Distributed Computing Column 57
Distributed Algorithms as Combinatorial Structures

Jennifer L. Welch
Department of Computer Science and Engineering
Texas A&M University, College Station, TX 77843-3112, USA
welch@cse.tamu.edu

This issue is devoted to an insightful article by Keren Censor-Hillel, explaining the relationships between distributed algorithms for a variety of problems, including synchronization and gossip, and corresponding combinatorial structures, such as sparse spanners. This connection can help prove both upper and lower bounds for distributed computing problems. The article features a lucid exposition, discussion of open problems, and extensive references for further exploration.

Many thanks to Keren for her contributions!

Call for contributions: I welcome suggestions for material to include in this column, including news, reviews, open problems, tutorials and surveys, either exposing the community to new and interesting topics, or providing new insight on well-studied topics by organizing them in new ways.

Distributed Algorithms as Combinatorial Structures

Keren Censor-Hillel
Department of Computer Science, Technion
Haifa, Israel
ckeren@cs.technion.ac.il

Abstract

For many distributed computing problems there exists a characterizing combinatorial structure. In other words, the existence of an algorithm for solving the problem on any graph G in time $T(G)$ implies the existence of the structure in G with some parameter related to $T(G)$, and vice versa. Such relations go back to classic examples, such as synchronizers and sparse spanners, and continue to emerge in recent studies of gossip algorithms and multicast algorithms in different communication settings. In this article, we give an overview of both old and recent results that illustrate these connections. We discuss how finding distributed algorithms as well as proving lower bounds can be reduced to studying combinatorial graph structures.

"An algorithm must be seen to be believed"
– Donald Knuth, The Art of Computer Programming[1]

1 Introduction

Three decades ago, the 1985 JACM paper by Awerbuch [1] proposed synchronizers as a method for running synchronous algorithms in an asynchronous environment, and gave the first constructions of synchronizers. The power of Awerbuch's abstraction lies in the fact that nothing about the synchronous algorithm needs to be known, not even the problem that it solves. A synchronizer is a simulation that simply takes as input any synchronous algorithm as a black box, and produces an asynchronous algorithm that provides the same output, with some overhead in the number of messages and the time it requires. The goal of synchronizers is to minimize these two measures. In order to obtain efficient synchronizers, Peleg and Ullman [41] introduced the notion of a spanner, which is a sparse subgraph that preserves distances up to some factor given as a parameter. They

[1]The complete quote ends with: "... and the best way to learn about what an algorithm is all about is to try it." Here we suggest the different viewpoint of visualizing an algorithm as a graph structure.

showed that a good spanner gives an efficient synchronizer. Moreover, they showed the reverse connection – that an efficient synchronizer induces a good spanner.

Such reciprocative relations between algorithms and combinatorial structures emerge in numerous additional examples in distributed computing, where the existence of an algorithm for solving a problem P on any graph G in time $T(G)$ implies the existence of a certain structure S_P in G with some parameters related to $T(G)$, and vice versa. The goal of this article is to survey several such connections in the context of this common thread. Some of the examples, such as synchronizers and spanners, are basic fundamental results that are taught today in standard distributed computing courses. Others are recent results from the past couple of years. The author finds it important to take a step back once in a while and try to identify the higher-level concepts that underly our field.

2 Synchronizers and Spanners

The simplest way to simulate a synchronous algorithm in an asynchronous system is to simply attach a round number to every message sent[2]. That is, if in round r of the synchronous algorithm A, node v sends message m to its neighbor u, then in the synchronizer S node v sends to u the message (r, m). To retain the correctness of A, node v has to wait until it receives all messages sent to it by its neighbors in round r before it can send out a message of round $r + 1$. But the synchronizer S is running in an asynchronous system, so how would v know whether it still has to wait for a message (r, m) from u or whether u has no message to send to v for round r? This is crucial in order to avoid waiting indefinitely. One way to solve this is to have u send an empty message $(r, \perp)$ in case it does not have any content to send for round r. The drawback of such a solution is that it requires communicating over all edges for simulating every single round of A, which can be much more than needed by A.

To formally capture the task and the measures of complexity of interest, we define a synchronizer S as an asynchronous algorithm that receives as input a synchronous algorithm A and its inputs, and produces the same output as A. The *message overhead* of S is the worst-case value of the ratio between the number of messages it sends per round and the number of messages sent by an input algorithm. Another measure of interest is the *time overhead*, which is defined similarly, except that instead of messages we count time, where one time unit is measured by the delivery of a message. For the simple synchronizer described above, the message overhead is $O(|E|)$ and the time overhead is $O(1)$.

One way to avoid the need for sending empty messages is to send an acknowledgement for every received message. In more detail, let every node v that receives a message (r, m) from its neighbor u send back to node u a message (r, ack). Once node u receives acknowledgements from each neighbor to which u has sent a message (r, m), then u knows that all its messages for round r have been received, and therefore it considers itself *safe* and sends out to all its neighbors a message (r, safe). Now, a neighbor v of u that does not receive (r, m) from u, but does receive an (r, safe) message from u, knows that it should not wait for a message (r, m) from u. If this holds for all neighbors of v then it can go ahead and send out its $(r + 1, m)$ messages. Clearly, sending safety messages as described above does not overcome the need to send information on every link in the graph. However, this idea can be used along with additional mechanisms to reduce the message overhead, as follows.

[2]We emphasize that we assume no faults in this setting.

Awerbuch [1] presented synchronizers that are based on spanning trees. Here, we will show the relation to *spanners*, as introduced by Peleg and Ullman [41]. A k-spanner is a subgraph $S = (V, E')$ of G, where $E' \subseteq E$ and for every $u, v, \in V$ it holds that $d_S(u, v) \leq k d_G(u, v)$. That is, the distance in S between any pair of nodes is at most k times their distance in G. The parameter k is called the *stretch* of the spanner.

As before, assume that we are given a k-spanner S of G. For each round r, let each node v send out (r, m) messages to its neighbors, and consider itself safe after receiving all (r, ack) messages from its neighbors. However, instead of sending (r, safe) messages to all of its neighbors, v now sends (r, safe) messages to its neighbors *within the spanner* S, and waits to receive (r, safe) messages from them. After repeating this exchange of safety messages for k iterations, v can move on to sending $(r+1, m)$ messages. The reason that this works is because a simple induction proves that at the end of iteration i, for $1 \leq i \leq k$, all nodes within distance i of v in S are indeed safe. Since any neighbor of v in G is within distance at most k from it in S, it holds that after k iterations, all neighbors of v in G are indeed safe. The time overhead of this spanner-based synchronizer depends on the stretch and is $O(k)$, while its message overhead depends also on the sparsity of the spanner and is $O(kM)$, where M is the number of edges in the spanner. In this sense, the existence of a sparse spanner with a small stretch implies the existence of a time- and message-efficient synchronizer, and the following theorem is proven.

Theorem 2.1. *[41] A k-spanner with M edges implies the existence of a synchronizer with a time overhead of $O(k)$ and a message overhead of $O(kM)$.*

As promised, this relationship goes both ways. We now show that the existence of an efficient synchronizer implies a sparse spanner with a small stretch.

Theorem 2.2. *[41] A synchronizer with a time overhead of T and message overhead of M implies the existence of a T-spanner with M edges.*

Proof. Assume there is a synchronizer which has a message overhead of M and a time overhead of T for simulating a single round of a synchronous algorithm. Let $S \subseteq E$ be the set of edges of the graph over which messages are sent in the synchronizer. Since the message overhead is M and messages are only sent on edges of S by definition, we have that the size of S is at most M. Further, since at least one bit of information must pass between each pair of neighbors of G when simulating a single round of a synchronous algorithm in order for them to know whether they should wait for a message from each other, we know that S contains a path between every pair of neighbors in G. This path has to be of length at most T since this is the bound on the time it takes for information to travel between every pair of nodes. Hence, S is a T-spanner with M edges. □

Since there are known lower bounds that relate the sparsity and the stretch of spanners, we immediately get lower bounds for the trade-off between time overhead and message overhead of synchronizers.

The power of relating distributed graph problems and combinatorial graph structures lies exactly in this duality, which appears in many additional cases, as exemplified in the following sections.

3 The Gossip Model and Spanners

In this section we discuss another distributed problem that corresponds to spanners. The model of computation addressed in the discussion about synchronizers is called the *Local* model (see [39]).

The Local model is a basic model for distributed computing, which focuses on the fundamental property of locality by abstracting away other characteristics, such as failures or bounded messages. Therefore, the model asks what is the number of rounds required for a computation, or, alternatively, what is the distance that information has to travel in order to solve a task in question. Being such a crucial model for understanding distributed computing, the Local model was widely studied by our community. Additional models are also important and bring to light other inherent properties of distributed systems, and one of them, the *Gossip* model, is the one we address next.

The Gossip model is a synchronous model, in which the following constraint is imposed over the communication. In each round, every node may initiate contact with at most one of its neighbors, and may communicate only with that neighbor and any other neighbor that initiated contacted with it. At best, at most n edges can be communicated over per round.

The Gossip model was introduced by Demers et al. [11], for studying replicated data bases, and has been afterwards used for many additional applications. The Gossip model is clearly not easier than the Local model, in the sense that any algorithm running in the Gossip model in T rounds can also run in the Local model in T rounds. For the other direction, it is easy to see that within Δ rounds, where Δ is the maximum degree in the communication graph G, one can simulate a single Local round, by contacting neighbors in round-robin. Therefore, any algorithm running in T rounds in the Local model can be executed in ΔT rounds in the Gossip model. Since Δ can be as large as n, this is a very expensive simulation, raising the question of how hard really is the Gossip model, compared to the Local model.

In many similar scenarios, a successful approach is to use randomization. Consider the context of *information spreading*, a problem in which each node has an initial input that needs to reach all other nodes. Karp et al. [30] studied a related *random phone call* model which corresponds to a complete graph. Their work has received much additional attention, such as in bounding the number of calls [12], bounding the number of random bits used [26, 25], and bounding the total number of bits [20].

For general graphs, choosing a neighbor uniformly at random in each round may not perform much better than the round-robin deterministic solution. A simple example is the barbell graph, consisting of two cliques connected by a single edge, in which for each of the two central nodes, the probability of contacting the other is $\Theta(1/n)$, which implies at least a linear number of rounds in expectation just in order to simulate one Local round. Indeed, the connectivity of the graph has a great impact on the efficiency of the uniform random algorithm. Many papers analyzed the uniform random algorithm with respect to some notion of connectivity[10, 37, 23, 44, 24], and additional papers studied other algorithms for information spreading [9, 15, 16, 18] and additional problems [31, 32], as well as different families of graphs [13, 14, 19] or with respect to different graph parameters [17].

Peeking back at the Local model, one can see that solving such a problem clearly requires exactly D rounds. If we could simulate a single round of the Local model, a problem referred to as *neighbor exchange*, faster than the trivial Δ-round solution, we could solve information spreading faster than ΔD rounds. Notice that, similarly to simulating a single round by a synchronizer, the neighbor exchange problem requires each node to obtain information from all of its neighbors, perhaps indirectly. The obstacle here, as before, is the possibility of many edges.

To study the relation between spanners and information spreading algorithms in the Gossip model deeper, we follow [8]. Assume that we are given a k-spanner S of G, with maximal degree Δ_S. If each node contacts all of its neighbors in S in round-robin for Δ_S rounds then it would

manage to collect the information from all of its neighbors in S. If the nodes repeat the above for k iterations then each node collects information from any node within distance at most k in S. Since S is a k-spanner of G, this implies that it collects information from all of its neighbors in G. This gives a solution to the neighbor exchange problem in $\Delta_S k$ rounds, and a solution to the information spreading problem in $\Delta_S kD$ rounds. More generally, we can simulate any T-round Local algorithm in $\Delta_S kT$ rounds.

The above result can be improved if we consider additive spanners: An (α, β)-spanner S of G is a subgraph for which for every two nodes u, v, the distance $d_S(u, v)$ is at most $\alpha d_G(u, v) + \beta$. The following statement reduces to the previous expression when $\alpha = k$ and $\beta = 0$, and can be shown for larger β with some additional arguments.

Theorem 3.1. *[8] The simulation of a T-round Local algorithm given an (α, β)-spanner S can be done in $O(\Delta_S \alpha T + \Delta_S \beta)$ rounds.*

While the above can be fast when Δ_S is small, it is important to notice that not every graph has a spanner with small degrees. The simplest example is a star, where the degree of the center is linear in n. As the star is a tree, its only spanner is the graph itself. However, any algorithm terminates in 2 rounds on the star graph, as all leaves always contact the center, sending their information to it in the first round, and collecting all information in the second round. So, the intuition here is that we do not need to care about the maximal degree, but rather we need to direct the edges of the graph such that the maximal *outgoing* degree is small.

Indeed, it is possible to direct the edges of a graph with *hereditary density* δ such that the maximal out degree would be $O(\delta)$. This can be done in the Gossip model within $O(\delta \log n)$ rounds. Formally, the hereditary density of a graph is the minimal integer δ such that the subgraph induced by any subset of nodes $U \subseteq V$ has at most $\delta|U|$ edges. Intuitively, it captures sparsity in a more subtle way than the maximal degree. Thus, it is possible to show the following theorem, which generalizes the usage of an (α, β)-spanner.

Theorem 3.2. *[8] Given an (α, β)-spanner S with hereditary density δ_S, simulating a T-round Local algorithm can be done in $O(\delta_S \log n + \delta_S \alpha T + \delta_S \beta)$ rounds.*

The reason that this is a significant improvement, is that it is possible to show that every graph has a sparse spanner with small hereditary density, as opposed to small maximal degree. For example, using the framework of Pettie [42], one can obtain a constant-stretch spanner with hereditary density $O(1)$ for any graph. Moreover, Pettie gives a construction of such a spanner in the Local model within $O(\text{poly} \log n)$ rounds. The work in [8] shows that this construction can be simulated in the Gossip model within $O(\text{poly} \log n)$ rounds, to produce an $(O(1), \text{poly} \log n)$-spanner with hereditary density $O(1)$. This gives that after the preprocessing of constructing the spanner in $O(\text{poly} \log n)$ rounds and directing its edges in $O(\log n)$ rounds (as δ_S here is $O(1)$), one can simulate a T-round Local algorithm in $O(T + \text{poly} \log n)$ rounds (as $\alpha = O(1)$ and $\beta = O(\text{poly} \log n)$ in S). In other words, since the preprocessing takes only $\text{poly} \log n$ rounds, this gives that any T-round Local algorithm can be simulated in the Gossip model in $O(T + \text{poly} \log n)$ rounds, which is only an additive $\text{poly} \log n$ overhead to the Local model.

To conclude this discussion, we discuss the other direction of the relation between spanners and information spreading algorithms in the Gossip model. We notice that any algorithm that solves the neighbor exchange problem in the Gossip model in T rounds induces a T-spanner with hereditary density $O(T)$, obtained by considering the edges that the algorithm uses. This allows us to derive lower bounds for the neighbor exchange problem, as follows.

Theorem 3.3. *[8] There exist graphs for which simulating a single Local round requires* $\Omega(\log n/\log\log n)$ *rounds.*

Proof. It is known [40] that there are graphs with girth (length of the shortest cycle) g and at least $\frac{1}{4}n^{1+1/g}$ edges. In such a graph any $(g-2)$-spanner has to be the original graph itself, since removing a single edge causes the end-points of the shortest cycle to have distance at least $g-1$. Thus, the only spanner, identical to the original graph, has at least $\frac{1}{4}n^{1+1/g}$ edges and density at least $\frac{1}{4}n^{1/g}$. Therefore, for such graphs, the number of rounds $T(n)$ required for solving neighbor exchange is at least $\max_g\{\min(g-2, \frac{1}{4}n^{1/g})\} = \Omega(\log n/\log\log n)$. □

4 Congestion Models and Tree Packings

We now leave the domain of spanners and address other graph structures that represent distributed graph algorithms, but we focus again on the problem of information spreading. We study two different communication models. The setting we consider in this section is when there is some limited capacity to communication per round. The first model we consider is the traditional *Congest* model [39], where there is a bandwidth of b bits that can be sent on any edge in a single round, where typically $b = O(\log n)$. The second model we will consider is the *Vertex-Congest* model. This model is similar in spirit to the Congest model, but here the congestion is imposed on the nodes rather than the communication links: in each round a node sends the same b-bit message to all of its neighbors. One way to view the Vertex-Congest model is as a model that abstracts a wireless network operating on top of a MAC layer that handles collisions.

We assume that the bandwidth b is large enough to carry any single piece of information that needs to reach the nodes of the network, and ask what is the throughput that we can obtain in such models. In other words, we ask how many messages can complete their spreading in each round.

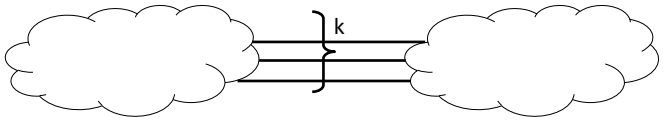

Figure 1: Toy example: Can we obtain a throughput of k in a k-connected graph?

Let us begin by considering the following toy example, depicted in Figure 1. Assume the graph has a bottleneck, that is, the size of its smallest cut is $k = 1$. Clearly, the throughput in this graph cannot be more than 1, as only a single piece of information can cross the cut in any given round. It is also easy to show [45] that N messages can complete in $O(D + N)$, where D is the diameter of the graph. But what if the graph has better connectivity? Can we hope for a throughput of k in a k-connected graph?

4.1 The Congest model and spanning tree packings

We first address the Congest model, and define formally what we mean by k-connected graphs. A k-edge connected (or k-connected) graph is a graph which does not get disconnected despite the removal of any $k-1$ edges. By Menger's Theorem [36] we know that in such a graph, any two nodes u, v are connected by at least k edge-disjoint paths. However, this alone is insufficient for our needs, as the paths between different pairs of nodes may intersect. What does help here is the notion of *tree packings*.

Suppose S is a spanning tree of a graph G. Sending messages over the edges of S guarantees that they reach all nodes of the graph. If there are two spanning trees for G that are edge-disjoint, then we could route two separate streams of messages over their edges in parallel, obtaining a throughput of 2. This generalizes easily, so that if P is a set of edge-disjoint spanning trees of G of size s, then one can obtain a throughput of s. The set P is called a *spanning tree packing*. In fact, for our purposes, one can relax the disjointness requirement, and talk about a *fractional spanning tree packing*. In a fractional tree packing P, a weight w_S is given to each tree $S \in P$, and the requirement is that the total weight of each edge does not exceed 1. That is, it is required that for every $e \in E$, $\sum_{\{S \in P \text{ s.t. } e \in S\}} w_S \leq 1$. A tree packing of disjoint trees is simply an assignment of weight 1 to all trees. The *size* of the packing P is now not the number of trees in it, but rather their total weight $\sum_{S \in P} w_S$.

Figure 2 shows an example of a spanning tree packing that consists of three spanning trees. Giving each tree a weight of $1/2$ results in a packing of size $3/2$. Any non-fractional packing of this graph has size at most 1, because any spanning tree must use three edges.

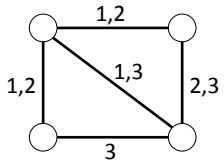

Figure 2: An example of a spanning tree packing of size $3/2$ with three trees $\{1, 2, 3\}$ of weight $1/2$ each. Each edge is labelled with the spanning trees to which it belongs.

Given a fractional tree packing P of size s one can obtain a throughput of s, by having each edge share time between the trees in P that it is part of, proportional to their weight.

The above implies that our goal would be to find tree packings as large as possible. A simple upper bound for the size of any tree packing in a k-edge connected graph is k, as every spanning tree in the packing must cross the min-cut.

Moreover, it is known by classical results of Kundu, Nash-Williams, and Tutte, that any k-edge connected graph has a spanning tree packing of size $\lceil (k-1)/2 \rceil$, and that this is tight [46, 33, 38]. This gives that we can achieve a throughput of $\lceil (k-1)/2 \rceil$ in any k-edge connected graph. Centralized algorithms for finding such tree packings have been given in [21, 2], and a distributed algorithm that finds a packing of size $\lceil (k-1)/2 \rceil (1-\epsilon)$ was given in [5].

As promised, the relation between algorithms for spreading information in the Congest model and spanning tree packings goes both ways. Suppose that an algorithm with throughput s is given. By definition, the edges over which any single message is sent induce a spanning subgraph, which,

by removing cycles, can be made into a spanning tree. With some additional details, it is possible to show that these trees induce a fractional spanning tree packing of size s.

4.2 The Vertex-Congest model and dominating tree packings

The second model of computation we address in this section is the Vertex-Congest model. Recall that in this model the same b bits of information are sent to all the neighbors of a node in any single round.

We are again interested in the achievable throughput of disseminating information throughout a network whose underlying graph has a better connectivity than the toy example of Figure 1. Edge-connectivity and spanning tree packings are insufficient notions for this model, because a node cannot exploit its multiple outgoing edges as it has to send the same message to all of its neighbors. Instead, we address graphs that are k-vertex connected, which are graphs in which the deletion of any $k-1$ vertices does not disconnect it. As in the case of edge-connectivity, Menger's Theorem [36] tells us that every two nodes in a k-vertex connected graph are connected by at least k vertex-disjoint paths. We need a structure that we can use in a similar way to the use of spanning tree packings in the Congest model, which will allow us to send information in parallel between different pairs of nodes, whose k vertex-disjoint paths may intersect.

The analogue structure that is useful here is a *dominating tree packing*. A dominating tree S is a tree connecting a set of internal nodes $V_S \subseteq V$ which dominate the graph. That is, V_S is a connected set of nodes such that every node in V is either in V_S or is connected to a node in V_S. Domination guarantees that if all nodes of a dominating tree send a message then it reaches all the nodes of the graph. In order to achieve a larger throughput, we can send messages in parallel over several vertex-disjoint dominating trees.

To formalize the above, we define a fractional dominating tree packing P as a set of dominating trees, where each tree $S \in P$ has a weight w_S, such that the total weight of each node is at most 1, i.e., for every $v \in V$ it holds that $\sum_{\{S \in P \text{ s.t. } v \in V_S\}} w_S \leq 1$. The size of the packing is the sum of all weights. When all weights are equal to 1 this gives a set of vertex-disjoint dominating trees, whose size is the number of trees. The main observation is that here again a dominating tree packing of size s provides a backbone for sending information over the network with a throughput of s, by having each node share its time between the dominating trees it belongs to, proportional to their weight. Hence, in order to obtain high throughput, we are interested in finding large dominating tree packings.

For a k-vertex connected graph, one cannot hope for a dominating tree packing of size more than k, and, in fact, it was shown [6] that there exist graphs for which the largest possible packing has size of at most $O(k/\log n)$. Unlike the edge-connectivity case, this implies that the gap between the connectivity and the possible size of a tree packing is at least logarithmic.

It turns out that the above is not only a bound on the size of a dominating tree packing but can actually be achieved. It has been shown [6] that dominating tree packings of size $\Omega(k/\log n)$ exist in k-vertex connected graphs, and efficient distributed and centralized algorithms for obtaining them were shown in [5]. The above also shows how to obtain an $O(\log n)$-approximation of the vertex-connectivity of the graph. These results require techniques that are entirely different from the classical tree packing results for graphs based on their edge-connectivity, due to inherent differences between minimum edge cuts and minimum vertex cuts.

Finally, as in previous examples in this article, the existence of an algorithm for disseminating information with throughput s implies the existence of a dominating tree packing with size s. This

follows, with some technical arguments, from the observation that the nodes which send a particular message must form a connected set and they also form a dominating set since the message reaches all nodes. Therefore, we have the following.

Theorem 4.1. *[6] There is an algorithm for disseminating information with throughput $\Theta(s)$ in G if and only if there is a dominating tree packing with size $\Theta(s)$ in G.*

However, it is worth mentioning that this does not imply that lower bounds on the size of dominating tree packings induce lower bounds on the throughput of algorithms for disseminating information, but rather the above argument only holds for the case of so-called store-and-forward algorithms, which do not manipulate the messages they send but only forward them as is. It is known (following the analysis of [27]) that using network coding, it is possible to obtain a throughput of k in a k-vertex connected graph. This, however, has the limitation that the coefficients needed for coding the messages are too large for the vertex-congest model, implying that this important question of the optimal throughput is still open.

5 Coloring and Acyclic Orientations

We now leave the task of disseminating information, and conclude with a classic and simple example. The last example we give is of coloring. A k-coloring is an assignment of colors in $\{c_1, \ldots, c_k\}$ to the nodes of the graph such that no two neighbors share the same color. In a distributed setting, we require each node to eventually output a color such that the above holds. We assume the Local model of communication, in which the nodes can send messages of arbitrary size to their neighbors in each round.

We discuss here the simple connection between colorings and *acyclic orientations*, and we follow the presentation in [4]. This connection was used in coloring algorithms by orienting edges, such as the algorithm of Barenboim and Elkin [3].

An acyclic orientation is an orientation of the edges of the graph such that no directed cycle is formed. The *length* of an acyclic orientation is the length of the longest directed path in it. The following is known as the Gallai-Hasse-Roy-Vitaver Thoerem, proven independently by several authors in the 1960's.

Theorem 5.1. *[22, 43, 29, 47] Given an acyclic orientation of G of length k, there is a $(k+1)$-coloring algorithm requiring $k+1$ rounds.*

Proof. Assume we are given an acyclic orientation of length k. Then, one can find a $(k+1)$-coloring in $k+1$ rounds as follows: In round i, every node for which all of its parents are already colored, chooses the color c_i and sends it to its neighbors. By a simple induction, it is easy to see that this gives a $(k+1)$-coloring. □

This can be improved if we know that the in-degree of the acyclic orientation is at most some value d. We simply change the color that is chosen to be the smallest one that is not used by the node's parents. That is, in round i, every node v for which all of its parents, denoted by the set P_v, are already colored, chooses the color c_j, where $j = \min\{i \mid c_i \text{ is not chosen by } P_v\}$ and sends it to its neighbors. This gives a $(d+1)$-coloring.

It is also easy to obtain an acyclic orientation of length k from a $(k+1)$-coloring, by simply directing each edge toward its endpoint with the larger color among the two [4], giving the other direction of the relation between distributed graph coloring and acyclic orientations.

6 Discussion

This article attempts to draw attention to a horizontal theme that is common to many research directions in distributed computing. However, it by no means provides any complete survey of research in which distributed algorithms correspond to combinatorial graph structures. The author is aware of additional examples and is always happy to learn about and discuss even more of those.

It is also the case that distributed algorithms correspond to combinatorial structures in a much wider sense. There are many examples where an algorithm for a given graph G corresponds to a structure in some related graph H_G, rather than in G itself.

Perhaps the simplest example is that of the relation between coloring and finding a maximal independent set (MIS). It is easy to see that a k-coloring gives an MIS algorithm that completes in k rounds, by having iterations (starting with the graph G itself) where all remaining nodes of color i enter the MIS and are then removed along with all of their neighbors from the processed graph. It is also known [35] that an algorithm for finding an MIS in $T(n, \Delta)$ rounds gives an algorithm for finding a $(\Delta + 1)$-coloring in $T(n(\Delta + 1), 2\Delta)$ rounds, but here the coloring algorithm uses the MIS algorithm on a different graph H_G that depends on G. In this sense this is not very different from the standard notion of a reduction between tasks.

Another example arises in Linial's $O(\log^* n)$ lower bound for MIS or 3-coloring in rings [34]. At a very high level, the proof shows that a distributed algorithm for coloring an n-node ring in T rounds requires the existence of a 3-coloring of a different graph, denoted by $B_{T,n}$. In this graph, each node corresponds to every possible local view of every node after T rounds, and two nodes are connected by an edge if their local views are consistent, that is, they correspond to some ring in which they are neighbors. The reason that a 3-coloring algorithm running in T rounds on a ring implies a 3-coloring on $B_{T,n}$ is that every node has to choose its output based only on its T-neighborhood, and the color should be different from the color chosen by any possible view its neighbors have after T rounds.

The above examples are slightly different than the ones given in the previous sections, in the sense that they address structures in graphs that differ from the input graph. Nevertheless, they of course shed light on the distributed tasks at hand.

Intriguing questions that arise in the context of this article are the following. The first is, when does a distributed computing problem correspond to a graph structure that is natural to describe? Does it always occur when considering simulating one network model in another? Can this be formalized in some way?

Regarding the specific examples covered in this article, one important question that remains open concerns the Gossip model. It has been shown in the algorithm of [7] and in the much-improved round complexity algorithm of [28] that neighbor exchange can be solved in a robust manner, that is, with tolerance to faults. This gives $O(T \log^2 n)$ rounds for simulating a T-round Local algorithm in a robust way. However, using the spanners mentioned in Section 3 is highly non-robust, as they are static structures. It remains open whether one can obtain both the $O(T + \text{poly} \log n)$ complexity for the simulation as well as robustness.

In fact, robustness becomes an issue in many algorithms that rely on static graph structures, and obtaining robust algorithms is an important goal for many of the discussed problems.

Acknowledgements: This article is based on invited talks the author gave at the 3rd Workshop on Advances in Distributed Graph Algorithms (ADGA), 2014, and at Yahoo! Research Labs, Haifa.

The author thanks the audience of both talks for lively and intriguing discussions, and Idit Keidar for interesting discussions about this topic.

References

[1] Baruch Awerbuch. Complexity of network synchronization. *Journal of the ACM*, 32(4):804–823, 1985.

[2] Francisco Barahona. Packing spanning trees. *Mathematics of Operations Research*, 20(1):104–115, 1995.

[3] Leonid Barenboim and Michael Elkin. Sublogarithmic distributed MIS algorithm for sparse graphs using Nash-Williams decomposition. In *Proceedings of the 27th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 25–34, 2008.

[4] Leonid Barenboim and Michael Elkin. *Distributed Graph Coloring: Fundamentals and Recent Developments*. Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool Publishers, 2013.

[5] Keren Censor-Hillel, Mohsen Ghaffari, and Fabian Kuhn. Distributed connectivity decomposition. In *Proceedings of the 33rd Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 156–165, 2014.

[6] Keren Censor-Hillel, Mohsen Ghaffari, and Fabian Kuhn. A new perspective on vertex connectivity. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 546–561, 2014.

[7] Keren Censor-Hillel and George Giakkoupis. Fast and robust information spreading. *Unpublished manuscript.*

[8] Keren Censor-Hillel, Bernhard Haeupler, Jonathan A. Kelner, and Petar Maymounkov. Global computation in a poorly connected world: fast rumor spreading with no dependence on conductance. In *Proceedings of the 44th Symposium on Theory of Computing Conference (STOC)*, pages 961–970, 2012.

[9] Keren Censor-Hillel and Hadas Shachnai. Fast information spreading in graphs with large weak conductance. In *Proceedings of the 22nd ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 440–448, 2011.

[10] Flavio Chierichetti, Silvio Lattanzi, and Alessandro Panconesi. Almost tight bounds for rumour spreading with conductance. In *Proceedings of the 42nd ACM Symposium on Theory of Computing (STOC)*, pages 399–408, 2010.

[11] Alan Demers, Dan Greene, Carl Hauser, Wes Irish, John Larson, Scott Shenker, Howard Sturgis, Dan Swinehart, and Doug Terry. Epidemic algorithms for replicated database maintenance. In *Proceedings of the 6th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 1–12, 1987.

[12] Benjamin Doerr and Mahmoud Fouz. Asymptotically optimal randomized rumor spreading. In *Proceedings of the 38th International Conference on Automata, Languages and Programming - Volume Part II (ICALP)*, pages 502–513, 2011.

[13] Benjamin Doerr, Mahmoud Fouz, and Tobias Friedrich. Social networks spread rumors in sublogarithmic time. In *Proceedings of the 43rd Annual ACM Symposium on Theory of Computing (STOC)*, pages 21–30, 2011.

[14] Benjamin Doerr, Mahmoud Fouz, and Tobias Friedrich. Asynchronous rumor spreading in preferential attachment graphs. In *Proceedings of the 13th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT)*, pages 307–315, 2012.

[15] Benjamin Doerr, Tobias Friedrich, and Thomas Sauerwald. Quasirandom rumor spreading. In *Proceedings of the 19th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 773–781, 2008.

[16] Benjamin Doerr, Tobias Friedrich, and Thomas Sauerwald. Quasirandom rumor spreading: Expanders, push vs. pull, and robustness. In *36th International Colloquium on Automata, Languages and Programming (ICALP)(1)*, pages 366–377, 2009.

[17] Uriel Feige, David Peleg, Prabhakar Raghavan, and Eli Upfal. Randomized broadcast in networks. *Random Structures and Algorithms*, 1(4):447–460, 1990.

[18] Nikolaos Fountoulakis and Anna Huber. Quasirandom rumor spreading on the complete graph is as fast as randomized rumor spreading. *SIAM Journal on Discrete Mathematics*, 23(4):1964–1991, 2009.

[19] Nikolaos Fountoulakis, Konstantinos Panagiotou, and Thomas Sauerwald. Ultra-fast rumor spreading in social networks. In *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1642–1660, 2012.

[20] Pierre Fraigniaud and George Giakkoupis. On the bit communication complexity of randomized rumor spreading. In *Proceedings of the 22nd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 134–143, 2010.

[21] Harold Gabow and Herbert Westermann. Forests, frames, and games: algorithms for matroid sums and applications. In *Proceedings of the 20th ACM Symposium on Theory of Computing (STOC)*, pages 407–421, 1988.

[22] Tibor Gallai. On directed graphs and circuits. In *Theory of Graphs (Proceedings of the Colloquium in Tihany 1966)*, pages 115–118, 1968.

[23] George Giakkoupis. Tight bounds for rumor spreading in graphs of a given conductance. In *Proceedings of the 28th International Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 57–68, 2011.

[24] George Giakkoupis and Thomas Sauerwald. Rumor spreading and vertex expansion. In *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1623–1641, 2012.

[25] George Giakkoupis, Thomas Sauerwald, He Sun, and Philipp Woelfel. Low randomness rumor spreading via hashing. In *Proceedings of the 29th International Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 314–325, 2012.

[26] George Giakkoupis and Philipp Woelfel. On the randomness requirements of rumor spreading. In *Proceedings of the 22nd ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 449–461, 2011.

[27] Bernhard Haeupler. Analyzing network coding gossip made easy. In *Proceedings of the 43rd ACM Symposium on Theory of Computing (STOC)*, pages 293–302, 2011.

[28] Bernhard Haeupler. Simple, fast and deterministic gossip and rumor spreading. In *Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 705–716, 2013.

[29] Maria Hasse. Zur algebraischen begrndung der graphentheorie. i. *Mathematische Nachrichten*, 28 (5-6):275–290, 1965. In German.

[30] Richard M. Karp, Christian Schindelhauer, Scott Shenker, and Berthold Vöcking. Randomized rumor spreading. In *Proceedings of the 41st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 565–574, 2000.

[31] David Kempe, Jon Kleinberg, and Alan Demers. Spatial gossip and resource location protocols. In *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing (STOC)*, pages 163–172, 2001.

[32] David Kempe and Jon M. Kleinberg. Protocols and impossibility results for gossip-based communication mechanisms. In *Proceedings of the 43rd Symposium on Foundations of Computer Science (FOCS)*, pages 471–480, 2002.

[33] Sukhamay Kundu. Bounds on the number of disjoint spanning trees. *Journal of Combinatorial Theory, Series B*, 17(2):199 – 203, 1974.

[34] Nathan Linial. Locality in distributed graph algorithms. *SIAM Journal on Computing*, 21(1):193–201, 1992.

[35] Michael Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM Journal on Computing*, 15:1036–1053, 1986.

[36] Karl Menger. Zur allgemeinen kurventheorie. *Fundamenta Mathematicae*, 10(1):96–115, 1927.

[37] Damon Mosk-Aoyama and Devavrat Shah. Computing separable functions via gossip. In *Proceedings of the 25th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 113–122, 2006.

[38] Crispin St. John Alvah Nash-Williams. Edge-disjoint spanning trees of finite graphs. *Journal of the London Mathematical Society*, 36:445–450, 1961.

[39] David Peleg. *Distributed computing: a locality-sensitive approach.* Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.

[40] David Peleg and Alejandro A. Schäffer. Graph spanners. *Journal of graph theory*, 13(1):99–116, 1989.

[41] David Peleg and Jeffrey D. Ullman. An optimal synchronizer for the hypercube. *SIAM Journal on Computing*, 18(4):740–747, 1989.

[42] Seth Pettie. Low distortion spanners. *ACM Transactions on Algorithms (TALG)*, 6:7:1–7:22, 2009.

[43] Bernard Roy. Nombre chromatique et plus longs chemins d'un graphe. *Rev. Française Informat. Recherche Opèrationnelle*, 1:129–132, 1967. In French.

[44] Thomas Sauerwald and Alexandre Stauffer. Rumor spreading and vertex expansion on regular graphs. In *Proceedings of the 23rd ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 462–475, 2011.

[45] Donald M. Topkis. Concurrent broadcast for information dissemination. *IEEE Trans. Software Eng.*, 11(10):1107–1112, 1985.

[46] W. T. Tutte. On the problem of decomposing a graph into n connected factors. *J. of the London Math. Society*, 36:221–230, 1961.

[47] L.M. Vitaver. Determination of minimal coloring of vertices of a graph by means of boolean powers of the incidence matrix. *Dokl. Akad. Nauk. SSSR147*, pages 758–759, 1962. In Russian.