平成 9 年度

筑波大学第三学群情報学類

卒業研究論文

題目：オブジェクト指向方法論に基づく オブジェクト図の自動レイアウト環境

主専攻	情報科学
著者名	野口 隆佳
指導教員	電子・情報工学系 田中 二郎

要旨

オブジェクト指向方法論を支援する既存の CASE ツールの多くは、各種図を描画するための図形エディタを備えている。オブジェクト指向方法論で用いる主要な図である、オブジェクト図においても例外ではない。オブジェクト図はソフトウェア開発の基板となる重要な図であり、一般に複数の開発者により利用されるため、見やすく可読性の高いレイアウトが要求される。そのため自動レイアウト機能を持つ図形エディタが開発されつつあるが、従来のツールでは良質なレイアウト機能を持ったものがない。そこで本研究では、オブジェクト指向方法論 OMT[1] で用いるオブジェクト図に対して適していると思われるアルゴリズムを実装し、実験・考察した。

目 次

第 1 章

序論

近年、オブジェクト指向に基づくソフトウェア開発において分析・設計といった上流行程が特に重要視されてきており、それに伴い上流行程を主に支援するオブジェクト指向方法論が各種提案されている [2]。オブジェクト指向方法論には OMT(Object Modeling Technique) 法 [1] や Booch 法 [3] など、すでに広く実用化されているものもあり、それらの方法論を支援する CASE ツールのソフトウェア開発への導入も活発化してきている [4]。一般にオブジェクト指向方法論に基づくソフトウェア開発では、開発対象システムを構成するオブジェクトの静的構造を表現するモデル (オブジェクトモデル) を、方法論固有の図式表記を用いて図により視覚的に記述する。この図はオブジェクト図 (クラス図) と呼ばれ、システム開発の基板となる最も重要な図である。システム開発ではこの種の図は一般に複数の開発者により利用され、開発者間のコミュニケーションの道具ともなるため、見やすく可読性の高いレイアウトが要求される。既存のオブジェクト指向 CASE ツールの大半はオブジェクト図を効率良く描くための図形エディタを備えている。しかし、それらの図形エディタでは、オブジェクト図を構成する基本部品をすべてユーザが配置するといった操作環境となっているため、ユーザは見やすいレイアウトを常に考慮しながら配置していかねばならず、オブジェクト図が複雑化するに従って作図に手間がかかる。

以前よりグラフ描画アルゴリズムを用いたグラフの自動レイアウト技術に関する研究は盛んに行なわれているが [5][6]、最近では RationalRose[7] などの商用ツールにおいても、自動レイアウト技術が採り入れられ実用化が進みつつある。しかしながら、既存のツールにおけるレイアウトシステムは単純なものであるため、オブジェクト図のような複雑なグラフに対しては有効なレイアウトができない。そこで中島により、オブジェクト図のための分割統治法を用いた新たなアルゴリズムが提案された [8]。このアルゴリズムは木構造を反映させたレイアウトを目的としている。よって全体としてのバランスよりも、一つのグラフに含まれる木のそれぞれの対称性などが優先される。

そこで本研究ではアルゴリズムを全体で統一し、エッジに与える力を制御する事で異なる種類のグラフを同時にレイアウトできる、マグネティックスプリング

アルゴリズム [9] に着目した。オブジェクト図は 3 種類のエッジからなるグラフなので、それぞれのグラフに異なる磁場を与える事によりそれらの方向を決めレイアウトを行なった。またこのアルゴリズムを中島のシステムに組込む事により、より多様な要求に答えるレイアウトシステムとしての完成度を高いものにした。

また本論文の構成は、第 2 章で今回レイアウトを行うオブジェクト図を扱う、オブジェクト指向方法論 OMT について解説し、第 3 章で今回オブジェクト図のレイアウトに用いた、グラフ描画アルゴリズムの一般的な分類、概要を述べる。そして第 4 章で中島が実装したレイアウトシステムについて、本研究においてどのようなアルゴリズムがオブジェクト図に適しているか、また実際に実装したマグネティックスプリングモデルについて述べ、考察する。最後に第 5 章で結論を述べる。

第 2 章

オブジェクト指向方法論 OMT

オブジェクト指向によるシステムの開発は、従来の機能中心の手続き型言語でのシステム開発に対し、実世界の自然なモデル化、再利用・拡張の容易さ、仕様変更などによる実装変更の容易さなどの点で優れているとされ、オブジェクト指向に基づくシステム開発が注目されている [2][4]。数あるオブジェクト指向方法論 [3][10][11] の中でも OMT(Object Modeling Technique) 法 [1] は他の手法と比べ、図表やモデルが詳細に決っている事から、十分な記述能力を備えている点と、作業プロセスを具体的に指示しており実用的である点が優れている [4]。 OMT 法ではシステムを 3 種類の異なる視点からモデル化し、その 3 種類のモデルを中心にシステムを開発していく。開発作業は分析、システム設計、オブジェクト設計、実装の 4 段階のフェーズに分類され、それぞれの過程で 3 種類のモデルを進化させていくことで開発を進める。本章では OMT において中心となる 3 種類のモデルについて概説する。

2.1 OMT に用いる 3 モデル

OMT において用いられる 3 種類のモデルは、オブジェクトモデル、動的モデル、機能モデルの 3 種類である。また、これらのモデルを視覚的に記述するために各種の図が用いられる。 (図 2.1)

オブジェクトモデル

システム内のオブジェクト、オブジェクト間の関係、そしてオブジェクトの各クラスを特徴づける属性や操作を示すことによって、システムの静的な構造をとらえるためのモデルである。またオブジェクトモデルはオブジェクト図を用いて表現される。オブジェクト図にはシステムを構築するオブジェクトの静的関係や、個々のオブジェクトの詳細 (属性、操作など) が記述される。

動的モデル

時間と操作の順序にかかわるシステムの側面を記述する。ここで、操作が何を行なうか、それらが作用する対象は何か、どのように実装されているかということには関係せず、“制御” すなわち起こるべき操作の列を記述するというシステムの側面のみをとらえたものである。また動的モデルは状態図と事象トレース図によって表現される。状態図は個々のオブジェクトの状態遷移を記述し、動的モデルは複数の状態図から構成される。事象トレース図はオブジェクト間のメッセージ送受信により発生するイベントの推移を記述する。これもシナリオとよばれる、システムが実行している際に発生する事象系列の数だけ複数存在する。

機能モデル

値の変換にかかわるシステムの側面を記述する。それは関数、写像、制約そして機能依存性といったものからなる。機能モデルはシステムが何を行なうか、いつ行なわれるかとは無関係である。また機能モデルはデータフロー図によって記述される。データフロー図は、値の間の依存性、入力値からの出力値の計算、そして機能を記述している。

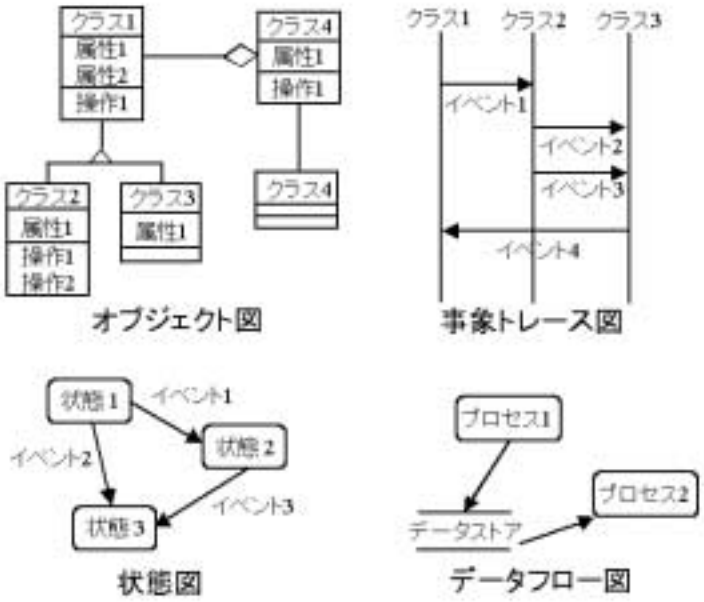

図 2.1: OMT 法で用いられる図

2.2 ３モデルにおける重要性

これらの３モデルは、それぞれがどのような問題においても同等に重要というわけではない。ユーザインターフェースやプロセス制御のような相互作用とタイミングにかかわる問題では、動的モデルが重要であり、コンパイラや工学計算のような重要な計算処理を含む問題では、機能モデルが重要になる。ここでオブジェクトモデルはどのような問題においても重要なモデルとなる。なぜなら、動的モデルや機能モデルにおいて表現される変化や変換は、その対象である何かが存在して初めて意味をもち、それはオブジェクトモデルにより表現されているからである。またそれぞれのモデルは最終的にはオブジェクトモデル上に操作として集約されるのである。

このようにオブジェクトモデルは OMT において最も重要なモデルであり、よってそれを表現するオブジェクト図も OMT における各種の図の中でもきわめて重要な図である。このことから、本研究ではオブジェクト図のレイアウト手法について論じることにする。

第 3 章

グラフ描画アルゴリズム

図による視覚的な表現はインターフェースとして最も基本的なものの一つであり、使われる範囲も広く、簡便で親しみやすいという特長を持っている。よって設計図、家系図、論理回路図など日常的にも多く利用されている。なかでも我々が普段、研究や仕事に扱う事の多い、フローチャート、組織図、システム構成図などは実体をノード、実体間の関係をエッジとしたグラフとして表現される。またこういったグラフはシステム工学、情報工学、ソフトウェア工学など様々な分野に置いて、基礎的なモデルとして広く使われている。最近ではコンピュータの発達にともない、グラフを視覚的に表示したり、操作したりする事が強く求められるようになってきている。そのためこのような図の自動レイアウトを行う、グラフ描画アルゴリズムが数多く提案されている。

グラフ描画アルゴリズムが対象とするグラフは、図 3.1に示すように大きく４種類に分けることができる。各クラスに属するグラフはそれぞれ構造が異なり、それゆえ美的基準もまちまちである。よって各クラスの特長を考慮した描画法がクラスごとに開発されている [9][12][13][14][15] [16]。

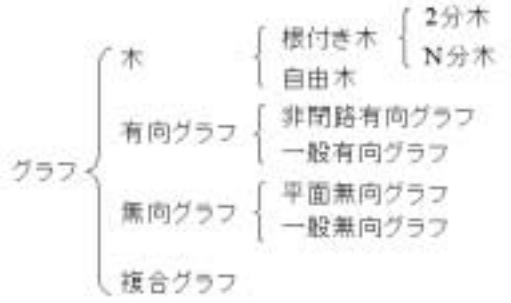

図 3.1: グラフのクラス分類

グラフでいう美的基準とは、描画規約と描画規則のことである [5]。描画規約はノードとエッジに関する基本的約束であり、描画の際に必ず満たされる制約である。描画規則は “良い” 描画の基準となるものである。ただどのようなグラフが

“良い” かは個人により変化し、主観による部分が大きい。しかし、グラフの構成はノードとエッジというように、極度に単純化できるため、細かい違いはあるものの、グラフの性質から来る、共通で、基本的ないくつかの良い描画の基準を識別する事ができる [5]。それらの描画規則の例をいくつか挙げる。

- エッジの折れ点数を最少とする
- エッジの交差数を最少にする
- 対称性 (がある場合) を顕示する
- ノードとエッジの配置や配線の密度を一様化する
- 子ノードを対象に配置する
- 階層構造を垂直あるいは水平に顕示する

グラフ描画アルゴリズムとは一般的に、対象とするグラフの特長から優先度の高い順に、これらの描画規則を取り出し、最適基準または制約条件とする。そして描画規約を最適化問題のゴールとして、複数のステップにおいて順次最適化問題を解くことにより、多くの描画規則を満たしていくものである。しかし、木などの交差の無いグラフを除くと、規則を満たせる効率的な方法が無いことが多く、最適解を得ることは困難なことが多い。したがって、ヒューリスティクスを用いた種々の発見的方法によるアルゴリズムも開発されてきている [13]。

第 4 章

オブジェクト図のレイアウト手法

本章ではオブジェクト図のグラフ構造についてまず述べ、中島により提唱されたオブジェクト図への分割統治法の適用 [8] を紹介し、その後、マグネティックスプリングモデル [9] とそのオブジェクト図のレイアウトへの適用を述べる。

4.1　オブジェクト図のグラフ構造とグラフ描画アルゴリズムの適用

オブジェクト図では、クラスを長方形で表し、クラス間の関係をクラス間を結ぶ線で表す。そこで長方形をノード、線をエッジとすることでオブジェクト図をグラフととらえ、グラフ描画アルゴリズムを適用することでオブジェクト図のレイアウトを行う。

第 3 章で述べたように、一般にグラフ描画アルゴリズムは、その描く対象のグラフの種類によって木、有向グラフ、無向グラフなどに分類され、それぞれ様々なアルゴリズムが開発されている [5][6]。 OMT 法で用いる主な図のうち、状態図、事象トレース図、データフロー図はいずれも比較的単純なグラフ構造で表されるため、既存の有向グラフ、または無向グラフ描画アルゴリズムの適用が容易である。一方オブジェクト図におけるグラフ構造は、 3 種類の関係エッジを有するため、 OMT における他の図とは大きく違う。関連関係は無向エッジ、継承関係は有向エッジ、集約関係は有向エッジまたは木構造と、エッジの種類が多様で、他の図に比べて非常に複雑なグラフ構造となる。図 4.1にオブジェクト図におけるそれぞれの関係の表記法を示す。また、図 4.1に従って描かれる一般的なオブジェクト図の例を図 4.2に示す。

4.2　中島による手法

中島はオブジェクト図において、特に継承関係の木構造 (以下継承木) に着目したレイアウトを行なった。まずオブジェクト図から継承木を取り出し、一つの継承木を一つのサブグラフとおく。オブジェクト図中には継承木が複数存在するので、それぞれの継承木をそれぞれ別のサブクラスとする。そしてそれぞれのサブ

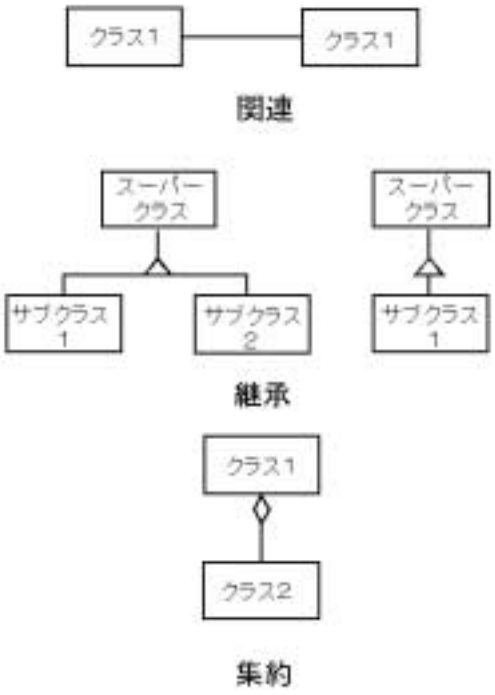

図 4.1: オブジェクト図の表記法

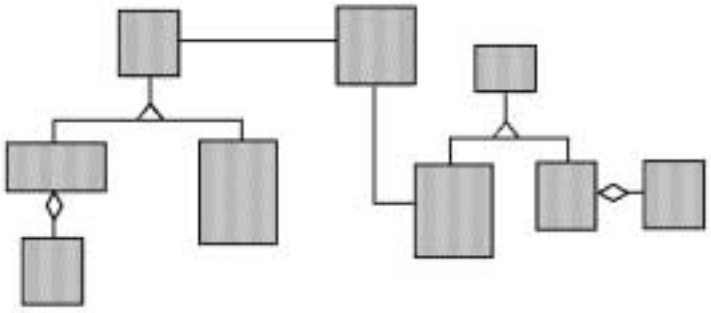

図 4.2: 一般的なオブジェクト図

クラスを、それぞれ一つのノードとして、今度は全体をメタグラフとする (図 4.3)。オブジェクト図をこれら二つのグラフとすることで、階層的に捉え分割統治法を用いた。グラフ描画アルゴリズムは、サブグラフ、メタグラフそれぞれの構造にあったものを別々に適用している。サブグラフは木構造であるため、木描画アルゴリズムを用い、メタグラフは無向描画アルゴリズムを用いている。

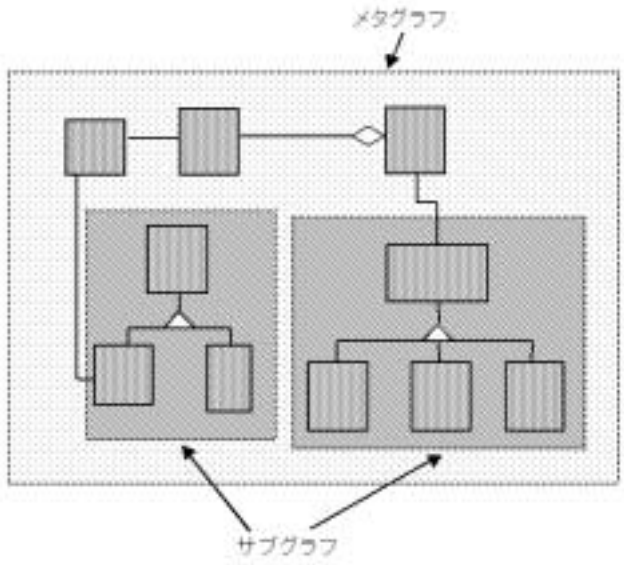

図 4.3: オブジェクト図の階層表現

手順として、最初にそれぞれのサブグラフのレイアウトを、 Walker の木描画アルゴリズム [12] を用いてレイアウトし、その後メタグラフを、 Eades のスプリングモデル [14] を用いレイアウトした。この方法による利点は木構造を示すものに木の描画アルゴリズムを用いるため、継承関係がはっきり示される点である。しかし他の関係に無向描画アルゴリズムを用いるために以下のような点が考慮されていない。

1. 本来有向線で表されるべき集約関係が無向線として扱われ、一定の向きに揃わない点。

2. サブグラフを四角形として表すため、実際にノードが存在しない場所もメタグラフではノードと認識され、全体が疎になりがちな点。

3. メタグラフのレイアウトを行う時、サブグラフの中のノードの位置を考慮しないため、サブグラフの中のどのノードと、外のノードに関係があるのか判断できず、ノード間に関係があるにもかかわらず距離が遠くなる場合がある点

4. 木のアルゴリズムという階層や方向を持つ描画法と、無向グラフのアルゴ

リズムという階層も方向も持たない描画法は性質が全く異なり同時に用いることで、木の階層や方向が十分活かされない点。

5. 木の描画により線が下を向いているのか、無向グラフの描画により偶然的に下を向いているのか、区別が無いという点。

よって我々は以上の点を考慮したアルゴリズムを 4.3 節で論ずることにする。

4.3 オブジェクト図に対するマグネティックスプリングの適用

前節で述べたように、中島によってはじめて OMT におけるオブジェクト図の自動レイアウトアルゴリズムが考案され、実装された。このアルゴリズムは継承関係における木構造を中心としたレイアウト手法であった。そこで本研究ではノード間のつながりによる位置関係に着目したレイアウトを行う。これは以下の理由からオブジェクト図のレイアウトに有効であると思われる。

1. オブジェクト図の表記法として、

   - 関連関係は左から右に読めるようにクラスの配置をする法が良い
   - 継承関係はできる限りスーパークラスを上に書きサブクラスをしたに書くべきである

   などノード間の関係によって推奨されるノードの位置関係が定義されている。

2. エッジの種類が多いグラフ (3 種類) なので、方向のみでノード間の関係がある程度把握できた方が良い

つながりのあるノード間の位置関係を決めるということは、つまりエッジの方向を決めることである。エッジの方向を決めるアルゴリズムとして、有向グラフのアルゴリズムがある。しかし一般に、有向グラフ描画アルゴリズムとして提案されているものは、できるだけ多くの線が一定方向を向くようなアルゴリズムであり、複数の種類の線をそれぞれ違う方向に向けるようなアルゴリズムは無い。そこでマグネティックスプリングモデル [9] が三末・杉山によって提案された。

このアルゴリズムの特徴として、複数種類のエッジを持つ有向グラフのレイアウトにおいてそれぞれの辺を一定方向に向かせることができる、また混在グラフ (有向エッジと無向エッジが混在したグラフ) のレイアウトにおいて有向エッジのみを一定方向にむけ、無向エッジも特定の方向にそろえることができるといった点がある。このアルゴリズムについては 4.3.1 節で詳しく述べる。これは我々がオブジェクト図に行おうとしているレイアウトに非常に適している。そこで本研究ではオブジェクト図のレイアウトアルゴリズムとしてこのマグネティックスプリングモデルを採用した。

4.3.1 マグネティックスプリングモデル

マグネティックスプリングモデルは力指向アルゴリズムの拡張であり、考慮される美的基準は

1. 辺の長さが均一
2. 辺の交差数が最少
3. 対称性を表示
4. 頂点を均一に分配
5. 辺が特定の向きまたは方向に従う

である。ここでいう頂点はそのままグラフのノードに相当するが、辺はノード間を最短で結んだ直線のことである。 1. ～ 4. は従来の無向描画アルゴリズムで広く採用されているものであり、オブジェクト図のレイアウトにも有効である。5. がこのアルゴリズムで新しく追加された基準である。この基準により、オブジェクト図における関係の種類でノードの位置関係が特定できることが期待できるため、このアルゴリズムをオブジェクト図のレイアウトに採用した。また、力指向アルゴリズムとは、無向グラフの描画法の一つであり、頂点を鉄のリングに、辺を力学系を形成するバネに置き換え、リングの安定状態見つけることで適切なレイアウトを求めるものである (図 4.4)。それに対して、マグネティックスプリングモデルでは、辺に働く「回転力」を定義する。これによって、辺の向きや方向の制御を行う (図 4.5)。回転力を定義するのにここでは「磁場」の概念と「磁針」の概念を導入している。辺を「磁針」とし、グラフが、ある「磁場」の中にあるとする。それにより、磁針である辺はグラフが置かれた磁場の北を向くように回転力が働く。

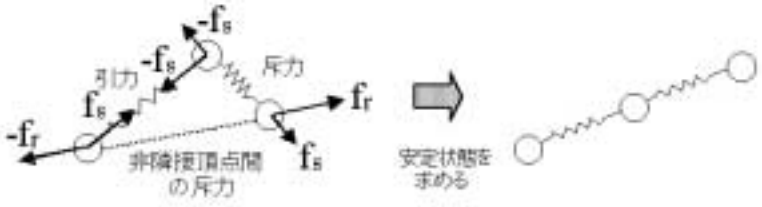

図 4.4: スプリングモデルによるグラフのレイアウト

ただし、ここでいう磁場は仮想的なもので現実のそれとは同じ性質を持たない。以下に、マグネティックスプリングで導入する力と磁場について説明し、マグネティックスプリングモデルのアルゴリズムを示す。

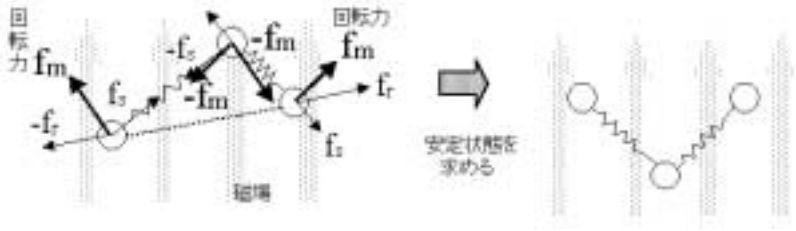

図 4.5: マグネティックスプリングモデルによるグラフのレイアウト

力

- スプリングによる力
  すべての辺はスプリングとみなされ両端の頂点にはそのスプリングによって引力あるいは斥力が働く。$d > d_0$ の時は引力、$d < d_0$ の時は斥力、$d = d_0$ の時は力は働かない。

$$f_s = c_s log(\frac{d}{d_0})$$

  c_s は他の力とのバランスを制御する係数 (以下に現れる c_r、c_m も同様)、d は現在のスプリングの長さ、d_0 はスプリングの自然長である。

- 非隣接頂点間の斥力
  非隣接頂点間には斥力が働く。

$$f_r = c_r \frac{1}{d^2}$$

  d は頂点間の距離である。

- 磁場から受ける回転力
  有向辺の場合、辺 (磁針) は、定義された磁場における北の向きに揃えられる (図 4.6)。無向辺の場合、辺 (磁針) は磁場の向きは関係なく方向のみが揃えられればよいので、北か南の近い方への回転力となる (図 4.7)。このような磁針を無向磁針と呼ぶ。

$$f_m = c_m b d^{\alpha} |t|^{\beta}$$

  b は基準点 (実現上は辺の中点) における磁場の強さ、d は現在の辺の長さ、t は辺の基準点における磁場の北からの終点のずれの角度である。定数 α は辺の長さの、定数 β は t の、それぞれ回転力への影響を制御する。

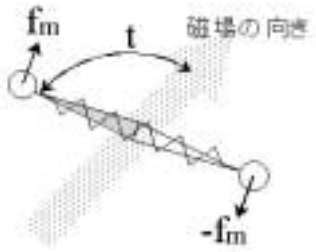

図 4.6: 有向辺が磁場から受ける回転力

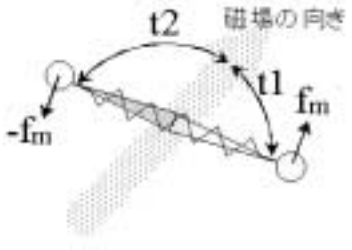

図 4.7: 無向辺が磁場から受ける回転力

磁場

マグネティックスプリングモデルでは磁場として以下のようなものがある。

- 平行磁場
  平面上のどの場所でも向きが一定の磁場。
- 放射状磁場
  ある点を中心に放射状に外側あるいは内側に向く磁場。
- 同心円磁場
  ある点を中心に同心円状に向く磁場。

先程述べたようにここで扱う磁場は現実の磁場とは違い、人工的な性質を自由に与えることができる。その一つとして上記のような磁場をおたがいに影響し合わないように複数組み合わせて与えることができる。これを複合磁場という。

マグネティックスプリングモデルのアルゴリズム

```
各ノードを初期配置する;
(定められた回数のループ)
{
    for v=1 to (ノード数)
    {
        for w=1 to (ノード数)
        {
            if (ノード v と w が辺 e で隣接)
                then
                    辺 e に関して v が受ける f_s を計算する;
                    辺 e に関して v が受ける f_m を計算する;
                    f_v := f_v + f_s + f_m;
                else
                    頂点 w に対して v が受ける f_r を計算する;
                    f_v := f_v + f_r;
        }
    }
    頂点 v を δf_v 移動する;
}
```

4.3.2 オブジェクト図への適用

マグネティックスプリングモデルによりオブジェクト図のレイアウトを行なうにあたり以下の点を考察する。

- それぞれの関係辺へどの磁場を採用するか
- 任意の大きさのノードを考慮したマグネティックスプリングモデル

それぞれの関係への磁場の与え方

辺の種類が３種類あることから、３種類の辺に別々に、影響し合わない磁場を与えることができる複合磁場を用いることにする。

それぞれの辺へどの種類の磁場を与えるかは、オブジェクト図におけるそれぞれのエッジの特性や意味的な要素から以下のようにした。

- 関連

  関連は本質的に双方向性を持つ。これは２クラス間の関係において、どちらの方向をたどるのも同じ意味を持つことから、方向の無い関係であるといえる。２つのクラス間に向きが無いということから両者のクラスに優劣が無い。よって関連辺に関しては無向磁針を採用する。これにより関連辺

の方向のみが決り、向きは磁場の北向きに近い方のノードが北を向くようにできる。また OMT の記法として関連は左から右に読めるような配置が望ましいとされている。そこで関連関係の磁場は平行磁場とし、横向きに磁場を与える (図 4.8)。

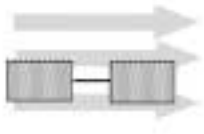

図 4.8: 関連エッジに対する磁場のかけ方

- 継承

  継承は、一つのクラスを特殊化し、その特殊化したクラス (サブクラス) と元のクラス (スーパークラス) の間の関係である。サブクラスは、スーパークラスの属性・操作を継承する。継承関係はスーパークラスからサブクラスへの有向線、またはサブクラスが複数ある場合は木構造によって表す。そして線の中間に、頂点とスーパークラスを結ぶ三角形をいれる。よって継承辺は有向磁針で表す。また OMT 記法において継承は、できる限りスーパークラスを上、サブクラスを下に書くべきであるとされている。よって磁場の向きを下向きにする (図 4.9)。

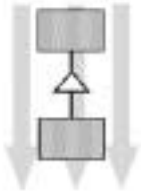

図 4.9: 継承エッジに対する磁場のかけ方

- 集約

  集約は「部分 - 全体」あるいは ”a-part-of” と表される関係である。一つのクラスがいくつかの部分クラスによって構成されている場合などがこれにあたる。集約関係は組み立てクラスから部分クラスへの有向線によって表し、組み立てクラスに小さな菱形をいれる。集約はある特別な意味づけを

強固に結びついた関連の一特殊形態であること、また継承関係のように任意の深さの階層構造を持つことから、関連および継承の磁場の方向と区別するため、関連・継承の中間である斜め右下 45 ° の磁場を与える (図 4.10)。

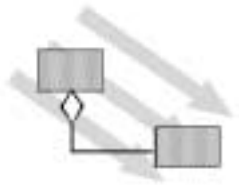

図 4.10: 集約エッジに対する磁場のかけ方

ノードの大きさを考慮したマグネティックスプリングモデル

マグネティックスプリングモデルでは、ノードはすべて無限小の点として扱われるため、そのままオブジェクトのレイアウトに用いてしまうと、ノードの間に重なりが生じてしまう。よって、オブジェクト図のようなグラフに適用するために、任意の大きさのノードが扱えるように改良した。本研究ではスプリングの自然長 d_0 と、非隣接頂点間の斥力の係数 c_r をノードの大きさにあわせて以下のように変化させた (図 4.11)。

$$
\begin{aligned}
&\text{if } (d_1 + d_2 < D0) \\
&\quad \text{then} \\
&\qquad d_0(n_1, n_2) = D0; \\
&\quad \text{else} \\
&\qquad d_0(n_1, n_2) = d_1 + d_2;
\end{aligned}
$$

$$
c_r(n_1, n_2) = (d_1 + d_2) * CR;
$$

$D0$ はスプリングの自然長の最小値、CR は定数である。

こうすることにより大きいノードはバネが長く、また斥力も大きくなり他のノードから遠ざかるようになり、ノード間に重なりが生じないようになる。

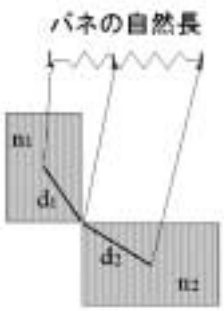

図 4.11: バネの自然長の定義

4.4 マグネティックスプリングモデルの適用結果

前節のようにマグネティックスプリングを改良して実際にオブジェクト図のレイアウトを行った結果を図 4.12〜図 4.14に示す。

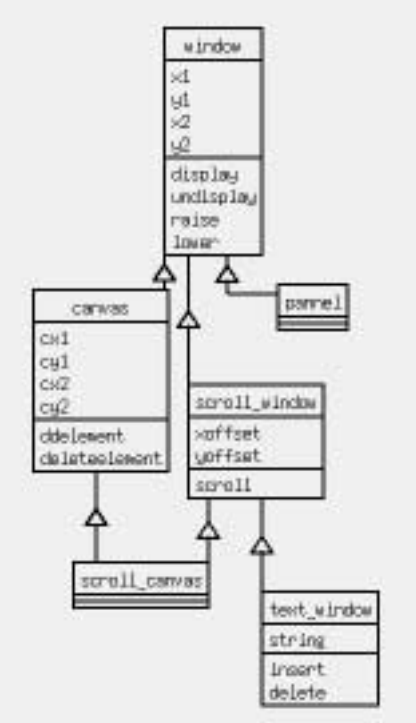

図 4.12: 実行結果 1

図 4.12はウィンドウシステムのオブジェクト図 [1] の一部である。一見継承関係による木構造のようであるが、 scroll_canvas クラスが canvas クラスと scroll_window クラスからの多重継承になっているため、木の描画アルゴリズムでは描画しきれ

ない問題があった。本研究では辺の方向のみを決めることにより、図 4.12のように木であるということを認識させなくてもサブクラスをスーパークラスの上に描画できるようになった。

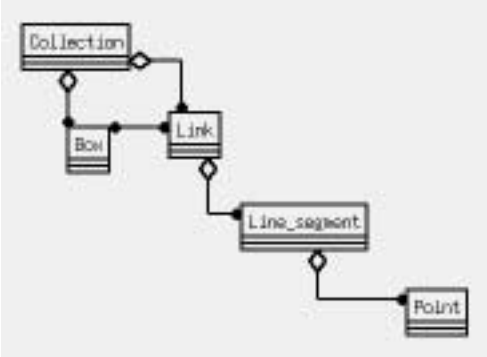

図 4.13: 実行結果 2

図 4.13は対話型図形エディタのオブジェクト図 [1] の一部である。 Collection クラスと Box クラスは集約関係で結ばれているにもかかわらず、ほぼ真下にきてしまっている。これは Collection クラスと Link クラス間の集約関係、 Box クラスと Link クラス間の関連関係により左に押されてしまった結果によるものと思われる。このようにノードの間の関係にグラフ的に競合が生じる場合、必ずしも定めた方向にノードが来ない場合もある。

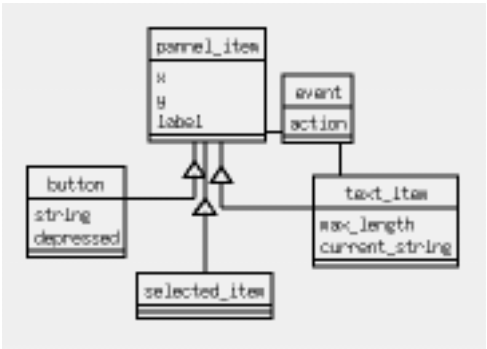

図 4.14: 実行結果 3

図 4.14はウィンドウシステムのオブジェクト図 [1] の一部である。中島によるアルゴリズムではグラフを階層化しレイアウトしたため、サブグラフと外のノードのつながりにおいて疎になる部分があった。本研究では階層化を行わないため、

図 4.14で示すように event クラスが pannel_item クラスがなす木に食い込むようにレイアウトが行われる。

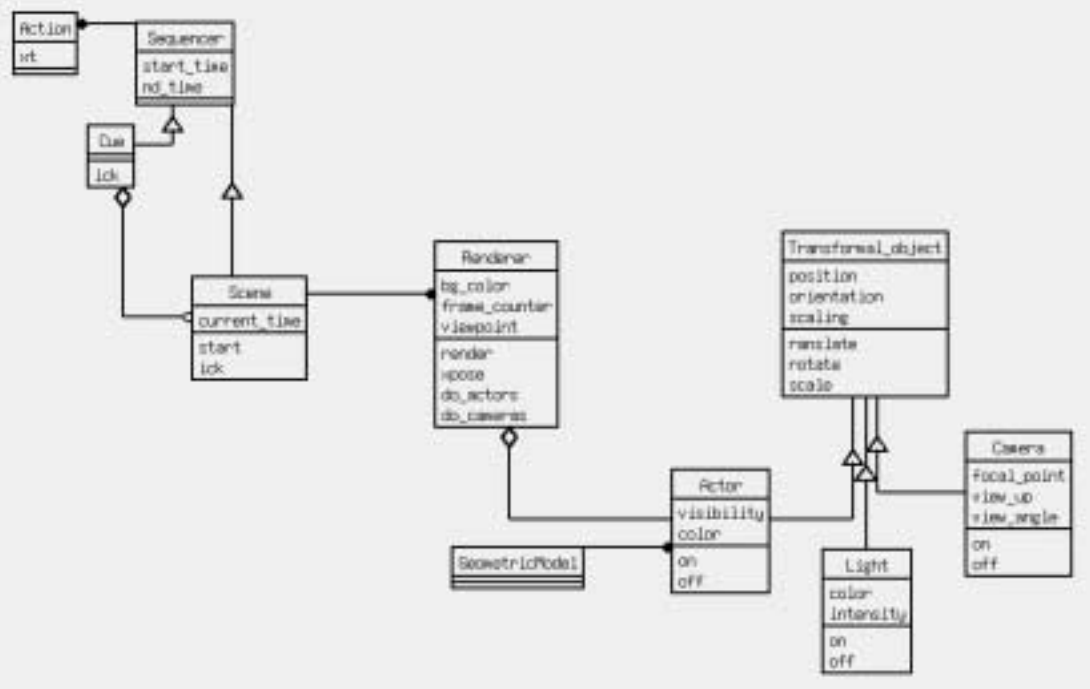

図 4.15: 実行結果 4

図 4.15は、関連、継承、集約を全て含むような実際のオブジェクト図として、アニメーションシステム OSCAR のオブジェクト図 [1] をレイアウトした結果である。それぞれの関係が指示された通りの方向を向いているのがわかる。しかし、Transformal_object クラスがサブクラスを 3 つもつため、サブクラスの一つである Camera クラスが Transformal_object クラスの右下にきてしまい、 45 ° という中間的な角度を与えた集約関係と区別がつきにくくなってしまっている。本研究において、レイアウトはノードの配置のみを行ったのでエッジの配線に関しては木構造を 1 対 1 でつなぐようにした。これを木として描画することにより集約関係のエッジとは区別がつくようになると思われる。なぜなら、継承関係において右下にノードがくるのは、サブクラスが複数ある場合のみでこの場合、木構造と有向エッジの区別は明確になるためである。

第 5 章

考察と結論

三末・杉山によるマグネティックスプリングモデルをオブジェクト図の特長にあわせ任意の大きさのノードに対応できるよう改良した。それをオブジェクト指向方法論 OMT に基づくオブジェクト図の自動レイアウトシステムに実装し、以下の点において中島のアルゴリズムより優れたアルゴリズムであることを確認した。

- 有効エッジである集約関係について磁場を与えることにより、向きをそろえ有効エッジとして扱えるようにした。これは 4.4 節の適用例からも明らかである。

- 階層関係を用いずにエッジに磁場を与えることにより方向を定めた。よって、中島のアルゴリズムでは木構造であるサブグラフと外のノード間のつながりが自然に行われ、疎になりなりがちであること、またノード間の距離が開ことを防げた (図 4.14)。

- 全ての関係エッジに方向性を与えることで、エッジが木の描画以外で下を向くことを極力抑えることができた (図 4.15)。

- 本研究で採用したマグネティックスプリングモデルは、中島のアルゴリズムにおいて、メタグラフのレイアウトに用いた Eades のスプリングモデル [14] の拡張である。よってプログラムのステップ数は中島のメタグラフのレイアウトに要するものとほぼ同等である。しかしながら、中島のアルゴリズムではその他にサブグラフのレイアウトを行っているため、その分ステップ数も多くなり、マグネティックスプリングモデルの方がレイアウトにかかる時間が少なくなっている。ソフトウェア開発においてレイアウト機能は、オブジェクト図を書いていく上でインタラクティブに利用されること

が考えられるため、一回のレイアウトにかかる時間はなるべく少ない方が良い。

オブジェクト図の「見やすさ」とは各個人や着目する点により異なり、決定的なものを定義するのは難しい。よって、それぞれ目的にあったレイアウトアルゴリズムが複数実装されているシステムの方がレイアウトシステムとしては優れていると思われる。継承関係の木構造に着目した中島のレイアウト手法は、継承関係が容易に理解できるという点で有効であるといえる。またオブジェクト図のノードの位置関係に着目した本研究によるレイアウトも、上に示したようにエッジの種類の多いオブジェクト図に対して、関係するノードの方向のみでそのノードとの関係が理解できるという点で有効であったといえる。これらを中島のレイアウトシステムに同時に実装することで、本研究の目的である「より多様な要求に答えるレイアウトシステム」としての質は向上したといえる。

今後の課題としてオブジェクト図におけるマグネティックスプリングモデルでの磁場のかけ方のさらなる検討が挙げられる。

謝辞

本研究を進めるにあたり、終始親切に指導して下さった、担当教官の田中 二郎助教授に深く感謝します。また富士通研究所の三末 和男氏、北陸先端科学技術大学院大学の杉山 公造教授からはグラフ描画アルゴリズムについて多くの貴重な助言を頂きました。富士通研究所マルチメディアシステム研究所ソフトウェア研究室の中島 哲氏にはレイアウトシステムについていろいろな形で助言を頂きました。筑波大学大学院工学研究科の馬場 昭宏氏、筑波大学大学院理工学研究科の南雲 淳氏には論文の書き方について様々な助言を頂きました。筑波大学第二学群日本語日本文化学類の河野 恵美氏には研究の合間に暖かいお茶をいれて頂きました。最後に、田中研究室の皆さんからはさまざまな形でサポートして頂きました。ここに深く感謝の意を表します。

参考文献

[1] J.Rumbaugh et al(羽生田栄一 監訳): オブジェクト指向方法論 OMT ～モデル化と設計
トッパン (1992)

[2] 本位田真一: オブジェクト指向によるシステム開発の実践と課題
日本ソフトウェア科学会 ISOTAS'96 チュートリアル資料 pp71-78 (1996)

[3] G.Booch(山城明宏 訳): *Booch* 法: オブジェクト指向分析と設計
アジソン・ウェスレイ・パブリッシャーズ・ジャパン (1995)

[4] 本位田真一 山城明宏: オブジェクト指向システム開発
日経 BP 社 (1993)

[5] 杉山公造: グラフ自動描画法とその応用 — ビジュアル ヒューマン インタフェース —
計測自動制御学会 (1995)

[6] G.Di Battista, P.Eades, R.Tamassia, and I.G.Tollis: *Algorithms for Drawing Graphs: an Annotated Bibliography*
ftp.cs.brown/pub/papers/compgeo/gdbiblio.ps.Z (1994)

[7] http://www.rational.co.jp/products/rose/rose.html

[8] 中島哲: オブジェクト指向方法論に基づくダイヤグラムの自動レイアウト
平成 8 年度筑波大学大学院修士課程理工学研究科修士論文 (1997)

[9] 三末和男 杉山公造: マグネティック・スプリング・モデルによるグラフ描画法について
情報処理学会研究報告 ヒューマンインタフェース 55-3 pp.17-24 (1994)

[10] S.Shlaer S.J.Mellor(本位田真一 伊藤潔 監訳) : オブジェクト指向システム分析
啓学出版 (1992)

[11] 岡部雅夫 小熊康弘 渡辺香里 羽生田栄一 皆川誠 佐藤英人: オブジェクト指向モデリング手法「MELON」 - ビジネスドメインに特化した手法 -
情報処理学会 OO'96 シンポジウム論文集 pp.1-7 (1996)

[12] J.Q.Walker: *A node-positioning algorithm for general trees*
Software Practice and Experience 20-7.pp685-705 (1990)

[13] 杉山公造: リニエーションに着目した階層グラフの描画法 -SKETCH 法の改良
日本シミュレーション学会 pp.79-84 (1983)

[14] P.Eades: *A Heuristics for Graph Drawing*
Congressus Numeranitium, Vol.42, pp.149-160 (1984)

[15] T.Fruchterman, E.Reingold: *Graphdrawing by force-directed placement*
Software Practice and Experience 12-4.pp45-55 (1994)

[16] 鈴木和彦 鎌田富久 榎本彦衛: 単純無向グラフ自動描画アルゴリズム
コンピュータソフトウェア, Vol.12, No.4, pp45-55 (1995)

付録 A

実装したレイアウトアルゴリズムのソースコード

```
#include "NodePlacer.h"
#include <math.h>
#include <LEDA/node_matrix.h>

#define ITERATION 1000
#define PI M_PI                                          //π
#define DC1 30.0                                         //スプリング定数
#define DC2 85.0                                         //スプリングの長さ
#define DC3 50.0                                         //非隣接頂点間定数 (使ってない)
#define DC4 0.1                                          //微小移動量定数
#define DC5 0.01                                         //関連エッジの磁力定数
#define DC6 0.01                                         //集約エッジの磁力定数
#define DC7 0.01                                         //継承エッジの磁力定数
#define DC8 10.0                                         //磁場の強さ
#define DC9 1                                            //辺の長さの回転力への影響
#define DC10 1                                           //角度の回転力への影響
#define DC11 0                                           //関連エッジの角度
#define DC12 PI/4.0                                      //集約エッジの角度
#define DC13 PI/2.0                                      //継承エッジの角度

extern node get_node_fromid(GRAPH<Node,Edge>& g, string id);

void MagneticNodePlacer::NodeLayout(GRAPH<Node,Edge>& Graph){

  node n1,n2, n3;
  edge e;
  int i;
  double x,y,w,h,d1,d2;
  double nx=0, ny=0;

  node_array<double> forcex(Graph),forcey(Graph);

  //init(Graph);
```

```
  node_array<double> C4(Graph);
  node_matrix<double> C1(Graph);
  node_matrix<double> C2(Graph);
  node_matrix<double> C3(Graph);

  //定数の決定
  forall_nodes(n1,Graph){
    forall_nodes(n2,Graph){
      d1 =
sqrt(pow((Graph[n1].GetWidth()/2),2)+pow((Graph[n1].GetHeight()/2),2));
      d2 =
sqrt(pow((Graph[n2].GetWidth()/2),2)+pow((Graph[n2].GetHeight()/2),2));

      //スプリング定数
      C1(n1,n2) = DC1;

      //スプリングの自然長
      if (d1 + d2 < DC2)
        C2(n1,n2) = DC2;
      else
        C2(n1,n2) = d1+d2;

      //非隣接頂点間定数
      C3(n1,n2) = (d1+d2)*1000;
    }
    //微小移動量定数
    C4[n1] = DC4;
  }

  for(i=0;i<ITERATION;i++){
    //for(i=0;i<1;i++){
      forall_nodes(n1,Graph){
        forcex[n1] = 0;
        forcey[n1] = 0;

        forall_nodes(n2,Graph){
          int flag = 0;

          if (n1 ≠n2){
            forall_inout_edges(e,n1){
              if (Graph.opposite(n1,e) == n2 && Graph.opposite(n2,e) == n1){

                //n1 と n2 が辺 e で隣接していた時スプリングによる力を計算
                forcex[n1] = forcex[n1] +
AttractiveForce(Graph,n1,n2,0,C1[n1][n2],C2[n1][n2]);
                forcey[n1] = forcey[n1] +
AttractiveForce(Graph,n1,n2,1,C1[n1][n2],C2[n1][n2]);

                //n1 と n2 が関連関係の時磁力を計算
```

```
                    if (Graph[e].GetEdgeType() ==1){
                      forcex[n1] = forcex[n1] +
MagneticForce(Graph,n1,n2,0,1,DC5,DC8,DC9,DC10,DC11);
                      forcey[n1] = forcey[n1] +
MagneticForce(Graph,n1,n2,0,2,DC5,DC8,DC9,DC10,DC11);
                    }

                    //n1が集約関係の親の時磁力を計算
                    if (Graph[e].GetEdgeType() ==2 && n1 == Graph.source(e)){
                      forcex[n1] = forcex[n1] +
MagneticForce(Graph,n1,n2,1,1,DC6,DC8,DC9,DC10,DC12);
                      forcey[n1] = forcey[n1] +
MagneticForce(Graph,n1,n2,1,2,DC6,DC8,DC9,DC10,DC12);
                    }
                    //n1が集約関係の子の時磁力を計算
                    if (Graph[e].GetEdgeType() ==2 && n1 == Graph.target(e)){
                      forcex[n1] = forcex[n1] +
MagneticForce(Graph,n1,n2,2,1,DC6,DC8,DC9,DC10,DC12);
                      forcey[n1] = forcey[n1] +
MagneticForce(Graph,n1,n2,2,2,DC6,DC8,DC9,DC10,DC12);
                    }

                    //n1が継承関係の親の時磁力を計算
                    if (Graph[e].GetEdgeType() ==3 && n1 == Graph.source(e)){
                      forcex[n1] = forcex[n1] +
MagneticForce(Graph,n1,n2,1,1,DC7,DC8,DC9,DC10,DC13);
                      forcey[n1] = forcey[n1] +
MagneticForce(Graph,n1,n2,1,2,DC7,DC8,DC9,DC10,DC13);
                    }
                    //n1が継承関係の子の時磁力を計算
                    if (Graph[e].GetEdgeType() ==3 && n1 == Graph.target(e)){
                      forcex[n1] = forcex[n1] +
MagneticForce(Graph,n1,n2,2,1,DC7,DC8,DC9,DC10,DC13);
                      forcey[n1] = forcey[n1] +
MagneticForce(Graph,n1,n2,2,2,DC7,DC8,DC9,DC10,DC13);
                    }


                    flag = 1;
                  }
                }
                //隣接していない時斥力を計算
                if (flag == 0){
                  forcex[n1] = forcex[n1] + RepulsiveForce(Graph,n1,n2,0,C3[n1][n2]);
                  forcey[n1] = forcey[n1] + RepulsiveForce(Graph,n1,n2,1,C3[n1][n2]);
                }
              }
            }
          }
```

```
      //ノード n1 を移動する
      forall_nodes(n1,Graph){
        x = Graph[n1].xcoord();
        y = Graph[n1].ycoord();
        x = x + C4[n1]*forcex[n1];
        y = y + C4[n1]*forcey[n1];
        Graph[n1] = Node(Graph[n1].GetNodeId(), Graph[n1].GetNodeType(),
                         x, y, Graph[n1].GetWidth(), Graph[n1].GetHeight());
      }
  }

  forall_nodes(n1,Graph){
    cout << Graph[n1].GetNodeId() << "," << Graph[n1].xcoord() << "," <<
Graph[n1].ycoord()<< endl;
  }

  cout << "---shift-----shift------shift------shift----" << endl;
  //全体の位置の調整
  ShiftGraph(Graph);

  forall_nodes(n1,Graph){
    cout << Graph[n1].GetNodeId() << "," << Graph[n1].xcoord() << "," <<
Graph[n1].ycoord()<< endl;
  }


  cout << "magneticspring-end" << endl;
}

double MagneticNodePlacer::AttractiveForce(GRAPH<Node,Edge>& Graph,
 node v, node w, int axis, double c, double d0){

  double d,dc,f,dx,dy;
  d = Graph[v].distance(Graph[w]);
  dc = d/d0;

  f = -c*log(dc);

  dx = Graph[v].xcoord() - Graph[w].xcoord();
  dy = Graph[v].ycoord() - Graph[w].ycoord();

  switch(axis){
  case 0: return f*(dx/d);
  case 1: return f*(dy/d);
  }
}

double MagneticNodePlacer::RepulsiveForce(GRAPH<Node,Edge>& Graph,
 node v, node w, int axis, double c){
```

```
  double d,dc,f,fx,fy,dx,dy;

  /*
  katamuki = (Graph[v].ycoord() - Graph[w].ycoord())/(Graph[v].xcoord() -
 Graph[w].xcoord());

  if (Graph[v].ycoord() > Graph[w].ycoord()){
    hantei = Graph[v].xcoord() + (Graph[v].ycoord()-(Graph[v].ycoord()-
Graph[v].GetHeight()/2.0))/katamuki;
    if ((Graph[v].xcoord()-Graph[v].GetWidth()/2.0) < hantei){
      x = Graph[v].xcoord()-Graph[v].GetWidth()/2.0;
      y = katamuki*(Graph[v].xcoord()-Graph[v].GetWidth()/2.0-
 Graph[v].xcoord())+ Graph[v].ycoord();
    }
    else if (hantei < (Graph[v].xcoord()+Graph[v].GetWidth()/2.0)){
      x = Graph[v].xcoord()+Graph[v].GetWidth()/2.0;
      y = katamuki*(Graph[v].xcoord()+Graph[v].GetWidth()/2.0 -
 Graph[v].xcoord())+ Graph[v].ycoord();
    }
    else{
      x = hantei;
      y = Graph[v].ycoord()-Graph[v].GetHeight()/2.0;
    }
  }
  else if (Graph[v].ycoord() < Graph[w].ycoord()){
    hantei = Graph[v].xcoord() + (Graph[v].ycoord()-
(Graph[v].ycoord()+Graph[v].GetHeight()/2.0))/katamuki;
    if ((Graph[v].xcoord()-Graph[v].GetWidth()/2.0) < hantei){
      x = Graph[v].xcoord()-Graph[v].GetWidth()/2.0;
      y = katamuki*(Graph[v].xcoord()-Graph[v].GetWidth()/2.0-
 Graph[v].xcoord())+ Graph[v].ycoord();
    }
    else if (hantei < (Graph[v].xcoord()+Graph[v].GetWidth()/2.0)){
      x = Graph[v].xcoord()+Graph[v].GetWidth()/2.0;
      y = katamuki*(Graph[v].xcoord()+Graph[v].GetWidth()/2.0 -
 Graph[v].xcoord())+ Graph[v].ycoord();
    }
    else{
      x = hantei;
      y = Graph[v].ycoord()+Graph[v].GetHeight()/2.0;
    }
  }
  */

  d = Graph[v].distance(Graph[w]);
  dx = Graph[v].xcoord() - Graph[w].xcoord();
  dy = Graph[v].ycoord() - Graph[w].ycoord();

  f = c/(d*d);
```

```
  switch(axis){
  case 0: return f*(dx/d);
  case 1: return f*(dy/d);
  }
}

double MagneticNodePlacer::MagneticForce(GRAPH<Node,Edge>& Graph,
 node v, node w, int axisST, int axisXY, double c, double b, int alpha, int beta,
double K){

  double d,dx,dy,dk,t,f,k;
  d = Graph[v].distance(Graph[w]);
  dx = (Graph[v].xcoord() - Graph[w].xcoord());
  dy = (Graph[v].ycoord() - Graph[w].ycoord());
  dk = atan2(dy,dx);

  //source のノードについて求める時は磁界の向きを逆転させる
  if (axisST == 1){
    if (K > 0)
      k = K - PI;
    else
      k = PI + K;
  }
  //target の時はそのまま
  else if (axisST == 2)
    k = K;

  //無向磁針の時は角度の小さい方を磁界の向きにする
  else if (axisST ==0){
    if (fabs(K-dk) > 3.0/2.0*PI || PI/2.0 ≥ fabs(K-dk))
      k = K;
    else if (3.0/2.0*PI ≥ fabs(K-dk) || fabs(K-dk) > PI/2.0)
      if (K > 0)
        k = K - PI;
      else
        k = PI + K;
  }

  //磁界と自分のなす角を求める
  if (fabs(k - dk) > PI)
    t = fabs(k - dk - 2.0*PI);
  else
    t = fabs(k - dk);

  //なす角が 0, または距離が 0 の時は磁力も 0
  if (d == 0 ||t == 0)
    return 0;

  f = c*b*pow(d,alpha)*pow(t,beta);
```

```
  if (k < 0){
    if ((dx < 0 && dy < 0) || (dx ≥0 && dy ≥0)){
      if (k < -PI/2.0 && ((PI/2.0 ≥ dk && dk >PI+k) || (k < dk && dk <
-PI/2.0))){
        //cout << "dai1or3tokusource" << endl;
        if (axisXY == 1){
          //cout << "x: " << -f*(fabs(dy)/d) << endl;
          return -f*(fabs(dy)/d);
        }
        else{
          //cout << "y: " << f*(fabs(dx)/d) << endl;
          return f*(fabs(dx)/d);
        }
      }
      else{
        //cout << "dai1or3source" << endl;
        if (axisXY == 1){
          //cout << "x: " << f*(fabs(dy)/d) << endl;
          return f*(fabs(dy)/d);
        }
        else{
          //cout << "y: " << -f*(fabs(dx)/d) << endl;
          return -f*(fabs(dx)/d);
        }
      }
    }
    else if ((dx < 0 && dy ≥ 0) || (dx ≥0 && dy <0)){
      if (k > -PI/2.0 && ((-PI/2.0 < dk && dk < k) || (PI/2.0 < dk && dk <
PI+k))){
        //cout << "dai2or4tokusource" << endl;
        if (axisXY == 1){
          //cout << "x: " << f*(fabs(dy)/d) << endl;
          return f*(fabs(dy)/d);
        }
        else{
          //cout << "y: " << f*(fabs(dx)/d) << endl;
          return f*(fabs(dx)/d);
        }
      }
      else{
        //cout << "dai2or4source" << endl;
        if (axisXY == 1){
          //cout << "x: " << -f*(fabs(dy)/d) << endl;
          return -f*(fabs(dy)/d);
        }
        else{
          //cout << "y: " << -f*(fabs(dx)/d) << endl;
          return -f*(fabs(dx)/d);
        }
      }
```

```
    }
  }

  if (k ≥ 0){
    if ((dx < 0 && dy < 0) || (dx ≥0 && dy ≥0)){
      if (k < PI/2.0 && ((PI/2.0 > dk && dk >k) || (k-PI < dk && dk <
-PI/2.0))){
        //cout << "dai1or3toku" << endl;
        if (axisXY == 1){
          //cout << "x: " << f*(fabs(dy)/d) << endl;
          return f*(fabs(dy)/d);
        }
        else{
          //cout << "y: " << -f*(fabs(dx)/d) << endl;
          return -f*(fabs(dx)/d);
        }
      }
      else{
        //cout << "dai1or3" << endl;
        if (axisXY == 1){
          //cout << "x: " << -f*(fabs(dy)/d) << endl;
          return -f*(fabs(dy)/d);
        }
        else{
          //cout << "y: " << f*(fabs(dx)/d) << endl;
          return f*(fabs(dx)/d);
        }
      }
    }
    else if ((dx < 0 && dy ≥ 0) || (dx ≥0 && dy <0)){
      if (k > PI/2.0 && ((-PI/2.0 < dk && dk < k-PI) || (PI/2.0 < dk && dk <
k))){
        //cout << "dai2or4toku" << endl;
        if (axisXY == 1){
          //cout << "x: " << -f*(fabs(dy)/d) << endl;
          return -f*(fabs(dy)/d);
        }
        else{
          //cout << "y: " << -f*(fabs(dx)/d) << endl;
          return -f*(fabs(dx)/d);
        }
      }
      else{
        //cout << "dai2or4" << endl;
        if (axisXY == 1){
          //cout << "x: " << f*(fabs(dy)/d) << endl;
          return f*(fabs(dy)/d);
        }
        else{
          //cout << "y: " << f*(fabs(dx)/d) << endl;
```

```
            return f*(fabs(dx)/d);
          }
        }
      }
    }

}

//ノードを円周上に初期配置する
int MagneticNodePlacer::init(GRAPH<Node,Edge>& Graph){

    int NodeNum;
    int ox = 300, oy = 300, r;
    double angle, da;
    node nd;
    node_array<double> xp(Graph), yp(Graph);

    NodeNum = Graph.number_of_nodes();                    //全ノード数
    angle = 6.28 / NodeNum;                               //角度差
    r = NodeNum * 5 + 40;                                 //半径
    da = 0.0;                                             //角度

    forall_nodes(nd,Graph){
      xp[nd] = r * cos(da) + ox;
      yp[nd] = r * sin(da) + oy;

      Graph[nd] = Node(Graph[nd].GetNodeId(), Graph[nd].GetNodeType(),
        xp[nd], yp[nd], Graph[nd].GetWidth(), Graph[nd].GetHeight());
      da += angle;
    }
}
```