目 次

第1章　はじめに

生体分子ネットワーク構造や, プログラムのフローチャート図など, データ間の関連性がある構造を表現するために, しばしばグラフ構造が用いられる. グラフとは有限のノード集合と, ノード間を結合するエッジ集合からなるデータ構造である. グラフ表現されたデータから人間が情報を得るには, グラフを適切に視覚化し, ノード間の関係やノードの情報が分かりやすく見えていることが望ましい. しかし, グラフを分かりやすく視覚化することは難しく, グラフが大規模になると人間が手作業で視覚化することは困難である. 任意のグラフからグラフの二次元描画を生成する問題は, グラフ描画問題と呼ばれ, 今日までに多くの手法が提案されている.

グラフ描画は, 人間が理解しやすいことが望ましいが, グラフごとにノードやエッジに与えられる意味は異なり, 人によって見易さの好みが異なるため, グラフ描画に最も良いという基準を与えることは出来ない. そのため, グラフの特徴などを考慮した適切なグラフ描画を考えていく必要がある.

特にノードの情報に着目したい場合, グラフ描画上にノード情報を表示させるために, ノード間に一定以上の空間を確保したいケースが考えられる.

この目的のため, 吉田英聡ら [1] によって, グラフノードをパターン, グリッド状の格子点をテキストとみなした二次元点集合の近似照合を用いて, 多項式時間で動作する格子状配置アルゴリズムが提案された. この手法により, 任意のグラフのグラフ描画において, 元々のグラフ描画の位置関係をある程度維持しながら, グラフのノード間にグリッド幅以上の空白を確保したグラフ描画の生成が可能となった.

しかし, 二次元点集合の近似照合における時間と領域の計算量は次数が高く, 現実

的なサイズのグラフ描画に対して直接適用することができなかった. これに対し [1] では, kd-木分割やメッシュ状分割などでグラフを分割統治し, 大規模なグラフに実行できるようにした. しかし, ユーザが対話的な環境で, グラフのグリッド配置アルゴリズムを利用するには, ユーザがストレスを感じない程度の素早い応答時間が要求され, より高速なアルゴリズムが求められている.

そこで本研究では, 二次元点集合近似照合におけるテキストに相当する点集合が, 一様な格子であることを利用する. 二次元点集合近似照合を, パターンに相当するグラフノード点集合を包含する境界矩形を, 格子点集合上に配置する問題にみなす.

本論文の以降の構成は次の通りである. まず, 第 2 章では, 任意の点集合に対する境界矩形および境界矩形拡張列と, 境界矩形拡張列の格子配置における編集距離の定義を行い, 境界矩形の編集距離の再帰的定義と, 時間・空間ともに $O(n^2)$ の計算量で実行可能な動的計画法 (DP) による解法を示す. 第 3 章では, グラフの格子配置アルゴリズムと, 重複点に関する例外的な処理について述べる. 第 4 章では, グラフの格子配置アルゴリズムの実行時間と, そのグラフ描画結果を示す. 第 5 章では, 本研究における結論と今後の課題について述べる.

第2章　境界矩形拡張列の格子配置問題とその解法

本章では, 二次元点集合のグリッド近似照合と, 境界矩形拡張列の格子配置について述べる. まずはじめに, これまで行われてきた研究について紹介する.

2.1　吉田英聡らの定義

二次元点集合の近似照合の定義として, 吉田英聡らは [1] で次のように定義した.

二次元平面上の点は正整数の組 $p = (p.x, p.y)$ で表す. ここでは, 辞書式順序により二次元平面上の点に全順序をつける. 点集合 P について X 軸, Y 軸の順, および Y 軸, X 軸の順に辞書式順序に並べた列をそれぞれ P_{XY}, P_{YX} とする. $(P)_R$ は P_{XY} の最後の点の添え字, $(P)_T$ は P_{YX} の最後の点の添え字とする. $|P|$ は P の要素数である.

任意の $1 \leq i, j \leq n$ について, P の部分集合 $P_{\leq p_i}$ を P_{XY} の最初から p_i までの点からなる集合, $P^{\leq p_i}$ を P_{YX} の最初から p_i までの点からなる集合とする. P の部分集合 $P[i, j]$ は

$$P[i, j] = P[j, i] = \{P_{\leq p_i} \cap P_{\leq p_j} \cap P^{\leq p_i} \cap P^{\leq p_j}\}$$

とする. また, $P[i, j]$ から $p(P[i, j])_R$ を除いた集合を $P[i, j]_-$, $p(P[i, j])_T$ を除いた集合を $P[i, j]^-$ とする. 以後, 点集合が明らかな場合は P を省略する.

定義 1 n 個の点集合をそれぞれ P, Q とする. このとき, $1 \leq i, j, k, l \leq n$ について

$P[i,j]$ と $Q[k,l]$ の編集距離 $d(i,j;k,l)$ を

1. $|P[i,j]| = |Q[k,l]| = 1$ なら $d(i,j;k,l) = 0$,

2. $|P[i,j]| \neq |Q[k,l]|$ なら $d(i,j;k,l) = \infty$,

3. それ以外なら $d(i,j;k,l) =$
$$\min \left\{ \begin{array}{l} d([i,j]_-;[k,l]_-) + |p([i,j])_R - p([i,j]_-)_R - (q([k,l])_R - q([k,l]_-)_R)|, \\ d([i,j]^-;[k,l]^-) + |p([i,j])_T - p([i,j]^-)_T - (q[k,l])_T - q([k,l])^-)_T)| \end{array} \right\}$$

と定義する. $|\cdot|$ は L_1 ノルムである.

定義 2 T と P をそれぞれサイズ m,n の点集合とし, $\epsilon \geq 0$ とする. T の部分集合 T' と P の編集距離が ϵ 以下となるとき, P は T と位置 $(T')_R$ において, ϵ 以下で近似照合するという.

この近似照合について [1] では, T と P の近似照合を動的計画法で計算するアルゴリズムが提案された. 二次元点集合の近似照合アルゴリズムは, $O(n^2m^4)$ で動作する.

2.2 境界矩形拡張列とその格子配置における編集距離

テキスト T が格子点集合であるならば, 任意の部分格子点集合 $T[k,l]$ とパターン $P[i,j]$ との編集距離は, $T[k,l]$ に依存せず $P[i,j]$ において一意である. この性質を利用することで, 格子点集合と二次元点集合の近似照合は, より小さい DP 表で計算可能であることを次に示す.

二次元平面上の点は正整数の組 $p = (p.x, p.y)$ で表す. 点集合 P について X 軸, Y 軸の順, および Y 軸, X 軸の順に辞書式順序に並べた列をそれぞれ P_{XY}, P_{YX} とする. $(P)_R$ は P に含まれる点のうち P_{XY} における最後の点の添え字, $(P)_T$ は P に含まれる点のうち P_{YX} における最後の点の添え字とする. $|P|$ は P の要素数である.

P の境界矩形とは, P の点を全て含み各辺が座標軸に平行で最小の長方形である.

任意の $0 \leq i, j \leq n$ について, P の部分集合 $P_{\leq i}$ を P_{XY} の最初から添え字 i までの点からなる集合, $P^{\leq j}$ を P_{YX} の最初から添え字 j までの点からなる集合とする. ただし, $P_{\leq 0} = P^{\leq 0} = \emptyset$ とする. P の部分集合 $P[i,j]$ は

$$P[i,j] = \{P_{\leq i} \cap P^{\leq j}\}$$

とし, $P[i,j]$ の境界矩形と $P[i,j]$ を同一視する. $P[i,j]$ から $p(P[i,j])_R$ を除いた点集合を $P[i,j]_-$, $p(P[i,j])_T$ を除いた集合を $P[i,j]^-$ とする.

二次元格子点集合 $\mathrm{Grid}(d)$ は

$$\mathrm{Grid}(d) = \{(dx, dy) | x, y \in \mathbb{N}\} (d \in \mathbb{Z})$$

とする. 境界矩形 $P[i,j]$ が $P[i,j] \subseteq \mathrm{Grid}(d)$ を満たすならば, $P[i,j]$ は格子配置である.

サイズ n の点集合を P_n とすると, 境界矩形拡張列とは, $P_n[0,0]$ $(|P_n[i,j]| = 0)$ から始まり $P_n[n,n]$ $(|P_n[n,n]| = n)$ まで, 1 点ずつサイズが増加する真に包含関係にある境界矩形の列 $P_0 \subset ... \subset P_n$ である. この時, $P_n[(P_k)_R, (P_k)_T] = P_k$ とすると, P_{k-1} は $P_n[(P_k)_R, (P_k)_T]_-$ または $P_n[(P_k)_R, (P_k)_T]^-$ である.

境界矩形拡張列において, P_k に含まれ P_{k-1} に含まれない点 p を, P_{k-1} から相対位置で格子配置された点を $\mathrm{ptrans}(p, P_{k-1}) = p'$ とし, P_k が ptrans によって拡張される矩形編集距離 $\mathrm{trans_d}$ を

$$\mathrm{trans}_d(p', P_k) = |(P_{k-1} \cap \{p'\}) - P_k|$$

とする. $|\cdot|$ は L_1 ノルムを表す.

点集合の格子配置における編集距離を次に定義する.

定義 3 n 個の点集合を P, 幅 d の二次元格子点集合を Grid(d) とする. このとき, $P[i,j]$ の Grid(d) における編集距離 $D_d(P[i,j])$ を

1. $P[i,j] = \{\}$ のとき, $D_d(\{\}) = 0$

2. それ以外のとき,

$$D_d(P[i,j]) = \min \left\{ \begin{array}{l} D_d(P[i,j]_-) + \mathrm{trans_d}(\mathrm{ptrans}(p(P[i,j])_R, P[i,j]_-), P[i,j]), \\ D_d(P[i,j]^-) + \mathrm{trans_d}(\mathrm{ptrans}(p(P[i,j])_T, P[i,j]^-), P[i,j]) \end{array} \right\}$$

と定義する.

2.3 二次元点集合の格子配置とその解法

定義 4 P をサイズ n の二次元点集合, Grid(d) を幅 d の二次元格子点集合とし, $\epsilon \geq 0$ とする. P に Grid(d) において編集距離が ϵ 以下となる境界矩形拡張列が存在するとき, P は ϵ 以下で格子配置されるという.

二次元点集合の格子配置の解法を以下に示す.

まず, $1 \leq i,j \leq n$ のすべての組 (i,j) から, $1 \leq r \leq n$ において, $P[i,j]$ のサイズが r となる組の集合のリストを作成する. このリストの各々の要素は, サイズ r の部分点集合を表す添え字の組全ての集合であり, リストの作成は $O(n^2)$ 時間でできる.

次に, 各々のリストの要素に含まれるすべての組 (i,j) から $D_d(i,j)$ への写像 D を生成する. p_i は, $P[i,j]$ に P_{XY} における i 番目の要素とする. p_j も同様に, P_{YX} における j 番目の要素とする. i_p は, $P[i,j]$ に含まれる $(P[i,j])_R$ を除く最右の点の P_{XY} における添え字である. j_p も同様に, $P[i,j]$ に含まれる $(P[i,j])_T$ を最上の点の P_{YX} における添え字である. $\mathrm{t_d}(p,P)$ は, P に含まれない点 p を P から相対距離で幅 d の格子に配置した点を p' としたときの, $P \cap \{p'\}$ と $P \cap \{p\}$ の L_1 ノルムの差である.

写像 D はアルゴリズム 1 で計算する. 写像 D には $O(n^2)$ 領域必要である. i_p および j_p は, 前の点を探すためにソート列を走査するループが必要だが, このループ回数は元々添え字の組のリストを生成する段階で省かれた組の数以下におさまるので, このアルゴリズムは, 時間・計算ともに入力の点集合のサイズ n に対して, $O(n^2)$ 時間で動作する.

Algorithm 1 写像 D の計算

```
for r from 1 to n do
  for each (i, j) in List(P, r) do
    if p_i = p_j then
      D_d(i, j) ← D_d(i_p, j_p) + t_d(p_i, P[i_p, j_p])
    else
      D_d(i, j) ← min{D_d(i, j_p) + t_d(p_i, P[i_p, j]), D_d(i_p, j) + t_d(p_j, P[i, j_p])}
    end if
  end for
end for
```

アルゴリズム 2 は, 写像 D をバックトレースすることで, 最小の編集距離で近似照合する境界矩形拡張列を求める.

境界矩形拡張列を生成するループは入力サイズだけ繰り返すので, このバックトレースは $O(n)$ 時間で実行できる.

アルゴリズム全体を通して, $O(n^2)$ 時間と $O(n^2)$ 領域で動作する.

Algorithm 2 境界矩形拡張列の計算

$L \leftarrow []$
$i \leftarrow n$
$j \leftarrow n$
while $i > 0$ or $j > 0$ **do**
  push $P[i, j]$ into L
  if $p_i = p_j$ **then**
    $i \leftarrow i_p$
    $j \leftarrow j_p$
  else if $D_d(i_p, j) + \mathrm{t_d}(p_i, P[i_p, j]) \leq D_d(i, j_p) + \mathrm{t_d}(p_j, P[i, j_p])$ **then**
    $i \leftarrow i_p$
  else
    $j \leftarrow j_p$
  end if
end while

第3章　グラフの格子配置アルゴリズム

3.1　グラフの格子配置アルゴリズム

境界矩形拡張列の格子配置にもとづく編集距離を求めるグラフ格子配置アルゴリズムは次のように定義できる.

1. 二次元点集合 P の格子配置における編集距離および, その境界矩形拡張列 $P_0, ..., P_n$ を計算する.

2. 各々の境界矩形 P_k において追加された点 p を, P_{k-1} を格子配置する編集距離だけ右上方向に平行移動と, P_{k-1} から相対的に格子配置する平行移動を行った点 p' に写像する.

ステップ 1 は $O(n^2)$ で動作する. ステップ 2 の写像計算はステップ 1 で生成した DP 表 D_d を利用することで定数時間で動作し, 計算は入力点それぞれに行うため, ステップ 2 全体では $O(n)$ で動作する.

アルゴリズム全体を通して, グラフの格子配置は $O(n^2)$ で動作する.

3.2　グラフに入力される重複した点の対策

二次元点集合に重複点が存在するとき, 現在の格子配置アルゴリズムでは冗長な形状に展開されることもある. 元々, 重複点が与えられるということは, グラフノード間に密接な関係があると考えることが人間にとって自然であるので, 隣接関係を

維持しつつグリッド状に展開する方法をアルゴリズムの拡張として次のように定義する.

二次元点集合 P において, 重複点を表現するため, 各々の点 p に対し, 重複点の個数 p^m を与える. ここで, p^m に対して L_1 ノルムが最小になる境界矩形による格子配置を与えることができる.

グラフの格子配置アルゴリズムにおいて, まず二次元点集合の重複点を同一視し, 各々の点の重複点の個数を記録する.

ステップ 1 において, 各々の境界矩形の編集距離を計算する際, 追加される点が

1. $p^m = 1$ ならば, 通常の方法で編集距離を求める.
2. $p^m \geq 2$ ならば, 重複点を全て格子配置するために必要な境界矩形だけ編集距離に追加する.

ステップ 2 において, 写像する点が $p^m \geq 2$ ならば, 各々の重複点を, P_{k-1} から相対位置で格子配置されるように格子配置する.

3.3 伸長したグラフの畳み込み処理

境界矩形拡張列の計算では, 矩形が右か上のどちらか一方に拡張されるとき, 最低でもグリッド幅だけのスペースを空けることになる. 入力サイズに対してグリッド幅が疎な場合, グリッド配置は大きく展開されるため, 後処理でグラフを畳み込む処理を追加する.

P がグリッド配置された点集合 P' において, 点集合の順序関係 P_{XY}, P_{YX} は保存される. shiftLeft およびshiftBottom は, 点列をグリッド幅だけ左および右に平行移動する関数であり, subList(L, i) は点列 L の i を含む残りの部分点列とする. グリッド配置後に, アルゴリズム 3 によりグリッド配置を編集する.

Algorithm 3 グラフの畳み込み計算

$p \leftarrow P_{XY}[1]$
for i from $P_{XY}[2]$ to $P_{XY}[n]$ **do**
  if $p.x \leq i.x - d$ and $p.y \leq i.y$ **then**
    shiftLeft(subList(P_{XY}, i))
  end if
  $p \leftarrow i$
end for
$p \leftarrow P_{YX}[1]$
for i from $P_{YX}[2]$ to $P_{YX}[n]$ **do**
  if $p.y \leq i.y - d$ and $p.x \leq i.x$ **then**
    shiftBottom(subList(P_{YX}, i))
  end if
  $p \leftarrow i$
end for

第4章　実験結果

この章では, 境界矩形拡張列によるグラフの格子配置アルゴリズムの実装と, アルゴリズムの性能を調べる実験の結果を述べる. プログラムは C++ (MinGW32-gcc-4.6.2 x86) で実装し, Intel Core i7-3770 CPU (3.40GHz) のマシン上で実行した. マシンの RAM は 8.00GB だが, プログラムは 32bit 環境向けにコンパイルしたため, プログラムが利用可能なメモリアドレス領域は 4GB 未満であった.

4.1　グラフの格子配置アルゴリズムの実行速度

実装したプログラムに実際に点集合を入力として与え, プログラムの実行時間と, 点集合のグリッド配置によるグラフの伸長を計測した. 図 4.1, 図 4.2 は, サイズ 100 程度の点集合に対して, アルゴリズムを実行した結果である. 実行時間は 1msec 以下で動作した.

次に, サイズ 10000 までの点集合に対して実行速度を計測するため, プログラムを実行した. 表 4.1 にその結果をまとめる.

実行速度は $O(n^2)$ におさまる範囲で遷移しており, サイズが 3000 程度の点集合に対しても, 十分に対話的環境に利用できることが確認された.

サイズが 8000 以上の入力に対しては, 実験環境では DP 表を生成するための連続した大きなヒープ領域を確保することができなかったので, プログラムが実行できず, 検証することができなかった.

表 4.1 入力サイズと実行時間 (標準)

サイズ (点)	実行時間 (sec)
1000	0.093
2000	0.343
3000	0.765
4000	1.435
5000	2.231
6000	3.260
7000	4.571
8000	6.035
9000	bad-alloc

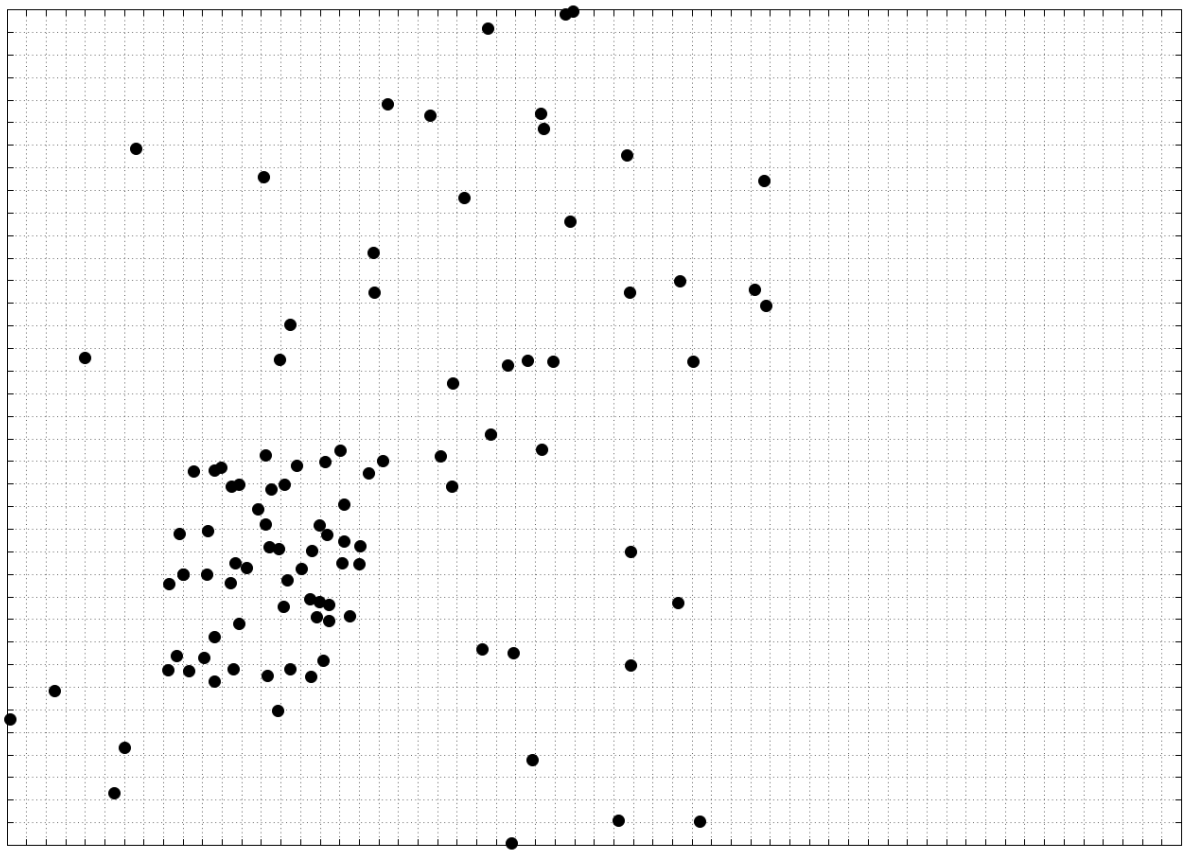

図 4.1 Size100 のグラフ

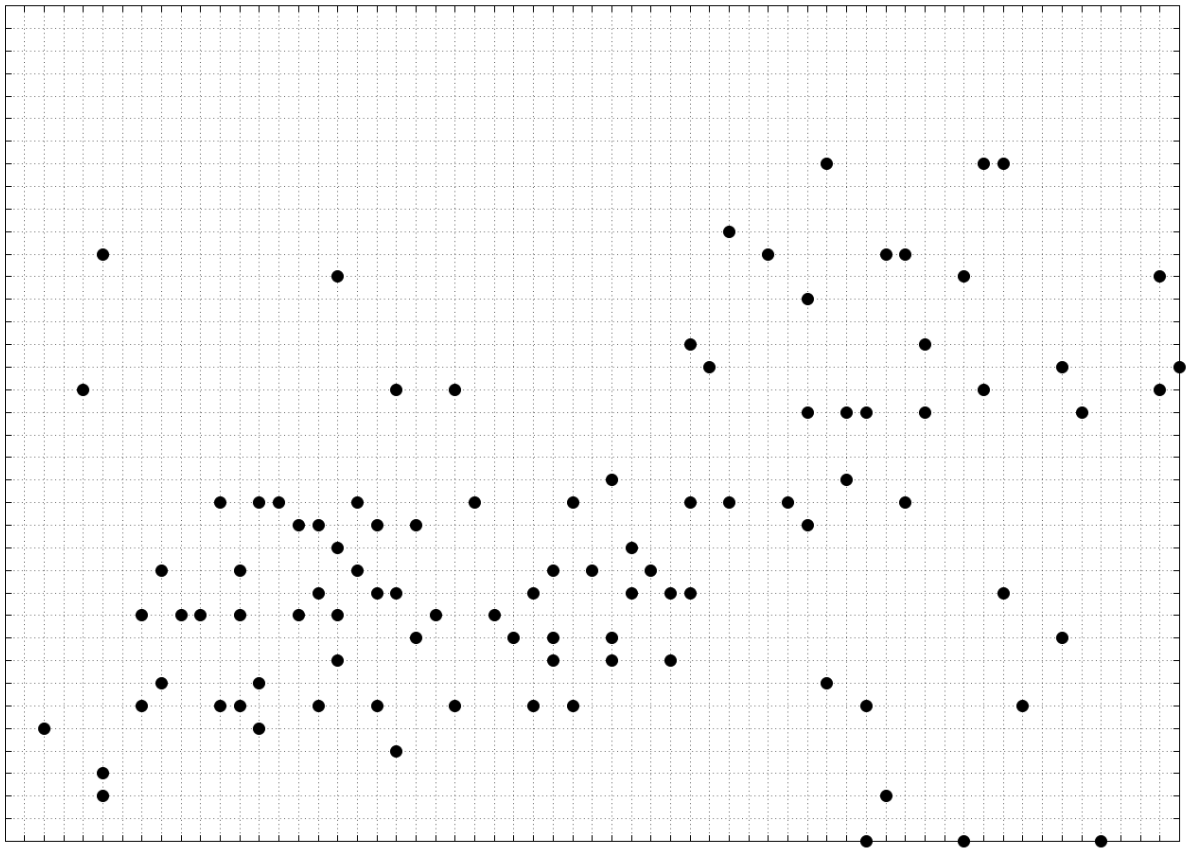

図 4.2 矩形拡張列による格子配置

第5章　終わりに

本研究では, マッチング対象を格子点に限定した点集合の近似照合を定式化した. まず点集合に対する境界矩形と境界矩形拡張列を定義し, 境界矩形拡張列の格子配置における編集距離を再帰的に定義した. 次に, 動的計画法による境界矩形拡張列の格子配置における編集距離を, 空間および時間ともに最悪計算量が $O(n^2)$ で実行できるアルゴリズムを提案した.

そして, 境界矩形拡張列によるグラフの格子配置アルゴリズムを提案し, 実行時間とグラフの伸長に関する実験を行った. さらに, 重複点の展開や, グラフ展開後の畳み込み処理の追加により, 格子配置の伸長を抑え, 元のグラフ描画における点の局所性を, 格子配置後のグラフに反映させることができた.

実装したプログラムは, サイズ 3000 程度まで十分に対話的に利用できるだけの速度で動作し, アルゴリズムが実用的な速度で利用可能な実装ができることを示した. しかし, グラフのサイズに対して疎なグリッド幅で展開する場合, グリッド配置は右上方向に伸長し, グラフの全体像は線形に近づいてしまうため, グリッド幅は入力点のサイズに対して適切な幅を与えなければならない.

謝辞

本研究を進めるにあたり, 細部に至るまでご指導頂いた下薗真一准教授に深く感謝いたします. また, 日々の研究でお世話になった研究室の皆様に深く感謝の意を表します.

参考文献

[1] 吉田英聡, 二次元点集合近似照合によるグラフの格子状配置アルゴリズム, 電子情報通信学会総合大会講演論文集 2008 年 情報・システム (1), ”S-17”-”S-18”, 2008-03-05