平面配線可能性検証アルゴリズムの実現

磯 直行[†] 平田富夫[††]

配線可能性検証とは，与えられた概略配線が詳細配線へ変換できるかどうかを判定することである．本論文では，まず，先に筆者らが提案した平面配線可能性検証アルゴリズムを計算機上に実現し，プリント配線板設計に適用した場合の性能について報告する．次に，使用する記憶量を削減するための方法を提案し，その性能について報告する．

Implementation of Routability Checking Algorithm for Planar Layouts

NAOYUKI ISO[†] and TOMIO HIRATA[††]

Routability checking is to decide whether the global wires can be transformed into the detailed ones or not. We proposed an efficient algorithm for routability checking. In this paper, we implement our algorithm and apply it to actual layout design. Furthermore, we modify our algorithm so that it use only linear space and give experimental results of its performance.

1. はじめに

配線可能性検証とは，概略配線が詳細配線へ変換できるかどうかを判定することである．平面配線の場合，いくつかの検証アルゴリズムが知られている[1)~8)]．これらは皆，概略配線を与えたとき，クリティカルカット（端子点または障害物の頂点で可視なものどうしを結ぶ線分）についてそれと交差する概略配線数（フロー）が，そこを通過することのできる詳細配線の数（容量）を超えるか否かを調べることで判定を行っている．筆者らは文献8)で部品（配線禁止領域）のない場合のレイアウトにおいて配線可能性検証のための2つのグラフ，“容量判定グラフ”と“フロー導出グラフ”を提案した．容量判定グラフはフローと容量の比較を行わなければならないクリティカルカットを辺とするグラフである．また，フロー導出グラフは容量判定グラフから一部の辺を除いてできるグラフで，このグラフの辺についてフローを知っていれば，容量判定グラフの他の辺についてもフローが計算で求めることができる．概略配線処理によってフロー導出グラフの各辺についてフローを定め，次に，それを用いて容量判定グラフのすべての辺のフローを計算する．最後に容量判定グラフの各辺ごとにフローと容量の比較をして配線可能性を判定する．文献4)では，配線禁止領域が存在するレイアウトモデルについても適用できるようにこれらのグラフを拡張した．この配線可能性検証アルゴリズムの実行時間は容量判定グラフの辺の数に比例する．端子点と配線禁止領域の頂点の数を n とするとき，容量判定グラフとフロー導出グラフの辺の数はそれぞれ平均で $O(n \log n)$，$O(n)$ である[4),8)]．

本論文では，まず，文献4)の配線可能性検証アルゴリズムを計算機上に実現し，実際のプリント配線板設計に適用して，その処理時間が文献5)のアルゴリズムに比べ大幅に短縮されることを確認する．文献4)のアルゴリズムにおいて，クリティカルカットの生成方法は文献6)においてMaleyが提案しているものと本質的には同じである．ただし，文献6)では配線禁止領域のないモデルしか扱っていないのと，生成したクリティカルカットのフローを求める効率の良い方法が示されていないため，本論文では文献5)と計算時間の比較を行っている．

次に，本配線可能性検証アルゴリズムをより実用的なものとするために，実行時に使用するメモリ量（記憶量）を削減する．本アルゴリズムで使用する記憶量は容量判定グラフの辺の数によって決まるので，改良版ではメモリ上に保持しておくデータはフロー導出グラフのみとし，容量判定グラフは保持しない．フロー

† 中京大学情報科学部
School of Computer and Cognitive Sciences, Chukyo University
†† 名古屋大学大学院工学研究科
Graduate School of Engineering, Nagoya University

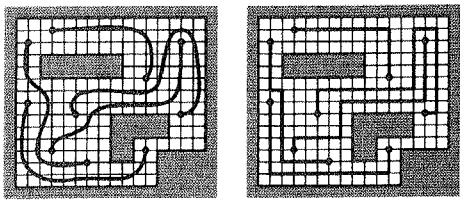
導出グラフに含まれない辺のフローが必要となったときに随時フロー導出グラフから計算して求める．これは以下の理由によるものである．計算機に搭載できるメモリには限界があり，その限られた資源の中でプログラムを実行しなければならない．最近の計算機は，作成されたプログラムが大量のデータを保持し実メモリを使い果たした場合でも，ハードディスクをスワップメモリとして使いアルゴリズムの動作を停止させないように工夫がなされている．しかし，速度の遅いスワップメモリを使いはじめると，計算機の動作は急に遅くなる．特にVLSIやプリント配線板の設計データを処理するプログラムでは，中間的に生成されるデータの量が小さく，すべてを実メモリに保持できるときには高速な動作が期待できる．その意味で，使用メモリ量を $O(n\log n)$ から $O(n)$ に削減する意義は大きい．

本論文の構成は以下のとおりである．まず，2章で本論文におけるレイアウトモデル及び言葉の定義を行う．3章で，文献4)の配線可能性検証アルゴリズムを計算機上に実現した場合，その処理時間が従来法[5]に比べ大幅に短縮されることを示す．4章では，記憶量を削減するための改良を行う．5章で改良された配線可能性検証アルゴリズムを計算機上に実装し，実際のプリント配線板設計に適用する．その結果，十分実用的な速度を維持したままメモリ量を削減できることを示す．6章でまとめを行う．

2. レイアウトモデル

本論文で扱う配線モデルについて述べる．配線領域は単層であり縦横格子が引かれている．配線領域の境界線は水平および垂直辺からなる多角形であり，領域内には端子点と，多角形モジュール（以下では単にモジュールという）が互いに重ならないように配置されている．多角形モジュールは水平および垂直辺を外周線とする多角形である．端子点，配線領域の境界線およびモジュールの外周線は格子点上に位置している．配線経路は端子点を始点および終点とし，互いに交わらず，かつ他の端子点上やモジュール内を通過しない．配線経路がトポロジで表現されたものが概略配線であり（図1(a)），配線経路をすべて格子上に乗せたものが詳細配線である（図1(b)）．

端子点の集合を $\mathbf{V}_t$ とする．モジュールの内部と配線領域の外部を配線禁止領域とよぶ．モジュールおよび配線領域の境界線の各辺の端点の集合を $\mathbf{V}_m$ とする．$\mathbf{V}_t$ と $\mathbf{V}_m$ の和集合を $\mathbf{V}_{t+m}$ とする．以下では，点は $\mathbf{V}_{t+m}$ 内の点を表すものとする．



(a) Global routing (b) Detailed routing

図1 配線モデル
Fig. 1 Routing model (global and detailed routings).

格子の縦と横をそれぞれ1単位長とし，2点 $a=(x_a,y_a)$，$b=(x_b,y_b)$ 間の距離を $||(a,b)||=max(|x_a-x_b|,|y_a-y_b|)$ とする．配線領域内の線分 (a,b) がモジュールまたは端子点と交差しないとき，a と b は互いに可視であるという．互いに可視な2点 $p,q(\in \mathbf{V}_{t+m})$ 間を結ぶ線分をカットとよぶ．カット (p,q) を通過することのできる配線の最大数を容量とよび，$cap(p,q)=||(p,q)||-1$ と定義する．概略配線が与えられたとき，カット (p,q) を通過している概略配線経路の本数をフローとよぶ．カット (p,q) に関して，容量がフローより大きいか等しいという条件をカット (p,q) に対する容量制約という．

a と b を配線領域内の2点とする．$a,b\in\mathbf{V}_{t+m}$ のとき，a または b を通る傾きが45°および−45°の4本の直線によって作られる長方形を $rect(a,b)$ と表記する．

a を通る傾きが45°および−45°の直線を考える．2つの直線に対してともに上にある領域を a の上の領域とよぶ．同様に a の右の領域，下の領域，左の領域を定める．また，座標軸を−45°回転したときの a の座標を $(\mu_a,\nu_a)=((x_a-y_a)/\sqrt{2},(x_a+y_a)/\sqrt{2})$ とする．点 $a=(\mu_a,\nu_a)$，$b=(\mu_b,\nu_b)$ について，$\mu_a>\mu_b$ ならば，−45°軸に関して a は b より大きいという．また，$\nu_a>\nu_b$ ならば，45°軸に関して a は b より大きいという．

次に，容量判定グラフ $\mathbf{G}_c$ およびフロー導出グラフ $\mathbf{G}_f$ を定義する．

定義1[4),8)] 容量判定グラフ $\mathbf{G}_c=(\mathbf{V}_c,\mathbf{E}_c)$ を次のように定義する．節点集合 $\mathbf{V}_c$ は $\mathbf{V}_{t+m}$ である．辺集合 $\mathbf{E}_c$ を次のように定める．$a,b\in\mathbf{V}_{t+m}$ について，a と b が互いに可視であり，かつ，$rect(a,b)$ 内に a と b から可視な他の点 $c(\in\mathbf{V}_{t+m})$ が存在しないならば $(a,b)\in\mathbf{E}_c$ である．

文献8)では，$\mathbf{E}_c$ の辺についてのみ容量制約を満足するかどうかを調べれば配線可能性検証が行え，また $\mathbf{E}_c$ の辺の数が平均 $O(n\log n)$ であることが示されている．以下では，$\mathbf{E}_c$ の辺をクリティカルカット

表1 プリント配線板データ
Table 1 Specifications of PCBs.

レイアウト	部品数	ネット数
A	68	828
B	124	1891
C	134	2744

という．

定義2[4),8)] フロー導出グラフ $\mathbf{G}_f = (\mathbf{V}_c, \mathbf{E}_f)$ は容量判定グラフ $\mathbf{G}_c = (\mathbf{V}_c, \mathbf{E}_c)$ の部分グラフであり，辺集合 $\mathbf{E}_f$ を次のように定める．点 a と点 b については $\nu_a \leq \nu_b$ であるとする．$a, b \in \mathbf{V}_c$ に対し，a の上かつ b の左の領域に a または b から可視な点が存在しない，または，a の右かつ b の下の領域に a または b から可視な点が存在しないならば，(a, b) は $\mathbf{G}_f$ に含まれる．

$\mathbf{G}_f$ は平面グラフであり，その辺の数 $|\mathbf{E}_f|$ は $O(n)$ である[8)]．$\mathbf{G}_f$ は，そのすべての辺についてフローを与えさえすれば $\mathbf{G}_c$ のすべての辺についてフローを求めることができるように定義されたグラフである．文献8)ではスイープ法と後述する“フリップ操作”により，$\mathbf{G}_f$ の辺のフローから $\mathbf{G}_c$ の辺のフローを計算している．文献4)のアルゴリズムでは，概略配線は $\mathbf{G}_f$ を用いて表現している．すなわち $\mathbf{G}_f$ の辺との交差数により表現している．

3. 実データへの適用

文献4)の配線可能性検証アルゴリズムではスイープ法により $\mathbf{G}_c$ と $\mathbf{G}_f$ を求めている．以下では，このアルゴリズムを，ワークステーション（SunMicrosystems 社製 SPARCstation 10 Model 41）上に C 言語（SunSoft 社製 SPARCompiler C 3.01）を用いて実現し，実際のプリント配線板設計データに適用した結果を示す．

適用したデータは，PCB/MCM レイアウトシステム Allegro で設計したパーソナルコンピュータ用プリント配線板（PCB）の部品配置データである．実装部品を置く表面層（第1層）のレイアウトについて，空中配置等により交差のある部品を取り除き，本配線可能性検証システムへの入力とした．

表1にあるように，3種類のプリント配線板のレイアウトについて本配線可能性検証システムを適用した．**表2**はこれらのレイアウトに対する容量判定グラフの頂点の数 $|\mathbf{V}_c|$，容量判定グラフの辺の数 $|\mathbf{E}_c|$，そしてフロー導出グラフの辺の数 $|\mathbf{E}_f|$ を示している．T_c は本アルゴリズムを用いて容量判定グラフを作成するために要したユーザ時間である．

表2 実際のプリント配線板への適用結果
Table 2 Results for actual PCBs.

レイアウト	$\lvert\mathbf{V}_c\rvert$	$\lvert\mathbf{E}_c\rvert$	$\lvert\mathbf{E}_f\rvert$	T_c[s]	T_1[s]	$\lvert\mathbf{E}_v\rvert$	T_v[s]
A	260	1835	732	0.47	0.96	5080	631
B	484	4452	1368	0.43	1.08	7941	852
C	524	4753	1488	0.50	1.16	8075	818

（計算機：SPARCstation 10）

従来法との比較のために，Leiserson ら[5)]によって提案されたアルゴリズムを用いた場合に容量制約を満足するかどうか調べなければならない辺の数を $|\mathbf{E}_v|$ として示す☆．実際，$\mathbf{E}_v$ は $\mathbf{V}_c$ を点集合とする可視グラフ（互いに可視の2点を辺で結んだグラフ）の辺集合である．T_v は $\mathbf{E}_v$ を生成するために費やしたユーザ時間である．時間 T_c と T_v では決定的な違いがあり，本アルゴリズムの圧倒的な優位性を示しているが，これは文献5)では $\mathbf{G}_c$ の各頂点でその周囲をスイープしているので $O(n^2 \log n)$ 時間を要するのに対し，本配線可能性検証アルゴリズムでは配線領域を1回右上から左下へスイープするだけで $O(n \log n)$ 時間で済むからである[4)]．

本アルゴリズムおよび文献5)のアルゴリズムはそれぞれ $\mathbf{E}_c$ および $\mathbf{E}_v$ の各辺についてその容量制約を満足するかどうかを調べており，その実行に必要な時間はこれらのサイズ $|\mathbf{E}_c|$ と $|\mathbf{E}_v|$ に比例する．表2から，本配線可能性検証アルゴリズムは文献5)のアルゴリズムに比べ辺の数が36%から58%と少なく，高速な処理が可能であることが分かる．本アルゴリズムの実行に実際に要したユーザ時間を表2の T_1 で示す．

本アルゴリズムでは $\mathbf{G}_c$ と $\mathbf{G}_f$ を隣接リストで保持しているため，必要な記憶量は容量判定グラフの辺の数 $\mathbf{E}_c$ に依存し，平均で $O(n \log n)$ の記憶量を必要とする．一方，文献5)のアルゴリズムは $\mathbf{E}_v$ の辺を最初にすべて生成しておく必要はなく，1本ずつ生成しながら配線可能性検証を行うことができる．そのため，記憶量は $|\mathbf{V}_c|$ に比例する量（$O(n)$）で済む．

4. 記憶量の削減

本章では，文献4)のアルゴリズムをより実用的なものとするために，使用するメモリ量（記憶量）を削減する方法について述べる．提案する方法は，メモリ上に保持しておくデータは $\mathbf{G}_f$ のみとし，それに含まれない $\mathbf{G}_c$ の辺のフローが必要となったときに随時 $\mathbf{G}_f$ の辺のフローから計算して求める．つまり図2の

☆ このほかに Cole ら[1)]と Maley[7)]によって提案されている $O(n \log n)$ 時間のアルゴリズムがあるが，複雑で実現が困難なため，ここでは比較しない．

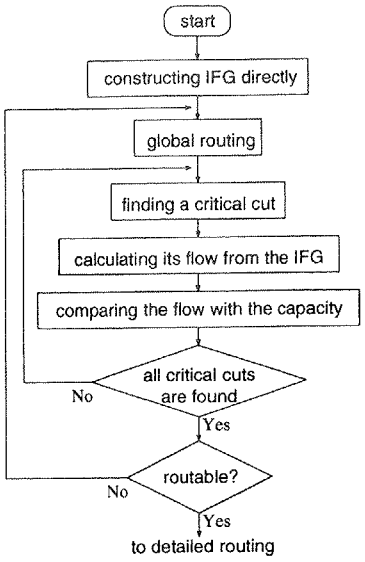


図2 提案する処理手順
Fig. 2 Flowchart for the global routing process.

ような処理手順を考える．この図でIFGはフロー導出グラフのことである．

図2の方法を実現するためには，次に述べる処理が必要である．1つは，$\mathbf{G}_c$ を作らずに $\mathbf{G}_f$ を作成することである．文献4)では一度 $\mathbf{G}_c$ を作りそこから不要な辺を取り除いて $\mathbf{G}_f$ を作成しているが，本方式では使用メモリ量を減らすために $\mathbf{G}_c$ を作らない．そのため $\mathbf{G}_f$ を直接求める必要がある．2つめは容量制約の判定のためにクリティカルカットを列挙することである．そしてもう1つは，クリティカルカットの2端点間を結ぶ線分のフローを $\mathbf{G}_f$ から計算することである．文献4)のアルゴリズムは配線領域をスイープしながら，$\mathbf{G}_f$ の辺とすでに求められた $\mathbf{G}_c$ の辺のフローからクリティカルカットのフローを計算していた．本方式では，$\mathbf{G}_c$ を保持しないため，$\mathbf{G}_f$ のみからクリティカルカットのフローを計算しなければならない．以下，これら3つの処理について述べる．

4.1 フロー導出グラフを求めるアルゴリズム

容量判定グラフを作らずにフロー導出グラフを直接求めるアルゴリズムは，容量判定グラフを作るときのスイープアルゴリズム[4),8)]を基本としている．フロー導出グラフ $\mathbf{G}_f$ は容量判定グラフ $\mathbf{G}_c$ の部分グラフなので，文献4)の $\mathbf{G}_c$ の辺 $\mathbf{E}_c$ を列挙するアルゴリズムで，発見された辺のうち $\mathbf{E}_f$ に該当するものだけを列挙すればよい．フロー導出グラフを直接求めるアルゴリズムの実行時間は $O(n\log n)$ である[4),8)]．

4.2 クリティカルカットの列挙

次に，クリティカルカットを列挙する方法について述べる．本方式では，1回目のスイープでフロー導出グラフ $\mathbf{G}_f$ を求め，概略配線からこの $\mathbf{G}_f$ の各辺に対しフロー（概略配線が何本通過するか）を決定する．次に，2回目のスイープで各イベント点（$\mathbf{V}_c$ の点）を端点とするクリティカルカットを1本ずつ発見し，その容量とフローを比較する．このクリティカルカットの発見には，文献4)の容量判定グラフ $\mathbf{G}_c$ の辺 $\mathbf{E}_c$ を列挙するアルゴリズムを再び利用する．文献4)の列挙アルゴリズムではスイープをしながら，停止した点 a（イベント点）を根とするヒープ探索木を構成し，その木の節点をたどることで a を端点とするクリティカルカットを見つけている．クリティカルカットの発見に必要な時間は $O(n\log n+k)$ で，k はクリティカルカットの数で平均 $O(n\log n)$ である．よって本方式による配線可能性検証に必要な時間はフロー導出グラフを直接求める時間（$O(n\log n)$）とクリティカルカットを列挙する時間（平均 $O(n\log n)$）とすべてのクリティカルカットのフローを計算する時間の和になる．

発見されたクリティカルカットのフロー値は次節の方法で計算するが，本方式ではスイープラインが次のイベント点へ移動するまでの間は，求めたフロー値は記憶しておくこととする．これらは a を端点とする他のクリティカルカットのフローを計算するために利用でき，次節で述べるフリップ操作の回数を削減できる．

4.3 クリティカルカットのフロー計算

本節では，クリティカルカットのフローを計算するアルゴリズムについて述べる．

フロー導出グラフ $\mathbf{G}_f$ は配線領域を三角形領域に分割している．概略配線が与えられた時点で，$\mathbf{G}_f$ の各辺についてそこを通るフローが分かっている．クリティカルカットの両端点を a, b とすると，線分 (a,b) は $\mathbf{G}_f$ の辺と一致するか，または交差する．$\mathbf{G}_f$ の辺と一致する場合はそのフローが分かっているので，その辺に関する容量判定を行えばよい．一方，$\mathbf{G}_f$ の辺と交差する場合には，交差する $\mathbf{G}_f$ の辺を対角辺とする凸四辺形に次々にフリップ操作[4)]を適用しクリティカルカットのフローを計算する．フリップ操作☆は，凸四角形について4つの外周辺と1つの対角線のフローが与えられたとき，それらをもとにもう1つの対角辺についてのフローを求める操作である．図3の例では，□$ABDC$ についてフリップを行い $flow(A,D)$ を求め，次に

☆ 計算幾何学においてフリップは，凸四角形内の対角辺を交換し三角形分割を変更する操作をいうものであるが，本論文中のフリップは単に未知の対角辺のフローを求める操作を指し，もとの対角辺のフローの情報を破棄してしまうものではない．

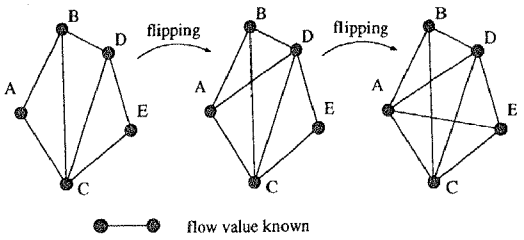


図3 フリップ操作
Fig. 3 Flippings.

$\square ADEC$ についてフリップを行い $flow(A,E)$ を求めている．1回のフリップにかかる計算時間は $O(1)$ である．

4.3.1 フリップ操作の適用順序

図4は，a に近い側の交差辺から順にフリップ操作を適用し (a,b) のフローを計算している例を示している（図ではフリップされた辺は消去している）．しかし，このような適用順序ではフリップ操作を適用できなくなる場合がある．たとえば，**図5**で，(a,b) との交差辺を a に近いものから順に消去するという場合を見てみよう．最初の2つの交差辺に関してはフリップ操作が可能であるが，3つ目の交差では交差辺 e_3 を対角線とする四辺形が凸四辺形ではないためフリップ操作を実行できない．また，フリップをしても新しい対角辺がまたクリティカルカットと交差してしまい交差辺の数が減らない場合もある（図5の e_4）．したがって，線分 (a,b) のフローを求めるために (a,b) と交わる対角辺を a または b に近い順にフリップするという方法ではうまくいかない．

以下では，クリティカルカットとの交差辺をどのような順序でフリップすればよいかを考える．クリティカルカット (a,b) と交差する辺の集合を $E_{ab}=\{e_1,e_2,\cdots,e_m\}$ とし，a から近い順に $e_1,e_2,\cdots,e_m$ とする．a，b を通る直線により分けられる2つの半平面を R，L とする．E_{ab} 内の辺の端点のうち L 内にあるものを E_{ab} の各辺のインデックス順に並べ $s_1,s_2,\cdots,s_{\alpha-1}$ とする．R 内にあるものも同様に $t_1,t_2,\cdots,t_{\beta-1}$ とする．$s_0=t_0=a$，$s_\alpha=t_\beta=b$ とする（**図6**(a)）．

フリップすべき四辺形を選択する方法の1つは，E_{ab} のインデックス順にその辺を対角辺とする凸四辺形を調べ，それがフリップ操作可能な場合にフリップを行うことである．図5のようにフリップ操作ができなくなる場合や直前にフリップした対角辺に再びフリップを適用するような場合にはその辺は飛ばして次のインデックスの辺に進む．このフリップの結果，フリップせずに飛ばしてきた辺に対する四辺形がフリップ可能になったときはその辺のフリップ操作を行う．このように端点 a または b に近い順にフリップ操作を実行してクリティカルカットのフローを計算する方法をNEARESTFIRSTFLIPPINGとよぶことにする．しかし，この方法でクリティカルカットのフロー計算をするには，交差辺の数の2乗に比例した時間を必要とする場合がある[9]．

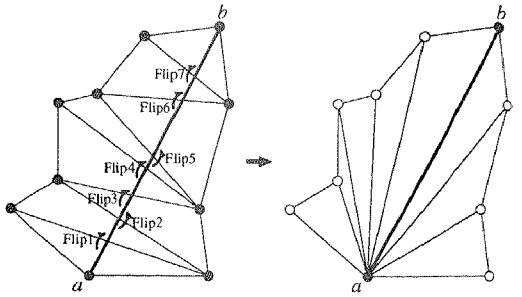


図4 フリップ操作によるフロー計算
Fig. 4 Flow calculation using flippings.

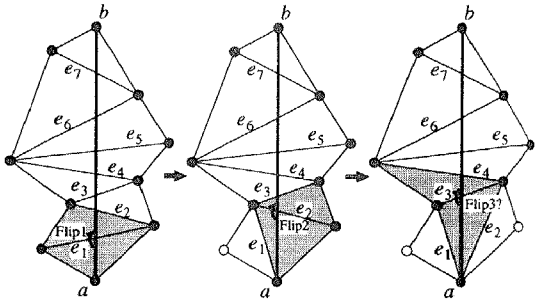


図5 フリップ操作を連続してできない例
Fig. 5 An example of flow calculation problem.

4.3.2 フロー計算アルゴリズムの提案

ここでは，最悪の場合にはNEARESTFIRSTFLIPPINGと同程度の手間を必要とするが，実用的には高速にクリティカルカットのフローを計算するアルゴリズムを与える．図5の例では，NEARESTFIRSTFLIPPINGよりもフリップ操作が1回少なくて済む．

クリティカルカットと交差する辺を保持するためのスタック $\mathcal{S}$ を用意する．アルゴリズムはまず前処理として次の手続き1を実行する．

手続き1

スタック $\mathcal{S}$ を空にする;
for $i:=1$ **to** m **do begin**
　スタック $\mathcal{S}$ に e_i を push する;
　repeat

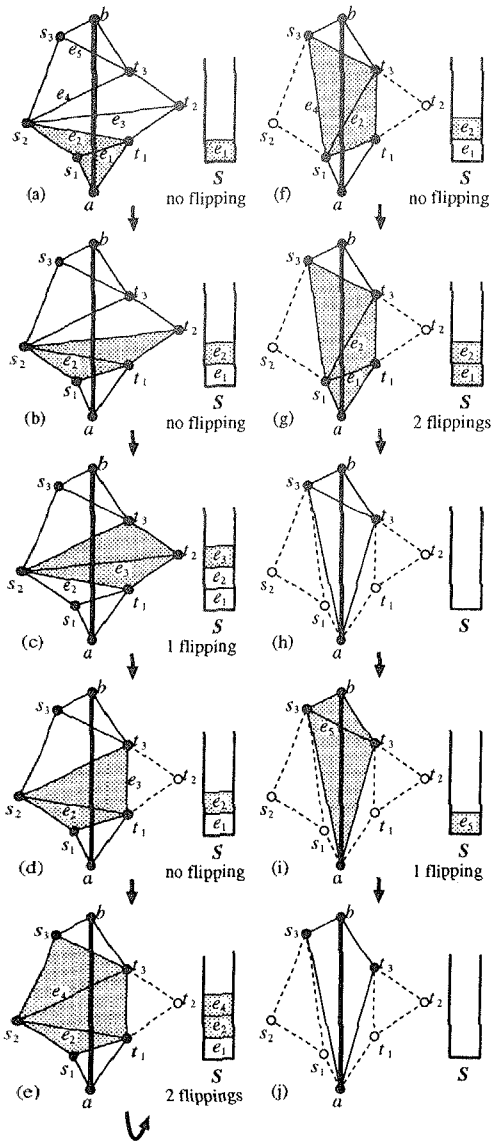


図6 フリップ操作の適用順序
Fig. 6 Flow calculataion using stack.

if (1. $\mathcal{S}$ の先頭の辺 e を対角辺とする四角形 $\mathcal{T}$ が凸で，かつ，$\mathcal{T}$ の点が R 内に3点または L 内に3点が存在する) **then begin**
その3点を u_i, u_{i+1}, u_{i+2} とする．$\mathcal{T}$ にフリップを適用し，新たな辺 (u_i, u_{i+2}) のフローを計算する（図6(c)）．スタックから e を pop する．
end
else if (2. $\mathcal{S}$ の先頭の辺 e' と2番目の辺 e'' を対角辺とする五角形 $\mathcal{P}$ が凸である) **then begin**

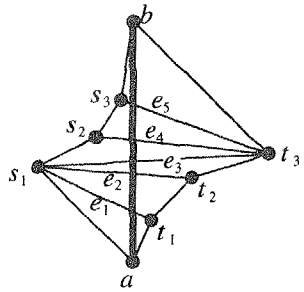


図7 特殊な点配置におけるフロー計算
Fig. 7 Rare case of flow Calculation.

このとき，L または R に $\mathcal{P}$ の輪郭線に沿って連続した3点 u_i, u_{i+1}, u_{i+2} が存在し，u_{i+1} は e' と e'' の共通の端点である（図6(e)）．e'' と e' についてこの順にフリップを適用し，辺 (u_i, u_{i+2}) のフローを計算する（図6(f)）．スタックから e' を pop する．
end;
until (1, 2のいずれにも該当しない)
end;

手続き1が終了すると，L, R それぞれに凸頂点を1つだけ残した図形を得る（図6(j)）．この凸四辺形の対角辺はクリティカルカット (a,b) であり，この辺に最後のフリップ操作を適用してそのフローを得ることができる．このアルゴリズムは1回または2回のフリップ操作で交差辺を1本消去するので，**図7** に示すような特殊な点配置の場合を除いて，クリティカルカットのフローは $O(k)$ 時間で計算できる．ここで，k は線分 (a,b) と交差する $\mathbf{G}_f$ の辺の数である．図7の場合には，手続き1の終了後に得られた図形が四辺形ではなく多角形となり，スタック $\mathcal{S}$ にクリティカルカットと交差する辺が残ってしまう．このときは，その得られた結果に対して再び NEARESTFIRSTFLIPPING を適用する．このようにすると，クリティカルカットと交差する辺を1つ削除するために $O(k)$ の時間を必要とし，クリティカルカットのフロー計算に $O(k^2)$ 時間を必要とする．まず，図7では e_3 だけがフリップ操作可能な辺である．辺 (a,b) のフローを計算するために，まず手続き1を行うが，フリップ操作を1回も適用できずスタック $\mathcal{S}$ にすべての交差辺を残し終了してしまう．次に NEARESTFIRSTFLIPPING を行う．フリップ操作を適用できる辺は e_3 のみであり，このフリップ操作の結果 e_2, e_1 の順でフリップ操作ができ，e_1 を交差辺から削除できる．以後，同様の処理を繰り返すことで辺 (a,b) のフローを計算できるが，$O(k^2)$

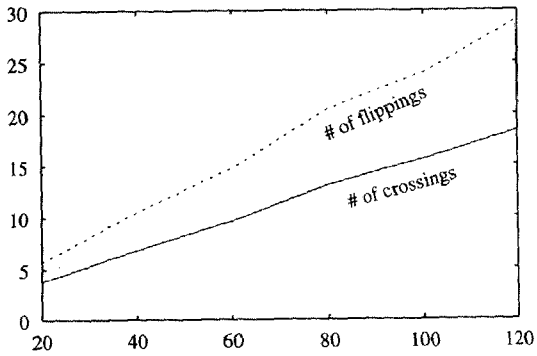


図8 クリティカルカット1本あたりのフリップ操作適用回数と交差辺数
Fig. 8 Flippings and crossings for a critical cut.

時間の手間が必要となる．

本アルゴリズムの性能を実験的に確かめるために，配線領域にモジュールをランダムに配置した場合についてクリティカルカットのフローを計算する実験を行った．**図8**は，1本のクリティカルカットと$\mathbf{G}_f$の辺との交差数（平均）と，1本のクリティカルカットのフローを計算するためのフリップ操作の適用回数（平均）を示す．横軸はモジュールの数である．この結果から，$\mathbf{G}_f$の辺との交差数（k）および適用したフリップ操作の数は，ともにモジュールの数に比例している．よって，平均的には$O(k)$回のフリップ操作を適用することで，任意のクリティカルカットのフローを計算できることが分かる．

5. 新方式の実データへの適用

本章では，前節の配線可能性検証アルゴリズムを計算機上に実装し，実際のプリント配線板設計データに適用した結果を示す．適用したデータは，3章で使用したデータと同じパーソナルコンピュータ用プリント配線板（PCB）の3種類の部品配置データである．

表3に，文献4)のアルゴリズムと本方式のメモリ量をそれぞれM_1，M_2で示す．M_vは従来法[5]のアルゴリズムを実行するために必要としたメモリ量である．この結果から，本章のアルゴリズムの実行時に使用する記憶量は文献4)に比べ26%から32%のメモリしか使用していないことが分かる（これは，表2における$(2\times|\mathbf{V}_c|+|\mathbf{E}_c|+|\mathbf{E}_f|)$と$(|\mathbf{V}_c|+|\mathbf{E}_f|)$の比に対応する）．

次に，辺のフローを計算するために必要とする時間について考える．文献4)のアルゴリズムでは，1回のフリップ操作で容量判定グラフの1本の辺のフローを求めることができるので，フリップ操作の回数は$|\mathbf{E}_c|$

表3 使用メモリ量と実行時間
Table 3 Space and running time.

レイアウト	M_1 [byte]	M_2	M_v	T_1 [s] $O(n\log n)$	T_2 $O(n^2)$	T_v $O(n^2\log n)$
A	161 k	52 k	8 k	0.96	15.72	631
B	380 k	98 k	15 k	1.08	82.66	852
C	408 k	106 k	17 k	1.16	94.47	818

（計算機：SPARCstation 10）

である．一方，提案した方式ではフロー導出グラフの辺に含まれないクリティカルカットのフローは随時計算して求めているので，フリップ操作の回数（したがって計算時間も）は増加する．表3のT_1は文献4)のアルゴリズム，T_2は本方式を実行したときのユーザ時間である．比較のため，従来法[5]の計算時間（T_v）も示し，それらのオーダを記した．T_1に比べるとT_2は増加しているが，それでもT_vと比較して十分小さいことが分かる．例としてレイアウトCの計算時間について考察する．$n=|\mathbf{V}_c|=524$とすると$\log_2 n\approx 9$である．$n^2/(n\log n)\approx 58$であり，T_1とT_2の比約81と比べるとそれほど大きく離れていない．T_vとT_2の比約8.6についても，$(n^2\log n)/n^2=\log n\approx 9$であり，理論的に求まる値と一致している．

このように，前章で提案した配線可能性検証アルゴリズムは，実用上十分な速度を持ちながら，使用するメモリ量が少ないことが分かる．

6. おわりに

水平垂直方向の配線のみを許す配線モデルにおいて，与えられた概略配線を実際に平面上に配線できるかどうかを判定する配線可能性検証アルゴリズムを計算機上に実装した．本アルゴリズムを実際のプリント配線板設計データに適用した結果，従来手法と比べて高速に配線可能性検証を実行できた．また，アルゴリズムに工夫を加えることにより，実行時に必要とする記憶量を部品数と端子点数の和に比例する程度とすることができた．

以下では，本研究の応用と今後の課題について述べる．本論文で述べたレイアウトモデルは平面配線を仮定している．多層配線の場合には，ネットを各層へ割り当て，層ごとに概略配線を行う．他層のネットとの接続はスルーホールを利用するが，それらは各層においては配線領域内の端子点として扱うことができる．したがって，層別割当てが済んだ後は本アルゴリズムにより配線可能性を検証することができる．

また，本論文では端子点は1本の配線のみ接続すると仮定しているが，次のようにして多端子点ネットに

図9 多端子点ネットへの適用

Fig. 9 Transformation of a multi-terminal net into two-terminal nets.

対応することができる．2本以上の配線が接続できる端子点として分岐点を考える．この分岐点の位置はスタイナ点を求めるアルゴリズムにより与えられるものとする．分岐点をいくつかの近接した端子点に置き換えることにより，多端子点ネットを2端子点ネットへ分解する．詳細配線が終了した後にこれらの近接した端子点を接続する．このとき，分岐点に対応した複数の端子点の位置はいくつか考えられるが，概略配線時はそれらのうち1つを定め配線可能性検証を行い，それが配線不可能と判定された場合には他の端子点配置を試みる（図9）．

謝辞 本研究に関し，実験用プリント配線板設計データを提供いただきました株式会社日立製作所オフィスシステム事業部岩本鉦二副技師長，山田則男元DA応用設計部長，堅田敏幸主任技師，森山隆志氏に感謝いたします．貴重なコメントをいただいた2人の査読者に感謝いたします．本研究の一部は財団法人堀情報科学振興財団および文部省科学研究費の援助を受けた．記して感謝します．

参 考 文 献

1) Cole, R. and Siegel, A.: River routing every which way, but loose, *Proc. 25th IEEE Symp. on Foundations of Computer Science*, pp.65–73 (1984).
2) Dai, W.W., Kong, R. and Sato, M.: Routability of a Rubber-Band Sketch, *Proc. 28th ACM/IEEE Design Automation Conference*, pp.45–48 (1991).
3) Hama, T. and Etoh, H.: Topological Routing Path Search Algorithm with Incremental Routability Test, *Proc. Asia and South Pacific Design Automation Conference*, pp.645–648 (1997).
4) Iso, N., Kawaguchi, Y. and Hirata, T.: Efficient Routability Checking for Global Wires in Planar Layouts, *IEICE Trans. Fundamentals*, Vol.E80-A, No.10, pp.1878–1882 (1997).
5) Leiserson, C. and Maley, F.: Algorithms for routing and testing routability of planar VLSI layouts, *Proc. 17th ACM Symp. on Theory of Computing*, pp.69–78 (1985).
6) Maley, F.: *Single-Layer Wire Routing and Compaction*, pp.45–53, MIT Press (1989).
7) Maley, F.: Testing Homotopic Routability Under Polygonal Wiring Rules, *Algorithmica*, Vol.15, No.1, pp.1–16 (1996).
8) 川口 泰，磯 直行，平田富夫：配線可能性検証のための容量判定グラフとフロー導出グラフ，信学論，A, Vol.J80-A, No.1, pp.135–142 (1997).
9) 譚 学厚：Private Communication.

(平成10年9月17日受付)
(平成11年2月 8 日採録)

磯 直行（正会員）

平成2年山梨大学工学部電子工学科卒業．平成4年同大学大学院修士課程修了．平成7年名古屋大学大学院博士課程後期課程修了．同大学大学院電子工学専攻助手を経て，現在中京大学情報科学部講師．VLSI，プリント配線板のレイアウト設計に関する研究に従事．電子情報通信学会会員．博士（工学）．

平田 富夫（正会員）

昭和51年東北大学工学部通信工学科卒業．昭和56年同大学大学院博士課程修了．昭和56年豊橋技術科学大学助手，昭和61年名古屋大学情報工学科講師を経て，現在，同大学大学院電子工学専攻教授．グラフアルゴリズムとデータ構造の研究に従事．電子情報通信学会，ACM，IEEE各会員．工学博士．