

円形デカルト木照合のための コンパクト複数テキスト索引

Roman Pauli

Dominik Köppl

スライド (クリック / スキャン)

コンピュテーション研究会

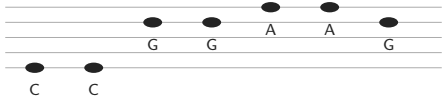
2026 年 9 月



動機：音高そのものではなく「形」を探す

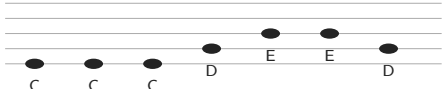
同じ旋律の形でも、移調・音域・局所的な変形により異なる音高列として現れる。

Twinkle



C, C, G, G, A, A, G

Row

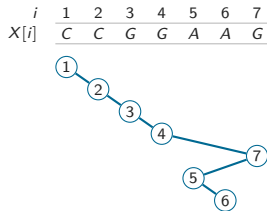


C, C, C, D, E, E, D

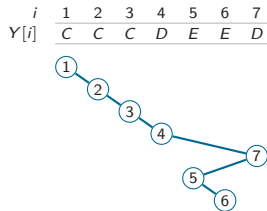
この2列は異なるが、**デカルト木の形は同じ**である。

Park et al. (2019)

デカルト木照合 (CTM)



$X = C, C, G, G, A, A, G$



$Y = C, C, C, D, E, E, D$

2つの音高列は異なるが、
同じデカルト木をもつ（同値は左端を優先する）。

X, Y を列とすると、

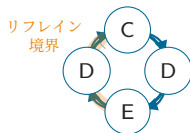
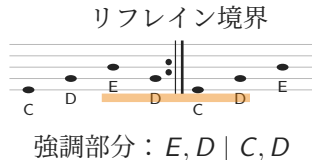
$$X \approx_{\text{CT}} Y \iff \text{ct}(X) = \text{ct}(Y).$$

以降、**デカルト木照合**を
CTM と略記する。

- ▶ 最小値を根とし、
左右の部分列へ再帰する。
- ▶ 各ノードを対応する位置
に揃えると、同一の木構造
が明確になる。
- ▶ 絶対値は消え、
再帰的な極値構造だけが
残る。

円形検索：リフレインには境界がない

リフレイン $R = (C, D, E, D)$ の円形の出現は末尾付近から始まり先頭へ続く。
細菌の染色体や **mtDNA** も同様に環状である。



反復したリフレイン：

$$R^\omega = C, D, E, D \mid C, D, E, D \mid \dots$$

3 番目から開始：

$$E, D \mid C, D, E, D \mid C, D$$

移調したクエリ：

$$B, A \mid G, A, B, A \mid G, A$$

問題設定：

円形テキスト集合 $T_1, \dots, T_k$ を

索引化し、

クエリ P に対し

$$\text{count}(P) = \#\{\text{円形回転 } R : \text{ct}(R[1..|P|]) = \text{ct}(P)\},$$

すなわち全テキストにわたる**開始位置**を数える。

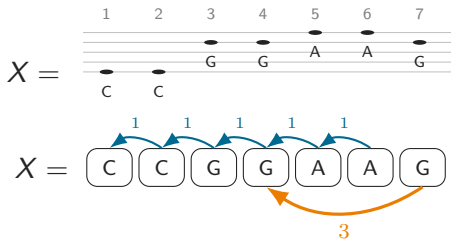
本研究の貢献

1. CTM のための円形 eBWT (ceBWT) の**初の実装**。
動的版と**静的版**の 2 種類。
2. 構築時のテキスト複製を、従来の **4 コピー**から **3 コピー**へ削減。
3. KMP 走査および完全一致・順序保存索引との**系統的な比較評価**。

代表値：静的索引は **0.50 bytes/symbol**、2,000 クエリを **0.08 秒**で処理する。

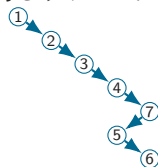
親距離符号化 (PDE) : 木を列として表す

位置 i から左を見て、音高が現在以下で最も近い先行音を探し、そこまでの距離を記録する。存在しなければ i を記録する。



$$\text{PDE}(X) = 1111113.$$

同じデカルト木



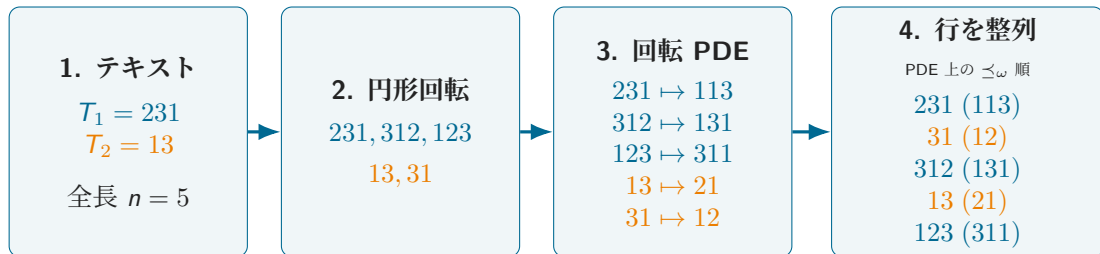
PDE は木の形を表す列である。

同じ木 $\iff$ 同じ PDE

2 つ目の例 : $Y = C, C, C, D, E, E, D \implies \text{PDE}(Y) = 1111113.$

円形テキストでは 2 コピーを符号化し後半だけを残す (回転 PDE)。

具体例で見る ceBWT の流れ



- ▶ クエリ $P = 453$ は 231 と同じデカルト木をもち、 $\text{ct}(453) = \text{ct}(231)$ 同値に、両者の PDE は 113
- ▶ **count クエリ** は、この形をもつ索引済み円形回転の個数を返す。
- ▶ 231 の行だけが一致するので $\text{count}(P) = 1$ 。

索引はこの PDE 行そのものを保持しない。**回転デカルト木シグネチャ**から導いた 2 配列 F/L を保持し、LF 写像による**後方探索**で答える (FM-index と同様)。 F/L の具体例は付録。

動的 ↔ 静的 ceBWT

処理内容が変われば、同じ索引を相互に変換できる。

動的 ceBWT

新しいテキストを挿入できる

F 動的 wavelet tree

L 動的 wavelet tree

LCP^∞ 動的 wavelet tree

E 動的 bit vector

構築時
のみ

重い：複数の完全動的構造を同時に維持する必要がある。

圧縮

更新用に
復元

静的 ceBWT

count クエリ向けに最適化

B_F 静的 bit-vector 群

B_L 静的 bit-vector 群

かなり軽い：構造数が少なく、すべて静的でコンパクトである。

要点：テキスト追加中は動的版、追加後はコンパクトで高速な **count 索引** として静的版を使う。

F/L は両版でクエリに使い、 LCP^∞ と E は構築時にのみ使う。

動的挿入：なぜテキストを反復するのか？

従来方式：常に **4 コピー** ($T^4\$$)。

本研究：最初のテキストは **2 コピー**、以後は **3 コピー**。

$T = a_1 b_1 c_1 d_1$ を一時的に反復

$T^4\$ =$

a_1	b_1	c_1	d_1
-------	-------	-------	-------

a_2	b_2	c_2	d_2
-------	-------	-------	-------

a_3	b_3	c_3	d_3
-------	-------	-------	-------

a_4	b_4	c_4	d_4
-------	-------	-------	-------

 $\$$

$T^3\$ =$

a_1	b_1	c_1	d_1
-------	-------	-------	-------

a_2	b_2	c_2	d_2
-------	-------	-------	-------

a_3	b_3	c_3	d_3
-------	-------	-------	-------

 $\$$

添字 = 由来するコピー

- ▶ $\preceq_\omega$ で長さ n の 2 つの回転を比較するには最大 **$3n$ 記号**が必要 (Lemma 3.3)。
4 コピーは常に十分な比較範囲を与える。
コピーは一時的で、最終 ceBWT には現れない。
- ▶ 挿入時に **3**：新テキストを**最小記号が先頭に来る**よう回転させ $\text{Rot}(R, i)^3\$$ を索引化する (Lemma 5、弱周期補題)。
- ▶ 初回は **2**：単一テキストの回転はすべて同じ長さなので、 F の**整合性チェック**で影響を受ける 1 箇所を直せば足りる (Lemma 2)。

	ゲノム ($\sigma = 4$)	MIDI ($\sigma \leq 128$)
データ	12 × E. coli K-12 MG1655, 24,000 塩基/本	
	16 実ゲノム (数千–50,000 塩基)	203 ファイル, トークン値 [0..127]
	16 × イネ (<i>O. sativa</i>) 部分列	109–7,079 トークン、中央値 1,470
全長	288,000 塩基 (E. coli 群)	363,164 トークン
パターン	50 × 1000 塩基	50 × 40 長さ ($m = 25, \dots, 1000$) = 2,000
使用箇所	結果 1–2 (構築・追加)	結果 3–5 (クエリ)

比較手法：

- ▶ ceBWT: 動的 / 静的
- ▶ CTM: KMP (PDE / CT シグネチャ)
- ▶ OPM-FM (順序保存)
- ▶ 完全一致索引: SDSL, RLCSA, r-index

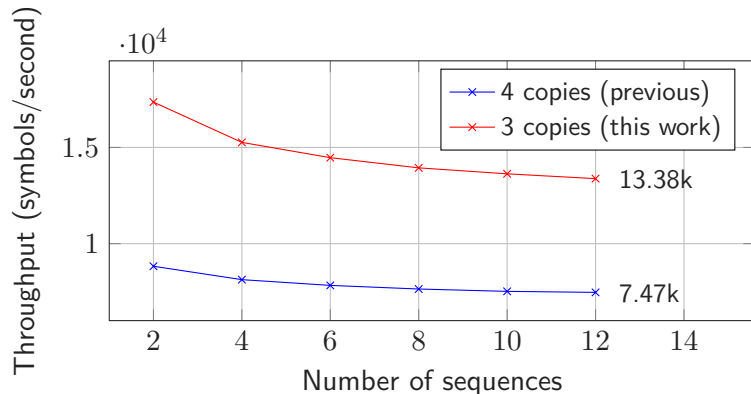
結果スライドの記法

- ▶ **OPM** = 順序保存照合 (**完全な** 相対順序を保存。CTM より厳しい)
- ▶ **OI** = 連結列に 1 つの索引
- ▶ **MI** = 入力テキストごとに 1 つの索引

構築時間、メモリ、索引サイズ、クエリ時間、スケーラビリティを測定する。

AMD Ryzen 9 5900XT, 128 GiB, g++ 14.2.0 -Ofast。ceBWT は全実験で CAsStorage を使用。

結果 1：反復削減により構築が高速化



観察

12 系列 = 288,000 塩基では：

7.47k $\rightarrow$ 13.38k

symbols/s

構築時間: 38.6 s $\rightarrow$

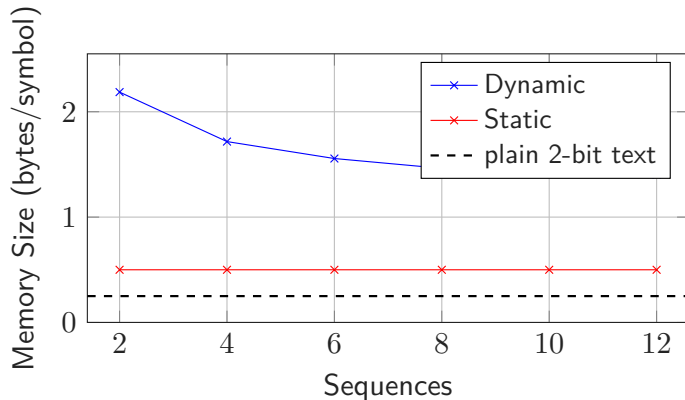
21.5 s

論理的に同じ ceBWT を保ったまま、一時的な作業量を削減する。

4 コピー版は削減版のほぼ **2 倍** の入力进行处理するため、実測の 1.79 倍と整合する。

Pauli-Osterkamp-Köppl, arXiv, Fig. 9 data

結果 2：静的化により索引を大幅に圧縮



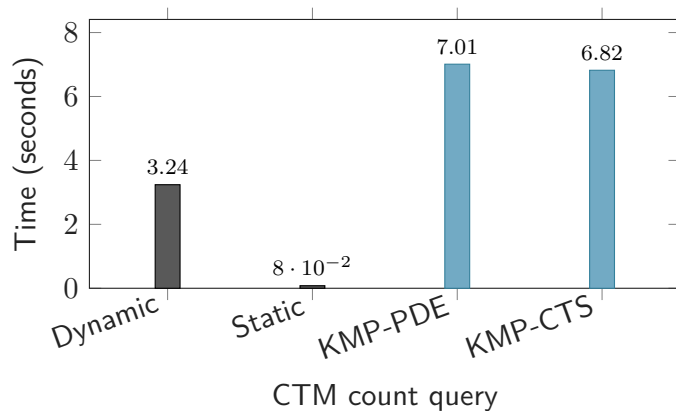
観察

- ▶ 動的: 2.19 → 1.40 bytes/symbol
- ▶ 静的: 0.50 bytes/symbol = 4 bits
- ▶ テキスト数が増えても一定
- ▶ 288 k 塩基: 144 kB / 動的 403 kB

更新が不要になれば静的版へ変換でき、索引は大幅に小さくなる。参考：テキストの 2-bit 符号化のみで 0.25 bytes/symbol (破線)、ただしクエリは不可。

Pauli-Osterkamp-Köppel, arXiv, Fig. 11a

結果3：構築済み索引のクエリは KMP 走査より高速

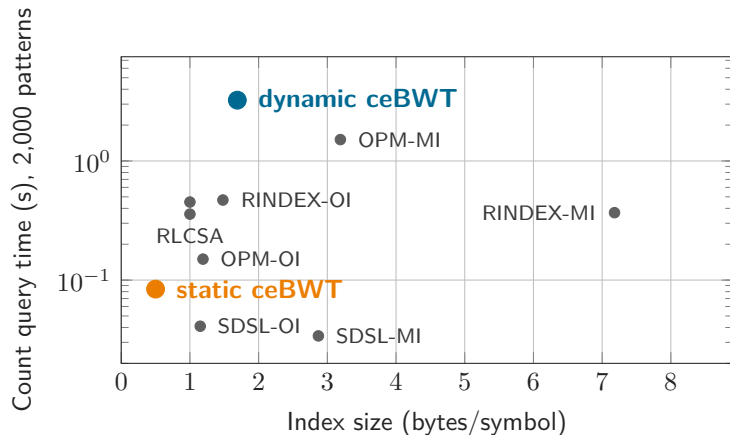


2,000 個の MIDI パターンに対して：

- ▶ 動的 ceBWT: 3.24 s
- ▶ **静的 ceBWT: 0.08 s**
- ▶ KMP-PDE: 7.01 s
- ▶ KMP-CTS: 6.82 s

同じコレクションへ多数のクエリを行う場合、索引構築のコストを償却できる。

結果4：サイズと速度のトレードオフ



観察

- ▶ 静的 ceBWT: 最小の索引、かつ OPM 両版より高速
- ▶ 完全一致索引 (SDSL) はより高速だが別の問題を解いている
- ▶ OPM-FM は構築が大幅に安い (25.7k 対 8.5k symbols/s)

工学的な比較であり意味論的な比較ではない。

exact $\subseteq$ OPM $\subseteq$ CTM で、一致数は 4,418 / 7,039 / **9,745** (付録参照)。

Pauli-Osterkamp-Köpl, arXiv, Figs. 19, 20

まとめ

1. ceBWT を動的版と静的版の両方として実装した。
2. 構築時の複製を 4 コピーから 3 コピーへ削減し、スループットを 1.79 倍にした。
3. 静的版は約 0.5 bytes/symbol まで圧縮でき、多数の CTM クエリを非常に高速に処理する。

ceBWT の実用上の強みは
「動的に収集し、その後静的に圧縮する」
という二段階ワークフローである。

<https://github.com/koepp1/cebwt>

ご清聴ありがとうございました。

付録：理論的位置づけ

- ▶ Kim–Cho (CPM 2021): 単一テキスト CTM のコンパクト BWT 型索引
- ▶ Osterkamp–Köppl (CPM 2025): $O(n \log \sigma)$ bits で構築し、複数円形テキストへ拡張
- ▶ 本研究：その ceBWT の動的・静的実装と系統的評価

理論構築時間：

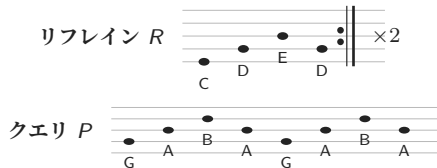
$$O\left(n \frac{\log \sigma \log n}{\log \log n}\right), \quad \text{空間 } O(n \log \sigma) \text{ bits.}$$

Osterkamp–Köppl, CPM 2025

付録：2 つの応用領域

音楽：リフレインは移調して再現する

- ▶ 短いリフレインは繰り返されることが多い。
- ▶ クエリは同じモチーフを**移調して反復**したものでもよい。
- ▶ 音高は異なってもデカルト木の形は保存される。



$$R = C, D, E, D, \quad P = G, A, B, A, G, A, B, A$$
$$R^2 \approx_{CT} P.$$

RNA ウイルスゲノム

変異の速いゲノムでは、完全一致より頑健な照合概念が有用である。

例：SARS-CoV-2 や MERS-CoV などの RNA ウイルス

ゲノムはしばしば環状

細菌の染色体・プラスミドや mtDNA は物理的に**環状**で、自然な先頭塩基がない。

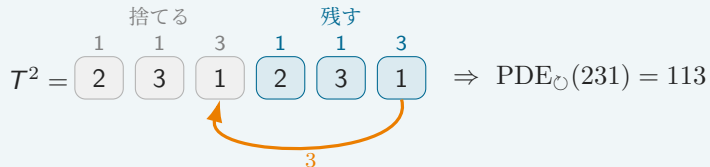
ACGUAC**G**UACGUACGU

ACGUAC**A**UACGU**G**CGU

共通点：値が変化しても、繰り返される**構造パターン**は有用な検索対象として残る。

付録：回転 PDE

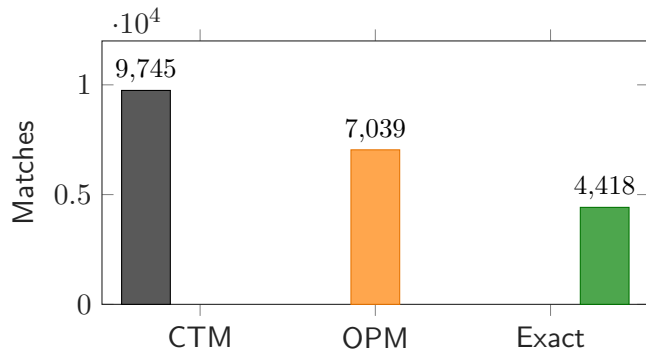
円形回転には先頭がないため「親なし $\Rightarrow i$ を記録」が定義できない。
そこで 2 コピーを符号化し、後半だけを残す。



橙の親はコピー境界を越える。これが符号化を円形にしている。残す後半では「親なし $\Rightarrow i$ 」の場合は起こらない。

全テキストの全回転に対してこれを行うと、索引が整列する行が得られる。

付録：照合モデルにより候補数が異なる



$$\text{exact} \subseteq \text{OPM} \subseteq \text{CTM}$$

- ▶ 完全一致: 4,418 一致
- ▶ OPM: 7,038–7,039
- ▶ **CTM: 9,745**

一致数が多いのは**より弱い照合意味論**の結果であり、実装の誤りではない。

したがって前スライドはサイズと速度の比較であり、品質の比較ではない。

Pauli–Osterkamp–Köppl, arXiv, Fig. 22; 2,000 MIDI patterns

付録：F, L と静的表現の具体例

単一テキスト 5473 とその 4 つの回転 ($\preceq_\omega$ 順)。

動的						
i	CA	回転	PDE	F	L	LCP^∞
1	4	3547	$\infty, 1, 2, 1$	3	0	0
2	2	4735	$\infty, 1, \infty, 1$	1	0	1
3	3	7354	$\infty, \infty, 1, 2$	0	1	1
4	1	5473	$\infty, \infty, 1, \infty$	0	3	2

静的				
	1	2	3	4
F	3	1	0	0
$B_F(c=0)$	1	1	0	0
$B_F(c=1)$	1	0		
$B_F(c=2)$	1			
$B_F(c=3)$	0			
L	0	0	1	3
$B_L(c=0)$	0	0	1	1
$B_L(c=1)$	0	1		
$B_L(c=2)$	1			
$B_L(c=3)$	0			

B_A^c は A の値が c 以上の位置を走り、値が c の位置でビット 0 となる。
この 2 群で F/L を $3n + o(n)$ bits に符号化し、同じ後方探索を $O(|P|)$ 時間で行える。

付録：動的挿入 II（縫い込みと切り詰め）

挿入前の ceBWT

挿入前

既存行
既存行
既存行
:
既存行
既存行
既存行

2. 符号化した全行を縫い込む

一時行をすべて挿入し、各所へ散らばり得る

既存行
$c_2 \ d_2 \ a_3 \ b_3$
既存行
$a_1 \ b_1 \ c_1 \ d_1$
既存行
$d_3 \ a_4 \ b_4 \ c_4$
$b_1 \ c_1 \ d_1 \ a_2$
既存行
$d_2 \ a_3 \ b_3 \ c_3$
$a_3 \ b_3 \ c_3 \ d_3$
既存行
$c_1 \ d_1 \ a_2 \ b_2$
$b_2 \ c_2 \ d_2 \ a_3$
$a_2 \ b_2 \ c_2 \ d_2$
既存行
$c_3 \ d_3 \ a_4 \ b_4$
$d_1 \ a_2 \ b_2 \ c_2$
$b_3 \ c_3 \ d_3 \ a_4$

3. 補助行を削除

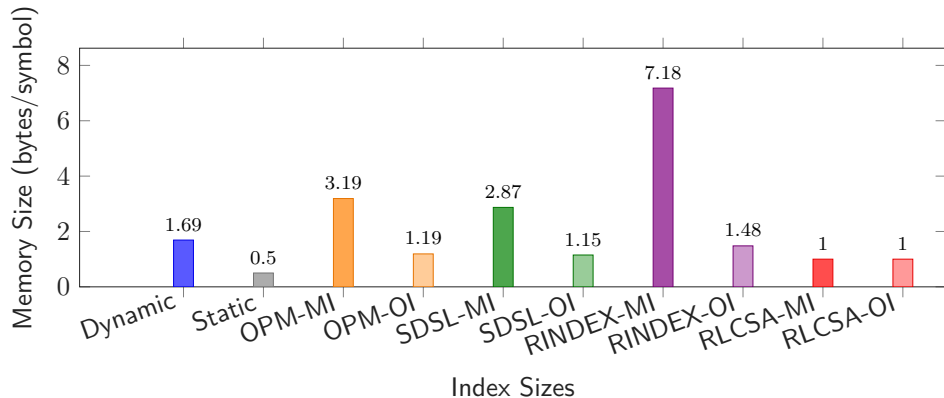
最終 ceBWT

既存行
既存行
$a_1 \ b_1 \ c_1 \ d_1$
既存行
既存行
$d_2 \ a_3 \ b_3 \ c_3$
既存行
$b_2 \ c_2 \ d_2 \ a_3$
既存行
既存行
$c_3 \ d_3 \ a_4 \ b_4$

LF/FL
ループ

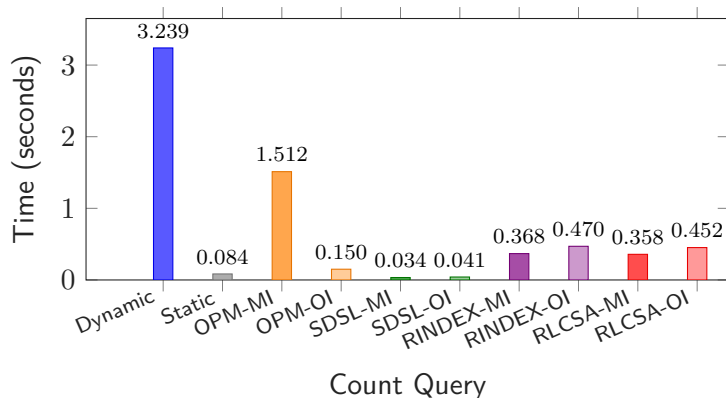
一時行は ceBWT 全体へ散らばり得る。切り詰め後には、元の各位置につき 1 つの符号化行だけが残る。

付録：索引サイズ（棒グラフ）



照合モデルが異なるため、これは意味論ではなく工学的な比較である。
静的 ceBWT はここでは **0.50 bytes/symbol** で最小である。

付録：カウントクエリ時間（棒グラフ）



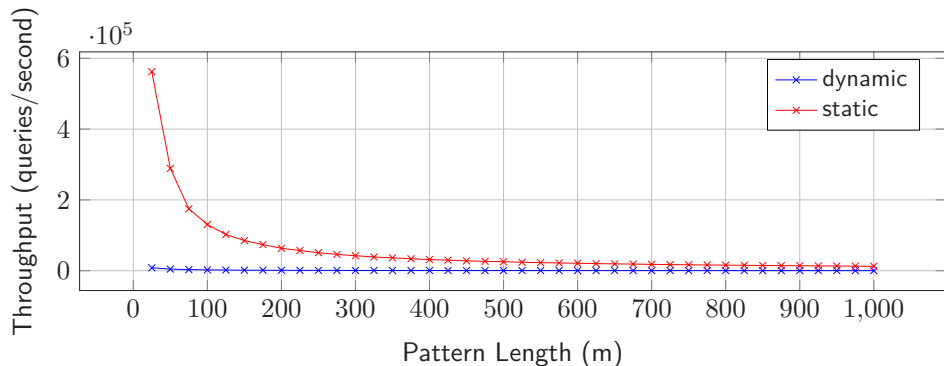
OPM = 順序保存照合（完全な相対順序を保存）。
OI = 連結列に 1 つの索引; MI = 入力テキストごとに 1 つの索引。

2,000 個の MIDI パターンに対して：

- ▶ 静的 ceBWT: 0.084 s
- ▶ OPM-OI: 0.150 s
- ▶ OPM-MI: 1.512 s
- ▶ 動的 ceBWT: 3.239 s

完全一致索引はさらに高速だが、異なる照合問題を解いている。

付録：パターン長とクエリスループット



静的版では plain bit vector 上の $O(1)$ rank/select が効き、
動的版との大きな差を生む。