FY2022/2022年度

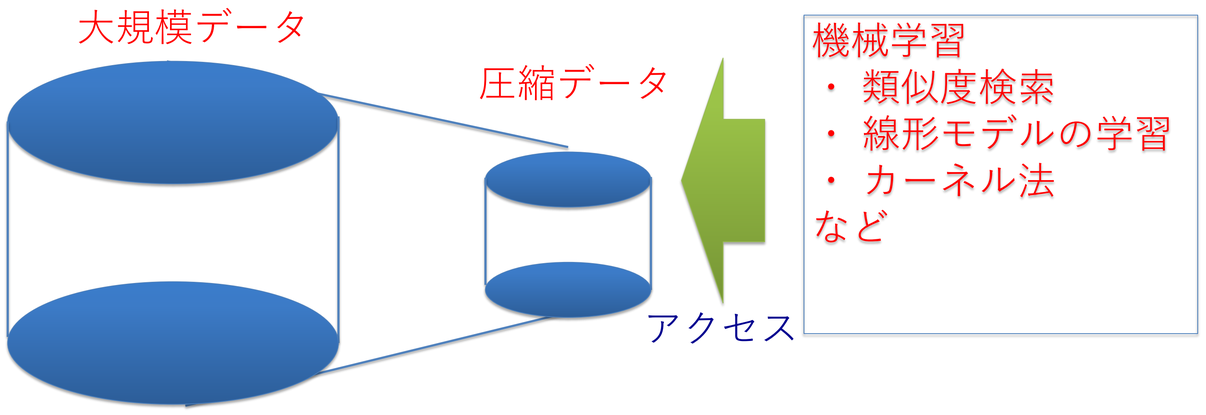
Succinct Information Processing Unit

Yasuo Tabei
圧縮情報処理ユニット　田部井 靖生

RIKEN Center for Advanced Intelligence Project

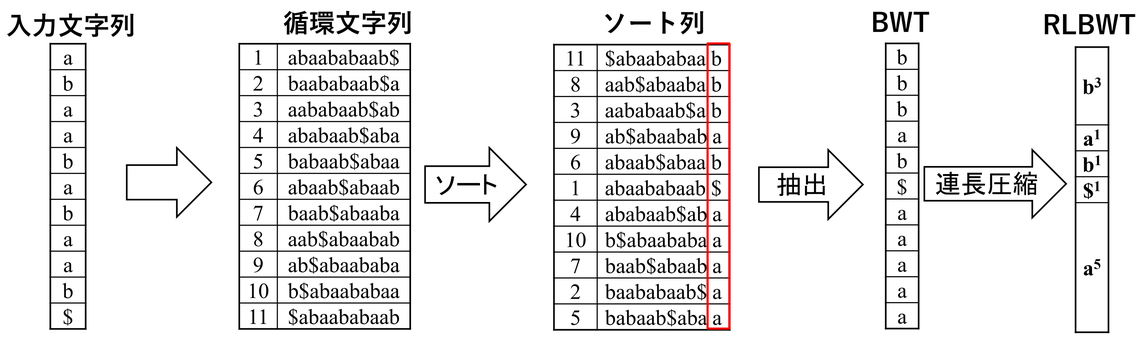
大規模機械学習に向けたデータ圧縮技術の開発

圧縮情報処理ユニットでは、ビッグデータを効率的に処理するためのデータ圧縮技術の研究を行っています。近年のセンサーを含む計測技術の発展により、様々な産業領域や研究分野において処理すべきデータ量は増大していて, ビッグデータを効率的に処理する技術の開発が必要とされている. 簡潔データ構造とは, データを圧縮した状態でデータに対する様々な操作をサポートするデータ圧縮技術であり, ビッグデータを効率的に処理する技術の一つとして期待されています. 本ユニットの研究目標は, 簡潔データ構造の基礎研究とデータマイニングや機械学習アルゴリズムを簡潔データ構造を応用することで効率化することである. ユニット発足以来、主な研究成果として、(1)大規模ゲノムデータを処理するための簡潔データ構造の一つである連長圧縮BWT(RLBWT)、(2)文字列検索のための簡潔データ構造、(3)データ圧縮に基づく文字列分類手法、(4)類似度検索と移動軌跡データやケモインフォマティクスへの応用を行ってきました.



Run-Length Burrows-Wheeler Transform (RLBWT)

- 文字列をBWT変換した後に連長圧縮して得られる文字列の圧縮表現
  - BWT変換：入力文字列の各循環文字列を辞書式順序でソートした後に各循環文字列の最後の文字からなる文字列を返す



・同じ部分文字列を多く含む文字列(反復文字列)に対して有効な圧縮法
例：ゲノム
・ヒトゲノムの個人間での違いは約0.01%
• 1,000人分の19番染色体ゲノム60GBが250MBに圧縮可能 (約1/240)

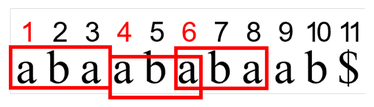
研究成果のまとめ

タスク	説明	時間	メモリ(ビット)
圧縮 [ICALP'22]	入力文字列からRLBWTを構築	$O(n)$	$O(r \log n)$
検索 [ICALP'21]	クエリーのすべての出現を検索	$O(m + occ)$	$O(r \log n)$
部分文字列復元 [ICALP'21]	指定した位置の部分文字列を復元	$O(m)$	$O(r \log n)$
部分文字列列挙 [CPM'21]	すべての部分文字列を列挙	$O(n \log\log n)$	$O(r \log n)$

n: 文字列長　m: クエリー長　occ: クエリーの出現回数　r: RLBWTのサイズ

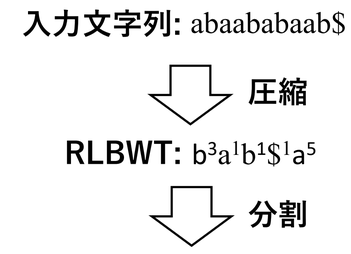
RLBWT上の最適計算量での文字列検索 [ICALP'21]

【動機】
- RLBWT上の最適計算量で文字列検索可能なデータ構造
- RLBWT上の文字列検索を最適計算量で行うアルゴリズムはこれまで提案されていない [JCB10, ACM20]



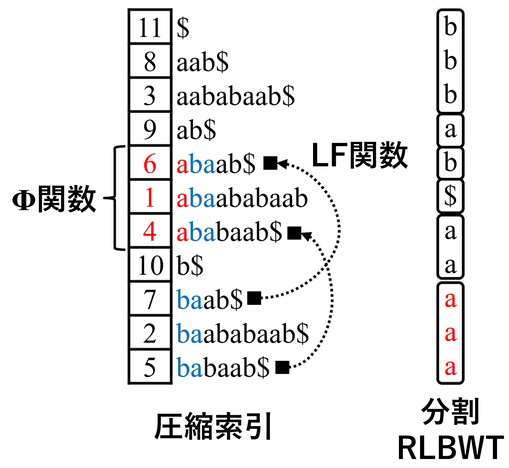
パターン検索
(abaの出現位置を報告)

【提案手法：OptBWTR】
- LF関数とΦ関数を使ってパターン検索
  - LF: パターンの出現範囲の計算に使用
  - Φ: パターンの出現位置の計算に使用
- RLBWTを適切に分割してから省メモリな索引を作ることで２つの関数を定数時間で計算できることを証明



【結果】
- 索引のメモリ：$O(r \log n)$ ビット
- **既存の圧縮索引と同じサイズを維持**
- パターンの検索時間：$O(m + occ)$ 時間
- **最適時間**（これ以上の高速化は不可能）

n: 入力文字列長、r: 入力文字列の圧縮サイズ
m:パターンの文字列長、occ: パターンの出現数



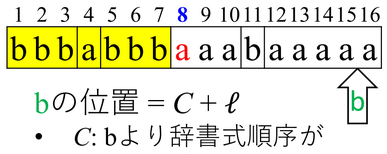
【応用】
RLBWTで圧縮された文字列に対してさまざまなクエリを最適時間で計算可能
- 入力文字列の復元
- 入力文字列の部分的な復元
- 複数文字列上の接頭辞パターン検索など

最適計算量でのRLBWT構築アルゴリズム [ICALP'22]

【背景】
- 最初のRLBWTの最適計算量構築アルゴリズム
- 既存手法[JDA18, ARGO'18]
  - １文字ずつ文字を挿入して $O(n \log r)$ 時間でRLBWTを構築
  - 挿入位置の計算に必要なB木がボトルネック（アクセスに $O(\log r)$時間）
    [n: 入力文字列長、r: RLBWTのサイズ]

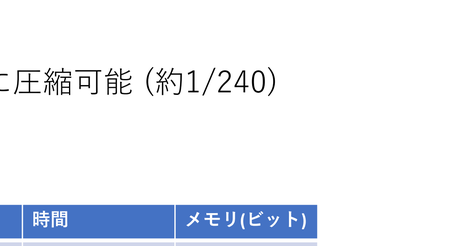
入力文字列: bababababaabbaaba

RLBWT: $b^3a^1b^3a^3b^1a^4b^1a^1$

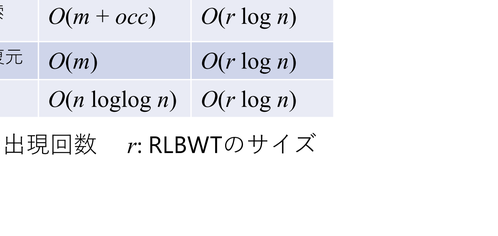


bの位置 $= C + \ell$
- C: bより辞書式順序が小さい文字の数
- ℓ: 黄色区間のbの数

【提案手法】
- RLBWTに文字を挿入する位置を $O(1)$時間で計算できるグラフ構造を開発
- RLBWT構築時間：$O(n + r \log r) = O(n)$（**最適**）
- メモリ：$O(r \log n)$ ビット（**最適**）
  - $\boldsymbol{r \ll n}$（反復の多い文字列の場合）



提案グラフ構造



【実験データセット】
- Wikipediaの編集履歴、1000人分のゲノムなど

【比較手法】
- Faster-PP[JDA18]、Big-BWT[AMB19]

データ	r [10^3]	本手法		Faster-PP		Big-BWT	
		Time	Mem.	Time	Mem.	Time	Mem.
boost (1GB)	65	152 sec	2.85 MB	769	**1.00**	**55.2**	73.0
enwiki (38GB)	70,190	24,550	3,042	55,367	**1,149**	**2,375**	7,022
chr19 (59GB)	45,143	39,138	**1,931**	n/a	n/a	**3,535**	4,911

【結果】
- 実験を用いて大規模文字列に適用可能であることを実証
- 実行時間：最大で**1/5以下**に削減（省メモリな手法と比較）
- メモリ：最大で**1/25以下**に削減（高速な手法と比較）

RLBWT上の全部分文字列列挙 [CPM'21]

【動機】
- 全部分文字列列挙は文字列の特徴量抽出やゲノム配列のモチーフ抽出などに応用可能な重要なタスク
- 全部分文字列は接尾辞木のすべてのノードをトラバースすることで列挙できる
- $O(n)$時間と$O(n \log \sigma)$ビットメモリ [n: 入力文字列長、σ: 文字種類数]

【提案手法：R-enum】
- 入力文字列のRLBWT表現上で接尾辞木のトラバースを模倣
- $O(n \lg\lg(n))$時間と $O(r \log n)$ビットメモリ ($<< O(n \log \sigma)$)
  r: 文字列をRLBWTに圧縮したときのサイズ
  類似部分文字列が頻出すると $\boldsymbol{r \ll n}$

接尾辞木
(banana$の全部分文字列を格納)

【実験１】
データセット：Pizza & Chiliコーパス（類似部分文字列を多く含む）
比較手法：SPIRE'15（$O(n \log \sigma)$ビットメモリ）

データ	r [10^3]	本手法		SPIRE'15	
		Time [sec]	Mem. [MB]	Time	Mem.
einstein.en.txt (467MB)	290	856	**4**	**387**	488
kernel (257MB)	2,791	559	**21**	**178**	292

【実験２】
100人分のゲノムの全部分文字列列挙

データ	r [10^3]	Time [hours]	Memory [MB]
100genome (307GB)	36,274,924	**25**	**319,949**

【結果】
- 既存手法と比べてメモリを最大で**1/100以下**に削減
- 100人分のゲノム(約300GB)の全部分文字列を**約１日**で列挙可能