平成 6 年度

筑波大学第三学群情報学類

卒業研究論文

題目　ビジュアルプログラミングシステムにおける
実行の視覚化

主 専 攻　　情報科学

著 者 名　　中野勝次郎

指導教員　　電子・情報工学系　田中二郎

要 旨

我々は、 Programming System は Non-Textual であるべきと考えた。

本論文では、従来のグラフィック・ディスプレイを有効に生かしているアプリケーションとの比較から、あるべき姿を考察し、その評価のためのプロトタイプシステムの設計を行なった。プロトタイプシステムの実現に際して、視覚要素の配置アルゴリズムについて、力学的モデルに基づく計算モデルを評価し、これに改良を加えその有用性を確かめた。

目　次

第 1 章

序論

我々は、 Programming System について、ユーザインタフェースの向上という立場から考察し、その改良案として Visual Programming System を提案し、その可能性を実験するべく、プロトタイプシステムを開発している。

この論文では、開発中の Visual Programming System に関する、プログラムの実行の視覚化の手法について説明する。特に、ユーザの視覚認知を意識した実行の視覚化の設計と、その効果の実験を行い、考察する。

第 2 章

プログラミングとそのユーザインタフェース

プログラミングという作業は、その特殊性から、コンピュータが一般化した現在でも、ユーザが日常的に行なえるものではない。しかし、より高度になりつつあるコンピュータの能力を活用するために、プログラミングに相当する能力をユーザが使用できるようにすることが必要である。

テキストによるプログラミング言語を使ってプログラミングできるユーザは限られている。しかしながら、プログラミング作業を Textual Programming Language を使わずに行なうことは、一般的ではない。

本章では、プログラミングを Textual Programming Language を用いない方法に関して分析を行なった。

2.1 プログラミングとプログラミング言語に関する遷移

プログラミングが文字列 (Text) によるプログラミング言語によって行なわれてきた背景には、

- コンピュータを使う用途が科学技術分野や統計処理に限られ、使用者はオペレータという特殊な能力を持つものとされていること。
- コンピュータの性能が十分ではなかったため、入力や機械語に変換させることが容易となるように設計されたこと
- コンピュータのコストが高く、性能も低いので、コンピュータ資源はバッチ処理を中心に運用された

といったことがある。逐次的モデル、数学的記法に類似し、英語に類似した構造を持った Textual Programming Lanugage による “compile → execution” プログラミングが、実用的なものとして普及してきたのは妥当であったといえる。

計算機の性能が向上すると、パーソナルコンピュータは、標準的にビットマップ・ディスプレイを装備するようになった。そのような環境に適応し、アプリケーションは WIMP (Window,Icon,Menu,Pointer) を採用して Textual Interface から脱却していった。

プログラミング環境でも、 WIMP による Grahical Interface によって、統合環境といわれるものができた。プログラムの入力以外の操作は Pointing Device を用いて行なうこと

ができ、ソースファイル中の特定の意味を持つものに色をつけたり、ソースファイルを直接操作によって扱うことができるようにはなった。しかし、その対象は Textual Programming Language であり、プログラミングにおける思考作業は今だテキストを中心としている。

2.2 プログラミングとモデルのインターフェース

アプリケーションでは、ひと足先に直接操作を模した WIMP によるユーザインタフェースを構築し、普及にいたっている。対称的に、プログラミング作業の根幹部分は Textual である。

Textual Interface から離れることに成功したアプリケーションを分析し、 Programming System における同様の変革について考察する。

ユーザインタフェースの改善によってアプリケーションが非常に良くなった例として、

- 表計算ソフト
- ファイルシステムの操作
- データベースのフロントエンド
- ワードプロセッサ
- グラフィックエディタ

があげられる。特に表計算ソフトは、ユーザインターフェースを直接的にすることで生じた需要であると考えられる。

これらのアプリケーションはユーザインタフェースに

- 親しみ易いメタファを用いてデータをわかりやすくユーザに提示している。
- 提示されたデータに、直接的な操作を加えることができる。
- 入力、評価といった段階がなく、手順に一貫したモデルを提供している。
- 操作の結果が即座に反映し、その結果はそれまでのモデルが直接変更されて見える。
- 2 次元平面を有効に生かしたデータ表示をしている。

といった特徴を持っている。

これと比べて従来の Textual Programming Language による典型的なプログラミング作業を考える。

- Programming 言語が設定したメタファは、言語・論理・仕様書といったもので馴染みが薄く、同時に文字の範囲を越えない。
- プログラミング言語によるプログラミングは、データではなく、手続きをならべることによって行なわれる。そのため、プログラミング言語をそのまま視覚化したモデルに対する直接操作は難しいと思われる。

- 段階を持っている。コンパイル言語では、入力、コンパイル (構文解析、意味解析)、実行という段階を踏み、さらに、それぞれのモデルが違う。インタプリタ言語であっても、評価時と、定義時のモデルの表示が違う。
- 実行を見ることは、デバッガ・インスペクタといった特殊な環境において可能であるが、それによって見えるデータの形態は、ソースコードとは全く違う。
- 手続きのならびは線形であり、平面を有効に生かすことができない。また、順序関係が必ず意識されなければいけないので、順序関係を表示するために、表示の自由度が低くなる。

このままでは、前述のアプリケーションに共通の性質を持たず、前述のアプリケーションと同様の効果を期待して Non-Textual Interface を適用することができない。

つまり、 Programming に Non-Textual Interface を適用させるためには、

- 従来のプログラミングという枠組にあった全く新しい、ユーザインタフェースの構築
- プログラミングという枠組をアプリケーションのように変える

が考えられる。

我々は、後者を選んだ。すなわち、 Textual Programming で用いられてきた Programming Model を見直し、 Non-Textual Programming に適合した Programming Model を考えることとした。

2.3 Programming Model for Visual Programming System

2.2章で分析した Application の性質を備えた Programming Model は次のようになる。

- 一貫したメタファあるいはシンプルで親しみの持てるルールを持つ。また、図形をもって表示する。
- データ、プログラム (あるいはそれに相当するもの) は、図形の直接操作によって変更される。
- 一つの画面の中で、データとプログラムの編集を行なうことができ、データもプログラムも操作上、表示上のモデルは同じものであるとする。
- データ、プログラムの変更は即座にその場に反映し、できるのならば評価が行なわれるようにすることができる。

我々は、このようなシステムをあらためて Visual Programming System と定義し、これを満足する Visual Programming System の prototype system の設計をする。

第 3 章

関連研究

我々のシステムの詳細を決定する前に、現在までに、あるいは現在行なわれている Visual Programming に関連のある研究で特徴的なものについて分析し、評価する。

3.1 ビジュアルプログラミング

Visual Programming Language については、 [1] に詳しい。比較的新しい概念なので研究者によって様々な解釈が存在することが紹介されている。ここでは、簡単にその内容を紹介する。

- ビジュアルプログラミングは、 2 次元以上でプログラムを表現するすべてのシステムを指す。
- 視覚的情報を視覚的対話によって操作できるものが Visual Language である。あるいはプログラムの提示が視覚的表現によるものである。
- 視覚的表現に変換した Programming Language を指す。表現には明らかなテキスト表現による program が含まれない。
- programming 作業で、視覚的表現を使うことを Visual Programming という。 graphical interface において使われたり、新たなパラダイムの programming lanugage の表現として使われたり、 program の振舞いやデータを視覚化したりすることである。

3.2 構造 editor

構造エディタ [2] では、 Textual Programming Language を図表現として、その編集によってプログラムを作るものもある。

手続き型言語の Textual Programming を図表現変換するものには、 Flowchart や PAD、NS 図式といったものがある。これらは、 2.2章で述べた条件を考えると、

- プログラムと実行の次元が違う。
- 効果的に図示できているのはプログラムの制御の流れだけであり、 procedure や variable を名前で参照するために、複雑になるとわかりにくい。

という問題点がある。

3.3 BALSA,Tango

Algorithm Visualization は、 program の挙動を可視化し、 debug や、教育に役立てようというものである。

BALSA や Tango は、 Algorithm を可視化するための System で、 animation を用いて可視化することができる。

このシステムを用いて algorithm animation を作るにはユーザが algorithm に相当する部分を抽出し、可視化 library を使って、プログラムを書かなければいけない。

これらを分析すると次の点が明らかになる。

- animation は、状態の遷移を非常に効果的に表すことができる。
- 教育目的であればともかく、 programming の可視化という観点からは、 animation は実行と同時に自動的に作られなければいけない。

3.4 Pictorial Janus

Pictorial Janus[12] は、主に、実行の可視化をする。

Programming Model として、並列論理型言語の Janus を用い、 Animation によって実行の過程を表示する。 animation は、実行の Model をユーザに認知させるのに非常に効果的であることがわかる。

入力は、 graphic editor を使っているため、入力即実行というわけにはいかない。

3.5 Prograph

Prograph[13][14] は、 Programming を可視化している。 Model は、 data flow model となっていて、アイコンで示された手続きを、線分で接続して data の流れを指定する。実行は可視化されるが、その能力は限られ、効果的とはいえない。

3.6 PP

PP[5][6] は、 FGHC の定義節の編集、 Query の評価を可視化し、直接操作による編集を可能としたシステムである。

Program の編集と実行が、全く同じ表示で行なわれる。

Pictorial Janus と違い、実行はスナップショットである。

3.7 宣言型言語と Visual Programmming System

PP、 Pictorial Janus では並列論理型言語がベースモデルとして使われている。 Prograph においても、 dataflow model であり、プログラムは基本的に宣言的に記述される。宣言的言語は以下のの理由から Visual Programming System のベースモデルに向いている。

- **プリミティブが簡単**
  高度に抽象化された少ないプリミティブによってすべてを表現するようにしている。

- **プログラムがコンパクト**
  宣言型言語、特に FGHC のような並列論理型言語では、 Program(述語定義) は基本的に入れ子になることがなく、単位が細かくコンパクトにまとまっている。同じモデルを表現するのに、 Visualized されたものは Text に比べてかさばるのであるが、 GHC の定義節ではそういったことが問題になりにくい。

- **プログラムは時間順序に依存しない**
  宣言型言語は評価が実行順序に依存しないため、暗黙の時間的関係を示す必要がない。。二次元空間上で時間関係を暗黙のうちに表すことは結果として、配置の自由度を一つ減らすことになる。また、ユーザは空間関係から厳密に認識することが困難である。宣言型言語は、その非逐次性からも、 2 次元以上に広がる空間に展開する program を書く上で有利であるといえる。

- **環境の明示性**
  宣言型言語の program は、その program の中で使われるデータは、すべて明示的に引数で受け渡される。手続き型言語などでは、暗黙のうちに様々な変数を参照できる状態にあるのだが、これをすべて視覚化するのはかなり難しい。宣言型言語では、明示的に受け渡すために、結果として、関係のないものは省かれ、環境の共有関係によって、 object が空間に適切に配置されることになる。(逆に、 programming が大変になることもある)。

- **単一代入変数 オブジェクトの永続性**
  宣言型言語では変数は単一代入である。つまり、一度作られた参照が変更されることはなく、データ自身が変更されることはない。この性質はモデルの視覚化においてよい特性であるといえる。参照関係を保存してデータ (項) をコピーする限り、モデルの意味は全く変わらないためで、表示の都合によって、見易くデータをコピーし、配置することができるのである。

3.8 他研究に関する考察

他研究を比べ見ると、 Visual Programming System において重要な keyword が見えて来る。

- アニメーション
- プログラムと実行の表現の差がすくない
- 統合環境
- Base Model は宣言型言語 (特に並列論理型言語)

我々の System は、これを含むべきであると考えた。

第 4 章

PP の設計

2、 3章を考慮しつつ、我々の Visual Programming の定義に従ったシステムを一部設計した。設計した部分は、プログラムの実行の部分である。

3.6章での PP がもっとも Visual Programming System に近いものであるため、これをベースに、我々は、新たな Visual Programming System の提案をする。新たな Visual Programming System を改めて PP と呼び、 [5] の PP を旧 PP と呼んで区別をする。

4.1 PP でのプログラム実行の概観

PP でのプログラム実行を簡単に説明すると、

- PP Field と呼ばれる面がユーザに提示される。
- ユーザはこの上に FGHC でいう Query を後述のルールにしたがって作る。
- 書かれていくモデルの goal は、 guard が満たされると即座に評価し、結果が Field 上に展開する。

となる。

入力即評価されてゆく様の例を図 4.1に示す。この例では、 query に次々に object を加えていく。ロボットの手のようなものがユーザが使っているカーソルである。

ちなみに、それぞれの goal の引数には入出力の区別は特に表示されない。 plus/3、 mult/3 のそれぞれの引数のうち、 *Result* が出力であり、その他は入力である (交換可能な演算子なので、入力に区別はいらない)

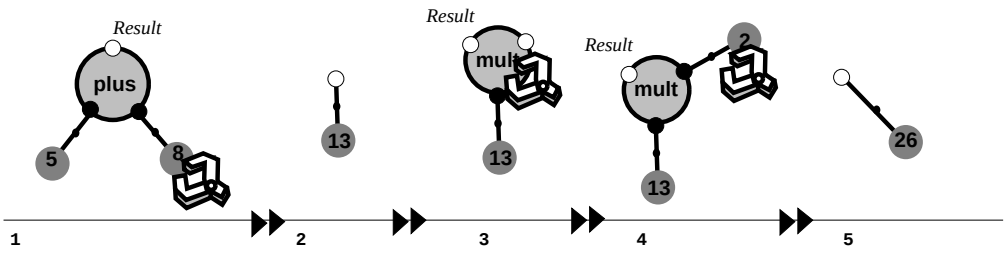

図 4.1: PP field における操作・評価例

図 4.1では、

1. ユーザは goal(class plus/3) に atom の 5 と 8 を接続した
2. plus のガードが満たされた (2 つの引数に具体的な値が与えられた) ため、 reduce し、atom の 13 が残った。
3. ユーザは 2 での答えを mult に繋げたが、 mult の guard が満たされていない (2 つの引数の値に具体的な値が与えられていない) ため、そのまま、2 つめの引数が具体化されるのを待っている。
4. ユーザが mult のもう一方の引数に atom 2 を与えた。
5. 4 の時点で gurad が満たされたため、 mult は reduce し、 26 の atom が残った。

以下に、この PP field における Model の表現についての詳細について説明する。

4.2 PP field における Model の静止表現

PP では、旧 PP と同様に FGHC をベースモデルとしている。

ベースモデルは、ゴール、項、論理変数と、これらの束縛関係によって定義される。これから、ゴール、項、論理変数を頂点とし、これらの束縛関係によって辺で接続した単純無向グラフを構造とし、対応する頂点にアイコンをおくことで、ベースモデルの表現をすることができる。(但し、変数は未単一化論理変数のみを表示する。)

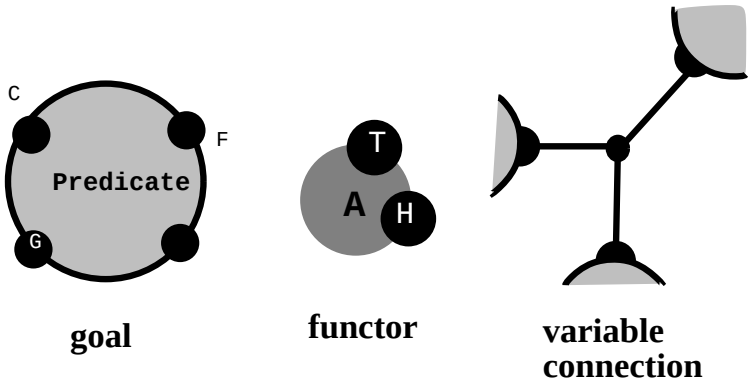

図 4.2: 頂点に対応する部品

ゴール、項、未単一論理変数の頂点には、図 4.2の図形を対応させる。

- ゴールは、頂点を中心に縁どられた円を描き、円周上に引数に対応する塗りつぶされた円を描く。
- 項は、ゴールと同様であるが、縁どられておらず、必ず他のゴール、項の引数、から参照されている。
- 未単一化論理変数は、ゴールの引数などと同様の塗りつぶされた円である。

それぞれすべてに名前をつけることができる。

大きさを比べた時、図 4.2のように、 goal>functor≫variable という関係になる。

4.3 Model の変形表現

モデルはルールにしたがって変形をしてゆく。変形の過程はアニメーションの形でユーザに提示することとする。

4.3.1 アニメーションと認知

人間は形や色だけで判断しているわけではなく、テクスチャ、動きといったものを重要な認知要素として使っている。

特に、動きに対しては敏感に反応する。相対的に動きの大きいものに反応は大きく、背景 (全体) と同じように動いているものに対しては反応が鈍い。この認知特性を考えると、アニメーションによって表現を強調することができる。

具体的には、変更部分を大きく動かし、変更の少ない部分は小さな動きとする。 (これについては 5.1章で改めて述べる)

4.3.2 モデルの変形とアニメーション

モデルの変形について検討することで、具体的なアニメーションの方法を考える。

- **committing goal**
  goal はガードが満たされると直ちに commit して、 commit された述語定義の body 部を作りだす。 body 部は、標準的な大きさの 1/5 の大きさで goal のあった位置に配置される。小さなゴールは、再配置されながら、大きくなる。その goal は消滅する。

- **unification**
  変数の腕のうち、 2 本が具体化された場合、それは unification を表す。具体化された両方が同じ functor である場合は、融合するように引数を接続する。

- **reducing variable**
  変数は、変数の腕の先に変数がついている場合は、これは一つにまとまる。

具体的にアニメーションの様子を図 4.3.2に示す。例では、 merge/3 の一方の入力に、新たにストリームが足された時に行なわれるアニメーションである。

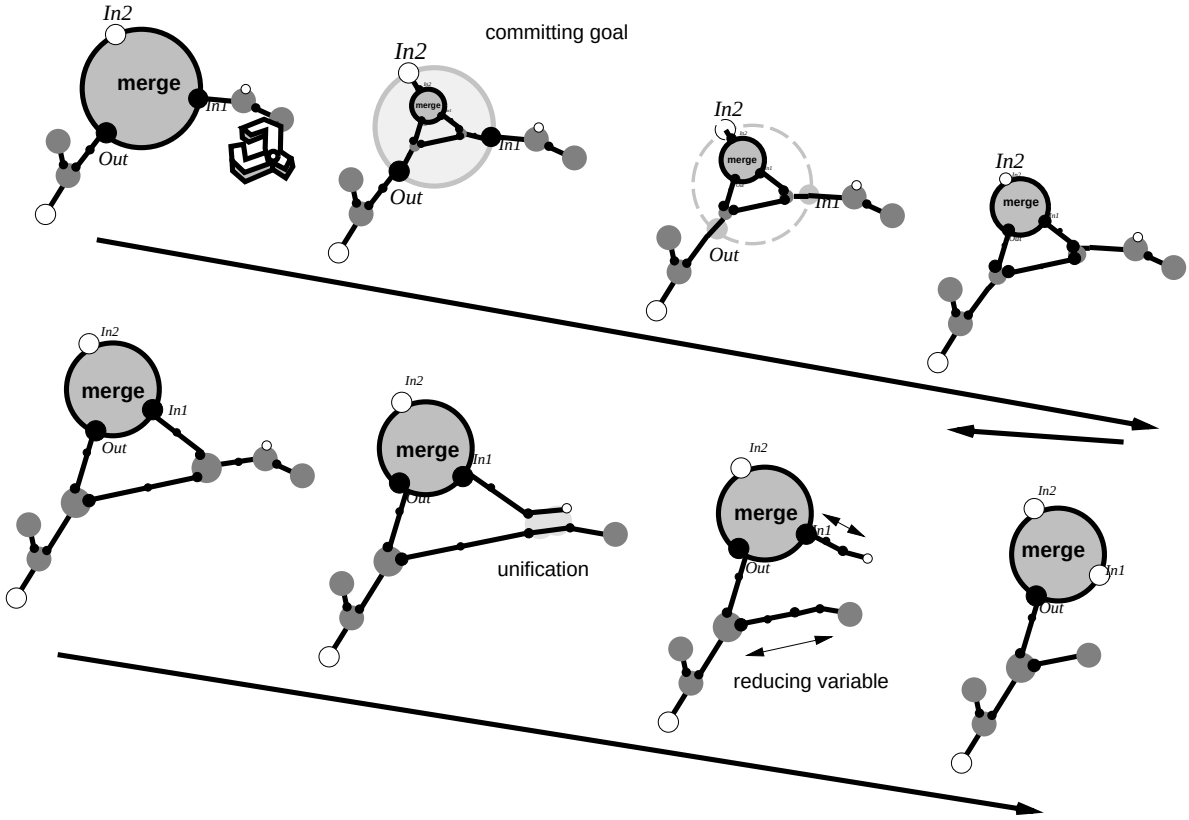

図 4.3: アニメーションの様子 (snapshots)

4.4 PP の設計の評価

PP の設計は、 2.3章での考察を考慮している。直接操作によるデータが、その場で反映し、モデルが変更される。プログラミング (述語の編集) も、同じ PP field において行なわれる枠組を用意し、我々の考える Visual Programming System の条件は満たすことができる。

第 5 章

PP の実装の要素技術

5.1 Layout

評価の結果、 Model が変更されると、 Model 表示の graph の layout も変更する必要がある。

layout は、鈴木のアルゴリズムを使い、さらに改良し、実験した。

5.1.1 単純無向グラフのレイアウト

実行モデルは単純無向グラフに写像されることは既に説明した。

単純無向グラフとしての頂点配置は有効グラフに比べ、あまり研究されていない分野である。しかし、 [3][4] を参考にした結果、力学モデルに基づくアルゴリズム [15] が様々な意味で、 PP における視覚化に向いていることがわかった。同アルゴリズムを改良した [16] を参考に、アルゴリズムを実装し、このアルゴリズムによるレイアウトの効果について調べた。

鈴木のアルゴリズム

まず、鈴木のアルゴリズム [16] を簡単に紹介する。

鈴木のアルゴリズムは [15] で紹介されているアルゴリズムをさらに改良したものである。どちらにしろ、頂点間の理想距離を設定し、その理想距離と実際の距離の差を縮めるように頂点を動かすという力学的モデルである。簡単な例が図 5.1.1である。

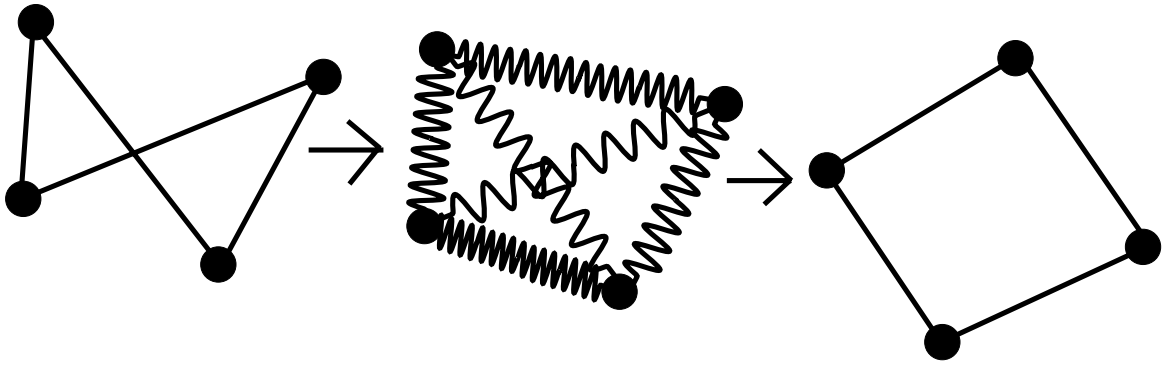

図 5.1: 力学的モデルの説明

具体的なアルゴリズムは、次のようになる。

まず、理想距離をすべての頂点間に関して計算する。理想距離は任意の 2 頂点間 (P_i, P_j) の最短パスに沿った辺の長さの合計 (l_{ij}) に比例するとして、理想距離と現在の配置におけるの距離から頂点間の引力 (F_a)・斥力 (F_r) が定義され、これから、 2 頂点間にある引力が定義される (F)。

$$
\begin{aligned}
F_a(p_i, p_j) &= c_0 \times x_{ij}/l_{ij} && (5.1)\\
F_r(p_i, p_j) &= c_0 \times l_{ij}/x_{ij} && (5.2)\\
F &= F_a - F_r = c_0(\frac{x}{l} - \frac{l}{x}) && (5.3)
\end{aligned}
$$

頂点間のエネルギーの総和 (E_0) がグラフの不均衡さであるとする。

$$
\begin{aligned}
E_0 &= \sum_{1\le i<j\le n} \int F dx = c_0 \sum_{1\le i<j\le n} \int_l^x (\frac{x}{l} - \frac{l}{x}) dx \\
&= c_0 \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \left\{ \frac{1}{2l_{ij}} (x_{ij}^2 - l_{ij}^2) - l_{ij}(\log x_{ij} - \log l_{ij}) \right\} \\
&= c_0 \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \left[\frac{1}{2l_{ij}} \left\{ (x_i - x_j)^2 + (y_i - y_j)^2 - l_{ij}^2 \right\} \right. \\
&\qquad \left. - l_{ij} \left\{ \log \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} - \log l_{ij} \right\} \right] \qquad (5.4)
\end{aligned}
$$

この E_0 を最小にするレイアウトが最適なレイアウトであると考えられる。しかしながら、多変数偏微分方程式であらわされる E_0 を最小にする頂点の配置を実時間で計算することは難しく、 Intaractive System 上では現実的ではない。よって、 local minimum を求め、それによって解の代わりとする。

まず、エネルギー (Δ_m) の一番高い頂点をを選び、それを v_m とする。

$$
\Delta_m = \sqrt{\left\{ \frac{\partial E_t}{\partial x_m} \right\}^2 + \left\{ \frac{\partial E_t}{\partial y_m} \right\}^2} \qquad (5.5)
$$

v_m の移動による E の local minimum の条件は、次のものである。

$$\frac{\partial E_t}{\partial x_m} = \frac{\partial E_t}{\partial y_m} = 0, \qquad \text{for } 1 \leq m \leq n \tag{5.6}$$

$$\frac{\partial E_t}{\partial x_m} = c_0 \sum_{j \neq m} \left\{ \frac{x_m - x_j}{l_{mj}} - \frac{l_{mj}(x_m - x_j)}{(x_m - x_j)^2 + (y_m - y_j)^2} \right\} \tag{5.7}$$

$$\frac{\partial E_t}{\partial y_m} = c_0 \sum_{j \neq m} \left\{ \frac{y_m - y_j}{l_{mj}} - \frac{l_{mj}(y_m - y_j)}{(x_m - x_j)^2 + (y_m - y_j)^2} \right\} \tag{5.8}$$

この条件を満たすために、 v_m を繰り返し少しずつ移動させる。

$$x_m^{(t+1)} = x_m^{(t)} + \delta_x, \quad y_m^{(t+1)} = y_m^{(t)} + \delta_y, \ t = 0, 1, 2, \cdots \tag{5.9}$$

移動量 δ_x、 δ_y は、次の式を満たすものである。

$$J(x_m, y_m) \begin{bmatrix} \delta_x \\ \delta_y \end{bmatrix} = - \begin{bmatrix} \frac{\partial E_t(x_m^{(t)}, y_m^{(t)})}{\partial x_m} & \frac{\partial E_t(x_m^{(t)}, y_m^{(t)})}{\partial y_m} \\ \frac{\partial E_t(x_m^{(t)}, y_m^{(t)})}{\partial x_m} & \frac{\partial E_t(x_m^{(t)}, y_m^{(t)})}{\partial y_m} \end{bmatrix} \tag{5.10}$$

$J(x_m, y_m)$ は、 Jacobian matrix であり、次のものである。

$$J(x_m, y_m) = \begin{bmatrix} \frac{\partial^2 E_t}{\partial^2 x_m} & \frac{\partial^2 E_t}{\partial x_m \partial y_m} \\ \frac{\partial^2 E_t}{\partial y_m \partial x_m} & \frac{\partial^2 E_t}{\partial^2 y_m} \end{bmatrix} \tag{5.11}$$

その要素は次のようにして求められる。

$$\frac{\partial^2 E_t}{\partial^2 x_m} = c_0 \sum_{j \neq m} \left[\frac{1}{l_m j} - l_m j \frac{(y_m - y_j)^2 - (x_m - x_j)^2}{\{(x_m - x_j)^2 + (y_m - y_j)^2\}^2} \right] \tag{5.12}$$

$$\begin{aligned} \frac{\partial^2 E_t}{\partial x_m \partial y_m} &= \frac{\partial^2 E_t}{\partial x_m \partial y_m} \\ &= c_0 \sum_{j \neq m} l_{mj} \frac{2(x_m - x_j)(y_m - y_j)}{\{(x_m - x_j)^2 + (y_m - y_j)^2\}^2} \end{aligned} \tag{5.13}$$

$$\frac{\partial^2 E_t}{\partial^2 y_m} = c_0 \sum_{j \neq m} \left[\frac{1}{l_m j} - l_m j \frac{(x_m - x_j)^2 - (y_m - y_j)^2}{\{(x_m - x_j)^2 + (y_m - y_j)^2\}^2} \right] \tag{5.14}$$

これを計算機で実装する際には、次のアルゴリズムによる。

compute l_{ij}
while ($\max_i \Delta_i > \varepsilon$) {
    let v_m be the vertex satisfying $\Delta_m = \max_i \Delta_i$;
    while ($\Delta_m > \varepsilon$) {
        compute δ_x and δ_y;
        $x_m = x_m + \delta_x$;
        $y_m = y_m + \delta_y$;
    }
}

実際にこれをC言語で実装し、仮想的な実験データを入力として、レイアウトをしてみた。レイアウトはX Window Systemの一つのWindowとして表示される。表示の中の[n]は、説明のために各頂点についている番号である。

仮想データとして、ゴール3つ、atom/termが12のデータで適当に変数によって接続されているものを考えた。これを入力としたところ、図5.1.1が出力された。(なお、未単一化論理変数に対応する頂点を表示しない様にしてある)

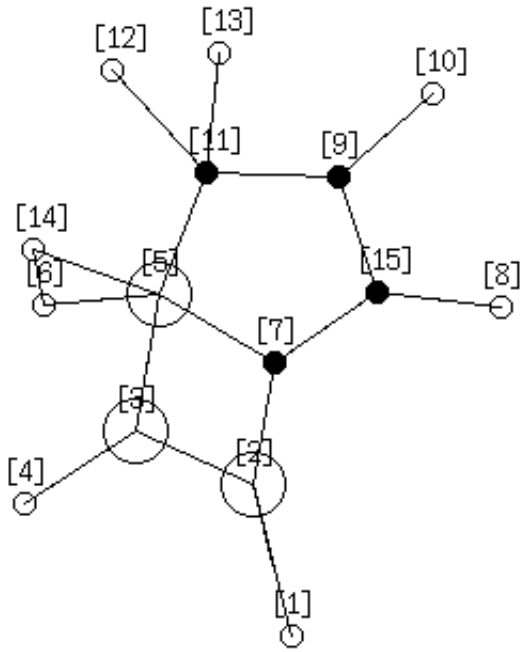

図 5.2: 鈴木のアルゴリズムによるレイアウト

この結果から次のことがいえる。

- 均整のとれたグラフを作ることができた。
- ゴールや項の接続関係をうまく反映した配置ができた
- 項とゴールが等しく扱われているため、単調であり、データが不自然に場所をとっているので、ゴールに主体性がない。結果としてプログラマの持つモデルとかけはなれた、見にくいものとなっている。

このアルゴリズムを改良し、PPに使うこととした。

- ゴールが点在する間に、項が散らばるような配置
- ゴール間の距離を十分に離す

をいう条件を満たすことで、ゴールに主体性を持たせ、見やすいものとする。
まず、ゴールとゴールを結ぶ辺の長さを単純に5倍にする実験を行なった。(図5.1.1)

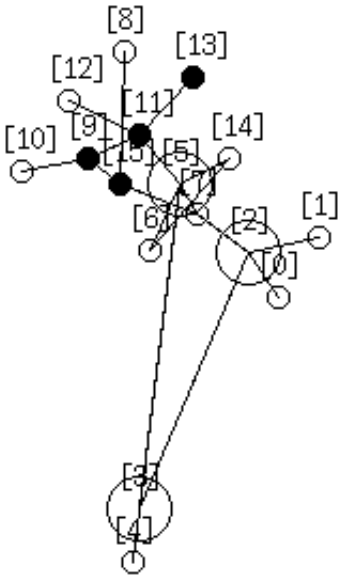

図 5.3: ゴール - ゴール間を 5 倍に設定して、鈴木のアルゴリズムを適応

[2] と [3]、[2] と [5] の間は十分離れており予定通りであるが、[2] と [5] の間が 5 倍の距離になっていない。原因は最短路を最適距離としているため [7] の項の頂点を通るパス ([2]-[7]-[5]) を使って、最適距離を計算していることである。

ゴールと変数の関係のみからなるグラフを作り、ゴール間の最適距離をそのグラフから計算するという方法が考えられるが、これでは、全く項などの共有関係が反映されず、いびつなグラフになってしまうことが予想されたため、次に説明する辺の抵抗という概念の導入によって、頂点間の距離と、その関係の強さとを分離した。

辺の抵抗の導入

辺の抵抗とは、すべての辺にある属性であり、長さとは独立している。辺の長さの導入によって、レイアウト計算の際の最適距離の計算の定義を次のように変更した。

> 頂点間の最適距離は、その 2 つの頂点を接続するすべてのグラフ上のパスのうち、パスに沿った抵抗の合計の最小のものを選び、そのパスに沿った辺の長さの合計とする。

例を図 5.1.1に示す。

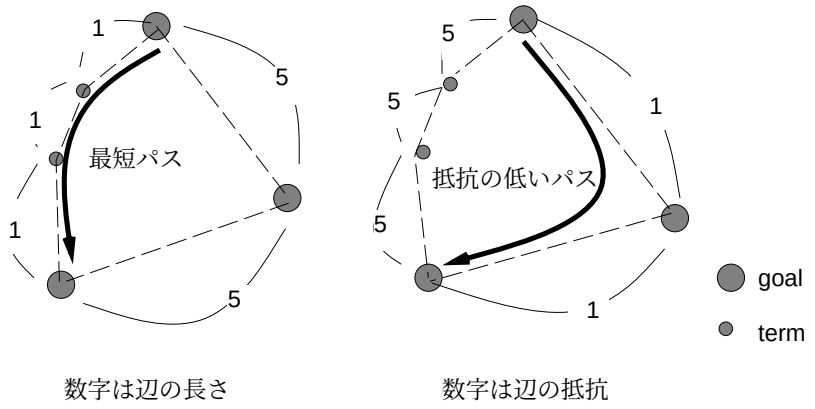

図 5.4: 辺の抵抗の概念

これを用いて、先ほどのアルゴリズムの最適距離計算部を変更し、同じモデルについてレイアウトをし直した。(図 5.1.1)

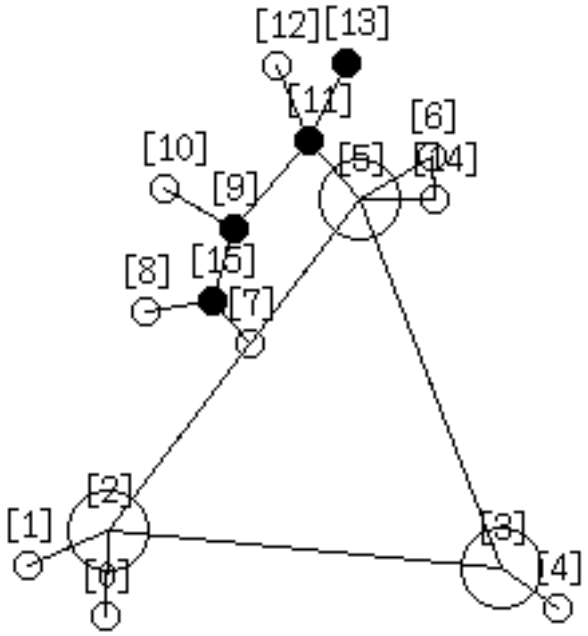

図 5.5: 辺抵抗の概念を導入したアルゴリズムによるレイアウト

結果、ゴール同士は十分はなれ、その間に functor が散らばるようになり、 functor の接続関係もうまく反映したものが得られた。

構造の強調実験

さらに、辺の抵抗を用いて、構造の持つ意味を強調する実験を行なった。この実験では具体的に、 [7][15][9][11] を list 構造の cons ととみなし、 cons 間の距離を短く、抵抗は従

来通りにして、計算した。結果は図 5.1.1である。

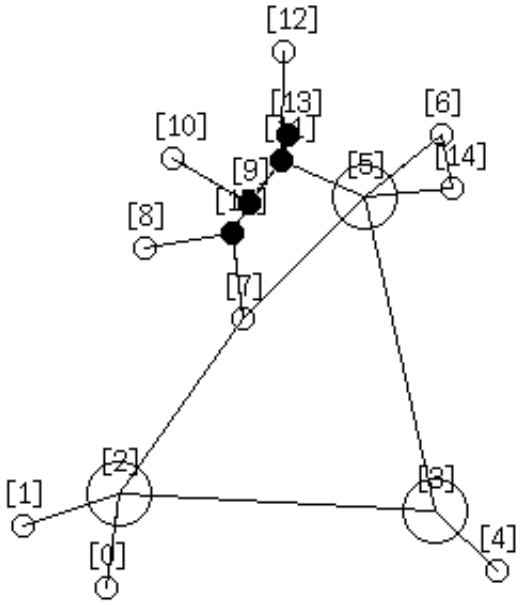

図 5.6: リスト構造を強調

少しであるが、 list 構造部分 (黒く塗りつぶされている) が近付いている。相関関係が強調されているのがわかる。

5.1.2 同アルゴリズムの評価

鈴木のアルゴリズムには、 Visual Programming System におけるモデルレイアウトに関して、次の特性がある。

- **初期配置の相対位置関係を反映した配置をする。**
  鈴木のアルゴリズムは、初期配置から力学的制約を満たす配置へと漸近的移動を繰り返すアルゴリズムといえる。これは、初期配置の位置を保存する傾向があることを意味する。これに関して実験をした。図 5.1.1のモデルから、 [5] が fork して、新たに [5]、 [16] というゴールができたとして、新たな [5][16] は前の [5] の頂点の位置付近、他の頂点は前の位置を初期配置としてアルゴリズムを適用した。結果として、図 5.1.2が得られた。幾何学的形状と相対位置はほぼ保存され、ユーザの認知に無理がないであろうことがわかる。ユーザがモデルを編集する際にも、そのモデルのレイアウトの結果はユーザの操作による配置関係を保存することが予想され、ユーザのモデル認知を容易とする。

- **部分的変更による再レイアウトにかかる時間の減少**
  状態の遷移によってモデルが変更を受けるが、モデルが大きくなると、一般的にはモデルの一部分しか書き換えられない。その際、モデルに対応するグラフは既にほとんどレイアウトは完了しており、新たに書き換えられた部分に関する頂点関係の制約の

みが満たされていないという状態になる。この場合、大きな頂点の移動がないので、制約を満たすための繰り返しは少なくて済むことが予想される。実際、図 5.1.2に要する計算は非常に短く済んだ。他にも、 editor などでは、ユーザの頂点配置はある程度レイアウトされていることが予想され、再レイアウトをするのにかかる時間は非常に短く済むと思われる。

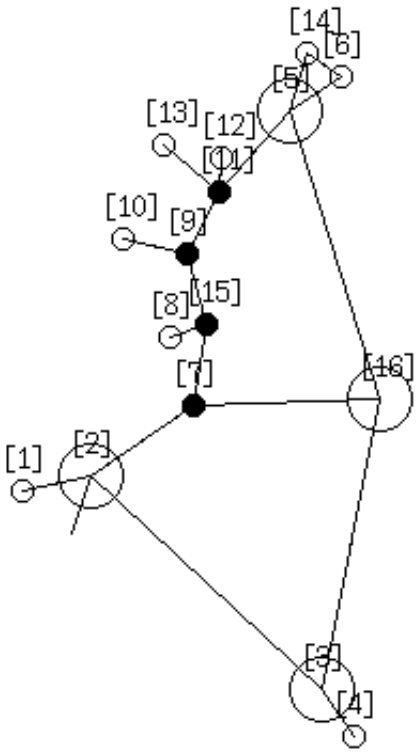

図 5.7: モデルが一部変更されたことを想定してのレイアウト

- **Threshold**
  アルゴリズム中の ε は、制約の強さであるといえる。これは、時間や、前の状態からの移動量とのトレードオフであり、これを調節することで、場合に応じたレイアウトを実現できる。

これらのことから、このアルゴリズムは、 PP のモデル配置に関しては適切な選択であることを確信した。

algorithm の PP への適用時の問題点

しかしながら、次の問題があり、今後の研究課題である。

- **Scalability**

  この algorithm は、実際の FGHC の program の視覚化において有効であろうか。

実用的なプログラムにおいては、 goal が 1000 単位になってしまう。表示上工夫をして、グラフ構造自体を小さなものにしたりする方法はあるが、基本的にグラフの数の 3 乗に比例して計算量は増えるものであるため、 goal の数が増えると、 Interactive 環境において十分な response をすることが難しくなってくる。

- **I/O Mode**

  FGHC の Program では、 goal の引数の入出力が一意に決まることが多く、これを配置に生かすことはかなり効果的であると考えられる。

  具体的には、配置計算をした後、 goal の引数につながる辺を探し、方向を保存したままで、入力ならば、そのゴールに向かうベクトル、出力ならば、 goal から出てゆく方向のベクトルと見なして、そのすべてのベクトルの合成ベクトルが下を向くように図全体を回転させることで実現できるであろう。

  プログラマの負担や、本来の FGHC の仕様から考えて、入出力は自動入出力 mode 解析によるのが望ましい。 [17]

5.1.3 PP field のモデルのアニメーション

アニメーションに関しては既に 4.3で説明した。ここでは実装の問題について言及する。

配置アルゴリズムとアニメーション

前述の配置アルゴリズムは、変移の少ないところは、相対的な位置関係などがあまり変わらなかった。この性質から、アニメーションに関しても、この配置アルゴリズムをそのまま使えば良いことがわかる。

アニメーションの手順

では、具体的にはどのようなアニメーションにするのが良いのか。 PP の実行モデルとアニメーションについて分析をした。

1. GHC という Programming 言語の計算モデルである、非同期細粒度並列ということを考えると、すべてのゴールが非同期にゴールや項の生成を行ない、それにともなって、配置アルゴリズムの制約を計算し直すのが一番自然である。

2. 小さな単位、例えばゴールのコミット、ボディユニファイの一ステップを同期させて、同期する度に配置し直すというのは、直観的にわかり易いと思われる。

どちらかといえば後者の方が実現し易いであろう。後者を用いて実現することを考えると、次の流れでアニメーションは進むことになる。

アニメーションの流れ

現在は、アニメーションは細かい単位で同期させる方針としている。

1. まず、変形をアニメーションとして見せる

(a) commited goal の body part の mini model の表示。

(b) Unify する functor の融合 / 引数の接続

(c) 参照されない functor の消滅

2. 次に、配置計算の結果を反映させる

(a) 配置計算のとき、 Unify される 2 つの functor は、同じ頂点として計算する。それぞれの引数はその頂点から出ている辺とする。

(b) 配置計算の結果の場所へとすべての object を同時に動かす。この時、すべての object は最終的な場所へ同時につくように速度を調整する。

アニメーションに関しては現在実装中である。

第 6 章

結論

Visual Programming System のあり方について考察し、それを踏まえて従来の Visual Programming System をベースに、 PP の Evaluator を設計した。

また、その System の実現のために要素技術、特にユーザの認知において重要と思われる Model の Layout の Algorithm を調べ、鈴木の algorithm について検討・改良・実験し、その有用性を確かめた。

さらに、 PP での使用について検討し、このアルゴリズムが十分に有用であることを確認した。

最終的な目標はこのレイアウトアルゴリズムと、アニメーションの流れを合わせて、さらに入力方法の設計をし、 PP を完成させることである。

謝辞

グラフレイアウトのアルゴリズムに関してお世話になった鈴木和彦氏 (慶応大学)、鎌田久氏 (ACCESS)、貴重なコメントをいただいた WISS'94 のメンバーに感謝する。

また、様々な形で支援をしていただいた筑波大学ソフトウェア研究室のメンバー、特に、田中研究室の古川英司氏に感謝する。

参考文献

[1] David McIntyre: Visual Lanugages (comp.lang.visual FAQ), 1994.
`http://union.ncsa.uiuc.edu/HyperNews/get/computing/visual.html`.

[2] 原田賢一編: 構造エディタ, 共立出版,1987.

[3] G. Battista, P. Eades, R. Tamassia and I. G. Tollis: Algorithms for Drawing Graphs: an Annotated Bibliography, 1993.
`ftp://wilma.cs.brown.edu/pub/gdbiblio.ps.Z`.

[4] P. Eades and R. Tamassia: Algorithms for automatic graph drawing: an annotated bibliography, Technical Report CS-89-09, Brown University, Department of Computer Science, Providence, RI 02912, 1989.

[5] J. Tanaka: Visual Programming System for Parallel Logic Languages, The NSF/ICOT Workshop on Parallel Logic Programming and its Program Environments, the University of Oregon, pp.175-186, 1994.

[6] 田中二郎: 並列論理型言語 GHC のビジュアル化の試み, インタラクティブシステムとソフトウェア I, 日本ソフトウェア科学会 WISS'93, 竹内彰一 編, 近代科学社, pp265-272, 1994.

[7] 中野勝次郎, 田中二郎: ビジュアルプログラミングシステムにおけるモデルの視覚化アルゴリズム, インタラクティブシステムとソフトウェア II, 日本ソフトウェア科学会 WISS'94, 竹内彰一 編, 近代科学社, pp205-214, 1994.

[8] 田中二郎, 神田陽治編: インタフェース大作戦, 共立出版, pp41-63,1995.

[9] M.H. Brown: Exploring Algorithms Using Balsa-II, IEEE Computer, Vol. 21, No. 5, pp. 14-36, May 1988.

[10] M.H. Brown: Zeus: A System for Algrothm Animation and Multiple-View Editing, Proc. of 1991 IEEE Workshop on Visual Languages, pp. 10-27, Oct.

[11] J.T. Stasko: Tango: A Framework and System for Algorithm Animation, *IEEE Computer*, Vol. 23, No. 9, pp. 27-39, Sep. 1990.

[12] K.M. Kahn: Concurrent Constraint Programs to Parse and Animate Pictures of Concurrent Constraint Programs, Poc. FGCS'92, Tokyo, pp.943-950, 1992.

[13] P. T. Cox and T. Pietrzykowski: Using a Pictorial Representation to Combine Dataflow and Object-Orientation in a Language Independent Programming Mechanism, International Computer Science Sonference 88, HongKong, pp. 605-704, 1988.

[14] The Gunakara Sun Systems: Prograph tutorial, the Gunakara Sun Systems, 1989.

[15] T. Kamada and S. Kawai: An Algorithm for Drawing General Undirected Graphs, Information Processing Letters, vol. 31, pp. 7-15, 1989.

[16] K. Suzuki and T. Kamada: Simple Undirected Graph Drawing Algorithm for Visual Understanding of Graph Structures (unpublished)

[17] K. Ueda: Guarded Horn Clauses, ICOT Technical Report TR-103, ICOT, 1985.