Vol.2018-AL-167 No.11
2018/3/8

有向グラフ描画における矢じり配置問題に対する厳密解法

城戸 直人[1,a)]　増田 澄男[1]　山口 一章[1]

概要：有向グラフ描画において辺の向きを示す矢じりを，他の矢じりや頂点，辺となるべく重ならないように配置する問題について考える．この問題を簡単化したある問題は NP 困難であることが知られており，現在，それに対する厳密解法が存在する．本研究では，簡単化のされていない問題に対する厳密解法を提案する．これは，矢じり配置問題を混合整数計画問題に帰着したものであり，実行時間削減のための工夫も行っている．提案する厳密解法によって，従来法に比べて良い矢じり配置が得られることと，高速化の処理が有効であることを，計算機実験により示す．

キーワード：グラフ，描画，NP 困難，厳密解法

An Exact Algorithm for the Arrow Placement Problem in Directed Graph Drawings

Naoto Kido [1,a)]　Sumio Masuda[1]　Kazuaki Yamaguchi[1]

Abstract: We consider the problem of placing arrowheads, which indicate the direction of edges in directed graph drawings, without making them overlap other arrowheads, vertices, and edges as much as possible. A simplified one of this problem is known to be NP-hard, and now there is an exact algorithm for it. In this report, we present an exact algorithm for this problem which is not simplified. We formulate the arrow placement problem as a mixed integer programming problem. We also present a method to reduce execution time of our algorithm. Computer experiments show that our algorithm gives a better placement than the conventional method and that the procedure for reducing execution time is effective.

Keywords: graph, drawing, NP-hard, exact algorithm

1. まえがき

グラフの自動描画アルゴリズムに関して多くの研究が行われており [1]，無向グラフについては力指向アプローチによる様々なアルゴリズムが知られている [2], [3]．有向グラフを描画する 1 つの自然な方法は，力指向描画法によって頂点と辺の配置を定めた後，辺の向きを示す矢じりを描き加えることである．しかし，各辺の先端に矢じりを配置するような機械的な方法では，1 つの頂点に多くの辺が接続している場合に矢じり同士の重なりが多く発生し，辺の向きの判別が困難になるという問題がある．

このような問題に対するアプローチとして，Binucci ら [4] による有向グラフ描画における矢じりの配置位置を求めるアルゴリズムが存在する．この手法は，頂点または辺と重ならないように矢じりを配置できる位置を矢じり配置位置の候補としていくつか定めておき，それらの中から矢じり同士の重なりが最小になる配置位置を選択するものである．Binucci らは，このモデルにおける最適な配置位置を求める問題が NP 困難であることを示したうえで，それに対する発見的手法と厳密解法を示した．

有向グラフ描画の矢じり配置に関連する問題として，地図ラベル配置問題（地名などを表すラベルを地図中の地点に配置する問題）が知られている．この問題に対して，あらかじめラベル配置位置の候補を定めておく方法も研究さ

[1] 神戸大学大学院工学研究科
Graduate School of Engineering, Kobe University, Kobe-shi 657-8501, Japan
[a)] 173t222t@stu.kobe-u.ac.jp

れている [5] が，各地点に接する範囲で自由にラベル位置を移動できるスライダーモデル [6] を用いた方が，より多くのラベルを配置できる可能性が生まれる．筆者らは，スライダーモデルを用いた発見的手法が，Binucci らによる厳密解法よりも良い矢じり配置を求めうることを示した [7]．

本研究では，主に以下の二つのことを行った．一つ目は，従来法に比べて矢じり配置の自由度を高めたモデルにおける矢じり配置問題に対する厳密解法の提案である．自由度を高めるとは具体的には，矢じりの配置位置を離散的に限定しないスライダーモデルを用いることや，矢じりと頂点，矢じりと辺の重なりを許すことなどである．二つ目は厳密解法の高速化の工夫である．厳密解法では，問題を混合整数計画問題として定式化し，それを数理計画ソルバーである CPLEX[8] に入力することで厳密解を求める．このとき，問題を CPLEX に入力する前に提案する前処理を行うことで，実行時間を削減することができる．

2. 矢じり配置問題

長さ 2 の閉路を持たない弱連結有向グラフ $G = (V, E)$ に対し，V の各頂点を半径 r の円で描いて平面上に配置し，各辺 $(u, v) \in E$ を u と v の中心を結ぶ線分によって描いた描画を $\tilde{G}$ とする．各辺 $(u, v) \in E$ の向きを示すための矢じりを $\tilde{G}$ に描き加えて得られる描画を Γ とする．矢じりは外接円半径 R の正三角形で，辺上のある点にその外心が位置するように，また u, v に重ならないように配置する．Γ における矢じり配置の適切さを表す指標として，以下の四つの値を定義する．

- $crossE$：矢じりと辺の重なりの数
- $crossV$：矢じりと頂点の重なりの数
- $overlap$：矢じり同士の重なりの数
- $distance$：各辺 $e = (u, v) \in E$ に対して，矢じりの外心から v の中心までの距離を，実数 d_e を用いて $r + (d_e + 1) \cdot R$ と表すものとする．このとき，$distance$ の値は $\sum_{e \in E} d_e$ とする．

これら四つの値はどれも小さいほど良い．

入力として $(G, \tilde{G}, r, R)$ が与えられたとき，矢じり配置問題を，以下の式 (1) の値が最小になり，かつそのなかで $distance$ が最小になる Γ を求める問題であると定義する．

$$W_1 \cdot crossE + W_2 \cdot crossV + W_3 \cdot overlap \qquad (1)$$

ここで，W_1, W_2, W_3 は定数である．上式は，矢じりと辺の重なりの数，矢じりと頂点の重なりの数，矢じり同士の重なりの数の重み付き総和を表す．

3. 従来法

矢じり配置問題に関して，Binucci ら [4] による先行研究がある．Binucci らは，2 章で定義した矢じり配置問題を少し簡単化した問題に対して，発見的手法と厳密解法を提示している．

Binucci らが矢じり配置問題をどのように簡単化しているのかを説明する．まず，各辺 $e = (u, v) \in E$ に対し，v の中心からの直線距離が $r + i \cdot R$ $(i = 1, 2, ...)$ となる e 上のすべての点に，矢じり配置位置の候補（矢じり候補）として，その点を中心とする半径 R の円を配置する．最終的に，辺 e の矢じりはこれらの候補のいずれか一つに配置する．矢じり同士の重なりは，対応する円同士が重なっているかどうかにより判定する．このように，矢じりをその外接円で表現する方法を**円形モデル**と呼び，また矢じりの配置位置を離散的な箇所に限定する方法を**離散モデル**と呼ぶ．次に，各辺 e に作成した矢じり候補のうち，頂点または e 以外の辺と重なっているものを取り除く．その結果，矢じり候補がすべて取り除かれた辺に対しては，$i = 1$ である矢じり候補に矢じりを配置する．

このように矢じり候補を設定したあと，各辺の矢じりをその辺の矢じり候補の一つに配置するという条件の下で，$overlap$ が最小になり，かつそのなかで $distance$ が最小になる矢じり配置を求める．この問題は NP 困難であり [4]，Binucci らは厳密解法と発見的手法を提示している．本稿では，それらのうち，評価値がより優れた値になる厳密解法を従来法と呼ぶ．詳細は省略するが，従来法では上記の問題を整数計画問題に帰着して解を求める．

4. 提案法

4.1 概要

矢じり配置問題に対する厳密解法を提案する．提案する厳密解法では，混合整数計画問題への定式化を行って，数理計画ソルバーである CPLEX[8] に入力することで厳密解を求める．また，厳密解法の高速化のための前処理も行う．問題全体をそのまま CPLEX に入力して解かせるのではなく，あらかじめ，最適な配置位置が明らかな矢じりの配置を確定させる，矢じりが配置される可能性が全く無いと分かる範囲を特定するなどの前処理を行うことで，元の問題に比べて問題サイズの小さないくつかの部分問題に分割する．そして，求めた部分問題のそれぞれに対して CPLEX を実行することで，実行時間を削減することができる．

4.2 従来法との比較

2 章で定義した矢じり配置問題に比べて，従来法において簡単化がなされている点は主に以下の 3 つである．

(i) 離散モデルを用いる点
(ii) 円形モデルを用いる点
(iii) 矢じりと頂点，矢じりと辺の重なりが起こる可能性をあらかじめ排除している点

(i) については，連続的な任意の位置に矢じりを配置できるスライダーモデルを用いた方が，矢じり配置位置の自由度が上がるため，従来法よりも良い矢じり配置が得られるこ

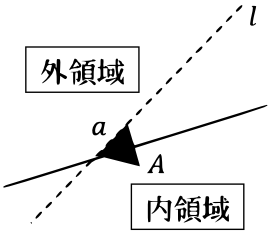


図 1 内領域と外領域
Fig. 1 Two regions devided by line l

とが分かっている [7]．(iii) は，式 (1) において W_1 と W_2 の値を十分大きくすることにより，提案法でも実現できる．(ii) については，矢じりを示す正三角形により重なりを計算する方法を用いる提案法のほうが，矢じりの重なりを正確に判定することができる．

以上から，簡単化した問題を扱う従来法に比べて，2 節で定義した矢じり配置問題に対する厳密解法である提案法のほうが，解としてより良い矢じり配置を求めることができる．

4.3 混合整数計画問題への定式化

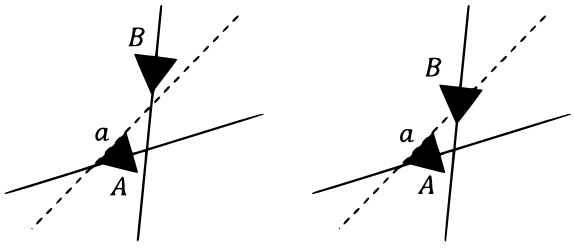
まず，矢じりの配置位置の表現方法について述べる．辺 $e = (u, v) \in E$ において，u, v の中心の位置ベクトルをそれぞれ $\vec{x}_u, \vec{x}_v$ とする．辺 e の矢じりの外心の位置ベクトルを $\vec{a}_e$ とすると，それはある実数 k_e を用いて，

$$\vec{a}_e = \vec{x}_v + k_e(\vec{x}_u - \vec{x}_v) \qquad (2)$$

と表される．つまり，辺 e の矢じり配置位置は，k_e の値と一対一に対応する．よって，辺 e の矢じりの配置位置を連続値をとる変数である k_e で表す．

次に，矢じりと辺の重なりについて述べる．辺 e の矢じりを辺上の任意の位置に配置したときに，辺 e の矢じりと重なる場合がある辺の集合を $\Phi(e)$ とする．各辺 $g \in \Phi(e)$ に対し，辺 $e = (u, v)$ の矢じりと辺 g が重なるときの k_e の取り得る値の範囲を計算し，その範囲を $\gamma_{eg} \leq k_e \leq \delta_{eg}$ とする．このとき，u と v の頂点の半径は 0 とする．矢じりと頂点の重なりについても同様で，辺 e の矢じりと重なる場合がある頂点の集合を $\Psi(e)$ とし，$\gamma'_{ev} \leq k_e \leq \delta'_{ev}$ のときに，辺 e の矢じりと頂点 $v \in \Psi(e)$ が重なるとする．

次に矢じり同士の重なりについて述べる．まず準備として，矢じり同士が重なるための必要条件を述べる．A, B をそれぞれ異なる辺の矢じりとする．また，矢じり A の任意の辺 a を延長した直線を l とし，l を境として分けられる領域のうち，a の両端点以外の A の頂点を含む領域を a の**内領域**，それと反対側の領域を a の**外領域**と呼ぶ（図 1）．このとき A と B が重なるための必要条件は，B の 3 つの頂点のうち少なくとも 1 つが内領域に属することである．またこの必要条件が満たされているとき，a は B の**包括辺**であるという（図 2）．A と B で辺は合計 6 本存在する．そ



(a) a は B の包括辺でない　(b) a は B の包括辺

図 2 a と B の関係
Fig. 2 Relations between a and B

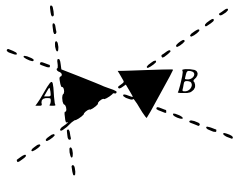
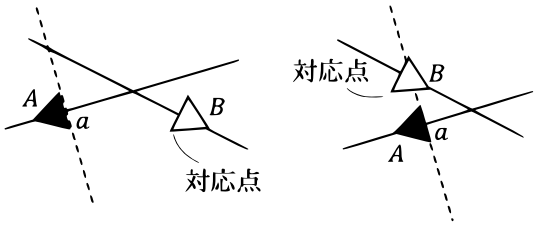


図 3 重なっていない矢じりの例
Fig. 3 An example of arrowheads not overlapping



(a)　(b)

図 4 a の対応点
Fig. 4 a and its corresponding point

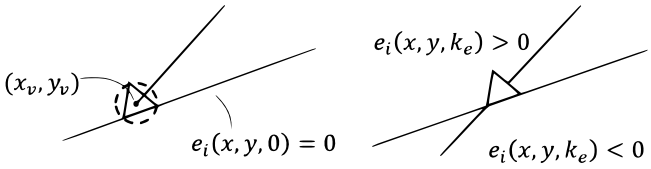
のそれぞれに対して延長線 l，外領域，内領域を考えることで必要条件も 6 つできることになる．ここで，以下の補題が成り立つ（証明は省略）．

補題 1 矢じり A, B が重なるための必要十分条件は，6 本の辺がすべて他方の矢じりの包括辺となることである □

一方の矢じりの 3 本の辺がすべて包括辺となるだけでは，必ずしも A と B が重なるとは限らないことに注意する．例えば，図 3 では矢じり A の 3 本の辺がすべて B の包括辺になっているが，A と B は重なっていない．

a が B の包括辺であるかどうかを判定する具体的な方法について述べる．矢じり配置の性質上，A, B はどちらも辺に沿って平行移動するだけで回転をしないので，三角形の各辺の傾きは変わらない．このことを考慮すると，a が包括辺であるとき，かつその時に限り a の内領域に存在するような B の頂点が存在する．このような頂点を a の**対応点**と呼ぶ（図 4）．よって a が B の包括辺であるかどうかは，対応点が内領域にあるかどうかで判定すればよい．

以上を踏まえて，辺 $e = (u, v)$ の矢じりと辺 g の矢じりの重なり判定を，混合整数計画問題の制約式として有効な一次式の比較のみで行う方法について述べる．辺 e の矢じりを表す正三角形の 3 本の辺を延長した直線の方程式をそ



(a) $k_e = 0$　　(b) 任意の k_e

図 5　直線 e_i により分けられる領域
Fig. 5　Two regions divided by line e_i

れぞれ $e_i(x, y, k_e) = 0$ $(i = 0, 1, 2)$ とする．これらはすべて，x, y, k_e に関する一次式で表せる．このとき，v の中心の x, y 座標をそれぞれ x_v, y_v とすると，$e_i(x_v, y_v, 0) > 0$ となるように，必要ならば $e_i(x, y, k_e) = 0$ の両辺を -1 倍しておく．こうすることにより，任意の k_e に対してそのときの内領域の任意の点の x, y 座標をそれぞれ x_{in}, y_{in} とすると，常に $e_i(x_{in}, y_{in}, k_e) > 0$ が成り立つ（図 5）．次に，辺 g の矢じりの 3 つの頂点の位置ベクトルを $\boldsymbol{G}_j(k_g)$, $(j = 0, 1, 2)$ とする．これらの要素はすべて，k_g に関する一次式で表される．$\boldsymbol{G}_j.x$, $\boldsymbol{G}_j.y$ をそれぞれ $\boldsymbol{G}_j(k_g)$ の x, y 成分とする．そして，$\boldsymbol{G}_j(k_g)$ の頂点を対応点として持つ e_i に $\boldsymbol{G}_j.x$, $\boldsymbol{G}_j.y$ を代入すると，2 変数の一次関数である θ^i_{eg} を用いて，$\theta^i_{eg}(k_e, k_g) := e_i(\boldsymbol{G}_j.x, \boldsymbol{G}_j.y, k_e)$ と表すことができる．すると，$(\boldsymbol{G}_j.x, \boldsymbol{G}_j.y)$ が e_i の内領域に存在するとき，つまり e_i が包括辺になるときかつそのときに限り $\theta^i_{eg}(k_e, k_g) > 0$ となる．辺 e の矢じりの他の辺，辺 g の矢じりの 3 本の辺が包括辺になるかどうかも同様に考えることができる．よって補題 1 より，辺 e の矢じりと辺 g の矢じりが重なるための必要十分条件は，$\theta^0_{eg}, \theta^1_{eg}, \theta^2_{eg}, \theta^0_{ge}, \theta^1_{ge}, \theta^2_{ge}$ の値がすべて正であることである．

以下に目的関数と制約条件を示す．

minimize

$$\sum_{e\in E}\left(W_1\sum_{g\in\Phi(e)}p_{eg}+W_2\sum_{v\in\Psi(e)}q_{ev}+W_3\sum_{g\in E-\{e\}}\frac{y_{eg}}{2}\right)+\frac{1}{M}\sum_{e\in E}L_e k_e \quad (3)$$

subject to

$$\frac{r+R}{L_e}\le k_e\le 1-\frac{r+R}{L_e} \qquad \forall e\in E \quad (4)$$

$$\begin{aligned} k_e-\gamma_{eg} &\le a_{eg}\\ \delta_{eg}-k_e &\le b_{eg}\\ a_{eg}+b_{eg} &\le p_{eg}+1 \end{aligned} \qquad \forall e\in E, \forall g\in\Phi(e) \quad (5)$$

$$\begin{aligned} k_e-\gamma'_{eg} &\le c_{ev}\\ \delta'_{eg}-k_e &\le d_{ev}\\ c_{ev}+d_{ev} &\le q_{ev}+1 \end{aligned} \qquad \forall e\in E, \forall v\in\Psi(e) \quad (6)$$

$$\begin{aligned} \theta^i_{eg}(k_e, k_g) &\le N^i_{eg}f^i_{eg}\\ \theta^i_{ge}(k_g, k_e) &\le N^i_{ge}f^i_{ge} \end{aligned} \qquad \forall e\in E, \forall g\in E-\{e\}, i=\{0,1,2\} \quad (7)$$

$$f^0_{eg}+f^1_{eg}+f^2_{eg}+f^0_{ge}+f^1_{ge}+f^2_{ge}\le 5+y_{eg} \qquad \forall e\in E, \forall g\in E-\{e\} \quad (8)$$

式 (3)〜(8) 中，k_e は矢じりの配置位置を表す連続値の変数で，添字付きの p, q, y, a, b, c, d, f はバイナリ変数である．その他の記号はすべて定数である．L_e は辺 $e = (u, v)$ の長さ，つまり u の中心から v の中心までの直線距離を表す．式 (3) の M は十分大きい数で，実験では $\sum_{e\in E} L_e$ としている．M は，評価値 *distance* の優先度を他の 3 つの評価値に比べて下げるために用いる．バイナリ変数である p, q, y は，それぞれ，

- p_{eg}: 辺 e の矢じりと辺 g が重なっていれば 1
- q_{ev}: 辺 e の矢じりと頂点 v が重なっていれば 1
- y_{eg}: 辺 e の矢じりと辺 g の矢じりが重なっていれば 1

となる．式 (7) の N^i_{eg} は十分大きい数で，実験では，対応する左辺の式 θ^i_{eg} 中の係数，定数項それぞれの絶対値の和とした．式 (4) は，辺 $e = (u, v)$ の矢じりは e 上の u, v とは重ならない位置に配置しなければならないことを表す．また，この式により全ての辺 $e \in E$ に対して $0 < k_e < 1$ となることが保証される．式 (5),(6) はそれぞれ矢じりと辺の重なり，矢じりと頂点の重なりの判定を行い，もし重なっていればそれぞれ対応する p, q の値を 1 にする．式 (7),(8) は矢じり同士の重なり判定を行い，もし重なっていれば対応する y の値を 1 にする．

4.4　高速化の工夫

4.3 節で述べた定式化に従って目的関数と制約式を CPLEX などの数理計画ソルバーに入力すれば，厳密解を求めることができる．しかし，単純にそうするのではなく，以下で述べる前処理を行うことで，より少ない実行時間で厳密解を得ることができる．この前処理では，まず矢じりの配置範囲を考慮する必要のあるものに絞る．そして，

- 配置位置をあらかじめ定めてよい矢じりはその位置に配置する
- 元の問題を独立した小さな部分問題に分割する

などの処理を行った後に，分割した問題ごとにソルバーに入力する．

高速化のための前処理では，まず矢じりが配置される可能性がある範囲を絞りこんでいく．例えば，図 6(a) の辺 e の矢じりは，斜線で示された位置に他の矢じり，辺，頂点と重ならずに配置できるため，そこより後ろに配置されることはない．同様に，辺 g の矢じりの配置位置は終点となる頂点に最も近い位置に確定される．すると今度は，e の矢じりが g の矢じりと重なる可能性が無くなったので，e の

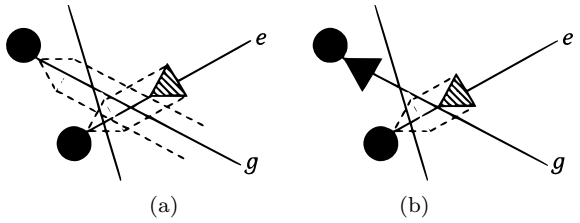


図 6 矢じり配置可能な範囲
Fig. 6 Ranges where arrowheads can be placed

矢じりの配置範囲をさらに狭めることができる（図 6(b)）．このように，各辺 (u,v) の矢じりの配置できる範囲を，u に近いほうから狭めていく．

処理内容を記述するための定義を行う．前述のように，辺 e の矢じりの配置位置を k_e で表す．これ以降，位置 k に辺 e の矢じりを配置する，というのを単に $k_e = k$ と表す．また，すべての辺の k_e の値が確定した描画を，2 章と同じように Γ と表す．まず，以下の 3 つを定義する．

***crE*(e,k):** $k_e = k$ のとき，辺 e の矢じりと重なる辺の集合

***crV*(e,k):** $k_e = k$ のとき，辺 e の矢じりと重なる頂点の集合

***ov*(e,Γ):** Γ において，辺 e の矢じりと重なる矢じりをもつ辺の集合

これ以降，簡単のため $(r+R)/L_e$ を F_e と表記することにし，以下を定義する．

***low*(e):** $F_e \leq k \leq 1-F_e$ のとき，$W_1 \cdot |crE(e,k)| + W_2 \cdot |crV(e,k)|$ の最小値

$low(e)$ は，辺 e の矢じりが関わる重なりによって，目的関数に加算される値の下限を表す．この前処理では，図 6 で示したように，各辺の矢じりを配置できる位置を後方（辺の始点となる頂点に近いほう）から狭めていく．このとき，$K[e](e \in E)$ として，各辺 e の矢じりを配置する最後方の k_e の値を記憶する．最後に，矢じり同士の重なりの数に関して，次の値を定義する．

***maxOV*(e,k):** $k_e = k$，$F_g \leq k_g \leq K[g]$ $(\forall g \in E - \{e\})$ を満たす任意の描画 Γ に対し，$|ov(e,\Gamma)|$ の最大値

位置 k に辺 e の矢じりを配置したとき，辺 e の矢じりが他のどの矢じりとも重なる可能性がないならば，$maxOV(e,k) = 0$ となる．

準備が整ったので，処理の流れについて述べる．まず最初に，各辺 e に対し，対応する K の要素を $K[e] \leftarrow 1 - F_e$ で初期化する．次に，ある辺 e を選び，$K[e]$ の値を更新する処理（***renewK*(e)** と呼ぶ）を実行する．この処理では，$maxOV(e,k) = 0$ かつ $W_1 \cdot |crE(e,k)| + W_2 \cdot |crV(e,k)| = low(e)$ を満たす k が存在すれば，その最小値を k_{min} とし，もしも $k_{min} < K[e]$ ならば $K[e] \leftarrow k_{min}$ とする．これを，どの $e \in E$ を選んでも K が更新されなくなるまで繰り返す．

上の処理では，繰り返しの度に辺をランダム（もしくは順番）に選んで K が更新されるかどうかを試すのではなく，あらかじめ，$K[e]$ の値を更新できる可能性がある辺 e を要素とする集合（Z とする）を保持しておき，その集合に含まれる辺に対してのみ $renewK$ を行うようにする．どの辺に対しても $renewK$ を一回は行う必要があるため，最初は，$Z \leftarrow E$ とする．そして，$renewK(e)$ を一回行うたびに，以下の処理を行う．まず，e を Z から削除する．次に，その $renewK(e)$ を行う前の $K[e]$ の値を b として，$conflict(e, K[e], b) \subset E$（以下で定義）を求め，$Z$ にその要素をすべて追加する．

***conflict*(e,l,r):** $l \leq k_e \leq r$，$F_g \leq k_g \leq K[g]$ $(\forall g \in E - \{e\})$ を満たす任意の描画 Γ に対し，$\bigcup_\Gamma ov(e,\Gamma)$．なお，$l \geq r$ のときは，空集合とする．

直感的に述べると，$conflict(e,l,r)$ は辺 e の矢じりと重なる場合のある矢じりを持つ辺の集合を表す．

そして，Z が空集合になったとき，これ以上 K を更新できる可能性は無いとして終了する．以下に，ここまでで述べた処理によって K を更新していく擬似コードを示す．

```
1: for all e ∈ E do
2:     K[e] ← 1 − F_e
3: end for
4: Z ← E
5: while Z ≠ ∅ do
6:     e ← an arbitrary element of Z
7:     b ← K[e]
8:     renewK(e)
9:     Z ← (Z − {e}) ∪ conflict(e, K[e], b)
10: end while
```

ここまでの処理によって，各辺 e の矢じりが配置される可能性がある範囲を $F_e \leq k_e \leq K[e]$ まで絞ることができた．ここからは，実行時間を削減するために，求めた K をどのように用いるのかを説明する．まず，各辺 e に対して $conflict(e, F_e, K[e])$ を求め，それを $adj(e)$ とする．そして，$adj(e) = \emptyset$ となるすべての辺 e に対し，$F_e \leq k_e \leq K[e]$ のときの，$W_1 \cdot |crE(e,k)| + W_2 \cdot |crV(e,k)|$ が最小になり，かつそのなかで k_e の値が最小になる位置に辺 e の矢じりを配置する．

そして，$adj(e) \neq \emptyset$ である各辺 e に対してこれ以降の処理を行う．まず，矢じりを配置できる範囲を表す制約式 (4) を，

$$F_e \leq k_e \leq K[e] + \rho(e) \tag{9}$$

のように変更する．上式中，$K[e] = 1 - F_e$ ならば $\rho(e) = 0$，そうでなければ $\rho(e)$ は十分小さい正の数とする．このようにするのは，もし $K[e] \neq 1 - F_e$ のときに $\rho(e) = 0$ としてしまうと，位置 $K[e]$ に境界があり，本来位置 $K[e]$ では重なってないとしたい重なり範囲について，位置 $K[e]$ で重なっていると判定されてしまうためである．また，$\Phi(e), \Psi(e)$

についても，以下のように変更する．

$$\Phi(e) := \bigcup_{F_e \le k \le K[e]} crE(e,k)$$
$$\Psi(e) := \bigcup_{F_e \le k \le K[e]} crV(e,k)$$

また，矢じり同士の重なりについても，すべての辺の組み合わせを調べる必要はなく，辺 e に対しては $adj(e)$ に含まれる辺の矢じりとの重なりのみを考えればよい．

次に，$adj(e) \neq \emptyset$ である各辺 e に対応して頂点を作り，e に対応する頂点と，$adj(e)$ の各要素に対応する頂点の組を辺とした無向グラフ $G' = (V', E')$ を考える．G' の連結成分の個数を c として，各連結成分に含まれる頂点ごとに V' を分割したものを $V'_1, V'_2, ..., V'_c$ とする．そして，V' の各部分集合の要素を対応する辺 $(\in E)$ に変換することで，辺 $(\in E)$ の集合 $E_1, E_2, ..., E_c$ が得られる．

このとき，E_i に含まれる辺の矢じりと E_j $(i \neq j)$ に含まれる辺の矢じりが重なる可能性は無いため，E_i と E_j の矢じり配置はそれぞれ独立に考えることができる．よって最後に，各 E_i ごとに目的関数と制約式を立て，それぞれ独立してソルバーに入力し，矢じり配置を求める．そして，求めた部分的な矢じり配置を合わせたものを，最終的な矢じり配置とする．この一連の処理によって，解の最適性が失われることは無い．つまり，前処理で求めた矢じり位置に矢じりが配置されているような最適解が存在し，上で述べたように独立した辺の集合ごとに目的関数と制約式を立てることで必ず最適解を得ることができる．

5. 計算機実験

5.1 実験方法

グラフ G として，頂点の平均次数 2.8 で，頂点数 60, 80, 100, 120, 140 のグラフと，頂点数 100 で，辺数 120, 160, 180, 200 のグラフをそれぞれ 100 個ずつランダムに作成した．各辺の向きもランダムに定めた．各グラフ G に対して，Kamada ら [3] のアルゴリズムにより頂点と辺の配置を定めたものを $\tilde{G}$ とした．また，文献 [4] と同様に，$r = R$ とし，それらの値を $\tilde{G}$ における最短辺長などを用いてグラフごとに定めた．

以上の実験データに対し，従来法，提案法及び前処理を行わない提案法を実行して，評価値と実行時間を求めた．4.4 節で述べた前処理の実行時間削減効果を確認するために，従来法，前処理を行わない提案法では，矢じり同士の重なりについて，すべての辺の組み合わせではなく，矢じり同士が重なる場合がある辺の組み合わせのみをソルバーに入力した．また，提案法において，実際に描画される矢じりの外接円半径 R に対し，計算上はそれを $1.2R$ とした．こうすることで，矢じりの辺が接するように配置されることを防いでいる．提案法では，$(W_1, W_2, W_3) = (1, 3, 2)$ とした．このような値に設定した理由は，何回か実験を行った結果，経験的にこの場合に最も良い描画が得られると判断したためである．

実験に使用した計算機の CPU は Intel Core i5-4460, 3.20GHz，メモリは 16GB，OS は Windows10，プログラミング言語は Java8.0 である．また，従来法における整数計画問題，提案法における混合整数計画問題のソルバーとして，CPLEX[8] を用いた．

5.2 実験結果

実験結果を表 1，図 7，図 8 に示す．表 1 は，2 章で述べた評価値を示している．スペースの都合上，従来法を「従来」，提案法を「提案」と略記している．表 1 を見ると，従来法に比べて提案法では，すべての入力に対しすべての評価値に関してより優れた結果が得られている．特に，提案法ではすべての入力に対し矢じり同士の重なりが発生することは一度もなかった．また表 1 中には明記していないが，評価値 *distance* に関しても，すべての入力に対し従来法に比べて提案法でより優れた結果が得られた．

図 7，図 8 はどちらも実行時間を示している．図 7 は頂点数を 100 で固定した場合，図 8 は平均次数を 2.8 で固定した場合である．図 7，図 8 を見ると，4.4 節で述べた前処理によって実行時間を大きく削減できていることが分かる．図 9 は図 7 を縦軸に関して拡大して表示したものである．提案法の実行時間は，図 9 では，辺数が多くなるにつれて急激に増加している．一方図 8 では，頂点数（辺数）に比例するように増加している．この差は，前処理で連結成分に分けるところで発生すると考えられる．平均次数が大きくなると，複数の辺が矢じり同士の重なりに関して影響しあって，連結成分に分解することができなくなるため，部分問題のサイズが大きくなる．また，図 9 で従来法の実行時間がほとんど変化しないのは，辺数が多くなると一辺あたりの矢じり候補の数が少なくなるからである．

提案法でも，平均次数が大きくなると実行時間は急激に増加してしまうことが分かった．しかし，平均次数が大きすぎるグラフはそもそも描画に適してないことが多い．また，実験ではランダムに生成されたグラフ描画のみを用いたが，平面描画に近い描画，頂点がクラスタリングされた描画など，現実世界でよく起こりうる描画に対しても前処理による実行時間削減は有効であると考えている．

提案法による矢じり配置の例を図 10 に示す．

6. むすび

本研究では，矢じり配置問題に対する厳密解法を提案した．提案法では矢じり配置問題を混合整数計画問題に帰着している．また，実行時間削減のための前処理を行っている．提案法によって従来法に比べてより優れた矢じり配置が得られることと，前処理による実行時間削減が有効であ

表 1　評価値の比較

Table 1　Comparison of evaluation values

頂点数	辺数	*crossV*		*crossE*		*overlap*	
$\|V\|$	$\|E\|$	従来	提案	従来	提案	従来	提案
60	84	0.03	0.00	3.1	2.6	0.26	0.00
80	112	0.10	0.00	5.9	4.8	0.48	0.00
	120	0.16	0.00	3.9	3.4	0.30	0.00
	140	0.15	0.00	8.9	6.8	0.56	0.00
100	160	0.21	0.00	15.5	10.9	1.26	0.00
	180	0.29	0.00	23.1	15.3	1.55	0.00
	200	0.40	0.00	36.1	21.4	2.44	0.00
120	168	0.41	0.01	11.2	8.4	0.83	0.00
140	196	0.40	0.00	14.0	10.1	0.88	0.00

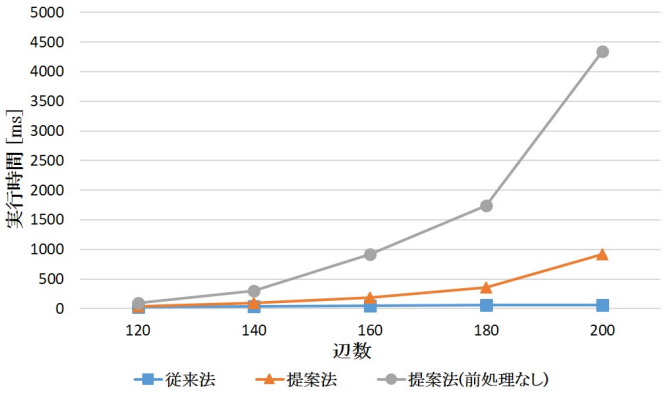


図 7　頂点数 100 のときの実行時間

Fig. 7　Running time in the case of 100 vertices

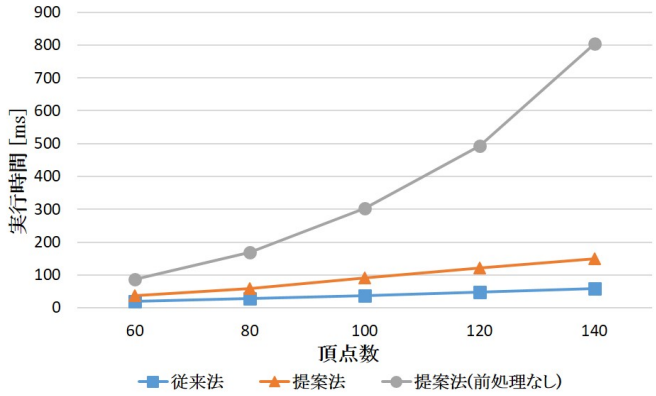


図 8　平均次数 2.8 のときの実行時間

Fig. 8　Running time in the case of average degree 2.8

ることを，計算機実験により示した．

前処理を工夫することにより，さらに実行時間を削減することが今後の課題である．

参考文献

[1] 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ーフェース，計測自動制御学会，1993.

[2] Eades, P.: A heuristics for graph drawing, *Congressus Numerantium*, vol.42, pp.146-160, 1984.

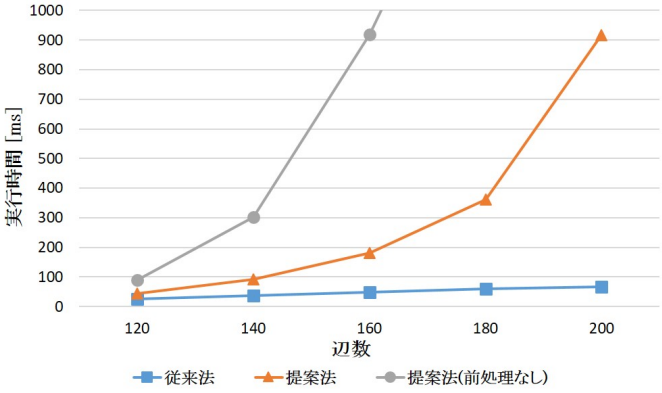


図 9　頂点数 100 のときの実行時間（拡大）

Fig. 9　Running time in the case of 100 vertices (magnified)

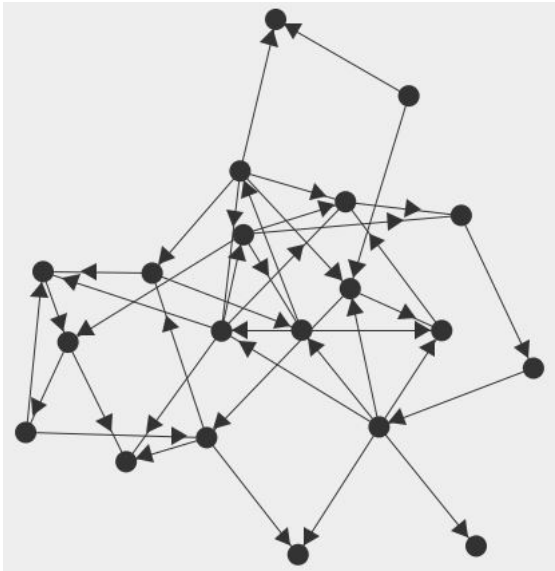


図 10　提案法による矢じり配置の例

Fig. 10　An example of arrowheads placement obtained by our algorithm

[3] Kamada, T. and Kawai, S.: An algorithm for drawing general undirected graphs, *Information Processing Letters*, vol.31, pp.7-15, 1989.

[4] Binucci, C., Chimani, M., Didimo, W., Liotta, G. and Montecchiani, F.: Placing arrows in directed graph drawings, *Proc.24th Int'l Symp. on Graph Drawing and Network Visualization, Lecture Notes in Computer Science*, vol.9801, pp.44-51, Springer, Berlin, 2016.

[5] Wagner, F. and Wolff, A.: A combinatorial framework for map labeling problem, *Proc.6th Int'l Symp. on Graph Drawing, Lecture Notes in Computer Science*, vol.1547, pp.316-331, Springer, Berlin, 1998.

[6] van Kreveld, M., Strijk, T. and Wolff, A.: Point set labeling with sliding labels, *Proc.14th Annual Symp. on Computational Geometry*, pp.337-346, Minneapolis, 1998.

[7] 城戸直人，増田澄男，山口一章：スライダーモデルを用いた有向グラフ描画における矢じり配置手法，平成 29 年電気関係学会関西連合大会講演論文集，G10，2016.

[8] IBM: IBM ILOG CPLEX, available from ⟨https://www-03.ibm.com/software/products/ja/ibmilogcple⟩ (accessed 2019-02-07)