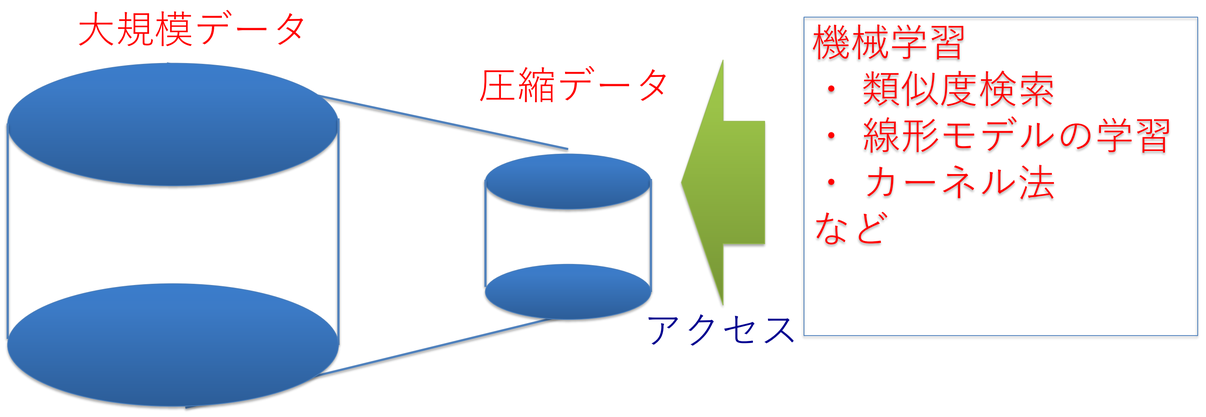
Succinct Information Processing Unit

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

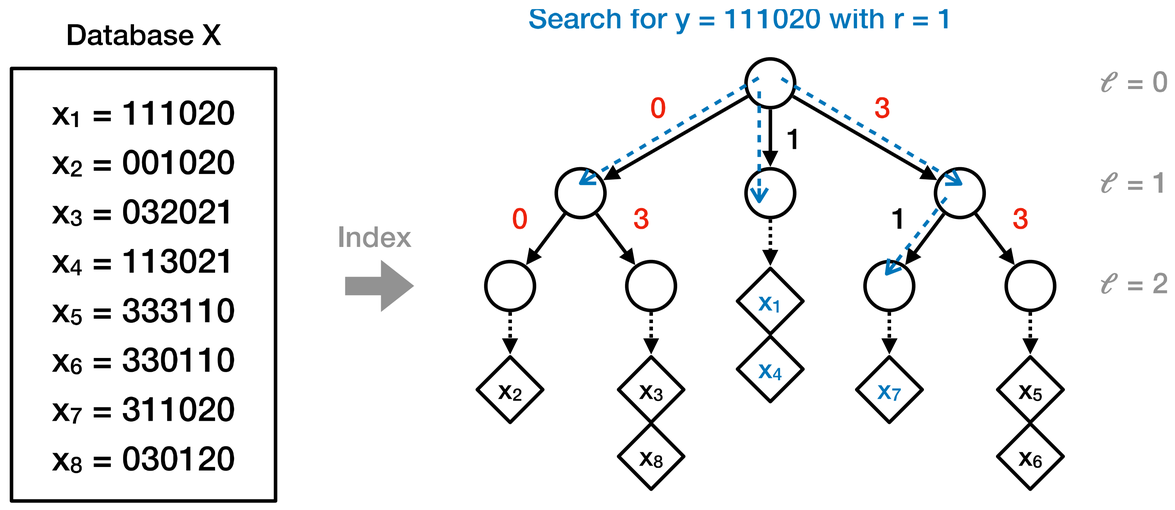
圧縮情報処理ユニットでは、機械学習モデルを大規模データ上で学習させるためのデータ圧縮技術の研究を行っています。近年のセンサーを含む計測技術の発展により、様々な産業領域や研究分野において処理すべきデータ量は増大していて, ビッグデータを効率的に処理する技術の開発が必要とされている. 簡潔データ構造とは, データを圧縮した状態でデータに対する様々な操作をサポートするデータ圧縮技術であり, ビッグデータを効率的に処理する技術の一つとして期待されている. 本ユニットの研究目標は, この簡潔データ構造の基礎研究とデータマイニングや機械学習アルゴリズムを簡潔データ構造に基づき開発することである. ユニット発足以来、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た.



類似検索のための動的更新可能なデータ構造 [KAIS'21]

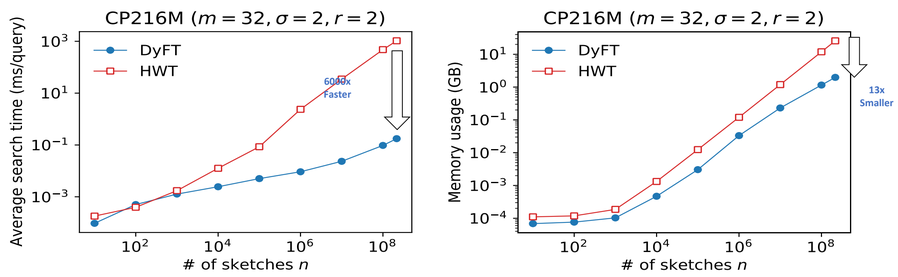
【動機】
- ハミング空間での離散系列（スケッチ）に対する類似検索はデータマイニングや情報検索における重要な問題のひとつ
- これまで多くの類似度検索のためのデータ構造が提案されているが、スケッチの追加・削除に対して新たにデータ構造を作り直す必要がある

【提案手法：DyFT】
- Trieと呼ばれる木構造をコストモデルに従って構築することで、バイナリスケッチと整数スケッチの両方で高速に検索可能
- スケッチの追加・削除に対して、Trieを新たに作り直す必要なく更新可能

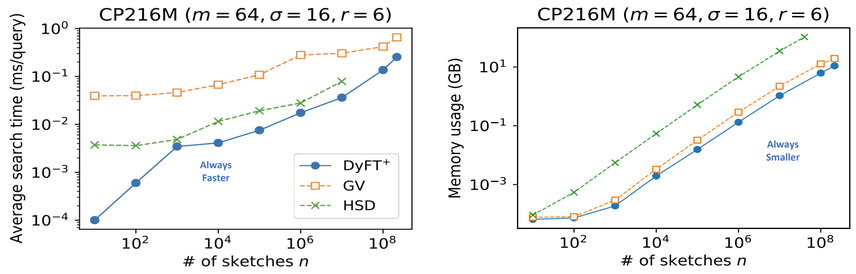


【実験データ】
- 2億の化合物データベクトル
- Liらのb-bit MinHashによりスケッチに変換 [WWW10]

【比較手法】
HWT [IEEE-TPAMI19] , GV [SIGIR16], HSD [SSDBM13]



バイナリスケッチに対する実験結果



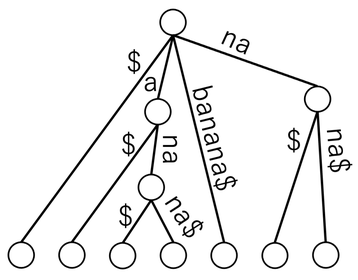
整数スケッチに対する実験結果

全部分文字列の省メモリー列挙 [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問題のひとつ
- 接尾辞木などの全部分文字列列挙のためのデータ構造が提案されているが使用メモリー量が$\Omega(n \log \sigma)$と大きい [n: 入力文字列長、σ: 文字種類数]

【提案手法：R-enum】
- 入力文字列を圧縮したまま処理して接尾辞木を模倣
- 接尾辞木を少しずつ模倣することで使用メモリー量を$O(r \log n)$に削減

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 $r \ll n$



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：BC [SPIRE15]（$O(n \log \sigma)$の使用メモリー）

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

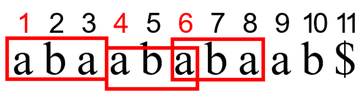
【実験２】
100人分のゲノムから全部分文字列列挙

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

【結果】
- 既存手法と比べて使用メモリを最大で**1/100以下**に削減
- 約300GBの実験データを**約１日**で処理できることを確認

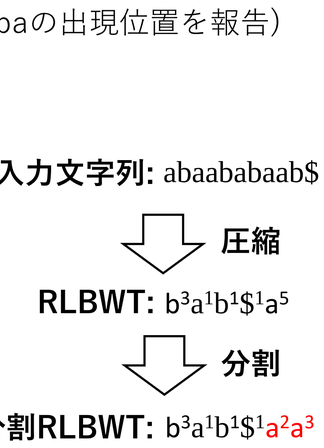
高速なパターン検索が可能な圧縮索引 [ICALP'21]

【動機】
- パターン検索は文字列処理の基本操作のひとつ
  大規模文字列処理のために
  高速検索＆省メモリーな索引の開発が重要
- RLBWTと呼ばれる文字列の圧縮を利用した省メモリーな索引が提案されているが検索が低速 [JCB10, ACM20]



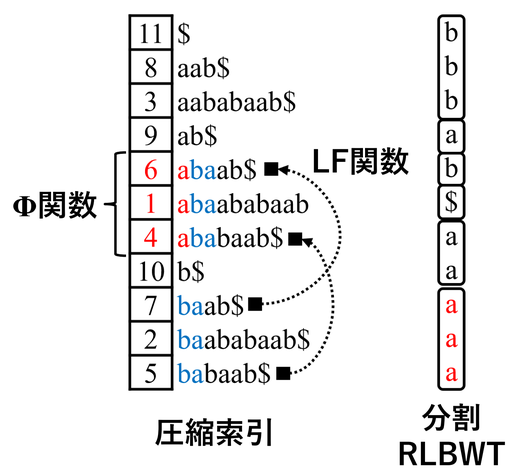
パターン検索
(abaの出現位置を報告)

【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ー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】
- 索引の使用メモリー量：$O(r \log n)$
- **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