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

名古屋大学 大学院 工学研究科

概要

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，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にアルゴリズムを変更し，その性能について報告する．

Abstract

Routability checking is to decide whether the global wires can be transformed into the detailed ones or not. We proposed two graphs, the capacity checking graph and the initial flow graph, for efficient routability checking. In this paper, we implement our routability checking algorithm and apply it to actual layout design. And we modify it so that it use only linear space.

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 ~ 5] ．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同士を結ぶ線分）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）を越えるか否かを調べることで判定を行っている．筆者らは [5] 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二つのグラフ，“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．[3] 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 [3, 5] ．

本論文では，まず，これらのグラフを用いた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，その処理時間が従来法に比べ大幅に短縮されることを確認する．

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

An Implementation of Routability Checking Algorithm for Planar Layouts
Naoyuki ISO and Tomio HIRATA
Graduate School of Engineering, Nagoya University

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通りである．まず，2で本論文におけるレイアウトモデルの定義を行う．3で，[3]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に比べ大幅に短縮されることを示す．4では，記憶量を削減するための改良を行う．5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でまとめを行う．

2 容量判定グラフとフロー導出グラフ

本論文で扱う配線モデルの詳細は[3]を参照されたい．

容量判定グラフ$\mathbf{G}_c = (\mathbf{V}_c, \mathbf{E}_c)$の節点集合$\mathbf{V}_c$は端子点及び部品（水平，垂直の辺を持つ多角形）の頂点であり，辺集合$\mathbf{E}_c$はクリティカルカットの集合で，フローと容量のチェックを行う必要のない冗長なものを除いたものである．

[3]における容量判定グラフの定義にはこれ以外の頂点と辺も含まれているが，それらは容易に求められるため本論文ではその説明を省略する．[5]では，$\mathbf{E}_c$の辺についてのみ容量制約（フローの値が容量を越えないこと）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．

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$\mathbf{G}_c = (\mathbf{V}_c, \mathbf{E}_c)$の部分グラフである．辺集合$\mathbf{E}_f$は，$\mathbf{E}_f$の辺のフローが分かっていれば，$\mathbf{E}_c$の他の辺のフローが“フリップ操作”で求められるように定められる[5]．詳細については[3, 5]を参照されたい．

$\mathbf{G}_f$は平面グラフであり，その辺の数$|\mathbf{E}_f|$は$O(n)$である[5]．

表 1: プリント配線板データ

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

表 2: 実際のプリント配線板への適用結果

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）

3 実データへの適用

[3]の配線可能性検証アルゴリズムではスキャン法により$\mathbf{G}_c$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．

従来法との比較のために，Leiserson[4]ら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[1]．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

[1] この他にColeらによって提案されている$O(n \log n)$時間のアルゴリズム[1]があるが，極めて複雑で実現が困難なため，ここでは比較しない．

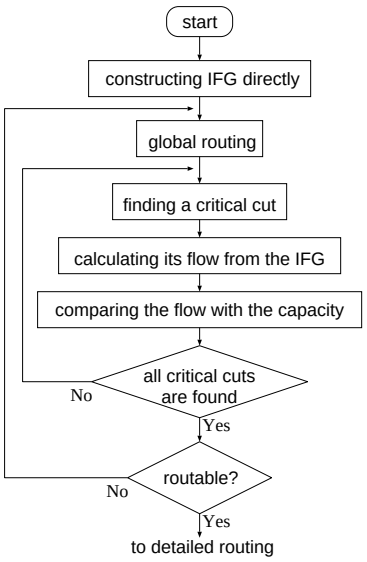

図 1: 提案する処理手順

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 [4] 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 [3]．

本アルゴリズムおよび [4] 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 [4] 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わかる．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，[4] 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，一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

本節では，[3] 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 1のような処理手順を考える．

図 1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．一つは，$\mathbf{G}_c$ を作らずに $\mathbf{G}_f$ を作成することである．[3] では $\mathbf{G}_c$ 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 $\mathbf{G}_c$ を保持しないため $\mathbf{G}_f$ を直接求める必要がある．二つめはクリティカルカットを列挙することである．そしてもう一つは，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．[3] のアルゴリズムは配線領域をスキャンしながら，$\mathbf{G}_f$ の辺とすでに求められた $\mathbf{G}_c$ の辺でできる四辺形に対して次々とフリップを適用し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 [3, 5] 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，[3] 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．

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．改良された配線可能性検証アルゴリズムは，1 回目のスイープでフロー導出グラフ $\mathbf{G}_f$ を求め，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を端点とするクリティカルカットを一本ずつ発見し，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，[3] 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．[3] の列挙アルゴリズムではスイープ

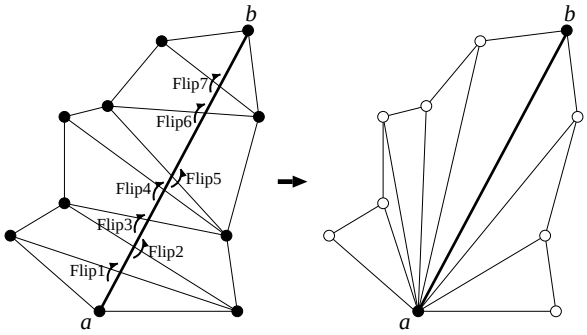

図 2: フリップ操作によるフロー計算

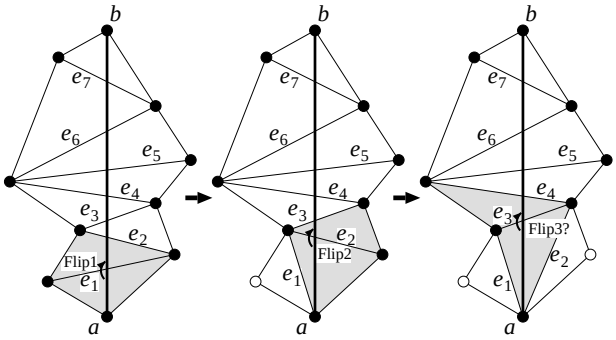

図 3: フリップ操作を連続してできない例

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．各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，スイープラインが次のイベントポイントへ移動するまでの間は，それらのクリティカルカットのフロー値を記憶しておく．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，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．$\mathbf{G}_f$ の各辺については，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 $\mathbf{G}_f$ の辺と一致または $\mathbf{G}_f$ の辺と交差する．$\mathbf{G}_f$ の辺と一致する場合は何もする必要がない．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を適用しクリティカルカットのフローを計算する ([5] の図 7)．図 2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)．

しかし交差辺へのフリップの適用順序によっては，フリップ操作を適用できなくなる場合がある．例えば図 3で，(a, b) と交差する分割辺を 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 3の e_4)．

本節では，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良いかを考える．クリティカルカットと交差する辺を a から近い順に $E_{ab} = \{e_1, e_2, \cdots, e_m\}$ とする．a, b を通る直線により分けられる二つの半平面を 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 4(a))．

フリップすべき四辺形を選択する方法の一つは，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 3 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に関するフリップ操作は行わず次のインデックスの辺から再び適用できるものを探す．このフリップの結果，今まで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るのであればそれらについてもフリップ操作を適用する．このようにしてクリティカルカットのフローを計算する方法を NearestFirst 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 [6]．

以下では，最悪の場合にはこれと同程度の手間を

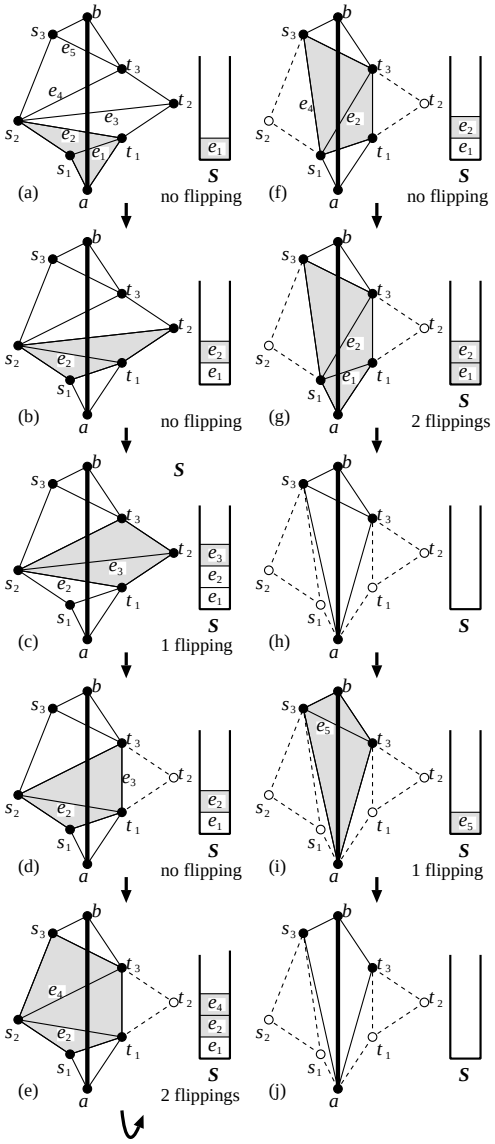

図 4: フリップ操作の適用順序

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 3 の例では，NearestFirst 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存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 を行う．

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
　while （*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）**do 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 (u_i, u_{i+2}) のフローを計算する（図 4(c)）．スタックから e を pop する．
　end;
　while（*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）**do begin**
　　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 4(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 4(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一つだけ残した図形を得る（図 4(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，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，大抵の場合，クリティカルカットのフローは $O(n)$ 時間で計算できる．

しかし，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場合がある．このときは，その得られた結果に対して NearestFirst を適用しフリップを繰り返す．

この操作では，クリティカルカットと交差する辺を一つ削除するために，$O(n)$ の時間を必要とし，クリティカルカットのフロー計算に $O(n^2)$ 時間を必要とする．しかし，これは特殊な点配置のときにしか起こらず，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ことができる．

5　プリント配線板データへの適用

本節では，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 節

表 3: 使用メモリ量と実行時間

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

(計算機: SPARCstation 10)

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，[3] のアルゴリズムおよび改良されたアルゴリズムを実行するために必要としたメモリ量をそれぞれ M_1, M_2で示す．M_vは [4] 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，本節のアルゴリズムの実行時に使用する記憶量は [3] のアルゴリズムより 26 %から 32 %のメモリしか使用していないことがわかる．(これは，表 2における $(|\mathbf{V}_c|+|\mathbf{E}_f|)$ と $(2\times|\mathbf{V}_c|+|\mathbf{E}_c|+|\mathbf{E}_f|)$ 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．[3] 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 $|\mathbf{E}_c|$ である．一方，本アルゴリズム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 (従って計算時間も) は増加する．

表 3は本アルゴリズムを実行したときの実行時間を示している．T_1は [3] のアルゴリズム，T_2は本アルゴリズムを実行したときのユーザ時間である．T_2は従来法の計算時間 (T_v) と比較して十分小さいことがわかる．すなわち，フローの再計算を行っているにも関わらず，実用的な時間で配線可能性検証が実現できていることがわかる．

このように，4節で述べた配線可能性検証アルゴリズムは，実用上十分な速度をもちながら，[3] のアルゴリズムに比べて使用するメモリ量が少ないことがわかる．

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，従来手法と比べて配線可能性検証を高速に実行できた．

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．

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 DA 応用設計部長，堅田敏幸 主任技師，森山隆志 氏に感謝致します．本研究の一部は財団法人 堀情報科学振興財団の援助を受けた．記して感謝します．

参考文献

[1] R.Cole and A.Siegel, “River routing every which way, but loose,” Proc. of 25th IEEE Symp. on Foundations of Computer Science, pp.65–73, 1984.

[2] Toshiyuki Hama, Hiroaki Etoh, “Topological Routing Path Search Algorithm with Incremental Routability Test,” Proceedings of the Asia and South Pacific Design Automation Conference, pp. 645–648, 1997.

[3] Naoyuki Iso, Yasushi Kawaguchi, Tomio Hirata, “Efficient Routability Checking for Global Wires in Planar Layouts,” IEICE Trans. Fundamentals, Vol. E80–A, No.10, pp. 1878–1882, 1997.

[4] C.Leiserson and F.Maley : “Algorithms for routing and testing routability of planar VLSI layouts,” Proc. of 17th ACM Symp. on Theory of Computing, pp.69–78, 1985.

[5] 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.

[6] 譚 学厚, Private Discussion.