社団法人　情報処理学会　研究報告
IPSJ SIG Technical Report

2004－AL－97　(14)
2004／10／15

DNA計算における対数時間ソートアルゴリズム

蘆塚　正一†　　藤原　暁宏†

† 九州工業大学情報工学部　〒820-8502 福岡県飯塚市川津 680-4
E-mail: †ashizuka@zodiac30.cse.kyutech.ac.jp, fujiwara@cse.kyutech.ac.jp

あらまし　近年の高性能計算に関する研究において，非シリコン型計算の1つとして，DNA分子を用いて計算を行うDNA計算が注目を集めている．本研究では，基本操作の1つであるn要素のソートに対する対数時間のアルゴリズムを提案する．このソートアルゴリズムの入力は，$O(mn)$種類の一本鎖DNAで表現されたn個のmビットの2進数の集合である．この入力に対して，提案アルゴリズムは，$O(mn^2)$種類の一本鎖DNAを用いることにより，$O(\log n)$ステップでソートを実行することができる．

キーワード　DNA計算，ソート

A Polylogarithmic Sorting Algorithm with DNA Strands

Shoichi ASHIZUKA† and Akihiro FUJIWARA†

† Faculty of Computer Science and Systems Engineering, Kyushu Institute of Technology
Kawazu 680-4, Iizuka, Fukuoka, 820-8502 Japan
E-mail: †ashizuka@zodiac30.cse.kyutech.ac.jp, fujiwara@cse.kyutech.ac.jp

Abstract　In this paper, we propose a polylogarithmic procedure for sorting n elements. An input of the procedure is a set of n binary numbers of m bits, and each number is stored in $O(m)$ DNA strands. For the input, the proposed procedure runs in $O(\log n)$ steps using $O(mn^2)$ DNA strands.

Key words　DNA computing, sort

1. はじめに

近年の高性能計算に関する研究において，非シリコン型計算の1つとして，DNA分子を用いて計算を行うDNA計算が注目を集めている．DNA計算は，DNAの持つ超並列性，及び，ワトソン・クリック相補性を利用することにより，データ量が増加すると指数的に計算時間が増大するような問題に威力を発揮し，従来の計算機では指数的な計算時間を必要とするNP完全問題を，多項式的なステップ数で解くことを可能にする．DNAを計算に利用した最初のアルゴリズムは，Adleman [1] によるクラスNPに属するハミルトニアン経路問題を全解探索によって解くというものである．そのアルゴリズムは，DNA分子を用いて，頂点数nのハミルトニアン経路問題を$O(n)$ステップで解くというものであり，小規模なグラフに対して実際にDNAを用いた実験を行い，ハミルトニアン経路を求めることに成功している．DNA計算に関しては，このアルゴリズムの提案以降，多数のNP完全問題を解くアルゴリズム [2], [3], [10], [12], [16] が提案されている．

一方，DNA計算をNP完全問題だけでなく，より広い問題の解決方法として利用するためには，論理演算や算術演算のような通常の計算機が備えている基本的な演算が必要となる．このような基本演算については，論理演算や算術演算についてのいくつかの研究が行われている [4], [6]～[9], [13]．Guarnieri らのアルゴリズム [7] は加算を逐次的に行うものであり，2つのmビットの数の加算を$O(m)$種類のDNAを用いることにより$O(m)$ステップで実行する．Hug らのアルゴリズム [9] はDNAチップ上で数を表現し，それらに対して操作を行うことで，限定された論理演算，及び，加算を並列に実行するものであり，複数のデータに対して並列に演算できるが，同時に実行可能である演算は一種類のみである．このアルゴリズムは，mビットの数に対する1回の演算を$O(m)$種類のDNAを用いることにより$O(1)$ステップで実行する．Fujiwara らのアルゴリズム [6] は複数のデータに対して異なる論理演算，及び，加減算を並列に実行するものであり，n個のmビットの2進数による$O(n)$個の対に対する1回の演算を，$O(mn)$種類のDNAを用いることにより$O(1)$ステップで実行する．

本研究では，基本操作の1つであるn要素のソートに対する対数時間のアルゴリズムを提案する．このアルゴリズムの手法は，n入力に対してソートの結果として考えられる全ての解を用意して，その中から正しい解を選び出し出力するというもの

である．ここでのソートアルゴリズムの入力としては，$O(mn)$ 種類の一本鎖 DNA で表現されている n 個の m ビットの 2 進数の集合であるとする．この入力に対するソートを実行するアルゴリズムとして，本研究では，$O(mn^2)$ 種類の一本鎖 DNA を用いることにより，$O(\log n)$ ステップで実行できるアルゴリズムを提案する．

2. 準　　備

2.1 DNA による符号化

DNA は 4 種類の核酸塩基の配列により構成され，この並び方により情報を表現する．4 種類の核酸塩基が一方向に連結している DNA を一本鎖 DNA と呼ぶ．ここで，任意の一本鎖 DNA は生化学的手法により合成でき [11]，有限の記号集合 Σ により構成される列で表現することができる．DNA 計算では，DNA が持つワトソン・クリック相補性と超並列性という特徴を利用する．ワトソン・クリック相補性とは，互いに相補的な DNA のみが分子間力によって引き付け合い結合するという特徴であり，ある 1 つの DNA に対して完全に相補的な DNA は一種類のみ存在する．この性質を演算とみなすことで，DNA 計算は従来の計算機において演算を実行するために必要な処理装置を必要とせず，演算対象のデータである DNA 同士が直接反応して演算を行うことになる．超並列性とは，DNA が蓄えられる情報量の密度が高く，かつ多数のコピーを作成するのが容易であることに由来する．DNA 分子は溶液 $1ml$ 中に 6×10^{16} 個程度存在可能であり，相補的な DNA 同士はワトソン・クリック相補性により結合する．したがって，問題を DNA で符号化できれば，溶液中の莫大な数の DNA 同士がそれぞれ多様な組み合わせで結合し，解候補を生成することができる．莫大な数の DNA 同士がそれぞれ並列に結合するという特徴が超並列性である．

本論文では，Σ を異なる記号の集合 $\{\sigma_0,\sigma_1,\ldots,\sigma_{m-1},\overline{\sigma_0},\overline{\sigma_1},\ldots,\overline{\sigma_{m-1}}\}$ とし，Σ の各要素が DNA を表現するものと仮定する．ここで，σ_i と $\overline{\sigma_i}(0\leq i\leq m-1)$ は相補的な一本鎖 DNA の対を表し，これらの互いに相補的な一本鎖 DNA はワトソン・クリック相補性により分子間力によって引き付け合い水素結合する．一本鎖 DNA の核酸塩基配列の決定には，完全に相補的な一本鎖 DNA 以外は結合しないように，部分的に相補性を持つ配列の出現を低く抑えるアルゴリズム [5] [15] を用いる．相補的な一本鎖 DNA の対が，ワトソン・クリック相補性により水素結合した DNA を二本鎖 DNA と呼び，互いに相補的な一本鎖 DNA からなる二本鎖 DNA を $\begin{bmatrix}\sigma_i\\ \overline{\sigma_i}\end{bmatrix}$ と表現する．任意の核酸塩基配列からなる一本鎖 DNA が生化学的に合成できることにより，2 つの一本鎖 DNAσ_0，σ_1 が共有結合により連結した一本鎖 DNA$\sigma_0\sigma_1$ もまた生化学的に合成できる．また，2 つの一本鎖 DNA の一部分が相補的になっているとき，2 つの一本鎖 DNA の一部分が結合した二本鎖 DNA を形成する．例えば，$\sigma_0\sigma_1$，$\overline{\sigma_1\sigma_0}$ という 2 種類の一本鎖 DNA の場合，2 種類の二本鎖 DNA $\begin{bmatrix}\sigma_0\sigma_1\\ \overline{\sigma_1\sigma_0}\end{bmatrix}$，$\begin{bmatrix}\sigma_0\sigma_1\\ \overline{\sigma_1\sigma_0}\end{bmatrix}$ を形成することができる．ここで，使用する一本鎖 DNA 及び二本鎖 DNA の集合を試験管と呼ぶ．また，試験管内の DNA は同じ DNA が複数存在する重複集合であると仮定する．

2.2 DNA による計算モデル

本論文では，Reif [14] によって提唱された DNA 計算モデルである RDNA モデルを使用する．この RDNA モデルは，DNA の表現及び DNA の生化学的操作を抽象化し，DNA 計算のアルゴリズムの簡潔な記述を可能にするものである．

RDNA モデルに基づき，DNA の基本操作として以下の 9 操作を用いる．

（1） *Merge*(T_1,T_2): 試験管 T_1 と試験管 T_2 の DNA を混合し，2 つの集合の和を試験管 T_1 とする．つまり，$T_1=T_1\cup T_2$ とする．

（2） *Copy*(T_1,T_2): 試験管 T_1 の DNA を複製し，同じ内容の試験管 T_2 を作成する．

（3） *Detect*(T_1): 試験管 T_1 に DNA が存在するならば真を，存在しないならば偽を返す．

（4） *Separation*(T_1,X,T_2): 試験管 T_1 から集合 X に含まれる記号列を含む一本鎖 DNA を試験管 T_2 に取り出す．

（5） *Selection*(T_1,L,T_2): 試験管 T_1 から長さ L の DNA を試験管 T_2 に取り出す．ここで，長さ L とは記号の個数とする．例えば，$\sigma_0\sigma_1\sigma_2$ で表現される DNA の長さは 3 とする．

（6） *Cleavage*$(T_1,\sigma_0\sigma_1)$: 試験管 T_1 において，DNA の二本鎖部分 $\begin{bmatrix}\sigma_0\sigma_1\\ \overline{\sigma_0\sigma_1}\end{bmatrix}$ を $\begin{bmatrix}\sigma_0\\ \overline{\sigma_0}\end{bmatrix}$ と $\begin{bmatrix}\sigma_1\\ \overline{\sigma_1}\end{bmatrix}$ に切断する．

（7） *Annealing*(T_1)[注1]: 試験管 T_1 内の相補的な一本鎖 DNA 同士を結合させ，二本鎖 DNA にする．

（8） *Denaturation*(T_1): 試験管 T_1 内の二本鎖 DNA を切り離し，2 つの一本鎖 DNA に分離する．

（9） *Empty*(T_1): 試験管 T_1 を空にする．

これらの DNA に対する操作は生化学的操作の組み合わせにより実行できるので，それぞれの操作の計算量を O(1) ステップと定義する．

2.3 DNA による 2 進数の表現

次に，DNA による 2 進数の表現を定義する．文献 [6] において，$O(mn)$ 種類の一本鎖 DNA を用いることにより，n 個の m ビットの 2 進数を表現する方法が定義されている．文献 [6] の方法は，n 個の 2 進数を 1 ビット毎に一本鎖 DNA で表現するというアイデアに基づいており，本研究では，同様の方法で DNA により数を表現するものとする．以下にその概要を示す．

RDNA モデルにおける全ての DNA は集合 Σ により表現されるので，集合 Σ の要素を組み合わせることにより数を表現する．数の表現には，以下の集合により定義される $O(m+n)$ 個の一本鎖 DNA を使用する．

(注1)：文献 [14] では，核酸塩基配列の間の隙間を埋める *Ligation* という操作も定義されているが，本論文では *Annealing* がその操作を含むものとする．

$$\begin{aligned}\Sigma = \{&A_0, A_1, \ldots, A_{n-1}, B_0, B_1, \ldots, B_{m-1},\\ &C_0, C_1, D_0, D_1, 1, 0, \#,\\ &\overline{A_0}, \overline{A_1}, \ldots, \overline{A_{n-1}}, \overline{B_0}, \overline{B_1}, \ldots, \overline{B_{m-1}},\\ &\overline{C_0}, \overline{C_1}, \overline{D_0}, \overline{D_1}, \overline{1}, \overline{0}, \overline{\#}\}\end{aligned}$$

ここで，$A_0, A_1, \ldots, A_{n-1}$ はアドレスを表し，$B_0, B_1, \ldots, B_{m-1}$ はビットを表すものとする．C_0C_1 及び D_0D_1 は基本操作 *Cleavage* により切断する部分を表し，二本鎖 DNA $\left[\begin{array}{c}C_0C_1\\ \hline \overline{C_0C_1}\end{array}\right]$ は $\left[\begin{array}{c}C_0\\ \hline \overline{C_0}\end{array}\right]$ $\left[\begin{array}{c}C_1\\ \hline \overline{C_1}\end{array}\right]$ のように，二本鎖 DNA $\left[\begin{array}{c}D_0D_1\\ \hline \overline{D_0D_1}\end{array}\right]$ は $\left[\begin{array}{c}D_0\\ \hline \overline{D_0}\end{array}\right]$ $\left[\begin{array}{c}D_1\\ \hline \overline{D_1}\end{array}\right]$ のように切断されるものとする．“0” 及び “1” はビットの値を表し，“#” は基本操作 *Separation* で用いるための記号とする．m ビットの数は 1 ビット毎にそれらの一本鎖 DNA を組み合わせた一本鎖 DNA で表現し，アドレス i の j 番目のビットの値が $V_{i,j}(\in \{0,1\})$ の場合の一本鎖 DNA である $S_{i,j}(0)$, $S_{i,j}(1)$ を以下の様に定義する．

$$S_{i,j}(0) = D_1A_iB_jC_0C_10D_0$$

$$S_{i,j}(1) = D_1A_iB_jC_0C_11D_0$$

ここで，各 $S_{i,j}$ をメモリ鎖と呼び，n 個の m ビットの 2 進数は $O(mn)$ 種類の一本鎖 DNA で表現する．つまり，アドレス i の値が $V_i = \sum_{j=0}^{m-1} V_{i,j} \times 2^j$ であるような 2 進数 $V_{i,m-1}V_{i,m-2}\ldots V_{i,0}$ は，$\{S_{i,m-1}, S_{i,m-2}, \ldots, S_{i,0}\}$ の $O(m)$ 種類のメモリ鎖の集合により表現される．このとき，$V_i(k)$ は，$V_i = k$ となるような値を保持したメモリ鎖の集合 $\{S_{i,m-1}, S_{i,m-2}, \ldots, S_{i,0}\}$ を表すものとする．

$$k = \sum_{j=0}^{m-1} V_{i,j} \times 2^j \text{のとき} \quad V_i(k)$$

また，メモリ鎖の集合 $\{S_{i,m-1}, S_{i,m-2}, \ldots, S_{i,0}\}$ をビット位置である j を基準に，降順に連結させた場合の一本鎖 DNA である S_i を以下の様に定義する．

$$S_i = S_{i,j}S_{i,j-1}\ldots S_{i,1}S_{i,0}$$

2.4 DNA による既知の演算

文献 [6] において，n 個の m ビットの 2 進数を $O(mn)$ 種類の一本鎖 DNA で表現した場合に，任意の個数のメモリ鎖に値を割り当てるアルゴリズム，及び，任意の論理演算，加算を実行するアルゴリズムについて，以下のような補題が証明されている．

[補題 1] 任意の個数のメモリ鎖への値の割り当ては，$O(1)$ 種類の DNA を用いることにより $O(1)$ ステップで実行可能である． □

[補題 2] n 個の m ビットの 2 進数における $O(n)$ 個の対に対する任意の論理演算は，$O(mn)$ 種類の DNA を用いることにより $O(1)$ ステップで実行可能である． □

[補題 3] n 個の m ビットの 2 進数における $O(n)$ 個の対に対する加算は，$O(mn)$ 種類の DNA を用いることにより $O(1)$ ステップで実行可能である． □

ここでは上記補題の詳細は省略するが，任意の試験管内のメモリ鎖に値 $V(\in \{0,1\})$ を割り当てる操作を，$O(1)$ 種類の DNA を用いることにより $O(1)$ ステップで行う手続きを $ValueAssignment\_V(T_{input}, T_{output})$ と定義する．これは，試験管 T_{input} 内の全てのメモリ鎖に対して値 V を割り当てた一本鎖 DNA の集合を試験管 T_{output} とする手続きである．また，n 個の m ビットの 2 進数における $O(n)$ 個の対に対する任意の論理演算を，$O(mn)$ 種類の DNA を用いることにより $O(1)$ ステップで行う手続きを $LogicOperation(T_{input}, L, T_{output})$ と定義する．これは，試験管 T_{input} 内のメモリ鎖に対し，真理値表を表す一本鎖 DNA の集合 L を利用して論理演算を実行し，論理演算の結果を表すメモリ鎖の集合を試験管 T_{output} とする手続きである．更に，本論文で提案するアルゴリズムでは加算，減算を使用するが，減算は加算を応用することで実行可能であるので，減算の計算量は加算と同じである．したがって，n 個の m ビットの 2 進数における $O(n)$ 個の対に対する加算，減算は，$O(mn)$ 種類の DNA を用いることにより $O(1)$ ステップで実行することができる．$O(n)$ 組の m ビットの 2 進数に対する加算を，$O(mn)$ 種類の DNA を用いることにより $O(1)$ ステップで行う手続きを $AdditionOperation(T_{input}, R, T_{output})$ と定義する．これは，試験管 T_{input} 内のメモリ鎖に対し，R で示される各アドレス対の 2 進数に対して並列に加算を実行し，加算結果を表すメモリ鎖の集合を試験管 T_{output} とする手続きである．加算と同様，ここでは詳細は省略するが，$O(n)$ 組の m ビットの 2 進数に対する減算を，$O(mn)$ 種類の DNA を用いることにより $O(1)$ ステップで行う手続きを $SubtractionOperation(T_{input}, R, T_{output})$ と定義する．これは，試験管 T_{input} 内のメモリ鎖に対し，R で示される各アドレス対の 2 進数に対して並列に減算を実行し，減算結果を表すメモリ鎖の集合を試験管 T_{output} とする手続きである．

2.5 DNA による入出力

本論文で提案するアルゴリズムに対する入力は，以下のような試験管 T_{input} により与えられる．

$$T_{input} = \{S_{i,j} \mid 0 \leqq i \leqq n-1, 0 \leqq j \leqq m-1\}$$

ここでは，簡単のため入力される数 n は 2 のベキ乗 ($n = 2^k$, かつ，k は整数) であると仮定する．出力は T_{output} に入力値をアドレスに対して昇順に並び換えたものが入れられる．

本論文では各 2 進数の最上位ビットである $m-1$ 番目のビットは符号を表すビットとし，アドレス i の $m-1$ 番目のビットの値が 1，つまり $S_{i,m-1}(1) = D_0A_iB_{m-1}C_0C_11D_1$ の場合はアドレス i の値 V_i は負であるとする．

3. ソートアルゴリズム

本章では，n 個の m ビットの 2 進数の集合を入力とし，入力集合に対して $O(mn^2)$ 種類の DNA を用いることにより $O(\log n)$ ステップでソートを実行するアルゴリズムを提案する．

3.1 移動鎖の生成

今回，提案したソートアルゴリズムでは，ソートを実行するため，アドレス間で値の移動を行う特別な一本鎖 DNA を必要とする．本論文ではこの特別な一本鎖 DNA を移動鎖と呼ぶことにする．本節では，提案したソートアルゴリズムの説明の前に，移動鎖を生成するアルゴリズムの説明を行う．

3.1.1 移動鎖の定義

移動鎖とは以下の様に定義される一本鎖 DNA である．

$$\#A_iA_kD_0D_1A_{4i+k+2n^2}B_jC_0C_1VD_0$$

移動鎖は，以下の様に構成されている．まず一番左側に基本操作 *Separation* のために使用される記号#があり，その次には，ソート結果を出力するアドレスと入力のアドレスを示す部分 A_iA_k が存在する．この部分は，アドレス A_k の値をソートした結果，その値をアドレス A_i に移動するという操作を表している．また，その次に続く部分 D_0D_1 は基本操作 *Cleavagae* により切断される箇所であり，その後ろに続くメモリ鎖は，移動後の値が格納されている場所を表している．4 入力 {7, 1, 2, 4} の具体例で示すと表 1 のようになる．表 1 において，V, V' はソート前，ソート後の状態を表す．

ただし，入力によってアドレスの変更方法は変化するので，入力が n 個ならば移動鎖は n^2 通りのものを準備する必要がある．表 2 に 4 入力の場合の移動鎖の例を示す．表 1 と表 2 の違いは，表 1 は 4 入力の具体例の 1 つの表であるのに対して表 2 は 4 入力の汎用的な表である．

3.1.2 移動鎖生成アルゴリズムの概要

移動鎖生成アルゴリズムの概要は以下の通りである．まず最初に n 個の入力を n^2 個のアドレスに対して論理演算を用いてコピーする．次に，以下のような入力とは別に用意された DNA 鎖を各値に対して付加して移動鎖を生成する．

$$\{\#A_iA_jD_0 \mid 0 \leqq i \leqq n-1, 0 \leqq j \leqq m-1\}$$

アルゴリズムは以下の 4 ステップから成る．

Step 1　試験管 T_{input} を試験管 T_{tmp} にコピーする．

Step 2　試験管 T_{tmp} を試験管 T_1 に混ぜ，入力値を論理演算を使って準備された別アドレスにコピーする．

Step 3　試験管 T_2 を試験管 T_1 に混ぜ，移動鎖を生成する．

Step 4　試験管 T_1 を試験管 T_{output} にコピーする

上記のアルゴリズムにおいて各試験管は以下のような役割を果たす．

- T_{input}：入力 DNA 鎖を保持
- T_{output}：出力 DNA 鎖を保持
- T_{tmp}：DNA 鎖を一時的に保持
- T_1, T_2：解を構成するために必要な DNA 鎖を保持
- T_{trash}：不要となった DNA 鎖を入れる

以下に各ステップの動作を説明する．

Step 1 は基本操作 *Copy* を用いることにより $O(1)$ ステップで実行可能である．

Step 2 では，試験管 T_{tmp} に以下の用意した試験管 T_1 を基本操作 *Merge* を使い，マージする．

表 1　4 入力 {7, 1, 2, 4} の具体例

Table 1　a table of an example in four inputs {7,1,2,4}

ソート前	ソート後	変化	移動鎖
$V_0(7)$	$V_0'(1)$	$V_1 \Rightarrow V_0'$	$\#A_0A_1D_0S_{33,j}$
$V_1(1)$	$V_1'(2)$	$V_2 \Rightarrow V_1'$	$\#A_1A_2D_0S_{38,j}$
$V_2(2)$	$V_2'(4)$	$V_3 \Rightarrow V_2'$	$\#A_2A_3D_0S_{43,j}$
$V_3(4)$	$V_3'(7)$	$V_0 \Rightarrow V_3'$	$\#A_3A_0D_0S_{44,j}$

表 2　4 入力の具体例 2

Table 2　a table of an example in four inputs

操作前	操作後	変化	移動鎖
V_0	V_0'	$V_0 \Rightarrow V_0'$	$\#A_0A_1D_0S_{32,j}$
	V_1'	$V_0 \Rightarrow V_1'$	$\#A_0A_2D_0S_{33,j}$
	V_2'	$V_0 \Rightarrow V_2'$	$\#A_0A_3D_0S_{34,j}$
	V_3'	$V_0 \Rightarrow V_3'$	$\#A_0A_0D_0S_{35,j}$
V_1	V_0'	$V_1 \Rightarrow V_0'$	$\#A_1A_1D_0S_{36,j}$
	V_1'	$V_1 \Rightarrow V_1'$	$\#A_1A_2D_0S_{37,j}$
	V_2'	$V_1 \Rightarrow V_2'$	$\#A_1A_3D_0S_{38,j}$
	V_3'	$V_1 \Rightarrow V_3'$	$\#A_1A_0D_0S_{39,j}$
V_2	V_0'	$V_2 \Rightarrow V_0'$	$\#A_2A_1D_0S_{40,j}$
	V_1'	$V_2 \Rightarrow V_1'$	$\#A_2A_2D_0S_{41,j}$
	V_2'	$V_2 \Rightarrow V_2'$	$\#A_2A_3D_0S_{42,j}$
	V_3'	$V_2 \Rightarrow V_3'$	$\#A_2A_0D_0S_{43,j}$
V_3	V_0'	$V_3 \Rightarrow V_0'$	$\#A_3A_1D_0S_{44,j}$
	V_1'	$V_3 \Rightarrow V_1'$	$\#A_3A_2D_0S_{45,j}$
	V_2'	$V_3 \Rightarrow V_2'$	$\#A_3A_3D_0S_{46,j}$
	V_3'	$V_3 \Rightarrow V_3'$	$\#A_3A_0D_0S_{47,j}$

$$\begin{aligned} T_1 \;=\; & \{S_{4i+k+2n^2,\,j}(0) \\ & \mid 0 \leqq i \leqq n-1,\, 0 \leqq j \leqq m-1, \\ & 0 \leqq k \leqq n-1\} \end{aligned}$$

ここで n 入力に対して n^2 通りの入れ換えを考えるために n^2 個のコピーを実行する必要があるが，これは表 3 の真理値表を用いて 2 節で説明した *LogicOperation* により $O(1)$ ステップで実行可能である．

Step 3 では，試験管 T_1 に対して以下の試験管 T_2 を基本操作 *Merge*, *Annealing*, *Denaturation* を実行することにより，移動鎖を生成することができる．

$$\begin{aligned} T_2 \;=\; & \{\#A_iA_kD_0, \overline{A_iA_kD_0D_1A_{4i+k+2n^2}} \\ & \mid 0 \leqq i \leqq n-1,\, 0 \leqq k \leqq n-1\} \end{aligned}$$

最後に，移動鎖以外の不要な DNA 鎖を取り除くために $\{\overline{D_0D_1}\}$ で基本操作 *Separation* を実行すると試験管 T_1 は以下のようになる．

$$\begin{aligned} T_1 \;=\; & \{\#A_iA_kD_0S_{4i+k+2n^2,\,j} \\ & \mid 0 \leqq i \leqq n-1,\, 0 \leqq j \leqq m-1, \\ & 0 \leqq k \leqq n-1\} \end{aligned}$$

Step 4 では，T_1 を T_{output} に基本操作 *Copy* することで移動鎖の集合が T_{output} に保持され操作が完了する．

3.1.3 アルゴリズムの詳細

移動鎖を生成するアルゴリズムの詳細を以下に示す．ここで，

L は以下のような一本鎖 DNA の集合であり，表 3 の真理値表と対応している．

$$L = \{\overline{0\#D_0S_{4i+k+2n^2,\,j}(0)S_{k,\,j}(0)D_10\#},\ \overline{1\#D_0S_{4i+k+2n^2,\,j}(0)S_{k,\,j}(1)D_11\#},\ \overline{0\#D_1S_{4i+k+2n^2,\,j}(0)S_{k,\,j}(0)D_10\#},\ \overline{1\#D_0S_{4i+k+2n^2,\,j}(0)S_{k,\,j}(0)D_11\#},\ |\ 0 \leqq i \leqq n-1,\ 0 \leqq j \leqq m-1,\ 0 \leqq k \leqq n-1\}$$

Procedure $Move(T_{input}, T_{output})$**{**

/* *Step* 1 */
/* 試験管 T_{input}を
試験管 T_{tmp}にコピーする */
$Empty(T_{tmp})$;
$Copy(T_{input}, T_{tmp})$;

/* *Step* 2 */
/* 入力値を別アドレスにコピーする */
$Merge(T_1, T_{tmp})$;
$LogicOperation(T_1, L, T_1)$;

/* *Step* 3 */
/* 移動鎖を生成する */
$Merge(T_1, T_2)$;
$Annealing(T_1)$;
$Denaturation(T_1)$;
$Separation(T_1, \{\overline{D_0D_1}\}, T_{trash})$;

/* *Step* 4 */
/* 試験管 T_1を
試験管 T_{output}にコピーする */
$Copy(T_1, T_{output})$;

}

このアルゴリズムの各ステップは $O(1)$ ステップで実行可能である．また，このアルゴリズムで必要な DNA の種類は $O(mn^2)$ 種類である．このことから，次の定理を得ることができる．

[定理 1] アルゴリズム $Move$ は，$O(mn^2)$ 種類の DNA を用いることにより $O(1)$ ステップで実行可能である． □

3.2 ソートアルゴリズム

3.2.1 アルゴリズムの概要

入力サイズ n のソートを $O(\log n)$ ステップで実行するアルゴリズムの概要を以下に示す．このアルゴリズムでは，まず最初に，入力された n の数 x_0, x_1, ... x_{n-1} に対して，n^2 通りのすべての 2 つの数の組み合わせについて，減算を利用して大小を比較する．このとき，入力の値 x_i について，x_i のランク $rank(x_i)$ を以下のように定義する．

$$rank(x_i) = \{x_k | x_k \leqq x_i, 0 \leqq k \leqq n-1\}$$

つまり，x_i のランクとは，x_i が入力において何番目に小さいかを表す数字である．このアルゴリズムでは，各値についてこのランクを計算することにより，入力の値を格納するアドレスを計算し，そのアドレスに値を保存するアルゴリズムとなっている．例として，7, 1, 2, 4 という 4 つの値が入力された場合について考える．7 はアドレス 0 に，1 はアドレス 1 に，2 はアドレス 2 に，4 はアドレス 3 に格納されている．この場合，(7, 7), (7, 1), (7, 2), (7, 4), (1, 7), (1, 1), (1, 2), (1, 4), (2, 7), (2, 1), (2, 2), (2, 4), (4, 7), (4, 1), (4, 2), (4, 4) という 4 × 4=16 通りの全ての 2 つの数の組み合わせについて減算を利用して大小を比較する．7 は 1, 2, 4 の 3 つの数より大きく，1 は他のどの数よりも小さく，2 は 1 より大きく，4 は 1, 2 の 2 つの数より大きい．つまり，7, 1, 2, 4 の各ランクは 3, 0, 1, 2 ということになる．このとき，各ランクがその値を格納するアドレスを表しており，各ランクにしたがって値をアドレスに格納すると，最終的なソート結果である 1, 2, 4, 7 という結果が得られることになる．このアルゴリズムは，大きくわけて以下の 7 ステップから構成される．

表 3 コピーの操作を行う真理値表 1
Table 3 truth table 1

入力		出力	
V_i	$V_{4i+k+2n^2}$	V_i	$V_{4i+k+2n^2}$
0	0	0	0
0	1	1	1
1	0	0	0
1	1	1	1

Step 1 試験管 T_{input} を試験管 T_{tmp} にコピーする．

Step 2 入力要素のすべての対 $R = \{(i,j) \mid 0 \leqq i \leqq n-1, 0 \leqq j \leqq m-1\}$ に対して並列に減算 $V_i - V_j$ を実行し，演算結果を表す $O(mn^2)$ 種類の一本鎖 DNA を試験管 T_{sub} に保存する．

Step 3 減算結果の符号ビットを試験管 T_{sub} から試験管 T_{sign} に取り出す．

Step 4 値がすべて 0 である集合 $\{V_{i+n^2} \mid 0 \leqq i \leqq n^2\}$ に対して，取り出した各符号ビット $\{S_{i,m-1} \mid 0 \leqq i \leqq n^2\}$ が 1 ならば，$V_{i+n^2} = 1$ とする．

Step 5 各入力のランクを調べるために $n' = n$ とし，以下の Step 5.1 から Step 5.2 の操作を $\log n$ 回繰り返す．

Step 5.1 各入力のランクを調べるために符号ビットだったメモリ鎖を加算する．試験管 T_{sign2} 内の一本鎖 DNA の各アドレス対 $R = \{(i, i+\frac{nn'}{2}) \mid n^2 \leqq i \leqq n^2 + 2n' - 1\}$ に対して並列に加算 $V_i + V_{i+\frac{nn'}{2}}$ を実行し，加算結果を試験管 T_{sign2} に保存する．

Step 5.2 $n' = \frac{n'}{2}$ とする．

Step 6 各入力の値を，Step 5 で求めたランクで指定されたアドレスに保存する．(指定されたアドレスに保存されたメモリ鎖を以降，解候補と呼ぶことにする．)

Step 7 Step 6 で得られたを表す DNA 鎖を出力試験管に保存する．解候補を論理演算を使って入力値にコピーする．その結果を試験管 T_{output} に保存する．

上記のアルゴリズムにおいて，各試験管は以下のような役割を果たす．

- T_{input}：入力 DNA 鎖を保持
- T_{output}：出力 DNA 鎖を保持
- T_{tmp}：計算結果などを表す DNA 鎖を保持
- T_{sub}：減算結果を表す DNA 鎖を保持
- T_{sign}：数の符号ビットを表す DNA 鎖を保持
- T_{sign2}：符号ビットを数字として再構成した結果を表す DNA 鎖を保持
- T_{add}：加算結果を論理演算した結果を表す DNA 鎖を保持
- T_{comp1}，T_{comp2}，T_{comp3}：解を構成するために必要な DNA 鎖を保持
- T_{trash}：不要となった DNA 鎖を入れる

以下に各ステップの動作を説明する．

Step 1 は基本操作 *Copy* を用いることにより $O(1)$ ステップで実行可能である．

Step 2 では，$R = \{(i,j) \mid 0 \leqq i \leqq n-1, 0 \leqq j \leqq n-1\}$ に対して並列に減算 $V_i - V_j$ を実行し，その減算結果を V_{np+q} に保存する．. その結果は試験管 T_{sub} に保存されるが，試験管 T_{sub} の内容は以下のようになる．

$$\begin{aligned} T_{sub} = \{ & S_{np+q,j} \\ & \mid 0 \leqq p \leqq n-1, 0 \leqq q \leqq n-1, \\ & 0 \leqq j \leqq m-1, V_{np+q} = V_p - V_q\} \end{aligned}$$

この減算は 2 節で説明した *SubtractionOperation* により，$O(mn^2)$ 種類の DNA を用いることにより $O(1)$ ステップで実行可能である．

Step 3 では，減算結果の符号ビットを試験管 T_{sub} から試験管 T_{sign} に取り出すという操作を行うが，この操作は基本操作 *Separation* により $O(1)$ ステップで実行可能である．

Step 4 では，符号ビットだったメモリ鎖を計算できるようにするために，以下の試験管 T_{sign2} の最下位ビットに試験管 T_{sign} をコピーする．

$$T_{sign2} = \{V_i(0) \mid n^2 \leqq i \leqq 2n^2 - 1\}$$

これは 2 節で説明した *LogicOperation* により $O(1)$ ステップで実行可能である．

Step 5.1 は試験管 R で定義される加算 $V_i + V_{i+\frac{mn'}{2}}$ を並列に実行し，その減算結果を V_i に保存する．その結果は試験管 T_{sign2} に保存される．この加算は 2 節で説明した *AdditionOperation* により $O(1)$ ステップで実行可能である．

Step 6 では，試験管 T_{sign2} に対して以下の試験管 T_{comp1} を基本操作 *Merge*, *Annealing*, *Denaturation* を実行することにより，同アドレスのものをビット位置 j を基準に昇順に連結する．

$$T_{comp1} = \{\overline{S_i(k)} \mid n^2 \leqq i \leqq n^2 + n - 1, 0 \leqq k \leqq n-1\}$$

これらの連結した DNA 鎖だけが必要なので，試験管 T_{sign2} から試験管 T_{add} に長さ $7(m+1)$ で基本操作 *Selection* を行い，その後，試験管 T_{add} に $\overline{C_0C_1}$ で基本操作 *Separation* を実行すると，試験管 T_{add} は以下のようになる．

$$T_{add} = \{S_i(k) \mid n^2 \leqq i \leqq n^2 + n - 1, 0 \leqq k \leqq n-1\}$$

次に，試験管 T_{add} に対して以下の試験管 T_{comp2} を基本操作 *Merge*, *Annealing*, *Cleavage*, *Denaturation* を実行することにより，どのアドレスからどのアドレスに書き換えたらいいかという印を作ることができる．図 1(a) の $\overline{\#A_iA_jD_0}$ がその印である．

$$\begin{aligned} T_{comp2} = \{ & \overline{\#A_iA_jD_0S_{i+n^2}(k)}, D_0 \\ & \mid 0 \leqq i \leqq n-1, 0 \leqq j \leqq n-1, \\ & 0 \leqq k \leqq n-1\} \end{aligned}$$

これら以外の不要な DNA 鎖を取り除くために試験管 T_{add} に $\{D_0, C_0C_1, \overline{C_0C_1}\}$ で基本操作 *Separation* を実行すると，試験管 T_{add} は以下のようになる．

$$T_{add} = \{\overline{\#A_iA_jD_0} \mid 0 \leqq i \leqq n-1, 0 \leqq j \leqq n-1\}$$

次に，以下の試験管 T_{comp3} は前述のアルゴリズム *Move* にしたがって準備された移動鎖が入っているとする．

$$\begin{aligned} T_{comp3} = \{ & \#A_iA_jD_0S_{4i+k+2n^2,j} \\ & \mid 0 \leqq i \leqq n-1, 0 \leqq j \leqq n-1, \\ & 0 \leqq k \leqq n-1\} \end{aligned}$$

試験管 T_{add} に対して試験管 T_{comp3}，$\{\overline{D_1}\}$ を基本操作 *Merge*, *Annealing*, *Cleavage*, *Denaturation* を実行することにより，解候補を作ることができる．図 1(b) の $S_{i,j}$ が解候補である．これら以外の不要な DNA 鎖を取り除くために試験管 T_{add} に $\{\overline{D_1}, \#, \overline{\#}\}$ で基本操作 *Separation* を実行すると，以下のようになる．

$$\begin{aligned} T_{add} = \{ & S_{4i+k+2n^2,j} \\ & \mid 0 \leqq i \leqq n-1, 0 \leqq j \leqq m-1, \\ & 0 \leqq k \leqq n-1\} \end{aligned}$$

Step7 では試験管 T_{add} に対して試験管 T_{input} をマージし，試験管 T_{output} に対して試験管 T_{add} に 2 節で説明した *LogicOperation* を実行する．

3.2.2 アルゴリズムの詳細

ソートを行うアルゴリズムの詳細を以下に示す．L_r 及び L_c は以下のような一本鎖 DNA の集合であり，表 4，表 5 の真理値表と対応している．

$$\begin{aligned} L_r = \{ & \overline{0\#D_0S_{i,\,m-1}(0)S_{i+n^2,0}(0)D_10\#}, \\ & \overline{1\#D_0S_{i,\,m-1}(0)S_{i+n^2,0}(1)D_11\#}, \\ & \overline{1\#D_0S_{i,\,m-1}(1)S_{i+n^2,0}(0)D_11\#}, \\ & \overline{1\#D_0S_{i,\,m-1}(1)S_{i+n^2,0}(1)D_11\#}, \\ & \mid 0 \leqq i \leqq n^2 - 1\} \end{aligned}$$

$$
\begin{aligned}
L_c \;=\; & \{\overline{0\#D_0S_{i+n^2,j}(0)S_{k,j}(0)D_10\#},\\
& \overline{1\#D_0S_{i+n^2,j}(0)S_{k,j}(1)D_11\#},\\
& \overline{0\#D_0S_{i+n^2,j}(0)S_{k,j}(0)D_10\#},\\
& \overline{1\#D_0S_{i+n^2,j}(0)S_{k,j}(0)D_11\#},\\
& \mid 0 \leqq i \leqq n^2-1,\ 0 \leqq j \leqq m-1,\\
& 0 \leqq k \leqq n-1\}
\end{aligned}
$$

また，R_a は以下のような加算を行うアドレス対を示す．

$$
R_a = \{(i+n^2,\ i+n^2+\frac{n'n}{2}) \mid 0 \leqq i \leqq 2n'-1\}
$$

Procedure $Sort(TestTube\ T_{input}, TestTube\ T_{output})$**{**

/* *Step* 1 */
/* 試験管 T_{input} を試験管 T_{tmp} にコピーする */
$Copy(T_{input}, T_{tmp})$;

/* *Step* 2 */
/* 全ての 2 つの数の組み合わせを
比較するために減算を行う */
$SubtractionOperation(T_{tmp}, R_s, T_{sub})$;

/* *Step* 3 */
/* 最上位ビットを取り出す */
$Separation(T_{sub}, \{B_{m-1}\}, T_{sign})$;

/* *Step* 4 */
/* 試験管 T_{sign2} の最下位ビットに
試験管 T_{sign} をコピーする */
$LogicOperation(T_{sign}, L_r, T_{sign2})$;

/* *Step* 5 */
/* ランクを決定する */
for$(k=0\ ;\ k \geqq \log n\ ;\ k{+}{+})${
/* *Step* 5.1 */
/* ランクを調べるために加算する */
AdditionOperation
$(T_{sign2}, R_a, T_{sign2})$;
/* *Step* 5.2 */
/* $n' = \frac{n'}{2}$ とする */
$n' = \frac{n'}{2}$;
}

/* *Step* 6 */
/* 同アドレスのメモリ鎖の集合を
ビット位置である j を基準に
昇順に連結させる */
$Merge(T_{sign2}, T_{comp1})$;
$Annealing(T_{sign2})$;
$Denaturation(T_{sign2})$
$Selection(T_{sign2}, 7m, T_{add})$;
$Separation(T_{add}, \{\overline{C_0C_1}\}, T_{trash})$;

$$
\left[\frac{D_0S_i}{\#A_iA_jD_0S_i}\right]
$$
$\Rightarrow$ (*Cleavage*) $\Rightarrow$ $\left[\frac{D_0}{\#A_iA_jD_0}\right], \left[\frac{S_i}{\overline{S_i}}\right]$
$\Rightarrow$ (*Denaturation, Separation*) $\Rightarrow$ $\overline{\#A_iA_jD_0}$

(a)

$$
\left[\frac{\#A_iA_jD_0S_{i,j}}{\#A_iA_jD_0\overline{D_1}}\right]
$$
$\Rightarrow$ (*Cleavage*) $\Rightarrow$ $\left[\frac{\#A_iA_jD_0}{\#A_iA_jD_0}\right], \left[\frac{S_{i,j}}{\overline{D_1}}\right]$
$\Rightarrow$ (*Denaturation, Separation*) $\Rightarrow$ $S_{i,j}$

(b)

図 1　各ステップにおける DNA 鎖
Fig. 1　DNA strands of each step

表 4　コピーの操作を行う真理値表 2
Table 4　truth table 2

入力		出力	
V_i	V_{i+n^2}	V_i	V_{i+n^2}
0	0	0	0
0	1	1	1
1	0	0	0
1	1	1	1

表 5　コピーの操作を行う真理値表 3
Table 5　truth table 3

入力		出力	
$V_{4i+k+2n^2}$	V_k	$V_{4i+k+2n^2}$	V_k
0	0	0	0
0	1	1	1
1	0	0	0
1	1	1	1

/* ランクをもとに保存するアドレスを
指定する DNA 鎖を生成する */
$Merge(T_{add}, T_{comp2})$;
$Annealing(T_{add})$;
$Cleavage(T_{add}, D_0D_1)$;
$Denaturation(T_{add})$;
$Separation(T_{add}, \{D_0, C_0C_1, \overline{C_0C_1}\}, T_{trash})$;

/* 生成した DNA 鎖をもとに
解候補をつくる */
$Merge(T_{add}, T_{comp3})$;
$Annealing(T_{add})$;
$Cleavage(T_{add}, D_0D_1)$;
$Denaturation(T_{add})$;
$Separation(T_{add}, \{\overline{D_1}, \#, \overline{\#}\}, T_{trash})$;

```
    /* Step 7 */
      /* 解候補を論理演算でコピーし出力する */
      Merge(T_add, T_input);
      LogicOperation(T_add, L_c, T_output);
}
```

以下に，上記のアルゴリズムにおける注意点を述べる．論理演算における真理値表を表す一本鎖 DNA の集合 L とは，使用する対についてのみ示すのが一般的である．提案したアルゴリズムでは，ソートを実行するために論理演算を行う際に，実行するときの状態が一般的な場合と異なり，存在しないメモリ鎖のついに対しても論理演算が定義されている．例えば，アドレス 0,1 を表すメモリ鎖は存在するがアドレス 2,3 を表すメモリ鎖が存在しない場合にも，対の集合 {(0,1), (0,2), (0,3)} に対して論理演算が定義されている．本研究では，この状況で論理演算を実行した場合，(0,1) の論理演算のみが実行されると仮定しているが，この仮定が一般的に成り立つか否かは検討が必要である．そこで，[6] で提案されている論理演算のアルゴリズムを検証したところ，上記の様な仮定が一般的な論理演算について成り立つことを確認できた．

最後に上記アルゴリズムの計算量の検証を行う．Step 2 では，$O(n^2)$ 組の各アドレス対が保持している値に対して並列に減算を行う．$O(n)$ 組の値に対する 2 節で説明した *SubtractionOperation* は，$O(mn)$ 種類の DNA を用いることにより $O(1)$ ステップで実行可能であるので，Step 2 は，$O(mn^2)$ 種類の DNA を用いることにより $O(1)$ ステップで実行可能である．Step 5 の *for* 文の内部において，k を繰り返し回数 $(0 \leq k < \log n)$ とすると，必要な DNA の種類は $O(m \times \frac{n}{2^k})$ 種類である．したがって，Step 5 を $\log n$ 回繰り返した場合に必要となる DNA の種類は $O(mn)$ 種類である．また，*for* 文の内部の操作は 2 節で説明した基本操作の定数回の実行により $O(1)$ ステップで実現できる．したがって，次のような定理を得ることができる．

[定理 2] n 個の m ビットの 2 進数に対するソートを行うアルゴリズム *Sort* は，$O(mn^2)$ 種類の DNA を用いることにより $O(\log n)$ ステップで実行可能である． □

4. ま と め

本論文では，DNA を用いて表現された 2 進数の集合に対して，ソートを行うアルゴリズムを提案した．このアルゴリズムの入力は $O(mn)$ 種類の一本鎖 DNA で表現されている n 個の m ビットの 2 進数の集合であり，各入力値を比較することでランクを決定し，それに対して各入力値をソートし，出力するというものであり，$O(mn^2)$ 種類の DNA を用いることにより $O(\log n)$ ステップで実行可能である．

本論文で提案したアルゴリズムは理論に基づくものであり，実際の DNA を使用した実験は行っていない．提案したアルゴリズムを実現するために，基本操作におけるエラー率を考慮し，計算精度の高いアルゴリズムの提案を行うこと，さらに，もっと準備が容易でかつ，さらに高速なソートアルゴリズムを考えることが今後の研究課題である．

文　献

[1] L.M.Adleman. Molecular computation of solutions to combinatorial problems. *Science*, Vol. 266, pp. 1021–1024, 1994.

[2] L.M.Adleman. Computing with DNA. *Scientific American*, Vol. 279, No. 2, pp. 54–61, 1998.

[3] E.B.Baum and D.Boneh. Running dynamic programming algorithms on a DNA computer. *Proceedings of the Second Annual Meeting on DNA Based Computers*, 1996.

[4] P.Frisco. Parallel arithmetic with splicing. *Romanian Journal of Infomation Science and Technology(ROMJIST)*, Vol. 2, No. 3, pp. 113–128, 2000.

[5] A.G.Frutos, Q.Liu, A.J.Thiel, A.M.W.Sanner, A.E.Condon, L.M.Smith, and R.M.Corn. Demonstration of a word design strategy for DNA computing on surfaces. *Nucleic Acids Research*, Vol. 25(23), pp. 4748–4757, 1997.

[6] A. Fujiwara, K. Matsumoto, W. Chen. Procedures for Logic and Arithmetic operations with DNA Molecules. *International Journal ofFoundations of Computer Science*, Vol. 15, No. 3, pp. 461-474, 2004.

[7] F.Guarnieri, M.Fliss, and C.Bancroft. Making DNA add. *Science*, Vol. 273(5272), pp. 220–223, 1996.

[8] V.Gupta, S.Parthasarathy, and M.J.Zaki. Arithmetic and logic operations with DNA. *Proceedings of the 3rd DIMACS Workshop on DNA Based Computers*, pp. 212–220, 1997.

[9] H.Hug and R.Schuler. DNA-based parallel computation of simple arithmetic. *Proceedings of the 7th International Meeting on DNA Based Computers(DNA7)*, pp. 159–166, 2001.

[10] R.J.Lipton. DNA solution of hard computational problems. *Science*, Vol. 268, pp. 542–545, 1995.

[11] R.B.Merrifield. Solid phase peptide synthesis. I. the synthesis of a tetrapeptide. *Journal of the American Chemical Society*, Vol. 85, pp. 2149–2154, 1963.

[12] Q.Ouyang, P.D.Kaplan, S.Liu, and A.Libchaber. DNA solution of the maximal clique problem. *Science*, Vol. 278, pp. 446–449, 1997.

[13] Z.F.Qiu and M.Lu. Arithmetic and logic operations for DNA computers. *Proceedings of the Second IASTED International conference on Parallel and Distributed Computing and Networks*, pp. 481–486, 1998.

[14] J.H.Reif. Parallel biomolecular computation: Models and simulations. *Algorithmica*, Vol. 25, No. 2-3, pp. 142–175, 1995.

[15] A.Suyama, N.Nishida, K.Kurata, and K.Omagari. Gene expression analysis by DNA computing. *Computational Molecular Biology*, pp. 12–13, 2000.

[16] H.Yoshida and A.Suyama. Solution to 3-SAT by breadth first search. *American Mathematical Society*, pp. 9–22, 2000.