数理解析研究所講究録
1120巻 1999年 68-77

線形ネットワークにおける 逐次・並列ソーティングの概念に基づいた分散ソーティング

A Distributed Sorting Algorithm on a Line Network: Adopting the Viewpoint of Sequential and Parallel Sorting

佐々木 淳
Atsushi SASAKI

NTT コミュニケーション科学基礎研究所
〒619-0237 京都府相楽郡精華町光台 2-4
atsushi@cslab.kecl.ntt.co.jp

摘要: 通常の分散ソーティングは逐次・並列ソーティングと問題設定・評価基準が異なるが，本稿では，逐次・並列アルゴリズムと同様の方針で分散ソーティング問題を扱う．簡単のために，対象とするネットワークを単純な線形ネットワークに限定する．同様の方針で作成されたアルゴリズムが既に存在するが，扱う要素数に制約があり，また，時間複雑度の解析が十分でない．そこで，まずこのアルゴリズムで扱われていない要素数に対する，ほぼ最適な時間複雑度を与える解決策を示す．この解決策は直感に反する性質を持つ興味深いアイデアである．次いで，要素数に制約のあるアルゴリズムの時間複雑度の解析を詳細に行い，その最適性を証明する．最後に，要素数に制約を設けないアルゴリズムへの拡張を行う．

キーワード: 分散アルゴリズム，ソーティング，時間複雑度，線形ネットワーク

1 はじめに

さまざまなアプリケーションにおいて，ソーティング問題は基本かつ重要な問題のひとつである．そのため，逐次・並列アルゴリズムだけでなく，分散アルゴリズムも研究されている．しかし，その基本的な問題設定，および，評価基準は逐次・並列アルゴリズムとは多少異なる．

Zaks[1] が分散ソーティングアルゴリズムを提案しているが，その評価はメッセージ複雑度のみで行われている．もちろん，多くのアプリケーションの混在により通信リンクが込み合っているネットワークにおいては，メッセージ数の増大が通信遅延を招くため，メッセージ複雑度を低く抑えることは重要である．しかし，単にメッセージ複雑度を低く抑えようとすると，同時に通信できるメッセージを減らすように設計されるため，大きな時間複雑度を要するアルゴリズムとなってしまう．また，一般に時間複雑度はメッセージ複雑度を上回ることがないため，メッセージ複雑度だけの評価で時間複雑度も同じ程度に抑えることができる．しかし，それぞれの下界は異なるため，メッセージ複雑度を最適にしたからといって時間複雑度が最適になるわけではない．この二つの複雑度を共に低く抑えることが望ましいが，それが困難な場合にはそのバランスを考慮する必要がある．

本稿では，Zaks のアルゴリズム [1] よりもメッセージ複雑度のオーダーを上げずに，時間複雑度のオーダーを下げ，下界値に近づけることを目的とする．つまり，逐次・並列アルゴリズムと同様に，時間複雑度を重視したアルゴリズム設計を行う．その第一歩として，最も単純なネットワークである線形ネットワーク上で，逐次・並列ソーティングと同じ設定の問題を取り扱う．同期モデルにおけるこの問題に対して，既に Hofstee ら [2] がアルゴリズムを考案している．しかし，そのアルゴリズムには，各プロセッサの保持する要素数が 2 以上という制約がある．そこで本稿では，この制約を必要としない

アルゴリズムを考える．まず，このアルゴリズムで実現できていないプロセッサの保持する要素数がいずれも1の場合のアルゴリズムを，直感には反するものの効果的なアイデアを導入することで実現する．一方，Hofsteeらはアルゴリズムの時間複雑度をnkと解析している[2]が，厳密な時間複雑度は$\frac{nk}{2}$でそれが最適であることを次に示す．なお，nはプロセッサ数，kは各プロセッサが保持する要素数である．最後に，Hofsteeらのアルゴリズム[2]と$k=1$の場合のアルゴリズムとを複合して，プロセッサが保持する要素数に関する制約のないアルゴリズムを提案する．

まず，次の章で線形ネットワークと分散ソーティング問題のモデルと定義を説明する．そして，3章においてプロセッサが保持する要素数がすべて1の場合の問題を解決するアルゴリズムを示し，4章においてHofsteeらのアルゴリズム[2]の時間複雑度を厳密に解析する．その後，プロセッサが保持する要素数に制約のないアルゴリズムを5章で提案する．最後に6章で本稿をまとめる．

2 モデルと定義

本稿では，プロセッサ$P_1, P_2, \ldots, P_n$を$P_i(1 \leq i < n)$とP_{i+1}の間に双方向リンクを持つように結合した線形ネットワーク[3]を扱う．一般性を失うことなく，P_1が左端となるように水平に置かれていると仮定する．このとき，各プロセッサは隣接プロセッサを“*left*”と“*right*”という局所的な名前でしか認識できない．但し，この左右の認識はどのプロセッサでも一致していることを仮定する．さらに，この認識には端の認識も含まれ，左端(P_1)では$left = null$，右端(P_n)では$right = null$とする．つまり，プロセッサP_iは*left*または*right*にプロセッサが存在することは認識できるが，それがどのプロセッサなのかは認識できず，さらに，自身の識別子iも知らない．また，プロセッサ数nも知らないと仮定する．

このような仮定の下で，各プロセッサは隣接プロセッサと適切な回数だけ通信を行うことで問題を解決する．このようなシステムのモデルには主に同期モデルと非同期モデルとがある[3, 4]．また，一般に，分散アルゴリズムの評価には時間複雑度とメッセージ複雑度の二つの基準が存在する．前者は同期モデルではラウンド数，非同期モデルではメッセージの最長鎖に属するメッセージ数で表され，後者はメッセージの総数で表される．

最後に，本稿で取り扱う分散ソーティング問題を定義する．まず，初期状態において，プロセッサP_iはソーティングの対象となる要素をk_i個保持している．ここで，このk_i個の要素を順序集合$B_i^I = \langle v_1^i, v_2^i, \ldots, v_{k_i}^i \rangle$で表記する．この要素列は初期状態において既にソートされている，つまり，$\forall j, 1 \leq j < k_i, v_j^i \leq v_{j+1}^i$を満たすと仮定する．なお，この仮定はアルゴリズムを分かりやすくすることと，記述を簡単にすることを目的にしており，実際に問題を解くときには必要としない．実際には，B_i^Iの中でmaxとminの演算ができれば問題はない．そして，各要素を通信により移動させて，最終状態において次の二つの条件を同時に満たすようにするのが問題である．なお，最終状態においてP_iが保持する順序集合をB_i^Fと表記する．

条件1: $\forall i, 1 \leq i < n, \max\{v_j^i | 1 \leq j \leq k_i\} \leq \min\{v_j^{i+1} | 1 \leq j \leq k_{i+1}\}$，つまり，$v_{k_i}^i \leq v_1^{i+1}$.

条件2: $\forall i, |B_i^F| = |B_i^I|$.

以下では，P_iが時刻tにおいて保持する順序集合をB_i^tと表記する．但し，非同期モデルにおける時刻は，本稿で提案するアルゴリズムにおいては，通信回数でカウントすることが可能である．

3 $\forall i, k_i = 1$の問題の解法

3.1 アルゴリズムの検討

本章では，直感には反するものの効果的なアイデアを導入することで，各プロセッサの保持する要素数がいずれも1の場合の問題を解くアルゴリズムを構築する．既に1章で述べたように，この問題に対応できる分散ソーティングアルゴリズムが存在する[1]．そのアルゴリズムはメッセージ複雑度を低く抑えることを目的にして作られているため，時間複雑度もメッセージ複雑度とほぼ同じ大きさになっている．しかし，時間複雑度の下界のオーダーは，メッセージ複雑度の下界のオーダーよりも低い．そこで，メッセージ複雑度よりも時間複雑度を重視してアルゴリズムを設計する．以下では同時に通信可

能なメッセージを最大限に利用して，時間複雑度を低く抑えることを目指す．なお，最終的には非同期モデルにおけるアルゴリズムを構築するが，理解を簡単にするために，アルゴリズムの本質的な処理は同期モデル上で説明する．

同期モデル上の本ソーティング問題は，線形アレイ上の並列ソーティング問題と非常に良く似ている．そして，並列ソーティングでは，奇偶交換ソート[5]により最適にソートすることができる．しかし，分散ソーティング問題においては，各プロセッサが大域的な位置を把握していないため，そのままでは奇偶交換ソートを実行することはできない．奇偶交換ソートを実行する前に大域的な位置を知るための処理を行うことは可能だが，その処理に $n-1$ ラウンド要するため，全体で $2n-1$[†] ラウンドを要する．さらに，この処理は左右端プロセッサから開始しなければならないため，非同期モデルにおいて，起動に要する時間が大きくなってしまう．

そこで，左右の隣接プロセッサと同時に通信し，どのプロセッサも起動から終了まで公平に動作する方法を考える．基本的なアイデアは，各プロセッサにおいて初期値 u を v_1, v_2 にコピーし，この二つの値を用いてネットワーク全体として奇偶交換ソートを行うことである．なお，具体的なアルゴリズムは紙面の都合で省略するが，付録に示したアルゴリズムと基本的な構成は同じである．以下では，このアルゴリズムを“DBS (distributed bubble sort) アルゴリズム”と呼ぶ．

3.2 アルゴリズムの正当性と複雑度

DBS アルゴリズムは要素数 $2n$ の奇偶交換ソートと同じ動作をし，1 ラウンドにつき 2 回の交換が行われるので，n ラウンド後には $v_1 = v_2$ となる．よって，DBS アルゴリズムの正当性は自明で，時間複雑度は n ラウンドである．一方，各ラウンドで $2(n-1)$ 個のメッセージが送信されるため，メッセージ複雑度は $2n(n-1)$ となる．なお，メッセージ複雑度の下界は $\frac{n^2}{2}$ である．これは，4.2節の補題1と同様の方法で証明できる．

次に，このアルゴリズムをそのまま非同期モデルに拡張した場合を考える．起動プロセッサは選択できないので，起動メッセージが全プロセッサに伝わるのに最大 $n-1$ 時間要する．よって，両端のプロセッサが DBS アルゴリズムの実行を開始するのは，最も遅いときで $n-1$ 時間後になる．従って，非同期モデルにおける時間複雑度は $2n-1$ 時間となる．さらに，どの時点においても両隣からメッセージを受信しないと内部処理が行われないので，全体の整合性が保証される．また，メッセージ複雑度は同期モデルと変わらず，$2n(n-1)$ である．

この複雑度は，すべての要素をあるひとつのプロセッサに集めてソートするという集中的で単純なアルゴリズムや既存のアルゴリズム[1]に比べ，メッセージ複雑度は多少悪くなっているものの，時間複雑度は良くなっている．なお，同期システムにおける時間複雑度はほぼ最適である．比較の詳細は文献[6]で行っているので，ここでは省略する．

4 $\forall i, k_i \geq 2$ の問題を解くアルゴリズムの時間複雑度

1章で述べたように，DBS アルゴリズムの時間複雑度は Hofstee ら[2]によって既に解析されている．しかし，この論文には詳細な解析は記述されておらず，結果だけが書かれている．そこで，この時間複雑度を詳細に解析したところ，実際には上記論文に記述されている値の半分になり，それが最適な時間複雑度となることが示された．さらに，上記論文では $\forall i, k_i = k$ の場合しか解析されていないが，本稿では，$\forall i, k_i \geq 2$ を満たすあらゆる値に対して解析した．

本章では，記述と理解を簡単にするために同期モデルを仮定する．なお，非同期モデルにおける時間複雑度は，すべてのプロセッサを起動するのに必要な時間である $n-1$ を，同期モデルの時間複雑度に加えることで得られる．また，メッセージ複雑度は，前章と同様に，同期モデルの時間複雑度に $2(n-1)$ を掛けることで得られ，これは同期モデルと非同期モデルとで差はない．

なお，具体的なアルゴリズムは紙面の都合で省略するが，前章同様に，付録に示したアルゴリズムと基本的な構成は同じである．DBS アルゴリズムの本質は，

1. $\max\{v_j^{i-1} | 1 \leq j \leq k_{i-1}\}$ と $\min\{v_j^i | 1 \leq j \leq k_i\}$，すなわち，$v_{k_{i-1}}^{i-1}$ と v_1^i，

[†] どちらか一方の端プロセッサからの距離を知ったプロセスから順に奇偶交換ソートを開始することで，$\frac{3}{2}n$ 程度にすることは可能．

2. $\max\{v_j^i | 1 \leq j \leq k_i\}$ と $\min\{v_j^{i+1} | 1 \leq j \leq k_{i+1}\}$, すなわち, $v_{k_i}^i$ と v_1^{i+1},

の二つの交換を同時に行うことである.

4.1 記号定義

次節におけるDBSアルゴリズムの時間複雑度解析において, 多くの記号を使用しているので, 本節でまとめて定義する.

$S(i,j,r)$: $B_i^r \cup B_{i+1}^r \cup \cdots \cup B_j^r$. 但し, $i \leq j$ であるとする. また, $j > n$ ならば, $S(i,j,r) = S(i,n,r)$ と定義する.

$S_i^+(r)$: ラウンド r までに P_i から P_{i+1} へ移動した要素の多重集合.

$S_i^-(r)$: ラウンド r までに P_i から P_{i-1} へ移動した要素の多重集合.

$e_i(S)$: 集合 S のなかで i 番目に小さな要素. 但し, $\exists i, \exists j > i, e_i(S) = e_j(S)$ ならば, $e_i(S)$ は $e_j(S)$ よりも左に存在する.

w_i: ネットワーク中のすべての要素のなかで i 番目に小さな要素. つまり, $w_i = e_i(S(1,n,0))$.

$M(S,i,j)$: $\{e_i(S), e_{i+1}(S), \ldots, e_j(S)\}$.

4.2 時間複雑度の解析

まずはじめに, 分散ソーティング問題を解くアルゴリズムの時間複雑度の下界を求める. 但し, 各リンクを1ラウンドで通過できる要素は高々ひとつであると仮定する. $\forall i, k_i = k$ を仮定すると, 次の補題が求まる.

補題 1 各プロセッサが k 個ずつの要素を保持する分散ソーティング問題を解くには,

1. n が偶数の場合:

   少なくとも $\frac{nk}{2}$ ラウンドを要する.

2. n が奇数の場合:

   少なくとも $\lfloor \frac{n}{2} \rfloor k + 1$ ラウンドを要する.

(証明) まず, n が偶数であると仮定する. プロセッサ $P_{\frac{n}{2}}$ と $P_{\frac{n}{2}+1}$ との間で必要な通信回数を考える. 最悪の場合, $S(1, \frac{n}{2}, 0)$ に属するすべての要素が $P_{\frac{n}{2}}$ から右へ送られる. このとき, $P_{\frac{n}{2}}$ から $P_{\frac{n}{2}+1}$ へ送られる要素の総数は $|S(1, \frac{n}{2}, 0)| = \frac{nk}{2}$ となり, 時間複雑度も少なくともこれと同じラウンド数を要する.

次に, n が奇数であると仮定する. まず, プロセッサ $P_{\lceil \frac{n}{2} \rceil}$ とその隣 (左右どちらでも議論は同じ) のプロセッサとの間で必要な通信回数が, n が偶数の場合と同様の議論によって求まる. さらに, $S(1, \lceil \frac{n}{2} \rceil - 1, 0)$ に属するすべての要素が $P_{\lceil \frac{n}{2} \rceil}$ より右へ移動する場合には, $P_{\lceil \frac{n}{2} \rceil}$ を介するために1ラウンドだけ余分に必要になる. その結果, 少なくとも $\lfloor \frac{n}{2} \rfloor k + 1$ ラウンド要することがわかる. □

詳細は省略するが, 同様の議論により, プロセッサごとに保持する要素数が異なる場合には, 次の補題が成り立つ.

補題 2 プロセッサ P_i が $k_i \geq 2$ 個の要素を保持する分散ソーティング問題を解くには, 少なくとも $\frac{m-d}{2} + f$ ラウンドを要する. 但し,

$$m = \sum_{i=1}^{n} k_i,$$

$$d = \min_{1 \leq j < n} \left\{ \left| \sum_{i=1}^{j} k_i - \sum_{i=j+1}^{n} k_i \right| \right\},$$

$$f = \begin{cases} 1, & \exists a, \exists b > a, \ s.t. \ d = \left| \sum_{i=1}^{a} k_i - \sum_{i=a+1}^{n} k_i \right| = \left| \sum_{i=1}^{b} k_i - \sum_{i=b+1}^{n} k_i \right|, \\ 0, & otherwise. \end{cases}$$

□

次にDBSアルゴリズムの時間複雑度を求める. 下界の解析と同様に, まず, $\forall i, k_i = k$ を仮定する. まず, 基本的な性質として, 任意のプロセッサ P_i に注目したとき, あるラウンドにおいて *left* に送られた要素が, 初期状態においてあるプロセッサ集合が保持する要素のなかで最小になることを示す. その前に, その準備として, r ラウンド以前に P_i が *right* へ送り, その後 r ラウンドまで P_i へは戻ってこない要素が, B_i^r の最小値よりも大きいことを示す. なお, 要素の大小比較は各ラウンドの終了時点, つまり, プロセッサ内でのソートが完了した時点を基準にして行う.

補題 3 $x \in S_i^+(r) \setminus S_{i+1}^-(r)$ に対し，$\min B_i^r \leq x$ が成り立つ．

(証明) $x \in S_i^+(r) \setminus S_{i+1}^-(r)$ ということは，$x \in S(1,i,0)$ かつ $x \in S(i+1,n,r)$ を意味する．x は r ラウンド以前に P_i と P_{i+1} の間を何度も往復している可能性もあるが，最後に P_{i+1} へ移動したラウンド以降のみを考慮すれば証明できる．

x が P_i から P_{i+1} へ r ラウンド以前で最後に移動したラウンドを $r' \leq r$ とすると，$\forall y \in B_i^{r'} \setminus v_k^i$ に対し，$x \geq y$ が成り立つ．さらに，ラウンド r' から r の間に交換により P_{i+1} から P_i に移動した要素の大きさは必ず x 以下である．一方，P_i には P_{i-1} から交換により x より大きな要素が送られてくる可能性があるが，それは 1 ラウンドにつき高々ひとつであり，その要素は次のラウンドにおいて必ず P_{i+1} へ交換により送られるので，P_i において常に $\forall y \in B_i^t \setminus v_k^i, r' \leq t \leq r$ に対し，$x \geq y$ が成り立つ．これは $\min B_i^r \leq x$ であることを意味する．この性質は，ラウンド r' 以降に x が P_{i+2}，またはそれよりさらに右のプロセッサに移動しても変わりない．従って，補題が成り立つことが示された． □

次に，あるラウンドにおいて *left* に送られる要素が，初期状態におけるあるプロセッサ集合が保持する要素の中で最小になることを示す．なお，以下で扱う集合は，いずれも多重集合として演算される．具体的には，和集合を求める演算では "∪" の代わりに "⊎" を用いる．

補題 4 ラウンド r における v_1^i の値は下記のようになる．

1. $r < n-i$ のとき：
$$v_1^i = \min\{S(i,i+r,0) \uplus S_{i-1}^+(r)\} \setminus S_i^-(r),$$
2. $r \geq n-i$ のとき：
$$v_1^i = \min S(i,n,r).$$

(証明) $r \geq n-i$ のときには，$S(i,i+r,0) = S(i,n,0)$ となるので，

$$\{S(i,i+r,0) \uplus S_{i-1}^+(r)\} \setminus S_i^-(r) = S(i,n,r)$$

となり，$r < n-i$ のときの式と一致する．従って，$r < n-i$ のときの式が，任意の r に対して成り立つことを r に関する帰納法を用いて証明する．

まず，$r = 0$ の場合には

$$S_{i-1}^+(0) = S_i^-(0) = \emptyset$$

であり，$S(i,i,0)$ は B_i^I と等しくなるので自明．

次に，すべての $i, 1 < i \leq n$ に関してラウンド $r-1$ で補題が成り立つと仮定して，ラウンド r でも補題が成り立つことを示す．帰納法の仮定により，ラウンド $r-1$ において P_{i+1} から P_i へ送られる要素 x は，

$$x = \min\{S(i+1,i+r,0) \uplus S_i^+(r-1)\} \setminus S_{i+1}^-(r-1)$$

であることが保証される．そして，補題 3 を用いると，$\forall y \in S_i^+(r) \setminus S_{i+1}^-(r)$ に対し $x \leq y$，$n > i+r$ のときに $\forall z \in S_{i+r}^+(r) \setminus S_{i+r+1}^-(r)$ に対し $x \leq z$ が保証されるので，ラウンド r においても

$$x \leq \min\{S(i+1,i+r,0) \uplus S_i^+(r)\} \setminus S_{i+1}^-(r)$$

であることが保証される．さらに，$S_{i+1}^-(r) \subseteq S(i+1,i+r,0) \uplus S_i^+(r)$ かつ $\min S_{i+1}^-(r) \leq x$ なので，

$$\min S_{i+1}^-(r) = \min S(i+1,i+r,0) \uplus S_i^+(r)$$

が得られる．従って，ラウンド r においては

$$\begin{aligned} v_1^i &= \min S(i,i,r) \\ &= \min\{S(i,i,0) \uplus S_{i-1}^+(r) \uplus S_{i+1}^-(r)\} \\ &\qquad \setminus \{S_i^+(r) \uplus S_i^-(r)\} \\ &= \min\{S(i,i,0) \uplus S_{i-1}^+(r) \\ &\qquad \uplus S(i+1,i+r,0)\} \setminus S_i^-(r) \\ &= \min\{S(i,i+r,0) \uplus S_{i-1}^+(r)\} \setminus S_i^-(r) \end{aligned}$$

となり，補題が成り立つことがわかる． □

なお，最大値に関しても補題 3, 4 と同様の性質が存在する．

次に，$1 < i \leq \lfloor \frac{n}{2} \rfloor$ を満たす P_i から P_{i-1} へ送られる要素について考える．まず，P_{i-1} から P_i へ送られる要素は，いずれも $w_{k(i-1)}$ よりも大きいと仮定する．このとき，$x \in M(S(i,n,0),1,k(i-1))$ が P_{i-1} へはじめて送られるのは，最も遅いときでもラウンド $n-2(i-1)$ である．ここで，$k(i-1)$ は $|S(1,i-1,0)|$，つまり，$\max_r\{|S_i^-(r) \setminus S_{i-1}^+(r)|\}$ を意味する．また，この最悪の状況においては，これら $k(i-1)$ 個の要素は $P_{n-i+2}, P_{n-i+3}, \ldots, P_n$ の

みにある．そして，$\forall x \in M(S(i,n,0),1,k(i-1))$ の P_{i-1} への送信が完了するのは，最も遅いときでもラウンド $n-2(i-1)+k(i-1) = n+(k-2)(i-1)$ である．これは，ラウンド $n+(k-2)(i-1)$ までには $n-2i+1$ 個だけ $M(S(i,n,0),1,k(i-1))$ に含まれない要素が送られていることを表している．そのなかに，$M(S(i,n,0),k(i-1)+1,ki)$ の要素がいくつ含まれるかを考える．これらは，P_i から P_{i-1} へ送られる要素の数が最も多い場合に，ソート完了時点において P_i に保持される要素となり得る．まず，$M(S(i,n,0),1,k(i-1))$ の要素がラウンド $n-2(i-1)$ まで送られない場合には，最悪でもラウンド $n-2(i-1)-1$ には $e_{k(i-1)+1}(S(i,n,0))$ が送られることが保証できる．そうでない場合には，ラウンド $n-2(i-1)-1$ には $x \in M(S(i,n,0),1,k(i-1))$ の送信が始まり，遅くともラウンド $n+(k-2)(i-1)-1$ には完了するので，ラウンド $n+(k-2)(i-1)$ には $e_{k(i-1)+1}(S(i,n,0))$ が送られることが保証できる．つまり，必ずラウンド $n+(k-2)(i-1)$ までには $e_{k(i-1)+1}(S(i,n,0))$ が P_{i-1} へ送られる．このことは，$e_{k(i-1)+1}(S(i,n,0))$ がラウンド $n+(k-2)(i-1)-1$ までには P_i に届くことも意味している．そして，さらに $k-1$ ラウンド経過すれば $\forall x \in M(S(i,n,0),k(i-1)+1,ki)$ が P_i に届く．

このことから，もし，$w_j, j \leq k(i-1)$ がすべて $w_j \in S(i,n,0)$ を満たすならば，P_{i-1} から P_i へ送られる要素はいずれも $w_{k(i-1)}$ よりも大きいという仮定を満たすので，次の補題が成り立つ．

補題 5 $1 < i \leq \lfloor \frac{n}{2} \rfloor$ に対し，$\forall w_j, j \leq k(i-1), w_j \in S(i,n,0)$ ならば，$w_{k(i-1)+1} \in B_i^{n+(k-2)(i-1)-1}$ となり，ラウンド $n+i(k-2)$ には P_i のソートが完了する．□

次に，$w_j \notin S(i,n,0)$ となる $w_j, 1 \leq j \leq k(i-1)$ がひとつ以上存在する場合を考える．この場合，先の議論における，P_{i-1} から P_i へ送られる要素はいずれも $w_{k(i-1)}$ よりも大きい，という仮定を満たすとは限らない．この仮定が満たされない場合，つまり，P_{i-1} から P_i へ $w_{k(i-1)}$ 以下の要素が送られたときには，アルゴリズムにより，同じラウンドにおいて P_i から P_{i-1} へそれよりも小さな要素が送られている．このとき，このような要素の交換が起こっても補題5と同様に次の補題が成り立つ．

補題 6 $1 < i \leq \lfloor \frac{n}{2} \rfloor$ に対し，$w_{k(i-1)+1} \in B_i^{n+(k-2)(i-1)-1}$ が成り立ち，ラウンド $n+i(k-2)$ には P_i のソートが完了する．

(証明) $n_s = |\{w_h | h \leq k(i-1), w_h \in S(i,n,0)\}|$ と定義し，$n_s < k(i-1)$ と仮定する．先の議論と同様に考えると，$x \in M(S(i,n,0),1,n_s)$ が P_{i-1} に送られ始めるのは，最悪の場合，ラウンド $r = n-(i-1)-\lceil \frac{n_s}{k} \rceil$ である．このラウンド以前には，先に述べたような交換は起こらない．さらに，$r = n-(i-1)-\lceil \frac{n_s}{k} \rceil \geq i-1$ であれば，補題4により P_{i-1} からは常に $S(1,i-1,r'), r' \geq i-1$ の最大値が送られてくるので，このラウンド以後も交換は起こらない．交換が起こらなければ，先の議論により得られたラウンド以前に P_i のソートが完了することは自明である．そこで，$r = n-(i-1)-\lceil \frac{n_s}{k} \rceil < i-1$ と仮定する．このとき，初期状態における各要素の分布によらず，$S(1,i-1,0)$ に含まれる $k(i-1)-n_s$ 個のうち $i-1$ 個は $S(1,i-2,r)$ に含まれるため，P_{i-1} から P_i に送られるこのような要素は最大 $(k-1)(i-1)-n_s$ 個である．従って，ラウンド $\{n-(i-1)-\lceil \frac{n_s}{k} \rceil\}+\{(k-1)(i-1)-n_s\} = n+(k-2)(i-1)-n_s-\lceil \frac{n_s}{k} \rceil$ には $\forall w_j, j \leq k(i-1)$ を P_i から P_{i-1} へ送ることができる．$n+(k-2)(i-1)-n_s-\lceil \frac{n_s}{k} \rceil < n+(k-2)(i-1)-1$ なので，補題5の状況が最悪のラウンド数になることがわかる．従って，補題5の仮定がない補題6も成り立つ．□

補題1, 6により，次の定理が成り立つ．

定理 1 線形ネットワークにおいて各プロセッサが $k \geq 2$ 個ずつ要素を保持する分散ソーティング問題は，*DBS* アルゴリズムを用いて下記の時間で最適に解くことができる．

1. n が偶数の場合：$\frac{nk}{2}$ ラウンド．
2. n が奇数の場合：$\lfloor \frac{n}{2} \rfloor k+1$ ラウンド．

(証明) まず，P_1, P_n には遅くとも $n-1$ ラウンドまでには，それぞれ最小値，最大値が届き，それから $k-1$ ラウンド経過すればソートが完了するので，ソート完了は $n+k-2$ ラウンドである．これは補題6で $i=1$ としたときと一致する．補題6により，プロセッサ $P_i, 1 < i \leq \lfloor \frac{n}{2} \rfloor$ がソートを完了するラウンドは $n+i(k-2)$ なので，P_1 から $P_{\lfloor \frac{n}{2} \rfloor}$ までのす

べてのプロセッサがソートを完了するラウンドは

$$\begin{aligned} & \max\{n+i(k-2) | 1 \le i \le \left\lfloor \frac{n}{2} \right\rfloor\} \\ = \; & n + \left\lfloor \frac{n}{2} \right\rfloor (k-2) \\ = \; & \begin{cases} \frac{nk}{2}, & n\text{が偶数の場合,} \\ \lfloor \frac{n}{2} \rfloor k+1, & n\text{が奇数の場合.} \end{cases} \end{aligned}$$

となる．また，ネットワークの対称性によって，$P_i, i \ge \lceil \frac{n}{2} \rceil + 1$ に対しても上記ラウンド数でソートが完了することを保証できる．さらに，n が奇数の場合，P_1 から $P_{\lfloor \frac{n}{2} \rfloor}$ までと，$P_{\lceil \frac{n}{2} \rceil + 1}$ から P_n までのソートが完了すれば，唯一残った $P_{\lceil \frac{n}{2} \rceil}$ もソートが完了していることがわかる．一方，この値は補題1と一致するので，最適であることが保証される．□

n が奇数の場合には，$\lfloor \frac{n}{2} \rfloor k + 1 = \frac{n-1}{2}k + 1$ である．そして，$k \ge 2$ なので $\frac{n-1}{2}k + 1 \le \frac{nk}{2}$ が成り立つ．従って，奇数の場合と偶数の場合を合わせて評価すると，次の系が成り立つ．

系 1 線形ネットワークにおいて，各プロセッサが $k \ge 2$ 個ずつ要素を保持するときの *DBS* アルゴリズムの時間複雑度は，$\frac{nk}{2}$ ラウンドである．□

各プロセッサの保持する要素数が異なる場合でも，同様の議論により次の定理と系が得られる．

定理 2 線形ネットワークにおいてプロセッサ P_i が $k_i \ge 2$ 個の要素を保持する分散ソーティング問題は，*DBS* アルゴリズムを用いて $\frac{m-d}{2} + f$ ラウンドで最適に解くことができる．但し，

$$m = \sum_{i=1}^{n} k_i,$$

$$d = \min_{1 \le j < n} \left\{ \left| \sum_{i=1}^{j} k_i - \sum_{i=j+1}^{n} k_i \right| \right\},$$

$$f = \begin{cases} 1, & \exists a, \exists b > a, \; s.t. \; d = \\ & \left| \sum_{i=1}^{a} k_i - \sum_{i=a+1}^{n} k_i \right| = \left| \sum_{i=1}^{b} k_i - \sum_{i=b+1}^{n} k_i \right|, \\ 0, & otherwise. \end{cases}$$

□

系 2 線形ネットワークにおいて，プロセッサ P_i が $k_i \ge 2$ 個の要素を保持するときの *DBS* アルゴリズムの時間複雑度は，$\frac{\sum_{i=1}^{n} k_i}{2}$ ラウンドである．□

なお，本章の議論は各プロセッサにおける終了判定を考慮しておらず，大域的に見たときのソート完了のみを議論している．実際の終了判定は，付録に示したアルゴリズムのように，定理2で示したラウンドの経過を知ることによって行うので，定理で与えられたラウンド数をアルゴリズム実行中に計算しなければならない．アルゴリズムの実行には少なくとも n ラウンドを要するため，この計算が n ラウンド以内に終了すれば問題ない．もし，P_i が $O(n)$ の記憶容量を持つならば，k_i をブロードキャストすることによって，n ラウンド以内で定理2を満たすアルゴリズムを設計できる．しかし，P_i が $O(1)$ の記憶容量しか持たない場合には，今のところ最大 $2(n-1)$ ラウンド要する計算方法しかできていないので，$m < 4(n-1)$ のときには実際にソーティングが完了しているにも関わらず，各プロセッサが終了を認識できずアルゴリズムの終了が定理2のラウンド数よりも遅れてしまう．しかし，系2で示したラウンド数以内にアルゴリズムが終了することは保証できる．

5 k_i に制約がない問題の解法

5.1 アルゴリズムの検討

本章では，プロセッサ P_i の保持する要素数 k_i に制約のないアルゴリズムを提案する．基本的に前章までの考え方に基き，$\forall i, k_i = k$ の場合には前章までのアルゴリズムと同じ動作をする，つまり最適に動作するアルゴリズムを作成する．

まず，$k_i = 1$ の場合でも $k_i \ge 2$ のときと同様に動作する必要があることから，交換を基本とする以上，$k_i = 1$ の場合には3章で使用している要素数を意図的に倍にするアイデアを使う必要がある．しかし，この倍にする処理をすべてのプロセッサで行うと，アルゴリズム自体には一切手を加える必要がないものの，$k_i = 1$ となるプロセッサがひとつもないときには実行時間も倍になってしまい，先に述べた $\forall i, k_i = k$ の場合に最適に動作するという方針に反する．そこで，$k_i = 1$ のプロセッサ P_i でのみこのアイデアを適用し，その代わりに P_i の保持する要素数を k_i に揃えるための後処理を設ける方法を考える．この場合には，$\forall i, k_i \ge 2$ であれば実行時間が増えず，効率的である．問題は $\exists i, k_i = 1$ のときにどの程度のオーバーヘッドを要するかである．な

お，$\forall i, k_i = k$ の場合，$k = 1$ のときと，$k \geq 2$ のときで時間複雑度が異なるため，これを合わせて評価すると，$\frac{n(k+[k=1])}{2}$[‡]となる．以下では，オーバーヘッドはこの時間複雑度からの増分として与える．

$\exists i, k_i = 1$ かつ $\exists i, j, k_i \neq k_j$ のときのみオーバーヘッドを要する方法を考える．なお，DBS アルゴリズムの終了時刻を T で表す．まず，$k_i = 1$ のプロセッサ P_i でコピーを作成する際，コピーには印を付けてオリジナルと区別することにする．そして，ソート完了後にコピーを消去することにする．以下では，コピー要素集合を C で表す．もし $|\{v|v \in C, v \in B_j^T, 1 \leq j < i\}| = |\{P_j|k_j = 1, 1 \leq j < i\}|$ が成り立つならば，P_i は *left* と通信する必要はない．*right* に関しても同様のことが言える．さらに，もしこの式が成り立たない場合には，左辺と右辺の差だけ *left* ないしは *right* へ要素を順次送れば良い．なお，*left* へ送るときには小さな要素から，*right* へ送るときには大きな要素から順に送る．そのため，この式の左辺と右辺の差を計算できる情報をソートが完了する前に用意する．

具体的には，初期値 0 の変数 c を用意し，コピーを *left* から受け取ったら $c := c+1$，コピーを *left* へ渡したら $c := c-1$ とすることで，$|\{P_j|k_j = 1, 1 \leq j < i\}|$ と $|\{v|v \in C, v \in B_j^T, 1 \leq j < i\}|$ の差と，$|\{P_j|k_j = 1, i \leq j \leq n\}|$ と $|\{v|v \in C, v \in B_j^T, i \leq j \leq n\}|$ の差を表現する．B_i^T のなかのコピーを消去したときに $c = 0$ かつ $|B_i^T| = k_i$ ならば，P_i はその時点で停止しても構わない．それ以外の場合には隣接プロセッサとの間で通信が必要になる．以下に場合分けして送受信する要素数を示す．なお，$|B_i^T| > k_i$ となるのは $k_i = 1$ かつ $|B_i^T| = 2$ の場合に限られ，また，P_i が *left* と *right* の両者に対し *send* 処理を行うことはない．

$c = 0$ で $|B_i^T| = k_i$ のとき： 終了．

$c = 0$ で $|B_i^T| < k_i$ のとき： $k_i - |B_i^T|$ 個の要素を *right* から受信．

$c = 0$ で $|B_i^T| = k_i + 1 = 2$ のとき： ひとつの要素を *right* へ送信．

$c < 0$ で $|B_i^T| - |c| = k_i$ のとき： $|c| = 1$ 個の要素を *left* へ送信．

$c < 0$ で $|B_i^T| - |c| < k_i$ のとき： $|c|$ 個の要素を *left* へ送信し，$k_i - |B_i^T| + |c|$ 個の要素を *right* から受信．

$c > 0$ で $c + |B_i^T| > k_i$ のとき： $c + |B_i^T| - k_i$ 個の要素を *right* へ送信し，c 個の要素を *left* から受信．

$c > 0$ で $c + |B_i^T| < k_i$ のとき： *right* から $k_i - c - |B_i^T|$ 個，*left* から c 個の要素をそれぞれ受信．

$c > 0$ で $c + |B_i^T| = k_i$ のとき： c 個の要素を *left* から受信．

なお，受信した要素は配列の受信した側の端に順次加えるだけで十分なので，送信する要素の数と方向が定まるだけでアルゴリズムを記述できる．$|c|$ の最大値は $n-1$ で，$|c| = n-1$ となるプロセッサは P_2, P_n しかないので，オーバーヘッドの最大値も $n-1$ となる．なお，この最大値を与える状況は，$\forall i, 1 \leq i < n$ に対し $k_i = 1$ で $k_n \geq 2(n-1)$ のときに，$\forall x \in S(1, n-1, 0)$ が $x \in B_n^T$ を満たす状況，あるいはこれと左右対称の状況で発生する．この後処理が最適な時間で処理されることは自明である．この後処理を含めた DBS アルゴリズムの同期モデルにおける詳細な記述を付録に示した．

5.2 複雑度解析

$\forall i, k_i = k$ のとき，先に述べたように $k = 1$ の場合と $k > 2$ の場合を合わせた時間複雑度は $\frac{n(k+[k=1])}{2}$ となる．この特殊な場合の時間複雑度は修正後も変わりない．そして一般の場合には，先の議論により，$\frac{\sum_{i=1}^{n}(k_i+[k_i=1])}{2} + n - 1$ となる．また，後処理のメッセージ複雑度は，先述の最大値を与える状況で最大のメッセージ数を与えるので，$\sum_{i=1}^{n-1} i = \frac{n^2-n}{2}$ となる．従って，全体のメッセージ複雑度は，$(n-1)\sum_{i=1}^{n}(k_i + [k_i = 1]) + \frac{n^2-n}{2}$ となる．この表現では複雑で直観的に分かり難いので，要素の総数を $m = \sum_{i=1}^{n} k_i$ とし，係数等を丸めてオーダー表記にすると，$m \geq n$ により，時間複雑度が $O(m+n) = O(m)$，メッセージ複雑度が $O(nm + n^2) = O(nm)$ となる．この拡張したアルゴリズムの複雑度は，3章での比較と同様に，既存のアルゴリズム [1] に比べ，メッセージ複雑度は係数レベルで大きくなっているものの，時間複雑度はオーダーがひとつ下がっている．

[‡] $[X]$ は，X が真のとき 1，偽のとき 0 となる．

6 おわりに

本稿では，線形ネットワークにおいて，文献[2]で示された分散ソーティングアルゴリズム（DBSアルゴリズム）の拡張と複雑度を議論した．論点は次の三つである．

まず，既存のアルゴリズム[1]とは異なる視点の分散ソーティング問題に対し，新たなアイデアを導入してアルゴリズムを構築した．ソートすべき要素の数を意図的に倍にすることでほぼ最適な時間複雑度が実現できるという興味深い結果を得た．

次に，DBSアルゴリズムの時間複雑度が$\frac{nk}{2}$となり，それが最適であることを証明した．文献[2]の解析ではnkとなっているので，この証明および結果には十分な意義がある．さらに，各プロセッサが保持する要素数が異なる場合に対しても時間複雑度を求め，最適性を証明した．

最後に，プロセッサが保持する要素数に制約のないアルゴリズムを提案した．このアルゴリズムは，付録に示すように，DBSアルゴリズムに後処理を加えたものである．この後処理は，DBSアルゴリズムの終了後に各プロセッサが保持する要素数を最適な時間で調整している．

DBSアルゴリズムの応用は既存のアルゴリズム[1]に比べ狭いものの，後処理を加えても，メッセージ複雑度のオーダーを等しく保ちつつ，時間複雑度を約$\frac{1}{n}$に削減し，共にオーダーとしては最適になっている．さらに，既存のアルゴリズムに比べ，プロセッサの動作の公平性が実現できているのも，大きな特徴の一つである．しかし，このアルゴリズムは$\exists i, k_i = 1$の場合に時間複雑度の最適性が保証されないため，検討の余地を残している．

その他の今後の課題としては，時間複雑度を低く保ちつつメッセージ複雑度を下げること，他の形状におけるアルゴリズムを構築すること，uidの順番に対応した並べ替えを行うアルゴリズムを時間複雑度を重視して構築すること，などが挙げられる．

参考文献

[1] Shmuel Zaks, Optimal Distributed Algorithms for Sorting and Ranking, *IEEE Transactions on Computers*, Vol. C-34, No. 4, pp. 376-379, 1985.

[2] H. Peter Hofstee, Alain J. Martin, and Jan L.A. Van De Snepscheut, Distributed Sorting, *Science of Computer Programming*, Vol. 15, No. 2-3, pp. 119-133, 1990.

[3] Nancy A. Lynch, *Distributed Algorithms*, Morgan Kaufmann Publishers, 1996.

[4] 亀田恒彦，山下雅史，分散アルゴリズム，近代科学社，1994.

[5] F. Thomson Leighton, *Introduction to Parallel Algorithms and Architectures: Arrays · Trees · Hypercubes*, Morgan Kaufmann Publishers, 1992.

[6] 佐々木淳，線形ネットワークにおける時間複雑度を重視した分散ソーティングアルゴリズム，*1999*年度日本オペレーションズ・リサーチ学会秋季研究発表会アブストラクト集，pp. 188-189, 1999.

付録

本稿で提案するアルゴリズム全体を同期モデル上で示す．本文での説明の都合上，Tラウンドまでを“DBSアルゴリズム”，その後を“後処理”と呼んで区別する．なお，非同期モデルにおけるアルゴリズムは，起動過程が多少異なるだけである．

まず，*null*プロセッサに関する動作を次のように仮定する．$left = null$で$receive((k_L, v_L), left)$が実行されたときには$k_L = 0, v_L = -\infty$，$right = null$で$receive((k_R, v_R), right)$が実行されたときには$k_R = 0, v_R = \infty$が得られる．さらに，$receive(x, P)$が正しく完了，つまり，$P$が直前のラウンドで$P_i$に$x$を送信していたら，この関数は*true*を返す．

1. 定数と変数の定義と初期値：

   k_{sum}: ネットワーク中の総要素数，初期値：$k_i + [k_i = 1]$.

   t: 時刻（ラウンド），初期値：-1.

   T: DBSアルゴリズムの終了時刻，初期値：∞．なお，最終的には

   $$T = \left\lfloor \frac{\sum_{i=1}^{n}(k_i + [k_i = 1])}{2} \right\rfloor$$

となる.

$v_1, v_2, \ldots, v_{k_i}$: 各時刻において P_i が保持する要素列で，初期状態においてはソートされた初期値を持つ．つまり，$\forall j, 1 \leq j < k_i, v_j \leq v_{j+1}$．さらに，$k_i = 1$ の場合には v_2 を用意し，$v_2 := v_1$ として，v_2 に *copy* とマーク付する.

B: $k_i = 1$ のときには，v_1 と v_2 を保持する一次元配列．それ以外のときには，要素列 $v_1, v_2, \ldots, v_{k_i}$ を保持する一次元配列．なお，B 中に *null* を含むときには，それを $|B|$ では数えない.

c: 後処理の必要性判定用の変数，初期値：0.

e_L, e_R: 左（右）のプロセッサ群が保持する要素の数，初期値：0.

u_L, u_R: B 中の *null* でない左（右）端の要素.

2. アルゴリズム (for P_i)：

$t := t + 1$
if $t = 0$ then
  if $left = null$ then
    $e_L := 0$
    $send((k_i + [k_i = 1], v_{k_i}), right)$
  if $right = null$ then
    $e_R := 0$
    $send((k_i + [k_i = 1], v_1), right)$
  if $left \neq null$ and $right \neq null$ then
    $send((0, v_{k_i}), right)$
    $send((0, v_1), left)$
if $1 \leq t < T$ then
  $receive((k_L, v_L), left)$
  $receive((k_R, v_R), right)$
  if $k_L > 0$ then
    $e_L := k_L$
    $k_{sum} := k_{sum} + k_L$
    $k_L := k_L + k_i + [k_i = 1]$
  if $k_R > 0$ then
    $e_R := k_R$
    $k_{sum} := k_{sum} + k_R$
    $k_R := k_R + k_i + [k_i = 1]$
  if $e_L > 0$, $e_R > 0$ and $T = \infty$ then
    $T := \lfloor \frac{k_{sum}}{2} \rfloor$
  if $v_1 < v_L$ then
    if v_1 is marked with *copy* then
      $c := c - 1$
    if v_L is marked with *copy* then
      $c := c + 1$
    $v_1 := v_L$
  if $v_{k_i} > v_R$ then
    $v_{k_i} := v_R$
  $sort(B)$
  $send((k_L, v_{k_i}), right)$
  $send((k_R, v_1), left)$
if $t = T$ then
  $receive((d_L, v_L), left)$
  $receive((d_R, v_R), right)$
  if $v_1 < v_L$ then
    if v_1 is marked with *copy* then
      $c := c - 1$
    if v_L is marked with *copy* then
      $c := c + 1$
    $v_1 := v_L$
  if $v_{k_i} > v_R$ then
    $v_{k_i} := v_R$
  delete elements marked with *copy* in B
  $sort(B)$
if $t > T$ then
  if $receive(v_L, left) = true$ then
    append v_L to the left-end in B
    $c := c - 1$
  if $receive(v_R, right) = true$ then
    append v_R to the right-end in B
if $t \geq T$ then
  if $c = 0$ and $|B| = k_i$ then
    sleep
  if $c = 0$ and $|B| > k_i$ then
    $send(u_R, right)$
    delete u_R from B
  if $c < 0$ then
    $send(u_L, left)$
    delete u_L from B
    $c := c + 1$
  if $c > 0$ and $c + |B| - k_i > 0$ then
    $send(u_R, right)$
    delete u_R from B