Bi-directional text indexes based on the run-length compression of the Burrows-Wheeler transform

(BWT の連長圧縮に基づく双方向文字列索引)

数理情報学専攻 48216203 荒川 侑馬
指導教員 定兼 邦彦 教授

1 はじめに

バイオインフォマティクスやソフトウェアリポジトリをはじめとする領域で，反復の多い大規模な文字列データに圧縮を加えつつ，検索操作が効率的に行えるような圧縮文字列索引の重要性が高まっている．その中でも r-index [3] は，Burrows-Wheeler transform(BWT) [2] に対する連長圧縮を利用して効率的なパターンの検索を可能にしたが，機能は限定的である．本研究では，r-index を拡張して，より複雑なクエリの実行を可能にした bi-directional r-index (br-index) を提案する．これは，パターンの検索に際して双方向の拡張及び縮小を可能にし，かつ任意の時点で現在のパターンの出現位置列挙をサポートすることで実現される．

2 既存研究

文字列 T に対し，接尾辞配列 [4] は $SA[i] =$ (辞書順 i 番目の接尾辞の開始位置) と定義される．また，Burrows-Wheeler transform (BWT) [2] は

$$L[i] = \begin{cases} T[SA[i]-1] & (SA[i] \neq 1) \\ \$ & (SA[i] = 1) \end{cases}$$

で定義される．BWT の同じ文字で構成された極大区間 (run) の総数 r を考えると，これは反復の多い文字列において小さくなる．したがって，BWT の連長圧縮は圧縮文字列索引を実現するうえで有効である．r-index は連長圧縮した BWT と run の境界における SA の値を備えたサイズ $O(r)$ 語の文字列索引であり，任意のパターン $P[1,m]$ に対する *locate* クエリ，すなわち T における P の出現位置列挙を $O(m \log\log_w(\sigma + n/r) + occ \cdot \log\log_w(n/r))$ 時間で実行可能である．ここで occ は P の出現数である． r-index における *locate* は後方探索アルゴリズムに基づいて実現される．検索中のパターンに対応する値として SA 上の区間 $[s,e]$ と $SA[e]$ の値 j を保持する．$P[i+1,m]$ が検索されている状態で，連長圧縮された BWT と run の境界に対応する SA の値を用いてこれらを更新し，$P[i,m]$ に対応する値を得る．これを繰り返してパターンの全体 $P[1,m]$ に対応する値を得た後，関数 ϕ を適用して $j = SA[e]$ の値から順次 $SA[e], SA[e-1], \ldots, SA[s]$ の値を列挙することで *locate* を解く．これらの操作に必要なデータ構造を全て $O(r)$ 語で実現できる．

3 成果：br-index

本研究で提案する br-index は，r-index における後方探索の概念を拡張し，6 つの値 $[s,e], [s_{\mathrm{R}}, e_{\mathrm{R}}], j, d$ を探索中に保持する．これらの値を，T に対する r-index，反転文字列 T^{R} に対する r-index に加えて，漸近的に同じサイズである追加のコンポーネントを用いて更新する．現在検索されているパターンを P としたとき，$[s_{\mathrm{R}}, e_{\mathrm{R}}]$ は T^{R} に対する接尾辞配列 SA^{R} 上で P^{R} に対応する区間，j は T におけるある P 内部の位置，d は j と P の開始位置の間の距離である．これらの値とコンポーネントの対称性から，パターンを左だけでなく右方向にも対称な操作で拡張することができる．加えて，拡張の逆操作である縮小アルゴリズムもサポートする．こうして更新した値をもとに，探索の任意の時点で現在のパターンの出現位置列挙が可能で，T^{R} における BWT の run 数を r_{R} とすると以下の定理を得る．実用的には $r \approx r_{\mathrm{R}}$ である．

定理 1. 任意の非負整数 b について，$O((1+b)\cdot(r+r_{\mathrm{R}}))$ 語のデータ構造が存在し，以下の操作を検索の任意の時点，表記の時間で実行できる．

- *left-extension:* $O(\sigma \log\log_w(\sigma + n/r))$
- *right-extension:* $O(\sigma \log\log_w(\sigma + n/r_{\mathrm{R}}))$
- *left-contraction:*
$$\begin{cases} O(\sigma \log\log_w(\sigma + n/r)) & (|P| \leq b+1) \\ O(\sigma \log\log_w(\sigma + n/r) + occ) & (|P| > b+1) \end{cases}$$
- *right-contraction:*
$$\begin{cases} O(\sigma \log\log_w(\sigma + n/r_{\mathrm{R}})) & (|P| \leq b+1) \\ O(\sigma \log\log_w(\sigma + n/r_{\mathrm{R}}) + occ) & (|P| > b+1) \end{cases}$$
- *count:* $O(1)$

- *locate*: $O(occ)$

ここで σ はアルファベットサイズ，w は計算機の語長，occ は P の T における出現数である．

パラメータ b は，短いパターンに対する縮小を高速に行うために導入される．短いパターンは T における出現数 occ が大きくなるため，基本的な縮小アルゴリズムの時間計算量 $O(\sigma \log\log_w(\sigma + n/r) + occ)$ が大きくなると推測される．そこで，predecessor データ構造 $B_l, B_l^{\mathrm{R}} (l = 0, 1, \ldots, b-1)$ を追加し，長さ $b+1$ 以下のパターンの縮小にこれを用いることで時間計算量から項 occ を削減する．ここで，b の大きさに応じた，実行速度と索引サイズのトレードオフが発生する．

また，wavelet tree を用いた range sum データ構造 [5] を連長圧縮した BWT の表現に応用することで，$left[right]$-$extension, left[right]$-$contraction$ の実行時間のうち，アルファベットサイズ σ に依存する項 $\sigma \log\log_w(\sigma + n/r[r_{\mathrm{R}}])$ を依存しない項 $\frac{1}{\epsilon}\log^{2+\epsilon} r[r_{\mathrm{R}}]$ に置き換えた，σ の大きい文字列に対してより有効と考えられるバリアントを提案した．

加えて，br-index の省メモリな構築アルゴリズムも提案した．構築は 2 段階に分けて行う．はじめに prefix-free parsing (PFP) [1] を用いて，SA と BWT を省メモリに構築する．次に，こうして構築された SA と BWT を基に br-index のコンポーネントを全て構築する．

4 実験

br-index を実装し，反復の多い文字列データに対するパフォーマンスを実験を通して評価した．はじめにパラメータ b の値に起因する時空間トレードオフを評価した．$b = 15$ 程度の値が，短いパターンに対する縮小操作を十分に高速にしつつ，サイズの増加が許容範囲に収まる値として優れていることが判明した．推定した値を踏まえて，同等の機能を持つ他の圧縮文字列索引と性能比較を行った．図 1 に示すように，ヒト 19 番染色体配列に対する構築では，配列数の増加に対してサイズの増加が抑えられ，また構築時のピークメモリ消費量も索引サイズに近い値をとった．一方，構築時間に関しては図 2 から分かるように比較した索引のうち 2 番目に遅く，課題であった．検索クエリの実行性能は最も高く，一例としてヒト 19 番染色体データにおける MEMs+locate クエリでは，図 3 に示されるように最も高速だった．

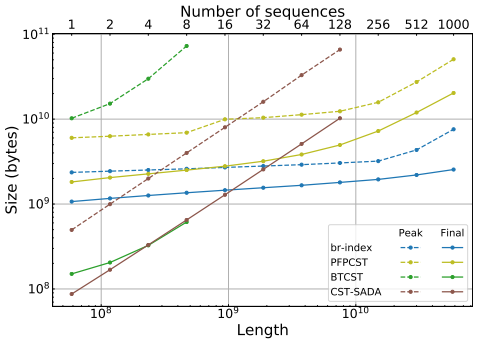

図 1. ヒト 19 番染色体データに対する索引サイズと構築時ピークメモリ消費量．

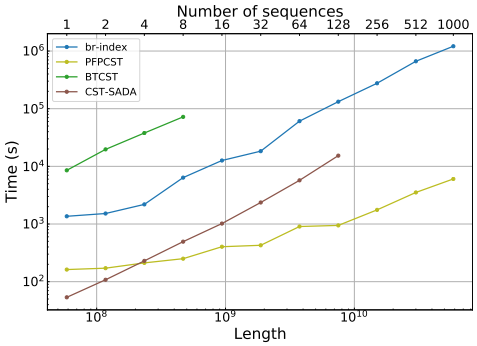

図 2. ヒト 19 番染色体データに対する構築時間．

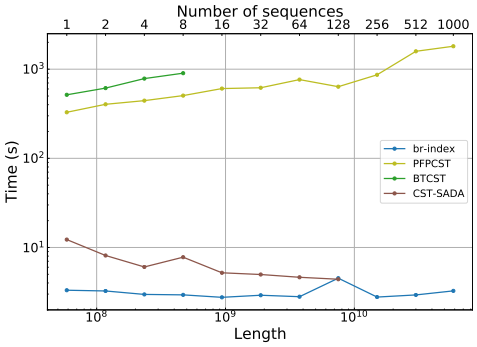

図 3. ヒト 19 番染色体データに対する MEMs+locate クエリの実行時間．

参考文献

[1] Christina Boucher, Travis Gagie, Alan Kuhnle, Ben Langmead, Giovanni Manzini, and Taher Mun. Prefix-free parsing for building big BWTs. *Algorithms for Molecular Biology*, 14(1):1–15, 2019.

[2] Michael. Burrows and David J. Wheeler. A block sorting lossless data compression algorithm. *Digital SRC Research Report*, 1994.

[3] Travis Gagie, Gonzalo Navarro, and Nicola Prezza. Fully functional suffix trees and optimal text searching in BWT-runs bounded space. *Journal of the ACM*, 67(1):1–54, 2020.

[4] Udi Manber and Gene Myers. Suffix arrays: a new method for on-line string searches. *SIAM Journal on Computing*, 22(5):935–948, 1993.

[5] Gonzalo Navarro. Document listing on repetitive collections with guaranteed performance. *Theoretical Computer Science*, 772:58–72, 2019.