Combsortによるソーティングの高速化とその評価

5L－5

柴田知範　志田晃一郎　藤川英司　山田新一
武蔵工業大学電気電子工学科

1　はじめに

クイックソートはとても高速であるがプログラムが複雑になるという問題点もある．そこでバブルソートが一般的に用いられてきたが平均的に途方もなく遅いという欠点がある[1]．しかし1991年にS. Lacy博士とR. Box氏がコムソートを開発しこのコムソート(その中でもコムソート11)がバブルソートにほんの少しプログラムを追加するだけのソーティング法としては最速であると発表した[2]．本研究ではコムソート11に改良を施し比較数，交換数，実行時間の面でコムソート11よりもさらに性能の良いソーティング法を開発することを試みている．本稿ではその概要を報告する．

2　コムソート11の原理

コムソートとはバブルソートのように隣接する要素同士を比較するのではなく遠く離れた要素同士をも比較するようにしソーティングの高速化を図ったものである．最初のストロークのギャップ値はデータの要素数を最適収縮率と言われる1.3の値で割ったものとし，ストロークが進むごとに前ストロークのギャップ値を1.3で割った値を次のギャップ値とする．ただし商が1未満になってもギャップ値は1とする．ギャップ値が1でありかつストローク中に要素の入れ換えが行なわれなくなった時点でソーティングは終了する．コムソート11とはギャップ値が9または10になったときそれを11に変更することにより能率を良くしたものある．

A fast sorting with Combsort, and an evaluation of it
Tomonori Shibata, Koichirou Shida, Hideji Fujikawa, Shinichi Yamada
Department of Electrical and Electronic Engineering, Musashi Institute of Technology

3　コムソートをもとにした改良

比較数，交換数そして実行時間の面でコムソート11に優るソーティング法を開発するためにいくつかの方法を試してみた．

まずギャップ値に素数ばかりを選んだ場合と，素数でない数を選んだ場合について調べた．コムソート11がたどるギャップ列の値に近い素数を選んだ場合，コムソートに優るとも劣らない結果を得ることができたが，コムソート11と比べると完全に劣った結果となった．約数をもつ数，その中でも特に約数を多く持つ数(多約数)を用いた場合，素数を用いた場合よりもかなり悪い結果となった．これは多約数の場合，ギャップの中に約数・倍数の関係にある数が多く存在し無駄な比較が多く行なわれ小さいギャップ値において交換を多く行なわなければならいからであると思われる．

コムソートでは最適収縮率として1.3の値が与えられているが，これは要素数が100の場合のものであり，要素数を100，1000，10000とした場合，平均して良いふるまいをする収縮率は1.32であることがシミュレーションの結果わかった．以降この値をもとに優良ギャップ列の選定を行なった．

コムソート11では1～11の間のギャップ値を選定している．これにならい収縮率が1.32であり1～11の間のギャップ値を選定することによりコムソート11に優ることを目指した．考えられるすべての組み合わせを試したところコムソート11に優るギャップ列は存在しなかった．試したギャップ列の中で比較的ふるまいの良かったギャップ列は{1,2,3,4,5,7,8,11}と{1,2,3,3,4,5,8,9,11}でありコムソートとはほぼ同性能であった．この結果より，コムソートの開発者が与えた1.3の収縮率と{1,2,3,4,6,8,11}のギャップ列を用いると最も優れたふるまいをすることがわかった．よって以降はこ

れらの値を用いるとともに11以上のギャップについても優れたふるまいを行なう値の選定を試みた.

ギャップの決定にあたり11を算出させる値は15であり15の前は20である. さらに20の前は26と27である. 11の前に15の値を設定したものを"コムソート15", そして15の前に20の値を設定したものを"コムソート20"とした. 20の前に値を設定するならば26または27であるがシミュレーションの結果,27の値の方が優れていたので"コムソート27"とした. 27の前は36であり36の前は47と48である. 27の前に36の値を設定したものを"コムソート36"とした. 36の前に47または48の値を設定した場合, そのふるまいがコムソート11に比べ悪くなった. 仮に"コムソート47"とすると48〜62という広い範囲の中のギャップが選ばれたときすべて47に移行されるが例えば$\{\cdots,40,53,70,\cdots\}$のパターンのギャップの場合, 70の次のギャップ値が53ではなく47になってしまう. コムソート36が比較的優れたふるまいをするとはいえギャップが53でソーティングできるところを47でソーティングすることにより交換の効率が悪くなってしまう. このため選定ギャップ列の拡大は36までが限度である. シミュレーションの結果, 要素数や要素の配列状態に関わらず常にコムソート11に優る改良型を開発することはできなかったが細かく見ていくとこれらの4つの改良型はコムソート11とほぼ同性能であるといえる.

4 シェルソートの導入

シェルソートはアルゴリズムが異なっているが収縮率およびギャップを用いる点がコムソートと類似している. このシェルソートとコムソートを組み合わせて新しいソーティング法の開発を試みた. シェルソートのギャップには要素数を$2^{n+1}-1$で割った値を用いているが, コムソートのギャップ決定の方法を用いた方がふるまいが良くなることがわかった. しかしそれでもコムソート11には劣る結果となった.

またあるギャップ値まではコムソートによりソーティングを行ないそこからシェルソートによりソーティングを行なう方法を試みた. シミュレーションの結果コムソート部の最適収縮率は1.39, 境界ギャップ値の最適値は15, シェルソート部の最適ギャップ列は$\{1,3,5,7\}$であることがわかったがコムソート11に優る結果は得られなかった.

逆に前半にシェルソートのふるまいを行ない後半にコムソート11のふるまいを行なうソーティング法を試みた. シミュレーションの結果境界ギャップ値の最適値は12であることがわかった. シェルソート部のギャップ値を様々な値にしてみたがコムソート11には劣る結果となった.

コムソートはバブルソート (ギャップ値:1) の回数が問題となるがシェルソートはギャップ値が小さいときに比較・交換が効率良く行なわれる. そこでシェルソートにおいてギャップ値が15のときを境に収縮率を変えたソーティング法を試みてみた. 具体的にはギャップ値が15になるまでは収縮率を1.69, そしてそれ以降のギャップ列を$\{1,3,5,7,11\}$としたものが最も優れているがコムソート11に劣る結果となった.

5 おわりに

コムソート11で選定されているギャップ列は11までであるが11以上のギャップ値の選定, そしてシェルソートの概念の導入によりコムソート11にかなり近づくことができた. 逆に言うとコムソート11は多少の改良を施しても劣ることのない優れたソーティング法である. しかし, 発表の場では常にコムソート11に優るソーティング法を紹介できると思う.

参考文献

[1] 平田 富夫, "アルゴリズムとデータ構造", 森北出版株式会社, 1990.

[2] S.Lacy and R.Box, "A Fast, Easy Sort", BYTE, pp.315-320, April 1991.

[3] 斎藤 英樹, "Combsortにおける評価と改良", 平成4年度武蔵工業大学電気電子工学科 卒業論文集.