Uncovering the Useful Structures of Complex Networks in Socially-Rich and Dynamic Environments

Jie Wu
Center for Networked Computing
Department of Computer and Information Sciences
Temple University, USA

***Abstract*—Many group activities can be represented as a complex network where entities (vertices) are connected in pairs by lines (edges). Uncovering a useful global structure of complex networks is important for understanding system behaviors and in providing global guidance for application designs. We briefly review existing network models, discuss several tools used in the traditional graph theory, distributed computing, distributed systems, and social network communities, and point out their limitations. We discuss opportunities to uncover the structural properties of complex networks, especially in a mobile environment, and we summarize three promising approaches for uncovering useful structures: trimming, layering, and remapping. Finally, we present some challenges in algorithmic techniques, with a focus on distributed and localized solutions, to represent various structures.**

***Index Terms*—Complex networks, distributed and localized solutions, dynamic systems, social networks, structural properties.**

I. Introduction

Many group activities can be represented by a complex network where entities (vertices) are connected in pairs by lines (edges). Such a complex network is applicable in multiple different fields and can impact everything from the Internet, food web, and metabolic networks, to social networks. Uncovering the useful global structure (or simply, the structure) of a complex network is important since structural properties can facilitate efficient implementation of various applications, for instance, information dissemination. Many structures are embedded and are influenced by various factors. For example, complex networks may consist of multiple layers [1] from application sessions and social relationships to physical network layers. Interactions and influences between layers may play important roles in shaping network structures.

There are several success stories of using special structures to support various applications. In a small-world network with six-degrees of separation, if node connection follows the inverse-square distribution (i.e., the probability that two nodes u and v are connected is proportional to $distance(u,v)^{-2}$), a localized solution [2] exists in which each node knows only its own local connections and is capable of finding short paths with a high probability. This paper will focus on three areas of research related to structures in a complex network: (1) determining the appropriate graph models, (2) uncovering useful structures for a given graph model, and (3) designing algorithmic techniques to represent structures.

Some discussion on each topic is in order. The traditional graph model is commonly used in modeling a complex network. However, the graph model is not convenient in modeling the dynamic nature of complex networks where node connections change over time. A structure can be "logical" like a special property associated with a network (e.g., small-world) or "physical" like a special subnetwork (e.g., the backbone in the Internet). A structure considered in this paper is global that spans the whole network. Research on social networks reveals some interesting metrics and properties, such as the centrality associated with nodes and the power-law and heavy-tail distributions on node degree distribution. These metrics and properties are not considered structures in this paper unless they are connected and can span the whole network. Algorithmic techniques deal with ways of representing structures. Structures can be determined either ahead of time in a static setting or on-the-fly in a dynamic setting.

We will discuss the challenges we faced dealing with all three areas and explore possible solutions. In a dynamic environment, node connections (or contacts) are based on the notion of "vicinity" among nodes across time and space. We will examine two special intersection graphs, unit disk graphs and interval graphs, and explore their limitations before looking at a more general time-evolving graph. In uncovering useful structures, we do not have simple solutions for different types of applications. Instead, we study three strategies that can be used for a range of applications: *trimming, layering*, and *remapping*. Each strategy is illustrated through several applications. Finally, each structure is represented in either a distributed or localized solution. We advocate distributed or local labeling schemes that use colors and labels to identify logical and physical structures.

Throughout the paper, we describe approaches used in different communities, including graph theory, distributed computing (such as PODC in the theoretical community), distributed systems (such as ICDCS in the system community), and social networks. Discussion will be focused on the differences between different methods and possible extensions. Because the area under study is vast, this position paper covers only a subset of problems in the computing field; it does not intend to be a comprehensive survey.

The remainder of this paper is organized as follows. Section

This research was supported in part by NSF grants CNS 1629746, CNS 1564128, CNS 1449860, CNS 1461932, CNS 1460971, and CNS 1439672.

II surveys relevant graph models for complex networks with a focus on special intersection graphs and general time-evolving graphs. Section III describes three ways of uncovering structures. Section IV discusses algorithmic techniques that are distributed or localized to represent a structure. Finally, Section V concludes the paper.

II. Graph Model

Traditionally, a complex network can be represented as a graph $G = (V, E)$ with a vertex (node) set V and an edge (link) set E. In computing, networks include, but are not limited to, the Internet, peer-to-peer (P2P), mobile ad hoc (MANET), sensor, vehicular ad hoc (VANET), social, and delay/disruption tolerant (DTN) networks. These networks operate under special environments which pose unique challenges to the network design. For example, in mobile networks like DTNs and VANETs, network connection is highly dynamic and disruptive due to node mobility. In these networks, links are also called *contacts* between two nodes with appropriate contact duration and inter-contact time. The traditional graph model cannot sufficiently capture the dynamic nature of the connections in the mobile networks.

- *Which graph model is suitable for representing a complex network?*

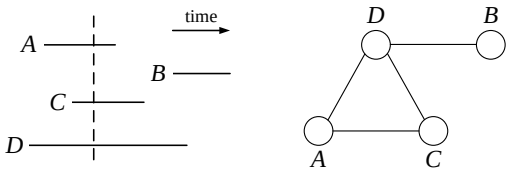
Unfortunately, there is no model that can fit all cases. We first explore models for special cases and follow with a more general model.

A. Special cases of the graph model

There are several types of networks where node connections are based on their vicinities in time (such as online social networks) and in space (such as sensor networks, MANETs, and VANETs). Intersection graphs can be used for these cases. *Intersection graphs* [3] are formed from a family of sets S_i, $i = 1, 2, ...$ by creating one vertex v_i for each set S_i and connecting two vertices v_i and v_j by an edge whenever the corresponding two sets have a nonempty intersection: $E = \{(v_i, v_j) | S_i \cap S_j \neq \emptyset\}$.

Among special intersection graphs, *unit disk graphs* are a family of unit disks (for set S_i) in the 2-D space. Each unit disk is a vertex. An edge exits wherever the corresponding vertices lie within a unit distance of each other. Unit disk graphs have been extensively studied for sensor network, MANET, and VANET applications. Note that not all graphs are unit disk graphs. A star graph with one center node and six or more leaves is such an example. As a special graph, unit disk graphs have some unique properties that general graphs do not have. A constant approximation algorithm exists to solve the minimum traveling salesman problem (TSP) in unit disk graphs, but not in general graphs.

An *interval graph* is an intersection graph of a family of intervals on the real line (for set S_i). Each line interval represents a vertex. An edge exists if the corresponding intervals intersect. If a line interval represents a time period, an interval graph can be used to represent an *online social network* (see Figs. 1 (a) and (b)). Not all graphs are interval graphs. cannot be part of an interval graph because the time is linear, not circular. In fact, if G is an interval graph, it must be a chordal graph. A *chordal graph* is one in which all cycles of four or more vertices have a chord, which is an edge that is not part of the cycle but connects two vertices of the cycle. The impossibility of a large chordless cycle is that time is linear, not circular. In an online social network, each user can be online multiple times, and *multiple-interval graphs* (interval graphs where each vertex may have more than one interval associated with it) can be used. The following question arises: what are the unique properties we can explore for multiple-interval graphs?



(a) Line intervals along time. (b) Interval graph for (a).

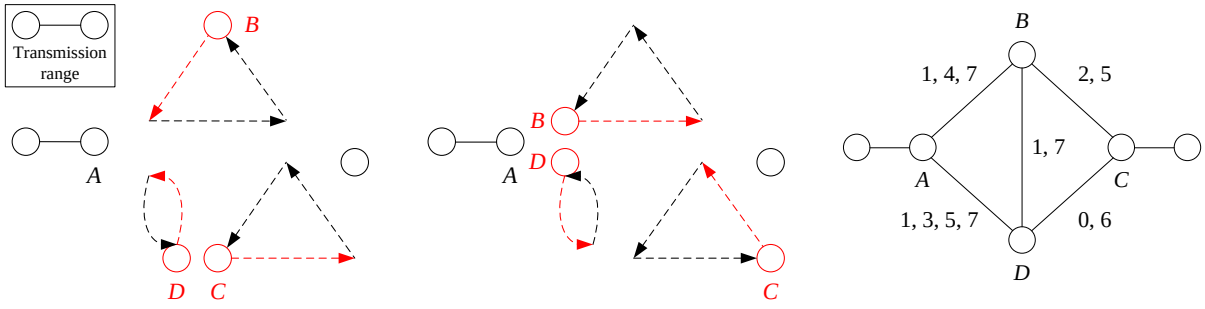
Fig. 1. Illustration of interval graphs.

Vertices in interval graphs are not necessarily connected in pairs. In Fig. 1 (a), three nodes A, C, and D are intersected at a particular time moment; this example is analogous to an online social network with three users online at the same time. A *hyperedge*, a generalized edge connecting more than two vertices, seems to be more appropriate in this case. An *interval hypergraph* can be defined where an additional hyperedge among A, C, and D should be added to Fig 1 (b). What type of distribution of hyperedge cardinality will follow? More importantly, what types of properties of online social networks can be revealed through the edge density distribution? Understanding the edge density distribution can also play an important role in understanding online social network behaviors like social influencing and recommendation.

B. Time-evolving graph model

Time-evolving graphs [4] try to present graphs in both time and space. Such graphs have also been called temporal graphs and time-varying graphs in different settings. We start with a brief overview of time-evolving graphs, time-sensitive connectivity, and different performance measures for a path.

Let $G = (V, E)$ be a graph. $G_0, G_1, ..., G_k$ is an ordered sequence of spanning subgraphs for the time sequence $t_0, t_1, ..., t_k$. $G_i = (V_i, E_i)$ is called a subgraph during $[t_i, t_{i+1})$. The corresponding *time-evolving graph*, denoted as EG, is the collection of G_i in which each edge (u, v) is associated with an *edge label set* $\{i | (u, v) \in E_i\}$. Figs. 2 (a) and (b) show two snapshots of a VANET that has three mobile nodes and three static nodes. Fig. 2 (c) is the corresponding time-evolving graph. Edge labels have cycles, e.g., (B, D) and (C, D) have a cycle of 6, (A, D) has 2, and (A, B) and (B, C) have 3. In EG, $u \xrightarrow{i} v$ denotes the edge label which indicates the existence of edge (u, v) during time unit i. Message transmission over an edge (also called a *contact*) is instantaneous. A path $u \xrightarrow{*} v$ is an alternative sequence of



(a) Snapshot at time unit 0. (b) Snapshot at time unit 1. (c) Corresponding time-evolving graph.

Fig. 2. A VANET example with three mobile nodes, B, C, and D, with moving cycles 3, 3, and 2, respectively and three static nodes.

vertices and edges with non-decreasing edge labels. Vertex u is said to be connected to v at time unit i if a path $u \xrightarrow{*} v$ exists whose first edge label is larger than or equal to i. In Fig. 2 (c), path $A \xrightarrow{4} B \xrightarrow{5} C$ exists, therefore, A is connected to C at starting time units 0, 1, 2, 3, and 4, assuming each vertex, including A, has sufficient storage capability to store messages from the previous contacts. At a particular time unit, two vertices may not be connected. In fact, A and C in Fig. 2 are not connected at any particular time unit. Hence, the network is not connected at any given time. However, carry-store-forward routing can still deliver messages. A *weighted time-evolving graph* has a definition similar to the time-evolving graph except that each edge at time unit i is associated with a weight w_i, which have different interpretations based on the application. For example, a weight can be the bandwidth, transmission delay, or reliability.

Using EG, any topological terminology can be extended to a temporal one. For example, path is extended to *journey* (which is a path over time), distance to *temporal distance*, and diameter to *dynamic diameter* (which is flooding time). We can consider the following path optimization problems as extensions of the traditional shortest path problem, but still solvable using the variations of the classical Dijkstra's shortest path algorithm: (1) *Earliest completion time path*: find a path with the earliest completion time at the destination. (2) *Minimum hop path*: find a path with the minimum number of hops to the destination. (3) *Fastest path*: find a path with the minimum span (i.e., elapsed time) between its first contact and its last contact. We assume that the transmission at each contact is instantaneous.

One challenge is how a graph model can capture the essence of complex networks while still being simple enough that many optimization problems are tractable. In this spirit, we should exclude non-essential parameters. Another approach uses a *macro-level model* instead of micro-level time-unit labels for each edge. In the system community, time-unit labels are abstracted as contacts that follow a certain distribution based on a given mobility model [5]. Two measures are often used: contact duration distribution and inter-contact time distribution. The exponential distribution is frequently used due to the simplicity of its mathematics. However, a random waypoint mobility without a boundary does not meet the exponential distribution for either contact duration or inter-contact time. In the theoretical community, the Markovian process in which network topology at time unit i depends only on its topology at time unit $i-1$ is commonly used, and it has a unique stationary distribution. For example, the elegant two-state edge-Markovian process is used to describe edge dynamics. If an edge exists at time i, at time $i+1$, it dies with probability p. If the edge does not exist at time i, it will appear at time $i+1$ with another probability q. This model has been successfully used to calculate the dynamic diameter [6]. However, Markovian models are still overly simple for various mobility patterns. The question of the existence of other models, which are both mathematically elegant and which better match practical mobility models, remains.

In general, there is a trade-off between the expressiveness and the decision power of a model. We should strive for the simplest model, e.g., a model focused on space or time only with just enough power to study the problem at hand. However, a more powerful model focused on both time and space potentially reveals more properties than a weaker one.

III. Uncovering Useful Structures

This section focuses on uncovering useful network structures. There has been a good amount of work done in social networks in terms of *centrality* [7]. Among various forms of centrality, *node degree* measures the number of neighbors of a node. *Closeness* is the average length of the shortest path between a node and all other nodes in the graph. *Betweeness* quantifies the number of times a node acts as the bridge along the shortest path between two other nodes. Eigenvector centrality, including PageRank, measures a node score as the score summation of its neighbors. Centrality primarily measures the importance of a single node. Here, we focus on structures or structural properties that span the whole network. In addition, such structures or structural properties are useful in supporting network-wide applications. Like centrality, a network structure that is optimal for one application is often sub-optimal for a different application. We discuss three approaches that can be used to uncover such useful structures for various applications.

A. Structural trimming

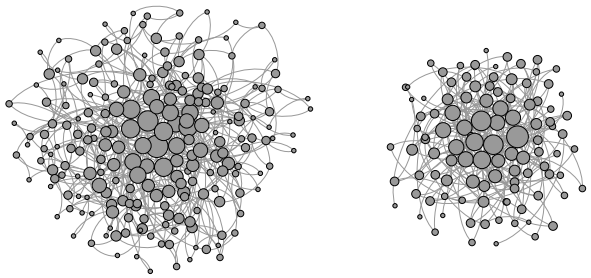
Structural trimming deals with removing links and/or nodes to form a subgraph as a useful structure. Usually a subgraph

maintains several of the global properties of the original graph. Basic properties include connectivity and inclusion of a minimum spanning tree or a shortest path tree. In some cases, a property is an approximate for a global measure. For example, subgraph distances closely resemble the distances in the original graph for designing the approximation algorithms for the graph problems [8]. There is also a vast literature concerning P2P networks [9] that add connections to form an overlay network. Here, we focus on structural trimming instead of overlay. The main purpose of trimming is to reduce the complexity of information dissemination or to reduce the complexity of network searching without losing the desirable properties of the original network topology. In wireless networks, topology sparsity also reduces the bandwidth contention that occurs during simultaneous wireless transmissions. Structural trimming can be conducted by static and dynamic trimming.

Static trimming is usually conducted through *topology control* [10]. Various localized trimming processes for unit disk graphs with known locations or with neighborhood connectivity information have been studied. Certain properties are more difficult to maintain than others. For example, maintaining a minimum shortest path tree using local location information is more involved than maintaining a minimum spanning tree.

As most existing work on trimming has been on the traditional graph, we use the time-evolving graph EG to illustrate the use of local information to trim "useless" or "redundant" nodes and links. Usually, local information (within k hops for a small k) does not cause excessive propagation delay. For example, for 2-hop information, each node maintains its own neighborhood information in the EG and exchanges this information with its neighbors. When a node or link is removed, the network connectivity remains the same. More specifically, if the network is time-i-connected, it remains connected after using the following trimming rule: *node u can be trimmed if for any path $w \xrightarrow{i} u \xrightarrow{j} v$ such that $i \leq j$ there is another path, called* a replacement path*, $w \xrightarrow{i'} u_1 \rightarrow ... \rightarrow u_k \xrightarrow{j'} v$ such that $i \leq i'$ and $j' \leq j$*. Here, we only compare the edge lables of the first and last hops in these two paths. In Fig. 2 (c), any path $A \rightarrow D \rightarrow C$ can be replaced by a path $A \rightarrow B \rightarrow C$. For example, $A \xrightarrow{3} D \xrightarrow{6} C$ can be replaced by $A \xrightarrow{4} B \xrightarrow{5} C$. Therefore, A can ignore neighbor D. It is possible that two paths can replace each other. To avoid circular replacement, each node u is assigned a distinct priority $p(u)$. A node can be replaced only if its priority is lower than all intermediate nodes in the replacement path. We assume that $p(A) > p(B) > p(C) > ...$ based on node IDs. We can also assign priority, say using node degree or node betweenness, based on the strategic importance of the node in the network topology.

Trimming rules can vary depending on the specific properties being preserved. In the current rule, the minimum completion time is preserved, but not necessarily the minimum hop count. To enforce this, we can require that each replacement path have, at most, one intermediate node. Other options include removing a specific label on a link and developing a *link replacement rule*. Note that the link replacement rule is a refinement of the node replacement rule. When a node is trimmed, all of its adjacent links are removed as well. In situations where link labels are not deterministically, but rather, probabilistically, known, it would be interesting to explore different probabilistic versions of the trimming rule. Clearly, more research is needed on local trimming in time-evolving graphs maintaining a given set of global properties.



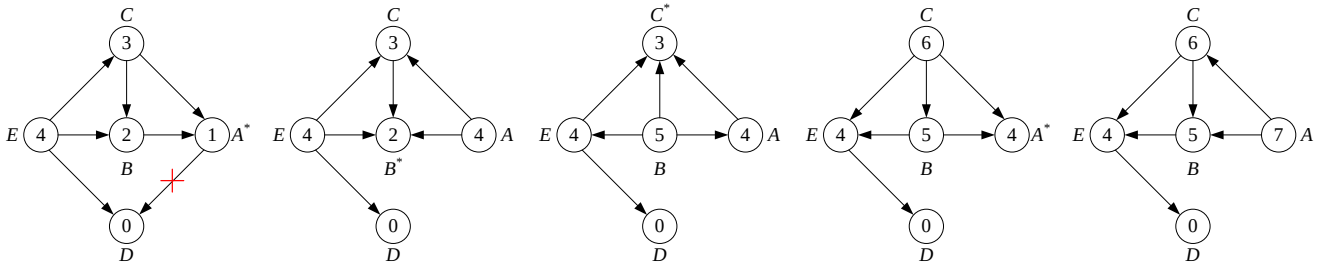
(a) A P2P network with all peers. (b) Top 50% peers.

Fig. 3. NSF in a Gnutella dataset [11].

Dynamic trimming is much more involved since its performance depends on various factors in different applications. Dynamic trimming is an online version of a trimming process based on local information only for a particular application (e.g., routing). This can lead to efficiency improvement because the decision is based on timely local information. In Fig. 2, path $D \rightarrow A \rightarrow B$ cannot be replaced by $D \rightarrow B$, but it can be replaced at time unit 1. A more complex trimming occurs in a time-evolving graph with a probabilistic contact distribution. In a routing process in a dynamic network, should a message be forwarded at a new contact (which may lead to a less favorable path) or at a future contact (which may lead to a favorable path)? The notion of a *forwarding set* (a neighbor subset) is used in [12]. In this approach, the single-copy message is forwarded to a new contact if the contact belongs to the forward set (i.e., the link connects to the neighbor in the set). This is analogous to a multi-bus riding: should a passenger ride on the first bus to arrive, even though its route may be longer, or should the passenger wait for a later bus with a shorter route but an uncertain arrival time?

- *How can we design a general methodology to derive a forwarding set in a dynamic network?*

The answer to this question depends on the contact distribution and the settings for objectives and constraints. For example, in a multi-copy message deliver application, the forwarding set becomes *copy-varying* if the objective is to minimize the delivery time of the first copy. On the other hand, if the utility of the message is time-sensitive and it decays over time, the forwarding set becomes *time-varying* when the objective is to maximize utility. [13] shows that if the inter-contact time follows the exponential distribution and the message utility decays linearly over time, an optimal time-varying forwarding set can be derived. In fact, the forwarding set at the same intermediate node shrinks over time.



(a) Reversal at A. (b) Reversal at B. (c) Reversal at C. (d) Reversal at A again. (e) Destination-oriented DAG.

Fig. 4. A full link reversal process (a)-(e) after a broken link (A, D) in destination-oriented DAG (a). Labels inside nodes correspond to heights.

B. Structural layering

The second approach is based on layering through the assignment of hierarchical levels to the nodes. Such a structure is either embedded in a given graph or man-made by assigning an appropriate height to each node that will either define the levels of nodes or control the orientations of the links.

Embedded layering. In many unstructured P2P systems, including Gnutella and Bitcoin, system structures are not only *scale-free* (SF) (i.e., node degree distribution follows the power-law distribution), but also *nested scale-free* (NSF) [11]. SF hierarchy is defined by ranking levels based on node degrees. Let $G^{'}$ denote the set of subgraphs generated by iteratively removing the local lowest-degree nodes and their connections in a network G. G satisfies NSF if (1) G and all subgraphs in $G^{'}$ satisfy SF, and (2) the standard deviation of power-law exponents in G and all subgraphs in $G^{'}$ is $o(1)$. Condition (2) states that G and all subgraphs in $G^{'}$ are "similar" in structure, as shown in Fig. 3. Fig. 3 (a) shows the largest strongly-connected component formed in a Gnutella dataset [14]. Fig. 3(b) is derived from G by iteratively removing the local lowest-degree peers until only half of the peers remain. The hierarchical levels can be maintained by a labeling scheme (to be discussed in the next section) that assigns each node a level called "height". The hierarchical structure can facilitate efficient implementations of the pub-sub systems through push (moving up through the layered structure) and pull (coming down through the layered structure).

- *Can we uncover more inherent layered structures, not only in the space dimension, but also in time-and-space?*

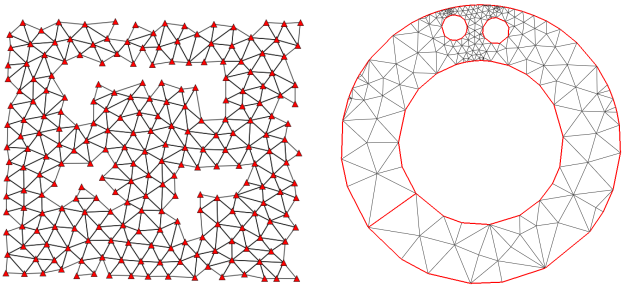
A possible solution is to present an application in a dynamic network. A good collection of data traces that can represent time-and-space dimensions is also needed to gain insights. The work done on the small-world behavior of the real-world in time-and-space dimensions [15] has the potential to explore the layered structure of a complex network.

Man-made layering. Several layered structures are man-made to assist certain applications. Suppose that when routing to a given destination, a destination-oriented directed acyclic graph (DAG) has been maintained initially, as shown in Fig. 4 (a) with one destination sink D (ignore labels inside nodes as they will be discussed in the next section). A given source node can take any route without using a routing table to reach the destination. When a non-destination node (say A in Fig 4 (b)) has no outgoing link (A becomes a sink) due to a broken link, it reverses all adjacent links through a *full link reversal* process [16]. This may trigger the full link reversal of other nodes, as shown in Figs.4 (c), (d), and (e). Eventually, the process terminates forming a new destination-oriented DAG, as shown in Fig. 4 (e). The orientation of each link is systematically determined by assigning appropriate heights of two connecting nodes (details to be discussed in the next section). Another application of the dynamic destination-oriented DAG is used to construct an efficient implementation of the classical max-flow problem [17]. In this approach, the orientations of the links are dynamically calculated and adjusted by the heights of each node (except the destination sink) in multiple rounds, while maintaining the destination-oriented DAG structure. Flows associated with neighboring nodes iteratively and gradually flow among themselves, but eventually flow towards the sink, based on link orientations determined by the node's heights. The efficient maintenance of DAG in a dynamic network is certainly a challenge, as we will discuss in the section. A related challenge is finding an efficient way of maintaining DAGs simultaneously for multiple destinations.

C. Structural remapping

In some applications, the complexity of a problem can be reduced or even removed by carefully remapping from one representation to another representation or from one domain to another.

Remapping representation. In the Euclidean space, greedy geographic routing [18] is commonly used to greedily reduce the Euclidean distance between the source and destination. However, such a greedy process may get stuck at a local minimum, such as at one of three non-convex hole shown in Fig. 5 (a). This situation happens when the bottom-right node sends a message greedily to the top-left node and the message becomes stuck at a non-convex hole; it cannot make further progress towards the destination. By mapping the Euclidean space to the hyperbolic space, [19] shows that carefully assigning each node a *virtual coordinate* in the hyperbolic plane allows the greedy algorithm to succeed in finding a route to the destination. *Conformal mapping* [20] using Ricci flow will guarantee delivery for the greedy forwarding. A key idea is transforming holes into perfect circles, as shown in Fig. 5 (b), to avoid being stuck at an intermediate node during a greedy routing process.



(a) Graph with non-convex holes. (b) Virtual coordinate map.

Fig. 5. Conformal mapping [20].

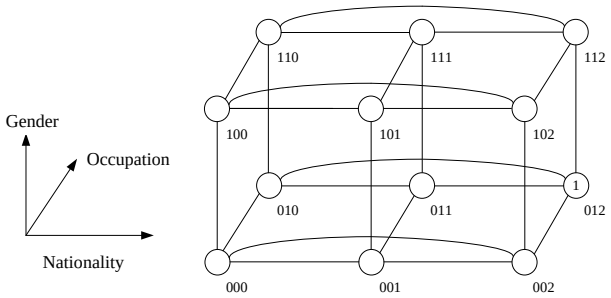


Fig. 6. A static and structured feature space as a 3-D generalized hypercube.

- *Instead of mapping from one representation to another, such as when mapping Euclidean space to non-Euclidean space, can we remap a problem from one domain to a different domain?*

Fourier analysis, which converts a signal from its original time or space domain to a representation in the frequency domain, provides a good example of how to address the question above. Fast Fourier Transform (FFT) is usually used to compute the discrete Fourier transform of a sequence. We show one potential mobile application below.

Remapping domain. In mobile applications in MANETs and VANETs, due to the high mobility of nodes, contacts are highly irregular. It is difficult to derive a useful structure for dynamic graphs. [21] observes that in social contact networks, node (personal) contacts in the physical layer are influenced by the corresponding social features in the social relationship layer. As confirmed from several real traces, including INFOCOM 2006 and MIT Reality Mining, the frequency of the personal contacts of two nodes (persons) is dependent on their feature distance. The closer the distance, the higher the contact frequency. Here, each person is represented by a social feature profile. The social features represent either physical features, such as gender, or logical ones, such as occupation. In Fig. 6, each person is described by three features: gender (male and female), occupation (professional and student), and nationality (say, country 1, country 2, and country 3). Suppose we group all individuals with the same features in one node. Two nodes are connected if they differ in exactly one feature; a *generalized hypercube* is generated. In this way, we convert a routing process in a *highly mobile and unstructured contact space* (M-space) to one in a *static and structured feature space* (F-space) represented as a generalized hypercube. A generalized hypercube can easily support shortest-path routing as well as node-disjoint multiple-path routing. Note that links in a generalized hypercube correspond to *strong links* (terminology commonly used in social networks) among nodes with one feature difference. Each node corresponds to one community of people with common features and the most frequent contacts. Although links that are not in a generalized hypercube do exist, they correspond to *weak links* with lower probabilities for their larger feature distance.

The above example shows one application of social networks' influence on the underlying contact network. As more and more applications are related to social networks, we need to address the following question:

- *How can we systematically uncover the influence that social relationships have on the structure of an underlying network for socially-rich applications?*

More data traces need to be collected to validate any hypothesis on the influence that social relationships and features have on underlying network contact structure.

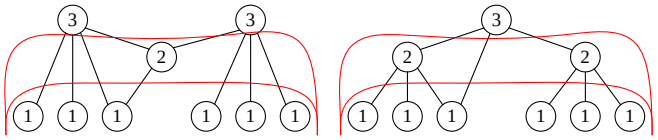
IV. Distributed and Localized Solutions

A *distributed solution* involves nodes that interact with others in a restricted vicinity. Each node usually performs simple tasks, such as maintaining and propagating information *labels*. Collectively, these nodes achieve a desired global objective. A *localized solution* is a distributed solution in which there is no sequential propagation of information. A set of labels are usually used to describe nodes at different levels or with different link orientations. It is assumed that each node knows k-hop information (or *local horizon* in the theoretical community) for a small constant k. Each node has a distinct ID within k-hop for symmetry breaking. A centralized solution can be converted to a distributed solution. For example, Dijkstra's shortest path can be implemented in a distributed way; each leaf node will report to the root its distance information at each round of relaxation. The root will inform whichever leaf node corresponds to the shortest path from the root to all potential new leaves. Back-and-forth propagation between the root and the leaves is not efficient because it requires multiple rounds of information exchanges.

A. Static Labels

The static labeling process refers to a labeling process in which each node is labeled a small number of times for a given network topology. In a localized *connected dominating set* (CDS)[1] selection for a virtual backbone in senor networks and MANETs [22], two colors are used: black for CDS nodes and white for non-CDS nodes. Initially, all nodes are white. If a node has two unconnected neighbors, it labels itself black. All black nodes form a CDS. In Fig. 8, all nodes except A are labeled black. A trimming process can be applied locally to change black back to white if a black node's neighborhood is

[1]Subset $D \subseteq V$ is a *dominating set* (DS) if every node in $V \backslash D$ has a neighbor in D. D is connected, i.e., CDS, if the induced $G[D]$ is connected.



(a) Node degree (b) Nested node degree

Fig. 7. Labeling based on (a) node degree and (b) nested node degree.

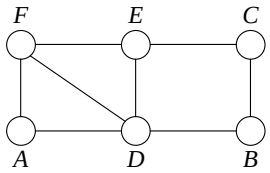


Fig. 8. An example for static labeling for DS, CDS, and MIS.

covered by other connected and higher priority black nodes. In Fig. 8, B, C, and D are three black nodes remained after the trimming process. Distributed clusterhead calculation uses three colors to determine a *maximal independent set* (MIS)[2] a DS also, in $\log n$ rounds, where n is the number of nodes. Initially all nodes are white. If a node is the local 1-hop maximum (in terms of priorities) among white neighbors, it is colored black (and becomes a clusterhead). A and B are colored black in Fig. 1 (c). A node with a black neighbor is labeled gray (and is removed from the next round of competition). This process repeats until there is no white node. The final MIS in Fig. 8 is A, B, and E, all colored black. The color of each node does not have to be *self-determined*. It can also be *neighbor-designated* like in the following localized DS calculation: each node selects one winner (say, the one with the highest priority) from its 1-hop neighborhood including itself. A node is colored black if it is selected by at least one node. This process terminates in one round. In Fig. 1 (c), A, B, and C are selected as DS (but not a CDS or an IS). One question can be posed: is there another efficient, localized labeling scheme besides self-determined and neighbor-designated?

Labels are frequently used to represent levels in the node hierarchy. For NSF [11], labels are given iteratively as follows: Initially, all nodes are unassigned. An adjusted node degree is defined as the number of unassigned neighbors. In the first round, nodes that are local minimum in terms of the adjusted node degree are assigned to level 1. These nodes become assigned. We repeat the above process and adjust the level by adding 1, until all nodes are assigned. Note the difference between Figs. 7 (a) and (b) in terms of node hierarchy. We aim to derive a structure with only one node at the top level. In NSF, there may still be multiple top-level nodes that are not connected to each other. In this case, it is assumed that an external server is used to connect them.

B. Dynamic Labels

The dynamic labeling process refers to a labeling process where several nodes are repeatedly labeled a large number of times. By large, we mean a non-constant number. The Bellmen-Ford algorithm maintains the shortest path and distance information from each node to a destination. Each distance estimation at a node can be considered a labeling process which involves many rounds of routing table update in case of a link failure. PageRank and HITS (also known as hubs and authorities) [23] are another two examples of dynamic labeling used to rank websites.

In link reversal, the levels associated with the nodes correspond to the node's heights. In Fig. 4, each node is initially assigned a distinct level, called height, with the destination being the lowest level: 0. The orientation of each link is defined by the relative heights of two end nodes; the node with the higher height points to the node with the lower height. Suppose that a connected digraph is no longer a destination-oriented DAG. Then, there are sinks other than the destination. We can simply raise the levels of these sinks, but only so that they are higher than their highest neighbors by 1. All links associated with these nodes reverse their links in the full link reversal. A link reversal may cause ripple-effects and trigger surrounding nodes to reverse their links. In fact, each node may be involved in multiple rounds of reversals, like node A in Fig. 4. Overall, the number of reversals is $O(n^2)$, where n is the number of nodes. This high cost in a slow convergence is detrimental to the practicality of this elegant approach. Although *partial link reversal* [16] improves performance by reversing a subset of links at each reversal, it does not reduce the overall complexity.

We pose two questions: (1) are there any other ways to represent labels? and (2) can a label be associated with a link? The answer to both queries is "yes". One recent result shows the use of a binary label associated with each link as a label for a link reversal [24]. A destination-oriented DAG is maintained initially. Two rules are used on non-destination sinks due to broken links. Rule 1: if at least one link incident on node i is labeled 0, then all the links incident on node i that are labeled 0 are reversed. The other incident links are not reversed, and the labels on all the incident links are flipped. Rule 2: if all the links incident on node i are labeled 1, then all the links incident on i are reversed, but none of their labels change. This approach not only unifies the full link reversal (1 for all links initially and using Rule 1 only) and the partial link reversal (0 for all links initially and using Rule 1 and Rule 2), but also provides a more convenient way of complexity analysis.

C. Challenges

A good amount of survey has been done in the theoretical community on the merits and limitations of distributed [25] and localized solutions [26]. We discuss here some additional challenges, starting with one fundamental question:

- *In a complex network, especially in a dynamic environment, how can we deal with the complexity of building a structure along with the change of topology?*

[2]Subset $I \subseteq V$ is an *independent set* (IS) if no two nodes in I are adjacent. I is maximal, i.e., MIS, if it cannot be extended. In a unit disk graph, because no node can have six neighbors that are mutually independent, no MIS can be more than five times minimum CDS. MIS is frequently used to construct a minimal CDS using a small number of gateways to connect nodes in MIS.

In a dynamic network, network topology and node status change; how can the construction of an underlying structure be better integrated with changes? Unless these changes are infrequent, many solutions will be rendered useless if they cannot be executed quickly enough. However, a quick execution may not offer optimal results. Localized algorithms seem to be the answer, but they have limitations in terms of obtaining a competitive approximation ratio [27]. An open discussion of the relative merits of an asymptotic bound analysis vs. an average case bound is needed. In some cases, an asymptotic bound analysis has a hidden large constant coefficient. An average case bound, on the other hand, has some rare bad cases, but is generally competitive. The notion of average case itself is a subject of research. *Smoothed analysis* [28] gives a promising approach of expressing the average case through a natural model of noise or perturbation on a given distribution.

Mobility will create another serious problem: *view inconsistency*. In a mobile application, both neighborhood information exchanges (for k-hop information) and asynchronous Hello message exchanges cause delays, which will generate inconsistent neighborhood and location information. Some early results on maintaining multiple views [29] seem promising, but more work is needed to address the general challenge in this area. Another promising area is integrating the process of building a structure with the change of topology due to node movement. This approach is different from most existing approaches where structure re-building occurs after a topology change. A simpler version of topology change is adding and deleting nodes and/or links that frequently appear in sensor networks. [30] shows that although constructing an MIS requires $\log n$ rounds, if MIS is built based on a graph with random priority nodes, an adding/deleting operation requires one round of adjustment in expecttion. Another challenge is the convergence time for the dynamic labeling process:

- *How do we handle the long convergence time for the dynamic label usually occurring in a distributed solution?*

A centralized solution is efficient and flexible, but it is less scalable and suffers from a single point of failure. A localized solution supports parallel executions for scalability and is quick to respond to changes because local information does not propagate. However, localized solutions are limited in problem solving capabilities [27]. Distributed solutions are fault-tolerant and scalable, and they offer good problem solving capabilities. They suffer from a slow convergence, as shown in the Bellman-Ford algorithm for distributed routing and in the link reversal for computing a destination-oriented DAG.

We can explore two fronts. The first is designing a *hybrid centralized-and-distributed* method that can enjoy the merits of both the centralized and distributed solutions. The key issue is how a centralized solution can offer some "guidance" to a distributed one. Internet routing offers some ideas. The distributed solution (the distance vector using the Bellman-Ford solution) is used in intra-domain routing, whereas the centralized (but implemented in a distributed way) path vector is applied in inter-domain routing. A recent work on central

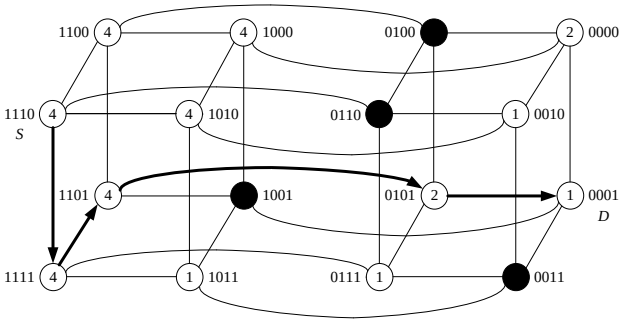


Fig. 9. Safety level labeling in a 4-D binary hypercube.

SDN control over distributed routing offers some interesting insights as well [31]. The proposed architecture achieves both flexibility and robustness by controlling over distributed routing; it inserts fake nodes and links to create an augmented topology for a distributed solution.

The second front is devising a *hybrid distributed-and-localized* method that has the problem-solving capability of a distributed solution and uses the static labels. The work in [32] sheds some light on such a labeling method, design for supporting optimal fault-tolerant routing in an n-dimensional binary hypercube (or n-D cube) with faulty nodes. Instead of maintaining labels for routing capabilities to a particular destination, it maintains a label that indicates the routing capability to a set of destinations within a hop-count. The distributed labeling process enjoys some of properties of a localized solution. This labeling method codes faulty information that propagates only to the affected region. Specifically, if a node is labeled i, then it can find a shortest path to any nodes within i hops, and there is at least one node $i+1$ hops away that cannot be reached through a shortest path. When the safety level of a node is n in an n-D cube, this node can reach any node through a shortest path since the diameter of an n-D cube is n. Such a node is called a *safe* node. The safety level of a node and the safety levels of its neighbors satisfy a special constraint[3]. The safety level of a node is iteratively determined based on the safety levels of its neighbors by rounds of calculation. Initially, all faulty nodes have a level 0 (the lowest) and all non-faulty nodes have a level n (the highest). *Differing from link reversal, each safety level is decided, at most, once*. In fact, if the safety level of a node is i, then the level of this node is decided exactly in round i. As the diameter of an n-D cube is n, at most, $n-1$ rounds are needed.

Safety levels not only provide routing capability information, but can also guide the routing process. No routing table is needed for optimal routing. Fig. 9 shows a 4-D cube with three faulty nodes (colored black). At an intermediate node

[3]The safety level of a faulty node is zero. For non-faulty node u, let (l_0, l_1, l_2, ..., l_{n-1}), $0 \le l_i \le n$, be the non-decreasing safety level sequence of node u's neighboring nodes in an n-D cube, such that $l_i \le l_{i+1}$, $0 \le i < n-1$. u's safety level, $l(u)$, is defined as follows: if (l_0, l_1, l_2, ..., l_{n-1}) $\ge$ (0, 1, 2, ..., $n-1$), then $l(u) = n$; otherwise, if (l_0, l_1, l_2, ..., l_{k-1}) $\ge$ (0, 1, 2, ..., $k-1$) $\wedge$ ($l_k = k-1$) then $l(u) = k$. $seq_1 \ge seq_2$ if all the elements in seq_1 are larger than or equal to the corresponding elements in seq_2.

of the self-guided process, the next hop is the highest safety-level neighbor selected from a subset of neighbors. This subset includes only neighbors that are on the shortest paths from the intermediate node to the given destination. The binary addresses of these neighbors are closer to the destination by one binary bit. In Fig. 9, node 1101 selects 0101 (with a safety level of 2) between two neighbors 1001 and 0101 on route to 0001. Overall, safety levels show a delicate balance between efficiency (in terms of quick structure building) and utility (in terms of structure utility). The application of safety level has been used in optimal fault-tolerant broadcast. The model itself has been extended to more sophisticated *binary safety vectors* and *directed safety levels*. Whether a similar approach can be adopted for a more general setting still remains to be seen.

V. Conclusions

In this paper, we take a snapshot of several key issues in uncovering useful structures in a complex network. We review the existing graph model for a complex network and then provide a discussion of intersection graphs with two special cases: a unit disk graph along space dimension and an interval graph along time dimension. Further, we discuss a general time-evolving graph across both time and space. We look at three possible ways to build a useful structure for a complex network: trimming, layering, and remapping. We offer concrete ideas and possible solutions for each approach in the hopes of triggering further discussions. Finally, we list some challenges in designing distributed and localized solutions used to represent useful structures, and we explore several possible ways to address these challenges.

References

[1] C. Wang, S. Tang, L. Yang, Y. Guo, F. Li, and C. Jiang, "Modeling data dissemination in online social networks: a geographical perspective on bounding network traffic load," in *Proceedings of ACM MobiHoc*, 2014, pp. 53–62.

[2] J. Kleinberg, "The small-world phenomenon: An algorithmic perspective," in *Proceedings of ACM STOC*, 2000, pp. 163–170.

[3] A. Brandstädt, V. B. Le, and J. P. Spinrad, *Graph Classes: a Survey*, 1999.

[4] A. Ferreira, "Building a reference combinatorial model for MANETs," *IEEE Network*, vol. 18, no. 5, pp. 24–29, 2004.

[5] T. Camp, J. Boleng, and V. Davies, "A survey of mobility models for ad hoc network research," *Wireless Communications and Mobile Computing*, vol. 2, no. 5, pp. 483–502, 2002.

[6] A. Clementi, R. Silvestri, and L. Trevisan, "Information spreading in dynamic graphs," *Distributed Computing*, vol. 28, no. 1, pp. 55–73, 2015.

[7] S. P. Borgatti and M. G. Everett, "A graph-theoretic perspective on centrality," *Social Networks*, vol. 28, no. 4, pp. 466–484, 2006.

[8] H. Räcke, "Optimal hierarchical decompositions for congestion minimization in networks," in *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing*. ACM, 2008, pp. 255–264.

[9] E. K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim, "A survey and comparison of peer-to-peer overlay network schemes," *IEEE Communications Surveys & Tutorials*, vol. 7, no. 2, pp. 72–93, 2005.

[10] P. Santi, "Topology control in wireless ad hoc and sensor networks," *ACM Computing Surveys*, vol. 37, no. 2, pp. 164–194, 2005.

[11] H. Zheng and J. Wu, "NSFA: Nested scale-free architecture for scalable publish/subscribe over p2p networks," in *Proceedings of IEEE ICNP*, 2016, pp. 1–10.

[12] V. Conan, J. Leguay, and T. Friedman, "Fixed point opportunistic routing in delay tolerant networks," *IEEE Journal on Selected Areas in Communications*, vol. 26, no. 5, 2008.

[13] M. Xiao, J. Wu, C. Liu, and L. Huang, "Tour: time-sensitive opportunistic utility-based routing in delay tolerant networks," in *Proceedings of IEEE INFOCOM*, 2013, pp. 2085–2091.

[14] J. Leskovec, J. Kleinberg, and C. Faloutsos, "Graph evolution: Densification and shrinking diameters," *ACM Transactions on Knowledge Discovery from Data*, 2007.

[15] J. Tang, S. Scellato, M. Musolesi, C. Mascolo, and V. Latora, "Small-world behavior in time-varying graphs," *Physical Review E*, vol. 81, no. 5, p. 055101, 2010.

[16] E. Gafni and D. Bertsekas, "Distributed algorithms for generating loop-free routes in networks with frequently changing topology," *IEEE Transactions on Communications*, vol. 29, no. 1, pp. 11–18, 1981.

[17] V. M. Malhotra, M. P. Kumar, and S. N. Maheshwari, "An $O(|V|^3)$ algorithm for finding maximum flows in networks," *Information Processing Letters*, vol. 7, no. 6, pp. 277–278, 1978.

[18] I. Stojmenovic, "Position-based routing in ad hoc networks," *IEEE Communications Magazine*, vol. 40, no. 7, pp. 128–134, 2002.

[19] R. Kleinberg, "Geographic routing using hyperbolic space," in *Proceedings of IEEE INFOCOM*, 2007, pp. 1902–1909.

[20] R. Sarkar, X. Yin, J. Gao, F. Luo, and X. D. Gu, "Greedy routing with guaranteed delivery using Ricci flows," in *Proceedings of ACM/IEEE IPSN*, 2009, pp. 121–132.

[21] J. Wu and Y. Wang, "Hypercube-based multipath social feature routing in human contact networks," *IEEE Transactions on Computers*, vol. 63, no. 2, pp. 383–396, 2014.

[22] J. Wu and F. Dai, "A generic distributed broadcast scheme in ad hoc wireless networks," in *Proceedings of IEEE ICDCS*, 2003, pp. 460–467.

[23] J. Kleinberg, "Hubs, authorities, and communities," *ACM computing surveys (CSUR)*, vol. 31, no. 4es, p. 5, 1999.

[24] B. Charron-Bost, M. Függer, J. L. Welch, and J. Widder, "Time complexity of link reversal routing," *ACM Transactions on Algorithms*, vol. 11, no. 3, p. 18, 2015.

[25] F. Fich and E. Ruppert, "Hundreds of impossibility results for distributed computing," *Distributed Computing*, vol. 16, no. 2-3, pp. 121–163, 2003.

[26] J. Suomela, "Survey of local algorithms," *ACM Computing Surveys*, vol. 45, no. 2, p. 24, 2013.

[27] F. Kuhn, T. Moscibroda, and R. Wattenhofer, "The price of being near-sighted," in *Proceedings of ACM-SIAM SODA*, 2006, pp. 980–989.

[28] D. A. Spielman and S.-H. Teng, "Smoothed analysis: an attempt to explain the behavior of algorithms in practice," *Communications of the ACM*, vol. 52, no. 10, pp. 76–84, 2009.

[29] J. Wu and F. Dai, "Mobility control and its applications in mobile ad hoc networks," *IEEE Network*, vol. 18, no. 4, pp. 30–35, 2004.

[30] K. Censor-Hillel, E. Haramaty, and Z. Karnin, "Optimal dynamic distributed MIS," in *Proceedings of ACM PODC*, 2016, pp. 217–226.

[31] S. Vissicchio, O. Tilmans, L. Vanbever, and J. Rexford, "Central control over distributed routing," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4, pp. 43–56, 2015.

[32] J. Wu, "Safety levels-an efficient mechanism for achieving reliable broadcasting in hypercubes," *IEEE Transactions on Computers*, vol. 44, no. 5, pp. 702–706, 1995.