ACO-inspired Acceleration of Gossip Averaging

Andreas Janecek
University of Vienna, Faculty of Computer Science, Vienna, Austria
andreas.janecek@univie.ac.at

Wilfried N. Gansterer
University of Vienna, Faculty of Computer Science, Vienna, Austria
wilfried.gansterer@univie.ac.at

ABSTRACT

Gossip ("*epidemic*") algorithms can be used for computing aggregation functions of local values across a distributed system without the need to synchronize participating nodes. Although several (theoretical) studies have proven that these algorithms scale well with the number of nodes n, most of these studies are restricted to fully connected networks and based on rather strong assumptions, *e. g.*, it is often assumed that all messages are sent at exactly the same time on different nodes. Applying gossip algorithms on non-fully connected networks significantly increases the number of messages / rounds, especially on weakly connected networks without a regular structure.

We present new acceleration strategies for gossip-based averaging algorithms based on ant colony optimization, which specifically target weakly connected networks with irregular structure, where existing gossip averaging algorithms tend to be slow. The proposed acceleration strategies reduce the message and time complexity of standard gossip algorithms without any additional communication cost. The overhead only consists of additional local computation which is proportional to the node degree. All findings are confirmed experimentally for different types of network topologies and for different network sizes.

Keywords

Applications of Ant Colony Optimization, Gossip-based Averaging, Epidemic Aggregation, Acceleration

1. INTRODUCTION

We focus on gossip-based averaging methods where every node only communicates with its direct neighbors, *i. e.,* only *only single-hop* communication is performed. We distinguish between different types of information exchange between nodes (Push vs. PushPull) and the choice of the communication partner in each round (uniformly random vs. biased based on the history of the process). The main contribution of this paper is the proposal of several acceleration strategies for gossip-based averaging algorithms inspired by ant colony optimization (ACO) for biased neighbor selection. The choice of ACO is motivated by the similarity of nature-inspired population-based algorithms such as ACO and distributed gossip algorithms: both are based on a population of entities who communicate via simple rule-sets, both iteratively improve current approximations of the solution, and both are stochastic in nature. The proposed acceleration strategies are able to reduce the communication cost by a factor of up to 2.7 compared to state-of-the-art gossip algorithms using standard sampling from a uniform distribution.

Related Work. Although the combination of gossiping with ACO and other techniques from swarm intelligence and evolutionary computation has been investigated for the task of rumor spreading [10, 13, 14, 26], to the best of our knowledge it has not been considered yet for gossip-based averaging. Various approaches to accelerating distributed aggregation methods have been studied. Since gossip averaging can be represented as an iterative linear iteration (with a distributed communication matrix), classical convergence acceleration techniques in the form of stationary iterative methods, such as the second-order Richardson method [12] (known in load balancing as *second-order scheme* [23]) or Chebyhsev acceleration with time-dependent coefficients [12], are in principle applicable. However, for gossip-based algorithms where the communication matrix is time-dependent and for the fully decentralized setting which we consider where no global view of the communication matrix is available, the applicability of such acceleration techniques is rather limited.

Besides classical convergence acceleration schemes for linear iterations, faster convergence can also be obtained via exploiting additional assumptions on the distributed environment. Several conceptually different approaches have been employed. For instance, one can optimize the communication patterns with respect to a fixed topology [4] or optimize the topology itself [17]. Moreover, by assuming that the nodes have additional (global) knowledge about the topology of the system as well as about their position and by using multi-hop communication, substantial improvements can be achieved [1, 6]. However, in this paper we consider a more general setup without such additional assumptions and only nearest-neighbor communication, and thus these acceleration techniques are not applicable in our context.

Compared to existing acceleration strategies, our approach has several advantages: Absolutely no topology information is needed, it can be applied to any type of network topology, and it does not cause any communication overhead (nei-

GECCO '16, July 20-24, 2016, Denver, CO, USA
 ISBN 978-1-4503-4206-3/16/07…$15.00
DOI: http://dx.doi.org/10.1145/2908812.2908832

ther in terms of messages nor in terms of data sent). Moreover, our approach is consistent with the fundamental "rules of thumb" for gossip protocols which have been formulated in [21]. Although we focus on averaging, our approach can be extended to other types of aggregation functions, such as sum, variance, etc.

2. GOSSIP-BASED AVERAGING

In distributed averaging tasks, the goal is to calculate the average $\bar{v}$ of a set of initial values v_i (e. g., sensor measurements, document attributes, media ratings, etc.) stored locally at n nodes (v_i is the initial value on n_i). One possibility for computing $\bar{v}$ in a completely distributed manner is to use gossip ("*epidemic*") algorithms. Because of their inherently probabilistic nature (randomized communication schedules), these algorithms have the potential for tolerating dynamic network changes, node failures, or data loss and thus for providing a high degree of resilience and fault tolerance [22]. Moreover, they allow for gradually trading runtime performance against communication cost, fault tolerance or energy consumption by adapting the intensity of interaction between nodes. This is very attractive in situations where classical parallel or fusion center approaches or flooding-based solutions are too expensive, where runtime is not the most important aspect, or where not only the final "correct" result, but also approximate intermediate results are of interest.

2.1 Push Averaging (the PushSum Algorithm)

In purely push-based algorithms information is pushed to other nodes without receiving an answer, *i. e.,* without pulling for information. For computing $\bar{v}$, each node i additionally holds a value x_i and a weight w_i. Initially, $x_i(0)$ is set to the local value at each node, *i. e.,* $x_i(0) = v_i$. For computing $\bar{v}$, all weights w_i are initialized with 1. In order to compute $\sum_{i=1}^{n} v_i$ instead of $\bar{v}$, the weight of a single node is set to 1 and the weights of all other nodes to 0. Typical push averaging algorithms are given in [15, 19] and consist of an active and a passive thread. In the active thread, each node selects a random communication partner p, sends half of its value ($x_i/2$) and half of its weight ($w_i/2$) to p, and halves its own value and weight. If the passive thread receives a message (*i. e.,* an "update") from node i, it adds the received value x_i and received weight w_i to its own value and weight, respectively. At all times t, the current estimate of the average $\bar{v}$ on each node can be computed as $x_i(t)/w_i(t)$. The *PushSum* algorithm preserves the mass conservation invariant — at any time the sum of all values (*i. e.,* approximations) in the network remains constant throughout the algorithm. As we will see in Section 2.2, mass conservation is an important advantage of PushSum over PushPull approaches.

2.2 PushPull Averaging

PushPull also consist of an active and a passive thread [15]. However, in contrast to *unidirectional* PushSum, averaging is based on *exchange* of information. In the active thread, each node i selects a random neighbor p as communication partner and pushes its current local estimate x_i to p. Node p receives the packet from i (the sender), replies with its own current estimate x_p, and stores the average of the received and its own estimate (x_i, x_p) as its new estimate. Finally, the sender receives the answer and updates its own local estimate. As opposed to PushSum, no separate weights are needed — if all weights are initialized with the same value (*e. g.*, 1), they would remain unchanged after each bidirectional information exchange. As an example, consider that the sender i would halve its weight (*i. e.,* $w_i = 0.5$) before sending the value/weight pair to p. Node p would do the same (*i. e.,* $w_p = 0.5$) before sending its own value/weight pair to node i, and both communication partners would store the sum of both weights ($w_i + w_p \equiv 1$) as new weight.

In information dissemination tasks (*i.e.,* rumor spreading), PushPull variants for rumor spreading can reduce the amount of update messages to only $O(n \log\log n)$ as compared to $O(n \log n)$ for Push variants, cf. [18]. However, for averaging problems there is no distinction between possibly large update messages and empty request messages. Thus, although PushPull approaches usually need fewer rounds than PushSum, it is not guaranteed that PushPull generally also reduces the number of messages, since PushPull has the drawback that the number of messages per round is doubled. Moreover, the mass conservation invariant can be violated in PushPull algorithms if an atomic violation happens, *i. e.,* if a node receives a push while it is waiting for a pull-reply. Existing extensions of PushPull averaging algorithms that guarantee mass conservations (cf. [16]) typically suffer from the drawbacks of either increasing the number of rounds or penalizing the selection of some nodes.

2.3 Implementation Strategies

The algorithms from Section 2 can be implemented as purely round-based (denoted as R) or as round *and* event-based (denoted as RE) implementations. Moreover, the *update process* may have a significant influence on the number of rounds (sometimes also called *cycles* or *iterations* in the literature) as well as on the total number of messages necessary to achieve a given accuracy. In a *separate update* strategy, all nodes send their messages at the same time — as often required in the literature, cf. [15]. The update of received information can only be performed after all nodes have finished sending in each round. On the contrary, in an *immediate update* strategy, all nodes send their packets at slightly different times. Thus, it is possible that the update based on received information is performed *before* a node performs the sending process in the current round. Although separate update strategies can be applied for PushSum, they cause problems for PushPull, since mass conservation is not guaranteed (cf. Section 2.2). In order to allow for a comparison of PushSum and PushPull, we focus only on immediate update strategies in the rest of this paper.

2.3.1 Implementing PushSum

In **purely round-based PushSum** implementations each node sends one packet in each round — independent of the number of packets a node has received in the last round. Algorithm 1 shows the main steps that are performed *in each round*. Due to the immediate update strategy, sending and receiving can be performed within the same loop, since it is assumed that all nodes perform sending at slightly different times, and that the passive thread at each node can receive and process packets immediately. In **purely round-based PushSum** implementations each node sends one packet in each round — independent of the number of packets a node has received in the last round. Algorithm 1 shows the main steps that are performed *in each round*. Due to the immediate update strategy, sending and receiving can be performed

Algorithm 1: PushSum_R

```
1 for i = random permutation over all n do
2 |   p ← random neighbor of node i;
3 |   x_i ← x_i/2;            //At sender:  update value
4 |   w_i ← w_i/2;            //At sender:  update weight
5 |   x_p += x_i;             //At receiver:  update value
6 |   w_p += w_i;             //At receiver:  update weight
7 end
```

within the same loop, since it is assumed that all nodes perform sending at slightly different times, and that the passive thread at each node can receive and process packets immediately. The random permutation over all n is important, since a loop with a fixed sequence (*for* 1 to n loop) would unrealistically let the same nodes perform sending prior to other nodes in each round.

Round and event-based PushSum. Instead of allowing each node to send one message in each round, it is also possible to let each node send as many packets as it has received in the previous round. There are still n packets sent in each round, but not all nodes are allowed to send packets — instead, some nodes may send several packets in the same round. This approach is still round-based as the communication is performed in rounds, but it is also event-based as the number of received packets indicates the number of packets that can be sent in the next round, cf. Algorithm 2. At each node, *nRecLastRound* stores the number of packets that each node has received in the last round. *nRecLastRound* is initialized with ones, *i. e.,* in the first round, each node sends one packet, and *necThisRound* is initially set to 0.

Algorithm 2: PushSum_RE

```
1 for i = random permutation over all n do
2 |   for j = 1:nRecLastRound(i) do
3 |   |   p ← random neighbor of node i;
4 |   |   nRecThisRound_p += 1;
5 |   |   ... //identical to lines 4-7 of Alg. 1
6 |   end
7 end
8 nRecLastRound ← nRecThisRound;
9 nRecThisRound ← 0;
```

2.3.2 Implementing PushPull

Purely round-based PushPull. Algorithm 3 shows the main steps that are performed in each round for PushPull_R. Each node sends one packet per round. Whenever a node p receives a (push) message from node i (the sender), the current local value of p is stored in a temporary variable tx_p (Line 4) and the local value of p is updated with received local value of i (Line 5). Finally, the local value of i is updated with the temporarily stored previous local value of p (Line 6). Recall that a delayed update at either node p or node i will probably lead to an atomic violation.

Round and event-based PushPull. The update of PushPull_RE in Algorithm 4 is similar to the purely round-based variant, but the selection process of sending nodes is different. Again, $2 \times n$ messages are sent in each round.

Algorithm 3: PushPull_R

```
1 for i = random permutation over all n do
2 |   p ← random neighbor of node i;
3 |   tx_p ← x_p;                  //store tx_p temporarily
4 |   x_p ← (x_i + x_p)/2;       //At receiver p:  update x
5 |   x_i ← (x_i + tx_p)/2;      //At sender i:  update x
6 end
```

Algorithm 4: PushPull_RE

```
1 for i = random permutation over all n do
2 |   for j = 1:nRecLastRound(i) do
3 |   |   p ← random neighbor of node i;
4 |   |   nRecThisRound_p += 1;
5 |   |   ... //identical to lines 4-6 of Alg. 3
6 |   end
7 end
8 nRecLastRound ← nRecThisRound;
9 nRecThisRound ← 0;
```

3. ACCELERATION BASED ON ACO

Now we present new acceleration strategies inspired by ACO. Compared to basic gossip-based averaging algorithms where communication partners are selected solely based on random decisions, our acceleration strategies feature a biased neighbor selection method to accelerate the diffusion speed of gossip algorithms. Our approach exploits a concept from ant colony optimization (ACO), a well-established swarm intelligence (SI) technique for finding optimal routes in graphs based on the concept of ants using pheromone trails.

Ant Colony Optimization (ACO) [7, 8, 9] is motivated by the behavior of living ants, which can find the shortest path to a food source by laying and following pheromone trails. ACO aims at finding approximate solutions to optimization problems, such as finding good paths through graphs. This mechanism of communication by modifying the environment is also referred to as *stigmergy*. Consider two solution states i and j (in a shortest-path problem, cf. [7], i and j could represent two different nodes). Following [8], the transition probability for ant k from i to j is defined as

$$p_{ij}^k = \frac{(\tau_{ij}^\alpha)(\eta_{ij}^\beta)}{\sum_{j \in allowed_i} (\tau_{ij}^\alpha)(\eta_{ij}^\beta)} \quad (1)$$

where τ_{ij} is the intensity of the pheromone trail on edge (i, j), η_{ij} is the desirability of the solution state transition ij (for example, the inverse distance between two nodes i and j), and α, β are parameters to control the influence of τ_{ij} and η_{ij}, respectively. The data structure $allowed_i$ contains all allowed solution states for ant k in state i. After all ants have completed a solution, the trail intensity is updated by $\tau_{ij} = \rho \cdot \tau_{ij} + \sum_k \Delta\tau_{ij}^k$, where ρ is the pheromone evaporation coefficient (1-ρ represents the evaporation of the trail) and $\Delta\tau_{ij}^k$ is the amount of pheromone deposited by the kth ant.

Motivation. The application of SI in distributed environments is motivated by the fact that many natural examples of SI such as ant colonies or bird flocking are based on (simple) individuals (also referred to as *agents* or *boids*) that communicate to develop collective behavior in a purely decentralized and self-organized fashion [3, 20]. These char-

acteristics make these natural systems robust to loss of members and adaptable to a changing problem domain — all properties which are highly demanded in the context of distributed computing environments. Indeed, SI methods are in many aspects similar to gossip algorithms: individuals communicate *iteratively* with one another in a self-organizing way, and their decisions are based on probabilistic components combined with some simple rule sets based on local information. However, whereas gossiping protocols usually communicate completely randomly, SI is biased towards an optimal solution — a very desired aspect in our context.
Despite some attempts to apply them in distributed environments (cf. [5]), many current SI implementations are often implemented as centralized, synchronous and sequential algorithms (see the discussions in [24, 25]). Examples of such centralization aspects include all kinds of rankings or elitism (e.g., the global best in particle swarm optimization, cf. [20]), but also global pheromone updates in ACO. Also partly because of these centralization aspects, the main application fields of SI are in the domain of meta-heuristics that aim at creating approximate solutions to hard search and optimization techniques, such as parameter optimization or routing problems (cf. [2]). We follow a different approach and utilize the concepts of ACO in a completely distributed network, which allows for applications in completely decentralized and asynchronous environments. Due to this distributed setting, the pheromone update has to be performed locally on each node. The learning encoded in the pheromones is not directly based on the pseudo-random proportional rule (Equ. 1) but rather on the similarity/dissimilarity of the estimates of neighbors compared to the local estimate of nodes. Moreover, our "ants" are restricted to local movements only.

3.1 Acceleration Using Local Knowledge

We propose four slightly different acceleration strategies for PushSum and PushPull gossip, which are based on the same problem statement and setup: Consider the scenario that each node in the network maintains a pheromone deposit for each outgoing link, as typical in ACO. The amount of pheromone along a directed link between a sending node and its neighboring nodes influences the probability of selecting a neighboring node as communication partner. The main idea of our approach is to accelerate the diffusion speed of gossip algorithms by selecting links with higher pheromone value with higher probability. In the following, we describe how the amount of pheromone along a path is calculated, and how the (inverse) concept of evaporation is included in our approach. For PushSum, the local estimate at node i of the global average $\bar{v}$ can be computed as x_i/w_i, while for PushPull the local estimate equals x_i (no weights in PushPull). In order to improve readability, we use the variable λ_i for all algorithms to refer to the local estimate at node i. For PushSum, $\lambda_i = x_i/w_i$, and for PushPull $\lambda_i = x_i$.

3.1.1 ACO-based Acceleration for PushPull

First, we discuss the concept for PushPull variants. We will see later that PushSum variants are more difficult in terms of neighbor selection since there is no bidirectional information exchange. At all times t, every node i has a current estimate λ_i of the average $\bar{v}$. Beyond that, node i also has (possibly outdated) information about the estimates of its neighbors, stored in the vector $\vec{y}_i$ of length $deg(i)$. The elements of $\vec{y}_i$ are ordered according to the IDs of the neighbors of i.

Example. Consider a node a connected to nodes b, c, d, e. Whenever a communicates with one of its neighbors, it updates its own estimate λ_a, *and* also $\vec{y}_a$. The absolute difference between $\vec{y}_a$ and λ_a is denoted as $\vec{d}_a$ ($\vec{d}_a = |\vec{y}_a - \lambda_a \mathbf{1}|$), and serves as the basis for our biased communication partner selection. Motivated by the concept of ACO, $\vec{d}_a$ represents the intensity of the pheromone trail along the edges between a and its neighbors. Node a will now choose edges with higher pheromone values with higher probability. In other words, it is more likely that node a selects a node with a strongly different local estimate than a node whose local estimate is very similar. The rationale behind this idea is that more progress towards the true average will be made if an information exchange brings more new information.
Clearly, since only *local knowledge* about neighboring nodes is available, most elements in $\vec{y}_a$ and therefore also $\vec{d}_a$ will be outdated since a does not always know the true current estimate of all of its neighbors. *E.g.,* consider that in round t nodes b and a exchange information, and that in round $t+1$ node a exchanges information only with node c while b exchanges information with another node. At the end of round $t+1$ the information of node a about the estimate of b is outdated and probably differs (at least slightly) from $b's$ current estimate. However, as we will see later, even partly outdated estimates are better than standard PushPull without any information about the neighbors.
In ACO, the attractiveness of a pheromone trail is reduced as the pheromones evaporate over time. We exploit this strategy in the opposite direction and increase the attractiveness of edges over time in order to increase the attractiveness of nodes which have not been chosen for a long time. This is especially important for PushSum, where this strategy also ensures that no neighboring node is excluded (see next section). Whenever node a communicates with a neighbor — either as the active sender which sends pull-requests, *or* as the receiver which returns pull-replies — it stores the number of the current round in the vector $\vec{t}_a$. The elements of $\vec{t}_a$ are also ordered according to the IDs of the neighbors of a. In our example, $\vec{t}_a(1)$ contains the information in which round the latest information exchange between a and b happened, independently of which of the two nodes initiated the information exchange. For all gossip algorithms, ACO-based acceleration can be implemented using a roulette-wheel selection or a greedy selection strategy.

Roulette-wheel selection. The vectors $\vec{d}_a$ and $\vec{t}_a$ are used to calculate the vector $\vec{p}_a$ (p for probability) according to

$$\vec{p}_a = \vec{d}_a^{\,\alpha} \otimes (t\mathbf{1} - \vec{t}_a\,), \tag{2}$$

where exponentiation is meant element-wise (the parameter α can be used emphasize edges with high pheromone trails), the symbol $\otimes$ represents element-wise multiplication, t refers to the number of the current round, and $\mathbf{1}$ refers to the identity matrix. After normalization, $\vec{p}_a$ contains probabilities for selecting each neighboring node. Based on these probabilities, the node selection is then performed as *roulette-wheel* or *fitness proportionate* selection ([11]), as commonly used in genetic algorithms, where a *fitness level* is used to associate a probability of selection for each chromosome. If f_i is the fitness of individual i, its probability of being selected is $p_i = \frac{f_i}{\sum_{j=1}^{N} f_j}$, where N is the number of individuals.
In order to exploit this principle for our acceleration strategies, we replace f_i with the probabilities $\vec{p}_i$ and N with the

number of $i's$ neighbors. In our example we replace f_i with the probabilities $\vec{p}_a$ of $a's$ neighbors (*i. e.,* nodes b, c, d, e) and N with the number of $a's$ neighbors (*i. e.,* 4).

Greedy selection. This strategy selects the node with the most different estimate. All values of $\vec{d}_a$ which are smaller than the maximum value of $\vec{d}_a$ are set to 0 before calculating the vector $\vec{p}_a$ in Equ. 2. If there is only one unique maximum value in $\vec{d}_a$, this node will be selected. If there are two nodes with the same maximum difference, then the roulette wheel selection is used. *I. e.,* frequency of iteration ($t\mathbf{1} - \vec{t}_a$ in Equ. 2) is also an issue in this case. The influence of the parameter α on the results of the greedy selection strategy is negligible, thus in our experiments we set $\alpha = 1$ whenever we applied greedy selection (however, any fixed value for $\alpha > 0$ will work). Both, the roulette-wheel selection and the greedy selection strategy can also be applied to PushSum.

3.1.2 ACO-based Acceleration for PushSum

A main difference to Section 3.1.1 is the vector $\vec{t}$, which is only updated at the sending node, but not at the receiving node, *i. e.,* $\vec{t}_a$ is only updated if node a has pushed information to a neighboring node, but not if it has received information. Besides this, the mechanism for PushSum is similar to PushPull. Again, for each node i the main goal is to select neighbors whose current estimate differs strongly from the current local estimate at node i. However, in PushPull, neighboring nodes can be asked "actively" (via *pull-request*) to reply their current estimate to the node that initiates the information exchange. In PushSum, this is not possible, since information exchange is only unidirectional. We highlight the potential complications in the PushSum communication partner selection compared to PushPull using the following representative scenario: Consider the situation that in round t node a pushes its estimate $\lambda_a(t)$=2.00 to node b, and that a and b do not communicate with each other in the next two rounds, and that in round $t + 3$ node b pushes its own estimate $\lambda_b(t+3)$=4.99 to node a. At the end of round $t+3$, $a's$ own estimate has changed to $\lambda_a(t+3)$=5.01, due to communication with other nodes. In the next round, there is a *very low* probability that node a selects node b, since $a's$ local knowledge about $b's$ estimate is 4.99, which is close to $a's$ current estimate of 5.01. On the other hand, there is a *rather high* probability that node b selects node a, since $b's$ local knowledge about $a's$ estimate is 2.00, which strongly differs from $b's$ current estimate 5.01. As a result, it may happen that node b keeps on sending messages to node a in the next round(s), although $\lambda_a(t + 4)$ and $\lambda_b(t + 4)$ are rather similar, because $b's$ information about $a's$ estimate is outdated. The negative effect is that node b pushes its information only to node a but not to other neighboring nodes, which may slow down the diffusion speed. We include two mechanisms to avoid such situations: 1.) We ensure that during early rounds all neighbors receive at least one message, *i. e.,* as long as there are neighboring nodes of node i which have not received a message from i, other neighbors of i which have already received a message will not be selected repeatedly. 2.) Whenever node i has pushed a message to a neighbor, it cannot select the same node as communication partner for the next $\lceil d_i/2 \rceil$ rounds, where d_i is the node degree of node i. In our example from Section 3.1.1 this means that if node a sends a push message to node b in round t, it cannot select node b again before round $t + 3$.

3.1.3 Overhead

Our acceleration entails only a small overhead compared to standard gossip averaging. The most important feature is that no additional communication is necessary. Only local computation and the following local memory space is required: at each node i, two additional vectors $\vec{y}_i$ and $\vec{t}_i$ with an average length of d_{avg} (the average node degree, cf. Section 3.1.1) must be stored. Since our acceleration strategies are designed to improve the diffusion speed of gossip-based averaging algorithms on weakly connected networks, d_{avg} is usually small, *e. g.* $ld(n)$ for hypercubes, 4 (constant) for 2d tori, and ≈ 10 for random geometric graphs.

3.2 Acceleration Using Global Knowledge

Reference algorithm: In order to demonstrate the best possible results that can be achieved with our ACO-based communication partner selection strategy ("*what is possible*"), we have simulated our algorithms based on the assumption that *perfect (global) knowledge* of the current estimates of all neighbors is available at all nodes. This obviously unrealistic scenario provides a bound for the possible improvement by the acceleration strategies presented in this paper. The closer our algorithms get to this bound, the better. Technically, this global knowledge can be simulated by replacing the possibly outdated values in vector $\vec{y}_i$ (cf. Section 3.1.1) with the current estimates of all neighboring nodes. In our simulations, this is possible since our setup allows us to have a global view of the state of the network at any time.

4. EVALUATION

The diffusion speed of gossip algorithms can be measured in terms of message complexity (*i. e.,* number of messages per node) or in terms of time complexity (*i. e.,* number of rounds). We recall that PushPull sends two messages per node per round, instead of one (as in PushSum). As expected, PushPull averaging usually needs fewer rounds than PushSum to converge to a desired approximation error, while PushSum needs fewer messages in total. However, the important aspect of this paper is the speedup achieved with the proposed acceleration strategies, and not the comparison of basic gossip algorithms. All algorithms are implemented in Matlab in a simulation that allows for executing different algorithmic variants of the gossip algorithms and the acceleration strategies, while being able to monitor the state of the network at any time from a bird's-eye-view perspective. All results are given as mean values over 10 simulation runs.

4.1 Experimental Setup

We evaluate PushPull and PushSum schemes, implemented as purely round-based (R) and round- and event-based (RE) variants (cf. Section 2). All algorithm / implementation variants are tested with the acceleration strategies proposed in Sections 3.1, 3.2 using local / global knowledge with roulette-wheel and greedy selection strategies (cf. Section 3.1.1). Moreover, for local knowledge variants we further compare different values of α. In order to allow for a better comparison between PushSum and PushPull variants, we measure the diffusion speed in terms of message complexity (however, the approximation error is calculated after each round). The following abbreviations are used in Figures 1 to 3:

- "*Orig*" denotes the basic version with uniformly random communication partner selection without acceleration, *i.e.,* the *basic gossip variants* such as basic PushPull

- "*loc* $\alpha = \{1, 5, 10\}$" is the ACO-based acceleration strategy using (possibly outdated) *local* knowledge and roulette-wheel selection with the parameter α set to $\{1, 5, 10\}$. The values of α were found empirically.
- "*loc max*" denotes the ACO-based acceleration strategy using *local* knowledge and greedy ("max") selection
- "*glo* $\alpha = 10$" denotes the ACO-based acceleration strategy using (perfect) *global* knowledge and roulette-wheel selection strategy with α set to 10 (reference algorithm # 1)
- "*glo max*" denotes ACO-based acceleration using *global* knowledge and greedy selection (reference algorithm # 2)

Network topologies and sizes. We investigated fully connected graphs, hypercubes, 2D-tori, and random geometric graphs. For all topologies, we considered network sizes $n = 2^k$ with $k \in \{6, 8, 10, 12\}$. This ensures that hypercubes as well as 2D-tori exist for all considered values of n. Random geometric graphs were generated such that the average node degree d_{avg} remains constant also for different network sizes. This can be achieved by computing the transmission radius r for each n based on the *expected* node degree d_{exp} as $r(n) = \sqrt{(d_{exp}/(\pi \times n))}$. Due to the random initialization of locations, d_{avg} varies slightly for each network but is very close to d_{exp}. All presented results are based on $d_{exp} = 10$ (for this setting, d_{avg} is always within $[9.5, 10.2]$). However, we mention the results in the text if other values of d_{exp} and therefore d_{avg} have a significant impact on the results.

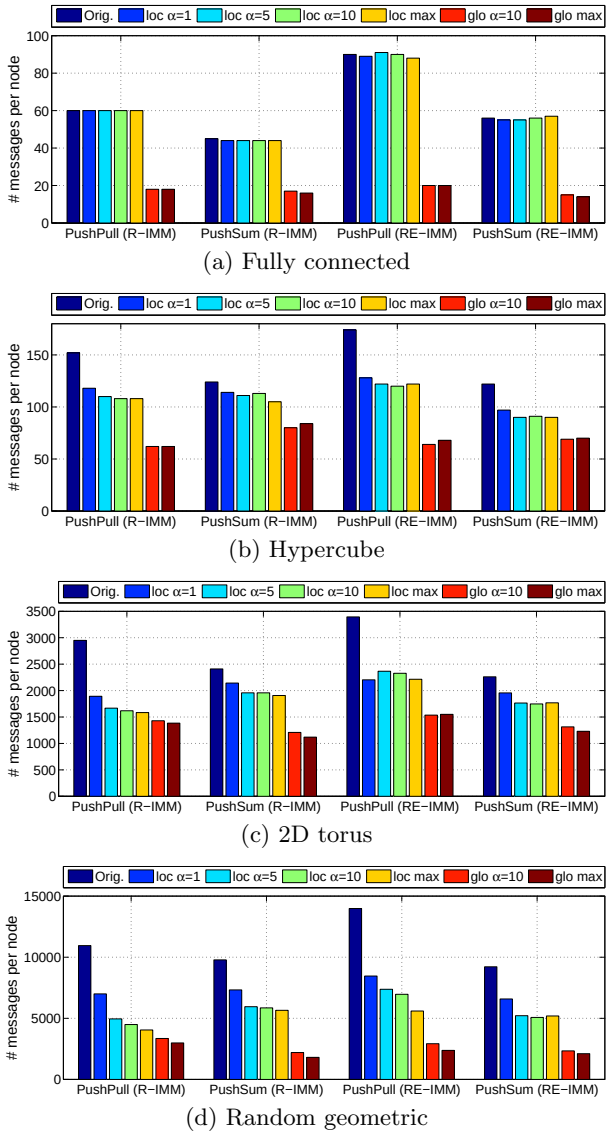
Initialization. Besides random initialization, we use three initialization fields adapted from [6], a linearly varying field, a smooth field modeling temperature, and a single spike field that is zero everywhere except in a sharp spike. Such initialization fields are typically used to simulate values monitored by a sensor network (*e. g.*, pressure or temperature) on planar graphs (typically random geometric graphs). Random geometric and fully connected graphs were evaluated on all four initializations fields, for 2D-torus and hypercubes we restrict the experiments to random initialization.

Termination criterion. We compute the error of the estimated average after each round as $(||\vec{v}(t) - \bar{v}\mathbf{1}||_2)/||\vec{v}(0)||_2$ (same as [6]). Here, $\vec{v}(t)$ is the n-dimensional vector of all estimates after round t and $\vec{v}(0)$ is a vector consisting of the initial estimates. In our experiments, the algorithms are terminated once the erroris less than 10^{-8} (single precision).

4.2 Discussion

Figure 1 shows the message complexity of different acceleration strategies on different network topologies with 1 024 nodes. The plots on the left side show the results for round-based gossip with immediate update (*R-IMM*), the plots on the right side show the results for round-and-event-based gossip with immediate update (*RE-IMM*). Since we observed only very small variations among several runs of the same algorithm, we show colored bars in Figure 1 instead of boxplots for better visualization. The maximum variation between the fastest and the slowest run of the same algorithm was less than 10% of the mean results for fully connected graphs, less than 6% for hypercubes, and less than 3% for 2D torus and random geometric graphs. Figure 2 further summarizes these results and shows the fraction of the number of messages relative to basic gossip algorithms without acceleration. Figures 1 and 2 reveal the following insights:

Influence of network topology. On *fully connected networks*, the proposed ACO-based acceleration strategies can-



(a) Fully connected

(b) Hypercube

(c) 2D torus

(d) Random geometric

Figure 1: Messages per node needed for target accuracy 10^{-8}, network size 1 024 nodes, random init.

not improve the basic gossip algorithms since the number of rounds required for convergence is much lower than the node degree. The algorithms converge before the acceleration strategies are able to influence the communication partner selection, as the information of the estimates of the neighboring nodes cannot spread fast enough for such a large d_{avg}. However, it is important to note that the results are in general not worse than those for the basic gossip algorithms. The results for the reference algorithms with global knowledge show that the number of rounds could be reduced significantly, however, only if all estimates of other nodes are known at all nodes. On *hypercube graphs*, all ACO-based acceleration strategies always improve upon the basic algorithms, which can be accelerated by a factor of up to 1.4. Moreover, the difference to the reference algorithms with global knowledge is reduced. On *2D-torus graphs*, an even higher acceleration factor of almost 2 can be achieved, and the reference algorithms are only slightly faster than the pro-

R-IMM	Fully conn.		Hypercube		2D torus		Rand. geo	
	PP	PS	PP	PS	PP	PS	PP	PS
Orig.	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
loc α=1	1.00	0.99	0.78	0.92	0.64	0.89	0.64	0.75
loc α=5	1.00	0.99	0.72	0.90	0.57	0.81	0.46	0.62
loc α=10	1.00	0.99	0.71	0.91	0.55	0.81	0.41	0.61
loc max	1.00	0.99	0.71	0.85	0.54	0.79	0.37	0.58
glo α=10	0.30	0.38	0.41	0.65	0.48	0.50	0.31	0.22
glo max	0.30	0.36	0.41	0.68	0.47	0.46	0.27	0.18

RE-IMM	Fully conn.		Hypercube		2D torus		Rand. geo	
	PP	PS	PP	PS	PP	PS	PP	PS
Orig.	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
loc α=1	0.99	0.98	0.74	0.80	0.65	0.87	0.61	0.71
loc α=5	1.01	0.98	0.70	0.74	0.70	0.78	0.53	0.57
loc α=10	1.00	1.00	0.69	0.75	0.69	0.77	0.50	0.55
loc max	0.99	1.01	0.70	0.74	0.65	0.78	0.40	0.56
glo α=10	0.22	0.27	0.37	0.57	0.45	0.58	0.21	0.25
glo max	0.22	0.25	0.39	0.57	0.46	0.54	0.17	0.23

PS = PushSum
PP = PushPull

Figure 2: Relative message counts of ACO-based accelerated gossip algorithms in terms of message counts of the original algorithms needed for a target accuracy 10^{-8}, network size 1 024 nodes, random initialization.

posed acceleration strategies. On *random geometric graphs*, an acceleration factor of even up to 2.7 can be observed for some settings, *e. g.* the *loc max* variant of PushPull requires only 37% of the messages of basic PushPull. For random geometric networks with larger d_{avg}, the acceleration is slightly lower. Changing the transmission radius r to the same setting as in [6] (*i. e.,* $r(n) = \sqrt{\log n \,/\, n}$, which leads to $d_{avg} \approx 20$ for $n = 1024$), the results are similar to 2D torus results. We conclude that not only the average node degree significantly influences the amount of acceleration, but also the irregularity of the topology. Our acceleration strategy works best for weakly connected, irregular networks.

Comparison of acceleration strategies. An evaluation of the different acceleration strategies reveals that for the roulette-wheel selection, larger values of α lead to a stronger reduction of the number of messages per node, as compared to small values of α. In most of the cases the greedy strategy *loc max* achieves an even larger reduction of messages than the more conservative roulette-wheel selection. However, we note that for greedy selection it is important to include frequency of iteration if there are two nodes with same maximum difference (cf. Section 3.1.1). Otherwise greedy selection would not be faster than roulette-wheel selection.

Reduction of communication cost. Compared to the basic gossip variants, the reduction of messages observed for PushPull is stronger than for PushSum. This is not surprising since in PushPull the neighboring nodes can be "actively" queried for their current estimate, which is not possible in the PushSum algorithm (cf. the discussion at the beginning of Section 3.1.2). Nevertheless, PushSum can still be accelerated by a factor of up to 1.9.

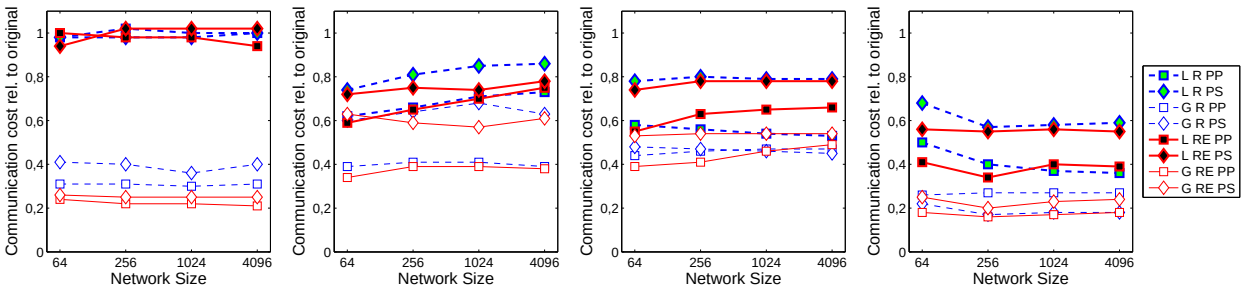
Which is the fastest algorithm? When characterizing the speed of a gossip algorithm by the number of *messages* needed for achieving a certain accuracy, PushSum is the winner when no acceleration is used. It needs fewer messages than PushPull for R-IMM as well as for RE-IMM (see the bars "Orig." in Figure 1). Using the proposed ACO-based acceleration, the round-based variant (R-IMM) of PushPull is significantly faster than PushSum on 2D-torus and random geometric graphs, and comparable to PushSum on hypercube graphs. It is interesting to note that for the same problem settings the reference algorithms with global knowledge achieve the best results with PushSum (see the results for *glo max* in Figure 1 (g)). This indicates that PushSum has an even higher acceleration potential than PushPull, but suffers from the fact that the current estimate cannot be pulled from neighboring nodes (cf. last paragraph). Overall, for RE-IMM accelerated PushSum is the winner, and for R-IMM accelerated PushPull is the fastest algorithm.

Influence of initialization. Initial values may have a significant impact on the diffusion speed of basic gossip algorithm and therefore also on the diffusion speed of the accelerated gossip variants. However, different initialization fields do not influence the impact of the acceleration strategy as compared to the based gossip algorithm, *i. e.,* we observed very similar relative message counts as in Figure 2 also for all non-random initialization fields.

Influence of network size. Figure 3 illustrates that the network size n has almost no influence on the acceleration strategies. For n ranging from 64 to 4 096, the message reduction compared to the basic gossip algorithms is very similar for all topologies, all gossip algorithms, and both update strategies (R-IMM and RE-IMM). Only for hypercube graphs, the reduction tends to decrease with increasing network size, which is due to the increasing node degree of hypercubes with increasing n. For clarity and ease of visibility we show only the curves for *loc max* and *glo max*, the results for the other acceleration variants are very similar.

5. CONCLUSIONS

We have proposed ACO-based acceleration strategies for gossip-based averaging algorithms. The pheromone concept of ACO is adapted such that each node in the network maintains a pheromone deposit for each outgoing link, which influences the probability of selecting a neighboring node as communication partner. Moreover, the (inverse) concept of pheromone evaporation is included in our approach in order to increase the attractiveness of nodes which have not been chosen for a long time. Like the original gossip algorithms, our accelerated versions are based on single-hop communication, where every node only communicates with its direct neighbors without any overlay network. Although the acceleration strategies cause a small overhead in terms of local computation, they significantly reduce the communication cost of standard gossip-based averaging. The message and time complexity can be reduced by factors up to 2.7 for random geometric graphs as compared to standard PushPull averaging. In general, our approach works best for PushPull schemes on weakly connected, non-regular networks. However, our evaluation has revealed that all different variants of PushSum and PushPull can be accelerated significantly on non-fully connected topologies for all tested sizes. An interesting next step is a concrete implementation of the proposed accelerations. We are currently working on concrete distributed implementations which allow not only for testing completely asynchronous and purely event-based variants of our acceleration strategies, but also for evaluating the influence of network congestion and node/message failures.



(a) Fully connected (b) Hypercube (c) 2D-torus (d) Random Geometric

Figure 3: Influence of network size. For each topology, the reduction as computed in Figure 2 is plotted for different network sizes. Legend: 'L' = loc max, 'G' = glo max, 'R' = R-IMM, 'RE' = RE-IMM.

6. REFERENCES

[1] F. Benezit, A. Dimakis, P. Thiran, and M. Vetterli. Order-Optimal Consensus Through Randomized Path Averaging. *IEEE T. Inf. Theory*, 56:5150–5167, 2010.

[2] C. Blum and X. Li. Swarm intelligence in optimization. In *Swarm Intelligence*, pages 43–85. Springer, 2008.

[3] E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm Intelligence: From Natural to Artificial Systems.* Oxford Univ. Press, 1999.

[4] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah. Randomized Gossip Algorithms. *IEEE T. Inform. Theory*, 52:2508–2530, 2006.

[5] F. F. De Vega and E. Cantú-Paz. *Parallel and Distributed Computational Intelligence*, volume 269. Springer, 2010.

[6] A. Dimakis, A. Sarwate, and M. Wainwright. Geographic gossip: Efficient averaging for sensor networks. *IEEE T. Signal Proces.*, 56:1205–1216, 2008.

[7] M. Dorigo and L. M. Gambardella. Ant colony system: a cooperative learning approach to the traveling salesman problem. *Evolutionary Computation, IEEE Transactions on*, 1(1):53–66, 1997.

[8] M. Dorigo, V. Maniezzo, and A. Colorni. Ant system: optimization by a colony of cooperating agents. *IEEE T. Syst. Man Cy. B*, 26:29–41, 1996.

[9] M. Dorigo and T. Stuetzle. Ant colony optimization: Overview & recent advances. In *Handbook of Metaheuristics*, volume 146, pages 227–263. Springer, 2010.

[10] F. Ducatelle, G. Di Caro, and L. Gambardella. Principles and applications of swarm intelligence for adaptive routing in telecommunications networks. *Swarm Intelligence*, 4:173–198, 2010.

[11] D. E. Goldberg and K. Deb. A comparative analysis of selection schemes used in genetic algorithms. In *Foundations of Genetic Algorithms*, pages 69–93. Morgan Kaufmann, 1991.

[12] G. H. Golub and R. S. Varga. Chebyshev semi-iterative methods, successive overrelaxation iterative methods, and second order richardson iterative methods. *Numer. Math.*, 3:157–168, 1961.

[13] C. Guéret, N. Monmarché, and M. Slimane. Autonomous gossiping of information in a P2P network with artificial ants. In *Proc. of Ant Colony Optimization & SI*, pages 388–395. Springer, 2006.

[14] C. Guéret, N. Monmarché, and M. Slimane. Self-organizing ant-based information gossiping algorithm for P2P networks. In *10th Int. Conf. on Innovative Internet Community Services*, pages 450–461, 2010.

[15] M. Jelasity. Gossip. In *Self-organising Software*, pages 139–162. Springer, 2011.

[16] P. Jesus, C. Baquero, and P. S. Almeida. Dependability in aggregation by averaging. *CoRR*, abs/1011.6596, 2010.

[17] S. Kar and J. M. F. Moura. Sensor Networks With Random Links: Topology Design for Distributed Consensus. *IEEE T. Sig. Proces.*, 56:3315–3326, 2008.

[18] R. Karp, C. Schindelhauer, S. Shenker, and B. Vocking. Randomized rumor spreading. In *Proc. 41st Annual Symposium on Foundations of Computer Science*, pages 565–574. IEEE Computer Society, 2000.

[19] D. Kempe, A. Dobra, and J. Gehrke. Gossip-based computation of aggregate information. In *Proc. 44th Symposium on Foundations of Computer Science*, pages 482–491. IEEE, 2003.

[20] J. Kennedy and R. C. Eberhart. *Swarm Intelligence.* Morgan Kaufmann, 2001.

[21] A. Montresor. Intelligent gossip. In *Intelligent Distributed Computing, Systems and Applications*, volume 162 of *Studies in Computational Intelligence*, pages 3–10. Springer, 2008.

[22] A. Montresor. Designing extreme distributed systems: challenges and opportunities. In *Proc. 8th ACM SIGSOFT Conf.*, pages 1–2. ACM, 2012.

[23] S. Muthukrishnan, B. Ghosh, and M. H. Schultz. First- and second-order diffusive methods for rapid, coarse, distributed load balancing. *Theor. Comput. Syst.*, 31:331–354, 1998.

[24] E. Ridge and E. Curry. A roadmap of nature-inspired systems research and development. *Multiagent Grid Syst.*, 3:3–8, 2007.

[25] E. Ridge, D. Kudenko, D. Kazakov, and E. Curry. Moving nature-inspired algorithms to parallel, asynchronous and decentralised environments. In *Proc. 2005 Conf. on Self-Organization and Autonomic Informatics*, pages 35–49. IOS Press, 2005.

[26] C.-C. Shen and S. Rajagopalan. Protocol-independent multicast packet delivery improvement service for mobile ad hoc networks. *Ad Hoc Netw.*, 5:210–227, 2007.