Doctoral Thesis

Visualization of Large-scale Directed Graphs by Physical Simulation and Application to Bow-tie Structure and Temporal Evolution of Complex Networks

Yuji Fujita

March, 2020

Graduate School of Simulation Studies
University of Hyogo

Abstract

Visualizing a complex network can provide us helpful insights to understand topological structures and temporal evolution of the network. Social and artificial networks often consist of asymmetric relations and possess flow from upstream to downstream and circulation of things, such as commodities and information. Because such structures have vital roles in those networks, expressing link direction in the visualization is essential. While many graph layout algorithms have been studied, there are only few tools that enable us to find detailed structure of such flow and circulation especially for a large-scale network with millions or billions of nodes, recently available to researchers.

I propose a new visualization method based on physical simulation that is applicable to large-directed graphs. The method consists of two steps. The first step is an initial layout of nodes using the standard method of multidimensional scaling, but with a similarly measure being consistent with directions of links in an approximate way. The second step is a physical simulation using the so-called Barnes-Hut's hierarchical algorithms to calculate electric repulsion force with an additional of magnetic field to visualize links' direction.

First, I apply the visualization method to real data of social networks in order to study the so-called bow-tie structure in social networks. Social networks often have the graph structure of strongly connected component (SCC) and its upstream and downstream portions (IN and OUT), known as a "bow-tie" structure since a pioneering study on the World Wide Web (WWW). By using the visualization, I discovered that a bow-tie in the WWW usually has clusters, which are locally-located mini bow-ties that are loosely connected to each other, resulting in a formation of SCC as a whole. To quantify the mutual connectivity among such local bow-tie, I define a quantity to measure how a local bow-tie connects to others in comparison with random graphs. I found that there are striking difference between the WWW and other social and artificial networks including a million firms' nationwide production network among firms in Japan and thousands of symbols' dependency in the programming language of Emacs LISP, in which a global bow-tie exits. I argue that the difference comes from a self-similar structure and development of the WWW as speculated by others.

Secondly, I applied the method to visualize the temporal evolution of the production network, namely how the bow-tie structure changes along with time, to find a stable core part and ephemeral peripheral part. Combined with a quantitative analysis, I found that it is the way of transformation of this structure that makes both integrity and flexibility possible within a single economic network.

Contents

1 Introduction **1**
- 1.1 Introduction to complex network study . . . 1
  - 1.1.1 Introduction to graph theory and bow-tie structure . . . 3
  - 1.1.2 Self-similarity and complex network . . . 5
- 1.2 Introduction to graph drawing . . . 6
- 1.3 Directed complex graph drawing . . . 8
- 1.4 Criteria of complex graph drawing . . . 9

2 Graph drawing methodologies **11**
- 2.1 Brief overview of multidimensional scaling . . . 11
- 2.2 Direction-aware multi-dimensional scaling . . . 12
- 2.3 Distance in DMDS . . . 12
- 2.4 Force-directed method and the proposed model . . . 15
- 2.5 Hierarchical force calculation and related optimization . . . 18
- 2.6 Successive graph drawing algorithm . . . 21
- 2.7 Discussion . . . 23
  - 2.7.1 Advantage of DMDS . . . 23
  - 2.7.2 DMDS Limitations . . . 25
  - 2.7.3 Continuity and benefit of the successive graph drawing algorithm . . . 25
- 2.8 Other methods . . . 27

3 Analysis **28**
- 3.1 Bow-tie structures of the Web . . . 28
- 3.2 Sparsely unified SCC . . . 29
- 3.3 Bow-tie locality index . . . 30
- 3.4 Japanese production network . . . 35

4 Conclusion and future work **41**
- 4.1 Conclusion . . . 41
- 4.2 Future work . . . 42

Acknowledgment **44**

A Appendix **48**

1 Introduction

Complex network study is an interdisciplinary research field to analyze a large collection of relations, such as the relation between cities, people, computers, chemical substances and genes. This field originated from a mathematical branch named "graph theory", which provides several fundamental research methods to complex network study, including the "bow-tie" structure, which is the key concept of this thesis.

Network is a general term which refers a collection of relations. If the relation is asymmetric and have direction, the network is referred to as "directed". The bow-tie structure divides a directed network into three main parts, i.e. the bidirectionally connected central part, its entrance and its exit. Although the bow-tie is a relatively simple graph theoretical concept, it helps to understand the directed network by giving an idea how substances or information flows. Therefore, it has been applied to many different kind of networks since it was proposed in [1].

Despite the fact that it is widely utilized, the bow-tie itself did not gather much attention. In reality, the bow-tie shows conspicuous diversity among domains. The diversity is visualized by applying advanced network visualization method, which is also developed as a part of this study.

The objective of this thesis is to find out structural characteristics of network and its implications by focusing on the bow-tie's diversity. Based on the visualizations, further structural characteristics are identified quantitatively. Combined with network structural analysis and domain-specific observations, the research objective will be achieved including the mechanism behind the stability and flexibility of Japanese companies' production network, and missing self-similarity of the Web link structure.

Visualization-driven approach is repeatedly applied in this thesis, mainly to find questions. Ancient Greek philosopher Aristotle said "seeing is understanding" in his book *Metaphisics*. In some cases it is true but it is more useful to find a problem to solve, especially if the studied subject is not enough visualized. On the other hand, visualization alone usually does not constitute solid answer which can be deployed in the future. Visualization works better if it is combined with domain-specific knowledge and quantitative methods.

This thesis is organized as follows: In this section 1, foundational concepts and related previous works are introduced. In section 2, the research methods employed in this study, particularly the visualization algorithms, their optimizations and related discussions are addressed. In section 3, the visual examination of complex networks from heterogeneous domains are presented. And based on the visualizations, quantitative structural analyses are executed. Finally, in section 4, analyses and associated methods are summarized and concluded, and the direction of future work is provided.

1.1 Introduction to complex network study

Currently, there are various kinds of networks: information and communications, biological spices, substances, genes, social relations such as friendship, commerce and companies. These networks' behavior, like the cascading failure of financial network, is frequently unable to be predicted from observations of individual elements. In addition, the networks share fundamental characteristics among heterogeneous domains such as self-similarity. The network is something more than a simple collection of its consisting components and we cannot understand it just by observing its individual parts.

The origin of complex network study can be traced back to the beginning of graph theory: a branch of mathematics that studies the collection of object relations. The paper on the problem of the seven bridges of Koenigsberg[1] written by Leonhard Euler is considered as the first published study of graph theory (See [2]).

Unfortunately, the current configuration of the bridges and streets in Koenigsberg is different

[1] This is a historic name. The city is currently known as "Kaliningrad"

compared to the time of Euler. There exists a single stroke path that can pass through all bridges (which are more than seven) without duplication. But the aforementioned problem could not be solved in the time of Euler.

Single positive solution is enough to show that a problem is solvable. For example in case of Koenigsberg's seven bridges problem, a single stroke path that goes through all the bridges consists a proof that the problem is solvable. However, it is not as simple to show that a problem is unsolvable. Euler developed a new mathematical tool to show that it is impossible to solve Koenigsberg's seven bridges problem, and this tool eventually turned into graph theory.

Graph theory has developed into several major mathematical disciplines such as topology. It has also developed into "complex network study" - the interdisciplinary study of large-scale complex relational data, which is performed by using data-driven, empirical, and stochastic approaches.

We use the following terms in this study:

1. A *graph* denotes a mathematical structure whose definition is provided in subsubsection 1.1.1;
2. A *network* denotes a graph with domain-specific knowledge and information such as vertices' properties, connection weights, or relational layers, which may change with time.

Similarly,

1. A *vertex* denotes the most atomic element of the mathematical structure *graph*.
2. A *node* denotes a vertex, possibly with associated information as a member of the *network*.

And

1. An *edge* simply denotes an ordered pair of vertices.
2. A *link* denotes an edge with weight, the layer the edge belongs to, or other properties provided as domain specific knowledge of the *network*.

As described previously, large-scale complex relational data are obtained from various domains; hence the analysis conclusions can have wide range of applications. Interestingly, different kind of networks share one or more of the following properties:

1. Degree (node-wise count of relations) distribution
2. Self-similarity
3. Time-dependent development

Such universality suggests that there are commonly shared mechanisms to produce features such as degree distribution across heterogeneous domains. One of the major purpose of complex network study is to find such universal mechanisms and their applications. Among these features, self-similarity plays particularly important role in this study and it is separately discussed in subsubsection 1.1.2.

In complex network study, the term "degree" usually denotes a number of links by node. This parameter, particularly its probability distribution, is known to have fundamental implications on the properties of complex networks. The degree distribution frequently satisfies the condition given by the following equation: Let $P(x)$ be the portion of nodes with a degree larger than x,

$$P(x) \propto x^{-\alpha} \tag{1}$$

.

A network that satisfies Eq. (1) is often referred to as a "scale-free" network (see Chapter 4 of [3]).

1.1.1 Introduction to graph theory and bow-tie structure

It is known that a network with asymmetric relations has a feature named a "bow-tie" structure, which was first published in [1] as a world wide web (WWW) link structure research. Bow-tie structure is found almost universally in different domains' networks with asymmetric relations, and plays an important role in understanding these networks. The necessary definitions and analysis methods related to the "bow-tie" concept are described in this subsection.

A graph, G, is defined by a set of vertices $v_i \in V$ and a set of edges, which are ordered pair (Cartesian product) of vertices, i.e., $(v_i, v_j) \in E$. Graph $G = [V, E]$ is an undirected graph if $(v_i, v_j) \in E \leftrightarrow (v_j, v_i) \in E$ for all edges of G, which implies all the edges are symmetric. Otherwise the graph is *directed* (asymmetric relations).

Let $G = [V, E]$ be a graph and $V' \subset V$. A subset E' of edge set E is given by $(v_i, v_j) \in E' \leftrightarrow v_i \in V'$ and $v_j \in V'$ and $(v_i, v_j) \in E$. These two sets V' and E' also consists a graph G'. This graph G' is referred to as a *subgraph* of G.

The *breadth-first search* of a graph is an operation recursively executed by the following steps: There are several minor variations within this operation, which are based on whether the edges are symmetric or not.

Let $G = [V, E]$ be a graph.

1. At the start of the search, the set of non-empty vertices $B_0 \subset V$ is given as the initial search sequence with a length of 1.

2. Suppose G is a directed graph and the ith set of breadth-first search B_i is given. Next *forward* step vertices B_{i+1} is defined as follows:

$$v' \in B_{i+1} \leftrightarrow \exists v_i \in B_i, (v_i, v') \in E \text{ and } v' \notin \bigcup_{k=0}^{i} B_k$$

   .

   Backward breadth-first search is defined as

$$v' \in B_{i+1} \leftrightarrow \exists v_i \in B_i, (v', v_i) \in E \text{ and } v' \notin \bigcup_{k=0}^{i} B_k$$

   Undirected breadth-first search is defined as

$$v' \in B_{i+1} \leftrightarrow \exists v_i \in B_i, (v_i, v') \in E \text{ or } (v', v_i) \in E \text{ and } v' \notin \bigcup_{k=0}^{i} B_k$$

3. In case of an undirected graph, B_{i+1} is defined as

$$v' \in B_{i+1} \leftrightarrow \exists v_i \in B_i, (v_i, v') \in E \text{ and } (v', v_i) \in E \text{ and } v' \notin \bigcup_{k=0}^{i} B_k$$

   This is equivalent to *undirected* breadth-first search of directed graph considering the edge symmetry.

The *connected segment* of a graph is given by the limit union of undirected breadth-first search sequence $\{B_i\}$ as $\cup_{i=0}^{\infty} B_i$. Any vertex pair which belongs to the same connected segment is *connected*.

From set B_0 and breadth-first search sequence $\{B_i\}$, all the vertices $v \in \cup_{i=1}^{\infty} B_i$ are said to be *reachable* from B_0. Similarly, such set ($\cup_{i=0}^{\infty} B_i$) can be obtained by performing directed breadth-first search in either direction; let B_f be the forward reachable vertices set and B_b be its backward counterpart. Let S be a subgraph of G defined by a vertices set B_s given by

$$B_s = B_f \cap B_b \tag{2}$$

.

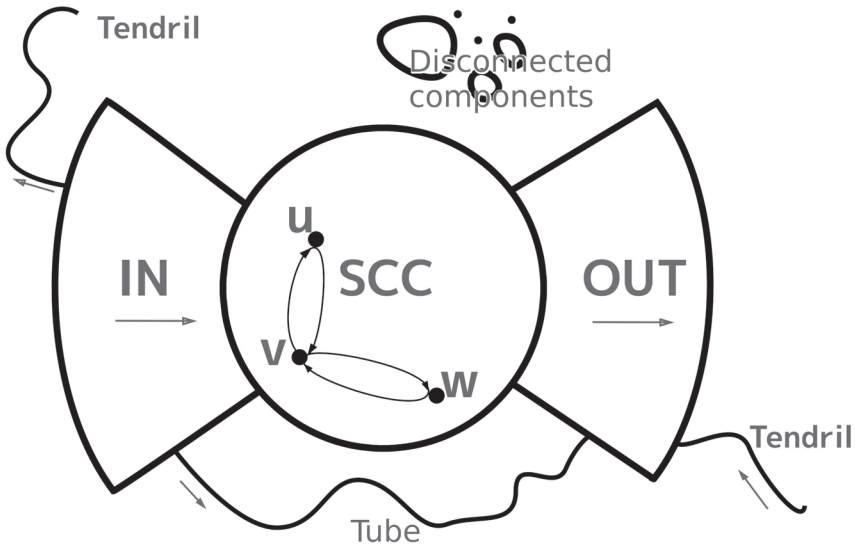

Figure 1: Schematic of Bow-tie. Three nodes u, v, w in SCC part are drawn to show back-and-forth mutual connectivity. Suppose v is the initial vertex of SCC detection breadth-first search and u, v belong to the intersection of forward and backward breadth-first search. u can be reached from w through v and u, w can be an arbitrary vertex of SCC.

It is evident that all vertices of S can be reached in the forward and backward direction from the initial vertices set B_0. As a result, $\forall v_i, v_j \in B_s$ there exists forward and backward paths from B_0. This implies that v_i can be reached from v_j both forward and backward by passing through B_0. Note that vertices v_i, v_j can be any member of B_s. This implies that any two vertices of S are mutually reachable in forward and backward direction. Such a subgraph, S, is referred to as a *Strongly Connected Component* (*SCC*) of G because it is connected *stronger* than undirected case. This is illustrated in the central part of Figure 1.

We introduce the bow-tie concept as follows: the segment that is forward-reachable from SCC is referred to as *OUT* and the segment that is backward-reachable from the SCC is referred to as *IN*. This IN-SCC-OUT configuration is called the "bow-tie structure" of a directed graph.

Figure 1 is a schematic of the bowt-ie structure. In addition to IN, SCC and OUT, there are other fragments such as disconnected components, which are not reachable from the main segment, or tendrils and tubes which can be reached following (or follwing backward) edge direction from other than the SCC.

As the definition of the SCC depends the initial vertices set B_0, the bow-tie structure itself also depends on B_0. Because the connectivity within the SCC is transitive and symmetric, the definition

of SCC delineates an equivalent class of vertices within the graph. This implies that there can be several disjoint SCCs within a single graph. In this study, we use the term SCC for the largest (in terms of the contained vertices size) of the (possibly) multiple SCCs. There are studies that use the term GSCC to explicitly denote the largest SCC ("G" stands for "giant").

Note that the operation of finding the SCC (and subsequently, IN and OUT) is constructively given by the definitions of breadth-first search and Eq. (2). The computational resource necessary to detect the SCC is essentially equal to that of breadth-first search, which is one of the most fundamental procedures of analyzing graphs. Practically, checking the size of an SCC is typically sufficient to ensure that it is the largest; however, this is not always the case.

Directed network analysis based on the bow-tie structure has been carried out in various domains for almost two decades since its discovery in 1999. Metabolism is a set of complex interactions that occurs among biological substances and enzymes. Such interactions can be represented as a directed graph, each node representing biological (or chemical) substances. The life takes necessary materials from the environment (IN), process them within the living organism (SCC), and excretes the resulting substance (OUT). The bow-tie structure helps to understand the vast network of biological substances to determine which step is critical to the entire organic system or how resilience to environmental fluctuations is implemented (see [4], [5], [6]).

The bow-tie structure is also applied to the safety assessment of complex industrial systems. An event of interest is placed at the SCC position of an event network, where the relation between the events is defined by causality. The causes and consequences are laid in the IN and OUT segments respectively. Critical events are pointed out by analyzing the network of causality ([5]).

The bow-tie structure has become an active topic in economics in recent years (see Chapter 9 of [7], [8], and [9]). It is known that the industrial sector configuration of IN, SCC and OUT segment is distinctively different. This implies that the three main segments of the bow-tie structure have different economic positions. The economical implications of this structure is also discussed in section 3 of this thesis.

1.1.2 Self-similarity and complex network

Self-similarity is a trait that an object is similar to a part of itself. Typical self-similar objects include fractals, such as space-filling curves and the Mandelbrot set. In the most strict sense, the terms "similar" and "part" are defined as follows: A compact topological set, X, is self-similar if there is an index set, I, and a set of nonsurjective homeomorphisms, $\{h_i : i \in I\}$, for which

$$X = \bigcup_{i \in I} h_i(X) \tag{3}$$

holds, where non-surjective homeomorphic function h provides the notion of similarity. In short, Eq. (3) states that X is the union of homeomorphic images of X.

However, in the case of complex network study, the meaning of "part" or "similar" is not always limited to the above definition because the real networks are finite and discrete. In case of finite discrete space, there's no nonsurjective homeomorphism which satisfies Eq. (3) exactly. Alternatively, for example in [10], the non-surjective map is defined between breadth-first search steps and homeomorphic similarity is defined by the invariant logarithmic ratio between the breadth-first search steps and the number of reachable vertices.

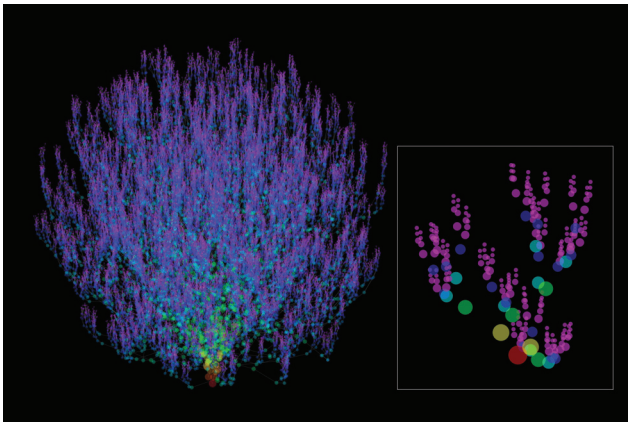

Figure 2: Example of self-similar network. Larger network can be created from inset smaller fragment by repeatedly replacing the fragment's terminal vertex by itself. The replacement operation defines homeomorphism h of Eq. (3). Note that the replacement operation is only approximately homeomorphic.

Despite theoretical limitations which come from being discrete and finite, self-similarity in complex network study has two significant benefits: predicting the property of an entire network by observing its subset and providing a network development mechanism. Fig. 2 depicts an example of an optimally self-similar network. The entire network can be created from the small fragment shown in the right inset by applying the simple rule of replacing the terminal vertex by itself. In this case, self-similarity provides not only the entire network structure, but also the network development mechanism. Unfortunately, finding homeomorphic function h of Eq. (3) is not always self-evident like the case of Figure 2.

1.2 Introduction to graph drawing

The English verb "see" means "to understand" beside the usage of visual sensing ability. The importance of visual sense in the intellectual context is emphasized by Aristotle at the beginning of his book, *Metaphysics*.

> All men naturally desire knowledge. An indication Book I What is Metaphysics? Universal desire for knowledge. of this is our esteem for the senses; for apart from their use we esteem them for their own sake, and most of all the sense of sight. Not only with a view to action, but even when no action is contemplated, we prefer sight, generally speaking, to all the other senses. The reason of this is that of all the senses sight best helps us to know things, and reveals many distinctions. ([11])

Visualization is also important in complex network study because such objects are generally both abstract and complicated. A good visual representation of complex large-scale graph can helps us to "see" (not only visually but also intellectually) them. In this subsection we will introduce necessary terms and concepts to discuss graph visualization and briefly review the previous works.

To keep the terminological convention of subsection 1.1, use the following terms:

1. A *graph layout*, or a *graph drawing*, denotes the process and its associated method of determining the coordinates of vertices to draw a graph.

2. The term *network visualization (or drawing)* denotes the result and its associated process of creating a graph layout with additional domain-specific knowledge such as the properties and names of the visualized objects.

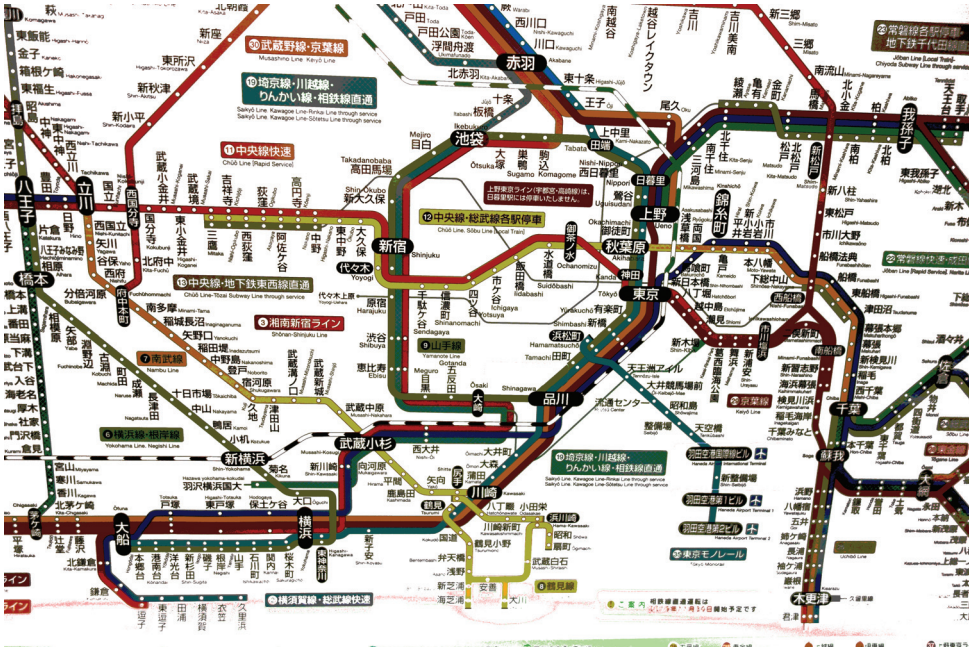

Figure 3: Central Tokyo railway network chart

For example, a *network visualization* is created based on a *graph layout* with additional information about the nodes and links.

In the case of a network with moderate size, the main purpose of visualization is to indicate each relation and provide the idea of general structure. It is also known that edges must be shown with the least possible crossings to maximize readability ([12]). A planar layout that can be printed on a piece of paper is also desirable.

The railway network chart of Figure 3 is a good example of moderate size network visualization. It shows East Japan Railway Company's railway lines of Tokyo area. It precisely shows which line connects which stations, so as to let the passengers reach where they want to go.

To summarize, a moderate-scale network drawing is evaluated based on the following conditions:

- It should be two-dimensional (planar).
- It should contain the least possible number of edge crossings.
- Symmetry should be reproducible for a symmetric graph.

The third condition of symmetry is occasionally considered to check the performance of graph layout method. It is to see if the method can reveal the structural symmetry.

A graph that can be perfectly expressed in a two-dimensional space is said to be *planar*. Related to planar graph processing, edge crossing minimization is a major topic in graph drawing, and this problem is NP-hard. Specifically,

> a procedure to determine if a given graph, G, has a layout with edge crossings less than a natural number, n,

is NP-complete. Proof is given in Chapter 2 of [13].

The problem of creating a planar graph layout and edge crossing minimization has an extremely important application in very-large-scale integration design, which consists a part of the process

of producing a large-scale integrated circuit. Planar graph layout is deployed to arrange circuit elements (transistor, condenser and other electrically functional parts) and wires on a two-dimensional surface. The number of wire crossings must be as small as possible to reduce production cost and improve product reliability. Despite the fact that it is NP-hard, several classes of this problem have been successfully optimized and solved in practice (see Chapter 2 of [13]). Such a solution should have contributed to the production of VLSI chips to reduce production cost and improve reliability.

1.3 Directed complex graph drawing

Although two-dimensional surface-based graph drawing with the least number of edge crossings has been studied extensively, and it has serious real-life applications, it does not meet today's complex network study's major demands. If a network contains tens of millions of links, it is not practical to ensure the readability of individual links. For the purpose of large-scale complex network study, the criteria for a graph layout should be different from being planar or minimizing edge crossings. For example, catching the overall feature of a network should be more emphasized than the readability of each connection.

The extraction of the features of a large-scale network has been studied for several decades. In recent years, graph drawing algorithms that use physical forces (e.g., spring tension, magnetism, and gravity) to arrange nodes have been developed and become widely available. They are conventionally referred to as "force-directed" algorithms. A review of force-directed methods is provided in [13, Chap.12 by Kobourov] and the references therein. Even though these methods tend to provide good layouts using only connectivity information, they have several limitations.

One of the first force-directed algorithms that provided a good layout is known as the "Kamada-Kawai spring model" (hereafter referred to as KK model), which was first published in [14]. This model assumes a single type of force, i.e., spring tension (or repulsion). Each pair of nodes is connected with a spring whose natural length is the shortest path length between these nodes. For example, consider a loop graph with four nodes $\{s, t, u, v\}$ and edges $\{(s, t), (t, u), (u, v), (v, s)\}$.

In the KK model, spring length is 1 except for two pairs of (s, u) and (t, v) for which the length is 2. There is no vertices layout that exactly satisfies this distance configuration. To adjust the layout, the springs of length 1 is expanded with tension, while the springs of length 2 are pushed and shrunk. As a result, the four vertices are placed in a square.

Although the KK model is one of the first force-directed methods that actually works, it has a problem that excessive computing resources are necessary. For a graph with n vertices, the model contains $O(n^2)$ springs, which consumes the same amount of working memory. It needs $O(^2)$ computation time, too. The algorithm does not scale to the number of vertices.

Since the late 1990s, this limitation has been overcome using several methods. A multiscale (or multilevel) approach is the method of coarsening a network to extract core feature to obtain an approximate layout, then further details are added to complete the visualization. One of the first applications of such an approach can be seen in [15]. Other approaches are based on a feature within an embedded space rather than connectivity structure. They spatially divide vertices set and obtain the layout, e.g., [16] and [17], even though [17] combines a multiscale method to improve quality.

Also, computing hardware has been developed to create highly-parallel computing environments. This trend is omnipresent in every layer of computing settings, from arithmetic logic units on microprocessors to users' applications. The implementation of graph layout algorithms has been developed to accommodate parallel computing environments of different levels.

One of the most difficult problems in the visualization of directed graphs is how to process the cyclic paths. In a few studies, cyclic paths are cut open to tree in creating visualizations (see [18]). However, edge loops cannot be cut open in this study because relational loops have an extremely important role in economics (see [19]) and the analysis of Japan's production network is one of the

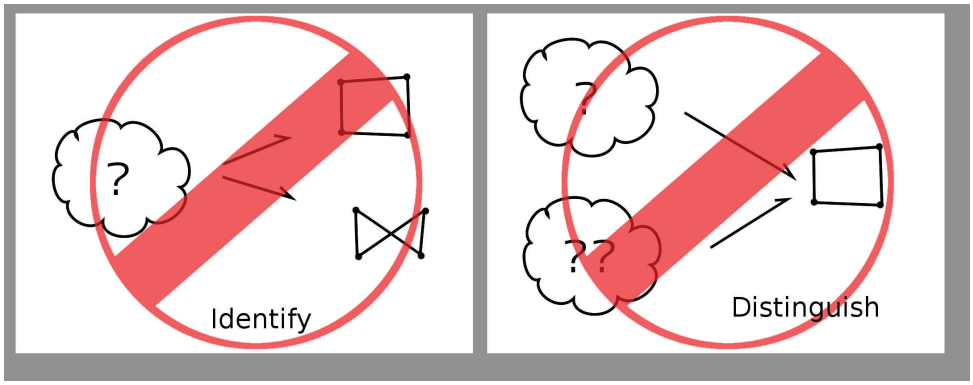

Figure 4: Identity and distinguish conditions of graph drawing.

major subject of this study. In fact, according to our data set, more than 13 percent of companies have back-and-forth customer–supplier relations. Cutting these cyclic paths open is a complete destruction of this structure, which is not acceptable.

The force-directed method was extended by Sugiyama and Misue [20] to express edge direction by magnetizing the edges, which is the method adapted in this study.

1.4 Criteria of complex graph drawing

As seen in in [13], there are several criteria for judging how "good" a network visualization is, such as the number of edge crossings or whether symmetry can be found in a symmetric graph image. These conventional criteria still retain value today. However, they have limitations in processing actual complex networks. For example, we cannot neglect the fact that there is no such thing as a "complex symmetric network". This implies expressing symmetry is not a major concern in case of complex network visualization. We propose alternative criteria for evaluating a drawing method.

In mathematical terms, graph drawing is an attempt to *map* a graph to a visually recognizable expression. We expect this *map* to satisfy the following two conditions:

1. No two visualizations are permitted to share their source graph.

2. No two graphs are permitted to share their visualizations.

In other words, an identical graph must consistently have the same layout, and no polymorphism is allowed. Moreover, different graphs must generate different drawings. We refer to these conditions as "identity condition" and "distinguish condition" respectively.

These conditions appear trivial at first; however, they make sense if we use generated graph layouts to analyze actual data. By using the network visualization result, we perform qualitative analysis on subjects, for example, identifying or detecting features. A feature is a property that can be found only in a limited class of subjects. If two networks that belong to different classes share a single visualization, the feature must be neglected in the visualization. The distinguish condition is essential for catching features visually.

The importance of the identity condition might look even more evident. It would be surprising that there exists a network drawing method that generates different images from identical data every time. The force-directed method is generally known for producing results with fine details, however, it has a limitation considering this condition; the arrangement of vertices is randomly rotated or flipped if the initial configuration is selected randomly. This implies that multiple images can be obtained from a single network.

In this study, the quality of a network visualization method is judged based on the two above-mentioned conditions, as summarized in Figure 4, and not by the number of edge crossings or symmetry preservation.

2 Graph drawing methodologies

In this section 2, we describe the analysis method used in this study, mainly the graph layout algorithms of directed graphs and successively transforming graphs.

The developed graph drawing method consists of two stages. In the first step, the DMDS[2], which is an extended MDS[3] to be able to show edge direction in the layout deterministically provides an approximate layout. In the second stage, physical simulation is performed by way of hierarchically optimized SIMD[4] pipeline calculations to extract fine details. It gives the final touch to the layout . These two stages are combined to satisfy the graph drawing criteria discussed in section 1.

Even though force calculation becomes more and more efficient, the graph layout problem may not be completely solved in terms of benchmark of subsection 1.4 owing to the structural diversity of graphs. Graphs can be extremely densely connected, extremely sparse or both of such parts coexist in a single graph. To overcome such difficulty, two different algorithms are combined to complete each other's short comings to realize a universally applicable visualization method.

These two abovementioned steps cannot be scaled to the data size of this study without appropriate optimization. Therefore, the algorithms described in section 2 are all optimized so that they can be applied to a wide range of data comprising millions of vertices and tens of millions of edges, with time-dependent changing. These optimization are also addressed in this section 2.

2.1 Brief overview of multidimensional scaling

MDS is a well-known method of arranging objects based on their mutual distances. It was developed by W. S. Torgerson in 1958 ([21]). It places objects in a low-dimensional vector space (generally two-dimensional or three-dimensional Euclidean space) to reproduce given mutual distances with the minimum error.

To illustrate this method, we first consider a problem of MDS in reverse order: starting from the layout coordinates. Suppose there are n objects scattered in a low-dimensional (again, two-dimensional or three-dimensional) vector space. Let v_i be their coordinate vectors ($i = \{1...n\}$). We can obtain n^2 product values from these n vectors. Let $x_{i,j} = v_i v_j$ be such values. Suppose an $n \times n$ matrix, X, is packed with these vector product values; X can be written as

$$v^T * v = X$$

.

Before reaching the goal of mutual distances of the reversed MDS task, we consider yet another case as follows: suppose matrix X is given as the mutual vector product of the coordinates of n objects. Then, X is decomposed back to the n coordinates of the objects through the eigenvector decomposition of matrix X. Let v be several larger eigenvectors of X with corresponding eigenvalues, λ. Then,

$$X \approx \lambda v^T * v$$

. This step is equivalent to principal component analysis in statistics. Even though this process gives solution to the problem similar to that of MDS, it still is not identical to the question that we want to answer because the matrix X is not given as mutual distances but as products. To solve the problem, we have to translate mutual distances into mutual vector products. Then, the layout of n objects can be obtained by eigendecomposition.

[2]DMDS stands for Direction-aware MultiDimensional Scaling.

[3]MDS stands for MultiDimensional Scaling, which is explained briefly in subsection 2.1.

[4]SIMD is an abbreviation of Single Instruction Multiple Data, which is a parallel computation architecture that simultaneously performs same operation on multiple different data.

This translation step can be a achieved by a two-dimensional normalization, which is referred to as Young-Householder translation. The translation was published in [22]. To summarize, MDS obtains the object layout by translating a mutual distance matrix into a mutual product matrix and processing this matrix with eigendecomposition.

When MDS is used in the context of graph layout, the distances between vertices are given by the shortest path length, which can be calculated by breadth-first search.

2.2 Direction-aware multi-dimensional scaling

The concept of distance is widely known to be symmetric (for example, see definition of *distance* in [23] Chapter 2, p. 30), which means it totally lacks the notion of direction. As the symmetry of the concept of distance is widely accepted as a solid rule, it seems impossible to show the edge directions in the MDS result.

However, if we carefully examine a desirable result, we can find a way to alter the distance to incorporate the edge directions.

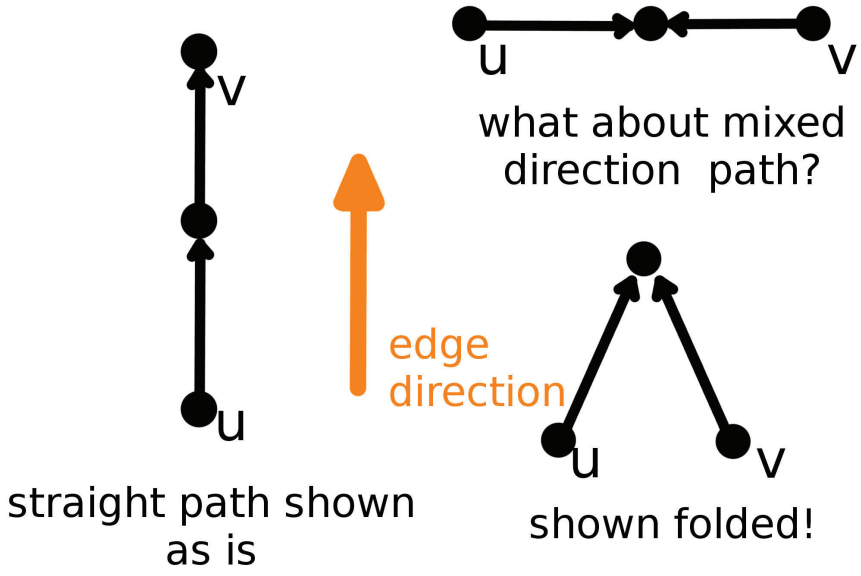

Figure 5: DMDS concept: Adjusting metric to incorporate the edge direction into visualization. Figure from Fig.2 of [8].

Consider the two cases shown in Figure 5. In the case shown on the left, starting from vertex u, we can reach vertex v by following the direction of the edges. This is impossible in the case on the right.

Suppose we want the directed edges to be aligned with the vertical axis. This implies that the "straight" path shown on the left-hand side of the figure should be drawn with a linear and vertical layout. Next we consider a "mixed path" case as shown on the right-hand side in Figure 5. As we want the edges to face "upward", we want the paths to be folded in an anhedral shape. Consequently, the distance between u and v must be smaller compared to the straight path case.

In conventional MDS, the distance of both cases are 2; hence, conventional MDS never recognizes the edge direction. If we could differentiate these cases, we can incorporate edge direction information into the MDS result.

2.3 Distance in DMDS

The definition of distance in DMDS is given as follows:

Let $d_1(u, v)$ be a function that returns the length of the shortest path that starts from u to reach v by following the edge direction; the function returns infinity if there is no such path (unreachable). Note that $d_1(u, v) \neq d_1(v, u)$ in general. Using d_1, we define another function d_d as

$$d_d(u, v) = min\{d_1(u, v), d_1(v, u)\}$$

, which provides the directed distance between u and v. In addition, note that it is symmetric: $d_d(u, v) = d_d(v, u)$. Let $d_u(u, v)$ be the undirected shortest path length from vertex u to vertex v, obtained by undirected breadth-first search. Using d_d and d_u, we can define the distance function, d, of the proposed algorithm, as follows:

$$d(u, v) = \begin{cases} d_d(u, v) & \text{if } d_u(u, v) = d_d(u, v) \\ \frac{d_u(u,v)}{2} & \text{otherwise} \end{cases}$$

As already noted that d_u and d_d are symmetric, d is also symmetric. In this definition subsection 2.3, the upper case represents the length of shortest path which follows the edge direction. In this case, we wish the path to be drawn as a straight segment, similar to the left side of Figure 5. The lower case in Section 2.3 corresponds to the case when the shortest path does not follow an edge direction, which corresponds to the right side of Figure 5. As the path must be shown in a folded manner, the distance is cut short to half of the shortest path length.

We also note that in case of connected graph, d always yields a well-defined finite value, even though d_d can be infinite. Unfortunately, function d does not satisfy the triangle inequality; hence, it does not qualify to be a mathematical metric function. To be added, we can easily extend Section 2.3 to be applicable to disconnected graphs by adding a case when d_d is infinite. Calculating adequate value for disconnected case can be an interesting problem, but we will not discuss it any further here. For reference, we use disconnected distance value $d = \frac{2log|V|}{log2|E|-log|V|}$ in the implementation of this study.

Let us provide a few examples of distance function d.

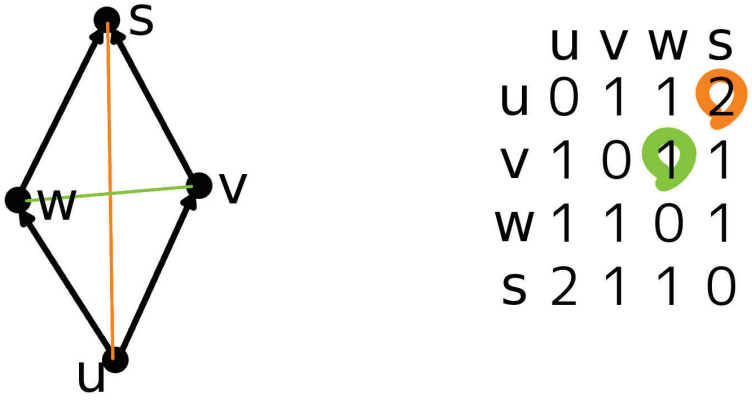

Figure 6: Example of 4-loop graph with diamond shape. The graph on the left is shown with edge directions. The table shows the distance between the vertices. Note that the distances between v, w and u, s are marked in green and orange, respectively. Figure from Fig.3 of [8].

Figure 6 shows how a diamond shape is obtained from the edge directions. As vertex w is unreachable from vertex v by following an edge direction, $d_d(v, w) = \infty$ while $d_u(u, s) = 2$.

Consequently, $d_u(v, w) \neq d_d(w, v)$, which implies that this distance corresponds to the lower case in the definition Section 2.3 of function d.

$$d(v, w) = \frac{d_u(v, w)}{2} = 1$$

In contrast, for $d(u, s)$, as vertex s is reachable from u by following edge directions, $d_d(u, s) = d_u(u, s) = 2$, which implies that

$$d(u, s) = d_d(u, s) = d_u(u, s) = 2$$

.

As a result, $d(u, s)$ is two times as large as $d(v, w)$, which yields an elongated diamond shape to meet the edge directions.

This algorithm will not work when the number of the vertices becomes large. Suppose the number of vertices is n, the necessarily working memory to house the distance matrix is $O(n^2)$. The eigendecomposition also requires $O(n^2)$ of computation time. With this amount of required resources, the algorithm cannot be applied to a large scale data. For the purpose of computational optimization, we follow the pivot-MDS (PMDS) method, which is described in [24]. We select k representative vertices (pivots) and fill an $n \times k$ matrix with the values of direction-aware distance function of Section 2.3. Then, the matrix goes through the Young-Householder translation and singular-value decomposition to obtain the plot.

Owing to this optimization, we can obtain a suitable layout that can be used for the initial layout of force-directed method within reasonable computation time. But there still remains a problem to select the pivot vertices.

As discussed in [24], there are several different ways to select pivots. The most simple way is a random selection, which is known to work not very well. In this study the pivot selection procedure is organized and executed as the following pseud-code, where the variable $pivots$ is the pivot vertices list and the function hub picks up the highest degree vertex from a given segment. The function bfs performs breadth-first search from the given vertices, to the given direction.

```
(let ((bowtie (list IN SCC OUT))
      (pivots (map (lambda (segment) (hub segment)) bowtie)))

 (repeat times
   (lambda
     (map
       (lambda (dir) (push pivots (pick (last (bfs pivots dir)))))
         (list 'forward 'undir 'back)))))
```

$pivots$ is initialized as the list of the hub vertex from IN, SCC and OUT segments. A vertex to be added to the pivot list is picked up from the most distant vertices by performing directed breadth-first search which starts from current pivot vertices list. This procedure is repeated for given times. The pivot pickup strategy is very important as it gives the view point to see how does a network look because each vector consists of distance list from the pivot vertex. The strategy described above is to try to pickup representative vertices both in terms of distance (shortest path length) and link direction. We gathered eighteen pivot vertices by repeating above procedure for five times.

Considering two graph layout conditions of subsection 1.4, consistent reproduction of the output is very important point of DMDS as it is to provide the initial coordinates for the second step. The consistency can be assured if we use the same pivot vertices list every time.

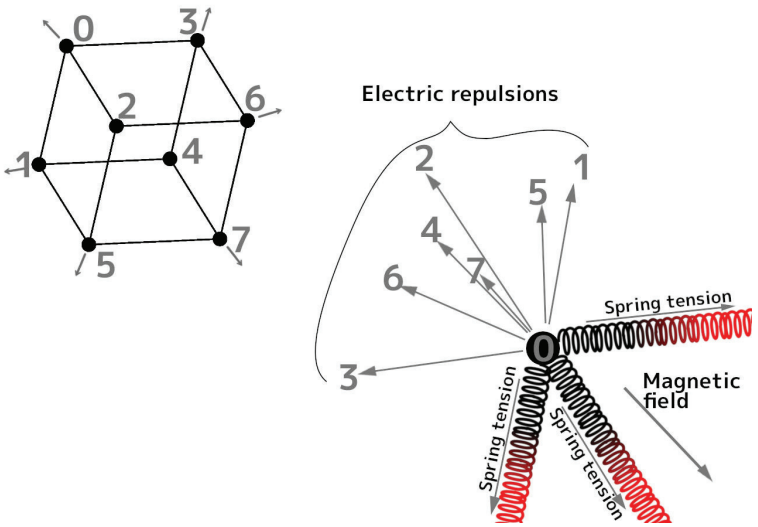

Figure 7: Physical simulation model of this study. The upper left figure shows that every vertex receives electric repulsion forces from all the other vertices (shown by the radiating gray arrows), which spreads them out in balance, while the edges pull linked vertices close by spring tension. Lower right figure shows the magnified view of vertex "0" to illustrate the forces in detail. The vertex "0" receives repulsion from vertices 1. . .7, which are compiled to push the vertex to upper left direction. Spring tension keeps linked vertices close: edge (0,3) pulls vertices to the right, (0, 1) pulls vertices downwards, and (0, 2) pulls vertices to the lower right. Finally, magnetized springs (polarization shown in red and black) align their orientations based on magnetic field.

2.4 Force-directed method and the proposed model

As described in subsection 1.3, force-directed algorithm is a group of methods that the vertices are laid out by the assumed kinetic forces which work between the vertices. In practice the procedure is done by the numerical simulation on the computer. In this study we use three types of kinetic forces, namely,

1. the spring tension between linked vertices,
2. the magnetic force applied to a directed edge,
3. and the electric repulsion between all vertices.

These forces work together to construct the arrangement which shows the network characteristics.

The advantage of the force-directed method is that it only requires connectivity information. This implies that the method is applicable almost universally.

Electric force is either repulsive or attractive depending on the electric charge of interacting bodies, and the attractive force of electricity is analogous to the gravitational force. The motion of numerous bodies under gravitational force is commonly referred to as the "n-body problem," and it is well known that a general analytical solution does not exist for $n \geq 3$. The numerical solution, or computer simulation, of the n-body problem has advanced considerably since the last half of 20th century along with the progress of information processing technology. This advancement benefits graph drawing because we just have to reflect the direction of gravitational force to obtain electric repulsion.

Based on the laws of physics, the following steps are repeated to move objects according to physical force:

1. Objects are moved by small distance according to the physical status and the applied force.

2. The physical status is updated based on the calculation of the new layout of the objects.

In terms of mathematics, repeating above two steps is equivalent to integration.

Let F be force, x be a coordinate and m be the mass of a vertex. Newton's equation of dynamics is

$$F = m\frac{d^2x}{dt^2}$$

which describes how objects move. To determine x, it can be obtained by integrating the force twice. This process can be implemented on a computer by iterating the following steps: Let V be the velocity of a vertex,

$$\begin{cases} V(t+\Delta t) = V(t) + \frac{F(t)}{m}\Delta t \\ x(t+\Delta t) = x(t) + V(t)\Delta t \end{cases} \tag{4}$$

. As spring tension, electric repulsion and magnetic force are conservative, performing simulation based on the scheme of Eq. (4) will let the vertices move forever, never determines the graph layout.

Instead of using integration scheme of Eq. (4), there is an alternative scheme that updates the position of the vertices by adding the force directly to the coordinate:

$$x(t+\Delta t) = x(t) + F(t)\Delta t. \tag{5}$$

Eq. (5) simulates the case when the network is dipped into considerably viscous non-conducting fluid.

Another method of integration is to add (kind of) air-drag resistance to the system. Let c be the air-drag coefficient:

$$\begin{cases} V(t+\Delta t) = V(t) + \frac{F(t)-cV}{m}\Delta t \\ x(t+\Delta t) = x(t) + V(t)\Delta t \end{cases} \tag{6}$$

Note that the velocity is updated differently compared to Eq. (4).

In this study the time-integration scheme of Eq. (6) is employed because it is less susceptible to the so-called entanglement problem compared to Eq. (5). Entanglement problem is a polymorphism of the layout which violates the first condition given in subsection 1.4. For example a loop graph can be optimally laid out in the shape of figure "0", while a layout of the shape of figure "8" is an entangled layout.

Figure 8 shows the result of time-integration scheme comparison experiment of Eq. (6) and Eq. (5). Remaining potential distributions of two compared schemes are plotted. The scheme of Eq. (6) consistently reaches lower potential (which means better layout) state compared to the scheme of Eq. (5). The result clearly justifies the selection of Eq. (6) by the conditions of subsection 1.4 because it reaches identical terminal status consistently. In addition, the entangled square graph layout of scheme Eq. (5) contains more edge crossings compared to the clean square layout of Eq. (6).

As illustrated in Figure 7, the force F is equal to $E + H + M + D$, where E, H, M, and D represent electric repulsion, edge spring tension, edge magnetism, and air drag, respectively. These forces can be computed from the coordinates or velocities of vertices, as follows: Let F_i be the force applied to vertex v_i, k_e be Coulomb's constant, m_i be the mass (or electric charge) of vertex i, h

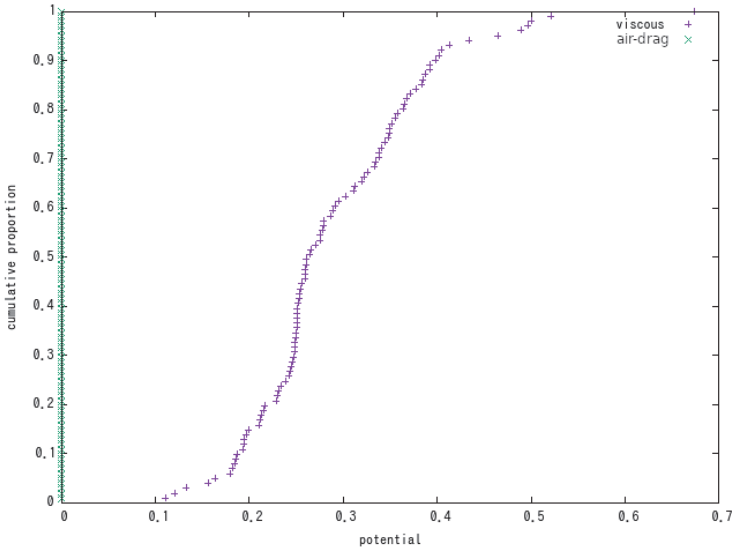

Figure 8: Comparison of the quality of layouts of viscous fluid scheme (Eq. (5)) and air-drag scheme (Eq. (6)) on a 4-loop (square) graph. Potential remaining after running time-integration 1000 times is measured from 100 trials. Smaller potential is better. Air-drag integration scheme of Eq. (6) clearly outperforms viscous fluid of Eq. (5).

be Hook's spring constant, f be the magnetic field vector, and c be the air-drag coefficient:

$$\begin{cases} E_i = \sum_j \frac{k_e m_i m_j (x_i - x_j)}{|x_i - x_j|^3} \\ H_i = \sum_l h(x_i - x_j) \text{ for } i \text{ and } j \text{ such that } l = (i,j) \in Edge \\ M_i = \sum_l f_l \\ D_i = -cV_i. \end{cases} \tag{7}$$

Among the forces listed in Eq. (7), computing E requires $O(n^2)$ time because $O(n)$ steps are required for the computation for each vertex, which is repeated $O(n)$ times. H and M depend only on the total number of edges, and D depends on the number of vertices, n. The physical simulation of the force given by Eq. (7) is carried out under the scheme of Eq. (6). The optimization of force E is discussed in subsection 2.5.

The actual simulation is executed by the following C code: Struct Nbodyinfo contains the particle (graph vertices) information of electric charge, coordinates, velocity, acceleration as

```
typedef struct {
  int n;
  double *m;
  double (*x)[3];
  double (*v)[3];
  double (*a)[3];
  double *p;
} Nbodyinfo;
```

Each type of physical information is materialized as an array of three double precision numerals, whose length is the size of vertices (the size is given when initialized). The time-integration step of Eq. (6) is implemented in C using the Runge–Kutta method, as follows:

```
static VALUE nb_dg_frog(VALUE self, VALUE dt)
{
  Nbodyinfo *nb;
  VALUE np;
  float slice;
  slice = rb_float_value(dt);
  np = rb_iv_get(self, "@ni");
  Data_Get_Struct(np, Nbodyinfo, nb);
  push_velocity_dg(self, 0.5*slice, nb);
  push_position(slice, nb);
  push_velocity_dg(self, 0.5*slice, nb);
  return self;
}
```

In the actual simulation the mass of vertices, electric charge, the length of the link (spring), and Coulomb's constant are all set to 1.0 for simplicity, even though we can designate each value separately if required. The time-step value, Δt, in Eq. (6) is given to this **nb_dg_frog** function as its argument, **dt**. In most cases, **dt** is set as 0.1. A few extremely densely connected graphs require the time-step value of 0.05 or less owing to high acceleration from accumulated spring tensions. We can change this value on demand to accelerate convergence.

Changing the natural length of the spring is the best method to adjust the edge length. Changing the elasticity parameter also alternates the finished layout's edge length, but it tends to produce too much tension and makes smaller timestep **dt** is required.

It is not required to directly use the C functions listed above. Instead, we call method of force calculation and time integration from the Read-Eval-Print Loop (REPL) environment in Ruby. See the Appendix for the usage of these methods.

2.5 Hierarchical force calculation and related optimization

As discussed previously, the physical model of this study has $O(n^2)$ pairwise repulsion. Overcoming this difficulty of computing time consists a major part of the development of the force-directed method. We use an optimization technique developed in astrophysics [25] to reduce the computation time to $O(n \log(n))$ while maintaining accuracy. This algorithm is known as "Barnes-Hut's Tree" method.

This optimization method can be separated into two major parts: the "divide" part and the "conquer" part. The first part divides a space into an oct-tree. A cubic space is divided into 2x2x2 smaller cubes repeatedly until the cube contains no particle (the cube is discarded in this case) or a single particle. The left chart in Figure 9 shows 2x2 planar division for simplicity. A cell that contains multiple cells is referred to as a "multicell;" otherwise, it is referred to as a "single cell." In this manner, the space is represented by a tree whose depth is of order $O(\log(ParticleCount))$. The point marked with "X" is the location of the particle that receives force.

The second part is illustrated on the right-hand side of Figure 9. The force applied to point "X" is obtained as follows: a parameter θ is given externally as a "view angle." A multicell with a view angle smaller than θ is reduced to a single cell, whose particle mass is aggregated and located at the center of mass. In the right-hand side of Figure 9, the cells with gray numbers are the multicells to be reduced. A cell with view angle larger than θ is calculated as is. Thus, the view-angle parameter defines the balance between optimization and precision.

As seen from the algorithm description, the entire visualization space is represented as a single tree, and the force calculation procedure is its traversal. The entire tree traversal is an aggregation of subtree traversals due to the self-similarity of trees. This property makes it easy to blend the

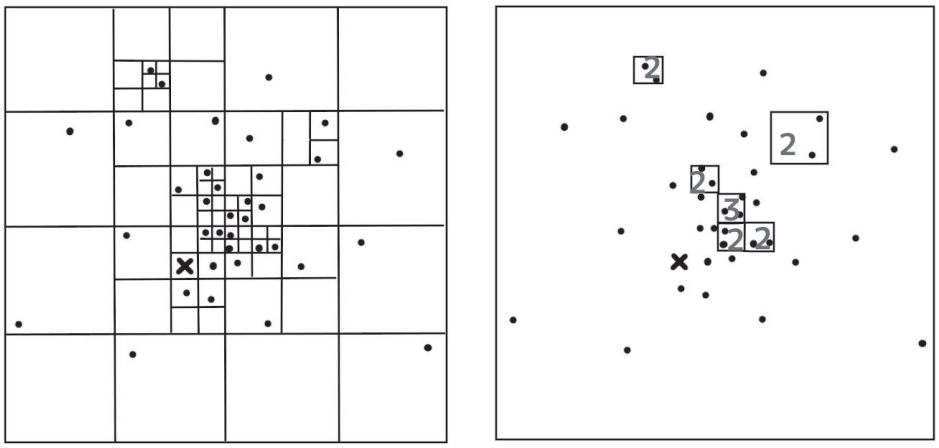

Figure 9: Hierarchical spatial cell partition (left) and force calculation (right) based on the partition (from [25]). Force to point "X" is to be calculated. In the right figure, higher-order cells that contribute to optimization are indicated by squares associated with numerals. During force calculation, these cells are simplified to a single particle located at the center of mass, with compiled mass of all contained particles. Simplification is performed according to the opening angle, θ, of the cell seen from point "X". θ is given as a hyperparameter; larger θ corresponds to lower precision with more computational optimization. Figures from [25] have been modified by the author.

calculation process into parallel computing scheme.

The amount of computation required to traverse the tree can be estimated as follows; suppose increase the number of particles exponentially by concatenating three more squares (or seven more cubes in case of three dimensional space) with the same size of the original space. It will double the scale of the space. With this new configuration, new tree only adds constant number of leaf nodes because of the view angle. It means exponential growth of the particle will result in a constant growth of tree nodes, hence the traversal computation takes $O(\log(n))$ of time. This process is repeated n times to obtain the acceleration of the all particles, which means overall computational time becomes $O(n \log(n))$.

In this study, the traversal of the space is implemented using space-filling curves such as Hilbert's curve or Lebesgue's curve. The first space-filling curve definition was provided by G. Peano in 1890 as "Sur une courbe qui remplit toute une aire plane" in Math. Annalen., 36, pp. 157-160. For the purpose of force calculation, the most important property of a space-filling curve is that it can realize spatial locality on a one-dimensional linear address space (which is the computer memory structure). See Figure 10 how spatial locality is implemented on the linear addressing space. Space-filling curves have considerably more significance beyond information processing, and they are strongly connected to the beginning of modern mathematics, particularly set theory (see [26]).

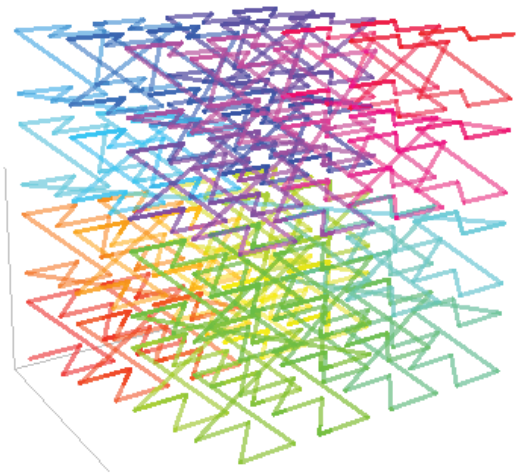

Figure 10: Spatial locality sensitive memory allocation is realized by a self-similar space-filling curve. Note that similarly colored lines are folded together in a spatially close space, which means spatially close points are allocated to nearby address of the working memory. Figure from https://commons.wikimedia.org/wiki/File:Lebesgue-3d-step3.png, created by Robert Dickau, distributed under conditions of CC BY-SA 3.0 (https://creativecommons.org/licenses/by-sa/3.0).

See [27] for the detailed procedure of allocating hierarchical cells using a space-filling curve. As the allocation achieves spatial locality on a linear addressing space of computer memory, it can naturally map stored data onto modern hierarchically parallel computing architecture, particularly hierarchical working memory cache. As frequently accessed spatially close data are allocated to adjacent addresses, it improves cache hit rate and data fetch performance.

Moreover, owing to the hierarchical force calculation algorithm, spatially distanced data tend to be reduced into a single particle, which contributes to the reduction of parallel processing cost and improves parallelism.

These optimization methods are implemented on the GRAPE platform (see [28], [29]), whose core SIMD pipeline is realized by the Phantom-GRAPE numerical software library (see [30]), which uses the Advanced Vector eXtension (AVX) machine instruction set. The AVX machine instructions enable the software to perform multiple vector calculations simultaneously. This is internally realized within the microprocessor as a data operation with a length of 256 bits. These core instructions are written in assembly language and translated to corresponding native code to perform the SIMD operation on a common CPU.

As the SIMD vector calculation considerably accelerates the vector calculation, the hierarchical space division process can be eliminated and the cell can be enabled to carry more than a single particle. This is a matter of balance based on practice, and it cannot be determined theoretically because it depends on actual computing environments. In the case of the author's laptop environment[5], 500 particles per cell provide the best performance, while the Core i7-6800K processor performs best when it is set to 1,200 particles per cell.

As described in subsection 2.4, these highly optimized numerical calculation software is packed into a Ruby extension. Calling the force calculation method sets the acceleration values in the Nbodyinfo struct member using the optimizations presented in this subsection. The Nbodyinfo struct members are accessible as Ruby data, and it is easily browsed and manipulated through the REPL environment. This overall construction enables users to flexibly design and execute various

[5]Intel(R) Core(TM) i7-6560U CPU at 3.2GHz, 16GBytes of RAM.

types of analysis without compromise of optimization.

2.6 Successive graph drawing algorithm

In this section, we introduce algorithm for incrementally creating the layout of a successively changing network. The algorithm generates a layout of current network based on its previous layout by processing the structural change in a graph.

The evolution of networks is one of the most fundamental topics of complex network study. For example, a model that generates a network with a power-law degree distribution goes through a process of increasing vertices and edges to reach the specified status. This has an implication how a specific type of complex network is realized, i.e., the dynamics of network evolution.

To obtain a time-series data of complex network is more difficult than to obtain the network of particular time. Despite the data acquisition obstacle, drawing time-dependent graphs has been extensively studied, particularly after the 2000s (see [31] and references therein).

The methods developed for this purpose are largely divided into two major categories, i.e., "online" and "offline," depending on the input required to obtain the output. The term "offline" generally refers to a method that requires an entire series as input. This type of method delivers the output after an entire sequence is known. The visualization is generated based on the optimization over the entire sequence.

In contrast, the term "online" refers to an incremental method. The entire sequence of information is not required in this type of method. For example, [32] proposed a method that only requires the previous layout and new (to be drawn) network information to obtain the next layout.

The method introduced here, which was first proposed in [33], generates a graph layout using only the previous layout and difference of the graph. Therefore, the algorithm is categorized as "online." We consider that being "online" is a necessary condition if the proposed method is to be applied to the scale of data of this study.

The algorithm is as follows: let $G_1, G_2, \cdots, G_t$ be a temporally changing sequence of graphs. Let $G_{t-1} = (V_{t-1}, E_{t-1})$ and $G_t = (V_t, E_t)$.

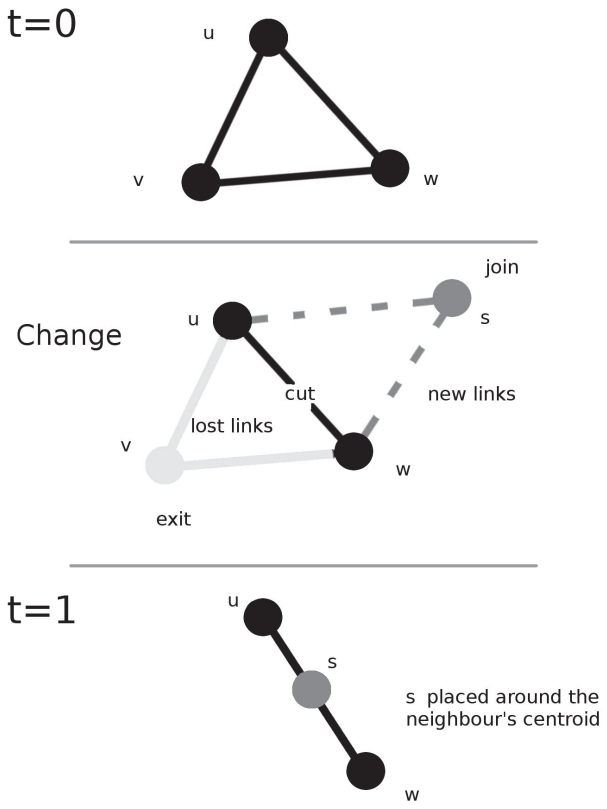

Figure 11: Schematic of placing the newly added vertex by using the layout of retained vertices. The change is as follows: Vertex v is lost; associated edges $(u, v), (w, v)$ are also lost. Vertex s joins with new edges (u, s) and (w, s). Figure from Fig.1 of [33], modified by the author.

The visualization function, f_t, is the function for graph g_t. This function maps each vertex, v, of g_t to the coordinate in a space that is adequate for visualization. For the visualization space we suppose R^3, the three-dimensional Euclidean space. For convenience, we also write $f(V)$ to denote the set of coordinates, $\{f(u) : u \in V\}$ for set V of vertices. Let $adj(G, v)$ be a set of the adjacent vertices of v within graph G. Function $dist(v, V)$ denotes the shortest path length vertex v to reach set V of vertices.

$$dist(v, V) = \begin{cases} 0 & \text{if } v \in V \\ n & \text{if } min\{dist(u, V) : u \in adj(G, v)\} = n - 1 \end{cases} \tag{8}$$

Suppose we have a layout of the previous graph, G_{t-1}, which is given by function f_{t-1}. Let the set of starting vertices be $S = V_t \cap V_{t-1}$. This set contains the vertices of the previous graph that also appear in the new graph. Next, we introduce a centroid function as $Centro(X)$ for set X of the elements of a vector space. Then, the visualization function of graph G_t, denoted by f_t, is given as follows in a recursive manner: Then the visualization function of the graph G_t, denoted by f_t, is given as follows in recursive manner.

$$f_t(v) = \begin{cases} f_{t-1}(v) & \text{if } v \in S \\ Centro\Big(f_{t-1}\big(adj(G_t, v) \cap \{u : dist(u, S) = n-1\}\big)\Big) & \text{if } dist(v, S) = n \\ \text{random coordinate} & \text{otherwise} \end{cases} \quad (9)$$

Figure 11 illustrates how the algorithm works on a simple example.

The initial triangular graph, G_0, which is shown in a triangle layout on "t = 0," contains three vertices $\{u, v, w\}$ and three edges. Its triangular layout is already given by f_0.

Then, the following changes occur in graph G_0: the edge between u and w is lost, vertex v exits from the graph, and new vertex s joins edges (u, s) and (w, s). Hence, the set of starting vertices, S (which is $\{V_0 \cap V_1\}$), is $\{u, w\}$.

On "t=1" in Figure 11, visualization map f_1 keeps the coordinates of remaining vertices u and w, which corresponds to the first conditional line of the definition (Eq. (9)).

The second equation of the definition (Eq. (9)) describes how the coordinate of newly joined vertex s is obtained.

As the distance from s to S is 1, the collection of vertices that determines the centroid is given by $adj(G_0, s) \cap \{v : dist(v, S) = 0\}$, which is the set of remaining vertices $\{u, w\}$ because $adj(G_t, s)$, the neighbor of s is $\{u, w\}$, and the collection of vertices whose has distance of 0 from the set S is the set S itself, which is $\{u, w\}$. Therefore, the centroid of vertices u, w provides the position of new vertex s.

The entire process provides the coordinate of the newly added vertex while maintaining the inherited vertices arrangement, except for the vertices that are unreachable from the set of remaining vertices, S (which is arranged randomly). Definition Eq. (9) can produce successive graph layout from graph difference data and previous layout.

2.7 Discussion

2.7.1 Advantage of DMDS

Figure 12 shows the comparison between DMDS and conventional MDS (edge direction is neglected) combined with the force-directed algorithm with magnetism. The example graph is a Bethe lattice with three neighbors and depth of three. The edges are directed from the root vertex to the terminal leaf vertices. The root vertex is indicated in red.

The upper left figure shows the initial configuration given by DMDS, which already depicts the overall feature of the graph including the edge direction (aligned upward). But the initial layout has a following problem that the dots indicating the leaf vertices actually contain three vertices packed. This degeneracy problem is resolved in the lower figure. Only a minor modification of the packed vertices is sufficient for reaching the final result (lower left) by going only twenty steps of force calculation.

In contrast, the initial arrangement of conventional MDS only shows graph connectivity with no edge direction information. Consequently, the force calculation must solely process the edge direction information. The final arrangement must accommodate the graph connectivity and edge direction. This implies that the layout provided by conventional MDS must be altered almost completely through physical simulation.

Besides necessary computation advantage, more important point is that the chance of producing entangled arrangement glows by applying more physical simulation. An entangled layout violates the identity condition of subsection 1.4.

In practice, DMDS has a major advantage considering the amount of computation and output quality.

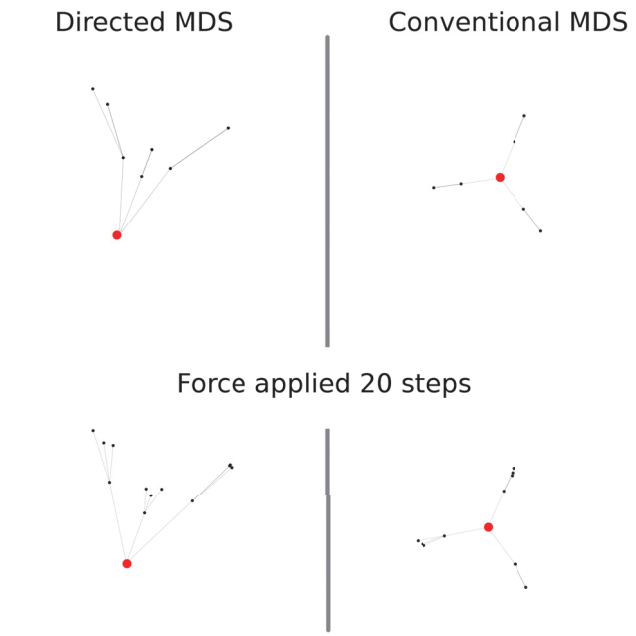

Figure 12: Advantage of DMDS. Same amount of physical force simulation applied to DMDS (left column) and conventional MDS (right column) coordinates. Even though edge-direction information can be shown by magnetism, a good initial layout of the vertices (upper left) considerably reduces required computation. Figure from Fig.4 of [8].

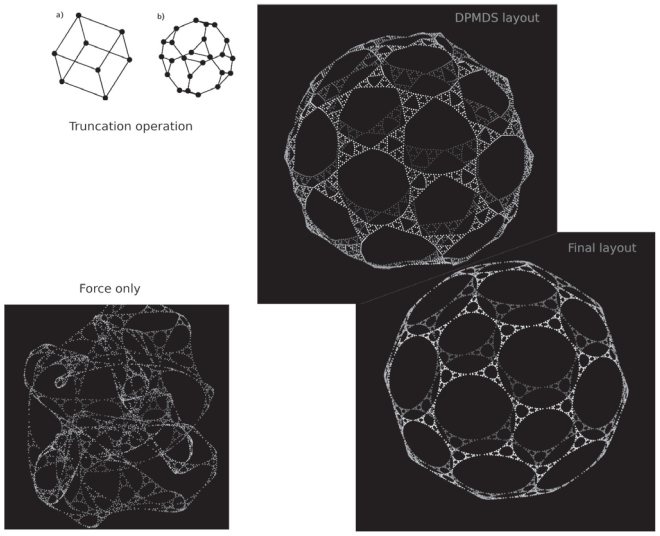

Figure 13: Another advantage of DMDS. The network is a four times truncated Fullerene (C60 molecule) graph with 4860 vertices, 7290 edges, and diameter of 147. Same amount of physical force simulation is applied to DMDS (right) and random initial layout (left). Truncation operation is illustrated in the upper left corner.

DMDS works very well in case of a sparsely connected network with a large diameter. Figure 13 shows the comparison of the initial layout obtained by DMDS and the result obtained using the pure

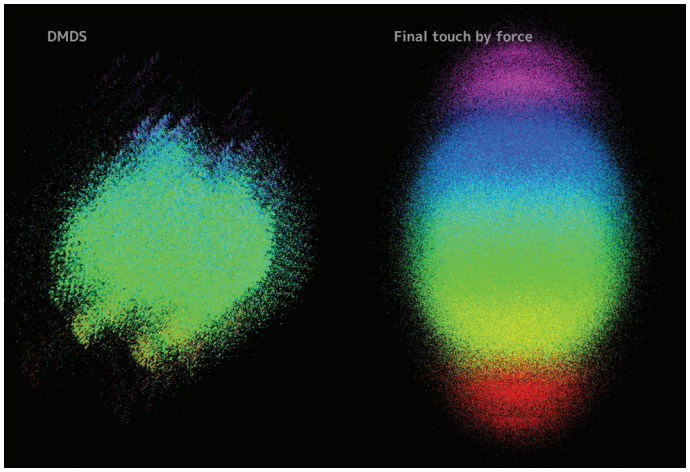

Figure 14: Visualization based on initial layout obtained by DMDS (left) and layout refined by force-directed method. The network is randomly generated by a preferential-attachment model and configured to have a bow-tie structure with power-law degree distribution. Colored to incoming link share value = $\frac{incomming\ edge\ count}{total\ degree}$, continuously from red for value = 0 to magenta for value=1.0.

force-directed method. The network is a four times truncated Fullerene (C60 molecule) connectivity graph. Truncation is the operation of removing a vertex and replacing it with a surface. The DMDS result already shows the entire graph structure. In contrast, the solely applied force-directed method has failed to extract the graph characteristics.

A more realistic case with theoretically generated data is shown in Figure 14. The degree distribution of this network obeys the power law with $\alpha = -1.5$. The data is configured to have the bow-tie structure. The layout shown on the left side of Figure 14 shows the initial layout created by DMDS and the right side shows its refinement by the force-directed method.

It is clearly seen that the DMDS already exhibits a bow-tie like configuration. The most important purpose of DMDS is to show overall structural characteristics, which is successfully accomplished. However, note that the IN and OUT parts have almost disappeared. These are recovered in the refinement process by the force-directed method. If the vertices share their metric vector in the source matrix, they will share the coordinates, too. This is a typical shortcoming of MDS, which likely to occur in the peripheral tree-like part of the network.

2.7.2 DMDS Limitations

The edge orientation in DMDS is known only after the layout is obtained. The first eigen vector (the X-axis) usually retains the edge direction information, but it is also known that this is not always the case. Before applying magnetism, we must know the edge direction of the DMDS layout because the edge direction must adjusted to the applied magnetic field direction.

At present, we calculate each edge direction vector[6] and summarize these vectors as a so-called global edge direction vector. Then rotate the layout to adjust the global edge direction vector to be parallel to the edge direction of the final layout.

2.7.3 Continuity and benefit of the successive graph drawing algorithm

The algorithm introduced in subsection 2.6 was developed to produce smoothly changing visualization from successively evolving network. In the previous sentence I purposely avoid to use the word “continuous” because the word “continuous” has some mathematical implication. To show

[6] a vector from source vertex to the destination vertex of the edge

that the algorithm described in subsection 2.6 is a continuous function from the set of networks to the set of layouts in mathematical context is the best.

But all the data examined in this study is existing real network or theoretically constructed network to reproduce properties of real network. Mathematical concept of continuity is usually given as follows: let g, g' be networks, $f(g)$ be a network layout of g.

If the visualization generating function f satisfies the condition

$$g' \to g \Rightarrow f(g') \to f(g), \tag{10}$$

then the function f is *continuous*.

Discussing mathematical implication of the term "continuity" in the context of Eq. (10) is not very useful. This is because all function whose domain is finite discrete mathematical entity is necessarily continuous. It means we have to device alternative concept of continuity to do useful discussion. The detailed examination of indiscriminate continuity is given in Section 3.3 of [33].

To save the unsuccessful continuity discussion above, an experiment to show that the proposed algorithm actually produces smoothly changing layout from successively changing network is incorporated in Section 3.3 of [33], which is repeated as Figure 15:

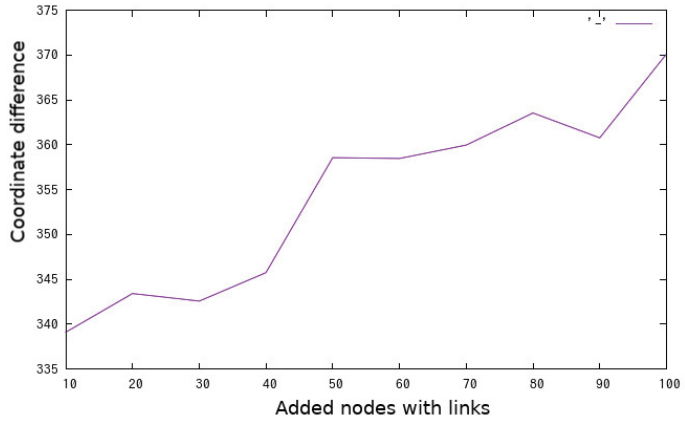

Figure 15: The vertical axis is the sum of layout differences $\sum_{i=1}^{n}(f_{t-1}(v_i) - f_t(v_i))^2$. Horizontal axis shows the number of randomly added nodes, which means the difference from the original network. Layout of the nodes is altered consistently according to the difference of the network except for irregularities at 30 and 90 nodes.

The network of the experiment of Figure 15 has 8,689 nodes and 40,127 links. IN, SCC and OUT segment size are 1,312, 160 and 1,192 respectively. The plot shows that the more the network is altered, the more change happens in the layout.

In the actual case to treat the real network which evolves along with time, the most beneficial properties of the proposed method is that

1. the layout is deterministically obtained incrementally (using only the preceding coordinates) and

2. by using relatively limited computational resources.

One of the alternative and more simple way to process a series of evolving network data is to simply produce each layout from the data, using previous coordinates. This approach actually works almost equally to the proposed method provided that the vertices membership is relatively

stable. It is a simplified version of the proposed method, by omitting the newly joined vertices' coordinate calculation.

Basically there is not much difference between the proposed algorithm and the simplified method described in the previous paragraph, at least in terms of the layout quality. The difference exists mainly in the necessary amount of computation to reach the final result because the simplified method puts new vertices randomly and they have to be moved to adequate position by running physical simulation (or other layout method). The proposed method reduces such computational waste by avoiding unnecessary randomness.

This randomness is unnecessary because most of the case, a new node joins the network by creating a link to the existing node. If we know where to place the new node's neighbours, we can do better than random placement, which is exactly the case when the proposed method is suitable. The discussion of this paragraph also gives the intention behind the proposed method.

2.8 Other methods

Network community detection also plays an important role in this study. A network community is a part of a network that is relatively densely connected compared to the other parts. For our purpose, we require an algorithm to works on a large-scale directed graph.

We use the Infomap algorithm, which was first proposed by [34]. This algorithm optimizes a *map equation* function to separate a network into parts. It is a flow-based method and operates on (supposed) traffic dynamics of a network.

The algorithm is supposed to work very well with directed networks. It can also detect modules within tightly connected clusters (multi-level clustering) or the optimal number of nested modules (multilevel clustering) in a hierarchical manner [35].

For more information see the original papers and references therein. We employ the code written by the original authors of the algorithm, which is optimized for parallel computing.

3 Analysis

In this chapter we analyze the real network data from several heterogeneous domains: WWW link structures, the Japanese production network between more than a million firms and symbol definition relations of a computer programming language. The bow-tie structure, which is found in all the directed network data analyzed in this study, have been widely utilized in the analysis of numerous different networks. We will see that this structure shows conspicuous diversity among different domains, which was not known in the previous works.

First, we examine this fundamental concept using the advanced visualization method introduced in section 2 and show its diversity of the "bow-tie" structure. Based on qualitative visual understanding, we further conduct quantitative analysis on their connectivity structures and explore the self-similarity of large complex networks. Among the examined networks, Japan's production network (actual economic network) shows high self-similarity, and its bow-tie structure contributes to the coexistence of economic stability, consistency, and flexibility.

3.1 Bow-tie structures of the Web

As the concept of bow-tie structure was originally conceived on a web-link network, the WWW link structure is a good starting case to apply the directed-network visualization method described in subsection 2.4. Data are obtained from Stanford Large Network Dataset Collection ([36]: hereafter referred to as SNAP). SNAP provides a wide range of network data including four WWW hyperlink data sets: Google web, Stanford, Berkeley-Stanford, and Notre Dame data sets. The data are provided as a list of links, which are pairs of nodes with no further information.

Table 1: Web data component size

Data	Nodes	Links	SCC	IN	OUT
Google	855,802	5,066,842	434,818	180,902	165,675
Stanford	281,903	2,312,496	150,532	32,785	67,516
Berkeley-Stanford	685,230	7,600,595	334,857	159,709	124,974
Notre Dame	325,729	1,497,134	53,968	0	271,761

The basic statistics of the four data sets are shown in Table 1. The Notre Dame data set lacks the IN segment, and the other three data sets have the complete bow-tie structure. The SCCs of the Google web, Stanford, and Berkeley-Stanford data sets are the largest. However, it is not clear if the listed SCC of the Notre Dame data set is the largest. Fortunately, as it is not very difficult to list all SCCs for this size of networks, it is confirmed that the segment of 53,968 nodes is actually the largest SCC. The degree distributions of these four networks are plotted in Figure 16.

The bow-tie structure is assumed to characterize an entire network, even though this is not explicitly stated in [1] and following studies. It is actually valid to say that in terms of pure connectivity, the bow-tie structures listed in Table 1 are network-wide features because they are the largest.

Figure 17 shows the result of the application of DMDS and the final modification by the hierarchical force calculation algorithm to the Google web data set. It is clearly seen that there are multiple small bow-tie structures. In addition, each small bow-tie structure consists of network communities, i.e., a part of a network that is relatively densely connected.

Even though the concept of the bow-tie structure is thought to be a network-wide property, the visualization shows that it is an accumulation of locally limited small structures. Hence, the schematic understanding of this structure should be updated, as shown on the left side of Figure 18.

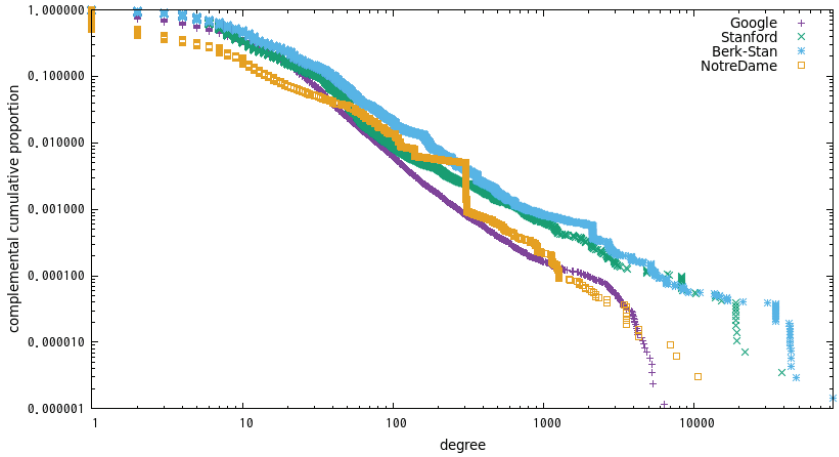

Figure 16: Degree distribution of four types of web data. Degree is indicated on horizontal axis in logarithmic scale and complementary cumulative proportion is indicated on vertical axis in logarithmic scale. Google web data are denoted by purple "+," Stanford data are denoted by green "x," Berkeley-Stanford data are denoted by pale blue "*," and Notre-Dame data are denoted by orange "□." Figure from Fig.2 of [37].

Before performing further analysis, we must ensure that this feature does not come by accident. For this purpose, we compare the community size distribution of the Google web data set and its randomized control because the local bow-tie structures are closely related to network communities.

As shown in Figure 19, the community size distribution of the Google web data set is clearly different from the control (randomized network result). We can now safely conclude that the local bow-tie structure of Google web data does not come by accident.

SNAP provides three more web link data sets. In the following figures, we check whether other web link networks share similar bow-tie locality.

Three more WWW link networks share the localized bow-tie feature.

Previous studies found a segmented bow-tie structure named "daisy" or "teapot" because peripheral segments were separated into several disjoint parts([38], [39], [40]). However, it was not explicitly stated that the SCC is also scattered over a network.

3.2 Sparsely unified SCC

The SCC has a stronger connectivity condition compared to the undirected graph's connectivity condition (see breadth-first search subsubsection 1.1.1). The existence of mutual back-and-forth paths looks like a stronger connectivity. However, there can be a case that this condition is not significantly different from the most sparse undirected (symmetric) connectivity.

To show that SCC condition is not much different from undirected connectivity, we consider the minimal construction of the SCC for n given vertices and compare it to undirected connectivity, as follows: Undirected connectivity is well known as the minimum spanning tree, which requires $n-1$ edges. The SCC condition can be satisfied by adding just one more edge if the vertices are connected in a circular topology.

This is an extreme case of the minimal SCC construction, which is unlikely to occur in naturally produced real data. Nonetheless, it is useful to point out that the SCC condition does not necessarily imply a close and dense connectivity. For example, let us consider a directed loop graph with seven vertices. This loop graph is clearly an SCC, where every pair of vertices has a mutual back-and-forth path. Suppose two vertices of this loop are replaced by SCC graphs, as shown in Figure 22.

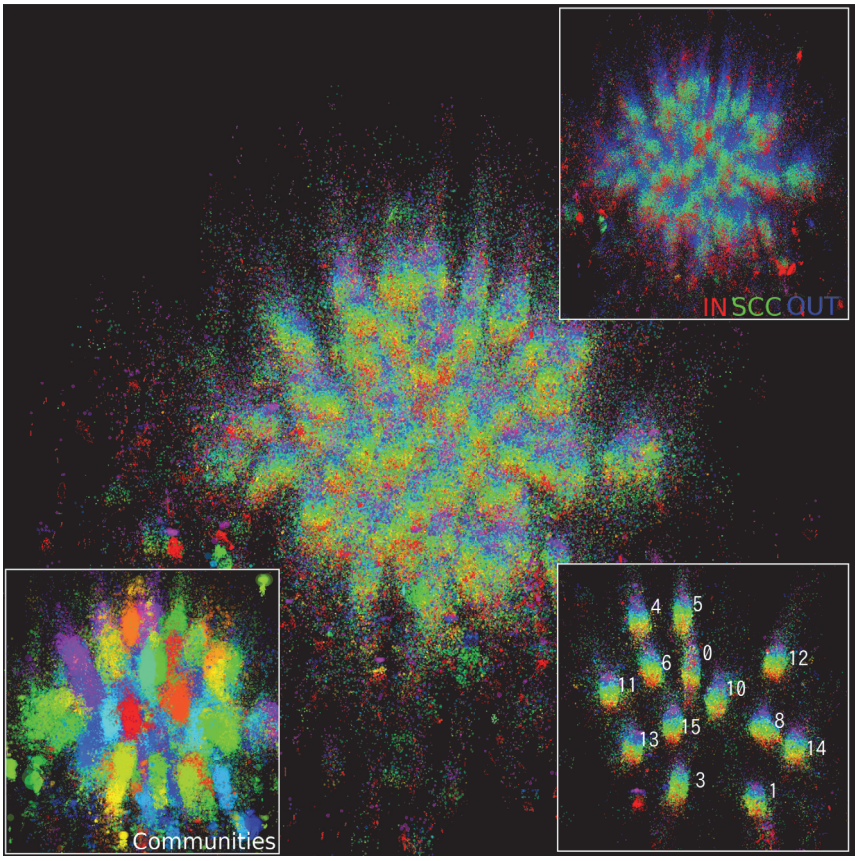

Figure 17: Google web data from SNAP visualization, generated by DMDS and force calculation refinement. The link direction is arranged vertically, facing upward. The main plate is colored to incoming link share value $=\frac{incoming\ edge\ count}{total\ degree}$, continuously from red for value = 0 to purple for value=1.0. Upper right inset shows the IN, SCC, and OUT segments in blue, green, and red, respectively. Lower left inset shows community detection result. Most localized bow-ties are identical to the detected communities. For convenience, several visually disjoint segments are selectively shown in the lower right inset. Figure from Fig.5 of [37]

As seen in Figure 22, any vertex, $v \in S_1$, can reach any vertex ,$u \in S_2$, and vice versa, by following an edge direction. This implies that S_1 and S_2 are unified to a single SCC owing to the backbone loop. If S_1 or S_2 have their own IN and OUT segments, they become the IN and OUT segments of the unified SCC by definition. In other words, a unified (or global) bow-tie structure could be created by loosely connecting multiple disjoint small bow-tie structures.

For the IN and OUT segments of S_1 or S_2 to become global IN or OUT segments, the peripheral segments should *not* be connected to the backbone loop, as this violates the IN or OUT condition. In other words, if multiple bow-tie structures are to be unified into a single bow-tie structure, the IN and OUT segments must be left untouched.

3.3 Bow-tie locality index

Visualization and graph theoretical argument are not sufficient to determine how a localized bow-tie structure is formed in the real network. Further detailed structural and quantitative analysis is required. To find the structural characteristics of the web network connection, the bow-tie locality

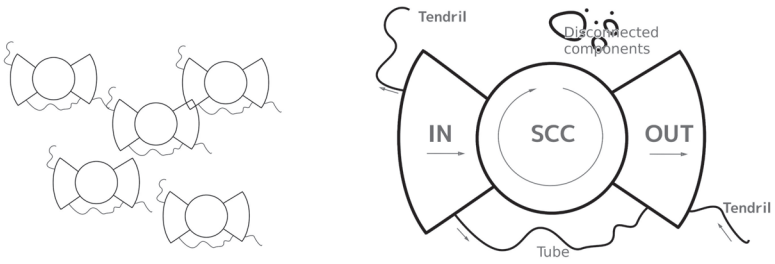

Figure 18: Web link network's bow-tie scheme (left) and conventional understandings (right).

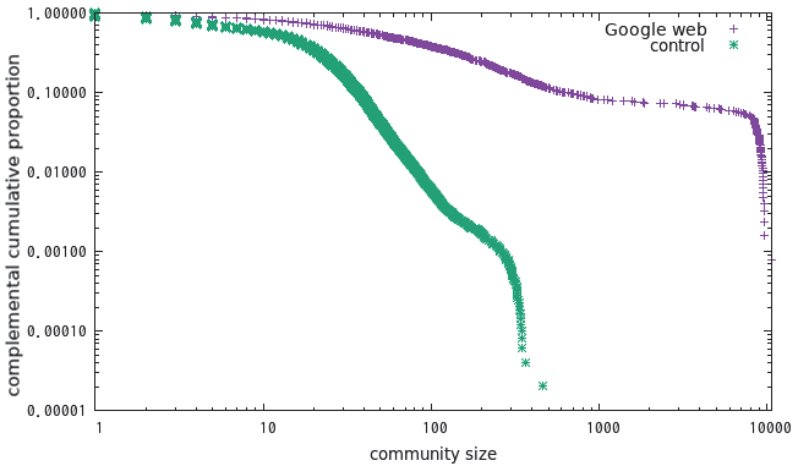

Figure 19: Community size distributions of Google web network (with purple "+" marks) and Google web randomized link (with green "x" marks).

index [7] is developed.

Let G be a graph and G' be a subgraph of G. Let $Out(G')$ and $In(G')$ be the bow-tie segments (if any) of the subgraph G'.

Let $Bf(G')$ be a set of edges that connect from $Out(G')$ to the outside of G'. Similarly, let $Bb(G')$ be a set of edges that connects from $(G - G')$ to $In(G')$. To paraphrase, the set $Bf(G')$ consists of out-going links from the local bow-tie's OUT part and the member of the set $Bb(G')$ is an incoming link from outside to the IN part of the local bow-tie. Letters f and b of Bf and Bb signify "forward" and "backward" bridge, respectively.

The mathematical formulation is as follows:

$$(v_i, v_j) \in Bf(G') \leftrightarrow v_i \in Out(G') \text{ and } v_j \in (G - G') \tag{11}$$

and

$$(v_i, v_j) \in Bb(G') \leftrightarrow v_j \in In(G') \text{ and } v_i \in (G - G') \tag{12}$$

.

Let E'_b be a set of edges of the bow-tie structure of subgraph G'. Then, the bow-tie locality $BTM(G')$ of subgraph G' is

[7] This index was first published in [37] to characterize visually identified locally limited small bow-ties of the web.

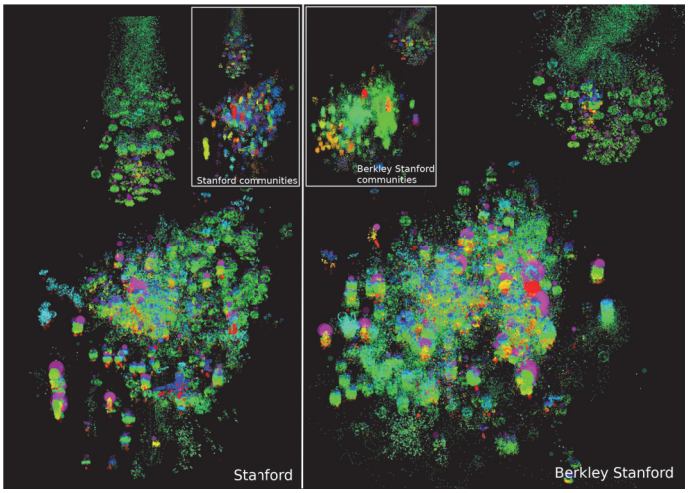

Figure 20: Network visualization of Stanford (left) and Berkeley-Stanford (right) data. Node color scheme is identical to that of Figure 17. Community detection results are shown in the upper corner inset frames. Figure from Fig.7 of [37]

$$BTM(G') = \frac{|Bf(G')| + |Bb(G')|}{|E'_b|} \tag{13}$$

The intention of Eq. (13) is to quantify the contribution of IN and OUT part in the connection between concerning community (which is subgraph G' in Eq. (13)) and its outside. If this index becomes smaller, the IN and OUT part of the community becomes more isolated. There is a similar and more common index named "modularity" to measure the degree of division of concerning part. The difference of bow-tie locality from modularity is that, it specifically quantifies the contribution of connection of flow of concerning part.

The procedure to calculate the bow-tie locality index is as follows: given a network and its subgraph, finding the bow-tie structure of the subgraph is described in section 1, which provides the denominator of Eq. (13). The forward and backward bridges (Bf, Bb) are obtained by forward and backward breadth-first search, which provides the numerator of Eq. (13).

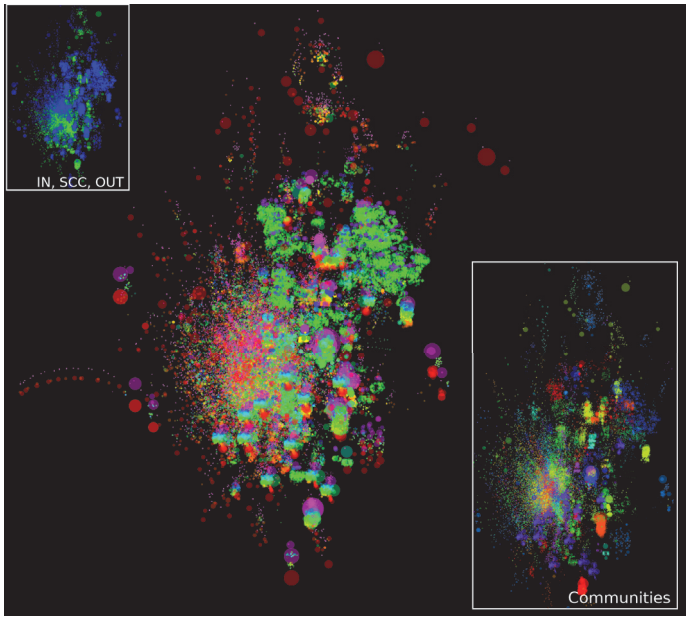

Figure 21: Notre Dame web network visualization with same color scheme as Figure 17. Lower right inset shows community detection result, and upper left shows only SCC (green) and OUT (blue) segments because the network lacks the IN segment. Figure from Fig.8 of [37]

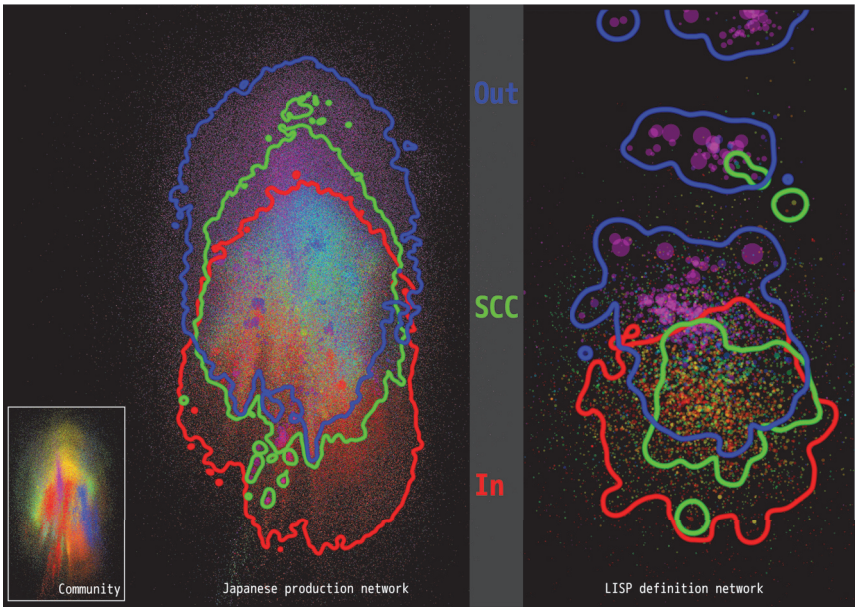

Figure 23: Bow-tie structure visualization of Japanese production network and Emacs LISP network. Link direction upward, main plate color scheme is identical to that of Figure 17. Japanese production network is a collection of supplier–customer relations of individual firms and organizations. LISP network is a relation between the definitions of LISP symbols (object with a unique name). IN, SCC and OUT segments' locations are shown by blue, green and red lines. Lower left inset shows community detection result. Both have this structure as their global features. Figure from Fig.9 of [37]

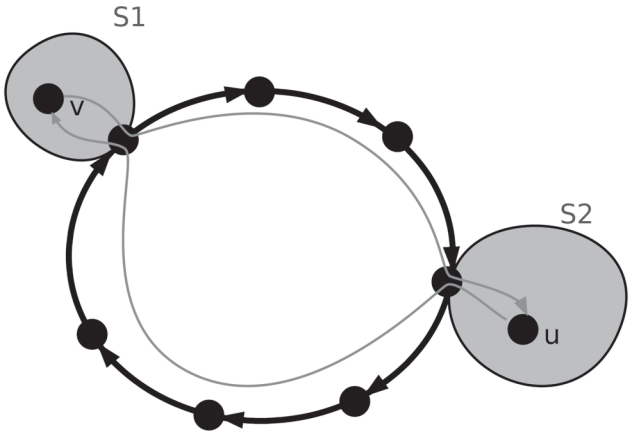

Figure 22: Two SCCs (S_1 and S_2) are unified to a single SCC by backbone loop. All node pairs, $v \in S_1$ and $u \in S_2$, have has forward and backward paths between them (dark gray curved arrow). Consequently, the entire graph shown in the figure consists of a single SCC.

For comparison, directed networks from other domains are visualized in Figure 23. These two networks have single bow-tie structure that covers the whole network. As seen from the bottom left inset, community structure can be found within this single bow-tie.

Based on the graph theoretical discussion in Section subsection 3.2 and the visualization results of Figure 17, Figure 21 and Figure 20, we can predict that the web network communities have lower bow-tie locality index values.

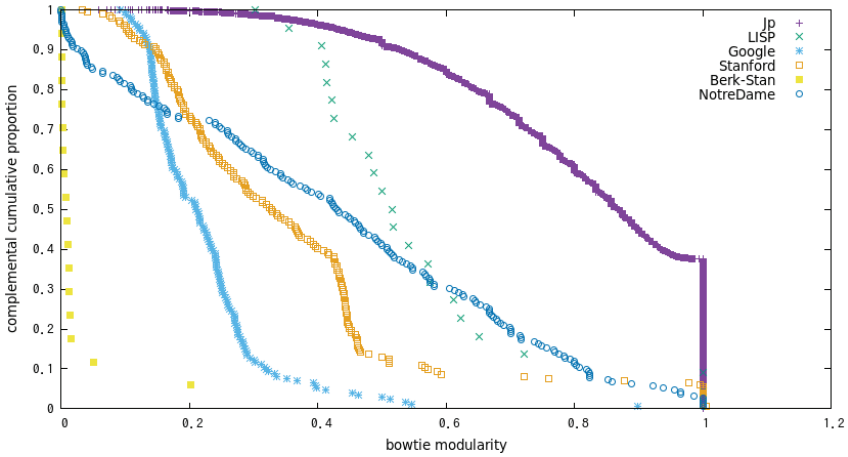

Figure 24: Bow-tie locality of the four web networks, Japanese production network and Emacs LISP symbols' definition network. Figure from Fig.12 of [37]

As predicted, Figure 24 shows that all three web link networks have communities with consistently smaller values of the bow-tie locality index [8]. Now we can conclude that all four web link datasets from SNAP have locally scattered bow-tie structure, in which densely connected local bow-tie structures are loosely joined, and the peripheral IN and OUT segments of the each local

[8] We can see that more than 30% of the Japanese production network's cluster has locality value of 1.0. This happens because of the very small size communities (# of nodes less than 3).

bow-ties are ended with isolated terminal nodes.

This connectivity trait implies a certain degree of self-similarity, but only in a limited sense, because it depletes the structural similarity between the sub network and the whole network: IN and OUT segments of the local bow-ties are not connected to the outside segment. That is, it is extremely difficult to determine the overall feature of WWW networks by exploring its subparts. In contrast, as shown in the next section, the structural characteristics of the production network of Japan differs from the WWW networks. The production network of Japan is more consistently self-similar, and each industrial sector has its core and peripheral segments, whose structural characteristics is similar to the global bow-tie's core and peripheral segments.

3.4 Japanese production network

In this subsection 3.4 the bow-tie structure of the Japanese production network and its economic implications are discussed.

As seen in subsection 3.3, unlike web networks, the company network of supplier-customer relations are more integrated, and each community is more closely connected with each other, which we could conclude from the distribution of the bow-tie locality index seen in subsection 3.3. Based on this conclusion, we will further analyze three main parts of the bow-tie structure and the time-series network data, and find out how two seemingly incompatible properties of economic flexibility and stability are realized within single network.

Data are obtained from the "Kigyo Soukan" database of Tokyo Shoko Research, Ltd. [41]. It houses 1.1–1.2 million companies and other organizations with approximately five million of supplier-customer relations. Furthermore, it is worth noting that the network data of 2011 and 2016 are used to create a time-series network. Table 2 and Figure 25 show basic statistics of the network data of 2011 and 2016 .

Year	Nodes	Links	α
2011	1,131,142	4,965,418	1.51
2016	1,234,687	5,481,427	1.48

Table 2: Statistics of 2011 and 2016 networks. Third column shows exponent value α from Eq. (1), which appears in section 1

	overall	IN	SCC	OUT
2011 nodes	1,131,142	309,847	542,350	224,915
2011 links	4,965,418	72,500	3,476,379	31,871
2016 nodes	1,234,687	337,916	575,686	263,986
2016 links	5,481,427	73,530	380,2497	39,404

Table 3: Bow-tie size, year 2011 and 2016

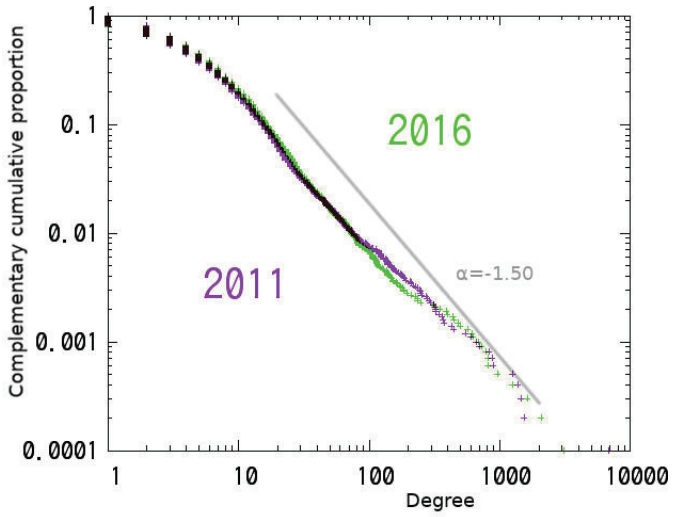

Figure 25: Degree (number of links associated with a node) distribution comparison of 2011 and 2016 networks. The plot shows complementary cumulative proportion of both years. Both degree distributions obey power law, with α as 1.51 in 2011 (purple) and 1.48 in 2016(green). $\alpha = 1.5$ fitting line is added for convenience. Figure from Fig.2 of [33]

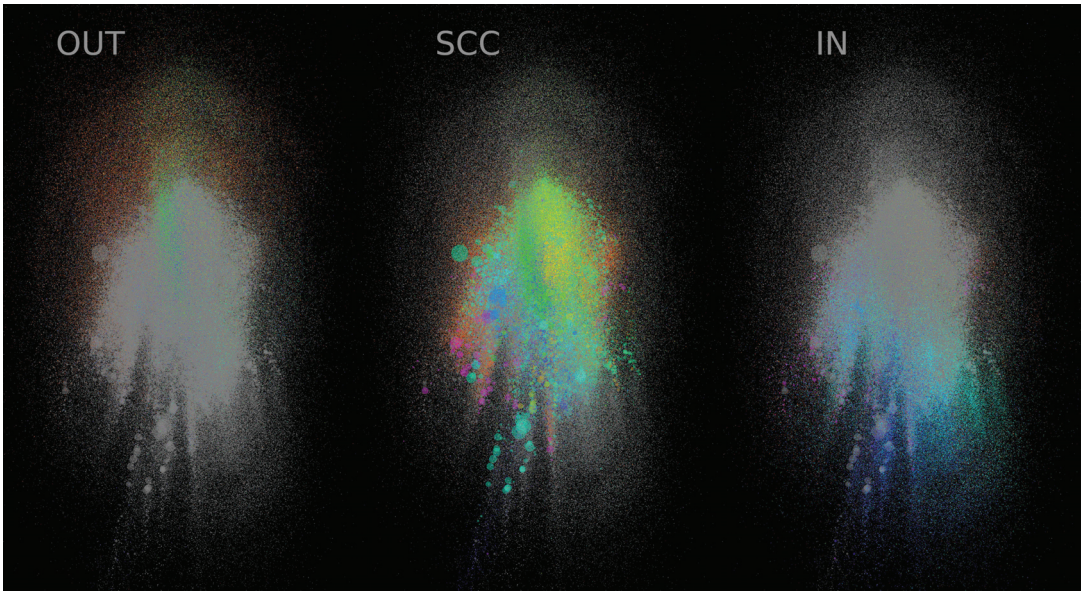

Figure 26: "Bow-tie" structure visualization of Japan's production network for 2016. Node size to the number of links. From left to right, IN, SCC, and OUT parts are selectively colored according to the industrial sectors; other parts are indicated in gray. The edge stands for supplier-customer relation, from supplier to customer. These relations are visualized vertically upwards, which means money goes down and goods goes up. IN, SCC, and OUT have different industrial sector configuration.

Figure 26 is another visually confirmed bow-tie structure of the Japanese production network. However, the shape of the structure does not particularly look like "bow-tie", like Figure 1. It may be better to devise an alternative name such as "walnut", which is proposed in [7]. Figure 26 shows the different configuration of industrial sectors among the segments of bow-tie. These differences

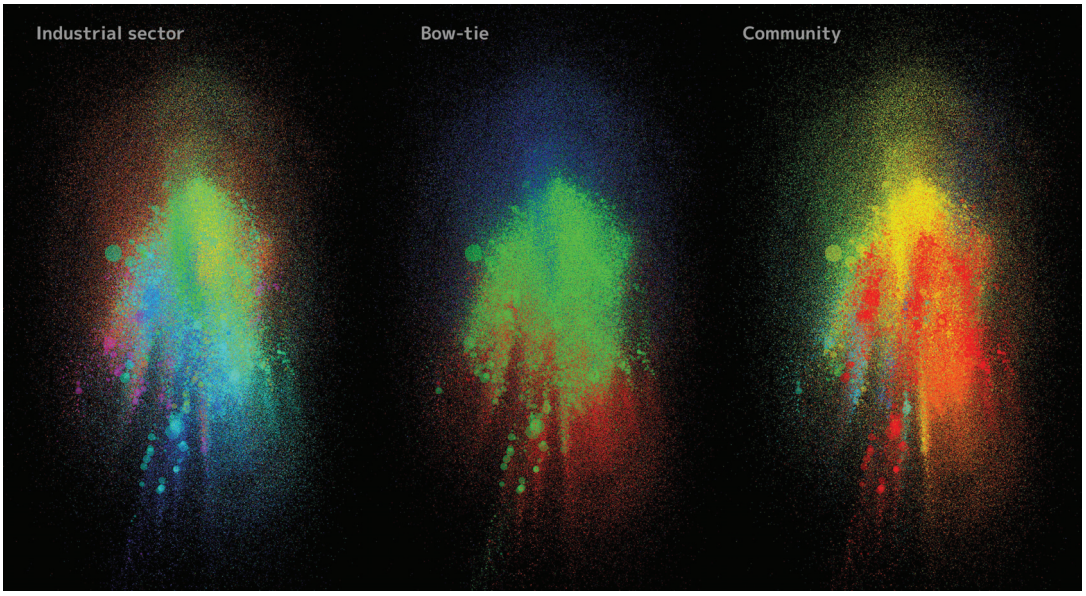

Figure 27: “Bow-tie” structure visualization of Japan’s production network for 2016. Node size and edge direction visualization schema are identical to those of Figure 26. In left figure the nodes are colored by the industrial sector each firm belongs to. Center figure colored by the IN, SCC or OUT segments, blue, green and red respectively. Right figure colored by the network community detection result. The companies tends to form community according to their industrial sectors, as well as each industrial sector contains their own IN-SCC-OUT parts.

are quantitatively studied in [42] for the same type of data for 2006.

Economy is known to show various transformation and development along with time: there are several periodic changes known as business cycles and there is a relatively long term monotonic change which are referred to as trend, and there is non-welcome short term rapid change called crisis.

Such a development and transformation is among major subjects of economics. By virtue of network analysis method advancement and data availability, time-dependent development of inter-firm relationship has been gathering attention as the origin of economic evolution. In [43], by using large-scale firm network data of Japan as this study, the relationship between the inter-firm transaction network evolution and firm growth is studied to reveal that firm ’ s sales increase is positively related with the transaction network growth. They suggest that the observed relationship between firm age and firm growth may be due to the lifecycle pattern of the companies’ trading network.

In this study I apply the successive visualization method described in subsection 2.6 to the data of 2011 and 2016 to find out which part is more subject to change.

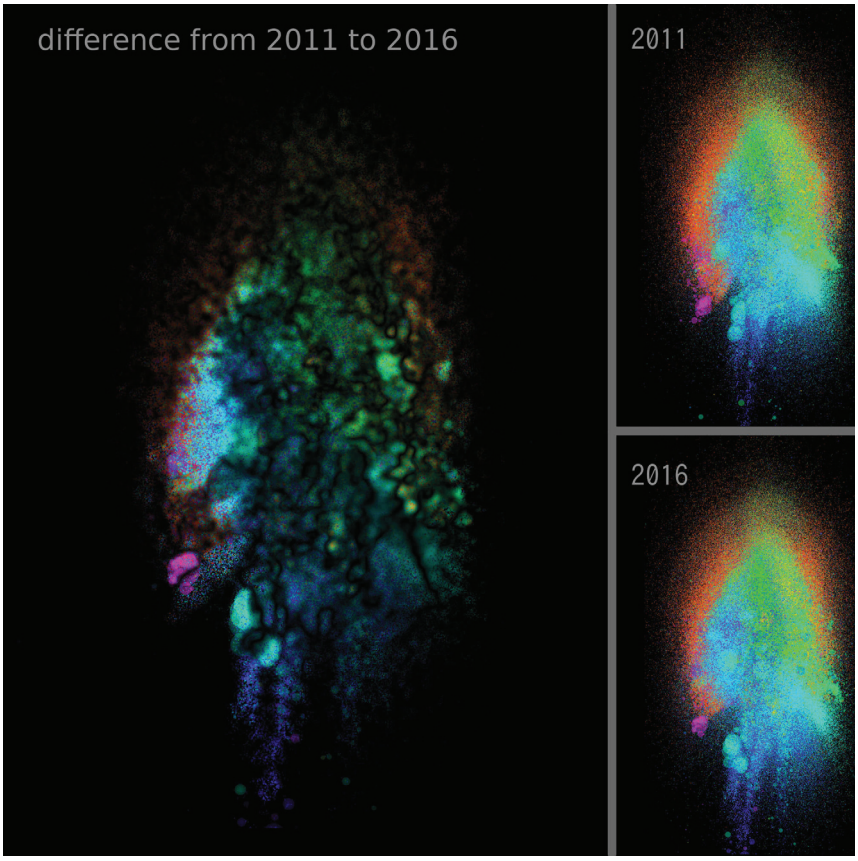

Figure 28: Difference between Japanese production network in 2011 and 2016, visualized using the proposed method. The left part of the figure shows highly different part in bright colors. The central part is relatively stable. Figure from Fig.5 of [33]

Figure 28 shows that the peripheral part of the production network is relatively unstable.

segment	fraction lost in 2016	fraction stayed
IN	0.194	0.667
SCC	0.112	0.818
OUT	0.200	0.606

Table 4: Bow-tie segments' tendency to change

The fraction of nodes that remained or lost from the three main parts of bow-tie are summarized in Table 4. Visually speculated stable membership of SCC and instability of IN or OUT in Figure 28 is quantitatively confirmed.

The IN and OUT segments have lost approximately 20 percent of their nodes in five years. This does not necessarily imply that these nodes went out of business. However, we can say these two peripheral segments are more susceptible to change compared to the SCC segment.

There are nodes that have moved to other segments, for example, from IN to the SCC. The nodes in the IN segment is more likely to remain within the same segment compared to the nodes in OUT, while more than 80 percent of the nodes remain in the SCC. The relocations of the companies is as follows: fifteen percent of the nodes in the OUT segment moved to the SCC, while only ten percent of the nodes in IN have moved to the SCC. The movement from IN to OUT or OUT to IN is rare,

such relocation happens to only one percent of the peripheral nodes.

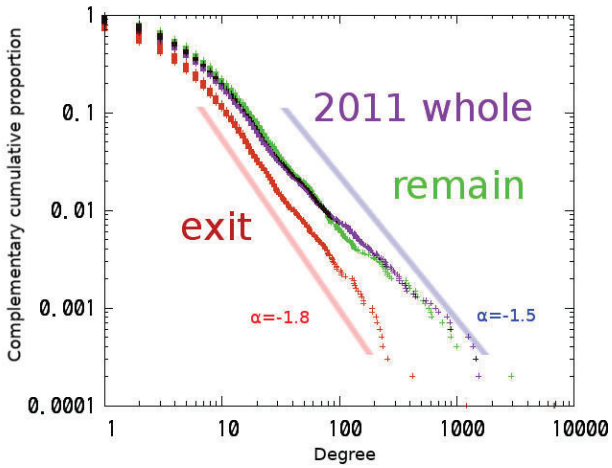

Figure 29: Degree distribution comparison of exited nodes, remaining nodes, and the entire network. All three group of nodes obey the same type of distribution, but the value α is significantly different (exited nodes have a higher value of 1.84 Figure from Fig.6 of [33]). $\alpha = 1.5$ (blue) and 1.8 (red) lines are added for convenience.

Figure 29 shows a comparison of the degree distribution between the nodes that are lost and the nodes that remain. The nodes that exit from the data tend to have fewer links, as predicted from Table 3 and Table 4.

As previously discussed, a company which belongs to the SCC tends to stay the same segment compared to two peripheral segments. However, IN and OUT are not equivalent in terms of membership stability. The possibility to move to the SCC from these segments are significantly different, and OUT marks higher possibility of SCC entering.

As there are other asymmetries between the IN and OUT segments (they also have industrial sectors' constituent difference), the explanation as follows is one of many hypothesis. It is based on the observation of economic transaction relations. For consumers it is often the case that they can arbitrate where to buy products. Stores usually have no choice but to sell their goods to any visiting consumer. This relation does not hold unconditionally in the case of a business-to-business relationship. In quite a few cases, it is difficult for new companies to buy goods from supplier companies because a business-to-business relation is based on credit. Therefore, suppliers never sell goods to a company that does not have credit. Until the settlement, a supplier is also a creditor and a customer is a debtor. This asymmetric relation results in unequal position of suppliers and customers, and suppliers are likely to take advantage of this unequal position.

The asymmetry of the credit relation can also be seen in the difference between the connecting points of the SCC, IN, and OUT. As seen in Figure 30, the OUT part selectively connects to the central part of the SCC, while IN tends to connect to the lower half (IN-side firms) of the SCC. A company must prove its credibility to become a customer. This should be easier for a company in the central part of the SCC.

Companies form network communities according to the industrial sector that they belong to, and these communities have bow-tie structures. The sub-bow-tie structures of a production network are not as isolated as those of web link structures, as seen in subsection 3.3. This is also understood based on the domain-specific knowledge about production networks. For example, iron ore is produced in OUT segment firms. In SCC segment firms, the ore is converted to iron, forged to

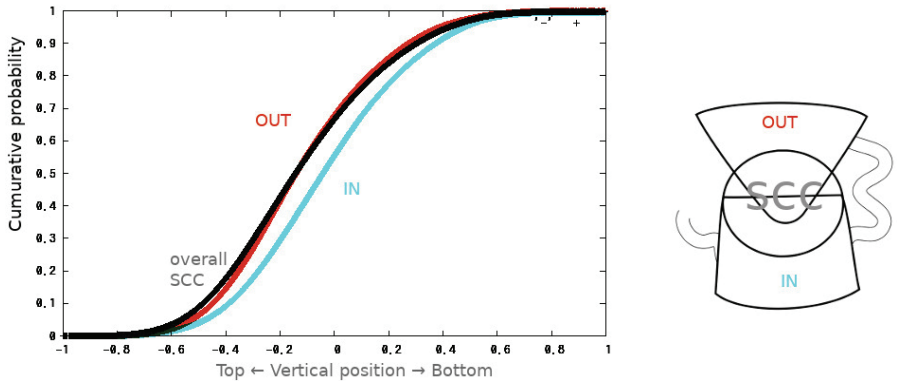

Figure 30: OUT and IN parts are connected to SCC in different manner. To demonstrate the difference quantitatively, the Y-coordinates of SCC nodes where IN and OUT parts are connected are plotted in this chart. IN segment attaching points in the SCC is lower than those of OUT segment. Additionally, OUT segment attaching points tends to gather central part of SCC. More realistic schematic drawing resulting from this analysis is added in the right. The coordinates of all the SCC nodes are plotted for comparison in black color. Figure from Fig.9 of [33].

steel sheets with approximate dimensions, cut and polished to a shape, assembled with other parts, checked, painted, adjusted, and tested. Most of these value-adding processes reside in the SCC. Finally, the product is moved to IN segment firms and delivered to the client. This entire process is realized as flow inside of the bow-tie structure.

If we look at this network from outside of supplier-customer relationships, the production network as real economy is not an isolated system. There are labor markets, natural environments, financial systems, governmental organizations, and consumers. These economic agents interact with each other to construct the whole human society. The production network consists of smaller bow-tie structures as its subsystems. However, these structures are more integrated, not as isolated as the structures of web link networks. To summarize, the production network of Japan has more consistent self-similarity.

The peripheral segments works as an interface to the outside, providing flexibility to the changing social and natural environment, while the core segment is responsible for the integrity and stability of Japan's economy.

4 Conclusion and future work

4.1 Conclusion

There are many different kinds of networks in modern society such as information communication, ecosystems, reaction of substances in living organisms, and transactions between companies. Inside these networks, information, substances, money, products and other things are traveling on complicated routes. These various networks often behave in ways that cannot be understood by simply extracting and observing their individual components. Also, completely different types of networks may share the same properties such as degree distribution and self-similarity.

The origins of complex network study can be traced back to graph theory that began in the first half of the 18th century. Graph theory is an area of mathematics that deals with relationships between objects that are abstracted to lose properties such as shape and size. Graph theory also developed into the leading mathematics fields such as topology. In addition to graph theory, statistical mechanical methods based on the properties of the data itself and methods of computing science are adopted in complex network analysis. To discover the properties and mechanisms which is common to different types of networks is one of the major objective of this interdisciplinary study.

Complex networks generally have very diverse structures, so it is often difficult to get clues for the analysis. In this study, advanced visualization techniques were developed and applied to visually reveal the characteristics of large, complex networks with directed link. Furthermore, based on the results, we designed and executed a quantitative analysis of the structure to clarify the mechanism that works in the network, especially by using the structural feature known as the “ bow-tie ” structure.

To perform these analyses, it is necessary to extract both the macroscopic features and details of the network. According to the conventional standard, the network visualization should satisfy the condition that the number of crossing edges must be small and that a symmetric drawing can be obtained from a symmetric network. However, these conditions are not appropriate to use the visualization as a clue for quantitative analysis. This is because in order for network drawing to be used as a clue for quantitative analysis, both the outline and detailed features of the entire network must be expressed consistently. In this research, instead of conventional conditions like the edge crossing count, the following two conditions are proposed:

1. the same visualization can always be obtained from the same network,
2. and that the visualization of different networks is not the same.

Using these conditions as a guide, two-stage visualization method that satisfies these standards was developed.

The first stage is a method that extends classical MDS which is applicable to directed networks, and always generates the same configuration deterministically from the same network. This first stage can extract the general structure of the network. The second stage uses the results of the first stage as the initial arrangement and performs mechanical simulation to improve the arrangement to reveal the details of the structure. Neither the first stage nor the second stage can be applied naively to actual calculations of the data of the scale analyzed in this study. Therefore, in the first stage the matrix dimensions are reduced to work with networks of millions of nodes. In the second stage, optimization and parallelization are performed so that the number of interactions between vertices are reduced. As a result, the combination of these two methods provided an excellent visualization within reasonable time that well represents both the overall features and the details. Using this visualization, it has become possible to ”understand” by looking at the network structure as Aristotle stated.

The mechanism of network development is one of the fundamental question of complex network study. On the other hand, the method for network development analysis is relatively limited,

except for several network generation models and network propagation theories. This is because analyzing single snapshot of large-scale network is often enough difficult. In this study I devised a visualization method that generates series of smoothly changing figures from successively changing network and applied it to the evolving Japanese production network.

As visualization-driven analysis is not very common, it is useful to find a new question which is not asked within already established disciplines like economics. On the other hand it has its own shortcomings that it is usually difficult to find quantitative (or anything other than visual) analysis that can give answers to the newly found question. But this step is absolutely necessary because we are interested in mechanism or dynamics that works on the complex network.

How and why such a large-scale and complex structures as discussed in this research have developed continues to be one of the central issues in complex network study. The concept of self-similarity is a very important guide to contemplate this issue. This is because the similarity of the structure between the parts and the whole is not only a feature of the structure itself, but also the process and mechanism that generated the structure.

The bow-tie structure was first proposed from the analysis of the web link structure, and the web data discussed in this study also have this structure. However, according to the visualization, it can be seen that a large number of small, locally limited bow-ties are sparsely connected, not a unified bow-tie as a whole. Through quantitative analysis, it could be affirmed that the relationship between local bow-tie structures strongly affects the properties of the entire network system. The local bow-tie structure of the web link network means limited self-similarity and integrity.

In contrast, production network of Japanese companies shows more consistent self-similarity. In addition, time series analysis showed that the central SCC part is more stable while the surrounding part of IN and OUT is susceptible to change. Furthermore, the difference in the economic role of the IN and OUT sections is reflected in the difference in how the two sections are connected to the central part. The important thing is that these parts are tightly integrated to keep self-similarity. Thanks to its characteristic topological structure, production networks have realized two seemingly contradictory properties of flexibility and stability within single network.

4.2 Future work

The future work for large-scale network visualization is to build a product that is easy to use for non-programmers, with the features listed within this study. Current status of the product which was used in this study is a programming library that can be applied to network analysis, which is useful for those who ware familiar with programming, but it may not so for non-programmers.

The web networks analyzed in this study partially lack self-similarity as seen in section 3. But it is unlikely that a huge network like the web will grow without self-similarity. There can be a hidden structure which orchestrates locally scattered bow-ties and complete self-similarity. When we browse the web, we not only need a method to browse the contents, but also a way to know where the contents exist. The knowledge of content location may come from a friend's remark in the social network service, or from a web search. By analyzing such "indexing" links, maybe more consistent self-similarity of the web link structure could be found. However, in recent years, there is a growing tendency that the so-called " open web" is divided by services that enclose users and by deceptive tyrannical nations. Today, it may be the last chance to find out the self-similarity that has created the web. To find out such link structure and self-similarity should be an important, interesting and challenging theme.

From the perspective of mainstream economics, the conclusion that the structure of the supplier-customer network contributes to sustainable and consistent economic growth is not meaningless but carries limited policy implication. The companies' relationship of the employed data of this study is an annually aggregated record. Therefore, each individual relationship may not exist at particular time and date. Accordingly, working on the aggregated relational data will overestimate

the flow and spreading of goods or credit through the relation. Future work on the economic network will be to make up for these shortcomings. Such an improvement will increase the availability of risk assessment methods and bring more economic policy implication.

Acknowledgment

This study started for author ' s doctoral thesis in April of 2017 at the Graduate School of Simulation Studies in the University of Hyogo to contribute to large-scale complex network analysis, especially the development of visualization method of directed networks, and its application to the quantitative structural analysis and its temporal evolution.

First, I would like to thank my supervisor Professor Yoshi Fujiwara at the University of Hyogo for his encouragement, advice and instructions. I also would like to thank for the joint research of the production network of Japan and WWW link structure analysis. I came into complex network study in 2007, which is the year we met for the first time at ATR.

I would like to thank Dr. Atsushi Kawai at OIST for his kind and precise advice on high performance numerical computation, and providing sophisticated optimized physical simulation code, as well as his long lasting friendship.

I would like to express acknowledgment to Professor Hideaki Aoyama at Kyoto University for his research advice, in particular his encouragement on studying bow-tie structure.

I would like to thank Professor Hiroshi Iyetomi at Niigata University for his research advice and guidance, especially the joint research of the analysis of WWW link network, in particular the analysis design of the structural characteristics of localized bow-ties.

I am sincerely grateful to Professor Hiroyasu Inoue at University of Hyogo for his research advice and guidance, especially the advice on the general plan of the research and useful comments on this thesis.

I would like to thank Professor Ulrik Brandes at ETH Zurich for his advice, discussion and insight of pivot-MDS, which is the predecessor of DMDS.

I would like to thank Professor Wataru Souma at Nihon University for his research advice and guidance, especially the concept of self-similarity of this study. Also I would thank for the joint research of production network of Japan and WWW link analysis.

I would like to thank Dr. Yuichi Kichikawa at Niigata University for his research advice and guidance, especially for the concept of locally limited bow-tie structure which he first conceived in the joint discussion with me.

I would like to thank Professor Yutaka Hata at University of Hyogo for his advice and comment on this thesis.

I would like to thank Japan Society for Evolutionary Economics for generous permissions to use the figures of 5, 6, 11, 12, 24, 27, 28 and 29.

I would like to thank SpringerOpen for helpful permissions to use the figures of 15, 16, 19, 20, 22 and 23.

Lastly, I would thank my family, Ayaco and Maya for their support and encouragement of this study and thesis.

References

[1] A. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, J. Wiener, Graph structure in the web, Computer Networks **33**(1-6), 309 (2000)

[2] N.L. Biggs, E.K. Lloyd, R.J. Wilson, *Graph Theory 1736-1936* (Oxford University Press, 1986)

[3] A.L. Barabasi, M. Posfai, *Network Science* (Cambridge University Press, 2016)

[4] E. Kawakami, V.K. Singh, K. Matsubara, T. Ishii, Y. Matsuoka, T. Hase, Network analyses based on comprehensive molecular interaction maps reveal robust control structures in yeast stress response pathways, npj Systems Biology and Applications **2** (2016). DOI 10.1038/npjsba.2015.18

[5] F. Yan, K. Xu, X. Yao, Y. Li, Fuzzy bayesian network-bow-tie analysis of gas leakage during biomass gasification, PLOS ONE (2016)

[6] J. Zhao, H. Yu, J.H. Luo, Z.W. Cao, , Y.X. Li, Hierarchical modularity of nested bow-ties in metabolic networks, BMC Bioinformatics **7**(386) (2006)

[7] A. Chakraborty, Y. Kichikawa, H. Iyetomi, T. Iino, H. Inoue, Y. Fujiwara, H. Aoyama, Hierarchical communities in the walnut structure of japanese production networks, PLoS ONE **13**. DOI 10.1371/journal.pone.0202739

[8] Y. Fujita, Y. Fujiwara, W. Souma, Large directed-graph layout and its application to a million-firms economic network, Evolutionary and Institutional Economics Review **07** (2016)

[9] H. Aoyama, Y. Fujiwara, Y. Ikeda, H. Iyetomi, W. Souma, H. Yoshikawa, *Macro-Econophysics* (Cambridge University Press, 2017)

[10] C. Song, S. Havlin, H.A. Makse, Self-similarity of complex networks **433**(27) (2005). DOI 10.1038/nature03248

[11] Aristotle, *Metaphysics*. No. 271 in Loeb Classical Library (Harvard University Press, Cambridge, MA, 1933)

[12] Which aesthetic has the greatest effect on human understanding?, Lecture Notes Computer Science **1353**, 248

[13] R. Tamassia, *Handbook of Graph Drawing and Visualization* (CRC Press, USA, 2013)

[14] T. Kamada, S. Kawai, An algorithm for drawing general undirected graphs, Information Processing Letters **31**(1), 7 (1989)

[15] R. Hadany, D. Harel, A multi-scale algorithm for drawing graphs nicely., Discrete Applied Mathematics **113**(1), 3 (2001)

[16] A. Quigley, P. Eades, Fade: graph drawing, clustering, and visual abstraction, Proceedings of the 8th Symposium on Graph Drawing(GD) **1984 of Lecture Notes in Computer Science**, 197–210 (2000)

[17] Y. Hu, Efficient, high-quality force-directed graph, The Mathematica Journal **10**(1), 37 (2006)

[18] E.R. Gansner, E. Koutsofios, S.C. North, G.P. Vo, A technique for drawing directed graphs, IEEE TRANSACTIONS ON SOFTWARE ENGINEERING **19**(3) (1993)

[19] Y. Fujiwara, M. Terai, Y. Fujita, W. Souma, Debtrank analysis of financial distress propagation on a production network in japan, RIETI DISCUSSION PAPER **Series 16-E-046** (2016)

[20] K. Sugiyama, K. Misue, Graph drawing by the magnetic spring model, Journal of Visual Languages & Computing **6**, 217–231 (1995)

[21] W.S. Torgerson, *Theory & Methods of Scaling* (Wiley, 1958)

[22] G. Young, H.A. S., Discussion of a set of points in terms of their mutual distances, Psycometrika **3**, 19 (1938)

[23] W. Rudin, *Principles of Mathematical Analysis* (McGraw-Hill, 1976)

[24] U. Brandes, C. Pich, Eigensolver Methods for Progressive Multidimensional Scaling of Large Data, Proceedings of 14th Symposium on Graph Drawing (GD) pp. 42–53 (2007). Lecture Notes in Computer Science 4372

[25] J. Barnes, P. Hut, A hierarchical O(N log N) force-calculation algorithm, Nature **324**(6096), 446 (1986)

[26] H. Sagan, *Space-Filling Curves* (Springer, 1994)

[27] J.K. Salmon, G.S. Winckelmans, M.S. Warren, Fast parallel treecodes for gravitational and fluid dynamical n-body problems **8**, 129

[28] J. Makino, M. Taiji, *Scientific Simulations with Special-Purpose Computers* (John Wiley and Sons, Chichester, UK, 1998)

[29] A. Kawai, T. Fukushige, in *Proceedings of the SC06 (High Performance Computing, Networking, Storage and Analysis) CDROM* (2006)

[30] A. Tanikawa, K. Yoshikawa, K. Nitadori, T. Okamoto, Phantom-grape: numerical software library to accelerate collisionless n-body simulation with simd instruction set on x86 architecture, arXiv.org/astro-ph/arXiv:1203.4037 (2012)

[31] P. Morreale, A. Goncalves, C. Silva, *Modeling and Processing for Next-Generation Big-Data Technologies. Modeling and Optimization in Science and Technologies* (Springer, 2015), vol. 4, chap. Analysis and Visualization of Large-Scale Time Series Network Data

[32] Y. Frishman, A. Tal, Online dynamic graph drawing, IEEE Transactions on Visualization and Computer Graphics **14**(4) (2008). DOI 10.1109/TVCG.2008.11

[33] Y. Fujita, Y. Fujiwara, W. Souma, Macroscopic features of production network and sequential graph drawing, Evolutionary and Institutional Economics Review **16**(1), 183 (2019). DOI 10.1007/s40844-019-00123-7. URL `https://doi.org/10.1007/s40844-019-00123-7`

[34] C.T.B. Martin Rosvall, Maps of random walks on complex networks reveal community structure, PNAS **105**(4) (2007)

[35] C.T.B. Martin Rosvall, Multilevel compression of random walks on networks reveals hierarchical organization in large integrated systems, PlosOne **6(4)** (2011). DOI 10.1371/journal.pone.0018209

[36] J. Leskovec, A. Krevl. SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data` (2014)

[37] Y. Fujita, Y. Kichikawa, Y. Fujiwara, W. Souma, H. Iyetomi, Local bow-tie structure of the web **4**(15) (2019). DOI 10.1007/s41109-019-0127-2

[38] J.J. Zhu, T. Meng, Z. Xie, G. Li, X. Li, in *Proceedings of the 17th international conference on World Wide Web* (ACM, 2008), pp. 1133–1134

[39] R. Meusel, S. Vigna, O. Lehmberg, C. Bizer, in *Proceedings of the 23rd international conference on World Wide Web* (ACM, 2014), pp. 427–432

[40] R. Meusel, S. Vigna, O. Lehmberg, C. Bizer, The graph structure in the web: Analyzed on different aggregation levels, The Journal of Web Science **1**(1), 33 (2015)

[41] TOKYO SHOKO RESEARCH, LTD. (2016). `http://www.tsr-net.co.jp`

[42] Y. Fujiwara, H. Aoyama, Large-scale structure of a nation-wide production network, European Physical Journal B **77(4)**, 565 (2010)

[43] D. Fujii, Y.U. Saito, T. Senga, The dynamics of inter-firm networks and firm growth, RIETI Discussion Paper **17**(110) (2017)

A Appendix

Essential part of the network analysis code and its usage is included as an Appendix. Breadth-first search method, SCC pickup method, and DMDS with pivot optimization methods are listed. All the source code is available at

```
https://github.com/fjt/ngraph
```

.

Everything is provided as Ngraph class and its methods. Ngraph instance provides network analysis methods like breadth-first search or visualization. The visualization takes two steps: vertices coordinate calculation and the picture rendering.

```
class Ngraph
  def bfs(stt, dir=:bf, done={}, &blk) ## variable done is a hash to handle bookkeeping
    if stt.class != Array
      stt=[[stt]]
    else if stt.first.class != Array
           stt=[stt]
         end
    end

    stt.last.each{|e|done[e]=true}

    case dir
    when :b ## backward breadth-first search
      nxt=stt.last.map{|v|self.hailist[v]}.flatten.uniq.map{|e|e if not done[e]}.compac
    when :f ## forward breadth-first search
      nxt=stt.last.map{|v|self.derulist[v]}.flatten.uniq.map{|e|e if not done[e]}.compa
    else ## undirected bfs
      nxt=stt.last.map{|v|self.tonalist[v]}.flatten.uniq.map{|e|e if not done[e]}.compa
    end

    if nxt.length == 0 or (blk.call(stt) if blk)
      stt
    else
      bfs(stt.push(nxt), dir, done, &blk)
    end
  end

  def scc(stt) ## follows exact procedure described in the Introduction
    (self.bfs(stt, :f).flatten & self.bfs(stt, :b).flatten).uniq
  end

  def dpmds(plist, opt=nil) ## given plist designates pivots
    if opt and opt[:dim]
      dim=opt[:dim]; p dim
    else
      dim=3
    end
    en=self.vertex.length
    eg=self.edge.length
```

```
    dmax=(Math::log(en)/(Math::log(2*eg) - Math::log(en))).ceil * 2
    mtx=plist.map{|v| ## pivot-wise vector obtained here.
      row=Array.new(self.vertex.length, dmax)
      sp=self.bfs(v)
      goout=self.dbfs(v)
      incom=self.rdbfs(v)
      sp.each.with_index{|d, i|
        d.each{|vi|row[vi]=i}
        (goout[i]&d).each{|vi|row[vi]=2*i} if goout[i]
        (incom[i]&d).each{|vi|row[vi]=2*i} if incom[i]}
      p "#{v} done"
      row}
    p 'mtx done'
    pmds(plist, mtx, opt) ## Young-Householder translation and eigen decomposition.
  end
end
```

DMDS sample session listed as follows:

```
irb(main):001:0> require 'ngraph'
true
irb(main):002:0> fl=Ngraph.fullerene
#<Ngraph:0x000055f89104d028 @settings={:tree_vector_param=>450}, @vertex=[0, 1, 2, 3, 4, 5,
irb(main):003:0> fl.bfs(0)
[[0], [1, 4, 5], [2, 6, 3, 9, 10, 19], [7, 18, 17, 8, 12, 11, 22, 20], [16, 15, 38, 36, 14,
irb(main):004:0> plist=[0]
[0]
irb(main):005:0> 10.times{plist.push(fl.bfs(plist).last.sample)}
10
irb(main):006:0> plist
[0, 56, 34, 28, 25, 39, 3, 14, 19, 46, 22]
irb(main):007:0> fl.dpmds(plist)
[[1.9811856891225792, 0.28408251039871857, 0.4478090305273955], [1.074146485771026, 0.172341
irb(main):008:0>
```

Pivot list of size 11 is obtained by repeating breadth-first search 10 times and pick up most distant vertex at line 005. DPMDS works with this pivot list given as its argument and returns the coordinates. DPMDS method obtains the layout coordinates, on line 007.

Preferential attachment complex network generation example. It can designate the degree distribution power-law parameter. By using this example we'll show how to perform bow-tie analysis.

```
irb(main):012:0> nt=Ngraph.prefa(size:20000, power:1.5)
#<Ngraph:0x000055f891366fe8 @settings={:tree_vector_param=>450}, @vertex=[0, 1, 2, 3, 4, 5,
irb(main):013:0> nt.edge.length
29157
irb(main):014:0> nt.vertex.length
20000
irb(main):015:0> nt.tonalist.map{|e|e.length}.mle_power(10)
```

```
1.5452854796620181
irb(main):016:0>
irb(main):016:0> nt.scc(0)
[0, 1, 24, 167, 2, 195, 29, 4, 317, 457, 883, 54, 158, 73, 61, 2429, 2075, 17, 25, 60,
irb(main):017:0> nt.scc(0).length
3392
irb(main):069:0> nt.bfs(nt.scc(0), :f)
[[0, 1, 4, 7, 46, 232, 248, 487, 865, 268, 1328 (snip)
```

Above example generates network object with degree obeys power law cumulative distribution of $\alpha = -1.5$, which is verified at line 15. A network of this degree distribution has SCC segment, which is verified on line 017: its size is 3392. At line 069, breadth-first search forward is performed from SCC segment, which will allows you to pick up OUT segment. The returned value is an array of vertices indices and the first element is the starting vertices set, which is SCC in this case. Consequently by doing as follows

```
irb(main):069:0> nt.bfs(nt.scc(0), :f)[1..-1]
```

you will obtain the set $\cup_{i=0}^{\infty} B_i$ defined in subsubsection 1.1.1. To get IN segment, you feed a symbol ":b" to the second argument of bfs method.

If the degree distribution power is 2, overall degree count will be $2n$, which means edge number is n; this is a minimum spanning tree with power-law degree distribution.

Thee network analysis class has a visualization generating method, which was used to generate pictures of this study. Vertices layout coordinate can be accessed as follows

```
irb(main):107:0> nt.pos=nt.dpmds(plist)
"0 done"
"56 done"
"34 done"
[[-0.0427758788196128, -0.25768952405682444, 0.03855704247872866], [-0.071780452672836
irb(main):108:0> nt.drawc
#<Cairo::ImageSurface:0x000055f89e2421a0>
irb(main):109:0>
```

You can designate the vertices' color, size, label by giving anonymous function as a block variable that gives color, size and so on. Following example gives vertices' size to the square root of corresponding degree.

At line 108 method "drawc" is called to generate picture. This will generate picture to file (PNG file by default). You can designate size, scale, edge drawing on/off and vertices index list you choose to draw as an argument of this method.

While these Ruby code implements network analysis processes, the code of "libvtc.c" written in C provides wrapping functions from GRAPE data structures and force calculation functions to Ruby objects. Additionally it also provides processing features related to graph edges. The force calculation process is handed over to GRAPE functions from Ruby by the native codes written in C. It means most part of force calculation is executed in native code of GRAPE. Ruby only performs data formatting for input and output, which can be achieved by user-friendly powerful Ruby objects.

Following example designates node's size by giving a block. The block argument is the vertex index.

```
irb(main):109:0> nt.size{|i|sqrt(nt.tonalist[i].length)}
#<Proc:0x000055f89a21a028@(irb):109>
```

You can run physical simulation adjustment 1000 times as follows; force sets repulsion acceleration, bane sets link spring force, mgn for link orientation magnetism. The last method performs time-integration by leap-frog with timescale 0.1

```
irb(main):110:0> 1000.times{nt.force.bane.mgn.frog(0.1)}
p: 1 npp: 1
reset_cellbuf cellpmax: 44100
reallocate_cellbuf done. cnt: 1 cellpmax: 44100
open GRAPE-5
OpenMP:enabled  Ndevice:4  MAXDEV:4  dev[ 1 1 1 1 ]  Tid2devid[ 0 1 2 3 ]
rsqrt: MSE = 1.158186e-04,  Bias = 8.375360e-08
rescaled. cnt: 0 xmax: 8.000000e+00 mmin: 2.357023e-03
nthread:4
rescaled. cnt: 13 xmax: 1.600000e+01 mmin: 2.357023e-03
```

Successive graph layout is generated by a method “update(update vertice list and link list)”. If you have successively updating network data, you can feed it to the initial network object which already has adequate vertices layout and generate successively changing images. As explained previously , you can select the output path name as the argument to “drawc” method, you can make sequentially numbered still images easily. From such sequential images, you can use tools like ffmpeg to generate movies.