FY2024/2024年度

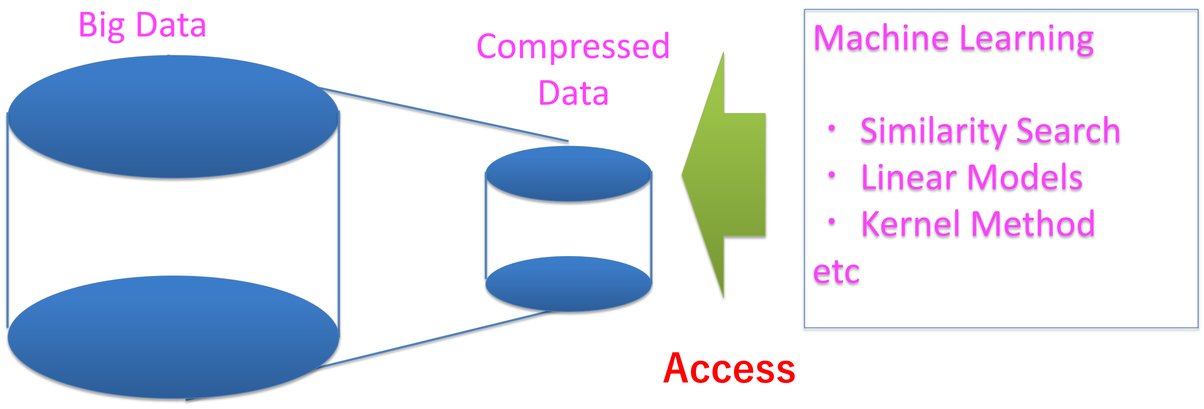
Succinct Information Processing Team

Yasuo Tabei
圧縮情報処理チーム　田部井 靖生

RIKEN Center for Advanced Intelligence Project

大量データを効率的に処理するためのデータ圧縮技術の開発

- コア技術：圧縮データ構造
  - データを圧縮した状態で、高速な操作を可能にするデータ構造
- 研究テーマ
  1. ゲノムなどの高頻度リピート文字列に対する圧縮データ構造
     - データ構造：文法圧縮、RLBWT
       成果：ICALP'22、ICALP'21、CPM'21、DCC'19
  2. ベクトル、グラフ、移動軌跡データの類似度検索
     - データ構造：簡潔トライ、Wavelet木
       成果：ICDM'20, SIGSPATIAL'19, IEEBigData'19
  3. 大規模データ上での機械学習
     - データ構造: 文法圧縮、簡潔木
       成果：ICDM'19, APBC'19



圧縮データ構造の理論と応用を展開

動的更新を実現する圧縮データ構造: データの更新時に再構築を必要とせず、データの追加や削除に対応するデータ構造

- 背景：IoTとエッジコンピューティングの台等により、リアルタイム処理の重要性が増加
- 課題：動的に更新可能な圧縮データ構造に関する研究は、理論・実用両面でまだ十分ではない
- 困難性：圧縮データ構造の更新は複雑な操作のため、領域を小さく保ちながら高速な更新と操作を両立することが困難

- 期待される成果：
  1. エッジコンピューティングアプリーケーションの強化
  2. 大量データ処理システムのパフォーマンス向上とストレージコストの削減

最適メモリ量で接尾辞配列クエリをサポートする動的圧縮データ構造

【接尾辞配列（SA）クエリ】
- 辞書式順序で整列された文字列の接尾辞の中から指定したi番目の接尾辞を返す
- 文字列処理において多くの応用を持つ[Gusfield'97]
  - 特徴的な部分文字列の列挙
  - 複数の文字列間の最長共通部分列の計算など
- SAクエリをサポートする圧縮データ構造や動的データ構造の開発が活発
  [FOCS'23, ACM'20, STOC'22など]

【問題点】
最適メモリ量でSAクエリをサポートする既存の圧縮データ構造は更新操作をサポートしない

【アイディア】
- Interval Attractor(IA)と呼ばれるSAクエリの処理に役立つ文字列上の区間の組を考案
  - IAは入力文字列を導出する文法木のノードに関連付けられた部分文字列の集合を表す
  - 条件を満たすIAを探すことでSAクエリを処理可能
- IAの集合を重み付きの点集合に写像
  - IAを探す代わりに点の集計でSAクエリを処理
  - **重み付き点集合は最適メモリ量で保存可能**
  - 文字列を編集したときに変化する点の個数は$O(\mathrm{polylog}\, n)$で抑えられることを証明

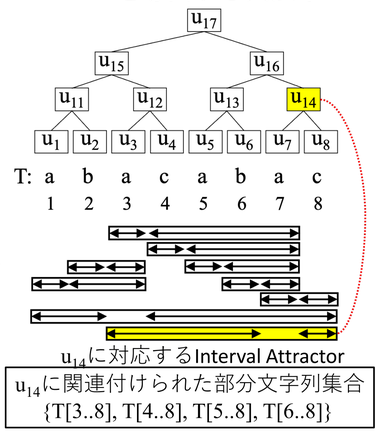
【結果】
- IAを利用してSAクエリをサポートする最適メモリ量の動的圧縮データ構造を開発
  - 使用メモリ（期待値）：$O(\delta \log (n/\delta) \log n)$ビット（最適メモリ量）
  - クエリ時間: $O(\log^7 n)$
  - 更新時間: $O(\log^8 n)$

T: abacabab

接尾辞配列

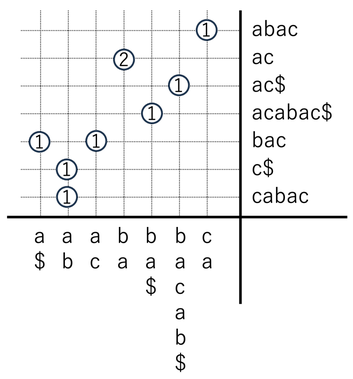
i	SA	Suffix
1	5	abac
2	1	abacabac
3	7	ac
4	3	acabac
5	6	bac
6	2	bacabac
7	8	c
8	4	cabac

Tを導出する文法木



u_{14}に対応するInterval Attractor

u_{14}に関連付けられた部分文字列集合
{T[3..8], T[4..8], T[5..8], T[6..8]}



Interval Attractorの集合から写像された重み付き点集合

n: 文字列長　δ:部分文字列複雑性(文字列の圧縮指標)
文字列の圧縮表現のビット長は最悪時$\Theta(\delta \log (n/\delta) \log n)$ [LATIN'20]

Locateクエリをサポートする動的圧縮データ構造

【Locateクエリ】
- 文字列中の入力パターンの出現をすべて出力
- **RLBWT**と呼ばれる圧縮表現を利用してクエリをサポートする圧縮データ構造の開発が盛ん（**r-index**[ACM'20], OPTBWTR[ICALP'21]など）
- RLBWT: BWT変換された文字列の連長圧縮
  - BWT：入力文字列の循環シフトを辞書順ソートし末尾を結合した文字列に変換
  - ゲノム集合などのリピートの多い文字列に有効
    （例）千人分のヒトゲノム（**60GB → 250MB**）

【問題点】RLBWTに基づく既存の圧縮データ構造は更新操作をサポートしない

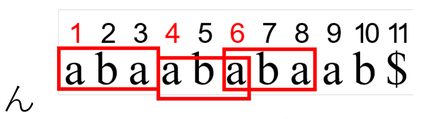
【提案手法】
- RLBWTと連の端にある循環シフトの開始位置を格納した動的圧縮データ構造を開発
  - r-indexと同じアルゴリズムでLocateクエリをサポート可能
  - RLBWTを更新しながら格納した情報を更新できることを示した

【結果】
- 使用メモリ: $O(r \log n)$ビット
  （RLBWTの圧縮サイズに線型比例）
- クエリ時間: $O((m+occ) \log r)$
- 更新時間: $O(L_{avg} \log r)$ ($L_{avg} << n$ならば高速)

【実験】
- 約60GBのゲノム集合を用いてr-indexと比較
  - 更新時間: r-indexの**約1/10000**（再構築と比べて）
  - クエリ時間: r-indexの**約35倍**
  - 使用メモリ: r-indexの**約4倍**
- ゲノム集合ではL_{avg}が非常に小さくなることを確認

手法	更新時間 [sec]	クエリ時間 [μs/occ]	メモリ [GB]
提案手法	5	4.25	2.00
r-index	50,981	0.12	0.56

n: 文字列長　m: パターン長
occ:パターンの出現回数　r: RLBWTの連の個数
L_{avg}:文字列のLCP配列に含まれている値の平均



Locateクエリ
（abaの出現位置を出力）