Title	Simulated Quenching法に基づく２次元セル配置最適化手法
Author(s)	平間，孝廉
Citation	
Issue Date	2001-03
Type	Thesis or Dissertation
Text version	author
URL	http://hdl.handle.net/10119/1469
Rights	
Description	Supervisor:金子 峰雄，情報科学研究科，修士

修 士 論 文

Simulated Quenching 法に基づく２次元セル配置最適化手法

指導教官　金子峰雄 助教授

北陸先端科学技術大学院大学
情報科学研究科情報システム学専攻

平間孝廉

2001 年 2 月 15 日

目 次

図 目 次

表 目 次

第 1 章

はじめに

1.1 本研究の背景

VLSI は計算機，通信機器を始めとする種々情報処理，信号処理システムの主要構成要素であり，製造技術の進歩に伴って回路の大規模化と微細化が急速に進んでいる．また，低消費電力化，小型化，高速高機能化の要求とも相伴って VLSI 設計は非常に困難な問題となっている．一方，多品種少量生産や設計期間の短縮などの要求が高まり，設計の自動化が重要な課題となっている．

VLSI の設計の工程は，システム設計，RTL 設計，論理設計，回路設計，そしてレイアウト設計と階層的に大きく分類することができる．この中でレイアウト設計は，チップ上にコンポーネントを配置しそれらの間の配線を行う工程である．多くの場合，チップ面積の縮小，総配線長の最小化などが求められるが，こうした評価に基づく最適配置問題は NP 困難であることが知られている．一般に NP 困難と言われいる問題の大域的最適解を探索するアルゴリズムとしては，Simulated Annealing(SA) 法 [5] や Genetic Algorithm(GA) といった確率的最適化手法が提案されている．SA 法は現在の解から隣接解を作成し，確率的に隣接解の受理と破棄を繰り返すものであり，十分な温度スケジューリング (アニーリング) の下で最適解に到達することが保証されているアルゴリズムである．また，GA 法は数種類の解により初期集団を用意しその中から評価の良い解を選択し交配させ，さらに局所解に陥ることを防ぐために突然変異を取り入れつつ良い解の集団を探索するアルゴリズムである．これらの手法は配置問題においても適用されてきているが，良質な解を求めるためには，アルゴリズム内部で多くの繰り返し回数を必要とすることから計算時間がかかり，大規模な問題を実用時間内で解くことのできる効率的手法が求められている．

1.2 本研究の目的

本研究では，１次元配置問題に対して，SA 法や GA 法といった代表的確率的最適化手法による解と同等の解をより高速に得ることが出来る Simulated Quenching (SQ) 法に着目し，２次元格子状に並ぶスロットへコンポーネントを重なりなく割り付ける２次元配置問題の最適化手法を提案する．１次元配置における SQ 法の成功の要因は各配置修正毎の (i) スロットとコンポーネントのサブグループ化，(ii) 部分問題に対する目的関数，(iii) 部分再配置問題の解法，にあるとの立場に立ち，２次元配置問題への拡張にあたっては，こうした特徴が保持される解法を目指した．

本稿では部分問題の構成の仕方が異なる３つの２次元 SQ 法を提案する．始めに，１次元 SQ 法と同じくサブグループの生成を一次元的に行うと共に，各ネットのバウンディングボックス上にあるコンポーネントのみに着目した部分問題構成方法に基づく手法 2DSQFVβ(2D Simulated Quenching based on ForceValue type β) 法を提案し，次に，2DSQFVβ 法の部分問題の生成方法を２次元に拡張した 2DSQFV(2D Simulated Quenching based on Force-Value) 法を提案する．最後に，より厳密に各ネットのバウンディングボックスのサイズを反映させた部分問題構成に基づく 2DSQSC(2D Simulated Quenching based on StepCost) 法を提案する．

1.3 本論文の構成

本論文は，第２章で１次元配置問題と２次元配置問題を説明し，１次元配置最適化手法である Simulated Quenching 法と，Simulated Annealing 法を用いた２次元配置最適化手法を説明する．第３章，第４章で 2DSQFVβ 法，2DSQFV 法を提案する．第５章で 2DSQSC 法を提案し，第６章で結論と今後の課題を述べまとめとしている．

第 2章

配置問題と確率的解法

入力をコンポーネント (component) の集合 V_H，コンポーネント同士を接続するネット (net) の集合 E_H からなるハイパーグラフ $G_H = (V_H, E_H)$ とする．出力を等間隔に並ぶスロット (slot) へのコンポーネントの割り当てとする．

2.1　１次元配置問題

１次元配置問題で，スロットは直線上に並んでいる．コンポーネントはスロットに重複せずに配置されるものとする．あるネット e_i のネット長を，

$$Len\ (e_i) = (\max\{x(v_j)|v_j \in e_i\} - \min\{x(v_k)|v_k \in e_i\})$$

と表す．ただし，$x(v)$ はコンポーネント v の x 座標を表わす．目的関数 $(COST_{total})$ を全ネットに対する Len の総和とする．

$$COST_{total} = \sum_{e_i \in E_H} Len(e_i)$$

2.2　２次元配置問題

２次元配置問題で，スロットは平面上に縦横等間隔に並んでいる．コンポーネントはスロットに重複せずに配置されるものとする．あるネット e_i のネット長をバウンディングボックス (boundingbox) の半周長；

$$\begin{aligned} HP\ (e_i) = {} & (\max\{x(v_j)|v_j \in e_i\} - \min\{x(v_k)|v_k \in e_i\}) \\ & +(\max\{y(v_\ell)|v_\ell \in e_i\} - \min\{y(v_m)|v_m \in e_i\}) \end{aligned}$$

とする (図 2.1)．ただし，$x(v)$，$y(v)$ はコンポーネント v の x 座標，y 座標を表わす．目的関数 ($COST_{total}$) を全ネットに対する HP の総和とする．

$$COST_{total} = \sum_{e_i \in E_H} HP(e_i)$$

HP は実際のネットの配線長を評価するものではない．しかし，あるネットに属するコンポーネント間の距離を見積もることはでき，2 次元配置問題では良く使われる評価方法である．

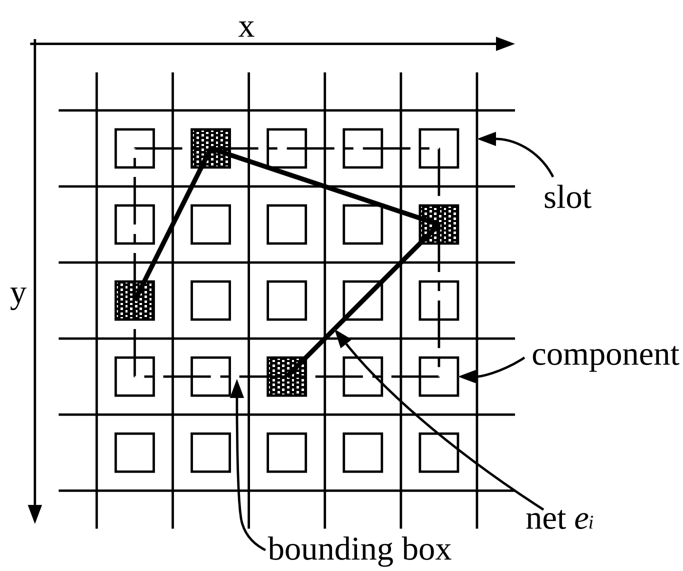

図 2.1: ネット e_i のバウンディングボックス

2.3 Simulated Quenching法

SQ(1DSQ) 法は1次元配置問題を解くために提案された確率的繰り返し最適化手法である．1DSQ 法のアルゴリズムは以下の 1,2,3 をピッチ (pitch) がスロット幅から 2 になるまで減少させつつ繰り返し実行し，１次元配置を出力するものである．

2.3.1 アルゴリズム

1. スロットとコンポーネントのサブグループ (subgroup) 化

   直線上のある配置に対して，適当な指針で制御されるピッチと確率的に選択されるオフセット (offset) に基づいてカットライン (cutline) を引くことにより，スロットとコンポーネントのサブグループ化を行う．

2. フォースバリューの設定

   一つのネットに含まれるコンポーネントがカットラインを越えて複数のサブグループに存在しているとき，フォースバリュー (force value) として，ネットの左端のコンポーネントに対して +1 の値を，右端のコンポーネントに対して −1 の値を与える (図 2.2)．

3. 再配置

   各コンポーネントに対し，そのコンポーネントに接続するすべてのネットに関するフォースバリューを合計し，その値をキーとして各サブグループ内でコンポーネントをソートしたものを新しい配置とする (図 2.3)．また，ソートにはバケットソートを用いている．

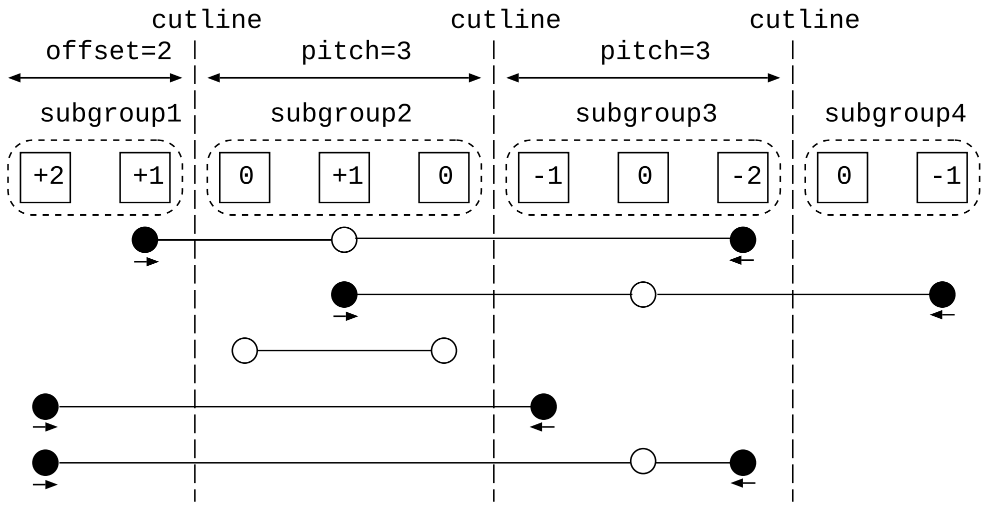

図 2.2: サブグループの生成とフォースバリューの計算

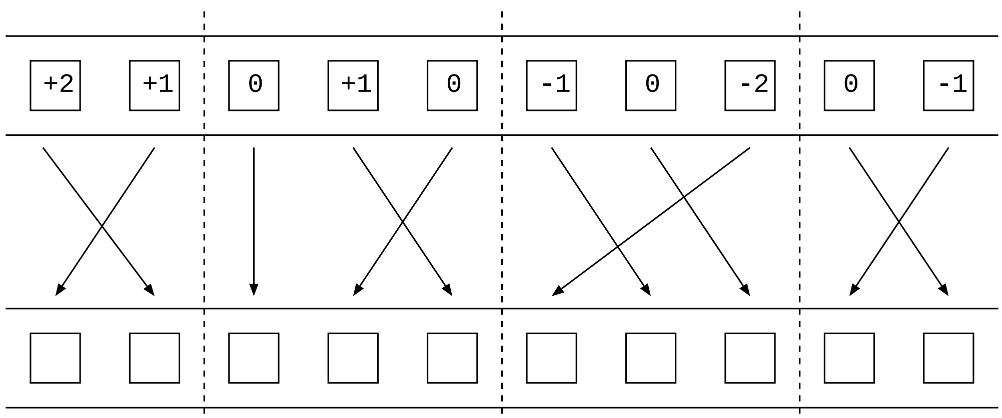

図 2.3: フォースバリューに基づく再配置

2.3.2　部分問題

以上の 1DSQ 法において，各サブグループ内のコンポーネント再配置を部分問題と捉えれば，この部分問題は等価的に次のように考えられる．

入力 ：コンポーネント，スロットの部分集合，及びコンポーネントのスロットへの (初期) 割り当て x_0

出力 ：コンポーネントのスロットへの割り当て x

目的関数 ：

$$\min \sum_{e_i \in E_H \setminus \tilde{E}_H} \left(x(v_j)\Big|_{\substack{v_j \in e_i \\ x_0(v_j) \geq x_0(v), v \in e_i}} - x(v_k)\Big|_{\substack{v_k \in e_i \\ x_0(v_k) \leq x_0(v), v \in e_i}} \right)$$

但し，$\tilde{E}_H$ はサブグループ内のコンポーネントのみからなるネットの集合とする．
部分問題と原問題との関係は以下の通りである．

1. 部分問題は原問題を忠実に反映してない．特にネットの部分集合 $\tilde{E}_H$ の要素 (一つのサブグループ内に閉じたネット) に対するネット長が無視される点が特徴的である．

2. ピッチが小さくなるにつれて部分問題と原問題の評価の差は小さくなり，ピッチ 2 では等価的に等しくなる．

プログラムは以下のように記述する．

```
スロットに各コンポーネントが重複せずに置かれている．
begin
pitch = slot 数;

for(pitch; pitch>=2; pitch-=SLOW) {
  for(j=0; j<LOOP; j++) {
    サブグループ生成;

    フォースバリューの計算;

    サブグループ内でコンポーネントのならびをソートする;
  }
}
end
```

（SLOW はピッチの減少率を表し，$SLOW = 0.03 \times pitch / \log_2 pitch$ とする．LOOP は，同一ピッチでの繰り返しの回数を表す.）

2.4 Simulated Annealing 法

SA 法は熱統計力学的現象を応用した確率的最適化手法であり，焼きなまし法とも呼ばれている．温度という値を調整することにより，アルゴリズムの収束を制御している．また，十分な温度スケジューリングの下で最適解に到達することが保証されている．

2.4.1 隣接解

SA 法を用いた 2 次元配置問題における隣接解は，任意の異なる 2 つのコンポーネントを選択し入れ替えを行うこととする (図 2.4).

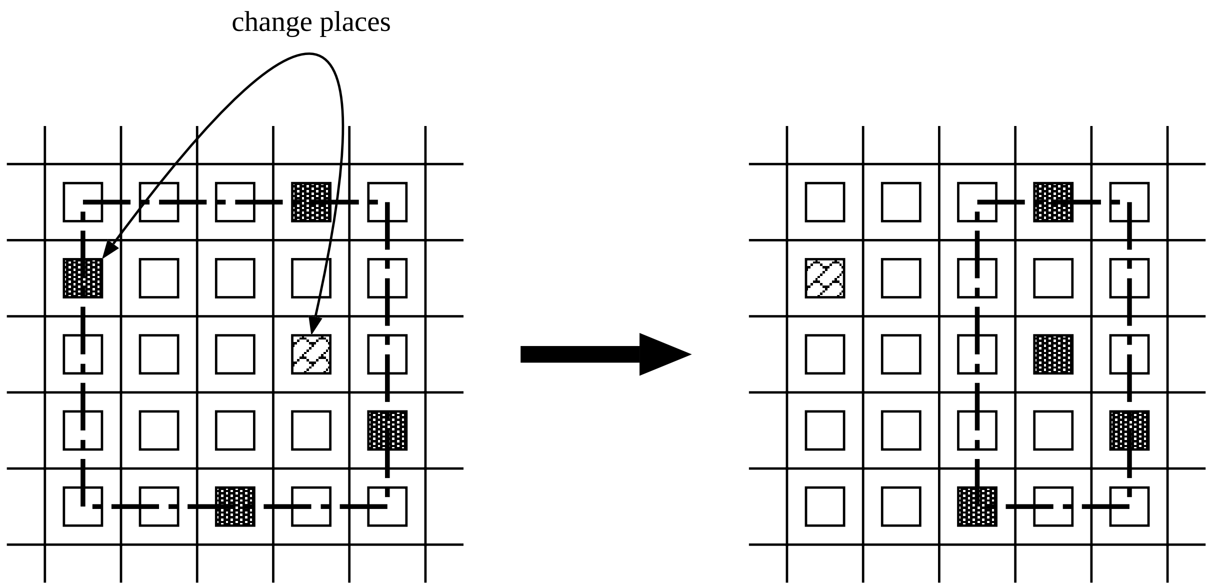

図 2.4: SA による入れ替え

2.4.2 アルゴリズム

入れ替えはSA 法に基づき，次のように行う．

ある配置状態の評価値を $E0 = COST_{total}0$，入れ替えを行った場合の評価値を

$E1 = COST_{total}1$ とし，

$$\varDelta E = E1 - E0$$

を設定する．また，k_b を $Boltzmann$ 定数，T を温度とすると，

$$p = \exp(-\frac{\varDelta E}{k_b T})$$

により，p が求められる．この p と 0 以上 1 以下の乱数を比較することにより，入れ替え後の配置状態を解として選択するかどうかを決定する．アルゴリズムは以下に示す．

スロットに各コンポーネントが重複せずに置かれているものとする．

```
begin
E0 = 評価値を計算;

for(TEMP=Start_temp; TEMP>TEMP_low; TEMP*=SLOW) {
  for(j=0; j<COMP_size*Loop_times; j++) {
    任意の異なる 2 つのコンポーネントを選択し，入れ替えを行なう;

    E1 = 評価値を計算;

    if ( E1 > E0 ) {
      if ( p < (0〜1 の乱数) ) {
        入れ替えたコンポーネントをもとの配置状態に戻す;
      }
    }
  }
}
end
```

第 3 章

2DSQFVβ法

SQ(1DSQ) 法は 1 次元配置問題を解くために提案された確率的繰り返し最適化手法である．これを 2 次元配置問題に拡張した手法を提案する (2DSQFVβ).

本手法は適当な初期解から始めて，配置修正を繰り返し行ない最終配置を得るものである．配置修正は垂直線分カットラインによるサブグループ化・配置修正と水平線分カットラインによるサブグループ化・配置修正を一組として行なうこととし，カットラインのピッチを急速に小さくする中で，同一ピッチに対して複数回オフセットをランダムに選択しながら配置修正を繰り返す方式を採用している．

2DSQβ 法のアルゴリズムは以下の 1,2,3 をピッチ (pitch) がスロット幅から 2 になるまで減少させつつ繰り返し実行し，１次元配置を出力するものである．

3.1　アルゴリズム

1. スロットとコンポーネントのサブグループ化

   平面上のある配置に対して，適当な指針で制御されるピッチと確率的に選択されるオフセットに基づいてカットラインを引くことにより，スロットとコンポーネントをサブグループ化する．

   ここでは垂直線分をカットラインとする平面分割に基づくサブグループ化及び水平線分をカットラインとする平面分割に基づくサブグループ化を交互に採用する．図 3.1 は垂直のカットラインに基づくサブグループ化の一例である．

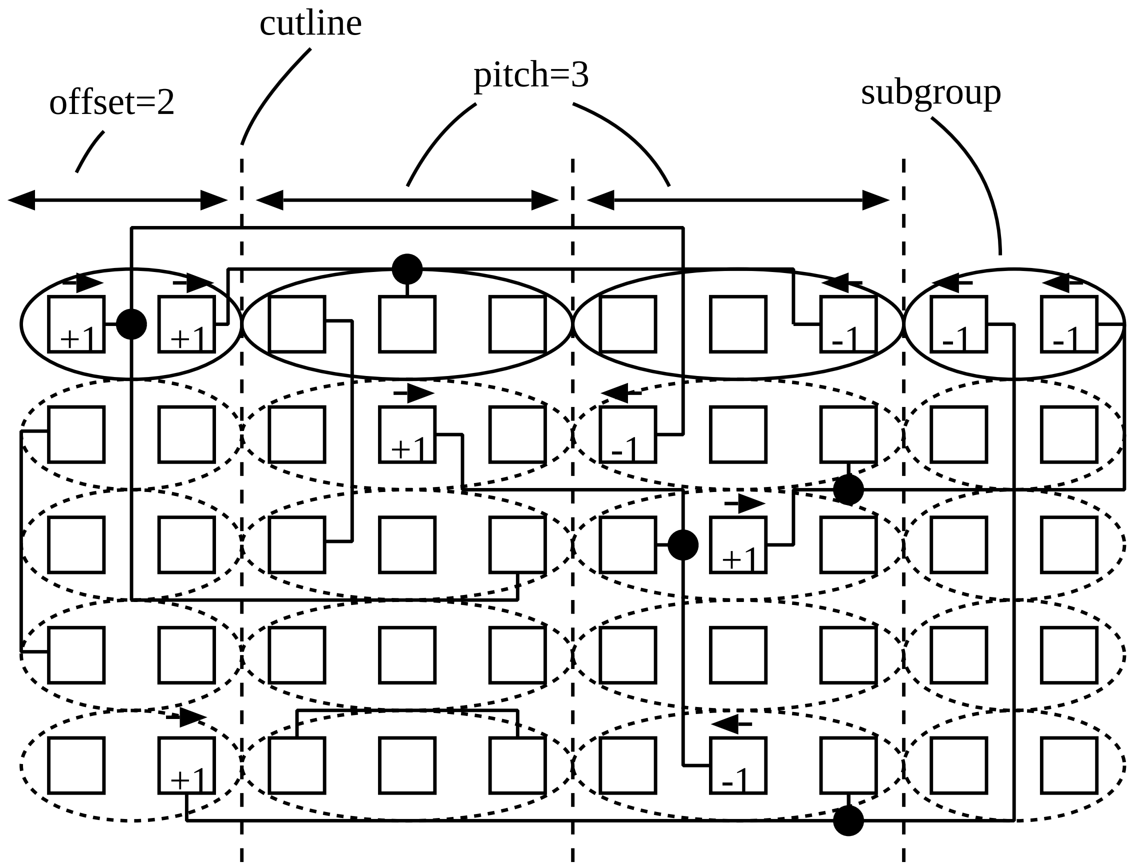

図 3.1: 垂直線分 cutline による分割と subgroup 化

2. フォースバリューの設定

   SQ を用いた 1 次元のレイアウトにおいてフォースバリューはネットの両端のコンポーネントに対して与えられていたが，2 次元のレイアウトで SQ を用いるときは，図 3.1 のように各ネットにおいてバウンディングボックスを形成しているコンポーネントに対して与える．

   評価関数の定義式を利用し，x 軸で最も座標の値が高いコンポーネント v_j に -1 を加算し，x 軸で最も座標の値が低いコンポーネント v_k に $+1$ を加算する．同じように，y 軸においても同様に，最も座標の値が高いコンポーネント v_l に -1 を加算し，y 軸で最も座標の値が低いコンポーネント v_m に $+1$ を加算する．

3. 再配置

   各コンポーネントに対し，そのコンポーネントに接続するすべてのネットのフォー

スバリューを合計し，その値を基に再配置問題を解く．再配置は，垂直のカットラインに基づくサブグループ化が行われたときは，一行づつ切り出して，1 次元レイアウトのときと同じように，ソートにより行う．水平のカットラインに基づくサブグループ化が行われたときは，一列づつ切り出して，同じようにソートにより行う．

3.2 部分問題

各サブグループ内のコンポーネント再配置を部分問題と捉え，その目的関数をカットラインを跨ぐ配線のみに着目して次の通り設定する．但し以下は，垂直線分をカットラインとしてサブグループ化されたグループの個々のコンポーネント集合を p_1,p_2, $\cdots$, p_K としたときの，p_k を再配置する部分問題に対するものである．

$$
\begin{aligned}
c_k &= \sum_{e_i \in L_k \cup R_k} [(\max\{x(v_j)|v_j \in e_i\} - \min\{x(v_j)|v_j \in e_i\})] \\
R_k &= \left\{ e_i \,\middle|\, e_i \in E_H, p_k \cap e_i \neq \emptyset, (\textstyle\bigcup_{\ell=1}^{k-1} p_\ell) \cap e_i = \emptyset, (\bigcup_{\ell=k+1}^{K} p_i) \cap e_\ell \neq \emptyset \right\} \\
L_k &= \left\{ e_j \,\middle|\, e_j \in E_H, p_k \cap e_j \neq \emptyset, (\textstyle\bigcup_{\ell=1}^{k-1} p_\ell) \cap e_j \neq \emptyset, (\bigcup_{\ell=k+1}^{K} p_\ell) \cap e_j = \emptyset \right\}
\end{aligned}
$$

垂直 (水平) 線分カットラインによってサブグループ化されたコンポーネントの集合に対してコンポーネントの水平 (垂直) 方向移動のみを許した再配置を行なう．以下先と同様に垂直線分をカットラインとしてサブグループ化されたものの内の k 番目のコンポーネント集合 p_k の部分再配置問題を考える．

先ず始めに，与えられた配置において各コンポーネント v に対して以下で定義されるフォースバリュー $f(v)$ を計算する．

$$
\begin{aligned}
f(v) &= |\{e_i|v \in e_i \in R_k \text{ and } v \text{ is the left most component in } e_i\}| \\
&\quad -|\{e_j|v \in e_j \in L_k \text{ and } v \text{ is the right most component in } e_j\}|
\end{aligned}
$$

次いで，各コンポーネントをフォースバリューの小さい順に左からスロットに割り付け，再配置とする．プログラムは次のように記述する．

```
スロットに各コンポーネントが重複せずに置かれている.
begin
pitch = 横軸方向のスロットの数;

for(pitch; pitch>=2; pitch*=SLOW) {
  for(j=0; j<LOOP; j++) {
    x 軸方向でサブグループ生成;
    x 軸方向のフォースバリューを計算する;
    x 軸方向においてサブグループ内のコンポーネントをソートする;

    y 軸方向でサブグループ生成;
    y 軸方向のフォースバリューを計算する;
    y 軸方向においてサブグループ内のコンポーネントをソートする;
  }
}
end

(SLOW は, pitch の減衰率を表し,
LOOP は, 同一 pitch での繰り返し回数を表すものとする. )
```

3.3 cutline の zigzag 化

SQ 法をこのようにして 2 次元配置問題へ拡張し適用したところ，ピッチが 2 であるときの図 3.2 の左側のような問題が考えられた．

そこで，次にカットラインの引き方を図 3.2 の右側のように zigzag にすることにより，この評価の向上を図った．

また，これは図 3.3 のように，奇数行 (列）か偶数行 (列）どちらかの座標を一つずらして考えていることと同じである．

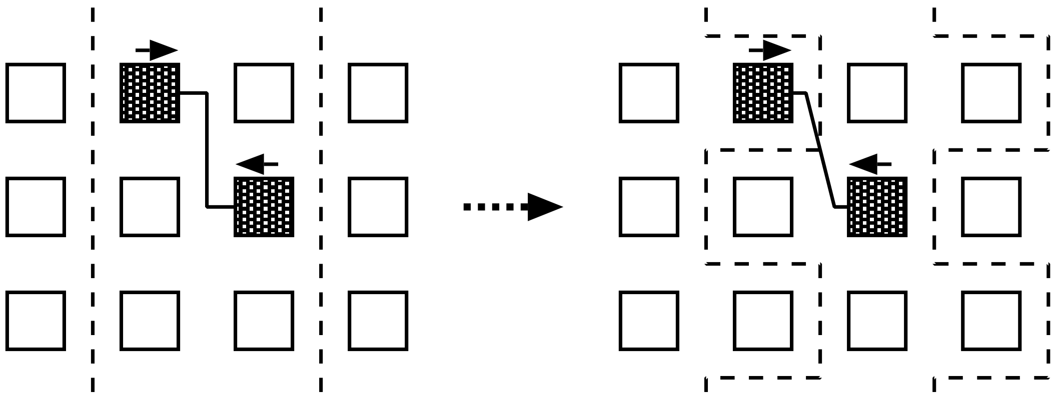

図 3.2: zigzag cutline の引き方

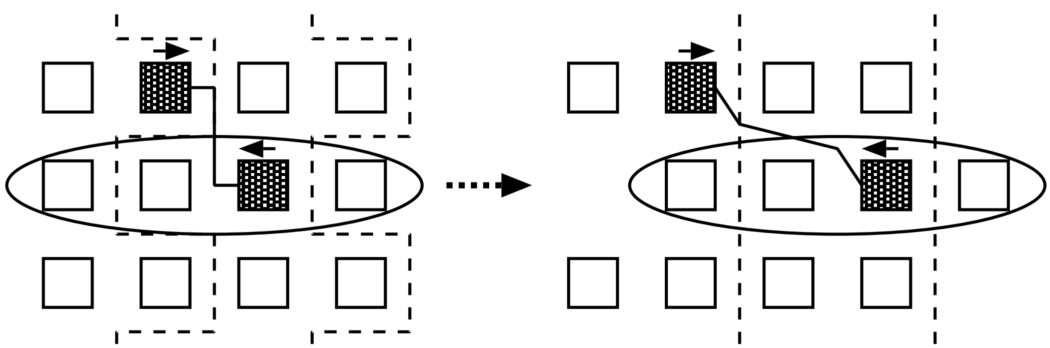

図 3.3: zigzag cutline の解釈

3.4 実験結果

実験はコンポーネント数 400，ネット数 3200，コンポーネント数 1600，ネット数 12800，コンポーネント数 3600，ネット数 28800 のものに対して行った．

SA による結果を plot したグラフを図 3.4 に，直線 cutline SQ による結果を図 3.5 に，zigzag cutline SQ による結果を図 3.6 に示す．

このグラフをみると，直線 cutline SQ よりも，zigzag cutline SQ のほうが値が落ち始めるのが早い事がわかる．

また，表 1 に SA と SQ zigzag の比較を示す．なお SQ zigzag については，同一 pitch に対してそれぞれ 640, 320, 160 回の配置修正を行なった場合の結果を示している．SQ における pitch 降下法，同一 pitch における修正回数等の修正スケジューリングに関して尚検討・改善の余地が大きいものの，SA に匹敵する性能が確認できた．

しかし Circuit 1 のように規模が小さいデータでは評価は SA には若干及ばず，計算時間を同程度掛けても僅かに及ばない結果となってしまっている．ソートの範囲を「行」と「列」で考えている限界が出ていると考えられる．

表 3.1: 実験結果 (総配線長，実行時間評価)

Test data	# components / # nets		SA	SQ(640)	SQ(320)	SQ(160)
Circuit 1	400 / 3200	Total net length	63710	65013	65053	65150
		Run time	81s	83s	40s	20s
Circuit 2	1600 / 12800	Total net length	508261	507323	508064	508574
		Run time	454s	559s	284s	146s
Circuit 3	3600 / 28800	Total net length	1707808	1701203	1702623	1704879
		Run time	1190s	1597s	825s	437s

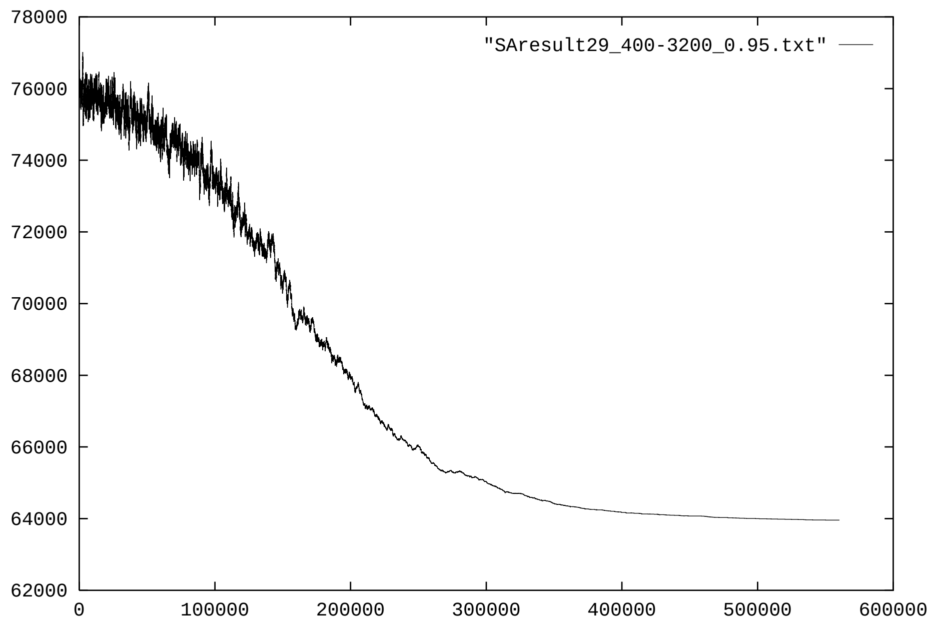

図 3.4: 400-3200 SA

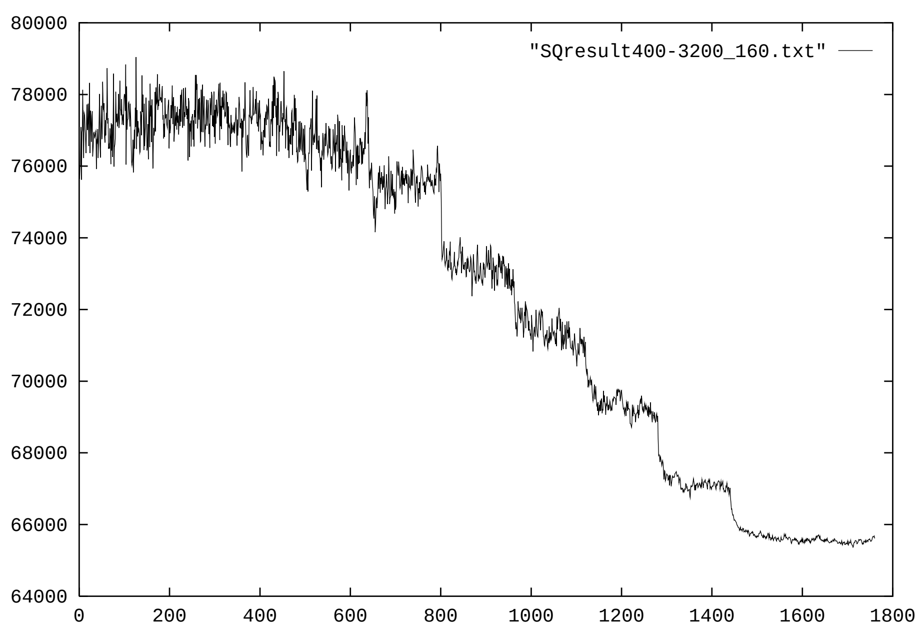

図 3.5: 400-3200 SQ 直線 cutline

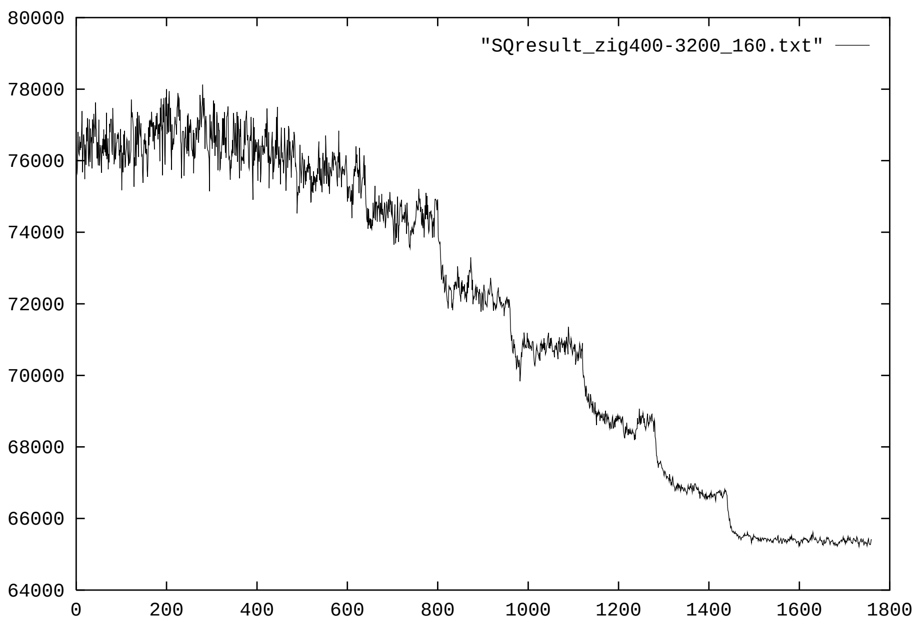

図 3.6: 400-3200 SQ zigzag cutline

第 4 章

2DSQFV 法

ここでは，各ネットのバウンディングボックス上にあるコンポーネントのみに着目した部分問題構成に基づく 2 次元 SQ 法のアルゴリズムと配置実験について述べる．採用した部分問題構成は，1DSQ における 1 次元的サブグループ化を 2 次元的サブグループ化に，ネット両端のコンポーネントをバウンディングボックス上のコンポーネントに，それぞれ対応させたものとなっている．

4.1 アルゴリズム

アルゴリズムは以下の 1,2,3 をピッチがスロット幅から 1 になるまで減少させつつ繰り返し実行し，2 次元配置を出力する．

1. カットラインとサブグループ生成

   平面上のある配置に対して，適当な指針で制御されるピッチと確率的に選択されるオフセットに基づいてカットラインを縦横に引くことにより，スロットとコンポーネントをサブグループ化する (図 4.1).

2. フォースバリューの設定

   一つのネットに含まれるコンポーネントが垂直のカットラインを跨いで存在しているときにその左右両端のコンポーネントに与えるフォースバリューを force$H$ とし，左端のコンポーネントに対して +1 の値を，右端のコンポーネントに対して -1 の値を与える．また，水平のカットラインを跨いで存在しているときにその上下両端のコンポーネントに与えるフォースバリューを force$V$ とし，上端のコンポーネントに対して +1 の値を，下端のコンポーネントに対して -1 の値を与える (図 4.2).

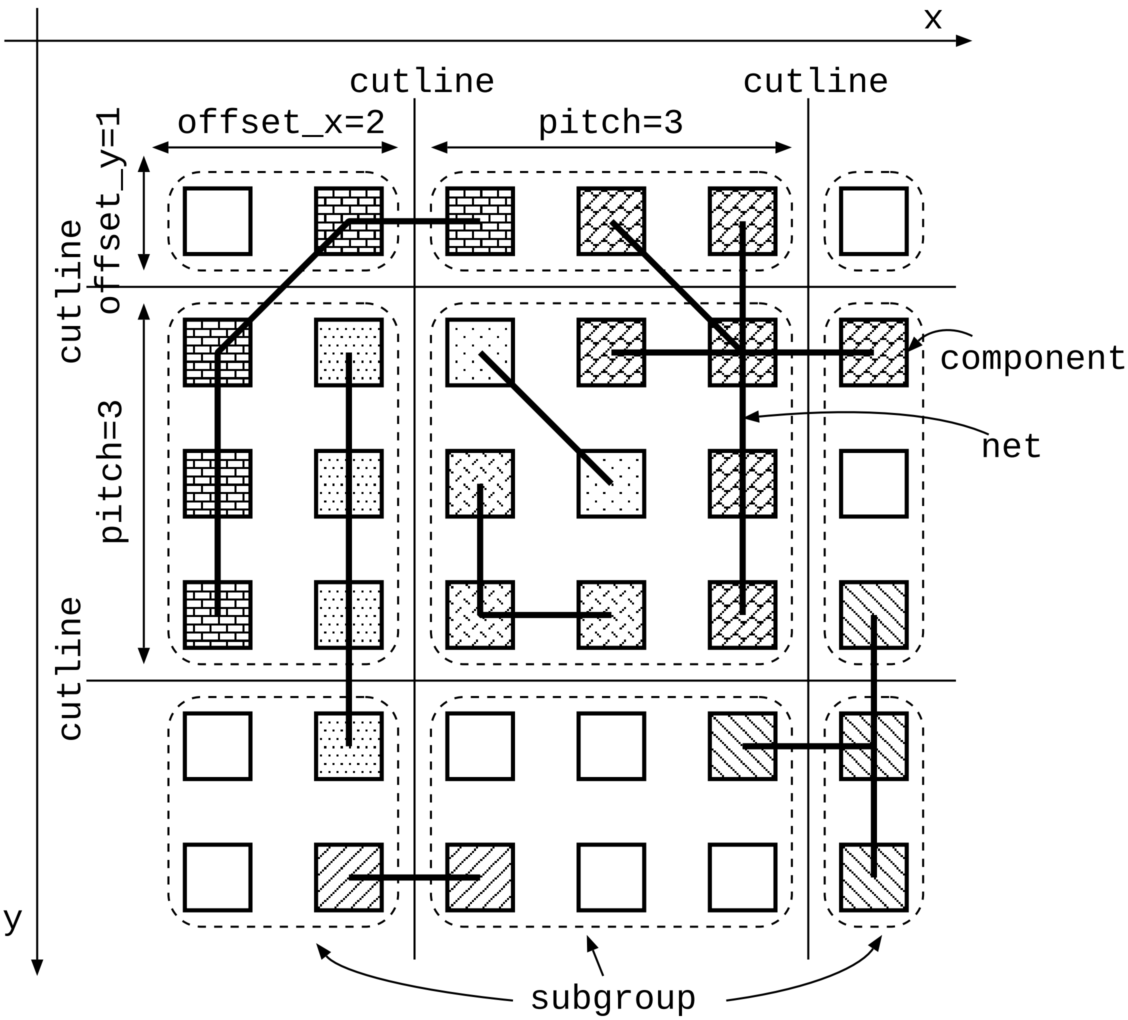

図 4.1: カットラインの引き方とサブグループ生成について

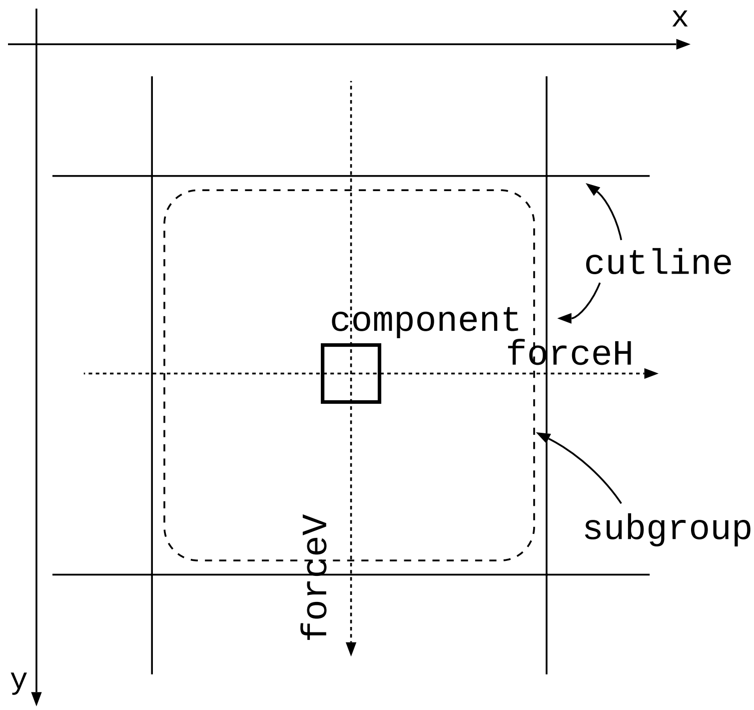

図 4.2: フォースバリュー

3. 再配置

   各コンポーネントに対し，そのコンポーネントに接続するすべてのネットのフォースバリュー force$H$，force$V$ それぞれを合計し，その値を基に再配置問題を解く．再配置は，コンポーネントの集合 V とスロットの集合 S に対し，$V \cup S$ を頂点集合とする重みつき完全 2 部グラフの最大コスト完全マッチングを求める問題として考える．すなわち，forceH_i,forceV_i を持つコンポーネント v_i と座標 (x, y) のスロットを結ぶ枝（図 4.3）に以下に示すコスト $(cost)$ を与えることにより，V と S を部分頂点集合とする重み付き完全 2 部グラフを構成する．

$$
\begin{aligned}
& cost(\mathrm{force}H_i, \mathrm{force}V_i, x, y) \\
& \quad = \alpha \cdot (\mathrm{force}H_i \times x + \mathrm{force}V_i \times y) - stb(i, x, y)
\end{aligned}
$$

   ここで，stb はコンポーネントの現在の配置位置からサブグループ内の各スロットへのマンハッタン距離を表す 4.4．これは，フォースバリューによるコストが同等である時に，現在位置に近いものを優先する役割を果たすものである．なお α の値は，$|stb(i, x, y)|$ より十分大きいものとする．このグラフ上で最大コスト完全マッチング

を求めることにより，コンポーネントを割り当てるスロットを決める．現在，最大コスト完全マッチングには，Hungarian method[3][4] を用いている．

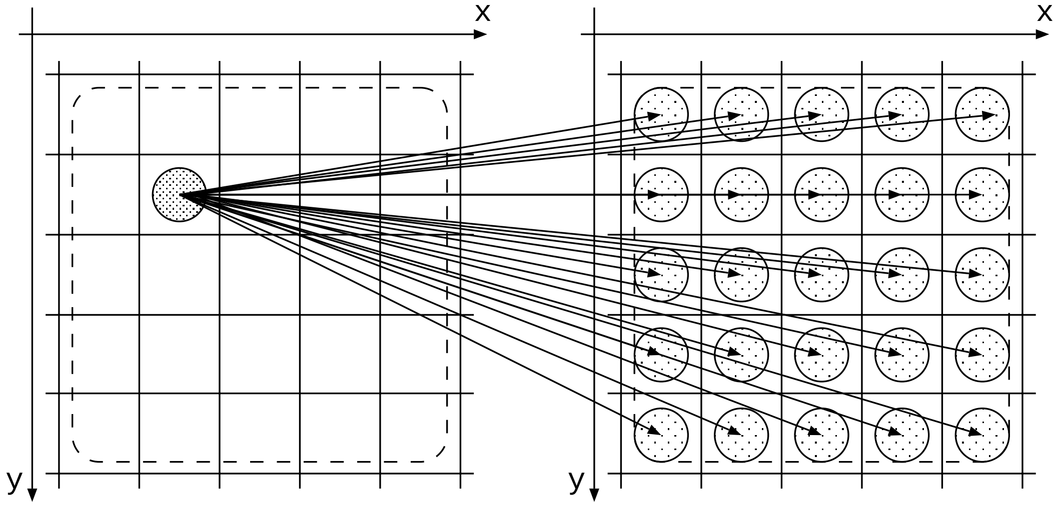

図 4.3: コンポーネントのスロットへの配置可能枝

4	3	2	3	4
3	2	1	2	3
2	1	0	1	2
3	2	1	2	3
4	3	2	3	4

図 4.4: stability

Hungarian method は重みつき２部グラフの最大コスト完全マッチングを求めるアルゴリズムであり，最適解が保証されている．また，このアルゴリズムの計算量は $O(n^3)$ である．以下に Hungarian method のアルゴリズムを示す．

HUNGARIAN(n,w;$mate$)
begin;
for $v \in V$ do $mate(v) = 0$ od;
for $i = 1$ to n do $u_i = \max\{w_{ij} : j = 1, ..., n\}$;$v_i = 0$ od;
$nrex = n$;
while $nrex \neq 0$ do;
    for($i = 1$ to n do $m(i) = false$; $p(i) = 0$; $\delta_i = \infty$ od;
    $aug = false$; $Q = i \in S : mate(i) = 0$;
    do
        remove some arbitrary vertex i from Q; $m(i) = true$; $j = 1$;
        while $aug = false$ and $j \leq n$ do
            if $mate(i) \neq j'$
            then if $u_i + u_j - w_{ij} < \delta_j$
                $\delta_j = u_i + v_j - w_{ij}$; $p(j) = i$;
                    if $\delta_j = 0$
                    then if $mate(j') = 0$
                        then
                            AUGMENT($mate$,p,j';$mate$);
                            $aug = true$; $nrex = nrex - 1$;
                        else $Q = Q \cup mate(j')$;
                        fi
                    fi
                fi
            fi
            $j = j + 1$;
        od;
        if $aug = false$ and $Q = 0$
        then
            $J = i \in S : m(i) = true$; $K = j' \in T$; $\delta_j = 0$;

$\delta = \min \delta_j : j' \in T \setminus K$;

for $i \in J$ do $u_i = u_i - \delta$ od;

for $j' \in K$ do $v_j = v_j + \delta$ od;

for $j' \in T \setminus K$ do $\delta_j = \delta_j - \delta$ od;

$X = j' \in T \setminus K : \delta_j = 0$;

if $mate(j') \neq 0$ for all $j' \in X$

then

for $j' \in X$ do $Q = Q \cup mate(j')$ od;

else

choose $j' \in X$ with $mate(j') = 0$;

AUGMENT(mate,p,j';mate);

$aug = true$; $nrex = nrex - 1; break$;

fi

fi

while $aug = true$;

od

end

AUGMENT($mate$,p,j';$mate$)

begin

do

$i = p(j)$; $mate(j') = i$; $next = mate(i)$; $mate(i) = j'$;

if $next \neq 0$ then $j' = next$; fi

while $next = 0$;

end

4.2 実験結果

実験では，コンポーネント数 16，ネット数 32 を持つハイパーグラフに対し，配置生成を 10 回行なった．表 4.1 にこのときの配線長の平均と分散，実行時間の平均を示す．また，ここでの type1 は上でのべた手法であり，type2 はフォースバリューが発生するコンポーネントと同一のネット，同一のサブグループに属すコンポーネントに対してもフォースバリューを加算する手法である．

表 4.1: 2DSQFVtype1,2DSQFVtype2 result

		2DSQFVtype1	2DSQFVtype2
16-32	ave	149.4	126.9
	var	41.16	5.43
	time	11s	21s

この結果，後者の方が収束が良く，最終結果も良い評価値を示すことがわかった．これは，フォースを与えるコンポーネントが増え，一度収束したバウンディングボックスの配置が崩れにくくなったためだと考えられる．

また，400 コンポーネント，3200 ネットのデータを使用し実験を行った結果をプロットしたものを，図 4.5，図 4.6 に示す．結果をみると type2 の方が良い最終結果を出していることが解る．この問題において，type2 の評価値は 63900 台を示すが，type1 の評価値が 64000 以下になることはなかった．

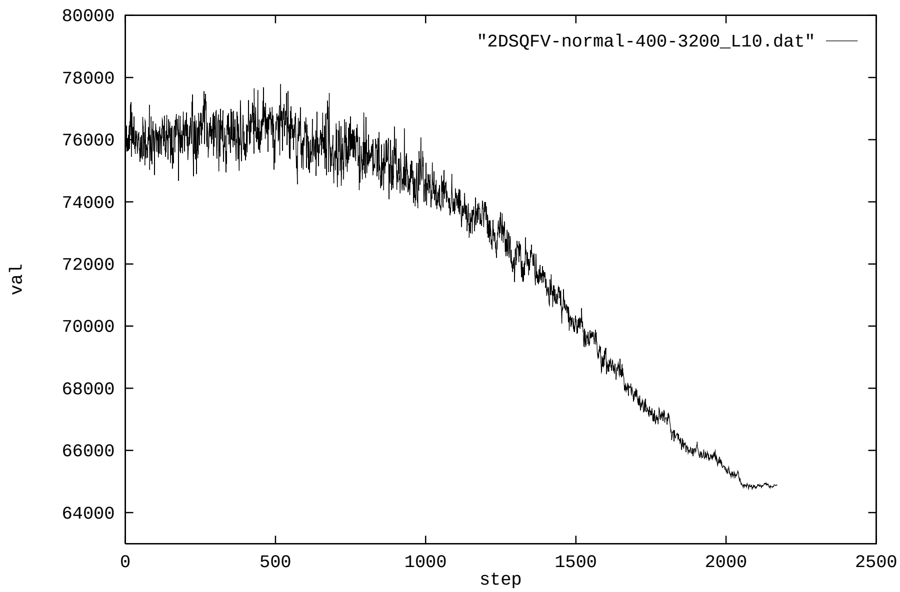

図 4.5: 400-3200 2DSQFVtype1

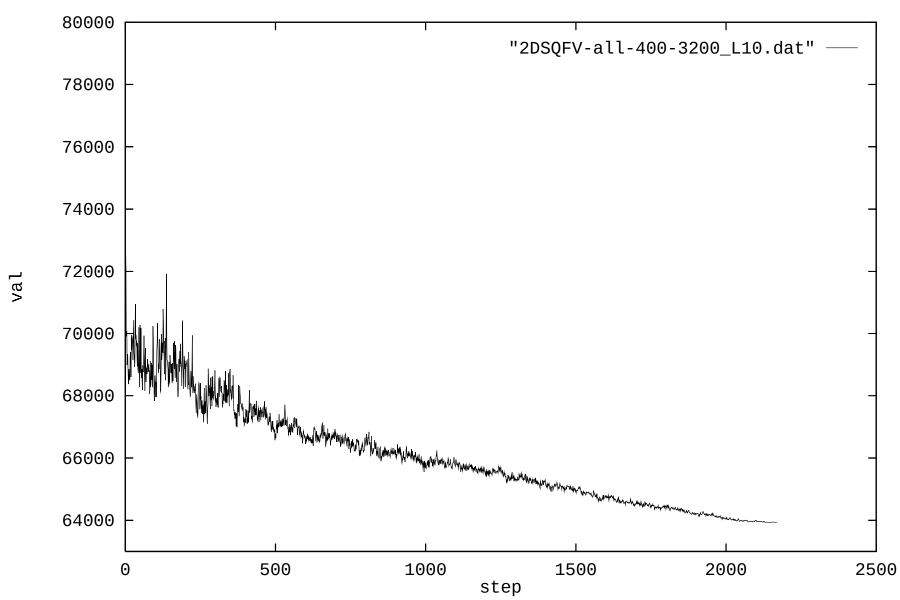

図 4.6: 400-3200 2DSQFVtype2

4.3　問題点

2DSQFV 法は，1DSQ 法の見掛け上の枠組みを比較的忠実に受け継ぐものであるが，次元の違いからピッチが 2 であっても部分問題と原問題の評価に差異が残る問題がある．例えば，図 4.7 はピッチ 2 における再配置前後を示したものであるが，ネットの #3 の評価は 1 良くなっているのに対し，ネットの #1 と #2 の評価はそれぞれ 1 悪くなっており，全体の評価は 4 から 5 へ悪化している．

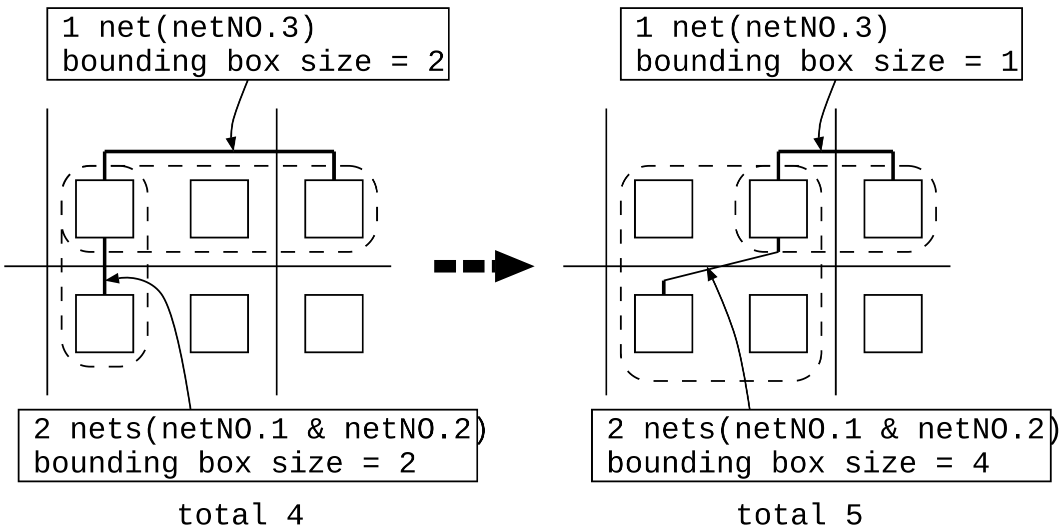

図 4.7: pitch2 において評価が悪くなる例

また，16 コンポーネント，32 ネットの時の実験結果を図 4.8 と図 4.9 に示す．

このように規模の小さいデータで実験を行うことにより，アルゴリズムの収束性がわかる．ここで，type1 の方が途中で良い値をとることがあるが，問題の規模が大きくなるに従い収束の悪さが問題全体の解を悪くしてしまう．これは，ある再配置を行った際に現在の解の性質を残しにくくしてしまうからである．

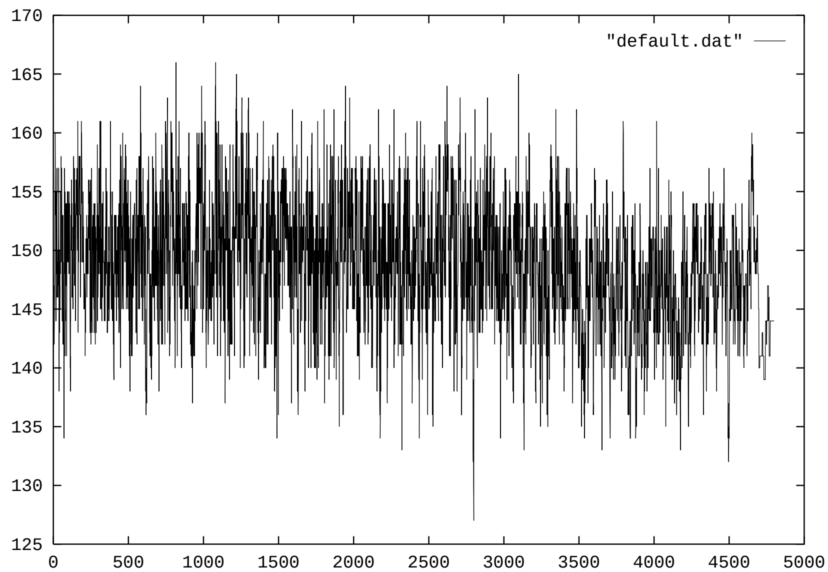

図 4.8: 16-32 2DSQFVtype1

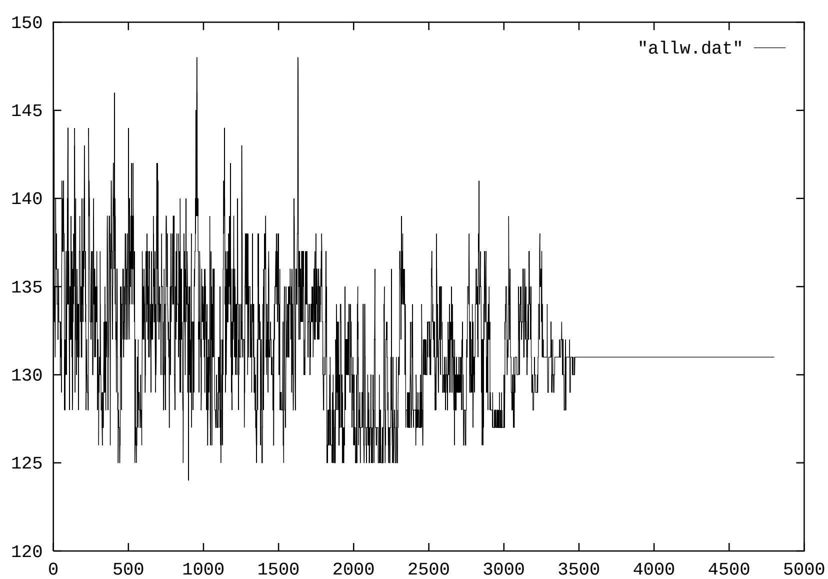

図 4.9: 16-32 2DSQFVtype2

第 5 章

2DSQSC 法

一つのサブグループ内に閉じたネットに関するネット長評価を無視する特徴を保持して，部分問題と原問題の評価の差を小さくすることが望ましいとの予想の下に，バウンディングボックスの評価値に応じたコストに基づいて再配置を行なう 2DSQSC 法を提案する．

5.1 アルゴリズム

アルゴリズムは，以下の 1,2,3 をピッチがスロット幅から 1 になるまで減少させつつ繰り返し実行し，２次元配置を出力する．

1. カットラインとサブグループ生成

   2DSQFV 法と同じく，ある配置に対して平面上に一定のピッチで表されるカットラインを縦横に引き，それらのカットラインで囲まれるコンポーネントとスロットをそれぞれ一つのサブグループとする．

2. コストの設定

   v_i の再配置を行なう際のコストを次のように設定する．

$$StepCost(v_i, x, y)$$
$$= \alpha \sum_{\substack{e \in E_H \\ v_i \in e}} \left[HP(e \backslash subgroup) - HP\left(e \backslash subgroup \cup \left\{ v_i \Big|_{\substack{x(v_i)=x \\ y(v_i)=y}} \right\} \right) \right] - stb(i, x, y)$$

   ここで，$subgroup$ は再配置を行なうサブグループに含まれるコンポーネントの集合を表し，$e \backslash subgroup$ はネット e から $subgroup$ の要素を除いた残りのコンポーネン

トの集合を表している．上記のコスト付においては stb を無視すると，$e \backslash subgroup \cup \left\{ v_i \middle| _{\substack{x(v_i)=x \\ y(v_i)=y}} \right\}$ が構成するバウンディングボックスと対象サブグループ領域の重なり方から数種類に分類できる．

まず，x 軸方向のみに着目した場合に $e \backslash subgroup$ の存在可能な場所は以下のように場合分けできる．

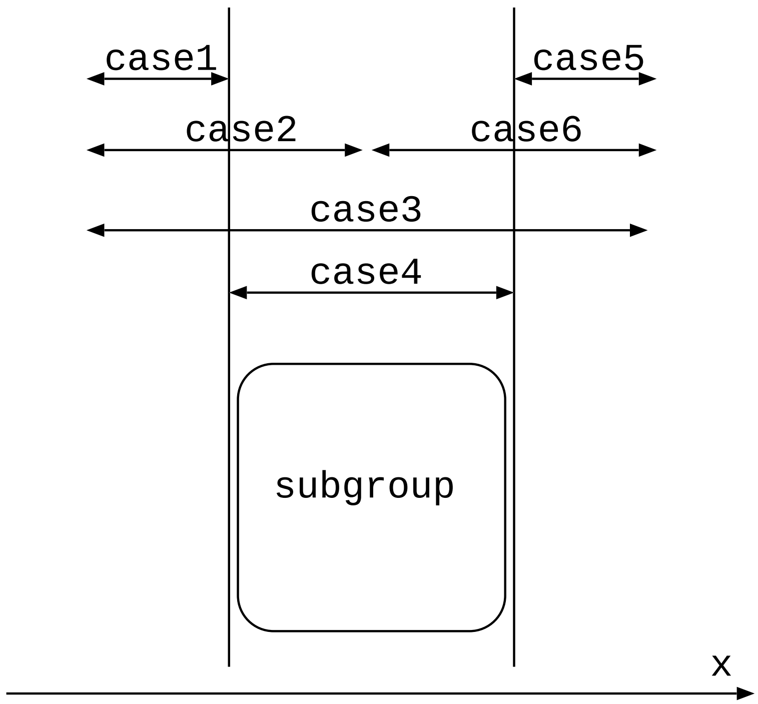

図 5.1: $e \backslash subgroup$ を x 軸方向から着目した場合

case1: $e \backslash subgroup$ がサブグループの左側のカットラインより左側に存在している場合．

case2: $e \backslash subgroup$ がサブグループの左側のカットラインのみに跨って存在している場合．

case3: $e \backslash subgroup$ がサブグループの左側のカットラインとサブグループの右側のカットラインの両方に跨って存在している場合．

case4: $e \backslash subgroup$ がサブグループの左側のカットラインとサブグループの右側のカットラインの間に存在している場合．

case5: $e\backslash subgroup$ がサブグループの右側のカットラインより右側に存在している場合．(**case1** と対象)

case6: $e\backslash subgroup$ がサブグループの右側のカットラインのみに跨って存在している場合．(**case2** と対象)

x 軸方向と同じく，y 軸方向においても６通りに分類可能である．しかし，図 5.2 のように x 軸が **case2**，y 軸が **case4** である組合わせや，図 5.3 のように x 軸が **case4**，y 軸が **case4** である組合わせは存在せず，このような場合が５通りありことから，実際に起りえる組合わせは３１通りとなる．

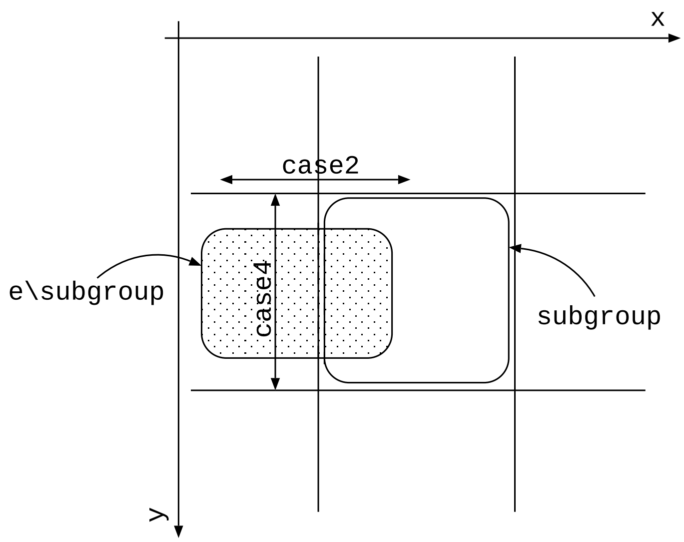

図 5.2: x:case2 y:case4

また，図 5.4 のように，x 軸が **case3**，y 軸が **case3** である場合にはコストはつかなくなる．

図 5.5 は，x 軸が **case1**，y 軸が **case1** の場合，図 5.6 は，x 軸が **case4**，y 軸が **case6** の場合，図 5.7 は，x 軸が **case3**，y 軸が **case4** の場合の例を表している．

3. 再配置

   再配置は 2DSQFV 法と同じく，コンポーネントの集合 V とスロットの集合 S に対し，$V \cup S$ を頂点集合とする重みつき完全 2 部グラフの最大コスト完全マッチングを求める問題として解く．

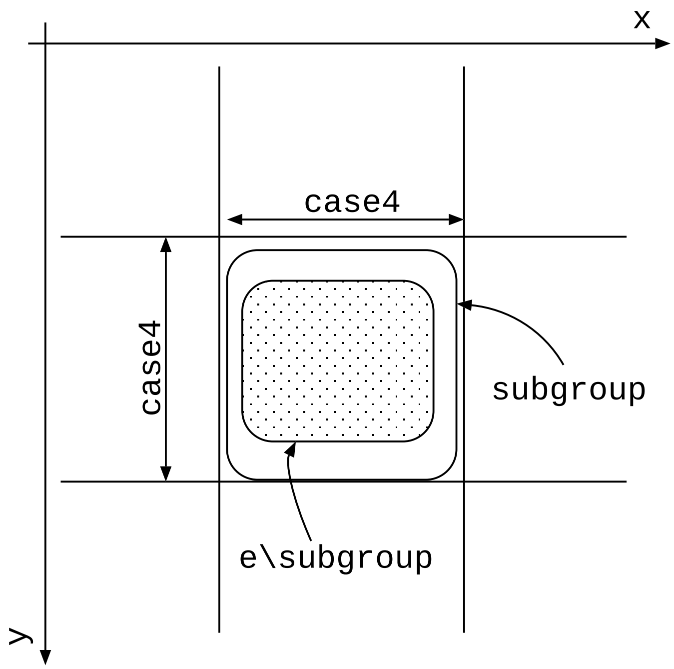

図 5.3: x:case4 y:case4

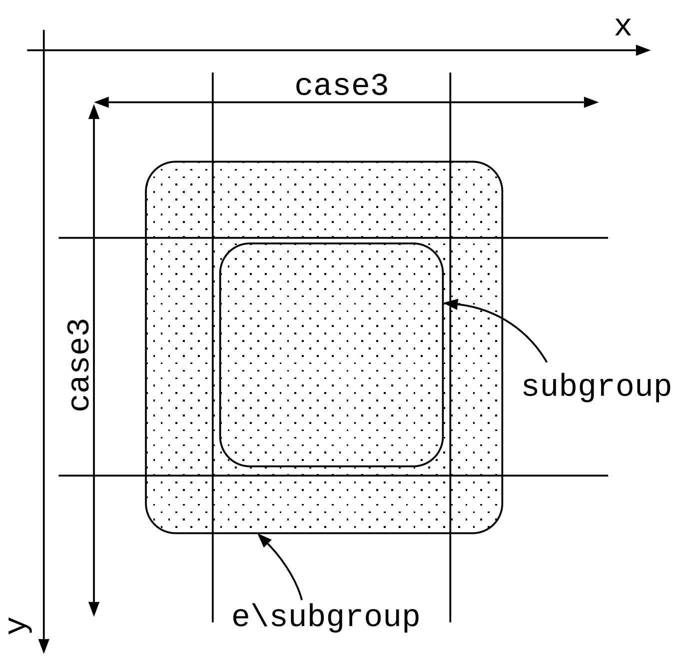

図 5.4: x:case3 y:case3

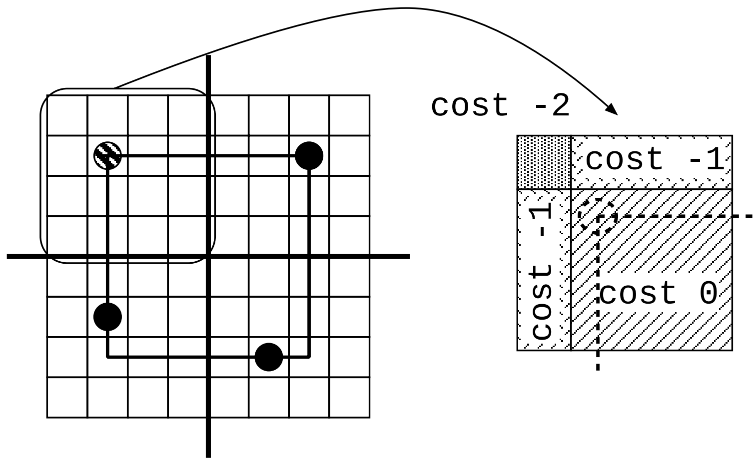

図 5.5: x:case1 y:case1

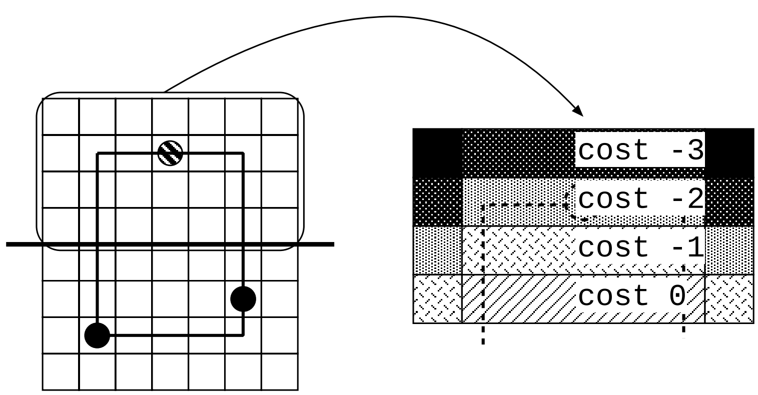

図 5.6: x:case4 y:case6

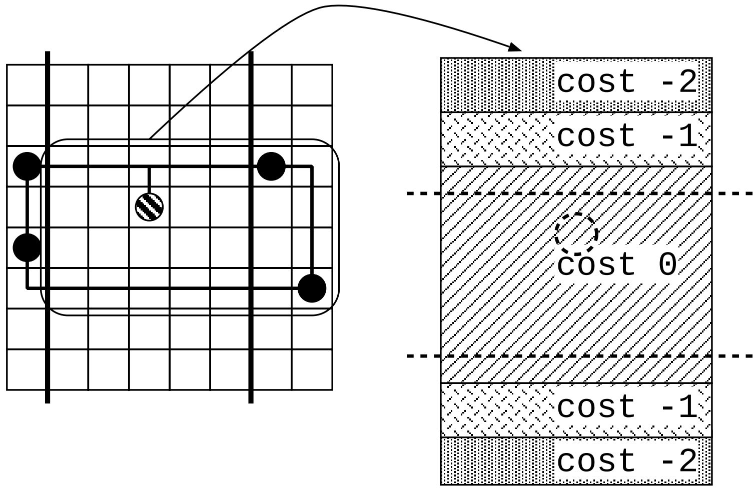

図 5.7: x:case3 y:case4

5.2 実験結果

5.2.1 2DSQSC 法と 2DSQFV 法との比較

2DSQSC 法と 2DSQFV 法の比較実験を行った．実験は，コンポーネント数 16，ネット数 32 を持つハイパーグラフに対して行った．実験の試行回数は 10 回であり，このときの配線長の平均と分散，実行時間の平均を表 5.1 に示す．

表 5.1: 2DSQFV,2DSQSC result

		2DSQFVtype1	2DSQFVtype2	2DSQSC
16-32	ave	149.4	126.9	124.7
	var	41.16	5.43	2.46
	time	11s	21s	22s

この結果，2DSQFV 法に対して，2DSQSC 法がより良い解を出すことが確認された．これは，ネットのバウンディングボックスの評価値に応じたコスト付を行なうことにより，2DSQFV 法と比較して部分問題と原問題の差が小さくなっているためと考えられる．

5.2.2　2DSQSC 法と SA 法との比較

次に，SA 法と 2DSQSC 法との比較実験を MCNC ベンチマーク ami33 及び人工的に作成した path36，rand36 を用いて行なった．ami33 はコンポーネントの大きさと端子位置を無視して使用している．path36 は 36 個のコンポーネントからなるパスである．rand36 はコンポーネント数 36 個，2 端子から 9 端子までのネットをそれぞれ 4 つづつ，計 32 のネットをもつデータである．各データにおける次数毎のネット数を表 5.2 に示す.

表 5.2: Circuit Structure

ami33		path36		rand36	
2pins	67	2pins	35	2pins	4
3pins	11			3pins	4
4pins	1			4pins	4
7pins	1			5pins	4
				6pins	4
				7pins	4
				8pins	4
				9pins	4

ami33 について 10 回の試行を行なった時の $COST_{total}$ と，その平均と分散，実行時間 (10 試行の平均) を表 5.3 に示す.

表 5.3: ami33 2DSQSC,SA

	2DSQSC		SA
L10	114	L10	110
	117		113
	113		111
	119		108
	113		112
	122		113
	117		114
	113		111
	110		114
	116		107
best	110		107
worst	122		114
average	115.4		111.3
variance	12.3		5.8
time	3.3s		4.02s
L100	106	L100	108
	111		110
	107		107
	112		108
	111		109
	115		108
	117		109
	112		106
	116		107
	113		110
best	106		106
worst	117		110
average	112.0		108.2
variance	12.7		1.7
time	34.8s		39.5s

次に，ami33を用いた実験において，2DSQSC法とSA法それぞれにおいて最良評価値と最悪評価値を出した解における次数毎の平均ネット長とその分散を図5.4に示す．次数が高いネットと比較して次数が低いネットに対するネット長が悪くなっていることが確認できる．

表 5.4: ami33:net box size

	2DSQSC		SA	
best	106		106	
	ave	var	ave	var
2pins	1.134328	0.148349	1.134328	0.118046
3pins	2	0	2	0
4pins	2	-	2	-
7pins	6	-	6	-
worst	117		110	
	ave	var	ave	var
2pins	1.238806	0.214835	1.164179	0.169607
3pins	2.181818	0.163636	2.090909	0.090909
4pins	3	-	2	-
7pins	7	-	7	-

path36について10回の試行を行なった時の$COST_{total}$と，その平均と分散，実行時間(10試行の平均)を表5.5に示す．2DSQSC法はSA法と比較して実行時間が短く，良い値を出していることが解る．比較的構造の単純なデータに対しては良い性能を示すのかも知れない．

表 5.5: path36 2DSQSC,SA

	2DSQSC		SA
L10	39	L10	39
	37		39
	38		35
	36		38
	37		39
	37		39
	36		37
	40		37
	37		37
	38		41
best	36		35
worst	40		41
average	37.5		38.1
variance	12.3		5.8
time	2.0s		2.2s
L100	35	L100	36
	37		39
	37		35
	36		36
	37		36
	36		37
	35		37
	36		36
	37		35
	36		36
best	35		35
worst	37		39
average	36.2		36.3
variance	0.6		1.3
time	19.6s		22.4s

rand36 について 10 回の試行を行なった時の $COST_{total}$ と，その平均と分散，実行時間 (10 試行の平均) を表 5.6 に示す．この実験では，端子数毎の収束の状態を調べる．

表 5.6: rand36 2DSQSC,SA

	2DSQSC normal		SA
L100	161	L100	157
	163		158
	163		158
	163		157
	161		157
	164		158
	161		157
	163		159
	165		159
	162		157
best	161		157
worst	165		159
average	162.6		157.7
variance	1.8		0.7
time	66.1s		79.5s

また，この実験の結果の次数毎の 2DSQ 法の結果を図 5.8 に，SA 法の結果を図 5.9 に示す．また，端子毎の平均をとったものを図 5.10 に示す．図 5.8 と図 5.9 では 2DSQSC 法の次数の低いネットの評価が大きくなっているのが解る．

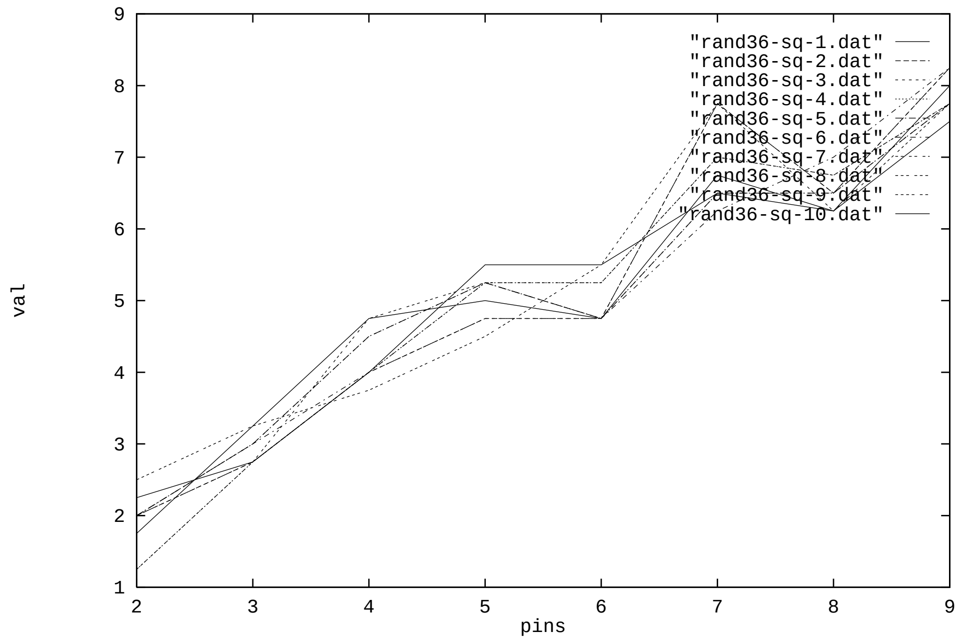

図 5.8: rand36-2DSQ

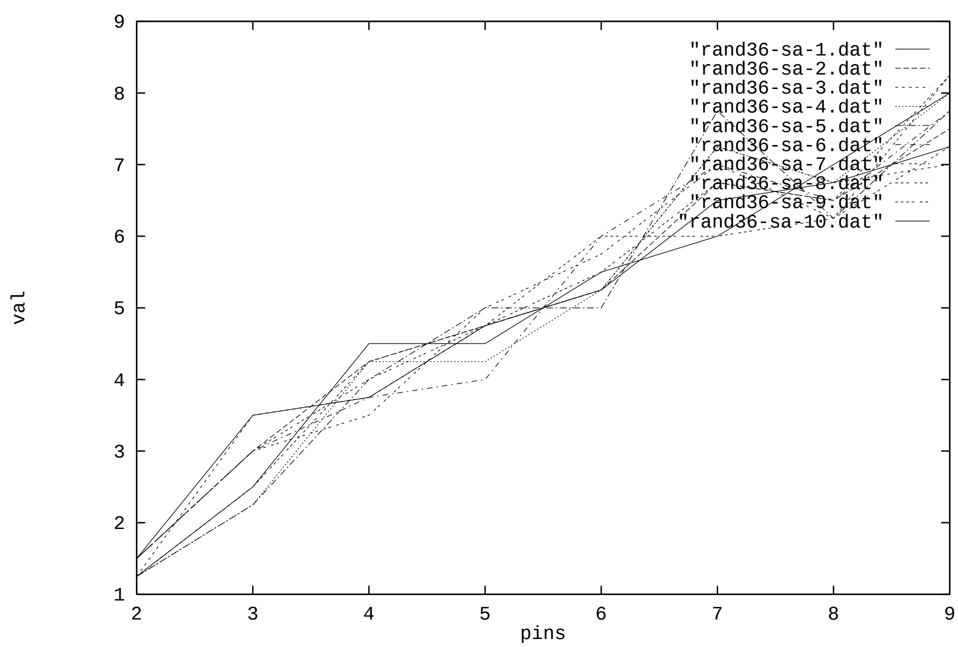

図 5.9: rand36-SA

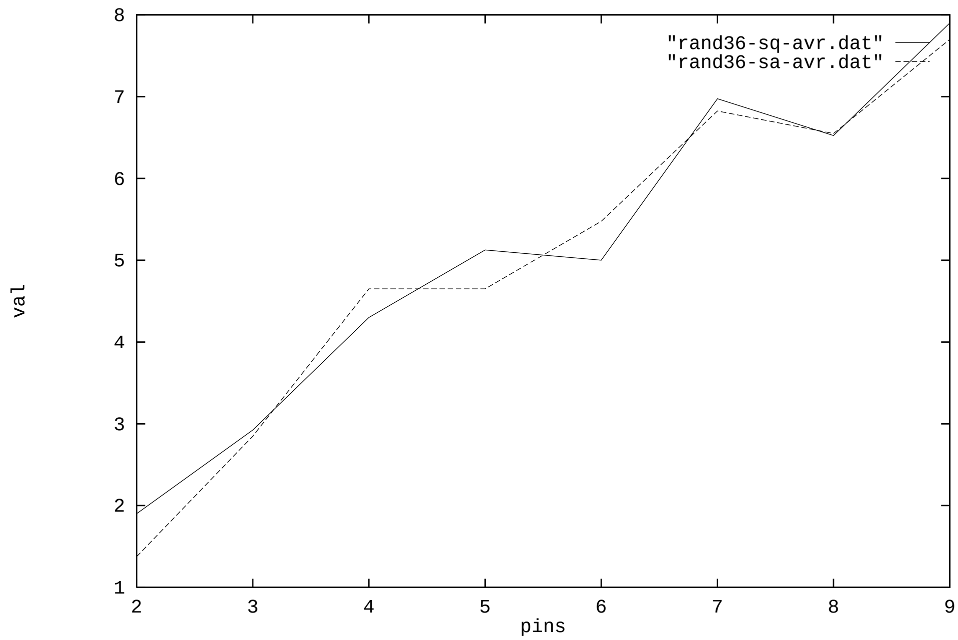

図 5.10: rand36-average/pins

5.3 ピッチの開始値

2DSQSC 法は再配置のアルゴリズムに Hungarian method を使用している．このアルゴリズムの計算量は $O(n^3)$ であるので，コンポーネントの個数が多いサブグループの再配置に要する演算時間がアルゴリズム全体の計算時間を大幅に伸ばしている．そこで，ピッチ幅の開始値をスロットの横幅の半分とし，１つのサブグループが含むコンポーネントの個数を減らし，計算時間の短縮を目指した．実験結果を表 5.7，表 5.8 に示す．(表において，ピッチ幅をスロットの横幅から開始したものを normal，半分としたものを div2 としている．) 実験の結果から，評価値は若干悪くなっているものの，約３倍の高速化を確認できた．

表 5.7: ami33 2DSQSC,pitch

	2DSQSC normal	2DSQSC div2
L10	114	115
	117	125
	113	114
	119	117
	113	121
	122	113
	117	111
	113	124
	110	116
	116	118
best	110	111
worst	122	125
average	115.4	117.4
variance	12.3	21.6
time	3.3s	1.1s
L100	106	113
	111	116
	107	114
	112	112
	111	116
	115	114
	117	109
	112	108
	116	115
	113	110
best	106	108
worst	117	116
average	112.0	112.7
variance	12.7	8.2
time	34.8s	11.0s

表 5.8: path36 2DSQSC,pitch

	2DSQSC normal	2DSQSC div2
L10	39	37
	37	37
	38	38
	36	39
	37	40
	37	38
	36	37
	40	38
	37	39
	38	39
best	36	37
worst	40	40
average	37.5	38.2
variance	12.3	21.6
time	2.0s	0.7s
L100	35	37
	37	37
	37	37
	36	36
	37	35
	36	38
	35	38
	36	36
	37	38
	36	36
best	35	35
worst	37	38
average	36.2	36.8
variance	0.6	1.1
time	19.6s	6.8s

5.4 スタビリティーの変更

2DSQSC 法で採用したスタビリティーはコストが等しい配置可能なスロットが複数存在したときに，現在の配置位置から近い方を選択するものであり，コンポーネント同士の位置関係の保持に直接働くものではない．これに対して，ここで提案するスタビリティーはコンポーネント同士の位置関係を保とうとするものである．2DSQFVβ 法においてこのスタビリティーを設定することで，解が向上した経験により，2DSQSC 法においても同様の効果が現れることを期待して導入してみる．コンポーネント $v_i(x,y)$ をスロット $s_j(x,y)$ に配置するときのスタビリティーを，

$$stb(i,x,y) = v_i(x) \times s_j(x) + v_i(y) \times s_j(y)$$

とする．実験結果は次のようになる．結果を見ると平均値の変化は少ないが，分散が悪くなっている場合があり，最終結果の評価値にばらつきがあることが解る．

表 5.9: スタビリティー変更後の実験結果

ami33	L10	L100	path36	L10	L100	rand36	L10	L100
	115	114		40	37		160	162
	114	109		36	36		164	161
	115	118		36	36		165	162
	114	115		37	38		167	159
	113	115		35	35		165	161
	113	113		37	37		164	164
	116	113		38	36		165	167
	116	112		37	38		165	168
	119	112		36	36		162	161
	116	117		40	35		162	161
best	113	109	best	35	35	best	160	159
worst	119	117	worst	40	38	worst	167	168
ave	115.1	113.8	ave	37.2	36.4	ave	163.9	162.6
var	3.2	6.8	var	2.8	1.2	var	4.1	8.3
time	4.0s	39.6s	time	2.9s	28.3s	time	7.4s	74.3s

第 6 章

まとめ

6.1　結論

本研究では，１次元配置問題の最適化手法である SQ を基として，２次元配置問題の最適化手法を提案した．まず，簡易的に拡張した 2DSQFVβ 法を提案し，実験により拡張の可能性を確認した．次に，性能を向上させた 2DSQFV 法を提案した．最後に，部分問題の構成方法をより原問題へと近づけた 2DSQSC 法を提案した．以上の手法を比較実験した結果，2DSQSC 法は 2DSQFV 法に対して，仮想配線長総和がより小さい解をより安定的に生成することが確認された．これは，ネットのバウンディングボックスの評価値に応じたコスト付を行なうことにより，2DSQFV 法と比較して部分問題と原問題の差を小さくした効果と考えられる．また，2DSQSC 法は SA 法に対して $0 \sim 3.7\%$ 大きい評価を示すことが確認された．以上のことから，１次元 SQ 法を２次元配置問題へ適用可能だと言うことを確認できた．

6.2　今後の課題

今後の課題として，再配置を行なうアルゴリズムの高速化があげられる．現在，部分再配置問題は最大コスト完全マッチングの問題に帰着させて，Hungarian method を用いて解いている．このアルゴリズムの計算量は $O(n^3)$ であり，提案手法の計算時間を長くしている．再配置により高速なアルゴリズムを導入し，解の質を落とさずに高速化を図る必要があると考えられる．また，より良い解を導出できる部分問題の構成方法を考案する必要がある．アルゴリズムの終了時に部分問題と原問題の差ができるだけ小さくなるように，部分問題を構成できるようになれば，より良い解を導出できる可能性がある．

謝辞

本研究を進めるにあたり，終始適切なご助言と暖かいご指導を下さった北陸先端科学技術大学院大学 金子峰雄助教授，同 田湯智助手，同 高島康裕助手，マイクロアーク株式会社 村田洋氏，同 佐藤真司氏，そして，研究室の学生の皆さんに深く感謝します.

参考文献

[1] Shinji Sato, ”Simulated Quenching:a New Placement Method for Module Generation”, *Proc. ICCAD* , 1997, pp. 538–541.

[2] 平間孝廉 高島康裕 金子峰雄, ”Simulated Quenching 法の 2 次元配置問題への拡張”, 電気関係学会北陸支部連合大会, 2000, p.105.

[3] Christos H. Papadimitriou, Kenneth Striglitz, ”Combinatorial Optimization Algorithm and Complexity”, Englewood Cliffs, NJ:Prentice-Hall, Inc., 1982.

[4] Dieter Jungnickel, ”Graphs,Networks and Algorithms”, Springer-Verlag Berlin Heidelberg, 1999.

[5] Thomas Lengauer, ”Combinatorial Algorithms for Integrated Circuit Layout”, Baffins Lane, Chichester, England:John Wiley & Sons Ltd, 1990.