# 最適部分列問題に対する統一的な並列アルゴリズム

森畑 明昌

最適部分列問題は、要素列が与えられたとき、その連続した部分列であって、与えられた条件の下で最適なものを求めるものであり、データベースからの相関ルールの発見、画像処理、DNA の解析など、様々な応用をもつ。広い範囲の最適部分列問題に対して線形時間の逐次アルゴリズムが知られているが、その多くについては効率の良い並列アルゴリズムは知られていない。本稿では、最適部分列問題に対して効率の良い並列アルゴリズムを与えるための一般的なアプローチを示す。骨子は最適部分列問題を saddleback search と呼ばれる問題に帰着することである。このために、まず、saddleback search の効率の良い分割統治並列アルゴリズムを示す。さらに、広い範囲の最適部分列問題を saddleback search に帰着する手法を示す。

Optimal segment problems require us to locate the best consecutive subsequence of a sequence according to a given criterion of optimality. They have several applications including association-rule mining, image processing, and DNA analyses. For a large class of them, linear-time sequential algorithms are known; however, it is largely unknown whether they have efficient parallel algorithms as well. In this paper, we propose an approach to efficient parallel algorithms for optimal segment problems. Our approach is based on a reduction from optimal segment problems to saddleback search problems. Our major contributions are the following two. First, we propose an efficient parallel saddleback search algorithm. Second, we study a method of reducing a large class of optimal segment problems to saddleback search problems.

## 1 はじめに

最適部分列問題は、要素列が与えられたとき、その連続した部分列であって、与えられた条件の下で最適なものを求めるものである。最も基本的な問題は最大和部分列問題であり、Bentley の Programming Pearls [2, 3] で紹介されて有名となった。Bentley は画像処理に関する問題として紹介しているが、似た問題は様々な文脈で現れている。Fukuda ら [9, 10] はデータベースからの知識発見に最適部分列問題が利用できることを指摘しており、また、Huan ら [11, 12] は DNA の解析への応用を述べている。

最適部分列問題に対する逐次アルゴリズムについては、上記のものを含め、多くの研究がある [5, 15, 16, 20, 21]。よく用いられる手法は動的計画法と windowing であり、これらの手法により広い範囲の最適部分列問題を効率よく解くことができることが知られている。動的計画法 [5, 16] は、最適性の原理に基づく枝刈りを行いながら、最適解になりうる解候補を列挙する。これに対し、windowing [15, 20, 21] では、現在の解候補から得られる情報なども用い、より注意深く解候補の延長・短縮を繰り返し、最適解を得る。一般に、動的計画法は広い範囲の問題に適用でき、系統的な導出も比較的容易だが、問題によっては windowing に比べ効率が悪くなる。

一方、最適部分列問題に対する並列アルゴリズムについての研究は比較的少ない。最大和部分列問題 [18] や長さ制約つき最大和部分列問題 [14] など、個々の問題を扱った研究はある。また、動的計画法で解くことのできる問題に対しての統一的なアプローチも存在する [7, 8, 13]。しかし、著者の知る限り、windowing でのみ効率の良い解法が知られているクラスの問題

A Parallel Algorithm for Optimal Segment Problems.
Akimasa Morihata, 東京大学大学院総合文化研究科, Graduate School of Arts and Sciences, The University of Tokyo.

に対する、統一的な並列アルゴリズムは知られていない。

本稿では、広い範囲の最適部分列問題、特に Zantema [20] や Zhao ら [21] が議論した、windowing による線形時間解法が知られている問題に対して、効率の良い並列アルゴリズムを導くための手法を与える。提案手法の骨子は、前原によって指摘された最適部分列問題と saddleback search との相似 [22] に基づき、最適部分列問題を Bird による分割統治 saddleback search アルゴリズム [6] に帰着することである。このアプローチを採るにあたっては 2 点の技術的な問題を解決しなければならない。まず、Bird のアルゴリズムは逐次評価を前提としており、そのまま並列アルゴリズムとして利用すると効率が悪い。そのため、並列化に適した分割統治アルゴリズムへの変換が必要である。さらに、少なくない最適部分列問題は直裁には saddleback search に帰着できない。前原はいくつかのケースについて議論を行っているが、を取り扱うための手法を示しているが、その一般性は不明瞭であった。しかも、前原の議論は windowing を用いた逐次アルゴリズムのためのものであり、分割統治並列アルゴリズムを用いる場合には直接は利用できない。以上の問題点を解決するのが本稿の目標である。

本稿の技術的な貢献は以下にまとめられる。以下プロセッサ数を $\mathrm{P}$ とする。

- Bird のアルゴリズムを改良し、$n \times m$ の二次元配列 ($n \leq m$) に対する saddleback search を $\mathrm{O}(n/\mathrm{P}\log(m/n) + \log m)$ 時間で行う並列アルゴリズムを与える。この計算量は $\mathrm{P} \in \mathrm{O}(n\log(m/n)/\log m)$ の範囲では $\mathrm{O}(n/\mathrm{P}\log(m/n))$ であり最適である。
- 最適部分列問題を saddleback search に帰着する手法を示す。さらに、直裁には帰着できない問題に対しても、それが saddleback search に基づき効率よく解けるための十分条件を与える。

これらの技術を組み合わせることで、広い範囲の最適部分列問題に対し、$\mathrm{P} \ll n$ の仮定の下で、$\mathrm{O}(n/\mathrm{P})$ 時間で解を求める並列アルゴリズムが得られる。よって、高々定数倍のオーバーヘッドを除けば、この並列アルゴリズムは逐次アルゴリズムに比べてプロセッサ数に比例する並列速度向上が期待できる。

## 2 準備

### 2.1 列の操作

本稿では、列の操作に以下の記法を用いる。要素 $a_0, \ldots, a_n$ からなる列を $[a_0, \ldots, a_n]$ で表す。特に断らない限り空列は考えない。列 $x$ および列 $y$ に対し、その連結を $x +\!\!+ y$ で表す。列 $x$ の長さを $|x|$ で表す。列 $x$ の $i$ 番目から $j$ 番目まで $(i \leq j)$ の要素からなる部分列を $x[i:j]$ で表す。列 $x$ の先頭部分列の集合を $\mathit{inits}(x)$、末尾部分列の集合を $\mathit{tails}(x)$、部分列の集合を $\mathit{segs}(x)$ で表す。それぞれの定義を以下に示す。

$$\mathit{inits}(x) = \{x[0:j] \mid 0 \leq j \leq |x| - 1\}$$
$$\mathit{inits}(x) = \{x[i:|x|-1] \mid 0 \leq i \leq |x| - 1\}$$
$$\mathit{segs}(x) = \{x[i:j] \mid 0 \leq i \leq j \leq |x| - 1\}$$

### 2.2 リスト準同型と範囲問い合わせ

列に対して値を計算する際には、リスト準同型 [4] を用いるのが便利である。リスト準同型は、要素の重み和、列の長さ、最大要素の値や最小要素の値など、標準的な計算を表現できる。

**定義 1** リスト準同型 $(\!|\oplus, f|\!)$ とは、列に対して以下の等式で定義される関数である。

$$(\!|\oplus, f|\!)([a]) = f(a)$$
$$(\!|\oplus, f|\!)(x +\!\!+ y) = (\!|\oplus, f|\!)(x) \oplus (\!|\oplus, f|\!)(y)$$

ここで $\oplus$ は結合的な演算子である。

リスト準同型は特に並列計算に有用であることが知られている。以下はよく知られた事実である。

**定理 2** 列に対する関数 $h = g \circ (\!|\oplus, f|\!)$ を考える。$g$、$f$、$\oplus$ が定数時間関数であるとき、$h(x)$、$\{h(y) \mid y \in \mathit{inits}(x)\}$、$\{h(y) \mid y \in \mathit{tails}(x)\}$ はいずれも $\mathrm{O}(|x|/\mathrm{P} + \log \mathrm{P})$ 時間で並列計算できる。

一般に、与えられた列の部分列に対して何らかの計算を繰り返し行うことを範囲問い合わせ (range query) [1, 17] と呼ぶ。定理 2 は、先頭部分列や末尾部分列に対する範囲問い合わせが効率よく並列計算できることを示している。より一般的な事実として、本稿では以下の事実を用いる。なお、詳細は本稿の範囲を超えるためここでは述べない。

**定理 3** 列 $x$、列に対する関数 $h = g \circ (\![\oplus, f]\!)$、および問い合わせ範囲の集合 $S \subseteq \{(i,j) \mid 0 \le i \le j \le |x|-1\}$ が与えられたとする。このとき、$k = \lfloor |x|/\mathrm{P} \rfloor$ として、局所的な問い合わせ $S' \subseteq S$ を以下で定義する。

$$S' = \{(i,j) \mid (i,j) \in S \wedge \lfloor i/k \rfloor = \lfloor j/k \rfloor\}$$

$S'$ に対する問い合わせ結果、すなわち $\{h(x[i:j]) \mid (i,j) \in S'\}$ を得るのに要する時間が $T(|x|, \mathrm{P})$ であるとき、$S$ に対する問い合わせ結果 $\{h(x[i:j]) \mid (i,j) \in S\}$ を $\mathrm{O}(|x|/\mathrm{P} + T(|x|,\mathrm{P}) + |S \setminus S'| + \log \mathrm{P})$ 時間で計算できる。

### 2.3 単調性

我々の議論では、列に対する関数の単調性が重要な役割を果たす。本稿では、最適部分列問題での必要性をふまえた、限定的な定義を行う。2 つの列 $y$ および $z$ について、$y \in \mathit{inits}(z) \vee z \in \mathit{inits}(y)$ であるとき、$y \overset{L}{\sim} z$ と記述する。同様に、$y \in \mathit{tails}(z) \vee z \in \mathit{tails}(y)$ であるとき、$y \overset{R}{\sim} z$ と記述する。$y$ と $z$ が共に $x$ の部分列であるとき、$y \overset{L}{\sim} z$ は $y$ と $z$ の左端が、$y \overset{R}{\sim} z$ は右端がそれぞれ一致していることを意味する。

**定義 4** 関数 $\phi$ が単調であるとは、以下の性質を共に満たすことである。

- $y \overset{L}{\sim} z$ かつ $\phi(y) \ge \phi(z)$ ならば、$\phi(l +\!\!+ y) \ge \phi(l +\!\!+ z)$。
- $y \overset{R}{\sim} z$ かつ $\phi(y) \ge \phi(z)$ ならば、$\phi(y +\!\!+ r) \ge \phi(z +\!\!+ r)$。

我々はしばしば関数が単調であることを要求する。このことの確認は、多くの場合でそれほど難しくない。

まず、関数 $\phi = g \circ (\![\oplus, f]\!)$ を満たす場合、$\oplus$ および $g$ が単調であれば $\phi$ は単調である。例えば、$\mathit{sum} = (\![+, \mathit{id}]\!)$・$\mathit{length} = (\![+, \lambda x.\ 1]\!)$・$\mathit{maximum} = (\![\mathit{max}, \mathit{id}]\!)$ などはいずれも単調である。

次に関数 $\phi$ が単調な関数 $\phi_1$ と $\phi_2$ を用いて $\phi(x) = \phi_1(x) \oslash \phi_2(x)$ と表される場合を考える。このような場合、$\phi_1$ および $\phi_2$ がポジティブかネガティブかが問題となる。$\phi_i$ がポジティブとは、$\phi_i(x) \ge \phi_i(y) \Rightarrow \phi(x) \ge \phi(y)$ を満たすことであり、ネガティブとは、$\phi_i(x) \ge \phi_i(y) \Rightarrow \phi(x) \le \phi(y)$ を満たすことである。ポジティブかネガティブかは、多くの場合は演算子 $\oslash$ から容易に判断できる。このとき、端的に言えば、各 $\phi_i$ の値の増減が、ポジティブかネガティブかもふまえれば一貫性を持っていれば $\phi$ は単調である。

**補題 5** 関数群 $\{\phi_i \mid i \in I\}$ によって関数 $\phi$ が定義されているとする。このとき、$y \overset{L}{\sim} y'$ または $y \overset{R}{\sim} y'$ なる任意の $y$ および $y'$ について以下が満たされれば、$\phi$ は単調である。

- あるポジティブな $\phi_i$ について $\phi_i(y) < \phi_i(y')$ ならば、全ての $\phi_j$ について、$\phi_j$ がポジティブなら $\phi_j(y) \le \phi_j(y')$、ネガティブなら $\phi_j(y) \ge \phi_j(y')$。
- あるネガティブな $\phi_i$ について $\phi_i(y) < \phi_i(y')$ ならば、全ての $\phi_j$ について、$\phi_j$ がポジティブなら $\phi_j(y) \ge \phi_j(y')$、ネガティブなら $\phi_j(y) \le \phi_j(y')$。

**証明** 単調性の定義から明らかである。

単調性は、最大の先頭部分列・末尾部分列を効率よく行うための求めるための十分条件でもある。

**定理 6** $h$ の値が最大の列を返す関数を $\max_h$ と表記する。$h$ が単調のとき、以下の等式が成り立つ。

$$\max_h \circ \mathit{inits} = \pi_1 \circ (\![\otimes, \lambda x.\ ([x],[x])]\!)$$
$$\max_h \circ \mathit{tails} = \pi_1 \circ (\![\oplus, \lambda x.\ ([x],[x])]\!)$$

**where**

$$(m_l, l) \otimes (m_r, r) = (\max_h\{m_l +\!\!+ r, m_r\}, l +\!\!+ r)$$
$$(m_l, l) \oplus (m_r, r) = (\max_h\{m_l, l +\!\!+ m_r\}, l +\!\!+ r)$$
$$\pi_1\ (a,b) = a$$

## 3 Saddleback Search

まずは、saddleback seach 問題と、それに対する Bird の解法を紹介する。以降の議論の簡便のため、以下では通常議論されるものよりも一般的な問題を考える

**定義 7 (Saddleback Search)** *Saddleback search* $\mathcal{SS}(A,h,p)$ は *3* つ組である。$A$ は *2* 次元配列である。以下、特に断らない限り、$A$ は $m \times n$ $(n \le m)$ であるとする。$h$ は評価関数、$p$ は制約条件であり、以下の性質を満たす。

- $i \le i'$、$j \le j'$ のとき $h(A[i][j]) \le h(A[i'][j'])$。
- $i \le i'$、$j \le j'$ のとき $p(A[i'][j']) \Rightarrow p(A[i][j])$。

$\mathcal{SS}(A,h,p)$ の解は

$$\max\{h(A[i][j]) \mid 0 \le i < m \wedge 0 \le j < n \wedge p(A[i][j])\}$$

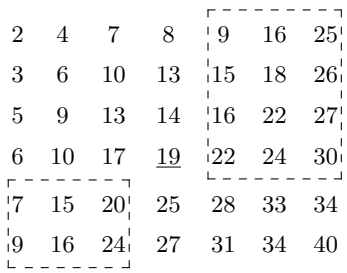


**図 1　分割統治 saddleback search アルゴリズムの概要**

すなわち $p(A[i][j])$ を満たす範囲での $h(A[i][j])$ の最大値である。

一般的な saddleback search では、二次元配列中から特定の値 $c$ があるかどうかを調べる。この場合、$h$ は恒等関数、$p$ は値 $c$ 以下かどうかを判定する述語ととればよい。

### 3.1　Saddleback search に対する分割統治並列アルゴリズム

Saddleback search に対しては、$O(n+m)$ 時間の古典的な解法がよく知られている。Bird [6] は、この問題に対する $O(n\log(m/n))$ 時間の解法を示した。これを以下に示す。

1. $j=\lfloor m/2\rfloor$ 列目で、$p$ を満たし $h$ が最大となる要素 $e^*$ を二分探索で発見する。
2. $e^*$ が $i$ 行目の要素だとする。このとき、$A[0][m-1]$ から $A[i+1][j-1]$ までの矩形領域、及び $A[i][j+1]$ から $A[n-1][0]$ までの矩形領域、の両者の最適値を求める ($v_1$ および $v_2$ とする) 。
3. $h(e)$、$v_1$、$v_2$ のうち最大のものが最適値である。

図 1 に概要を示す。図に示した $6\times 7$ の配列が与えられており、値 20 を発見したいとする。まず、$j=3$ 列目の要素で 20 より小さく最大の要素を発見する。この場合は $i=3$ 列目の 19 である。このとき、19 の左下ないし右上の、破線で囲まれた 2 領域が解がある可能性のある領域であり、再帰的に探索される。

Bird のアルゴリズムは逐次アルゴリズムとしては高速である。しかし、分割統治で得られる 2 つの矩形領域の大きさが必ずしも均等ではないため、そのまま並列アルゴリズムとして実行すると効率が良くない。以下ではコスト最適な並列アルゴリズムを示す。逐次での時間計算量が $f(n)$、並列アルゴリズムの時間計算量が $O(f(n)/\mathrm{P})$ であるとき、これをコスト最適であるという。以下では、説明の簡単のため、$n$ および $m$ は $\mathrm{P}+1$ で割り切れるとする。

1. $m/(\mathrm{P}+1),\ldots,\mathrm{P}m/(\mathrm{P}+1)$ 行目のそれぞれについて、$p$ を満たし $h$ が最大となる要素 $e_1,\ldots,e_{\mathrm{P}}$ を二分探索で発見する。
2. $n/(\mathrm{P}+1),\ldots,\mathrm{P}n/(\mathrm{P}+1)$ 列目のそれぞれについて、$p$ を満たし $h$ が最大となる要素 $e'_1,\ldots,e'_{\mathrm{P}}$ を二分探索で発見する。
3. $e_1,\ldots,e_{\mathrm{P}}$ および $e'_1,\ldots,e'_{\mathrm{P}}$ を、その行をもとに整列し、$e^*_1,\ldots,e^*_{2\mathrm{P}}$ とする。
4. 各 $i$ について $e^*_i$ から $e^*_{i+1}$ までの矩形領域の最適値を求め、$v_{i+1}$ とする。また、これに加え、$A[0][m-1]$ から $e^*_0$ まで、$e^*_{2\mathrm{P}}$ から $A[n-1][0]$ までの矩形領域の最適値を求め、それぞれ $v_0$・$v_{2\mathrm{P}+2}$ とする。
5. $v_0,\ldots,v_{2\mathrm{P}+2}$ のうち最大のものが最適値である。

**定理 8**　$\mathcal{SS}(A,h,p)$ は、$h$ および $p$ が定数時間であれば、$O(n/\mathrm{P}\log(m/n)+\log m)$ 時間で実行できる。ただし $n\geq \mathrm{P}$ とする。

**証明**　手順 *(1)* および手順 *(2)* は、それぞれの二分探索を並列に行うことで、それぞれ $O(\log n)$ 時間、$O(\log m)$ 時間で実現できる。手順 *(3)* は、$e_1,\ldots,e_{\mathrm{P}}$ および $e'_1,\ldots,e'_{\mathrm{P}}$ がそれぞれ整列されていること、また $e_1,\ldots,e_{\mathrm{P}}$ の行番号が既知であることをふまえると、$e'_1,\ldots,e'_{\mathrm{P}}$ の行番号からバケツソートの要領で $O(\log \mathrm{P})$ で整列が可能である。手順 *(4)* で扱うのは、高々$n/(\mathrm{P}+1)+1$ 行 $m/(\mathrm{P}+1)+1$ の矩形領域 $2\mathrm{P}+2$ 個であり、それぞれを並列に逐次アルゴリムで処理すれば $O(n/\mathrm{P}+\log(m/n))$ 時間で扱うことができる。手順 *(5)* は標準的な並列アルゴリズムにより $O(\log \mathrm{P})$ 時間で十分である。

## 4　最適部分列問題の saddleback search への帰着

本節では、最適部分列問題について議論する。まずは最適部分列問題を定義する。

**定義 9**　最適部分列問題 $\mathcal{OSP}(x,h,p)$ は *3* つ組であ

る。$x$ は要素列である。特に断らない限り、この長さを $\mathrm{N}$ とする。$h$ は評価関数である。$p$ は制約条件である。$\mathcal{OSP}(x,h,p)$ の解は

$$\max\{h(x[i:j]) \mid 0 \le i \le j < \mathrm{N} \wedge p(x[i:j])\}$$

すなわち $p(x[i:j])$ を満たす範囲での $h(x[i:j])$ の最大値である。

それでは、前原 [22] の述べた関係に従い、最適部分列問題 $\mathcal{OSP}(x,h,p)$ を saddleback search $\mathcal{SS}(A,h,p)$ に帰着する基本的な戦略は、$x[i:j]$ を $A[i][j]$ に対応させることである。この対応の元で、$h$ や $p$ が saddleback search に要求される要件を満たすためには以下の条件が成り立つ必要がある。

**定義 10** 列に対する評価関数 $h$ が *longer-better* であるとは、以下を満たすことである。

$$\forall x,y,z:\ h(x) \le h(y +\!\!+ x +\!\!+ z)$$

**定義 11** 列に対する制約条件 $p$ が *segment-closed* であるとは、以下を満たすことである。

$$\forall x,y,z:\ p(y +\!\!+ x +\!\!+ z) \Rightarrow p(x)$$

最適部分列問題 $\mathcal{OSP}(x,h,p)$ は、$h$ が longer-better かつ $p$ が segment-closed であれば、saddleback search に帰着できる。

**定理 12** 最適部分列問題 $\mathcal{OSP}(x,h,p)$ に対し、$A$・$h'$・$p'$ を以下の通り定義する。ただし $h$ は *longer-better*、$p$ は *segment-closed* であるとする。

- 任意の $0 \le i \le j < \mathrm{N}$ に対し、$A[i][j] = x[i:j]$。また、$j < i$ に対し、$A[i][j] = \bot$。ここで $\bot$ は $x$ の部分列とは区別できる特別な値である。
- $h'(\bot) = -\infty$ かつ $y \neq \bot$ のとき $h'(y) = h(y)$。
- $p'(\bot) = \mathsf{True}$ かつ $y \neq \bot$ のとき $p'(y) = p(y)$。

このとき、$\mathcal{SS}(A,h',p')$ は *saddleback search* であり、その解は、$\mathcal{OSP}(x,h,p)$ に解があればそれと一致する。なお、後者に実行可能解がなければ、前者の解は $-\infty$ である。

**証明** 自明。

しかし、saddleback search に帰着するだけでは不十分である。帰着先での評価関数や制約条件は列を入力とするため、これを単純に評価すると、各要素についての計算に列長に比例する時間がかかってしまう。しかし、リスト準同型で定義される評価関数・制約条件については、定理 3 を用いてこれを改善できる。

**定理 13** 最適部分列問題 $\mathcal{OSP}(x,h,p)$ が以下の条件を満たすとする。

- $h = \rho_1 \circ (\!|\oplus, f|\!)$ は *longer-better*
- $p = \rho_2 \circ (\!|\otimes, g|\!)$ は *segment-closed*
- $\rho_1$、$\rho_2$、$\oplus$、$\otimes$、$f$、$g$ は定数時間で評価可能

このとき、$\mathcal{OSP}(x,h,p)$ の解を $\mathrm{O}(T(\mathrm{N}/\mathrm{P}) + \log \mathrm{N})$ 時間で発見できる。ここで $T(m)$ は長さ $m$ の列 $y$ に対して *1* プロセッサで最適部分列問題 $\mathcal{OSP}(y,h,p)$ の解を求めるのに要する時間であり、$T(m) \ge m$ とする。

**証明 (概略)** 定理 *12* により並列 *saddleback search* に帰着する。定理 *3* より、各部分列に対する $h$ や $p$ の計算を $\mathrm{O}(1)$ 時間で実行できる。そのため、行・列に対する二分探索は $\mathrm{O}(\log \mathrm{N})$ 時間で処理できる。

## 5 Longer-better でない、または segment-closed でない問題の取り扱い

前節では最適部分列問題が saddleback search に帰着できることを述べた。最も重要な要請は、目的関数が longer-better であること、および制約条件が segment-closed であることである。しかし、既存研究で議論されている問題の多くはいずれかの性質を満たさない。以下に例を 2 つ述べる。

**例 14** 長さ制約付き最大重み和問題 *[12, 14, 15]* は、特定の長さ以下の部分列の中で重み和が最大となるものを求める問題である。この問題は、*segment-closed* ではあるが *longer-better* ではない。目的関数 *sum* は、例えば $sum([3]) = 3 \not\le sum([3,-2]) = 1$ である。

**例 15** *Leftmost at most rightmost* 問題 *[20, 21]* は、左端の要素が右端の要素以下である部分列の中で最長のものを求める問題である。この問題は、*longer-better* であるが *segment-closed* ではない。$[-1,2,1]$ は条件を満たすが、この部分列である $[2,1]$ は条件を満たさない。

この問題に対し、前原 [22] は、windowing 技法に基づき、最適解があり得る領域を詳細に解析しながら部分列を延長・短縮する方法を述べ、それが最適解を効率よく発見するための十分条件を示した。しかし、前原の議論は以下の欠点があった。

- 彼の議論は windowing に基づいており、分割統

治法を用いる場合にはそのまま適用できない。

- 彼は重み和問題の場合と segment-closed でない場合は考えたが、一般に longer-better でない場合は議論していない。

本節では、より一般的でより使いやすい条件を与えることを目指す。

### 5.1 Segment-closed でない場合

まず、制約条件 $p$ は segment-closed ではないが、目的関数 $h$ は longer-better である最適部分列問題 $\mathcal{OSP}(x, h, p)$ について考える。

実は、3 節で述べた分割統治 saddleback search アルゴリズムは、より弱い条件下でも利用することができる。このアルゴリズムの本質は、二分探索によって、その右下、およびその左上の矩形領域には最適解が存在しないことが保証できることである。このことから、以下の条件が満たされるなら、分割統治アルゴリズムは利用できる。

> 添え字 $i$ および $j$ に対し、$p(A[i'][j'])$ が成り立つ $i' \leq i$、$j' \leq j$ が存在するか否かを判定するオラクルが存在する。

このとき、「$p$ を満たし $h$ が最大となる要素」ではなく「$p(A[i'][j'])$ が成り立つ $i' \leq i$、$j' \leq j$ が存在する中で $h$ が最大となる要素」を発見する。このとき、この要素の右下の矩形領域は制約条件を満たす解はない。さらに、この要素の真下または真右には制約条件を満たす要素 (仮に $e$ とする) があり、それゆえ左上領域の要素は $e$ より目的関数値が小さいため考える必要が無い。以上のことから、分割統治アルゴリズムの正しさが保証される。これを最適部分列問題に翻訳すれば、リスト $x$ 中の $y$ ($x = l +\!\!+ y +\!\!+ r$) に対し、

$$\exists l' \in \mathit{inits}(l), r' \in \mathit{tails}(r).\ p(l' +\!\!+ y +\!\!+ r')$$

を効率よく判定できれば良い、となる。これは一般には難しいため、効率的な評価のための十分条件が求められる。

既存で議論されてきた多くの問題において、制約条件 $p$ は $p(y) = \phi(y) \leq C$ の形で表現される。このとき、上記条件は、$y$ を含む部分列に対する $\phi$ の値の最小値が $C$ 以上であるか否かの確認となる。$\phi$ が単調であるとき $\phi(l') \leq \phi(l'')$ かつ $\phi(r') \leq \phi(r'')$ ならば $\phi(l' +\!\!+ y +\!\!+ r') \leq \phi(l'' +\!\!+ y +\!\!+ r'')$ が満たされるため、この判定には $l$ の末尾部分列、$r$ の先頭部分列の中で $\phi$ 値が最小となるものを求めれば良い。定理 3 および定理 6 より、$\phi$ がリスト準同型で実装されている場合、これは $\mathrm{O}(\mathrm{N}/\mathrm{P} + \log \mathrm{P})$ 時間の前処理を行えば、それぞれ $\mathrm{O}(1)$ 時間で得ることができる。

**例 16** *Leftmost at most rightmost* 問題を考える。制約 $p$ は、列の左端の要素を返す関数 *left* と右端の要素を返す関数 *right* を用いて、$p(y) = \mathit{left}(y) - \mathit{right}(y) \leq 0$ と記述できる。*left* と *right* はリスト準同型で定義でき、かつ単調である。しかも補題 5 が適用できる。なぜなら、$y \overset{L}{\sim} y'$ なら $\mathit{right}(y) = \mathit{right}(y')$ であり、$y \overset{R}{\sim} y'$ なら $\mathit{left}(y) = \mathit{left}(y')$ であるためである。よって、$\mathit{left}(y) - \mathit{right}(y)$ は単調であり、提案手法が適用できる。このとき、オラクルは「右方にある最大の値が左方にある最小の値より大きい」ことを確認する。

### 5.2 Longer-better でない場合

次に、目的関数が longer-better でない最適部分列問題 $\mathcal{OSP}(x, h, p)$ について考える。ただし $p$ は segment-closed であるとする。このとき、以下の戦略に基づいて最適解を発見する。

1. $h$ が longer-better である場合のアルゴリズムを用いて、各要素に対し、その要素から始まる最適な部分列を発見する。
2. その最適な部分列の先頭部分列で $h$ の値が最大となるものを発見する。

戦略の要旨は $p$ を満たす部分列の発見と $h$ の値の最大化を別々に扱うことである。まず、第 1 ステップでは、どの長さの部分列まで $p$ を満たすのかを調べる。次に、第 2 ステップでは、それより短い列の中で $h$ の値を最大化する。

第 1 ステップは今まで述べたとおり、効率よく計算できる。しかも、前小節の議論と同様にして、$h$ が単調性を満たしリスト準同型で実装されている場合、定理 3 と定理 6 より、これは $\mathrm{O}(\mathrm{N}/\mathrm{P} + \log \mathrm{P})$ 時間の前処理を行えば、各部分列に対する先頭部分列を $\mathrm{O}(1)$ のならし時間で得ることができる。

**例 17** 長さ制約付き最大和部分列問題を考える。目的関数 *sum* は単調であるため、上記アルゴリズムを

利用できる。この場合、長さが長さ制約 $k$ に等しい各部分リストに対して、最大和となる部分リストを求めることになる。

### 5.3 Longer-better でも segment-closed でもない場合

Longer-better でも segment-closed でもない問題を上述の手法で扱うのは困難である。例えば、前小節では、条件を満たす最長の部分列に対し、目的関数値が最大となる部分列を求めたが、条件が segment-closed でない場合、一般にはこうやって求めた部分列が制約条件を満たすとは限らない。この点についてのさらなる探求は今後の課題である。

## 6 まとめと今後の課題

本稿では、最適部分列問題に対する効率的な並列アルゴリズムについて議論した。このアルゴリズムは Bird の分割統治 saddleback search を基礎としている。さらに、単調性に基づき、直裁には saddleback search に帰着できない問題の取り扱いについても議論をした。このアルゴリズムは、最長部分列問題、最大和部分列問題に限らず、広い範囲の部分列問題に対して適用できる。

4 節で述べた saddleback search への帰着は基本的に前原 [22] の結果である。さらに、5 節で議論した十分条件は、細部において多少の違いはあるものの、前原のものに非常に近いものとなっている。そのため、本稿の結果は前原の議論が並列アルゴリズムの導出へと拡張できることを示したと言える。このことは、この議論が windowing 技法に限るものではなく、より広く最適部分問題の構造を捉えている可能性があることを示唆している。

本稿のアルゴリズムを与える上では、範囲問い合わせの効率的な手法が肝となっている。範囲問い合わせの逐次アルゴリズムについては非常に多くの研究があるが、並列アルゴリズムについての研究は少ない。より広い範囲の最適部分列問題に対する解法を与える上では、範囲問い合わせの並列アルゴリズムを発展させることが不可欠だと言える。

また、本稿では最適部分列問題を考えたが、この問題は入力が多次元配列であれば最適部分超立方体問題へと自然に拡張できる。最も単純な問題である最大和部分超立方体問題については、Wen [19] が効率の良い並列アルゴリズムを与えている。本稿の議論と Wen の手法を組み合わせることで、より高次元の問題に対して並列アルゴリズムを与えることができるかどうかは、今後の課題である。

**謝辞** 本研究は日本学術振興会学術研究助成基金助成金 (若手研究 (B) 15K15965) の助成を受けている.

## 参 考 文 献

[1] Agarwal, P. K. and Erickson, J.: Geometric Range Searching and Its Relatives, *Advances in Discrete and Computational Geometry: Proceedings of the 1996 AMS-IMS-SIAM Joint Summer Research Conference, Discrete and Computational Geometry–Ten Years Later, July 14-18, 1996, Mount Holyoke College*, Chazelle, B., Goodman, J. E., and Pollack, R.(eds.), Contemporary Mathematics, Vol. 223, American Mathematical Society, 1999, pp. 1–56.

[2] Bentley, J. L.: Algorithm Design Techniques, *Commun. ACM*, Vol. 27, No. 9(1984), pp. 865–871.

[3] Bentley, J. L.: Perspective on Performance, *Commun. ACM*, Vol. 27, No. 11(1984), pp. 1087–1092.

[4] Bird, R. S.: An Introduction to the Theory of Lists, *Logic of Programming and Calculi of Discrete Design*, Broy, M.(ed.), NATO ASI Series F, Vol. 36, Springer-Verlag, New York, NY, USA, 1987, pp. 3–42. *Technical Monograph PRG*-56.

[5] Bird, R. S.: Maximum marking problems, *Journal of Functional Programming*, Vol. 11, No. 4(2001), pp. 411–424.

[6] Bird, R. S.: Improving Saddleback Search: A Lesson in Algorithm Design, *Mathematics of Program Construction, 8th International Conference, MPC 2006, Kuressaare, Estonia, July 3-5, 2006, Proceedings*, Uustalu, T.(ed.), Lecture Notes in Computer Science, Vol. 4014, Springer, 2006, pp. 82–89.

[7] Emoto, K., Fischer, S., and Hu, Z.: Filter-embedding semiring fusion for programming with MapReduce, *Formal Asp. Comput.*, Vol. 24, No. 4-6(2012), pp. 623–645.

[8] Emoto, K., Hu, Z., Kakehi, K., Matsuzaki, K., and Takeichi, M.: Generators-of-Generators Library with Optimization Capabilities in Fortress, *Euro-Par 2010 - Parallel Processing, 16th International Euro-Par Conference, Ischia, Italy, August 31 - September 3, 2010, Proceedings, Part II*, D'Ambra, P., Guarracino, M. R., and Talia, D.(eds.), Lecture Notes in Computer Science, Vol. 6272, Springer,

2010, pp. 26–37.

[ 9 ] Fukuda, T., Morimoto, Y., Morishita, S., and Tokuyama, T.: Mining Optimized Association Rules for Numeric Attributes, *J. Comput. Syst. Sci.*, Vol. 58, No. 1(1999), pp. 1–12.

[10] Fukuda, T., Morimoto, Y., Morishita, S., and Tokuyama, T.: Data Mining with optimized two-dimensional association rules, *ACM Trans. Database Syst.*, Vol. 26, No. 2(2001), pp. 179–213.

[11] Huang, X.: An algorithm for identifying regions of a DNA sequence that satisfy a content requirement, *Comput. Appl. Biosci.*, Vol. 10, No. 3(1994), pp. 219–225.

[12] Lin, Y.-L., Jiang, T., and Chao, K.-M.: Efficient algorithms for locating the length-constrained heaviest segments with applications to biomolecular sequence analysis, *J. Comput. Syst. Sci.*, Vol. 65, No. 3(2002), pp. 570–586.

[13] Matsuzaki, K., Hu, Z., and Takeichi, M.: Derivation of Parallel Programs for Maximum Marking Problems on Lists, *IPSJ Transaction of Programming*, Vol. 49(2008), pp. 16–27. In Japanese.

[14] Morihata, A.: Calculational Developments of New Parallel Algorithms for Size-Constrained Maximum-Sum Segment Problems, *Functional and Logic Programming - 11th International Symposium, FLOPS 2012, Kobe, Japan, May 23-25, 2012. Proceedings*, Schrijvers, T. and Thiemann, P.(eds.), Lecture Notes in Computer Science, Vol. 7294, Springer, 2012, pp. 213–227.

[15] Mu, S.-C.: Maximum segment sum is back: deriving algorithms for two segment problems with bounded lengths, *Proceedings of the 2008 ACM SIGPLAN Symposium on Partial Evaluation and Semantics-based Program Manipulation, PEPM 2008, San Francisco, California, USA, January 7-8, 2008*, Glück, R. and de Moor, O.(eds.), ACM, 2008, pp. 31–39.

[16] Sasano, I., Hu, Z., Takeichi, M., and Ogawa, M.: Make it practical: a generic linear-time algorithm for solving maximum-weightsum problems, *Proceedings of the 5th ACM SIGPLAN International Conference on Functional Programming, ICFP'00*, ACM, 2000, pp. 137–149.

[17] Skala, M.: Array Range Queries, *Space-Efficient Data Structures, Streams, and Algorithms - Papers in Honor of J. Ian Munro on the Occasion of His 66th Birthday*, Brodnik, A., López-Ortiz, A., Raman, V., and Viola, A.(eds.), Lecture Notes in Computer Science, Vol. 8066, Springer, 2013, pp. 333–350.

[18] Smith, D. R.: Applications of a Strategy for Designing Divide-and-Conquer Algorithms, *Sci. Comput. Program.*, Vol. 8, No. 3(1987), pp. 213–229.

[19] Wen, Z.: Fast Parallel Algorithms for the Maximum Sum Problem, *Parallel Computing*, Vol. 21, No. 3(1995), pp. 461–466.

[20] Zantema, H.: Longest Segment Problems, *Science of Computer Programming*, Vol. 18, No. 1(1992), pp. 39–66.

[21] Zhao, H., Hu, Z., and Takeichi, M.: A Compositional Framework for Mining Longest Ranges, *Discovery Science, 5th International Conference, DS 2002, Lübeck, Germany, November 24-26, 2002, Proceedings*, Lange, S., Satoh, K., and Smith, C. H.(eds.), Lecture Notes in Computer Science, Vol. 2534, Springer, 2002, pp. 406–413.

[22] 前原貴憲: プログラム運算による Windowing 技法の定式化とその応用, 卒業論文. 東京大学工学部. 2007.