ハイパフォーマンス
コンピューティング 54－8
（１９９４．１２．１４）

ソーティングネットワークに関する研究
— 区間減少ソート —

黒田久泰

京都大学 工学部 応用システム科学教室

本論文では，比較区間を減らしていく区間減少ソートというソーティングネットワークを提案し，0－1原理と最良優先探索を用いて，最も比較回数が小さくなる場合を計算機上で求める方法について述べる．それに伴い，０－１原理については効率よく検証する方法を提案し，最良優先探索については，効率のよいヒューリスティック評価関数を適応させた．その結果，区間減少ソートはバイトニックソートと大差のない比較回数が得られることがわかった．

A Study of Sorting networks
— Gap Decrease Sort —

Hisayasu Kuroda

Department of Applied Systems Science, Faculty of Engineering, Kyoto University

This paper proposes sorting networks for which the distance between compared elements (the gap) is reduced step by step what is called “Gap Decrease Sort” and explains how to seek the minimum number of key comparisons using zero-one principle and best-first search. An efficiency technique of zero-one principle and more informative heuristic function on best-first search are presented. As a result, Gap Decrease Sort is found to be equal to bitonic sorters with respect to the number of key comparisons.

1 はじめに

本論文では，区間減少ソートというソーティングネットワークを提案し，最適な区間減少ソートを求めるための手法について述べる．

2 区間減少ソート

2.1 区間減少ソートの提案

本論文では，図1のようなソーティングネットワークを提案する．

以後このようなソーティングネットワークを区間減少ソートと呼ぶことにする．

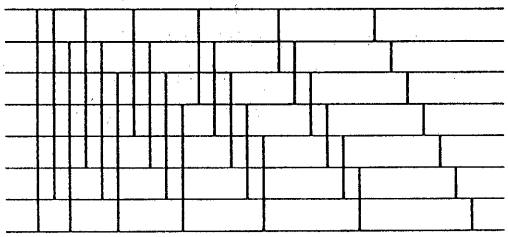

図 1: 本論文での提案 (区間減少ソート)

区間減少ソート

n 個の入力が与えられた場合，比較区間の最大幅は $n-1$ である．そこで，比較区間 $n-1$ から始め，1ずつ比較区間を減らしていき，最後に比較区間 1，つまり隣あったものを比較する．

同じ比較区間 k のとき，$n-k$ 箇所の比較を行なうことになるが，この場合は図1のように上から順序よく比較を行なうことにする．

2.2 区間減少ソートで正しくソートされることの証明

ここで，この区間減少ソートが正しくソートされる理由についてであるが，ワイヤの中の取り得るすべてのペアを比較しているということでは証明できない．なぜなら，図2のような場合，逆順の数値データをソートできないことからも明らかである．

そこで，区間減少ソートで正しくソートされることを証明する．ただし，a_i は i 番目のワイヤが保持している値とする．

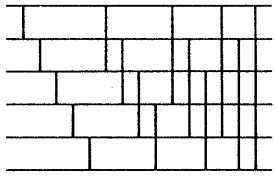

図 2: 比較区間を広げたためにうまくいかない例

証明

1. 最初に区間 $n-1(a_0$ と $a_{n-1})$ を比較交換するので，$a_0 \leq a_{n-1}$ となる．

2. 区間 k が終了した時点で，下の不等式が成立しているとする．

$$a_i \leq \min_{j \geq i+k} \{a_j\} \qquad (0 \leq i \leq n-1-k)$$

区間 $k-1$ を比較交換していく途中で，上の条件を壊すことなく a_i と a_{i+k-1} を比較交換することができる．そして，このとき，そのときの i について $a_i \leq \min_{j \geq i+k-1} \{a_j\}$ の条件に置き換わっていく．

3. 上の1，2より，最後に $k=1$ となったときは，

$$a_i \leq \min_{j \geq i+1} \{a_j\} \qquad (0 \leq i \leq n-2)$$

となり，これは，$a_0 \leq a_1 \leq \cdots \leq a_{n-1}$ と同値である．

これより，区間減少ソートで正しくソートされることがいえる．

証明終

2.3 $n-1$ 段のソーティングネットワークから n 段のものを作る方法

$n-1$ 段のソーティングネットワークから n 段のものを作る方法としては，図3に示すような $n-1$ 個の比較が増える方法が知られている．ここでは四角で囲った部分はすでに証明されている $n-1$ 段のソーティングネットワークであるとする．

ここで新たに次の方法を提案する．

まず前段に置く場合は，長さが1から $n-1$ までのワイヤをどういう順番でもよいので，図4のように張ればよい．

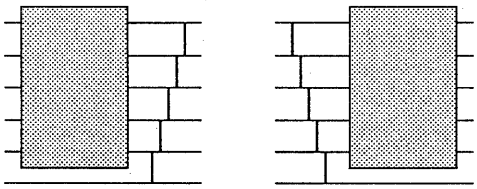

図 3: $n-1$ 段のソーティングネットワークから n 段のものを作る方法 [従来]

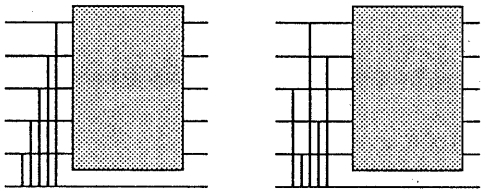

図 4: n 段のソーティングネットワークを作る方法 [前段に置く場合]

後段に置く場合は，図 5 のように長さが 1 から $n-1$ までのワイヤを長いものから順番に張る．後段に置く場合は，任意の順番に置けないことに注意する．

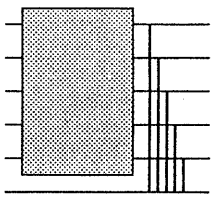

図 5: n 段のソーティングネットワークを作る方法 [後段に置く場合]

ところで，ここで示した長さが 1 から $n-1$ までのワイヤを長いものから順番に張る方法は，後段だけでなく任意の位置におけそうである (図 6)．実際，区間減少ソートは，$n-1$ 段のソーティングネットワークに長さが 1 から $n-1$ までのワイヤを長いものから順番に張っていることになる．

しかし，実際にはこの方法では，n 段のソーティングネットワークが完成されないことを図 7

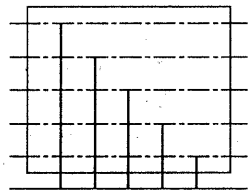

図 6: 任意の位置に置いて n 段のソーティングネットワークが完成するか？

に示しておく．

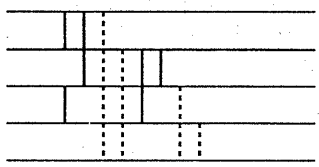

図 7: 4 段のソーティングネットワークから 5 段のものを作る場合にうまくいかない例

これより，区間減少ソートであれば，正しい位置にこれらのワイヤを順番通り張ることで，$n-1$ 段から n 段のものができているが，一般の場合にはいえないことになる．

注意：追加するワイヤをくっつけて置いた場合にも，うまくいかないと思われる．

3 改善された区間減少ソート

しかし，このままでは，同じ比較回数 $\frac{n(n-1)}{2}$ である直接挿入法やバブルソートを使わないで区間減少ソートを使うという利点は一つもない．しかし，あらかじめ比較する必要のない箇所を見つけておいてそこは比較しないようにすればこの比較回数よりも少なくできるのではないかというのが，本論文の研究テーマである．(直接挿入法やバブルソートではあらかじめ不必要箇所を削除するようなことはできない)

ただし，ここでの比較の不必要な箇所を見つけるのは，同じ区間 (入力数 n で，比較区間 k のときは，$n-k$ 箇所ある) をまとめて扱うものとする．

また，事前に比較箇所を取り除くのはソーティングネットワークという性質を壊すものではない

ことも大事な点である．

ここで，この不必要箇所をみつけるための手法について考える．そこで，区間減少ソートから不必要箇所を1区間 (広義には1箇所であるが，ここでは，同じ比較区間をまとめて考えているので1区間とする) でも除いたものを，「改善された区間減少ソート」と呼び，その中で最も比較回数を少なくしたものを「最適な区間減少ソート」と呼ぶことにする．

3.1 0－1原理

ソーティングネットワークに限定すれば，正しくソートされることの証明に，0－1原理という効率のよい検証手法が利用できる．これは，次のようなものである．

0－1原理

n 個の入力をもつソーティングネットワークAが，$\{0,1\}$ 上のあらゆる入力列 (2^n 個存在する) を正しくソートできれば，Aは整数，実数など線形順序が仮定される任意のデータ集合から選ばれた任意の入力列を正しくソートできる．

この方法によれば，入力数 n について $O(2^n)$ 時間で検証できる．次節でこれを効率よく計算機上で実行させる方法について考察する．

3.2 0－1原理を利用した検証アルゴリズム

0－1原理では，$\{0,1\}$ 上のあらゆる入力例を与えるのであるが，最初の状態ではすべて $\{\#\}$ の入力列を与えておく．(実際に計算機上で実行させるときには-1などを与える)

比較の箇所で入力される2つのデータに基づき次のように処理を行なう．

- (# #)・・・・・ (0 0)(# 1) の2種類に分け，それぞれ次のステップにいく．
  (0 #)(1 1) の2種類に分けてもよい．
- (# 0)・・・・・・ (0 #) として次へいく．
- (1 #)・・・・・・ (# 1) として次へいく．
- (1 0)・・・・・・ (0 1) として次へいく．
- (# 1) (0 #) (0 0) (0 1) (1 1) ・・・変更せず次へいく．

最終ステップまたは途中の段階で $(0\cdots01\cdots1)$ あるいは，$(0\cdots0\#1\cdots1)$ となれば，正しくソートされるので正常終了とする．

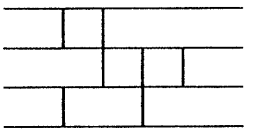

図 8: 問題1

問題1の検証

```
(# # # #)
  (0 0 # #)
    (0 0 0 0)
    (0 0 # 1)
  (# 1 # #)
    (# 1 0 0)(0 1 # 0)(0 0 # 1)
    (# 1 # 1)
      (0 1 0 1)(0 1 0 1)(0 0 1 1)
      (# 1 1 1)
```

この例だと正しくソートされる．

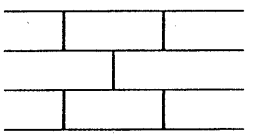

図 9: 問題2

問題2の検証

```
(# # # #)
  (0 0 # #)
    (0 0 0 0)
    (0 0 # 1)
  (# 1 # #)
    (# 1 0 0)(# 0 1 #)(0 # 1 #)(0 # # 1)
    (# 1 # 1)
      (# # 1 1)
        (0 0 1 1)
        (# 1 1 1)
```

この場合には最終段階で (0 # # 1) となっている箇所があり，正しくソートされない．

4 ギャップ数列探索アルゴリズム

区間減少ソートにおいて，前節のアルゴリズムを用いて不必要箇所を見つける方法を述べる．

4.1 ソーティングネットワークの性質

性質1・・・ソーティングネットワークにおいて比較箇所の順番入れ換えは一般にできない．

性質2・・・「すでにソートされることのわかっているソーティングネットワークに任意に比較箇所を付け加えたものもまたソーティングネットワークである」に対する真偽は偽である．

性質2は，ソーティングネットワークとして完成していないものから，あるワイヤをいくつか削除することで，それがソーティングネットワークとして成立することがあることを示唆している．そのため，ある未完成のソーティングネットワークが与えられたとき，0－1原理などの手法で，大小関係がまだ明らかになっていない区間を見つけて，そこの比較を追加する方法だけではなく，逆に比較の箇所を無くすことでソーティングネットワークが完成することもある．

4.2 区間減少ソートの性質

区間減少ソートに関する性質を述べる．

$(a\ b\ \cdots)$ というのは比較の不必要な箇所を表し，比較区間が $a, b, \cdots$ のときその区間全体で削除可能であるということを表すとする．

またこの比較区間のことをギャップ値と呼ぶことにする．

性質3・・・n−1段の場合に削除可能なギャップ値の組は，n段においても削除可能な組である．

一般的に，図10に示すように合成されたものがソーティングネットワークであることを証明する．ここで，四角で囲った部分はn−1段のソーティングネットワークとする．

証明

ここで，証明には0－1原理を利用する．まず，1段目が0であるあらゆる入力列に対し，下側のソーティングネットワークによりこれは正しくソーティングが行なえる．また，最下段が1であるあらゆる入力列に対しても，上側のソーティングネットワークにより正しくソーティングが行なえる．では，それ以外で，1段目が1で，最下段が0のあらゆる入力列の場合であるが，最初に1段目と最下段を比較することにしているので，1段目は0に，最下段は1に変わり，これも上のことから正しくソーティングが行なえる．これより，この方法でのソーティングネットワークの合成は可能である．

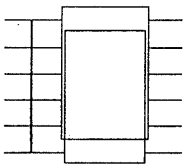

図10: ソーティングネットワークの合成

証明終

これより，性質3が真であることは明らかである．また，最初の1段目と最下段の比較は「必ず必要なもの」ではないことに注意する．改善された区間減少ソートを見つけることは，この最初の比較(図10での最初のワイヤの部分)がどういうときに削除できるのかを調べることと同じなのである．

性質4・・・n段において$n-1$段で削除できないギャップ値の組，及び，それにギャップ値として$(n-1)$を加えた組は，現れることはない．

証明

この$n-1$段の下に1本ワイヤを入れたものを考える．最下段に1を入れた場合，最初の比較区間$n-1$の比較をするしないに関わらず，この1との交換があり得ないため，これは正しくソーティングが行なえるものではない．

証明終

これより，n段のとき増えるものは$(n-1, [n-1$のときに削除可能なギャップ値の組$])$となるものに限る．もちろんすべてが増えるわけではない．

また，性質4を利用すれば次のようなことがいえる．

「削除できる比較区間として1，2，3の値はいかなるnについても現れない」

証明

入力が$1 \leq n \leq 4$のときに削除可能なギャッ

プ値として現れてこないから．($n=5$ のとき削除可能なギャップ値として始めて (4) が現れる)

証明終

4.3 ギャップ数列探索アルゴリズム

各 n の値について，削除可能なギャップ値の組は次のように見つける．

性質 3 を使うことで，まず $n-1$ 段の場合に削除可能なギャップ値の組は，n 段においても削除可能な組である．そして，その組にギャップ値として $(n-1)$ を付け加えた組だけを削除可能な組かどうか調べていけばよい．また逆に，性質 4 よりそれ以外を調べる必要はない．

この方法により，すべての改善された区間減少ソートを効率よく見つけることが可能である．

5 最良優先探索によるギャップ数列探索アルゴリズム

n が与えられたとき，最適な区間減少ソートを見つけるだけでよいならば，ヒューリスティック評価関数を用いた最良優先探索を用いることが考えられる．そこで，前節の性質 3，性質 4 を使い効率よく最適解を見つける方法について述べる．

ここでは，下のような木構造を考える．

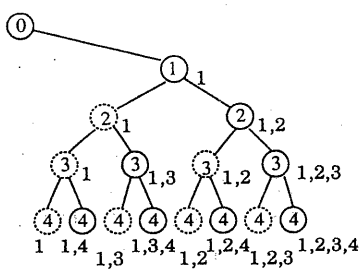

図 11: 比較区間を要素とする木構造

各節点は 2 つの子を持ち，右側の子は，比較箇所として，自分の持つ要素 (深さから 1 を引いた値) を追加するが，左側の子は追加しないものとする．また，頂点の深さを 1 とする．

ここで，最良優先探索でのヒューリスティック評価関数 f は節点を p とすると，次のように定義される．

$$f(p) = g(p) + h(p)$$

g，h はそれぞれ g^*(初期節点から節点 p までの最小コスト)，h^*(節点 p から目標節点までの最小コスト) の推定値である．また，木探索の場合は，常に $g(p) = g^*(p)$ である．

ここで，最適な区間減少ソートを見つけるには，$g(p)$ には，節点 p での比較回数 (入力数はその節点の深さと同じ値でよい)，$h(p)$ は，節点 p での比較箇所を深さ n のときに適応させた場合，どれだけ比較回数が増えるのか，その値を入れればよい．

$$g(p) = g^*(p) = \sum_{i=1}^{k} \{\mathrm{depth}(p) - a_i\}$$

$$h(p) = k \times \{\mathrm{depth}(P) - \mathrm{depth}(p)\}$$

k は節点 p での比較の対象となる区間の個数，a_i は i 番目の区間の値である．$\mathrm{depth}(p)$ は節点 p の深さ (あるいは入力データの数) であり，$\mathrm{depth}(P)$ は n に等しい．(P は目標節点)

最良優先探索では，常に $f(p)$ が最小となる節点を見つけ，それが目標節点である (この場合なら $\mathrm{depth}(p)$ が n に等しい) なら，探索が終了する．$\mathrm{depth}(p) < n$ なら，節点 p を展開する．

ここでは具体的に $n = 13$ の場合の実行後の木構造の例を示す．

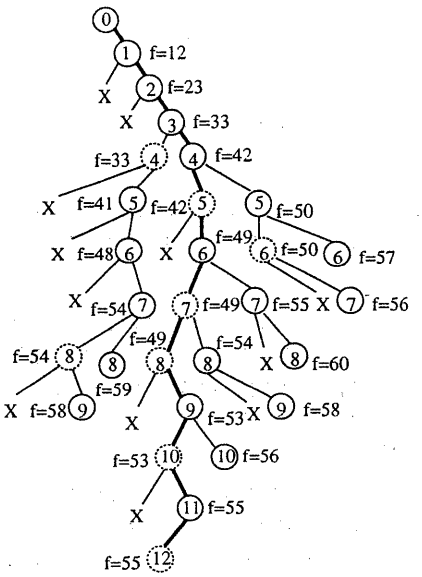

図 12: $n = 13$ の場合の探索例

$n = 13$ の場合には，このように区間 1 2 3 4 6 9 11 の比較を行なえばよいことがわかる (実際には大きい順に比較する)．ただし，ここで，注意すべきことは，入力数 n に対する最適な区間減少ソートは得られるが，このとき探索した木構造からは，n より小さい場合の最適な区間減少ソートの解は得られないことに注意を要する．

5.1 結果

最適な区間減少ソートは表 1 のようになった．

この表から，実際のプログラミングにおいては，ソーティング対象の入力数により次のように比較区間を設定すれば，区間減少ソートとしては，最も効率のよいものが得られる．

入力数	比較を行なうギャップ値の組
2 ～ 7	6 5 3 2 1
8 ～ 15	14 13 11 9 6 4 3 2 1
16 ～ 17	15 9 7 6 4 3 2 1
18 ～ 20	18 13 12 6 5 4 3 2 1
21 ～ 26	27 26 25 23 18 12 9 8 6 4 3 2 1
27 ～ 31	29 27 26 22 18 12 9 8 6 4 3 2 1
32 ～ 36	34 33 27 18 13 12 9 8 6 4 3 2 1
37 ～ 38	37 36 30 27 18 13 12 9 8 6 4 3 2 1
39 ～ 40	36 27 24 18 16 12 9 8 6 4 3 2 1

また，例えば入力数 10 に対して 39~40 の比較区間数列を利用することも可能である．(区間 9 を越えるものについては比較しないようにする)

これより実際の C プログラムの例を次に示す．

```
sort40(a,n) /* 区間減少ソート */
int *a;     /* ソートする配列 */
int n;   /* 個数 (n は 40 以下であること)*/
{
  int i=0,j,k,tmp;
  static int gap[]=
    {36,27,24,18,16,12,9,8,6,4,3,2,1};
  do{
    j=gap[i++];
    for(k=0;k<n-j;k++){
      if(a[k]>a[k+j]){
        tmp=a[k];
        a[k]=a[k+j];
        a[k+j]=tmp;
      }
    }
  }while(j>1);
}
```

入力数	比較を行なうギャップ値の組 (比較回数)
3	2 1 (3)
4	3 2 1 (6)
5	3 2 1 (9)
6	5 3 2 1 (13)
7	6 5 3 2 1 (18)
8	7 6 5 3 2 1 (24)
	6 4 3 2 1 (24)
9	6 4 3 2 1 (29)
10	9 6 4 3 2 1 (35)
11	9 6 4 3 2 1 (41)
12	11 9 6 4 3 2 1 (48)
13	11 9 6 4 3 2 1 (55)
14	13 11 9 6 4 3 2 1 (63)
15	14 13 11 9 6 4 3 2 1 (72)
16	15 9 7 6 4 3 2 1 (81)
17	15 9 7 6 4 3 2 1 (89)
18	17 15 9 7 6 4 3 2 1 (98)
	13 9 8 6 4 3 2 1 (98)
	13 12 6 5 4 3 2 1 (98)
19	18 13 9 8 6 4 3 2 1 (107)
	18 13 12 6 5 4 3 2 1 (107)
20	18 13 9 8 6 4 3 2 1 (116)
	18 13 12 6 5 4 3 2 1 (116)
21	20 18 13 9 8 6 4 3 2 1 (126)
	20 18 13 12 6 5 4 3 2 1 (126)
	18 12 9 8 6 4 3 2 1 (126)
22	18 12 9 8 6 4 3 2 1 (135)
23	18 12 9 8 6 4 3 2 1 (144)
24	23 18 12 9 8 6 4 3 2 1 (154)
25	23 18 12 9 8 6 4 3 2 1 (164)
26	25 23 18 12 9 8 6 4 3 2 1 (175)
	22 18 12 9 8 6 4 3 2 1 (175)
27	26 22 18 12 9 8 6 4 3 2 1 (186)
28	27 26 22 18 12 9 8 6 4 3 2 1 (198)
29	27 26 22 18 12 9 8 6 4 3 2 1 (210)
30	29 27 26 22 18 12 9 8 6 4 3 2 1 (223)
31	29 27 26 22 18 12 9 8 6 4 3 2 1 (236)
32	30 19 18 8 7 6 5 4 3 2 1 (249)
	27 18 13 12 9 8 6 4 3 2 1 (249)
33	27 18 13 12 9 8 6 4 3 2 1 (260)
34	33 27 18 13 12 9 8 6 4 3 2 1 (272)
35	34 33 27 18 13 12 9 8 6 4 3 2 1 (285)
36	34 33 27 18 13 12 9 8 6 4 3 2 1 (298)
37	36 34 33 27 18 13 12 9 8 6 4 3 2 1 (312)
	36 30 27 18 13 12 9 8 6 4 3 2 1 (312)
38	37 36 30 27 18 13 12 9 8 6 4 3 2 1 (326)
39	36 27 24 18 16 12 9 8 6 4 3 2 1 (341)
40	36 27 24 18 16 12 9 8 6 4 3 2 1 (354)

表 1: 最適な区間減少ソートにおける比較区間

5.2 考察

最適な区間減少ソートの比較回数と他のソーティングネットワークの比較回数とを比べてみる[1][2].

n	=	4	5	6	7	8	9
バブルソート	=	6	10	15	21	28	36
区間減少ソート	=	6	9	13	18	24	29
バイトニック	=	6				24	
奇偶マージ	=	5	9	12	16	19	26

10	11	12	13	14	15	16	···	32
45	55	66	78	91	105	120	···	496
35	41	48	55	63	72	81	···	249
						80	···	240
31	37	41	48	53	59	63	···	191

これをみると，比較回数はバブルソートに比べると随分小さくなることがわかる.

区間減少ソートでの比較回数はどのようになるのであろうか. $n = 2^m$ 個の要素をソートするのに奇偶ソートでは $(m^2 - m + 4)2^{m-2} - 1$ 個，バイトニックソートでは，$m(m+1)2^{m-2}$ 個の比較を行ない，どちらも，$O(n(\log n)^2)$ である. しかし，区間減少ソートにおける比較回数についてはまだわからない.

5.3 区間減少ソートとコムソート

Combsort(コムソート)[3] は，バブルソートのように隣接する要素を比較するのではなく，1 以上離れた要素を比較するようにしたものである. 具体的には，最初の比較区間として入力数を 1.3 で割った商を当てはめる. 以下，その区間での比較が終了すると前回の比較区間を 1.3 で割った商を新しい比較区間としていく. この値が 1 未満になると，それ以後の比較区間は 1 とし，一連の操作中に要素の位置が変化しなくなったら終了する. (最後はバブルソートに帰着する)

ここで提案する区間減少ソートでは，区間 1 での比較の終了時点でソートが完了しているということ，また入力数 n が決まれば比較回数があらかじめ決定されているという点で，大きく異なっている.

ところで，このコムソートでの収縮率 1.3 であるが，これはシュミレーション結果から導き出されたものである. 区間減少ソートで，要素数 40 に対して収縮率を計算してみる. 1.333, 1.125, 1.333, 1.125, 1.333, 1.333, 1.125, 1.333, 1.5, 1.333, 1.5, 2 となり，もちろん一定の値とはならないが，平均すれば 1.36 となり，コムソートでの値と似たような結果が得られる.

6 まとめ

本論文で，区間減少ソートというソーティング方法を考案し，0－1 原理や探索手法を用い，その比較回数が最小となる組合せを計算機により求めた.

この区間減少ソートは，途中でソーティングが終了していたとしても，それの判断は行なっていない. 比較とそれに付随した交換が 1 ステップで実行されるとすれば，区間減少ソートはどのような入力列に対しても一定の時間で処理を終えることができる. 区間減少ソートは必ずしも平均実行時間では，ソーティング終了条件がないなどもあって，決して早くはないが，どのような入力列に対しても一定の時間で終わるプログラム (終了条件のないバブルソートなど) に比べれば効率がよい.

ところで，現在はまだ要素数が 40 までのものしか，対象となる比較区間が求められていない. 一般の n についてどのようになっていくのかを今後検討していく予定である.

また，0－1 原理についても，本論文で紹介した検証手法でも果たして $O(2^n)$ の時間計算量がかかるのであろうか. このあたりは，与える問題に対して，大きく実行時間が変化するため，判断が難しい. 例えば，ソーティングネットワークとして正しいものと，正しくないものを検証すれば，もともと正しかったものを検証する方が一般的に時間がかかる. そのため，今後これらの時間計算量等についても考えていきたい.

謝辞：コムソートのことを教えて頂いた谷中清直氏に感謝いたします.

参考文献

[1] Batcher,K.E."Sorting networks and their applications", Proc AFIPS 1968 Spring Joint Comput. Conf. 1968,pp.307-314

[2] Knuth,D.E. "Sorting and Searching, Volume 3:The Art of Computer Programming"

[3] Stephen Lacey and Richard Box, "A Fast , Easy Sort", APRIL 1991,BYTE