Inferring Partial Orders of Nodes for Hierarchical Network Layout

Hsiang-Yun Wu, Keio University, Yokohama, Japan
Shigeo Takahashi, University of Aizu, Aizu-Wakamatsu, Japan
Hiroko Nakamura Miyamura, Japan Atomic Energy Agency, Kashiwa, Japan
Satoshi Ohzahata, The University of Electro-Communications, Tokyo, Japan
Akihiro Nakao, The University of Tokyo, Tokyo, Japan

Abstract

Extracting hierarchical structures from networks provides us with an effective means of visualizing them, especially when they contain complicated node connectivities such as those in traffic and distributed networks. Although many techniques have been developed for such purposes, they often deterministically break unwanted cycles that may arise from inconsistencies in the network hierarchies, and thus never seek the best compromise among possible partial orders of nodes inherent in the cycle. This paper presents an algorithm for inferring such partial orders by optimizing the network hierarchies along flow paths that are given as input. Our idea is to extract network hierarchies from round-trip paths as well as one-way ones by deriving reasonably consistent multi-layered structures even from possibly inconsistent flow data over the networks. This problem is formulated as mixed-integer programming where we incorporate additional constraints into fundamental layout criteria according to the type and/or expected use of the network. For better visual readability of the network layout, the nodes in individual layers are clustered and reordered for minimizing edge crossings, which is followed by

fine adjustment of intervals between neighboring nodes. We study several network examples to demonstrate the feasibility of the proposed approach including course dependency charts, railway networks, and peer-to-peer (P2P) networks.

1 Introduction

Real world networks consist of a large number of entities that have complicated relationships in between. These network structures are usually dominated by their interior relationships that include several dependency information such as feeding relationships in ecosystem, task ordering for management, and visiting orders of network nodes in the Internet. Visualizing such dependency relationships allows us to understand the overall trends of the internal communication flows inherent in the network. In practice, this can be accomplished by laying out the entities as network nodes along the communication flows, which amounts to visually elucidating the underlying hierarchy of the network. Directed graphs are commonly employed for this purpose since relationships between the nodes can be easily represented as directed edges in the graphs. Hierarchical representation of directed graphs also alleviates visual clutter problems we often encounter when visualizing complicated networks.

However, conventional techniques primarily focus on hierarchical layout of directed acyclic graphs (DAGs) and often suffer from hair-ball effects as the number of paths increases (Figure 1(a)). This is because they cannot effectively identify hierarchical structures inherent in general directed graphs due to unwanted cycles arising from deadlocks in the dependency relationships. In practice, most techniques try to cut edges until all the cycles are removed from the network, while this forces us to ignore possibly significant influence of eliminated edges when inferring the underlying hierarchical layout. Furthermore, especially in distributed networks, we are more interested in data traffic along a path between a specific pair of nodes for understanding the overall trend of the network communication. Incorporating such information about

communication paths over the network will provide us with more options for laying out the network in a hierarchical fashion.

These factors are indeed significant when we try to understand the hierarchical structure in the Internet. Here, the Internet consists of autonomous systems (ASs), which are defined as groups of IP networks maintained under the same administrative control. This implies that data transmission paths between a pair of ASs usually go up and down in the Internet hierarchy, often by way of the topmost network layer (such as the Tier 1 layer), and thus investigating such an AS hierarchy gives us a hint on the structure of specific sub-networks such as peer-to-peer (P2P) networks [20] (Figure 1(b)).

In this paper, we present an algorithm for inferring such hierarchical structures from available network flows through constrained optimization. Our idea is to construct reasonably consistent network hierarchy by referring to the ordering of nodes along all the communication paths, regardless of whether the associated paths are consistent or not. This is accomplished by formulating the problem as mixed-integer programming, where we incorporate several fundamental design criteria as hard and soft constraints. As a post-process, the visual readability of the hierarchical layout is improved by rearranging nodes in the same hierarchical layer to minimize edge crossings and overlaps.

The remainder of this paper is structured as follows: In the next section, we briefly summarize related work. We then present an overview of our algorithm for laying out networks in a hierarchical fashion. In practice, the proposed algorithm consists of the extraction of network hierarchies from both one-way and round-trip communication paths, while nodes in the individual layers are further grouped and rearranged both for reducing visual clutter and accelerating the computation. Several experimental results are also presented to demonstrate the feasibility of our formulation, which are followed by the conclusion of this paper together with future work.

2 Related Work

Visualizing networks is one of the promising approach for mining them for their specific characteristics [1]. Among them, real world networks are often ubiquitous and have long been thought of as hierarchical in nature [34]. This can be easily imagined because networks often exhibit clustered organization, as suggested by many recent studies [22, 27, 37]. Nonetheless, the hierarchical structure implicitly embedded in the network is usually more informative than the clustered organization of network nodes, especially when the network contains cycles that incur deadlocks in the dependency relationships between them. For example, in the Internet, ASs are considered as groups of routers that have the same routing policy. As described in Gao et al. [20], these routing policies are commonly constrained by the commercial strategies between administrative domains, and the order of visiting nodes along each communication path helps to infer AS dependency relationships in the Internet. Despite this, we sometimes cannot uniquely fix the dependency between some pairs of ASs especially when they are in the same network layer. It thus gives us a strong motivation to extract reasonably consistent hierarchical structure even from such possibly insufficient and inconsistent communication data, so that we can maximally clarify the underlying trend in data communication over the multi-layered network.

For visualizing network structures in a hierarchical fashion, directed graphs with layered drawing styles are commonly used to reveal the node dependency. According to the survey of hierarchical drawing algorithms [24], the techniques can be categorized into two main groups: one is for *directed acyclic graphs* (DAGs) and the other is for *general directed graphs*. The pioneering and most popular method for drawing DAGs is *Sugiyama's framework* [33], which distributes the network nodes to each layer by incorporating several criteria such as uniform edge orientation. Various extensions have been introduced to improve this framework, which include edge crossing minimization by using confluent drawings [16] and minimal insertion of dummy nodes [29] for better

network layouts. An accelerated version of Sugiyama's framework was developed by Eiglsperger et al. [15], where they kept the number of dummy nodes and edges linear in the size of the graph without increasing the number of crossings. Di Battista et al. [11] proposed an experimental study for comparing the performance of Sugiyama's layer-based layout and the grid-based layout [9], and showed trade-offs between the two layout aesthetics and computation times. Afterward, constrained versions of DAG drawing algorithms have been devised, so that we can control the number of layers and that of nodes at each layer in the network layout [5, 30].

Although drawing typical directed graphs as well as constrained DAGs is NP-hard, several heuristic algorithms have been proposed to tackle the problem. The most intuitive way is to break the cycles by cutting less important edges and then apply conventional DAG layout algorithms [19]. Selecting edges to be removed is commonly known as a *feedback arc set (FAS)* problem [10] and has been intensively studied so far. In practice, this edge removal strategy allows us to identify the backbone structure of the network as a tree, and then employ the depth of each node in the tree as its layer number in the network hierarchy [19, 24]. Another idea is to distribute the network nodes to a set of concentric circles according to its layer number [32, 2]. This concentric layout style can be further sophisticated by taking into account the distinguished cycles, which often appear as recurring processes in biosciences for example [4, 3]. Of course, 3D versions of hierarchical drawing techniques are also helpful for alleviating visual clutter in the network layout, by taking advantage of depth perception [21, 25].

Rearranging the nodes at the respective layers for aesthetic network layouts will also become an crucial issue especially when we have to handle a large number of network nodes and edges. Gansner et al. [19] and Gange et al. [18] devised techniques for minimizing edge crossings while drawing every flow path within the limited screen space will cause severe visual clutter. For example, Dwyer et al.[14] presented a concept of *power graph* for alleviating this problem, where they grouped a set of nodes into

a cluster in order to reduce distracting visual clutter in networks of general type. Onoue et al. [31], on the other hand, proposes edge concentration techniques for contracting edge connectivities to enhance the associated network readability.

As shown in the survey papers by Nettleton [28], conventional graph mining techniques focus on the analysis of shortest paths among topological structures of networks. In our approach, instead of such simple pairwise relationships, we take observed path information between end nodes as input, on the assumption that most paths go up and down consistently with respect to the hierarchical levels. Unlike conventional energy-based optimizations with soft constraints only [6, 12, 13, 7], we employed mixed-integer programming formulation to incorporate additional hard constraints in addition to soft constraints for various application purposes. Several relevant approaches are done [19, 26, 18] in that they also introduced mixed-integer programming, while it was employed not for inferring optimal network hierarchies but for optimizing the placements of network nodes at given hierarchical levels. On the other hand, our approach incorporates the mixed-integer programming in order to infer the consistent network hierarchy, and also allows us to adaptively design the hierarchical layout according to the type and/or expected use of the target networks, by introducing additional constraints and costs to our formulation. We will further optimize horizontal ordering of nodes at the respective hierarchical layers for better readability of the network layout.

3 Overview

This section presents an overview of our approach to visualizing network hierarchies by taking as input dependency paths over the networks. A typical case of such network paths corresponds to *one-way* dependency paths that are expected to ascend or descend monotonously in a single direction through the hierarchical layers, as shown in Figure 2(a). Examples of this one-way type include predator-prey relationships in ecosystems, course dependency in school curricula, and task orderings in decision graphs.

On the other hand, we also encounter a different type of dependency relationship especially in networks having a relatively small number of hub nodes. In this case, a route is likely to compose a mountain like path as shown in Figure 2(b), since it often has to travel between two end nodes by way of several hub nodes at upper hierarchical layers. Railway transfer and flight plan routes are examples of such *round-trip* paths as well as traffic routes in the Internet. More details of the formulation will be addressed in what follows.

Figure 3 represents the overall flow of the algorithm, which primarily consists of two stages. On the left of the figure, a conventional force-directed layout of the network is exhibited. In the first stage, we extract network hierarchies by inferring optimal partial orders of nodes from given dependency paths as shown in the middle. After that, we reorder the sequence of nodes at each hierarchical layer to minimize edge crossings and overlaps for better readability as shown on the right. Technical details of these two stages will be covered in the next two sections.

4 Inferring Network Hierarchies

In this section, we first tackle one-way paths to extract consistent network hierarchies, and then extend the idea to handle round-trip paths.

4.1 Extracting hierarchies from one-way paths

Here, we formulate one-way hierarchy extraction problems as the mixed-integer programming, which helps us visualize the network structure by arranging its nodes in a single hierarchical order. In our formulation, we employed three criteria, each of which will be detailed below.

Consistent hierarchical ordering Suppose that two nodes p_i and p_{i+1} appear in this order along a one-way dependency path. A natural idea is to place these nodes in a way

that p_{i+1} is higher than p_i in the network hierarchy. This is achieved by introducing the following constraint:

$$l(p_i) - l(p_{i+1}) \geq 1, \tag{1}$$

where $l(p_i)$ represents a positive integer that corresponds to the layer ID of node p_i, and the constant 1 on the right side corresponds to the minimal difference in the layer ID between the neighboring nodes along the path. Here, we assume that a higher hierarchical level has a smaller layer ID, as shown in Figure 4. This formulation allows us to infer a consistent ordering of nodes along each path with respect to the hierarchical level.

Nonetheless, it is still possible that we have another path that is inconsistent with the dependency relationships of the existing paths. Figure 4(b) shows such a case where the node ordering in the new path H-D-F-A, which is drawn by blue dotted arrows, conflicts with that of the previous path H-F-D-A. Actually, incorporating such an inconsistent path may let us find cycles in the dependency network. To make this unexpected case still feasible in our framework, we introduce an integer penalty value $v_{i,i+1}$ into Eq. (1) as:

$$l(p_i) - l(p_{i+1}) \geq 1 - v_{i,i+1}. \tag{2}$$

Eq. (2) implies that we can place p_i and p_{i+1} at the same hierarchical layer or reverse the order of these two nodes by increasing the penalty value $v_{i,i+1}$. However, we may still want to keep the monotonicity of the node ordering with respect to the hierarchical level, even in the worst case, by equalizing the layer IDs of neighboring nodes. This leads to the idea of restricting the upper limit of $v_{i,i+1}$ to 1 to a certain extent, and then making the penalty value $v_{i,i+1}$ optionally increase. This is accomplished by

decomposing $v_{i,i+1}$ in Eq. (2) into two penalty integer values $\mu_{i,i+1}$ and $\delta_{i,i+1}$ as:

$$\begin{aligned} l(p_i) - l(p_{i+1}) \quad &\geq \quad 1 - \mu_{i,i+1} - \delta_{i,i+1}, \\ \text{where} \quad 0 \leq \mu_{i,i+1} \leq 1 \quad &\text{and} \quad 0 \leq \delta_{i,i+1}. \end{aligned} \tag{3}$$

As described earlier, setting the penalty value $\mu_{i,i+1}$ to 1 amounts to placing p_i and p_{i+1} in the same hierarchical layer as shown in Figure 4(c), and further increasing the value $\delta_{i,i+1}$ will reverse the original dependency ordering between these two nodes. Thus, when defining the objective function to be minimized, we assign a small weight value to $\mu_{i,i+1}$ for readily equalizing the hierarchical levels of the two nodes, while we employ a large weight value for $\delta_{i,i+1}$ to further penalize for reversing their dependency relationship. Here, we minimize the objective function E, which is defined as a weighted sum of penalty values $\mu_{i,i+1}$ and $\delta_{i,i+1}$:

$$\begin{aligned} &E = w_e C_e + w_r C_r, \quad \text{where} \\ &C_e = \sum_k \sum_{i,i+1 \in P_k} \mu_{i,i+1}, \quad \text{and} \quad C_r = \sum_k \sum_{i,i+1 \in P_k} \delta_{i,i+1}. \end{aligned} \tag{4}$$

Here, w_e and w_r ($w_e \leq w_r$) are the weights for $\mu_{i,i+1}$ and $\delta_{i,i+1}$, respectively, and P_k is the set of node indices along the k-th path.

Note that, in our formulation, we can adhere to equalizing the layers of two nodes in conflict by raising the relative ratio $s = w_r/w_e$ (Figure 4(c)). On the other hand, we can lower the ratio in order to respect the predominant dependency relationship between them (Figure 4(b)). Suppose that we have m directed edges from p_i to p_{i+1} and n edges for its reverse. The cost E of Eq. (4) will be:

$$E = \begin{cases} (1+s)w_e n & \text{if } l(p_i) > l(p_{i+1}) \\ w_e(m+n) & \text{if } l(p_i) = l(p_{i+1}) \\ (1+s)w_e m & \text{if } l(p_i) < l(p_{i+1}). \end{cases} \tag{5}$$

This suggests that $l(p_i) > l(p_{i+1})$ when $m > sn$, $l(p_i) < l(p_{i+1})$ when $m < n/s$, and

$l(p_i) = l(p_{i+1})$ otherwise. We set $s = 1.5$ by default to keep a fair balance among the three cases.

Minimal difference in layer On the other hand, we also want to remove redundant layers in the hierarchical layout of the network, by maximally reducing the difference in the layer ID between each pair of neighboring nodes along the path. Indeed, this allows us to keep the network hierarchy as compact as possible. Toward this goal, we introduce another integer value $\lambda_{i,i+1}$ to represent the absolute difference between a pair of successive nodes as follows:

$$\begin{aligned} l(p_i) - l(p_{i+1}) &\leq 1 + \lambda_{i,i+1}, \\ l(p_i) - l(p_{i+1}) &\geq -1 - \lambda_{i,i+1}, \quad \text{and} \\ \lambda_{i,i+1} &\geq 0. \end{aligned} \tag{6}$$

Note that, in this formulation, we allow the minimal absolute difference 1 from the beginning, since we want to avoid any conflict between the penalty values. We can minimize the total number of hierarchical layers by summing up the values $\lambda_{i,i+1}$ to the new cost term:

$$C_d = \sum_k \sum_{i,i+1 \in P_k} \lambda_{i,i+1}. \tag{7}$$

This means that, in the case of one-way paths, the final version of the objective function E can be defined as

$$E = w_e C_e + w_r C_r + w_d C_d, \tag{8}$$

where w_d is a weight value assigned to the term C_d in Eq. (7).

Number of hierarchical layers Sometimes, we are forced to excessively suppress the number of hierarchical layers in designing the network layouts, especially when we have to fit the entire network layout to the space of predefined size. In our formulation,

we can explicitly set the limit number of layers to N, by restricting the layer ID of each node p_i as

$$0 \leq l(p_i) \leq N-1. \tag{9}$$

4.2 Extracting hierarchies from round-trip paths

Inferring network hierarchies from round-trip network paths is more involved because this amounts to identifying the hub node at the topmost layer along each communication path. In the remainder of this section, we formulate the constraints for the round-trip paths by extending the previous formulation for the one-way case.

Valley-free paths As pointed out by several literature, the routing policy under Border Gateway Protocol (BGP) is usually *valley-free* [20, 34, 36]. Figure 5(a) shows an example of such a valley-free round-trip path G-E-B-A-C, which has only one turning point at A with respect to the hierarchical level. For example, in the case of distributed networks, nodes G and C can be considered as end user nodes, while other intermediate nodes correspond to routers provided by Internet service providers (ISPs). The aforementioned literature suggests that we can encode all communication paths in this case as valley-free paths when inferring the hierarchical structure of distributed networks.

Recall that Eq. (3) corresponds to the constraint that node p_{i+1} is higher than node p_i in the network hierarchy along an ascending path. If two nodes p_i and p_{i+1} are on a descending path instead, we can write the corresponding constraint as:

$$l(p_i) - l(p_{i+1}) \leq -1 + \mu_{i,i+1} + \delta_{i,i+1}. \tag{10}$$

However, along a valley-free round-trip path, we have to switch the constraint from Eq. (3) to Eq. (10) when we pass through the turning point, such as node A in Figure 5(a) for example. This requires us to encode the orientation of each edge along the path with respect to the hierarchical level. In our approach, we employ two binary

values $\alpha_{i,i+1}$ and $\beta_{i,i+1}$, where $\alpha_{i,i+1}$ becomes 1 if the corresponding edge $\overline{p_i p_{i+1}}$ is ascending while $\beta_{i,i+1}$ is 1 if the edge is descending. Introducing these two binary values allows us to rewrite Eqs. (3) and (10) as:

$$\begin{aligned} l(p_i) - l(p_{i+1}) &\geq \alpha_{i,i+1} - M(1-\alpha_{i,i+1}) - \mu_{i,i+1} - \delta_{i,i+1} \\ l(p_i) - l(p_{i+1}) &\leq -\beta_{i,i+1} + M(1-\beta_{i,i+1}) + \mu_{i,i+1} + \delta_{i,i+1}, \end{aligned} \tag{11}$$

where M is some large value and used to validate the two inequality constraints when $\alpha_{i,i+1}$ and $\beta_{i,i+1}$ vanish. Since the edge $\overline{p_i p_{i+1}}$ is either ascending or descending,

$$\alpha_{i,i+1} + \beta_{i,i+1} = 1. \tag{12}$$

Now we are ready to formulate the criterion for valley-free paths as a constraint, which can be written as:

$$\begin{aligned} &\sum_{i=0}^{N_k-2} \sum_{i,i+1,i+2 \in P_k} (\alpha_{i,i+1} \oplus \alpha_{i+1,i+2}) = 1, \\ &\alpha_{0,1} = 1, \qquad \beta_{N_k-2,N_k-1} = 1, \end{aligned} \tag{13}$$

where $\alpha_{i,i+1} \oplus \alpha_{i+1,i+2}$ is defined as an "XOR" operation on $\alpha_{i,i+1}$ and $\alpha_{i+1,i+2}$, and N_k represents the number of nodes on the k-th path P_k. Clearly, the above equation ensures that each path has only one turning point between ascending and descending phases. Figure 5(b) shows an example of the k-th path, where $\alpha_{2,3} \oplus \alpha_{3,4} = 1$ in this case. Note that we set the first edge of P_k as ascending by $\alpha_{0,1} = 1$, and the last edge as descending by $\beta_{N_k-2,N_k-1} = 1$.

5 Drawing Nodes at Each Hierarchical Layer

After having distributed the network nodes to the hierarchical layers, our next task is to seek better ordering of the nodes at each individual layer to maximally reduce the visual clutter arising from edge crossings and overlaps as shown in the middle of

Figure 3. We carry out this layout enhancement as a post-process, by taking advantages of conventional algorithms developed by Gange et al. [18] and Gansner et al. [19].

5.1 Grouping and Rearranging Nodes

In this approach, we incorporated a new step for grouping the network nodes, which allows us to accelerate the layout computation by reducing the number of primitive components in the network as well as to improve visual readability by concentrating edges between these groups. This means that the new step lets us optimize the horizontal orders of node clusters at each layer first, then rearrange those of nodes inside the respective clusters, and finally conduct fine adjustment of space between every pair of adjacent nodes, This visualization pipeline effectively provides us with better readability of the network layout, especially when the network has a large amount of nodes. The overall optimization process consists of three steps as described below.

Inserting dummy nodes After having inferred partial orders of network nodes, it is possible that two nodes are not immediate neighbors in the network hierarchy, which means that the corresponding edge inevitably passes across one or more intermediate layers in between. In this case, we insert dummy nodes as intersections of the edge with the intermediate layers to guide the shape of that edge in the final layout. Now the edge is replaced by an alternating sequence of intermediate dummy nodes and edges, which is bounded by original two end nodes. The corresponding process is illustrated in Figures 6(a) and (b).

Grouping nodes into a set of clusters In the second step, we consider nodes sharing common neighbors (including dummy nodes) as similar, and group them into a set of clusters. Indeed, this process successfully reduces the complexity of network connectivity and thus leads to accelerate optimization of node ordering at the respective layers. For grouping the network nodes at each hierarchical layer, we introduced

the conventional Jaccard coefficient as the similarity for evaluating every pair of nodes at the same layer [8], which is defined as $\mathrm{J}(A,B) = |A \cap B| / |A \cup B|$. Here, A and B represent the sets of network nodes at the neighboring layers that are incident to the two nodes, respectively. For grouping nodes at each hierarchical layer, we conduct hierarchical clustering based on the Jaccard coefficient [38], which selectively groups the most similar pair of nodes one by one until the total number of node clusters is no larger than the predefined number c, where $c = 10$ by default. Figure 6(c) shows such a case where the nodes are grouped into three clusters, which are drawn in red, green, and blue, respectively. Note that we can adjust the number of clusters at each layer by interactively controlling the similarity threshold (indicated in green) over the dendrogram that represents the hierarchical clustering of nodes. Figure 7 shows such an interface where dendrograms (from left to right) corresponds to the hierarchical clustering of nodes at the respective hierarchical layers (from top to bottom) in Figure 6.

Rearranging Horizontal Ordering of Nodes The third step is to rearrange the order of node clusters (including single nodes) at each hierarchical layer, so that we can minimize the edge crossings between neighboring layers and edge overlaps within the same layer. For this purpose, we employ the mixed-integer programming formulation of Gange et al. [18], while we improved the associated algorithm by introducing a two-step optimization approach. Figures 6(d) and (e) provide steps of this process, where we first rearrange the ordering of node clusters, and then optimize the order of nodes inside each cluster, respectively. Note that each optimization step reduces distracting edge crossings.

Adjusting horizontal spacing between nodes Once the overall order of the nodes has been fixed, we minimize a horizontal displacement between end nodes for each edge if it bridges neighboring layers. Here, we incorporated another mixed-integer programming formulation by Gansner et al. [19], while we again assigned different

weights to the edges according to the types of their end nodes. More specifically, we assigned relatively larger weights for edges having dummy end nodes so that we can strictly penalize the horizontal displacements of those edges. This effectively allows us to align edges that bridge two dummy nodes as vertical as possible while minimizing their intersections with other edges of the same type, which helps us enhance our rendering style as described later. Additionally, so as to allow users to effectively explore a specific path in the networks, we newly incorporated constraints that keep the selected path, which is colored in red red, as straight as possible, by minimizing horizontal displacements of edges along the path as shown in Figure 6(f).

5.2 Edge Concentration based on Node Clusters

Grouping nodes into clusters at each layer permits us to enhance the visual readability of the network by reducing the number of *node primitives* such as single nodes and node clusters, while it also incurs a different type of visual clutter arising from high degree of such node primitives. Thus, we newly invented rendering styles for the node clusters and their associated edges using edge concentration techniques (e.g. [31]).

Rendering node clusters As described earlier, we apply hierarchical clustering to the nodes at each layer by taking advantage of the Jaccard coefficient, so as to group similar nodes into clusters. We reflect the node clustering results into our representation of the hierarchical network. Suppose we have a hierarchical network as shown in Figure 8(a). In this case, we actually draw each node cluster as a region that encloses the associated nodes as shown in Figure 8(b). Recall that the number of node clusters at each layer can be adjusted by vertically sliding the similarity threshold bar with our interface as shown on the left of Figure 8(b).

Rendering concentrated edges Once we enclosed node clusters by green regions, we draw each traversal path passing through the node primitives. However, in this

process, we just draw a single edge if two node primitives at adjacent levels share multiple edges. Figure 8(c) shows such an example where multiple edges between two node primitives are concentrated in a single edge. Our aforementioned clustering policy decomposes the edge crossing problem into cluster ordering problems together with node ordering problems. Based on this strategy, we can effectively reduce the number of edges and crossings by further incorporating the edge concentration technique [31]. In our implementation, a greedy algorithm has been incorporated for finding complete bipartite subgraphs, i.e., *biclique covers*, in order to maximally reduce the number of edges in the hierarchical network layout. You can find an example of a biclique within a red rectangle in Figure 8((coffee)), which will be bundled by the cross mark icon node around the midpoints of the edges as shown in Figure 8((drink)). Note that this edge concentration process is performed independently for each pair of two adjacent layers.

Although this network representation maximally avoids visual clutter arising from the edge crossings and overlaps, it is too simplified to explore the details of the associated network flows. In our implementation, we also prepared an option to associate a visual metaphor that offers a symbolic representation on how each node in the cluster is connected with adjacent node clusters, as shown in yellow in Figure 8(e). In practice, in Figure 8(e)), we draw red edges from the nodes G, H, and I to the right yellow node at the top of the cluster, since they originally have connections with the right cluster (containing L, F, and M) at the upper layer, while only one edge to the left yellow node from G only because H and I has no connections with the left cluster (containing N and E) at the upper layer.

6 Results and Discussion

This section provides implementation details, experimental results, and discussion on limitations of this approach.

6.1 Implementation details

For demonstrating the feasibility of our approach, we have developed a visualization system that infers the underlying consistent hierarchies of the given network data. Note that our system has been implemented on a desktop PC with Quad-Core Intel Xeon CPUs (3.7GHz, 10MB cache) and 12GB RAM, and the source code was written in C++ using OpenGL for drawing network hierarchies, OpenCV for handling images, IBM ILOG CPLEX for solving MIP optimization problems. To explicitly clarify the direction of network flow, we followed the design principles collected by Ware [35], where we incorporated the gradient color from yellow to purple for guiding the flow direction. Moreover, we assigned different depth values to the network edges according to the types of their end nodes, so that we can effectively hide edges associated with dummy nodes behind other types of edges. We also introduced halo effects [17] to our rendering style for further visual clarity, and maximally aligned dummy edges as vertical as possible to reduce their visual conflict with other types of edges spanned by regular end nodes.

6.2 Application examples

Here we demonstrate three application examples, course dependency in a school curriculum, accessibility zones of metro networks, and communications over peer-to-peer (P2P) networks. The first case illustrates how we can visualize network hierarchies by investigating one-way ordering paths, while the last two cases are more involved in the sense that we have to infer the network hierarchies from round-trip paths on the assumption that the majority of the paths are valley-free. Note that we set $w_e = 100$, $w_r = 150$, and $w_d = 1$ in Eq. (8) and choose the number of layers N to be sufficiently large unless specifically stated otherwise. Table 1 shows computation times for optimizing the network layouts, where we employed node clustering only in the last case on P2P networks.

Course dependency in school curriculum Dependency relationships among courses in the university curriculum lead to a typical network example, from which we can infer consistent hierarchical structures using the proposed approach. Suppose that we are currently designing the hierarchy of courses in a computer science 6-semester curriculum, where we expect to take as input a set of students' course completion records. Figure 9(a) shows the network of course nodes arranged using the conventional force-directed algorithm, while from this layout, unfortunately, we cannot retrieve any meaningful structures hidden behind the dependency among courses intuitively. By taking advantage of our formulations described in Section 4.1, we can design the hierarchy of courses. We first employ the weight assignment $w_e = 100$, $w_r = 100$, and $w_d = 1$ to design the curriculum as exhibited in Figure 9(b). By increasing w_r to 150, we can allow more courses to be included in the individual semesters (i.e., semesters) as given Figure 9(c).

We can also maximally equalize the number of course credits students can take in each semester. Let us denote the number of credits approved for the j-th course by s_j. We also define the binary value $\tau_{i,j}$ so that $\tau_{i,j} = 1$ if the j-th course is scheduled at the i-th hierarchical level; otherwise $\tau_{i,j} = 0$. This setup makes it possible to count the total number of credits available for the i-th hierarchical level as S_i, which can be given by:

$$S_i = \sum_j s_j \tau_{i,j}, \quad \sum_i^{N-1} \tau_{i,j} = 1, \text{ and } \sum_i^{N-1} i \cdot \tau_{i,j} = l(p_j). \tag{14}$$

If we constrain the number of credits so that it decreases as the hierarchical level goes higher by applying $S_{i+1} - S_i \geq 0 \quad (0 \leq i \leq N-2)$, we can obtain the revised layout of the courses as demonstrated in Figure 9(d), where the sums of credits are 14, 16, 17, 17, 17, and 17 as the layer ID increases (from top to bottom).

Accessibility zones of metro networks As the first application example of handling round-trip paths, we employed metro lines running in the Tokyo area and tried to infer the underlying accessibility zones from the city center. This is made possible by assuming that every metro line in Tokyo leaves a station in the suburb, runs throughout the center of Tokyo, and finally reaches another station in the suburb. Figure 10(b) shows the Tokyo metro map, where 225 stations are connected by 13 metro lines.

In our experiment, we tried to infer the accessibility zones of stations by taking as input all the 13 metro lines as round-trip paths. Figure 10(a) presents an optimized layout where we finally obtained 28 hierarchical layers due to the effect of Eq. (7), where we employed the default setting of weights $w_e = 100$, $w_r = 150$, and $w_d = 1$ in Eq. (8) to discriminate between station zones. Note that nodes annotated with "T" correspond to interchange stations at which two or more metro lines meet. We also rendered the background of the Tokyo metro map in Figure 10(b) by classifying the inferred layer IDs into six groups, in order to visualize accessibility zones from the city center. Here, we assign a different color to the Voronoi cell of each station according to its corresponding zone, where brighter colors are assigned to stations closer to the center.

We asked an expert in cartography to evaluate this visualization result. He found this zone partitioning interesting in that we can intuitively identify hub stations in the metro network. He also suggested that the result will serve as a basis for planning new public transportation service, and can be further augmented by incorporating statistics on the numbers of passengers along the lines. Figure 10(c) shows the result with our visualization with node clusters and contracted edges, and allows users to trace the paths more easily.

Peer-to-peer (P2P) networks As the last application example, we tackle the visualization of P2P networks by inferring the underlying hierarchical layers from communication paths, which was obtained through the measurement method in [39]. In

this example, we assume that almost all the available communication routes are non-valley round-trip paths [20, 34, 36], in the sense that each path starts from some end node to ascend through the AS hierarchy and then descend to reach another end node, while having a single turning point with respect to the hierarchical layer. Figure 1(b) exemplifies an optimal layout of a relatively small-sized P2P network from round-trip communication paths. Note that, in this figure, we always annotate end nodes by icons of computer monitors, while other intermediate nodes by those of server PCs.

In the same way, we can infer the hierarchical structure of another medium-sized P2P network as shown in Figure 11(a). Here, we try to sophisticate the network layout by seeking suggestions from domain experts. First, we tried to adjust the overall shape of the network hierarchy to look like a pyramid. This can be accomplished by penalizing the network nodes at the higher layers by incorporating a new cost term as follows:

$$C_l = \sum_{i=0}^{n-1} (N - l(p_i)), \tag{15}$$

where N represents the predefined number of hierarchical layers as described earlier. Domain experts also suggested a common principle that network nodes at higher hierarchical layers are more likely to have a larger number of connections with other nodes. This suggestion enables us to formulate an additional cost term as follows:

$$C_g = \sum_{i=0}^{n-1} deg(p_i) \cdot l(p_i), \tag{16}$$

so that we can penalize nodes having more incident edges if they stay in lower hierarchical layers. Note here that $deg(p_i)$ represents the degree of the i-th node in the P2P network. By employing the new combination of cost terms:

$$E = w_e C_e + w_r C_r + w_d C_d + w_l C_l + w_g C_g, \tag{17}$$

where $w_e = 10^2$, $w_r = 10^4$, $w_d = 1$, $w_l = 10$, and $w_g = 2$, we can rearrange the network layouts as shown in Figure 11(b).

Another visualization example of concetrated a large-sized P2P network is demonstrated in Figure 11(c), where we can still identify global trends of data communication among end nodes as a set of non-valley routes over the network. Figure 11(c) shows our final result with node clusters and contracted edges. We also applied the node ordering algorithm by Gange et al. [18] and Gansner et al. [19] to the same dependency data, and the result is summarized in Table 2.

6.3 Discussions

In the previous subsections, we detailed the algorithm and demonstrated several application results of our approach. In this subsection, we provide side-by-side comparison showing how the choice of path and topological structures will affect the visualization results. To provide some intuition on the relationship between the hierarchy of a graph and its topological connectivity, we compared our inferred hierarchy of the respective node with its accumulated geodesic distances over the graph [23]. Figure 12 provides comparison between the hierarchical levels and accumulated geodesic distance at the respective nodes over the P2P network that is the same as that in Figure 11. Note that the corresponding color legend in the figure indicates the hierarchical layer and accumulated geodesic distance at each node, respectively. The comparison suggests that the nodes at the top hierarchical layer, which are colored in red, are relatively far away from the geodesic center of the network in this specific case of the P2P network. This implies that the dominant nodes in the network hierarchy can be different from those for the ordering of distances from its topological center.

Besides path information, difference in the topology of the network also influences on the inferred hierarchy in the generated visualization. For this purpose, typical network structures were investigated using the proposed approach. Figure 13 demon-

strates the results on several representative topological structures, including a $(16,5)$-banana tree, a small-world network, a scale-free network, and a random network. To generate a set of traversal paths, we randomly select the starting node of the individual path and recursively perform a random walk on non-visited incident edges under certain probability until all edges are covered. Note that these four networks have the same number of nodes for the comparison. Moreover, the color of the network node in the figure ranges from red to purple according to the corresponding hierarchical level, and the number indicates the ID assigned to each node cluster.

In the case of the $(16,5)$-banana tree in Figures 13(a) and (b), top layer nodes, colored in red, are very close to the topological center. Simultaneously, nodes with the same cluster ID are more likely to share their parent at the adjacent layer due to the similarity based on the Jaccard coefficient. Basically, the same can be applied to the small-world network in Figures 13(c) and (d), while more number of conflicts among the paths were resolved since the networks cycles this time, which increases the number of nodes at intermediate layers as a result. The scale-free network in Figures 13(e) and (f), on the other hand, produces more complicated hierarchy where high-degree nodes stay around high layers, while nodes with low degree again shares the same cluster if they are connected to the same adjacent node. Finally, the random graph again produced a complicated hierarchy while it has a relatively large number of nodes at the intermediate layers since it contains more conflicts among traversal paths over the network as shown in Figures 13(g)-(h). Of course, the resulting network hierarchy strongly depends on the given set of parts over the network while the topology of the network also influences on the appearance of the network hierarchy to a certain extent.

The scalability of the proposed approach is primarily limited by the number of dependency relationships between pairs of nodes in the network. A naive approach for clustering the network nodes may alleviate the problem by simplifying the network

complexity, although at the risk of overlooking important local dependencies among network nodes. Another possibility suggested by our experiments is that we first spare a sufficient number of hierarchical layers N in our optimization, and then merge multiple layers into one according to the given requirements later. Indeed, this considerably accelerates the overall computation because we can maximally avoid penalizing inconsistency in the partial ordering of nodes along each path, while a new algorithm for fitting the entire network hierarchy to a smaller number of layers remains to be carefully formulated. Another option is to permit feasible solutions as well as fully optimal ones in the mixed-integer programming optimization, especially when we are forced to optimize a large network layout within a certain period of time. Prior knowledge obtained from domain experts can also allow us to sophisticate the formulations of constraints and objective functions, and thus further accelerate the computation, which is again left as future work.

7 Conclusion

This paper has presented an approach to inferring network hierarchies from available dependency relationships among nodes. Our algorithm can calculate consistent hierarchical structures hidden behind the input network even from possibly inconsistent orders of nodes, both from one-way and round-trip paths. The contribution of this paper lies in the formation of constrained optimization for inferring such consistent hierarchies in the network, by taking advantage of the mixed-integer programming formulation. Visual clutter arising from the network layouts have been further suppressed by rearranging the horizontal order of nodes at each hierarchical layer. We tested our proposed formulations on three application examples, which demonstrated the robustness of our approach against various types of networks.

Incorporating further improvements in terms of scalability is still an important theme for future research. Semantic reasoning of the network hierarchies also helps

us devise new formulations of constraints and objective functions for better clarifying the underlying hierarchies in the network. More aesthetic rendering styles will enhance our visual understanding of hierarchical structures irrespective of the complexity of the network.

Acknowledgments

This work has been partially supported by the Telecommunication Advanced Foundation, MEXT KAKENHI Grant Number 25120014, and JSPS KAKENHI Grant Numbers 16H02825, 26730061, and 15K12032.

References

[1] A. E. Attipoe, J. Yan, C. Turner, and D. Richards. Visualization tools for network security. In *Proceedings of the Visualization and Data Analysis 2016. Part of IS&T Electronic Imaging 2016*, pages VDA–489.1–VDA–489.8, 2016.

[2] C. Bachmaier. A radial adaptation of the sugiyama framework for visualizing hierarchical information. *IEEE Transactions on Visualization and Computer Graphics*, 13(3):583–594, 2007.

[3] C. Bachmaier, F. J. Brandenburg, W. Brunner, and R. Fülöp. Drawing recurrent hierarchies. *Journal of Graph Algorithms and Applications*, 16(2):151–198, 2012.

[4] C. Bachmaier, F. J. Brandenburg, W. Brunner, and G. Lovász. Cyclic leveling of directed graphs. In *Proceedings of the 16th International Conference on Graph Drawing*, volume 5417 of *Springer Lecture Notes in Computer Science*, pages 348–359, 2008.

[5] J. Branke, S. Leppert, M. Middendorf, and P. Eades. Width-restricted layering of acyclic digraphs with consideration of dummy nodes. *Information Processing Letters*, 81(2):59–63, 2002.

[6] L. Carmel, D. Harel, and Y. Koren. Drawing directed graphs using one-dimensional optimization. In *Proceedings of the 8th International Conference on Graph Drawing*, volume 2528 of *Springer Lecture Notes in Computer Science*, pages 193–206, 2001.

[7] L. Carmel, D. Harel, and Y. Koren. Combining hierarchy and energy drawing directed graphs. *IEEE Transactions on Visualization and Computer Graphics*, 10(1):46–57, 2009.

[8] F. Chierichetti, R. Kumar, S. Pandey, and S. Vassilvitskii. Finding the Jaccard median. In *Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 293–311, 2010.

[9] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. Algorithms for drawing graphs: An annotated bibliography. *Computational Geometry: Theory and Applications*, 4(5):235–282, 1994.

[10] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice Hall, 1998.

[11] G. Di Battista, A. Garg, G. Liotta, A. Parise, R. Tamassia, E. Tassinari, F. Vargiu, and L. Vismara. Drawing directed acyclic graphs: An experimental study. *International Journal of Computational Geometry and Applications*, 10:623–648, 2000.

[12] T. Dwyer and Y. Koren. Dig-cola: Directed graph layout through constrained energy minimization. In *IEEE Symposium on Information Visualization*, pages 65–72, 2005.

[13] T. Dwyer, Y. Koren, and K. Marriott. Ipsep-cola: An incremental procedure for separation constraint layout of graphs. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):821–828, 2006.

[14] T. Dwyer, C. Mears, K. Morgan, T. Niven, K. Marriott, and M. Wallace. Improved optimal and approximate power graph compression for clearer visualisation of dense graphs. In *Proceedings of the 7th IEEE Pacific Visualization Symposium (PacificVis 2014)*, pages 105–112, 2014.

[15] M. Eiglsperger, M. Siebenhaller, and M. Kaufmann. An efficient implementation of Sugiyama's algorithm for layered graph drawing. In *Proceedings of the 12th International Conference on Graph Drawing*, volume 3383 of *Springer Lecture Notes in Computer Science*, pages 155–166, 2004.

[16] D. Eppstein, M. T. Goodrich, and J. Y. Meng. Confluent layered drawings. In *Proceedings of the 12th international conference on Graph Drawing*, volume 3383 of *Springer Lecture Notes in Computer Science*, pages 184–194, 2004.

[17] M. Everts, H. Bekker, J. Roerdink, and T. Isenberg. Depth-dependent halos: Illustrative rendering of dense line data. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1299–1306, 2009.

[18] G. Gange, P. J. Stuckey, and K. Marriott. Optimal k-level planarization and crossing minimization. In *Proceedings of the 17th International Conference on Graph Drawing*, Springer Lecture Notes in Computer Science, pages 238–249, 2010.

[19] E. R. Gansner, E. Koutsofios, S. C. North, and K.-P. Vo. A technique for drawing directed graph. *IEEE Transactions on Software Engineering*, 19(3):214–230, 1993.

[20] L. Gao. On inferring autonomous system relationships in the internet. *IEEE/ACM Transactions on Networking*, 9(6):733–745, 2001.

[21] A. Garg and R. Tamassia. GIOTTO3D: A system for visualizing hierarchical structures in 3D. In *Proceedings of the 3th International Conference on Graph Drawing*, volume 1190 of *Springer Lecture Notes in Computer Science*, pages 193–200, 1997.

[22] D. Gibson, J. Kleinberg, and P. Raghavan. Inferring web communities from link topology. In *Proceedings of the 9th ACM Conference on Hypertext and Hypermedia: Links, Objects, Time and Space - Structure in Hypermedia Systems*, pages 225–234, 1998.

[23] F. Harary. *Graph Theory*. Westview Press, 1994.

[24] P. Healy and N. S. Nikolov. Hierarchical drawing algorithms. In R. Tamassia, editor, *Handbook of Graph Drawing and Visualization (Discrete Mathematics and Its Applications)*, chapter 13, pages 409–446. Chapman and Hall/CRC, 2013.

[25] S.-H. Hong, N. S. Nikolov, and A. Tarassov. A 2.5D hierarchical drawing of directed graphs. *Journal of Graph Algorithms and Applications*, 11(2):371–396, 2007.

[26] M. Jünger, E. K. Lee, P. Mutzel, and T. Odenthal. A polyhedral approach to the multi-layer crossing minimization problem. In *Proceedings of the 5th International Conference on Graph Drawing*, Springer Lecture Notes in Computer Science, pages 13–24, 1997.

[27] A. Lancichinetti, S. Fortunato, and J. Kertész. Detecting the overlapping and hierarchical community structure in complex networks. *New Journal of Physics*, 11(3), 2009.

[28] D. F. Nettleton. Data mining of social networks represented as graphs. *Computer Science Review*, 7:1–34, 2013.

[29] N. S. Nikolov and A. Tarassov. Graph layering by promotion of nodes. *Discrete Applied Mathematics*, 154(5):848–860, 2006.

[30] N. S. Nikolov, A. Tarassov, and J. Branke. In search for efficient heuristics for minimum-width graph layering with consideration of dummy nodes. *Journal of Experimental Algorithmics*, 10:1–27, 2005.

[31] Y. Onoue, N. Kukimoto, N. Sakamoto, and K. Koyamada. Minimizing the number of edges via edge concentration in dense layered graphs. *IEEE Transactions on Visualization and Computer Graphics*, 22(6):1652–1661, 2016.

[32] M. G. Reggiani and F. E. Marchetti. A proposed method for representing hierarchies. *IEEE Transactions on Systems, Man and Cybernetics*, 18(1):2–8, 1988.

[33] K. Sugiyama, S. Tagawa, and M. Toda. Methods for visual understanding of hierarchical system structures. *IEEE Transactions on Systems, Man and Cybernetics*, 11(2):109–125, 1981.

[34] H. Tangmunarunkit, R. Govindan, S. Jamin, S. Shenker, and W. Willinger. Network topologies, power laws, and hierarchy. *Computer Communication Review*, 32(1):76, 2002.

[35] C. Ware. *Information Visualization: Perception for Design*. Morgan Kaufmann Publishers, third edition, 2012.

[36] J. Xia and L. Gao. On the evaluation of as relationship inferences. In *Proceedings of the Global Telecommunications Conference*, volume 3, pages 1373–1377, 2004.

[37] B. Yang, J. Di, J. Liu, and D. Liu. Hierarchical community detection with applications to real-world network analysis. *Data and Knowledge Engineering*, 83:20–38, 2013.

[38] G. Yi, H.-Y. Wu, K. Misue, K. Mizuno, and S. Takahashi. Visualizing bag-of-features image categorization using anchored maps. In *Proceedings of the 7th International Symposium on Visual Information Communication and Interaction*, VINCI '14, pages 39:39–39:48, 2014.

[39] M. Yoshida and A. Nakao. Measuring bittorrent swarms beyond reach. In *IEEE International Conference on Peer-to-Peer Computing (P2P2011)*, pages 220–229, 2011.

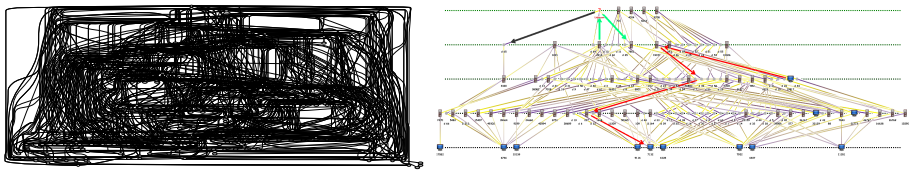
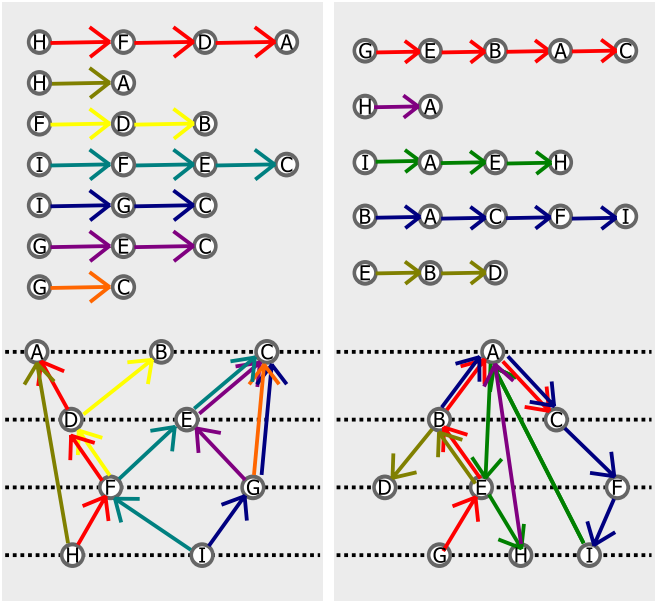


Figure 1: Examples of P2P network representation. (a) Hierarchical layout of the network obtained using the algorithm by Gansner et al. (b) Inferring optimal hierarchies from dependency between nodes of a P2P network extracted from round-trip communication paths.



(a) (b)

Figure 2: Networks with two different types of dependency paths. (a) One-way paths. (b) Round-trip paths. The sequences of nodes on the top represent input dependency paths and network layouts on the bottom show the inferred hierarchical structures of the networks.

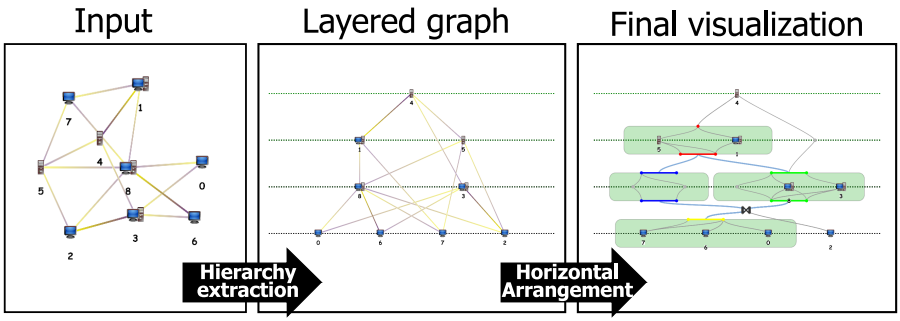
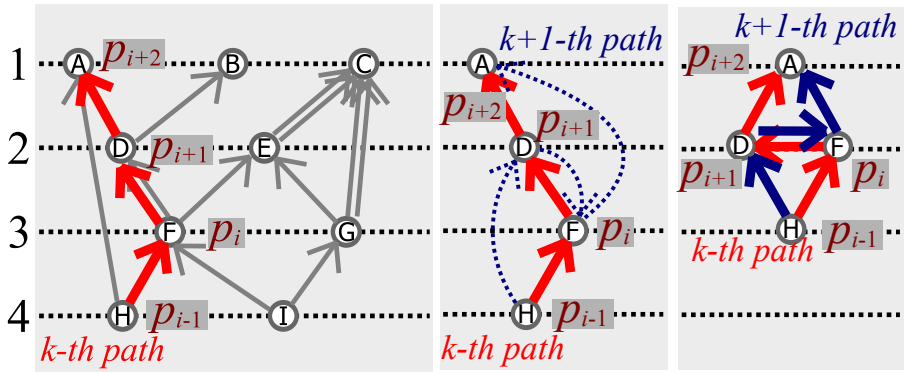


Figure 3: Overview of the present framework.



(a) (b) (c)

Figure 4: Extracting hierarchy from one-way paths. (a) Consistent edge orientations along a path. (b) Inserting an inconsistent path into the network. (c) An optimized layout.

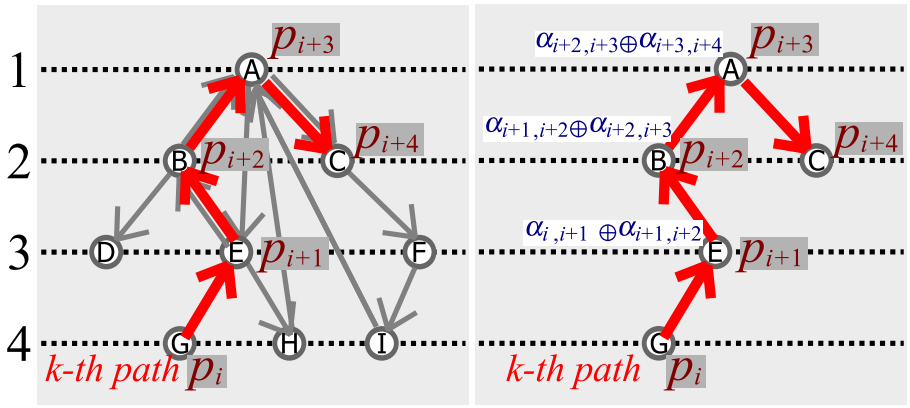


Figure 5: Extracting hierarchy from Round-trip paths. (a) A path with reversed direction at a certain node. (b) Computing α values along the path.

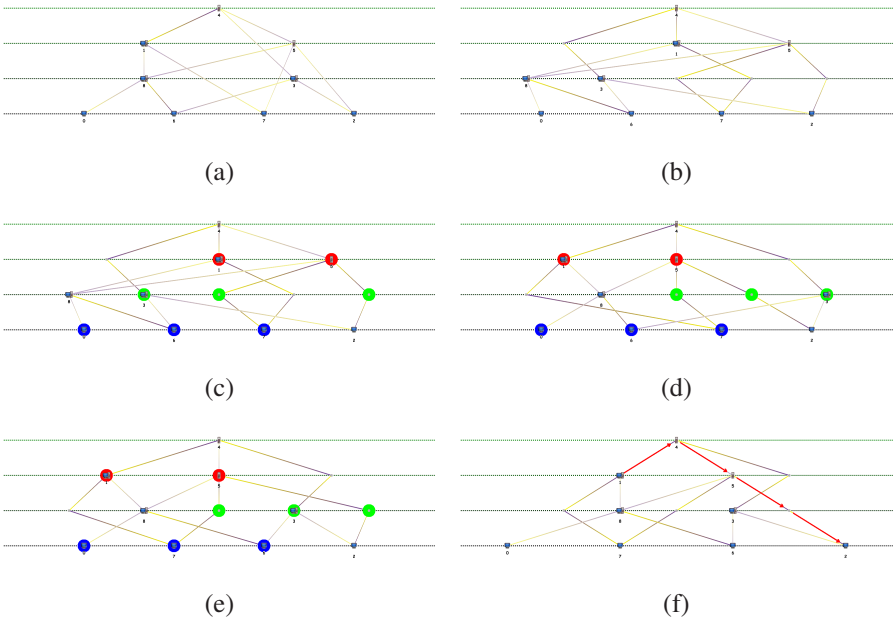


Figure 6: Rearranging nodes at hierarchical layers. (a) Initial network layout. (b) Dummy nodes added. (c) Grouping nodes into clusters at each layer. (d) Node clusters rearranged. (e) Nodes rearranged within each cluster. (f) Space adjustment between neighboring nodes.

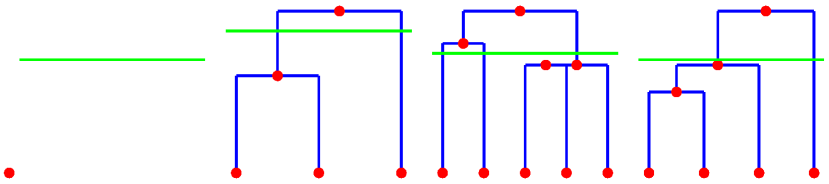


Figure 7: Dendrogram interface that represents the hierarchical clustering of nodes at the respective layers in Figure 6.

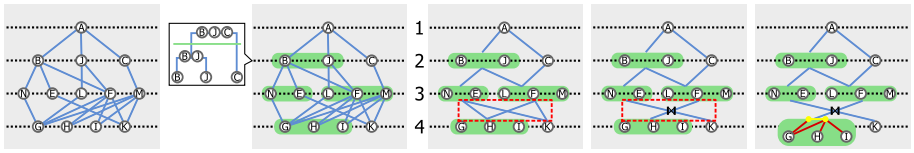
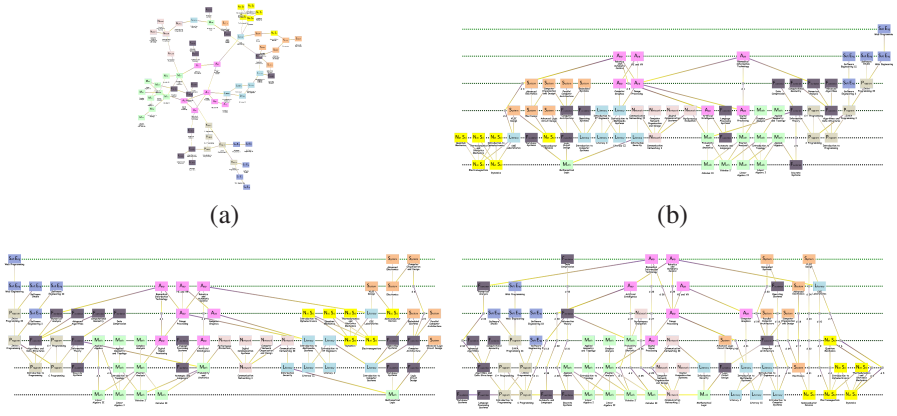
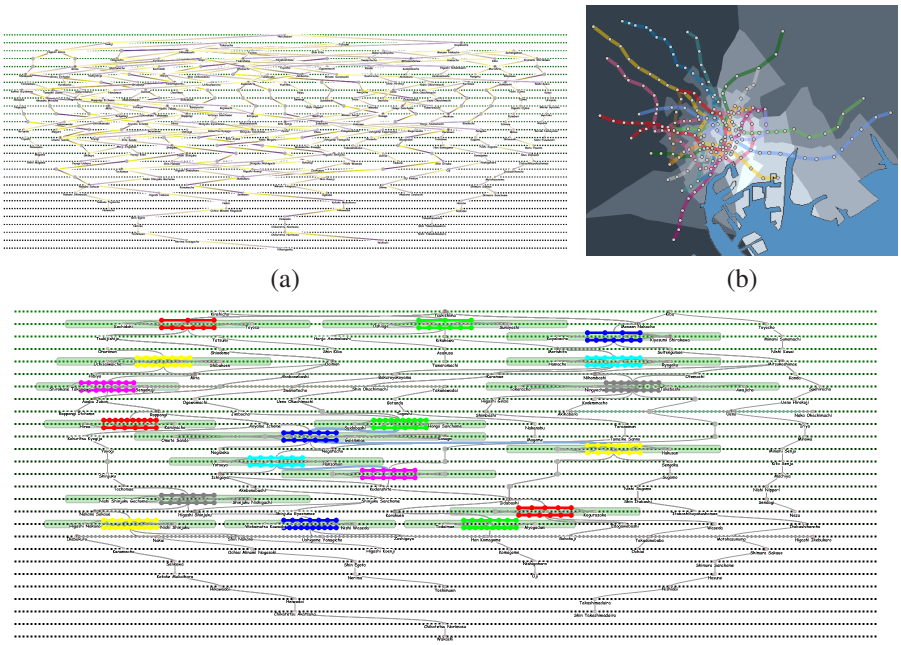


Figure 8: Rendering node clusters and contracted edges. (a) Initial network layout. (b) Nodes are grouped into clusters at each layer. A dendrogram on the left shows how we can adjust the similarity for controlling the number of node clusters. (c) Edges are contracted by referring to the node clusters. (d) Concentrated edges between clusters where the cross mark corresponds to a biclique (i.e., complete bipartite subgraph). (e) Symbolic representation that reveals how each node in the cluster is connected with the adjacent node clusters.



(a) (b) (c) (d)

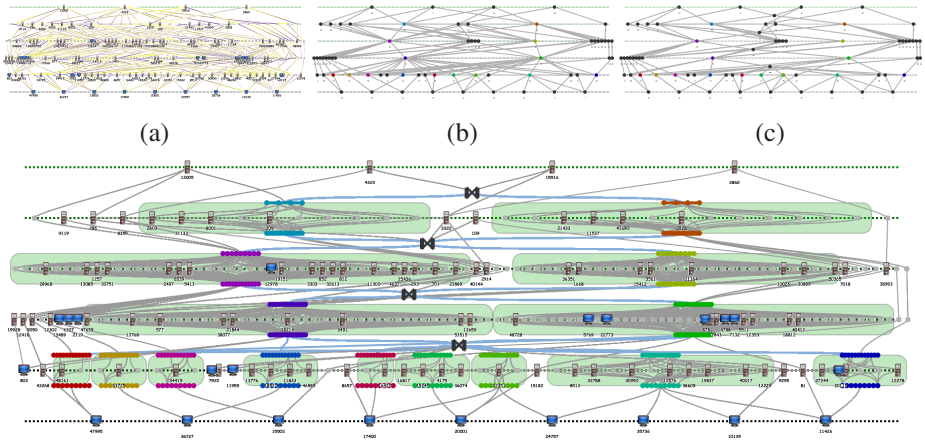
Figure 9: Designing curricula in the department of computer science. (a) Network layout obtained using the conventional spring-based method. (b) Hierarchical layout with $w_e = 100$, $w_r = 100$, and $w_d = 1$. (c) Hierarchical layout with $w_e = 100$, $w_r = 150$, and $w_d = 1$. (d) Additional constraints are imposed for equalizing the number of credits at the respective layers.



(a) (b)

(c)

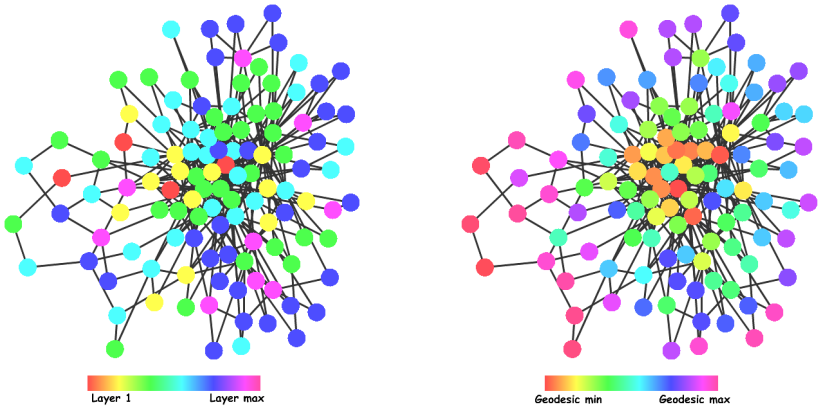
Figure 10: Tokyo metro network. (a) Layered representation of the Tokyo metro network where interchange stations are annotated with "T." (b) Accessibility zones deduced from the hierarchy of the metro network, where downtown areas are rendered in brighter colors. (c) Visualization with node clusters and contracted edges



(a) (b) (c)

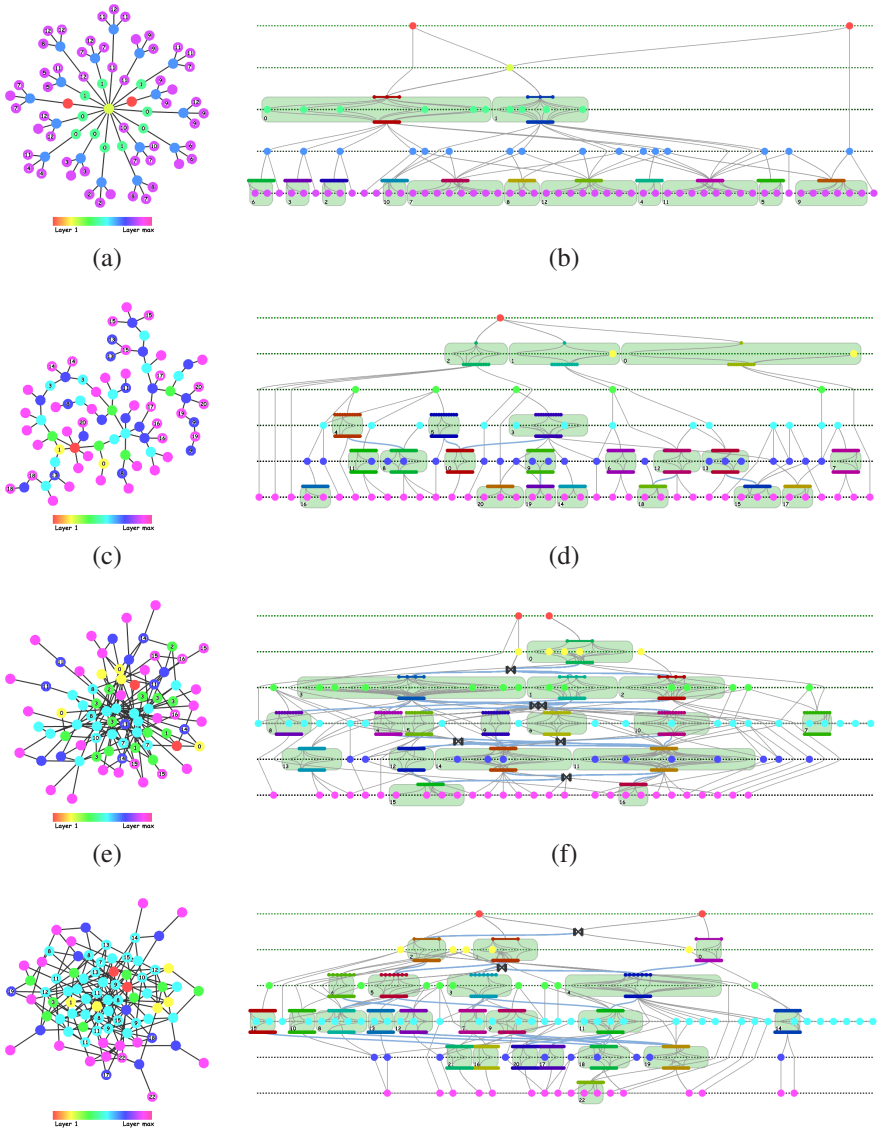
(d)

Figure 11: Visualizing hierarchical structures of P2P networks using our approach. (a) Clusters distributed at each hierarchical layout. (b) Cluster connectivity graph. (c) Optimized layout of another relatively large-sized P2P network.



(a) (b)

Figure 12: Comparison between the (a) hierarchical levels obtained using our approach and accumulated geodesic distances at the respective nodes of the P2P network.



(a) (b)

(c) (d)

(e) (f)

(g) (h)

Figure 13: Networks with different topological shapes. This includes (a) a banana-tree with $\#\{N\} = 81$, $\#\{E\} = 80$, and $\#\{P\} = 45$, (c) a small-world network with $\#\{N\} = 81$, $\#\{E\} = 81$, and $\#\{P\} = 43$, (e) a scale-free network with $\#\{N\} = 81$, $\#\{E\} = 170$, and $\#\{P\} = 72$, and (g) a random graph with $\#\{V\} = 81$, $\#\{E\} = 183$, $\#\{P\} = 70$.

Table 1: Computation times (in seconds) at steps of inferring hierarchical layers (layer), minimizing visual clutter (cross), and optimizing node spacings (space). #{N}, #{E}, and #{P} are the numbers of nodes, edges, and paths, respectively.

Example	layer	cross	space	#{N}/#{E}/#{P}
Fig. 1(b)	2.65	10.72	0.01	61/136/156
Fig. 10(a)	12.68	0.11	0.00	151/160/31
Fig. 11	36.99	2.57	0.01	116/306/525

Table 2: Comparison of crossing number and computation times (in seconds) at steps of node ordering optimization by us, Gansner et al. [19], and Gange et al. [18], while the formulation provided by Gange et al. cannot solve the problem in a reasonable time.

Examples	Ours		Gansner et al.		Gange et al.	
	cross	time	cross	time	cross	time
Fig. 1(b)	1129	10.73	1267	0.01	N/A	N/A
Fig. 10(a)	3	0.041	86	0.01	41	4.53
Fig. 11	7445	2.57	6292	0.01	N/A	N/A