus should be considered equally friends to a. Yet, it is very likely that d is *closer* to a since they have two friends in common, namely c and b. This could result in d having twice more chances than f of being invited to a's birthday party.

Such social information is typically what SoCS aims at capturing in the resulting social embedding: this is achieved since the attractive forces applied between c and b bring d closer to a in the social space. In addition, due to d's repulsions, c and b are closer to a than e. As a result, the list of friends of a in the resulting space is then ordered from the closest to the farthest: b and c, e, d, f.

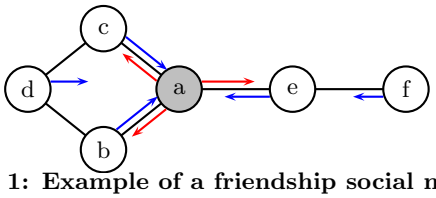

Figure 1: Example of a friendship social network

These results can be interpreted as follows: since (a, b, d, c) is a rectangle, it has more *cohesion* than a single line, and members of that structure get close positions. The generalization of this intuition is that nodes within the same community get closer positions than nodes from different communities. This is exactly the approach advocated by Noack, that recently bridged community detection and force-based graph embedding [16].

Link prediction algorithms are empowered by the knowledge of communities (see *e.g.* [2]). SoCS achieves such a social embedding, capturing, as the evaluation will confirm, the notion of clusters and communities by placing nodes of a community close to each other in the host space. Since links are more likely to appear between nodes of the same community, the social distance (*i.e.* the Euclidean distance over the social space) can be directly leveraged to predict links between nodes.

3.3 Local repulsions

In SoCS, in order to compute its position, a node needs to compute the sum of attractions and repulsions that apply to it. As explained in Section 2, attractions take into account the graph neighbors. Since a SoCS node is directly connected to all its graph neighbors, computing the attractions is straightforward. We now detail how repulsions are approximated and computed.

Computing repulsion forces based on all nodes is both extremely costly in a large system and hard to achieve in a decentralized way. Instead, SoCS only considers repulsion forces of the *close* nodes in the social space. This optimization was introduced by Fruchterman and Reingold [7] to reduce the time complexity of graph drawing algorithms. It is relevant for several reasons. First, in both force models (especially the HC model), the repulsion force quickly decreases with respect to the social distance between nodes: distant nodes' contribution to the force equilibrium is negligible. Second, we target link prediction. We aim at predicting links binding nodes that are socially close. Long social distances do not necessarily need to be precisely respected. We empirically show in Section 4 that considering only local repulsions provides better results while reducing the cost of the algorithm. Let $B(i, r)$ to be the set of i's r closest neighbors in P. SoCS discovers such neighbors and computes the following repulsion force:

$$\overrightarrow{R_i} = - \sum_{j \in B(i,r)} ||\overrightarrow{P_i P_j}||^{fr} . \overrightarrow{e_{ij}}.$$

3.4 The SoCS decentralized algorithm

SoCS is a fully decentralized algorithm: each node knows only a limited portion of the network, namely its graph neighbors and its social neighbors. SoCS relies on gossip to discover the social neighbors. Each node runs a clustering protocol, similar to [20], that we call the *Neighbors Peer Sampling* (NPS). The algorithm is very simple: each node maintains a list of its r closest social neighbors and communicates with them to share this information and discover closer nodes. Gossip protocols have been shown to be cheap, robust against churn, and to converge quickly [9].

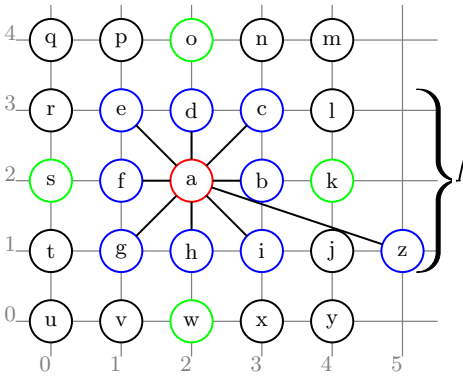

Figure 2: Kleinberg small-world grid

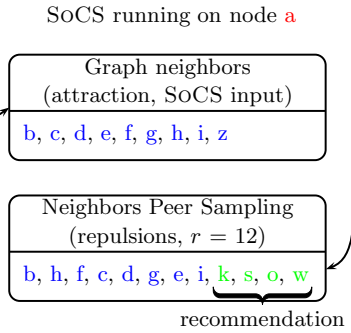

Figure 3: SoCS on node *a*

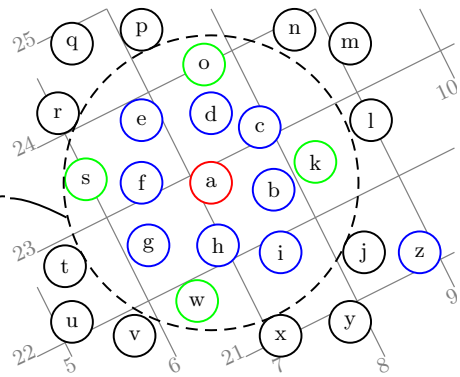

Figure 4: SoCS coordinates space

Algorithm 1 depicts the SoCS algorithm[1]. Lines $1-3$ show initialization. In a static setting (G does not change over time), we assume that nodes are provided with a function `Converged()` that stops the execution of the protocol on a given node. For instance, a node that has not changed its position over the last couple of rounds, may stop. In a dynamic setting, `Converged()` always returns `false`. Two key parameters allow to tune the algorithm: the number of neighbors considered for computing the repulsions, r and the dimension of the host space, d. The impact of both parameters is detailed in Section 4.

Each node retrieves the lists of its social neighbors by calling `GetSocialNeighbors()` (this list is obtained from the NPS) and its graph neighbors by calling `GetGraphNeighbors()` (this list is extracted from the graph). Note that a node can be in both sets, and that the social neighbors list may only contain node positions instead of node identities.

Instead of computing directly a new node position, the presented algorithm computes a couple (speed, direction) toward the ideal position (lines $12-13$). This improvement, also used in [7], allows a faster convergence by speeding up nodes in the early steps while avoiding oscillations during the late ones.

Figures 2,3 and 4 illustrate SoCS running on a small-world grid. They respectively illustrate Kleinberg's graph generation process, SoCS' input and ouptput on node a, and an illustration of SoCS' system-wide output. Blue nodes represent a's graph neighbors, green nodes represent a's predicted links.

Algorithm 1 Graph embedding algorithm, code for Node i

1: $P_i \leftarrow$ `GetBarycenter(GetGraph())`
2: $v_{prev} = 1$
3: $\overrightarrow{d}_{prev} \leftarrow$ new `RandomDirection()`
4: **while not** `Converged()` **do** ▷ Execute a cycle
5: $\quad\overrightarrow{d} \leftarrow \overrightarrow{0}$
6: $\quad$**for** $j \in$ `GetGraphNeighbors()` **do**
7: $\quad\quad\overrightarrow{d} \leftarrow \overrightarrow{d} + ||\overrightarrow{P_iP_j}||^{fa}.\overrightarrow{e_{ij}}$ ▷ Compute attractions
8: $\quad$**end for**
9: $\quad$**for** $j \in$ `GetSocialNeighbors()` **do**
10: $\quad\quad\overrightarrow{d} \leftarrow \overrightarrow{d} - ||\overrightarrow{P_iP_j}||^{fr}.\overrightarrow{e_{ij}}$ ▷ Compute repulsions
11: $\quad$**end for**
12: $\quad\overrightarrow{d} \leftarrow \overrightarrow{d}/||\overrightarrow{d}||$
13: $\quad v = (\frac{1}{2}.\cos(\overrightarrow{d}, \overrightarrow{d}_{prev}) + 1)v_{prev}$ ▷ Compute speed
14: $\quad\overrightarrow{d}_{prev} = \overrightarrow{d}$; $v_{prev} = v$
15: $\quad P_i = P_i + v.\overrightarrow{d}$ ▷ Move
16: **end while**

4. EXPERIMENTAL EVALUATION

We experiment SoCS through two different approaches. First, we measure SoCS' accuracy in a link prediction application using two different datasets: a computer science co-authorship graph (namely DBLP) and a terrorist interaction graph. We compare these results against some existing approaches. Then, in the second part of this section, we evaluate SoCS on synthetic small-world graphs. This setup provides a full knowledge of the desired social space and allows us to perform a more precise evaluation of SoCS, both in a static setting and in a dynamic network.

In our experiments, we evaluate the impact of SoCS' parameters, namely the number of repulsions applied to nodes (r, see Section 3.3), the force model (LinLog or HC, as described in Section 2.3), and the number of dimensions in the SoCS coordinates space. We show that, quite surprisingly, the configurations that achieve the best prediction quality are very close to the ones that perform well in a distributed environment. Finally, please note that, although SoCS is implemented in a java simulator, it runs in a fully distributed setting: each node is only provided with the list of its graph neighbors identities. In particular, SoCS does not rely on any assumption about the nature or the size of the graph.

[1]Java binaries and complete source code are available: `http://www.irisa.fr/asap/coord`

4.1 SoCS link prediction quality

In this section, we consider two different social graphs. The first one is a terrorist association network [13] of 62 nodes and 152 edges, used in [2] to evaluate HRG. The second one is the DBLP dataset, from which we generate a co-author graph. We create an edge between the authors having at least one publication in common, while removing the authors that have less than 20 publications. We keep the largest connected component: $27,680$ nodes connected by $211,078$ edges. In order to perform the link prediction, we rely on the standard approach consisting in removing some edges from the graphs and trying to predict them. The edges removed are always selected at random, yet without disconnecting the graph.

We compare SoCS' performance with SP, CNSP and HRG (described Section 2.1). We consider the case of a static network: the social graph is not modified during the execution and SoCS runs until convergence. We use ROC curves, as well as the AUC measure to display the results.

Figure 8 displays the AUC of the link prediction on the

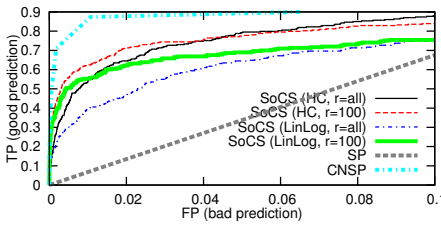

Figure 5: DBLP link prediction

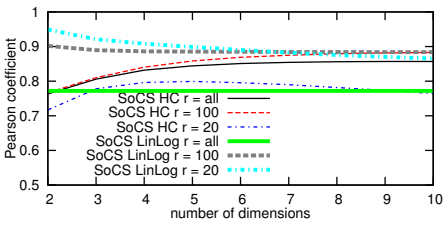

Figure 6: local Pearson coefficient on converged state

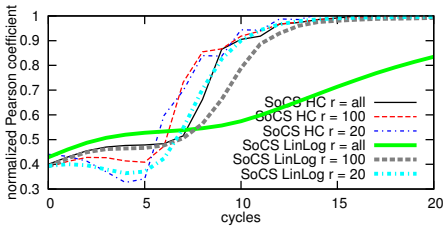

Figure 7: normalized local Pearson during cold start

terrorist association graph for an increasing number e of edges removed. We configure SoCS to use 10 dimensions. As already shown in [14], SP does not perform well, while CNSP is very accurate. We observe that SoCS performs best with the HC force model. It's also very interesting to notice that a lower value of r also improves the accuracy of the prediction. By reducing the amount of information gathered by each node, SoCS eliminates the impact of distant nodes and reduces the noise in the data: nodes get a better position. With HC, SoCS outperforms HRG at each level of sparsity.

The results obtained on the DBLP dataset, presented on Figure 5, confirm these observations. In that case, 1% of the edges were removed for prediction. We configure SoCS to use 20 dimensions. We were not able to evaluate HRG on this graph for the implementation provided by the authors quickly runs out of memory on a 16GB memory computer when applied to such a large graph. On smaller graphs, SoCS runs faster and uses much less memory than HRG.

Again, SoCS performs best in the HC force model with a small number of repulsions. This is a very positive indicator of SoCS' scalability, since repulsions consume bandwidth.

SoCS is always outperformed by CNSP. However, we argue that CN and SP both require the computation of all shortest paths, which has a prohibitive cost especially in a P2P environment facing churn: each node joining or leaving potentially impacts many shortest paths. In SoCS, nodes only exchange lists of neighbors in the SoCS coordinates space. Additionally, CNSP is known to perform very well in many social graphs [14], and we believe that the setup of this experiment largely favors the CNSP metric: scientific publications often have more than 2 authors, inducing a highly redundant common neighbors (CNSP) set. SP also outperforms SoCS after predicting 60% of the missing edges. However, similarly to a top-k ranking, we believe that first (say, 1%) predictions are the most important.

4.2 SoCS applied to a small-world network

In this section, we consider a synthetic small-world network. We rely on the Kleinberg model [11] to generate a small world network. In this model, nodes are first linked according to a lattice graph. These links represent short links. Then, long links are drawn between nodes representing the inter-community links. Figure 2 illustrates the process: node a has short links to nodes b to i, and a long link to z. We extract the resulting graph and use it as the SoCS input. All the results we present are averaged over 100 different randomly generated graphs.

e	SP	CNSP	HRG	SoCS HC		SoCS LinLog	
				$r = 20$	$r = all$	$r = 20$	$r = all$
1	.851	.914	.881	.899	.894	.869	.864
10	.841	.904	.884	.900	.896	.868	.864
20	.843	.888	.868	.883	.881	.854	.850
30	.837	.871	.859	.869	.869	.837	.835

Figure 8: Terrorist network link prediction AUC

4.2.1 *Static system evaluation*

After SoCS converged, it provides each node with a position in the social space, as illustrated on Figure 4. It is then possible to compute how closely the SoCS social coordinates match the ones in the grid used during the graph generation (Figure 2). We are not interested in the absolute value of the coordinates, nor by any scaling or rotation that could occur, as illustrated on Figure 4. Our goal is to have distances in the SoCS social space proportional to the ones in the grid. We first compute the Pearson correlation between all node pairs' distances. Contrary to the previous results, the best performance (0.95) is obtained with the LinLog force model and with a large value of r, and the performance slightly decreases when the number of dimensions increases. However, as observed in Section 4.1, the local positioning of nodes has much more impact on the quality of the link prediction than the accuracy of the distance between far away nodes. Hence, we define a second measure that we call the local Pearson coefficient. For each node, we compute the Pearson correlation by only considering the distances to graph neighbors, and we average this result over all the nodes. As Figure 6 shows, the best results are achieved through small values of r. Increasing the number of dimensions improves the results with HC, while Linlog performs better with a small d. These results confirm our experiments on link prediction: SoCS should be configured with a small r in order to generate an accurate short-range positioning without taking into account noise from distant nodes.

A second experiment consists in using SoCS to allow nodes to differentiate between short and long range links. This problem was posed by Kleinberg in [12] and was already theoretically addressed [6]. Due to space limitations, we do not present these results. SoCS provides an accurate and more flexible experimental approach to this problem.

4.2.2 *Dynamic system evaluation*

In this section, we consider SoCS in a dynamic system, typical of a deployed P2P network. Users may join and leave the system (aka churn), creating perturbations. SoCS relies on gossip protocols, which have been extensively studied within different churn conditions and have been shown to be very resilient, even in the case of massive failures. In SoCS, the arrival or departure of a node modifies the underlying social graph, therefore, the SoCS coordinates should be updated in order to reflect these modifications. In this section, we study the quality of the SoCS coordinates during different churn conditions.

We first consider the case of a "cold start". The $1,024$ nodes of the small-world network simultaneously join, without running any intermediate SoCS coordinate adjustment. Figure 7 displays the normalized local Pearson coefficient for different SoCS configurations. The value 1 is obtained once the system has converged. The HC force model reaches sta-

bility much faster (8 cycles), than the LinLog one (up to 40 cycles). We also observe that the convergence time decreases with r. A massive churn scenario, in which half of the network converges and the other half simultaneously joins leads to the same observations, but with a faster convergence. Indeed, nodes joining the network leverage the other nodes' positions to obtain a fairly accurate initial placement. Regular churn, in which nodes join and leave the network at each cycle, has almost no impact on SoCS' precision. When r is low, the perturbations caused by churn only affect the few nodes that are socially close to the node leaving or joining. Such local perturbations are quickly absorbed.

These experiments show that the HC force model is more adapted to churn than LinLog. This is easily explained by the impact of repulsions: their weight is much lower in the HC model ($fr = -2$), so they generate less perturbations. Taking only the closes nodes into account, i.e. choosing a small r, is crucial for handling churn. It increases the convergence speed of SoCS and makes the system more resilient to churn. The previous experiments showed that a small r and the HC force model led to a better link prediction. Thus, SoCS does not suffer from its distributed nature, link prediction is inherently a distributed application in which each node considers its neighbors while ignoring the perturbation from the distant other nodes.

5. RELATED WORK

To the best of our knowledge, this paper presents the first distributed link prediction algorithm based on embeddings. As opposed to FBE, another approach to graph embedding relies on the graph eigenvectors [8]. A distributed algorithm to find these eigenvectors was proposed by Kempe and McSherry [10]. However, the application of this work to link prediction is not straightforward, and the stability of this algorithm when facing churn or dynamics of the graph has yet to be studied.

Assigning Internet coordinates to nodes in a fully distributed way is a problem that received a lot of attention. One of the most well known system is Vivaldi [3], a distributed protocol that aims to predict Round Trip Times (RTT) between nodes on the Internet. The main difference with our work is the goal (predicting RTTs) and the fact that we do not target isometric embeddings. Furthermore, in Vivaldi, nodes can easily compute, through a ping measure, their optimal distance to any given node and adjust their position accordingly. In SoCS, the nodes only have information about their graph neighbors and cannot do such a measure with respect to any node in the graph.

Another class of works that relate to SoCS is dimensionality reduction (DR). DR consists in discovering close items from a set of items described by highly dimensional data. These methods are heavily used in machine learning and recommender systems, where considering variables correlated in a manifold (such as the distances between cities of the Earth globe) is common. DR algorithms are used to flatten this data into a Euclidean space (a world map). The approach followed by these methods (*e.g.* KPCA, LMDS [1], LLE [18]) is to achieve DR while preserving the "local structure" of the data. Recently, Chen and Buja [1] pointed out the close link between graph drawing and non linear DR: SoCS can be seen as a non linear DR algorithm that "flattens" the social manifold. The importance of locality in non-linear DR fits SOCS local repulsion approximation.

6. CONCLUSION

In this paper, we presented SoCS, a fully distributed algorithm for social graph embedding. SoCS distributes traditional force-based embedding algorithms to embed graphs while preserving their community structure [16]. SoCS exploits the benefits of the community structure knowledge for link prediction [2] to achieve meaningful recommendations in a social network. SoCS relies on gossip protocols to approximate the forces applied to the nodes and achieve scalability in large and dynamic graphs.

We extensively analyzed the performance of SoCS on both synthetic and real data. Our experiments show that SoCS is able to accurately assign social coordinates to nodes. The best results are obtained with configurations favoring the local accuracy over the global one. Not only does this setting improve the quality of the results, but it also offers better scalability and more resilience to churn.

SoCS can be used directly to predict links in a social network. In addition, this can form the basis for a more complete recommender system. Future work includes other force models evaluation, especially from their p2p applicability perspective, as well as achieving a better understanding of the algorithm parameter space. The privacy-protecting properties of SoCS have also to be studied.

7. REFERENCES

[1] L. Chen and A. Buja. Local multidimensional scaling for nonlinear dimension reduction, graph drawing, and proximity analysis. *JASA*, 104(485):209–219, 2009.

[2] A. Clauset, C. Moore, and M. Newman. Hierarchical structure and the prediction of missing links in networks. *Nature*, 453(7191):98–101, 2008.

[3] F. Dabek, R. Cox, F. Kaashoek, and R. Morris. Vivaldi: A decentralized network coordinate system. In *SIGCOMM*, pages 15–26, 2004.

[4] P. Eades. A heuristic for graph drawing. *Congressus Numerantium*, 42:149–160, 1984.

[5] A. Evfimievski, J. Gehrke, and R. Srikant. Limiting privacy breaches in privacy preserving data mining. In *PODS*, pages 211–222, 2003.

[6] P. Fraigniaud, E. Lebhar, and Z. Lotker. Recovering the Long-Range Links in Augmented Graphs. In *SIROCCO*, page 118, 2008.

[7] T. M. J. Fruchterman and E. M. Reingold. Graph drawing by force-directed placement. *SPE*, 21(11):1129–1164, 1991.

[8] K. M. Hall. An r-dimensional quadratic placement algorithm. *Management Science*, 17(3):219–229, 1970.

[9] M. Jelasity, S. Voulgaris, R. Guerraoui, A.-M. Kermarrec, and M. van Steen. Gossip-based peer sampling. *TOCS*, 25(3), 2007.

[10] D. Kempe and F. McSherry. A decentralized algorithm for spectral analysis. In *STOC*, pages 561–568, 2004.

[11] J. Kleinberg. The small-world phenomenon: An algorithmic perspective. In *STOC*, pages 163–170, 2000.

[12] J. Kleinberg. Complex networks and decentralized search algorithms. In *ICM*, volume 3, pages 1019–1044, 2006.

[13] V. Krebs. Mapping networks of terrorist cells. *Connections*, 24(3):43–52, 2002.

[14] D. Liben-Nowell and J. Kleinberg. The link-prediction problem for social networks. *ASIS&T*, 58(7), 2007.

[15] A. Noack. An energy model for visual graph clustering. In *Graph Drawing*, pages 425–436, 2003.

[16] A. Noack. Modularity clustering is force-directed layout. *Physical Review E*, 2009.

[17] G. Palla, I. Derényi, I. Farkas, and T. Vicsek. Uncovering the overlapping community structure of complex networks in nature and society. *Nature*, 435(7043):814–818, 2005.

[18] S. Roweis and L. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323, 2000.

[19] F. van Ham and J. J. van Wijk. Interactive visualization of small world graphs. In *INFOVIS*, pages 199–206, 2004.

[20] S. Voulgaris and M. V. Steen. Epidemic-style management of semantic overlays for content-based searching. In *EuroPar*, pages 1143–1152, 2005.