Vol.2018-SLDM-185 No.13
Vol.2018-EMB-49 No.13
2018/12/6

集合対間配線問題ソルバと引きはがし再配線のADC2018問題への適用

大和田 真由[1,a)]　和田 邦彦[1]　赤木 佳乃[2]　佐藤 真平[2]　高橋 篤司[2]

概要：多層ナンバーリンク問題である ADC2018 問題の解を求めるアルゴリズムを提案する．集合対間配線問題を解く既存手法を用いて，異なる端子との配線を許した初期解を求める．続いて 2 種類の引きはがし再配線を行い，正しい端子と配線するように初期解を修正する．計算機実験により提案手法が ADC2018 問題の解を正しく出力することおよび引きはがし再配線を 2 種類用いたことの効果を確認した．

1. はじめに

ナンバーリンク [1] はマス目状の盤面において，同じ数字同士を結ぶ互いに交差しない線を求めるペンシルパズルであり，配線問題と親和性が高い [2]．また NP 完全であることが証明されている [3]．先行研究では充足可能性問題 [4]，遺伝的アルゴリズム [5]，ZDD 法 [6]，CHORD-LAST 法 [7] を利用する手法などが提案されている．ADC2018 問題はナンバーリンクを多層に拡張した問題である．

本稿では ADC2018 問題を解くアルゴリズム「とりあえずつないではんせいするほうほう（以下 T2H2）」を提案する．T2H2 では集合対間配線ソルバを用いて「とりあえずつなぐ」ことで異なる数字をつなぐ配線を許容し初期解を求める．初期解の修正に引きはがし再配線を用いて「はんせい」を繰り返し行い，正しい解を効率良く求める．

集合対間配線問題 [8] とはソース端子集合とシンク端子集合が与えられ，ソース端子とシンク端子を接続する配線を求める問題である．集合対間配線ソルバを用いることで全ての数字が何かしらの数字とつながった初期解が求まる．この初期解には部分的に正しい配線が含まれる可能性が高いことが予想される．初期解で正しい配線を部分的にでも含むことで，他の数字をつなぐ配線を避けた配線をその後の修正において求めやすい，つまり少ない修正回数で正しい解に近づける．これにより，短い実行時間で正しい解を求めることを目指す．

修正には 2 種類の引きはがし再配線を順に用いる．初めに引きはがし再配線 1（以下 h1），続いて引きはがし再配線 2（以下 h2）を行う．h1 では，初期解に異なる数字をつなぐ配線が含まれているとき，同じ数字をつなぐように交差なくつなぎ直す処理を行う．h1 で未配線の数字が残ることがあり，h2 では未配線の数字全ての配線を求める．

h1 では，すでにある配線との重なりを許さずに経路を探索し，同じ数字同士をつなげる．この場合，すでにある配線が障害となりつなぐことができない数字ができる可能性がある．一度同じ数字同士でつないだ配線を引きはがすことはせずに，経路の探索は初期解で異なる数字をつないでいた配線の数だけ行う．

h2 では，すでにある配線との重なりを許して経路を探索する．配線が重なった時には元々あった配線を引きはがして新しく見つけた経路で数字をつなぐ．このようにして全ての数字をつなぐ配線を求める．この方法では同じ数字同士をつなぐ配線を引きはがすことになるので，経路を探索する回数が h2 開始時の未配線数より多くなることがある．

h1 によりできる限り多くの数字をつなぐ配線を求めることで，経路を探索する回数の多い h2 を行う回数をできる限り減らす．これにより少ない経路探索回数で全ての数字をつなぐ配線を求める．短い実行時間で正しい解を出力することを目指している．

実験では T2H2 を，h2 を行わなかった場合と実行時間を比較し，引きはがし再配線を 2 種類用いたことによる効果を確認する．

2. ADC2018 問題

ADC2018 問題は多層ナンバーリンクである．ナンバーリンクとは各数字が 2 つずつ配置されたマス目状の盤面において，重なりや交差，分岐を禁止して同じ数字同士を結ぶ線を求める問題である．ADC2018 問題はこれを多層に

[1] 東京工業大学　工学部　情報工学科
[2] 東京工業大学　工学院　情報通信系
[a)] owada@eda.ict.e.titech.ac.jp

「ナンバーリンク」は株式会社ニコリの登録商標です．

Vol.2018-SLDM-185 No.13
Vol.2018-EMB-49 No.13
2018/12/6

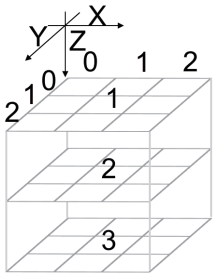


図 1 3 × 3 × 3 の盤面

```
SIZE 3X3X2
LINE_NUM 3

LINE#1 (0,0,1) (2,2,1)
LINE#2 (0,2,1) (2,0,1)
LINE#3 (2,1,1)-(0,0,2)
```

(a) 入力

```
SIZE 3X3X2
LAYER 1
1,2,2
1,2,3
2,2,1
LAYER 2
3,3,3
1,0,3
1,1,1
```

(b) 出力

図 2 ADC2018 問題の入出力例

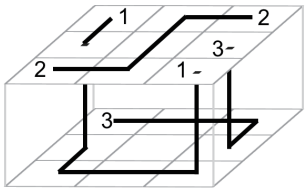


図 3 図 2 の出力の配線の様子

拡張することで 3 次元 IC 接続に対応する問題である [2].

図 1 に 3 × 3 × 3 の盤面を示す．各マスは (X,Y,Z) の座標で表され，左上奥の原点の座標は (0,0,1) とする．線は X，Y，Z 各方向に引くことができ斜め方向に引くことはできない．**図 2** に ADC2018 問題の入出力例を示す．

入力では同じ数字を 2 つの端点とするラインが定義されており，盤面サイズ，ライン数，各ラインの ID および 2 つの端点の座標が与えられる．出力では，全てのマスについて，そのマスを使用している配線があればその ID，なければ 0 として出力する．なお最大盤面サイズは 72 × 72 × 8 でライン数に上限はない．**図 3** に図 2 の出力を実際に盤面で表した図を記す．

また以下の式で表される解の品質が導入されており，この値が大きいとより良い解となる．

$$\frac{1}{(\text{総配線長}) + (\text{曲がった回数}) + 1/3 \times (\text{隣接配線境界数})} \quad (1)$$

つまり，配線長および折れ曲がりが少なく，他の配線と隣接しない配線が良い解となる．なお配線長は配線が通ったマスの数で測る．

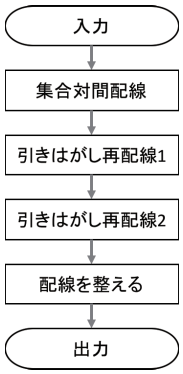
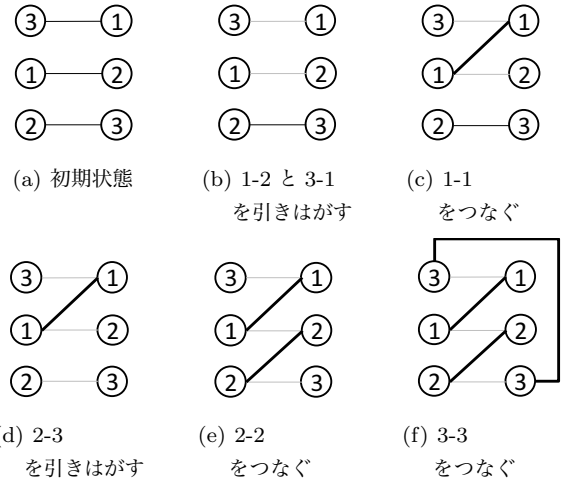


図 4 T2H2 全体の流れ



(a) 初期状態　(b) 1-2 と 3-1 を引きはがす　(c) 1-1 をつなぐ

(d) 2-3 を引きはがす　(e) 2-2 をつなぐ　(f) 3-3 をつなぐ

図 5 h1 で行う操作のイメージ

3. とりあえずつないではんせいするほうほう

3.1 概要

とりあえずつないではんせいするほうほう（T2H2）では各マスを点とし隣接マスを辺でつないだ配線グラフを用いる．全体のステップの流れを**図 4** に示す．

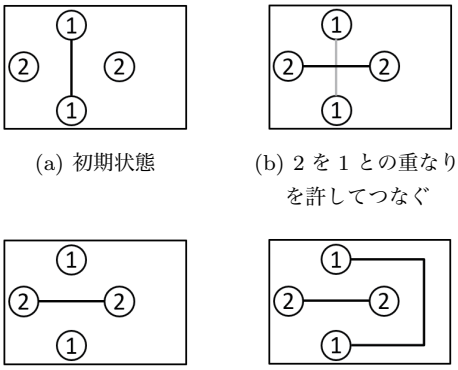
集合対間配線により初期解を求め，それを h1，h2 により修正する．最後に全ての配線を配線長最小になるように整えた解を出力する．

初めに集合対間配線問題を解く既存手法 [9] により異なるラインの端点とのつながりを許容した初期解を求める．次にこの初期解を 2 種類の引きはがし再配線 h1，h2 を順に用いて正しい端点とつなぐように修正する．

図 5 と**図 6** に h1，h2 で行う操作のイメージ図を示す．

h1 では異なるラインの端点をつなぐ配線を含む解を入力とし，ラインを正しくつなぐ配線のみを持ち，未配線のラインを含む可能性のある解を出力する．異なるラインの端点をつなぐ配線の集合（図 5(a)）ごとに正しい端点へとつなぎ直す操作を繰り返し行う．初期解で正しくつなげていない全てのラインを，他の配線と重なりのない経路があればつなぎ直す．

h2 では未配線のラインを含む解を入力とし，全てのライ



(a) 初期状態　(b) 2 を 1 との重なりを許してつなぐ

(c) 1 を引きはがす　(d) 1 をつなぎ直す

図 6　h2 で行う操作のイメージ

ンを重なりなくつなぐ配線を求めることを目指す．すでにある配線との重なりを許して未配線のラインをつなぎ，重なった時は元々あった配線を引きはがす操作を繰り返す．

h1 はその入力における異なるラインの端点をつなぐ配線の数だけ経路の探索を行う．h2 はその入力における未配線のライン数よりも多く経路の探索を行う可能性がある．h2 で多くの配線をつなぐと経路探索回数が多くなり実行時間が長くなることが予想される．初めに h1 を行い，h2 開始時点での未配線のライン数を減らすことで，少ない経路探索回数で全てのラインをつなぐ．

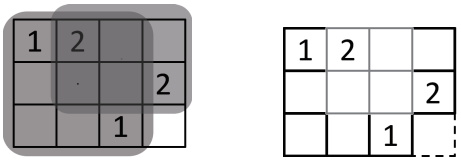
引きはがし再配線により求まる解は迂回する配線が含まれる可能性がある．そこで全ての配線に対して，配線を 1 つずつ引きはがしその他の配線との重なりを許さずに配線長最小でつなぎ直す操作を行い，総配線長を短くする．

3.2　初期解生成（t）

集合対間配線問題ソルバ [9] を用いて初期解を生成する．全ての端点が異なるラインの端点を含む何かしらの端点とつながる初期解を生成する．ただしソース端子集合の端点同士，シンク端子集合の端点同士をつなぐことはない．

既存手法 [9] は，目標端子対集合を設定しこれらをできる限り接続することを目指した配線を行う．以下の操作を繰り返して全ての端子をつなげる配線を求める．入力を全てのラインをつなぐ解が存在する ADC2018 問題であることを前提としており，配線は全ての端子に対して必ず求まる．

全てのソース端子，シンク端子とそれぞれ辺でつながる大ソース，大シンクを配線グラフに加えたフローグラフを用いる．全ての点の容量を 1 として大ソースから大シンクへの単位フローを繰り返し探索することで配線を求める．フロー増加路の探索は他の目標端子対の配線が使用する可能性の高い領域を避けて行う．そのためにバウンディングボックスを設定している．バウンディングボックスとはラインの端点を囲う最小矩形のことである．配線グラフの各辺に対して、その辺を囲う未配線の目標端子対のバウンディングボックスの数を重なり度と定義する（**図 7**）．フロー増加路の探索は，初めは重なり度が 1 の辺のみで行う．フロー増加路が見つからなければ重なり度 2 以上の辺も加えて探索し，それでも見つからない時は全ての辺で探索する．以下ではこのステップを t とする．



(a) バウンディングボックス　(b) 重なり度
グレー:2
黒実線:1
黒点線:0

図 7　バウンディングボックスの重なり度

T2H2 では各ラインの 1 つ目の端点をソース端子，2 つ目の端点をシンク端子としてソース端子集合とシンク端子集合を与え、ラインの 2 つの端点を目標端子対として目標端子対集合を設定している．図 2 の入力を例にとるとソース端子集合={(0,0,1),(0,2,1),(2,1,1)}={S1,S2,S3}，シンク端子集合={(2,2,1),(2,0,1),(0,0,2)}={T1,T2,T3}，目標端子対集合={(S1-T1),(S2-T2),(S3-T3)} となる．

3.3　引きはがし再配線 1（h1）

h1 では t の出力を入力とし，ラインを正しくつなぐ配線のみを持ち未配線のラインを含む可能性がある解を出力する．h1 で行う操作を**図 8** にフローチャートで示す．異なるラインの端点をつなぐ全ての配線は図 5(a) のような配線の集合に含まれる．このような集合の配線は，一般性を失わずに 1-2，2-3，3-4，…，n-1 のように異なるラインの端点をつないでいると言える．h1 ではこの配線の集合ごとに正しい端点へとラインをつなぎ直す操作を行う．

ここで配線 A は異なるラインの端点をつなぐ配線の集合の最初の配線である．また配線 B は 2 本目以降の配線であり，配線 B がないというのはこの集合の配線が全て引きはがされている状態である（図 5(d)）．ライン x の両端点の配線を引きはがし，ライン x の両端点がどの端点ともつながっていない状態にして，ライン x をつなげるということを繰り返している．なおライン x を他の配線と重なりなくつなぐ経路がない場合には他のラインの端点とつなぐことはせずにつながないままにしておく．

他のラインの配線が使用する可能性の高いマスを避けるためバウンディングボックスを利用してコストを調節した．ソースがどの端点ともつながっていないラインのバウンディングボックスを設定し，つなごうとしているライン x 以外のどのラインのバウンディングボックスにも囲われない辺のコストを 1，それ以外を 2 として最小コストでライン x をつなぐようにしている．

図 9 にバウンディングボックスによるコスト設定の例

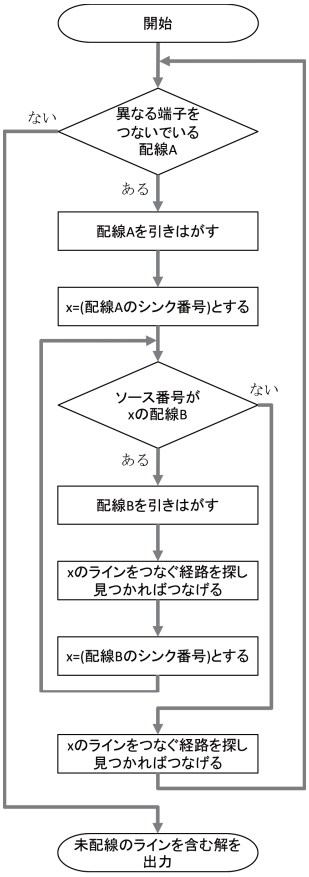
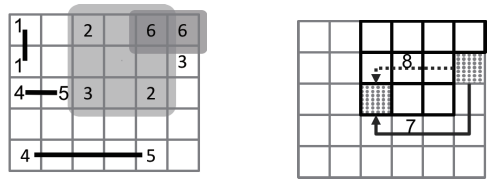


図 8 h1 のフローチャート



(a) 3 をつなぐ時のバウンディングボックス

(b) コスト
グレー:1
黒:2

図 9 h1 のコスト

を示す．この例では，点線で描かれた距離最小の経路ではなく実線で描かれたコスト最小の経路でつなげることにより，この後つなげる 2 の経路を避けることができている．

3.4 引きはがし再配線 2（h2）

h2 では未配線のラインを含む解を入力とし，全てのラインをつないだ解を出力することを目指している．h2 で行う操作を**図 10** にフローチャートで示す．h2 では以下の操作を繰り返し行うことで未配線のラインをなくすことを目指している．未配線のラインをすでにある配線との重なりを許してつなぎ，元々あった配線を引きはがす．引きはがした配線はすでにある配線との重なりを許さずに経路を探索し，経路があればつなぐ．なおループ回数が一定回数に達して終了した場合には全てのラインを正しくつなぐ配線

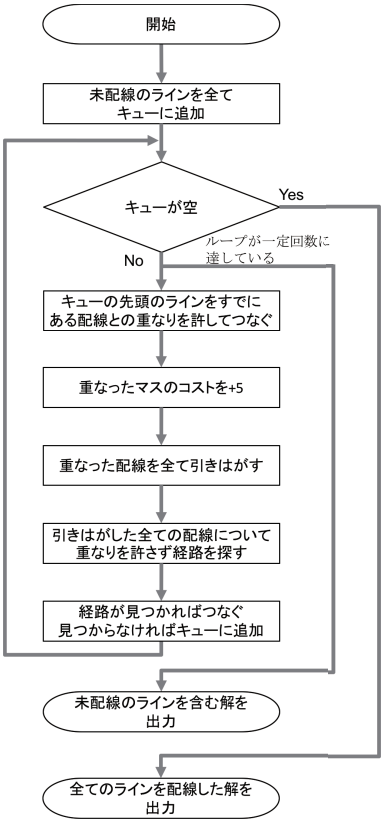


図 10 h2 のフローチャート

を見つけることができず正しい解が出力できない．

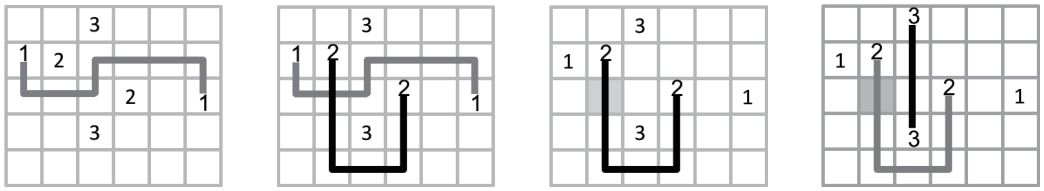
また h2 では未配線のラインを保持するキューを用いる．引きはがしたラインの経路を探索する際に，経路の見つからなかったラインはキューに追加する．これによりそのようなラインの優先度を下げ経路を見つかるラインからつなげる．

次に，マスにコストを設定し最小コストの経路を探索することで、他の配線が使用するマスをできる限り避けた経路でラインをつなぐ．重なりを許して経路を探す際には配線のあるマスのコストを 20 とし，配線のあるマスをできる限り避けて経路を探す．さらに配線が重なったマスはコストを 5 増やすことで，今後の配線において経路探索の優先度を下げる．なお全ての辺のコストは 1 としている．

図 11 に h2 の操作の様子を示す．グレーのマスは配線が重なりコストが 5 増やされていることを表す．(e) では重なったマスのコストを 5 増やしたことでライン 1 が元とは違う経路でつながる．

3.5 配線を整える（h3）

このステップでは全ての配線に対して，配線を 1 つずつ引きはがしその他の配線との重なりを許さずに配線長最小でつなぎ直す操作を行うことで総配線長を短くすることを目指す．同時に，配線が曲がる回数を減らすことも行う．全ての辺のコストを 1，マスのコストを 0 とし，さらに曲がり角でコストを 0.01 増やして最小コストで経路を求め

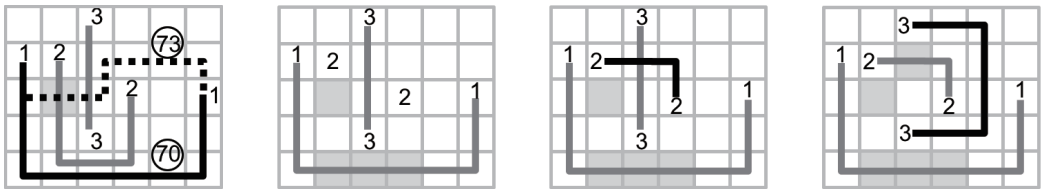


(a) 初期状態
キュー:2,3

(b) 2 をつなげる
(重なった配線:1)
キュー:2,3

(c) 1 を引きはがす
1 はつながらない
コストを上げる
キュー:3,1

(d) 3 をつなげる
(重なった配線:なし)
キュー:3,1



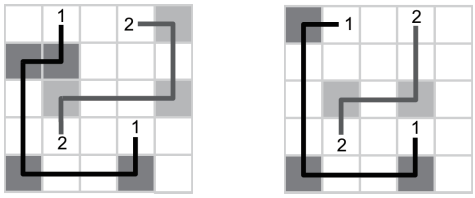
(e) 1 をつなげる
(重なった配線:2)
キュー:1

(f) 2 を引きはがす
2 はつながらない
コストを上げる
キュー:2

(g) 2 をつなげる
(重なった配線:3)
キュー:2

(h) 3 を引きはがす
3 をつなぎ直す
コストを上げる
キュー:空

図 11 h2 の操作の例



(a) h3 を行う前
総配線長:18
曲がった回数:6

(b) h3 を行った後
総配線長:16
曲がった回数:5

図 12 h3 の操作の例

る．以上より配線長，曲がった回数がともに少ない解が出力される．以下ではこのステップを h3 とする．**図 12** にこの操作の例を示す．総配線長および曲がった回数が減っている．

4. 実験

T2H2 の 2 種類の引きはがし再配線を用いたことの有効性を確かめるため，t の後に h1 を行わずに，異なる端点とつながっている配線を全て引きはがしてから h2，h3 を行った場合（以下 T2H）と比較する．

T2H2 の実装には C++を用いる．h1，h2，h3 において最短経路の探索は，ダイクストラ法を用いる．計算機環境は Ubuntu 16.04 LTS，Intel(R) Xeon(R) CPU E3-1240 v5 @ 3.50GHz，メモリ 32GB を用いた．コンパイラは gcc version 5.4.0 を使用し，最適化オプションには O2 を用いる．ベンチマークとして ADC2018 で使用された問題 29 問を用いる．

T2H2，T2H ともに ADC2018 で使用された 29 問について全てのラインを配線する正しい解を求めることができ，そのうち 19 問が T2H2 の方が実行時間が短いことが確認できた．

表 1 に 29 問のうち 6 問の実行結果を示す．各問題の上の段に T2H2，下の段に T2H の実行結果を示す．平均ライン長を各ラインの端点間のマンハッタン距離に 1 を足した値の平均，平均配線長を終了時の各配線長の平均と定義する．ここで平均ライン長に 1 を足したのは配線長を通ったマスの数で測っているためである．盤面サイズ，ライン数，平均ライン長，平均配線長，各ステップで正しく配線できたライン数，ループ回数，引きはがし回数，実行時間をそれぞれ SIZE，#LINE， ALL，AWL，#WIRE，#LOOP，#RipUp，T と表す．なお AWL の括弧内には ALL と AWL の比を示す．また#RipUp の括弧内には引きはがした後すでにある配線との重なりがない経路がなく，つなぐことができななかったライン数を示す．T 内の h12 とは T2H2 では h1，h2 の実行時間の合計，T2H では t 終了後，異なるラインの端点をつなぐ配線を引きはがす操作と h2 の実行時間の合計を表す．実行時間は 5 回の実験の平均をとる．

ライン数に対してループ回数の多い Q2，Q26 と，少ない Q6，Q15，Q16，Q17 の 2 つに分けて考察する．

Q2 や Q26 はライン数に対してループ回数が多いため，h1 終了時の配線の多くが大幅に変更されていることが予想される．つまり h1 で求められるような最短路を使わない配線が多く含まれる問題であると言える．このような問題は T2H2 により実行時間が短くなるとは限らず Q2 のように長くなってしまうこともあることがわかる．

ループ回数の少ない問題については，Q16 は T2H2 の方

問題	SIZE	#LINE	ALL	AWL		#WIRE			h2			T(s)	
						t	h1	h2	#LOOP	#RipUp		h12	total
Q2	50*50*1	320	5.86	6.76	(1.15)	97	163	60	1230	1402	(1170)	4.86	6.16
				6.72	(1.15)	97	-	223	1382	1408	(1159)	4.71	5.98
Q6	72*72*8	375	51.87	58.40	(1.13)	1	367	7	8	13	(1)	143.36	258.41
				56.20	(1.08)	1	-	374	374	32	(0)	978.33	1094.95
Q15	16*16*5	30	17.50	18.63	(1.06)	4	25	1	2	2	(1)	0.05	0.13
				18.50	(1.06)	4	-	26	27	8	(1)	0.11	0.19
Q16	72*72*4	1027	6.49	6.59	(1.02)	458	569	0	0	0	(0)	23.83	91.32
				6.59	(1.01)	458	-	569	569	0	(0)	18.21	85.27
Q17	72*72*8	99	25.56	25.60	(1.00)	18	81	0	0	0	(0)	12.08	29.33
				25.58	(1.00)	18	-	81	81	0	(0)	18.46	36.08
Q26	8*8*8	49	8.22	8.96	(1.09)	1	35	13	218	305	(205)	0.18	0.21
				8.92	(1.08)	1	-	48	251	299	(203)	0.20	0.23

表 1 実行結果（上:T2H2　下:T2H）

が実行時間が長く，その他は短くなっている．Q6，Q15，Q17 と比較すると Q16 は平均ライン長が短いことから，h2 の計算コストがあまり大きくなくループ回数を減らしたことの影響があまりなかったものと考えられる．

また 6 問すべてにおいて T2H の方が平均配線長が短くなっている．これは h1 でバウンディングボックスを用いてコストを調整したことにより迂回する配線が多く含まれたことが原因と考えられる．このような配線を配線長最小でつなぎ直すために h3 を行っているが，これは全てのラインに対して 1 回しか行っていない．配線長最小でつながっていない配線が障害となり迂回した経路が求まる可能性がある．このことにより迂回している配線を元々多く含んでいた T2H2 の方が最短経路でつないでいないラインが多くなったと考えられる．

以上より T2H2 は，引きはがし再配線 1 で求められるような最短路を使わない配線が多い問題に対しては実行時間が長くなる可能性があるが，それ以外の問題に対しては問題サイズや配線長が大きく引きはがし再配線 2 の計算コストが大きくなるものについては実行時間が短くなることが確認できた．

5. まとめ

多層ナンバーリンク問題である ADC2018 問題を解くアルゴリズムを提案した．提案手法は集合対間配線を解く既存手法 [9] を用いて初期解を求め，2 種類の引きはがし再配線を繰り返し行うことで初期解を修正する．計算機実験により T2H2 が正しい解を出力すること，引きはがし再配線を 2 種類用いたことによりサイズの大きな問題に対して実行時間が削減できることが確認できた．なお ADC2018 と同様の問題である ADC2017 問題の中には，正しく解を求めることができない問題もあり，その全てが全てのマスを配線に使用する問題であった．今後の課題として h2 における効率の良い終了条件の考察や，隣接配線境界数を減らすアルゴリズムの考察，h3 の実行回数の考察および全てのマスを配線に使用する問題に対して正しい解を求めるようなアルゴリズムの考察があげられる．

参考文献

[1] 株式会社ニコリ：ナンバーリンクの遊び方、ルール、解き方 — WEB ニコリ - Nikoli，株式会社ニコリ（オンライン），入手先 ⟨https://www.nikoli.co.jp/ja/puzzles/numberlink/⟩（参照 2018-11-03）．
[2] DA シンポジウム実行委員会：DA シンポジウム 2018 アルゴリズムデザインコンテスト，DA シンポジウム実行委員会（オンライン），入手先 ⟨https://dasadc.github.io/adc2018/⟩（参照 2018-11-01）．
[3] 古妻浩一，武永康彦：ナンバーリンクの NP 完全性と問題の列挙，電子情報通信学会技術研究報告. COMP, コンピュテーション， Vol. 109, No. 465, pp. 1–7 (2010).
[4] 迫　龍哉，川原征大，宋　剛秀，番原睦則，田村直之，鍋島英知：SAT 型制約ソルバーによるナンバーリンクの解法とその評価，2016 年度人工知能学会全国大会（第 30 回）(2016).
[5] 高澤一彰，山内ゆかり：遺伝的アルゴリズムを用いたナンバーリンクの解法，日本大学生産工学部第 42 回学術講演会，pp. 109–110（オンライン），入手先 ⟨http://www.cit.nihon-u.ac.jp/kouendata/No.42/7_sujo/7-032.pdf⟩ (2009).
[6] 吉仲　亮，岩下洋哲，川原　純，斎藤寿樹，鶴間浩二，湊　真一：DK-2-2 フロンティア法の種々のリンクパズル問題への応用 (DK-2. 第 3 回 ERATO 湊離散構造処理系シンポジウム-グラフ列挙索引化アルゴリズムの新展開-, ソサイエティ特別企画, ソサイエティ企画)，電子情報通信学会総合大会講演論文集，Vol. 2012, No. 2, pp. SS–5–SS–8 (2012).
[7] 田中雄一郎，高橋篤司：領域分割を用いた CHORD-LAST 法に基づくナンバーリンク解法，DA シンポジウム 2014 論文集，Vol. 2014, pp. 221 – 226 (2014).
[8] 高橋篤司：集合対間配線問題に関する一考察，電子情報通信学会技術研究報告，Vol. 111, No. 216, pp. 23–28 (2011).
[9] 赤木佳乃，佐藤真平，高橋篤司：目標端子対接続の実現を目指す集合対間配線アルゴリズム，第 30 回　回路とシステムワークショップ論文集，pp. 180–185 (2017).