Abschlussthemen in der Datenkompression / Sequenzanalyse / Bioinformatik

Institut für Theoretische Informatik
enno.ohlebusch@uni-ulm.de
uwe.baier@uni-ulm.de

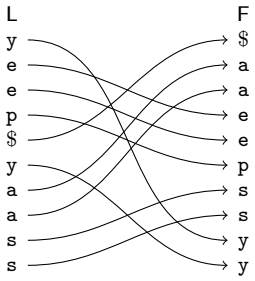
Die Burrows-Wheeler-Transformation (BWT) [2] ist eine invertierbare Permutation eines gegebenen Eingabetextes S. Die BWT ist aus verschiedenen Gründen sehr nützlich für Datenkompression als auch Sequenzanalyse. Zur Konstruktion werden alle Suffixe eines Strings S lexikographisch sortiert; die BWT ergibt sich als die Konkatenation der zyklisch vor den Suffixen stehenden Buchstaben.

i	$\mathsf{SA}[i]$	$S[1..\mathsf{SA}[i]]$	$S_{\mathsf{SA}[i]}$	$\mathsf{L}[i]$	$S_{\mathsf{SA}[i]}$
1	10	easypeasy	$	y	$
2	7	easype	asy$	e	asy$
3	2	e	asypeasy$	e	asypeasy$
4	6	easyp	easy$	p	easy$
5	1	ε	easypeasy$	$	easypeasy$
6	5	easy	peasy$	y	peasy$
7	8	easypea	sy$	a	sy$
8	3	ea	sypeasy$	a	sypeasy$
9	9	easypeas	y$	s	y$
10	4	eas	ypeasy$	s	ypeasy$

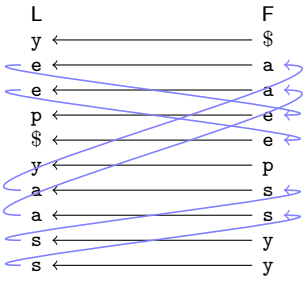
Abbildung 1: Suffix Array SA, Präfixe $S[1..\mathsf{SA}[i]]$ der sortierten Suffixe sowie BWT L des Strings $S = \texttt{easypeasy\$}$.

Abbildung 1 zeigt ein Beispiel einer BWT L für den Text $S = \texttt{easypeasy\$}$. Zur Rekonstruktion des Originaltextes aus einer BWT L kann wie folgt vorgegangen werden: Man schreibe neben L den jeweils ersten Buchstaben der sortierten Suffixe, die sogenannte F-Spalte. Nun besteht eine Korrespondenz zwischen L und F wie folgt:

Sei i ein Eintrag im obigen Suffixarray (Abbildung 1) mit Suffix S_j, und sei $\mathsf{L}[i]$ das k-te Auftreten des Buchstabens $\mathsf{L}[i]$ in L. Dann kann das Suffix $\mathsf{L}[i]S_j$ an derjenigen Stelle in F gefunden werden, an der der Buchstabe $\mathsf{L}[i]$ zum k-ten Mal in F auftritt (siehe Abbildung 2). Anders gesagt, mit dieser Korrespondenz ist es möglich, den Text von hinten nach vorne wieder aus einer BWT zu rekonstruieren. F kann im übrigen durch eine einfache Sortierung der Buchstaben aus L gewonnen werden, also ist zur Rekonstruktion lediglich L notwendig.



(a) F kann aus L per Sortierung berechnet werden



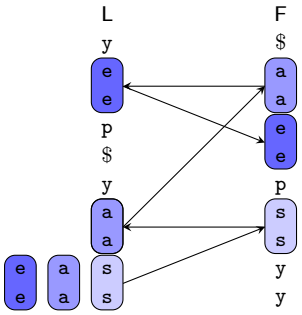
(b) k-ter Buchstabe in L entspricht k-tem Buchstaben in F

Abbildung 2: Korrespondenz zwischen L und F aus Abbildung 1.

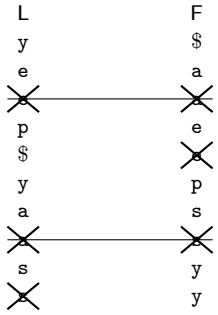
Wie in Abbildung 1 einzusehen ist, besitzen sortierte Suffixe oftmals ein längeres gemeinsames Präfix, das im Text vor den Suffixen steht, im Beispiel z.B. der Text `eas` an den Positionen 9 und 10, ein sogenannter $d - [i, j]$-Block, wobei d der Präfixlänge minus 1 und $[i, j]$ dem Intervall des gemeinsamen Suffixes entspricht. Dies kann ausgenutzt werden, um eine BWT weiter zu komprimieren (siehe Abbildung 3):

1. Man stellt zuerst fest, wo sich die Spalten eines Blockes in der BWT befinden
2. Man streiche alle Einträge in L für jede in der BWT befindliche Spalte, außer den Einträgen in der ersten Zeile
3. Man streiche die korrespondierenden Einträge der Streichungen in F
4. Positionen, die in L und F gestrichen werden, können komplett entfernt werden

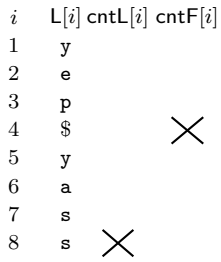
Als Resultat erhält man so eine gekürzte BWT L sowie zwei Bitvektoren cntL und cntF, die die restlichen Streichungen in L bzw. F beinhalten. Interessanterweise ist auch diese



(a) Spalten des Blockes in L und F identifizieren

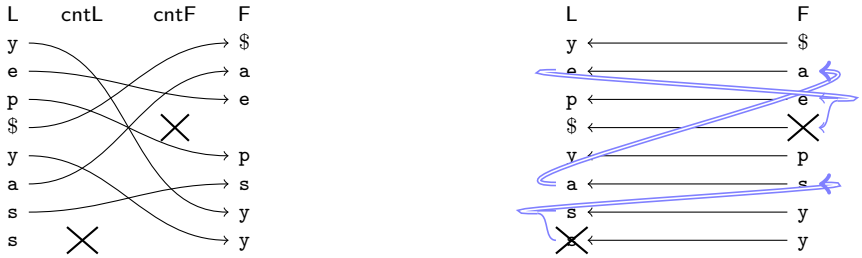


(b) Streichen und doppelte Streichungen entfernen



(c) „Getunnelte“ BWT

Abbildung 3: Beispielhaftes „Tunneln“ des $2 - [9, 10]$-Blockes mit Text `eas`.



(a) F kann aus L per Sortierung berechnet werden

(b) Idee der Rekonstruktion: benutze die obere Reihe für den ganzen Block

Abbildung 4: Korrespondenz zwischen L und F in einer getunnelten BWT.

gekürzte Version wieder invertierbar: Spalte F kann dadurch gewonnen werden, dass diejenigen Buchstaben, die in L nicht gestrichen wurden, sortiert auf die freien Plätze gemäß der Streichung in F aufgeteilt werden. Für alle Einträge in L, die nicht gestrichen wurden, besteht wieder die Korrespondenz zwischen k-tem Buchstaben in L und k-tem Buchstaben in F. Für Einträge in L, die gestrichen sind, kann zur obersten Zeile des Blockes gewechselt, von dort das Präfix wiederhergestellt, und anschließend wieder zur ehemaligen Zeile des Blockes gewechselt werden, siehe Abbildung 4.

Da die Bitvektoren cntL und cntF sehr gut zu komprimieren sind, und L im Normalfall um viele Einträge gekürzt wird, entsteht so eine deutlich besser komprimierbare Repräsentation, die je nach den weiteren benutzten Verfahren eine Verkleinerung der BWT-Kodierung um $8 - 16\%$ im Durchschnitt und $30 - 55\%$ in den besten Fällen erlaubt (Abbildung 5; [1]).

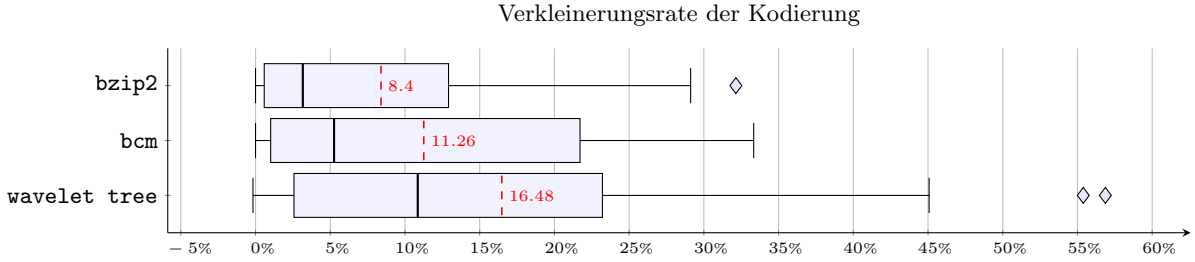


Abbildung 5: Verbesserung der Kompression durch „Tunneling“, bezogen auf verschiedene BWT-Komprimierungsverfahren und dargestellt als Boxplots. Die Boxplots enthalten unteres Quartil, Median und oberes Quartil, die durchschnittliche Verbesserungsrate ist als rote gestrichelte Linie aufgetragen.

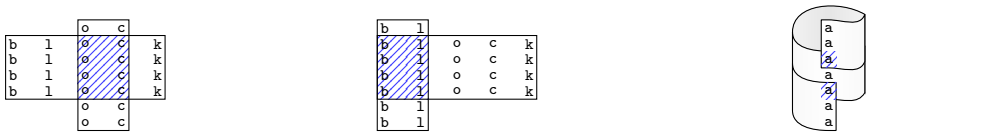
[1] Uwe Baier. On Undetected Redundancy in the Burrows-Wheeler Transform. `https://arxiv.org/abs/1804.01937`, 2018.

[2] Michael Burrows and David J Wheeler. A block-sorting lossless data compression algorithm. Technical Report 124, Digital Equipment Corporation, 1994.

Thema 1: Blockwahl

Blöcke bilden den Ausgangspunkt für das „Tunneln“ einer BWT, daher ist die Blockwahl für die zu erwartende Kompressionsverbesserung entscheidend. Problematisch an der Blockwahl ist der Aspekt, dass jeder zusätzlich getunnelte Block die Komprimierbarkeit der Bitvektoren verringert, d.h. es lohnt sich nicht unbedingt, jeden Block zu tunneln.

Weiter können auch Kollisionen zwischen Blöcken bestehen, siehe Abbildung 6. Auch kollidierende Blöcke können getunnelt werden, solange ihr Überschneidungsbild die Form eines Kreuzes hat, hier spricht man von einer kompensierbaren Kollision [1].



(a) Kompensierbare Kollision (b) Kritische Kollision (c) Kritische Eigenkollision

Abbildung 6: Beispiele für Blockkollisionen.

Kollisionen erschweren jedoch die Blockwahl, da sie Seiteneffekte nach sich ziehen, die die Größe eines Blocks je nach Wahl der bisherigen Blöcke verringern kann. Bisher [1] werden die Blöcke gemäß einer Greedy-Strategie gewählt, wobei nach der Wahl eines Blockes die Größen der kollidierenden Blöcke angepasst werden, wodurch eine Art sortierte Liste von Blöcken entsteht. Innerhalb dieser Liste werden nun die t-jenigen größten Blöcke gewählt, für die ein bestmöglicher Kompressionsgewinn erzielt wird. Das Verfahren funktioniert zwar recht gut, ist aber recht kompliziert, sehr zeit- und speicheraufwendig und liefert auch nicht immer eine optimale Blockauswahl.

In diesem Thema sollen daher alternative Ansätze untersucht werden, wie z.B.

- Wahl aller möglichen Blöcke
- Wahl der größtmöglichen Blöcke mit Abbruchbedingung
- optimale Wahl, z.B. per Branch and Bound

Weiter soll auch untersucht werden, inwiefern Kollisionsbehandlung sich auf die Blockauswahl auswirkt. Folgende mögliche Kollisionsbehandlungen sind denkbar:

- Kollisionen ignorieren
- Wahl auf kollisionsfreie Blöcke beschränken
- exakte/approximative Kollisionsbehandlung

Ziel der Arbeit ist es durch Vergleich der Ansätze herauszuarbeiten, welche Strategien sich hinsichtlich Optimalität und Resourcennutzung für die Praxis eignen.

[1] Uwe Baier. On Undetected Redundancy in the Burrows-Wheeler Transform. `https://arxiv.org/abs/1804.01937`, 2018.

Thema 2: Textindizierung

Eine wichtige Aufgabe in der Sequenzanalyse besteht darin, einen Text derart aufzubereiten, um in ihm diverse Operationen möglichst effizient durchführen zu können (z.B. Patternsuche, Spektren von Teilstrings bestimmen, ...). Dieser Begriff ist im Allgemeinen unter Textindizierung zusammengefasst, welche in dem Versuch besteht, einen Text so mittels Datenstrukturen darzustellen, dass die oben genannten Operationen möglichst effizient durchführbar sind.

Ein recht einfacher und sehr beliebter Textindex besteht dabei aus dem Wavelet Tree der BWT eines Textes, gemeinhin bekannt als FM-Index [2]. Der FM-Index unterstützt viele Operationen sehr effizient, und kann zusätzlich auch noch in komprimierter Form gespeichert werden. Diese ist oftmals kleiner als der Ausgangstext und stellt so einen signifikanten Mehrwert zur normalen Textdarstellung dar.

Das sogenannte „Tunneln“ lässt sich fast eins zu eins auch auf den FM-Index anwenden, da sich ein Rückwärtssuchschritt in einer getunnelten BWT mittels

$$\mathsf{select}_{\mathsf{cntF}}(1, \mathsf{rank}_{\widetilde{\mathsf{cntL}}}(1, \mathsf{C}_{\mathsf{L}}[\mathsf{L}[i]] + \mathsf{rank}_{\mathsf{L}}(\mathsf{L}[i], i)))$$

darstellen lässt [1]. Leider besteht beim Tunneln aber noch das Problem, bei der Patternsuche die richtige Anzahl an Suchtreffern zu ermitteln, da ein Suchtreffer auch innerhalb eines Tunnels liegen kann, und so statt n-fach nur einfach gezählt wird. Das Problem lässt sich prinzipiell lösen, indem man bis zum Ende eines jeden Tunnels navigiert und hier die entsprechende Zahl ausgibt, ist aber im Detail etwas trickreicher.

Das Ziel dieser Arbeit ist es, die angesprochene Problematik zu lösen und so einen voll funktionsfähigen getunnelten FM-index zu beschreiben und auch zu implementieren. Weiter ist auch vorstellbar, zusätzliche Indextechniken wie in [3] beschrieben zu verwenden.

[1] Uwe Baier. On Undetected Redundancy in the Burrows-Wheeler Transform. `https://arxiv.org/abs/1804.01937`, 2018.

[2] Paolo Ferragina and Giovanni Manzini. Indexing Compressed Text. *Journal of the ACM*, 52(4):552–581, 2005.

[3] Travis Gagie, Gonzalo Navarro, and Nicola Prezza. Optimal-Time Text Indexing in BWT-runs Bounded Space. `https://arxiv.org/abs/1705.10382`, 2017.

Thema 3: Komprimierte deBruijn-Graphen

Ein deBruijn-Graph ist eine in der Bioinformatik beliebte Möglichkeit, einen Text (z.B. ein oder mehrere aneinandergehängte Genome) als Graph darzustellen. Hierbei wird zunächst für jeden k-Zeichen langen Substring des Textes ein Knoten erstellt. Zwei Knoten werden immer dann mit einer Kante verbunden, wenn es eine Stelle im Text gibt, an der sich die entsprechenden Substrings im Text um $k-1$ Zeichen überlappen (das bedeutet, dass auch Mehrfachkanten erlaubt sind), siehe Abbildung 7.

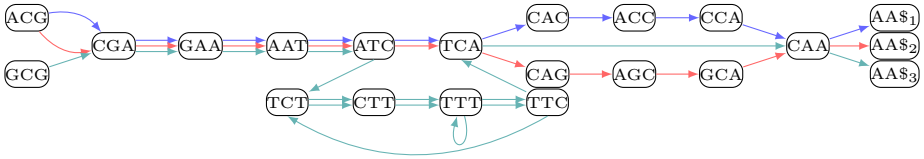


Abbildung 7: Kombinierter deBruijn-Graph für $k = 3$ und die Texte $S_1 =$ ACGAATCACCAA$\$_1$, $S_2 =$ ACGAATCAGCAA$\$_2$ und $S_3 =$ GCGAATCTTTCTTTTCAA$\$_3$.

Da derartige Graphen sehr groß werden können, ist man an Möglichkeiten interessiert, den Graph zu komprimieren. Dies geschieht in Form von komprimierten deBruijn-Graphen: Immer wenn zwei Knoten die einzigen Vorgänger bzw. Nachfolger voneinander sind, können sie verschmolzen werden, siehe Abbildung 8. Durch sukzessives Verschmelzen von Knoten entsteht so ein deutlich kleinerer Graph, was bei großen Datenmengen entscheidend ist.

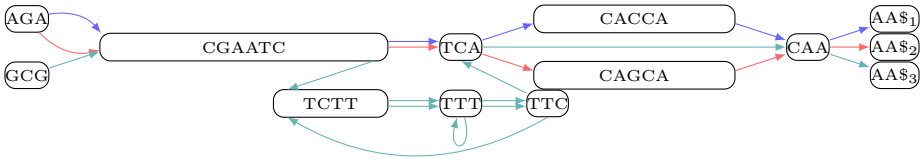


Abbildung 8: Komprimierter deBruijn-Graph aus Abbildung 7.

Komprimierte deBruijn-Graphen können mittels der BWT des Textes und einiger Zusatzinformationen implizit und recht platzsparend dargestellt werden [2]. An der entsprechenden Darstellung kann man erkennen, dass jeder Knoten mit $l > k$ Zeichen einen Block der Breite $l - k$ induziert, der in der BWT direkt getunnelt [1] werden kann.

Das Ziel der Arbeit ist es also, den komprimierten deBruijn-Graphen durch „Tunneling“ weiter zu komprimieren, aber dieselbe Funktionalität zu erhalten.

[1] Uwe Baier. On Undetected Redundancy in the Burrows-Wheeler Transform. `https://arxiv.org/abs/1804.01937`, 2018.

[2] Timo Beller and Enno Ohlebusch. A representation of a compressed de Bruijn graph for pan-genome analysis that enables search. *Algorithms for Molecular Biology*, 11(1), 2016.