人物相関図の可視化-課題研究３-

文理学部情報科学科
5419045 高林 秀

2021 年 8 月 9 日

概要

本稿は、今年度コンピューティング 2 のネットワークの可視化に関する課題として、小説レ・ミゼラブルの人物相関図をグラフで可視化する実験を行うものである。なお、本課題では processing for java を使用した。本課題は、小説レ・ミセラブルにおいて、同じ章に出演する人物をグラフで表したデータを格納した外部テキストファイルを読み込み、Force-directed アルゴリズムによって可視化することである。結果、各人物をノードとし、人物同士をリンクでつなげることができ、ネットワークの可視化に成功した。

1　目的

本稿は、今年度コンピューティング 2 の課題研究として「ネットワークの可視化」に関する問題に解答するものである。また同時に、問題に関する計算理論についても復習するものとする。

2　計算理論

今回の実験で使用した Force-directed アルゴリズムについて、関連する用語もセットで説明する。

2.1　グラフ描画について

数学的なグラフ理論におけるグラフは、ノードの集合とノード間を結ぶ辺、すなわちリンクより成立している。グラフはノード間のつながりという関係のみを持つが、グラフの構造までは 2,3 次元に描画してみないと視覚化することができない。画面にノードとリンクを直接的に描画した図をノードリンク図と呼ぶ。

ノードリンク図を描画するにあたって、ノードを画面上のどの位置に描画させるか決定しなければならない。すなわち、グラフの描画とはノードの画面上での位置を決定する操作を指す。

一般に、グラフのどの特徴を描画するかによってグラフ描画の手法は異なってくる。グラフ描画によって可視化されるグラフ構造は様々なネットワークや、データ構造の理解に役立てることが可能である。

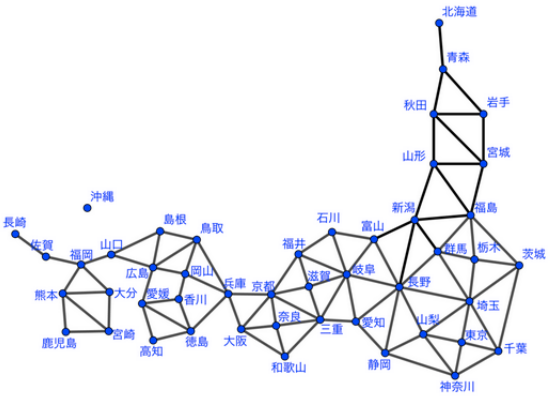

図 1　グラフ描画の例

出典：`https://qiita.com/drken/items/4a7869c5e304883f539b`

グラフをプログラム上で考えるとき各ノードについて、ノードの個数分の座標と速度を配列に保持しておく必要がある。これは、複数の物体の運動と同じである。リンクに関しては、どのノードとどのノードが接続されているか片方のノードの添字 sorce ともう片方のノードの添字 target をそれぞれ整数型の配列で表現することになる。

ノード間のリンクをバネと見なして、その運動をシミュレーションし描画する。このように、リンクをバネと見なしてグラフ描画を行うことをバネモデルと呼ぶ。バネモデルの有名なグラフ描画として、Eadas の方法が挙げられる。Eadas の方法に関して本稿では、詳細な説明は扱わないが、下記のリンクを参照いただきたい。

- Eadas の方法：`http://www2.kobe-u.ac.jp/~ky/labo/research/graph-drawing.htm`

本稿では、後述する Verlet 法により実現できるバネモデルの描画を行う。

2.2　Force-directed アルゴリズム

グラフ描画の手法の 1 つとして、Force-directed アルゴリズムが存在する。このアルゴリズムは、ノード間に作用する力をシミュレーションすることによりノードの座標を決定する。ノード間のリンクをバネと見なして、その運動をシミュレーションし描画する。

バネモデルをシミュレーションするにはまず、ノードの初期座標を適当に与える必要がある。このとき random() 関数を利用し初期位置を任意に決定することができる。次に、全リンクに対しフックの法則[*1]に従う力の計算を行う。しかしそれだけでは座標は収束しない。したがって、空気抵抗を計算しノードの座標が静止状態になるまで繰り返し計算を行う。なお、ノードが描画範囲から大きく外れることも考えられるので、速度更新のときは全ノードの重心が描画領域の中央に来るようノード座標の平行移動を行う。

以下は、完全グラフ K_3 のバネモデルを可視化した p5.js コードである。

[*1] 1687 年にイギリスの R・フックにより発見された物理法則。バネのような弾性をもつ物体に力を加えて変形させると、変形の小さい間は力と変形が比例する、つまり弾性状態では応力とひずみが比例関係にあるという法則。σを応力、E をヤング率、ϵをひずみとすると、$\sigma = E\epsilon$ と示せる・

```
const n = 3;
const x = new Array(n);
const y = new Array(n);
const vx = new Array(n);
const vy = new Array(n);
const r = 25;

const m = 3;
const source = [0, 0, 1];
const target = [1, 2, 2];

function setup() {
  createCanvas(600, 600);
  for (let i = 0; i < n; ++i) {
    x[i] = random(width);
    y[i] = random(height);
  }
  vx.fill(0);
  vy.fill(0);
}

function draw() {
  drawObjects();
  updatePosition();
  updateVelocity();
}

function drawObjects() {
  background(255);
  for (let i = 0; i < m; ++i) {
    const s = source[i];
    const t = target[i];
    line(x[s], y[s], x[t], y[t]);
  }
  for (let i = 0; i < n; ++i) {
    fill(255);
    ellipse(x[i], y[i], 2 * r, 2 * r);
    fill(0);
    textSize(32);
    textAlign(CENTER, CENTER);
```

```
    text(i, x[i], y[i]);
  }
}

function updatePosition() {
  for (let i = 0; i < n; ++i) {
    x[i] += vx[i];
    y[i] += vy[i];
  }
}

function updateVelocity() {
  applySpringForce(0.001, 200);
  applyResistanceForce(0.01);
  applyCentering();
}

function applySpringForce(k, L) {
  for (let i = 0; i < m; ++i) {
    const s = source[i];
    const t = target[i];
    const d = dist(x[s], y[s], x[t], y[t]);
    const theta = atan2(y[s] - y[t], x[s] - x[t]);
    const w = k * (L - d);
    vx[s] += w * cos(theta);
    vy[s] += w * sin(theta);
    vx[t] -= w * cos(theta);
    vy[t] -= w * sin(theta);
  }
}

function applyResistanceForce(k) {
  for (let i = 0; i < n; ++i) {
    vx[i] += -k * vx[i];
    vy[i] += -k * vy[i];
  }
}

function applyCentering() {
  let cx = 0;
```

```
    let cy = 0;
    for (let i = 0; i < n; ++i) {
      cx += x[i];
      cy += y[i];
    }
    cx /= n;
    cy /= n;
    for (let i = 0; i < n; ++i) {
      x[i] += -cx + width / 2;
      y[i] += -cy + height / 2;
    }
  }
```

この実行結果は下図。

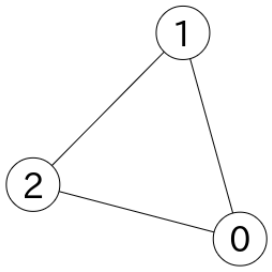

図 2　完全グラフ K_3 のバネモデル

出典：https://qiita.com/drken/items/4a7869c5e304883f539b

下記の p5.js コードは Force-directed アルゴリズムによって可視化したノード数 10 個のグラフを描画したものである。各ノードのペアは $\frac{1}{2}$ の確率でリンクを持つようにランダムにグラフを生成している。

```
let n;
let x;
let y;
let vx;
let vy;

let m;
let source;
```

```
let target;

const r = 25;

function setup() {
  createCanvas(600, 600);
  initializeGraph(10);
  for (let i = 0; i < n; ++i) {
    x[i] = random(width);
    y[i] = random(height);
  }
  vx.fill(0);
  vy.fill(0);
}

function draw() {
  drawObjects();
  updatePosition();
  updateVelocity();
}

function initializeGraph(k) {
  n = k;
  x =  new Array(n);
  y =  new Array(n);
  vx = new Array(n);
  vy = new Array(n);
  m = 0;
  source = new Array();
  target = new Array();
  for (let i = 0; i < n; ++i) {
    for (let j = i + 1; j < n; ++j) {
      if (random(1) < 0.5) {
        source.push(i);
        target.push(j);
        m += 1;
      }
    }
  }
}
```

```
function drawObjects() {
  background(255);
  for (let i = 0; i < m; ++i) {
    const s = source[i];
    const t = target[i];
    line(x[s], y[s], x[t], y[t]);
  }
  for (let i = 0; i < n; ++i) {
    fill(255);
    ellipse(x[i], y[i], 2 * r, 2 * r);
    fill(0);
    textSize(32);
    textAlign(CENTER, CENTER);
    text(i, x[i], y[i]);
  }
}

function updatePosition() {
  for (let i = 0; i < n; ++i) {
    x[i] += vx[i];
    y[i] += vy[i];
  }
}

function updateVelocity() {
  applySpringForce(0.001, 10);
  applyRepulsiveForce(10);
  applyResistanceForce(0.01);
  applyCentering();
}

function applySpringForce(k, L) {
  for (let i = 0; i < m; ++i) {
    const s = source[i];
    const t = target[i];
    const d = dist(x[s], y[s], x[t], y[t]);
    const theta = atan2(y[s] - y[t], x[s] - x[t]);
    const w = k * (L - d);
    vx[s] += w * cos(theta);
```

```
    vy[s] += w * sin(theta);
    vx[t] -= w * cos(theta);
    vy[t] -= w * sin(theta);
  }
}

function applyRepulsiveForce(q) {
  for (let i = 0; i < n; ++i) {
    for (let j = 0; j < n; ++j) {
      if (i === j) {
        continue;
      }
      const d = distance(i, j);
      const w = -q / (d * d);
      vx[i] += (x[j] - x[i]) * w;
      vy[i] += (y[j] - y[i]) * w;
    }
  }
}

function applyResistanceForce(k) {
  for (let i = 0; i < n; ++i) {
    vx[i] += -k * vx[i];
    vy[i] += -k * vy[i];
  }
}

function applyCentering() {
  let cx = 0;
  let cy = 0;
  for (let i = 0; i < n; ++i) {
    cx += x[i];
    cy += y[i];
  }
  cx /= n;
  cy /= n;
  for (let i = 0; i < n; ++i) {
    x[i] += -cx + width / 2;
    y[i] += -cy + height / 2;
  }
```

```
}

function distance(i, j) {
  const minDistance = 10;
  const d = dist(x[i], y[i], x[j], y[j]);
  if (d < minDistance) {
    return minDistance;
  }
  return d;
}
```

2.3 Verlet 法

2.3.1 ニュートンの運動方程式と Verlet 法

■ニュートンの運動方程式 物体にかかる力を F、質量を m、加速度を α としたとき、以下の関係を運動方程式という。

$$F = m\alpha$$

このとき、物体の速度を v としたとき α は v の微分、物体の位置を r としたとき v は r の微分になる。したがって、運動方程式は r の 2 階微分により以下のように示すことができる。

$$\frac{d^2}{dt^2}r = \frac{F(t)}{m} = \alpha$$

上記式は、α と r の関係性を示した微分方程式で、これを解くと物体の運動を計算することができる。また、ある時点 t における加速度 $\alpha(t)$ が示されている場合、Runge-Kutta 法を使用することでより高精度な解を得ることができる。ただし、ふくすうの　物体が複雑に相互作用しているような場合、解を得ることが難しくなる。

■速度 Verlet 法 $r(t)$ の t_0 まわりにおけるテイラー展開は下記のようになる。

$$r(t) \simeq r(t_0) + (t - t_0)v(t_0) + \frac{(t - t_0)^2}{2}\alpha(t_0)$$

t_0 から微小時間 Δ 分だけ変化した時点の位置 $r(t + \Delta)$ は $t = t_0 + \Delta$ のときに得ることができる。実際に代入すると以下のようになる。

$$r(t_0 + \Delta) \simeq r(t_0) + \Delta v(t_0) + \frac{\Delta^2}{2}\alpha(t_0)$$

上記は、$r(t)$ の更新式となる。これを整理すると、速度 $v(t)$ に関する更新式を得ることができる。

$$v(t_0 L \Delta) \simeq v(t_0) + \frac{\Delta}{2}(\alpha(t_0 + \Delta) + \alpha(t_0))$$

したがって、ある時点 t における加速度 $\alpha(t)$ が今の位置 $r(t)$ によって得ることができたら、速度 $v(t + \Delta)$ と位置 $r(t + \Delta)$ を更新することができる。これを速度 Verlet 法という。

速度 Verlet 法は、加速度を計算して速度を更新し、その速度を利用して物体の座標を更新していく手法である。粒子の衝突を考慮する場合やm粒子にかかる力が時点ごとに変化する場合に適しており、実際の分子動力学シミュレーションで利用されている計算法である。

2.3.2 簡易版速度 Verlet 法

速度 Verlet 法は一般に 2 次のテイラー展開が利用される。しかし実際の物理現象を正確に再現するにはより高精度な精度の計算が必要になる。ただし、本稿で行うネットワークの可視化は粒子間の物理的な相互作用を隠喩にするが、現象のシミュレーションではないので精度は必要ない。したがってより簡易的な Verlet 法の更新式を利用することができる。

1 次のテイラー展開を使用して、$\Delta = 1$ とすれば更新式は下記のようになる。

$$r(t+1) = r(t) + v(t)$$
$$v(t+1) = v(t) + \alpha(t)$$

2 次元空間上の現在の座標 $r = (x, y)$ と速度 $v = (v_x, v_y)$ が得られており加速度 $\alpha = (\alpha_x, \alpha_y)$ を計算することで、物体の座標を更新する。コード内では以下のように記載することで、更新できる。

```
x += vx;
y += vy;
vx += ax;
vy += ay;
```

3 実験方法

3.1 計算機環境

本課題を解いた際の計算機環境を下記に示す。

- ホスト OS : Window10 Home Ver.20H2
- 仮想 OS : Ubuntu 20.04.2 LTS
- CPU : Intel(R)Core(TM)i7-9700K @ 3.6GHz
- GPU : Nvidia Geforce RTX2070 OC @ 8GB
- ホスト RAM : 16GB
- 仮想 RAM : 4GB

なお、本演習には p5.js の元である Processing を使用した。

3.2 課題

小説レ・ミセラブルにおいて、同じ章に出演する人物をグラフで表したデータが以下の 5 つのファイルに格納されている。

- name.txt
- source.txt

- target.txt
- group.txt
- weight.txt

name.txt には登場人物の名前が、source.txt と target.txt にはネットワークのリンクにおける始点ノードと終点ノードの添字がそれぞれ格納されている。 group.txt にはノードをグループ化した際のグループ番号が、weight.txt にはリンクの重みが格納されている。

ネットワークのノード数は 77、リンク数は 254 である。 name.txt と group.txt の行数はノード数分、source.txt と target.txt 、 weight.txt の行数はリンク数分となっている。

■Q 1　上記のネットワークを Force-directed アルゴリズムによって可視化せよ。

■Q 2　以下のような工夫点を含めること。

1. ノードの大きさをノードの次数に応じて変える。
2. リンクの太さをリンクの重みに応じて変える。
3. ノードの登場人物の名前を表示させる。
4. その他ノードの配置を綺麗に行うための工夫や高速に行うための工夫

以下は、完成例である。

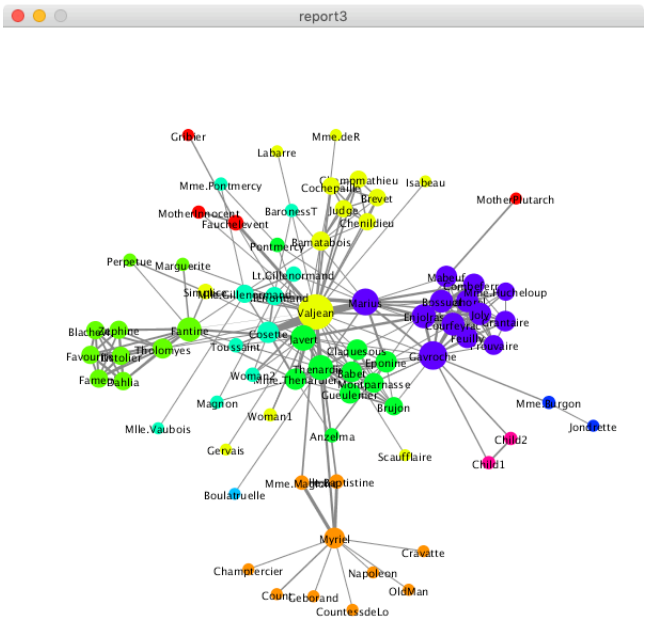

図 3　グラフ描画の例

3.3　実験準備

本課題は Processing を使用して行った。そのため、課題に必要なテキストファイルを読み込むため、各テキストファイルは以下のディレクトリ構成で保管する必要がある。これは、Processing の使用で、外部ファイルはすべて「data」と言う名のフォルダに入れなければ読み込みができないからである。

1. sketch
   （a）sketch.pde
   （b）data
      i. name.txt
      ii. source.txt
      iii. target.txt
      iv. group.txt
      v. weight.txt

4　結果と考察

本課題のソースコードは巻末付録から入手できるので、必要ならばそちらからご参照いただきたい。
以下 Q1,Q2 を実施した最終的な結果のスクリーンショットを示す。

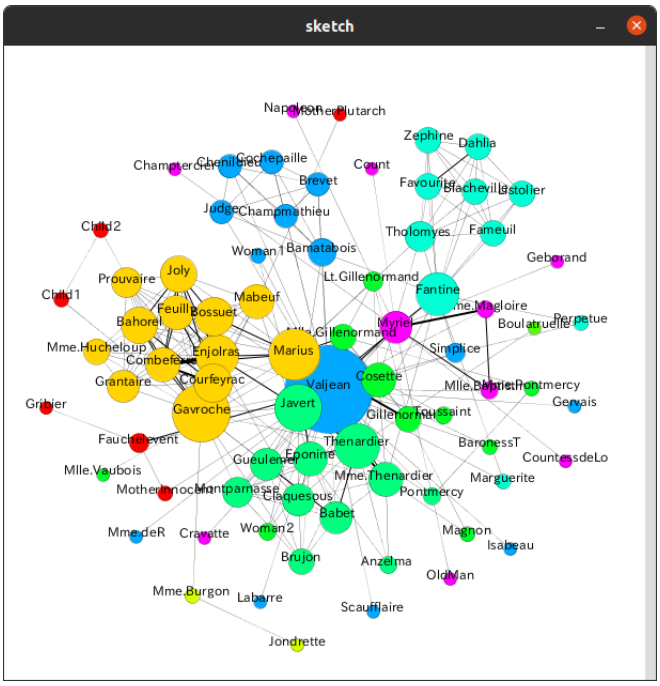

図 4　グラフ描画の結果

- 実行時の動画リンク:https://drive.google.com/file/d/109GkuCWN-_nzi6TysK7cIyPXSdD70Y9x/

view?usp=sharing

なお、実行により各ノードの初期位置は任意に定まるので、座標収束の結果により多少ノードの位置は変化する。

描画プログラムに関して少し説明する。はじめに、外部テキストファイルを読み込むため、配列変数に loadStrings() 関数の返り値を格納している。この配列の要素値を呼び出すことで、各行ごとの情報を読み込むことができる。次に、Q2 の「ノードの大きさをノードの次数に応じて変える」だが、これは単純に drawObjects() 内でノードの描画処理を行っている、ellipse() 関数の第 3，4 引数の値を配列 degree の値によって変化させれば良い。次に、「リンクの太さをリンクの重みに応じて変える」だがこれは、weight.txt に定義されているリンクの重み値にしたがって、strokeWeight() の引数を調節すれば良いだけである。今回は、読み込んだ重み値をそのまま strokeWeight() に渡すと、リンクが見えにくくなってしまうので $\times\frac{1}{5}$ することで、ちょうどよい太さに調節した。次に、「ノードの登場人物の名前を表示させる」は、name.txt の値を行ごとにノードの描画処理を行ったあとに text() 関数を使用し画面に描画することで実現できる。最後の「その他ノードの配置を綺麗に行うための工夫や高速に行うための工夫」は、関数 applyCenteringForce() を使用して、ノードの中心化を行うことで実現できた。各ノードを画面中央に引き寄せる力を加えた場合を計算し、描画している。

今回描画したグラフのリンクの太さは、人物間の関係性の強さを示していると見なすことができるのではないだろうか。またノードの大きさは、その人物がどれくらいの人数の登場人物と関わりをもっているかを示していると考えられる。これは、先に述べたようにノードの大きさはそのノードの次数の大きさを反映しているためである。したがって、Valjean という人物が一番多くの人物と関わりを持っていると考えることができる。これを裏付ける証拠として実際、小説レ・ミセラブルの主人公の名前は Jean Valjean であることが挙げられる。主人公ならば、登場する多くの人物と関わりを持つのは当然であろう。

各ノードの色が何を示しているか気になり、レ・ミセラブルの Wiki サイトを調べた※章末リンク参照。調査前の仮説として「各人物の血縁関係を示している」と言う仮説を立てた。しかし、これは間違いである可能性が高いと考える。例えば、Champmathieu と Valjean は同じノード色で区分されているが、この二人に血縁関係はない。加えて、Marius は黄色のノード色で示されているが、母である Pontmercy とノード色（濃緑）は異なっている。これらの点から各ノードの色は血縁関係を示しているわけでは無いと推測する。最終的にこの部分に関しては何を基準に色を塗り分けているのか見当をつけることができなかった。

5　まとめ

本稿では、今年度コンピューティング 2 の課題研究 3 として、ネットワークの可視化を行った。課題は、小説レ・ミセラブルにおいて、同じ章に出演する人物をグラフで表したデータを格納した外部テキストファイルを読み込み、Force-directed アルゴリズムによって可視化することであった。結果は、画像で示したとおりの結果となり各人物をノードとし、人物同士をリンクでつなげることができた。ただし、各ノードの色がどの様な基準で区分けされているのか見当をつけることができなかった。

6　巻末付録

本稿で使用したソースコード等は以下のリンクから取得できる。必要なら参照いただきたい。

- GoogleDrive:https://drive.google.com/drive/folders/1VwNU_T6XjySMzyiFxUrvNyUqlnMTers4?usp=sharing
- GitHub リポジトリ：https://github.com/tsyu12345/data-science2-report3