Grundlagen des Burrows-Wheeler-Kompressionsalgorithmus

JÜRGEN ABEL •
Universität Duisburg-Essen

Der Burrows-Wheeler-Kompressionsalgorithmus hat als ein universelles Kompressionsverfahren in den letzten Jahren aufgrund seiner hervorragenden Kompressionsraten und hohen Verarbeitungsgeschwindigkeiten eine bemerkenswerte Beachtung erfahren. Das Herzstück des Algorithmus stellt hierbei die sogenannte Burrows-Wheeler-Transformation dar. Es handelt sich bei dieser Transformation um eine Permutation der Eingabedaten, welche Zeichen mit ähnlichem Kontext nahe beieinander anordnet. Die Transformation sowie die zugehörige Rücktransformation werden zusammen mit den weiteren Stufen des Algorithmus vorgestellt und die Ergebnisse mit denen anderer Datenkompressionsalgorithmen verglichen.

Schlüsselwörter: Burrows-Wheeler-Transformation, Blocksortierung, Datenkompression, BWT, BWCA, MTF, WFC, RLE, EC, GST

The Burrows-Wheeler Compression Algorithm as a universal compression scheme has received considerable attention over recent years for both outstanding compression rates and high speed. The Burrows-Wheeler transform represents the heart of such an algorithm. The transform is a permutation of the input sequence, which sorts symbols with similar contexts close together. The transform and the corresponding inverse transform will be presented together with further stages of the algorithm. The results of this algorithm will be compared to the results of other compression algorithms.

Key Words: Burrows-Wheeler transform, block sorting, compression, BWT, BWCA, MTF, WFC, RLE, EC, GST

CR Subject Classification:
E.4 [Coding and Information Theory] - Data compaction and compression;
H1.1 [Models and Principles]: Systems and Information Theory - Information theory; Value of information

1. EINFÜHRUNG

In der Informationstheorie unterscheidet man zwischen zwei Arten von Datenkompression. Bei der ersten Art handelt es sich um die sogenannte verlustlose Datenkompression. Die verlustlose Datenkompression wandelt eine Eingangsdatei in eine komprimierte Datei um, welche in der Regel kleiner als die Eingangsdatei ist. Aus der komprimierten Datei wird mit Hilfe der Dekompression - auch Datenexpansion genannt - eine dekomprimierte Datei erstellt. Die dekomprimierte Datei entspricht hierbei exakt der ursprünglichen Eingangsdatei. Bei der zweiten Art von Datenkompression handelt es sich um die sogenannte verlustbehaftete Datenkompression. Bei dieser Art wird ebenfalls eine Eingangsdatei in eine komprimierte Datei transformiert. Bei der Dekompression kann sich jedoch die dekomprimierte Datei von der ursprünglichen Eingangsdatei unterscheiden und mehr oder weniger große Abweichungen aufweisen. Der Vorteil der verlustbehafteten gegenüber der verlustlosen Datenkompression liegt in den deutlich besseren Kompressionsraten. Einsatz findet die verlustbehaftete Datenkompression vorwiegend bei digitalisierten Analogsignalen im Bereich von Audio- und Videobearbeitung. Zum einen haben digitalisierte Analogsignale aufgrund Ihrer Entstehung bereits einen Digitalisierungsfehler; zum anderen empfindet der Mensch zwei digitalisierte Audio- oder Videoaufnahmen als gleich, sofern deren Abweichungen klein genug sind. Die verlustfreie Datenkompression ist im Gegensatz hierzu universell einsetzbar, da die Datenintegrität erhalten bleibt, was in vielen Einsatzbereichen wichtig ist, wie beispielsweise im Finanzwesen oder im wissenschaftlichen Bereich.

Als Datenkompressionsrate bezeichnet man bei beiden Arten das Verhältnis der Größe der Ausgangsdatei zu der Größe der Eingangsdatei. Die beiden wichtigsten Ziele der Datenkompression sind hierbei die Minimierung der Datenkompressionsrate sowie die Maximierung der Verarbeitungsgeschwindigkeit bei der Kompression und Dekompression.

• Adresse: Jürgen Abel, Fachgebiet "Nachrichtentechnische Systeme", Fakultät Ingenieurwissenschaften, Universität Duisburg-Essen, Bismarckstraße 81, D-47057 Duisburg; Email: juergen.abel@acm.org

Bislang haben zumeist sequentielle Verfahren den Bereich der verlustlosen Datenkompression dominiert, unterteilt in referenzzierende Verfahren, basierend auf Wiederholungen, sowie statistische Verfahren, bestehend aus einem Kodierer und einem Datenmodell. Typische Vertreter der referenzzierenden Verfahren sind die Lauflängenkodierung sowie Lempel-Ziv-Methoden [16]. Bei statistischen Verfahren gibt ein Datenmodell die Wahrscheinlichkeiten für das jeweils nächste Zeichen an, und ein Kodierer erzeugt eine möglichst kurze Bitfolge aus diesen Wahrscheinlichkeiten. Beispiele für statistische Verfahren sind die Huffman-Kodierung und die arithmetische Kodierung [23].

Im Jahre 1994 wurde ein neues Verfahren, welches nicht sequentiell sondern auf ganzen Blöcken arbeitet, von David Wheeler und Michael Burrows eingeführt [7]. Es handelt sich um den sogenannten Burrows-Wheeler-Kompressionsalgorithmus. Das Herz dieses Algorithmus stellt hierbei die Burrows-Wheeler-Transformation dar, welche auch als Blocksortierung bezeichnet wird. Der Algorithmus wurde inzwischen von vielen Autoren aufgegriffen und verbessert. Fenwick hat in seinem BWT-Forschungsbericht aus dem Jahre 1996 [10] einige Verbesserungen beschrieben einschließlich der Sortierung von mehreren Zeichen gleichzeitig anstelle eines einzelnen. Zur Berechnung der Burrows-Wheeler-Transformation wurde im Jahre 1997 von Sadakane ein schnelles Verfahren mit Hilfe eines Suffixfeldes vorgestellt [19], welches im Jahre 1999 von Larsson erweitert worden ist [15]. Im Jahre 1998 präsentierte Kurtz für die Burrows-Wheeler-Transformation einige Implementierungen von Suffixbäumen, die weniger Speicher brauchen als bisherige Implementierungen und einen linearen Zeitaufwand besitzen [14]. Itoh und Tanka entwickelten 1999 ein neues Verfahren - das sogenannte "Two Stage Suffix Sort" - [12], welches von Kao im Rahmen einer Dissertation aufgegriffen und speziell für Folgen von sich wiederholenden Zeichen optimiert worden ist [13]. Auch die der Burrows-Wheeler-Transformation nachgeordneten Stufen sind Gegenstand zahlreicher Untersuchungen gewesen. Hierzu zählen insbesondere die Erweiterungen von Arnavut and Magliveras aus dem Jahre 1997 [2], von Balkenhol und Shtarkov aus dem Jahre 1999 [4], von Deorowicz aus dem Jahre 2000 [8] und 2002 [9] sowie von Abel aus dem Jahre 2003 [1].

Der vorliegende Beitrag beschreibt die Burrows-Wheeler-Transformation und ihre Rücktransformation zusammen mit den restlichen Stufen eines Burrows-Wheeler-Kompressionsalgorithmus. Zum Schluß erfolgt ein Vergleich von Kompressionsraten und Geschwindigkeiten der vorgestellten Implementierung mit denen anderer Datenkompressionsprogramme.

2. DER BURROWS-WHEELER-DATENKOMPRESSIONSALGORITHMUS

2.1 Der Aufbau des Algorithmus

Grundsätzlich besteht ein Burrows-Wheeler-Kompressionsalgorithmus aus drei Stufen, denen gegebenenfalls noch weitere Vorverarbeitungsstufen vorgeschaltet sein können. Bei den drei Stufen, welche in Abbildung 1 dargestellt sind, handelt es sich um einzelne Transformationen, die nacheinander durchlaufen werden. Hierbei werden die Daten in Form eines eindimensionalen Feldes - Block genannt - verarbeitet und der Block anschließend an die nächste Stufe weitergegeben. Die Blockgröße liegt im Bereich von einem oder mehreren Megabyte, wobei größere Blockgrößen in der Regel eine bessere Kompression nach sich ziehen, aber neben höherem Speicherbedarf auch eine höhere Verarbeitungsgeschwindigkeit zur Folge haben. Um die komprimierten Daten wieder zu dekomprimieren, werden die Stufen in rückwärtiger Reihenfolge durchlaufen, wobei die einzelnen Transformationen durch die entsprechenden Rücktransformationen ersetzt werden.

Die erste Stufe des Algorithmus ist die Burrows-Wheeler-Transformation (BWT). Die BWT ist eine Permutation der Eingabezeichen. Die Häufigkeiten der einzelnen Zeichen werden nicht geändert. Es findet durch die BWT daher keine Kompression statt, diese erfolgt erst in späteren Stufen. Aufgabe der BWT ist es, Zeichen mit ähnlichem Kontext nahe beieinander anzuordnen, wodurch die nachfolgenden Stufen die Daten besser verarbeiten können.

Die zweite Stufe des Algorithmus bildet die sogenannte Global-Structure-Transformation (GST) [1]. In dieser Stufe werden die lokalen Kontexte der BWT-Ausgabe in einen globalen Kontext überführt. Für die GST existieren eine ganze Reihe von unterschiedlichen Verfahren. Meistens wird dabei jedes Zeichen aus der BWT-Ausgabe in einen Index transformiert, wobei häufig vorkommende Zeichen in kleine Indices umgewandelt werden.

Die Entropiekodierung (EC) bildet die letzte Stufe des Algorithmus und hat die Aufgabe, die Folge der Indices der GST in eine möglichst kurze Bitfolge umzuwandeln. In dieser Stufe findet die eigentliche Kompression der Daten statt, die anderen Stufen hatten die Aufgabe, die Daten so vorzubereiten, daß sie möglichst gut komprimiert werden können.

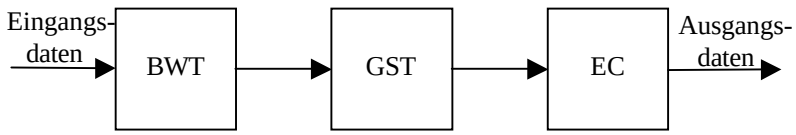

Abbildung 1: Schema des Burrows-Wheeler-Kompressionsalgorithmus

2.2 Der Aufbau der erweiterten Variante

In der nachfolgende Implementierung kommt eine Variante der GST zum Einsatz, bei welche die GST aus zwei Unterstufen besteht, in Abbildung 2 als gepunkteter Bereich zu erkennen. Zunächst wird eine Lauflängenkodierung (RLE) vorgenommen, bei welcher die Lauflängeninformationen direkt an die EC geleitet werden. Nach der RLE erfolgt eine sogenannte Weighted-Frequency-Count-Stufe (WFC) [9], in welcher die Indexfolge erstellt wird. Durch die Vorschaltung der RLE wird eine bessere Kompressionsrate und eine höhere Verarbeitungsgeschwindigkeit erzielt.

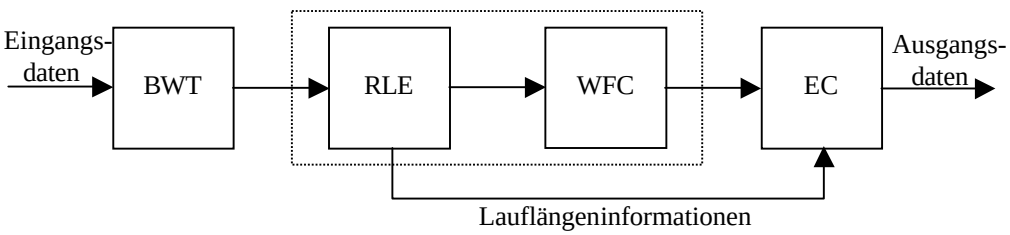

Abbildung 2: Schema der erweiterten Variante des Burrows-Wheeler-Kompressionsalgorithmus

Um das Schema dem Leser näherzubringen, ist in Abbildung 3 ein Beispiel aufgeführt, bei welchem die Ausgabedaten jeder Stufe für die Eingabedaten *ABRAKADABRAABRAKADABRAABRAKADABRA* (a) aufgeführt sind. Wie man sehen kann, befinden sich in den Ausgabedaten der BWT (b) eine Anzahl von Folgen gleicher Zeichen. Dies ist eine charakteristische Eigenschaft der BWT. Die Folgen gleicher Zeichen werden mit Hilfe der

Lauflängenkodierung der RLE auf eine kleinere Länge reduziert (c). Die Daten werden anschließend von der WFC in eine Indexfolge mit kleinen Indices umgewandelt (d). Zuletzt erfolgt die Umwandlung der Indexfolge in eine kurze Bitfolge mit Hilfe der Entropiekodierung (e).

```
(a) BWT input     : 41 42 52 41 4B 41 44 41 42 52 41 41 42 52 41 4B 41 44 41 42 52 41 41 42 52 41 4B 41 44 41 42 52 41
(b) BWT output    : 41 52 52 52 44 44 44 41 41 4B 4B 4B 52 52 52 41 41 41 41 41 41 41 41 41 41 41 42 42 42 42 42 42
(c) RLE output    : 41 52 52 52 44 44 44 41 41X4B 4B 4B 52 52 52 41 41 41 41 41 42 42 42 42
(d) WFC output    : 41 52 00 00 45 00 00 02 00 4C 00 00 03 00 00 02 00 00 00 00 45 00 00 00
(e) EC output     : 00 3F F8 E0 14 00 ED 00 0A 1B 4D 55 33 8E 5D C3 51 C1 51
```

Abbildung 3: Transformierte Daten nach jeder Stufe der erweiterten Variante
für die Eingabe *ABRAKADABRAABRAKADABRAABRAKADABRA* in hexadezimal

2.3 Die Burrows-Wheeler-Transformation (BWT)

Als erste Stufe des Kompressionsalgorithmus bildet die BWT das Kernstück des gesamten Algorithmus. Die Idee zu dieser Transformation stammte ursprünglich von Wheeler aus dem Jahre 1983, welcher sie jedoch zunächst nicht veröffentlichte [18]. Im Jahre 1994 wurde sie schließlich von Burrows und Wheeler in dem Forschungsbericht „A Blocksorting Lossless Data Compression Algorithm“ veröffentlicht [7]. Die Transformation stellt eine Umsortierung der Eingabedaten dar mit dem Ziel, Zeichen mit ähnlichem Kontext nahe beieinander anzuordnen. Hierbei werden die Daten nur in einer anderen Reihenfolge angeordnet und ansonsten nicht geändert. Diese Umsortierung führt dazu, daß sich nach der BWT lange Folgen von gleichen Zeichen bilden. Um die transformierten Daten wieder in die ursprüngliche Reihenfolge zu überführen, existiert eine Rücktransformation, welche die Daten wieder in exakt dieselbe Reihenfolge wie zuvor versetzt. Die Rücktransformation ist eine lineare Transformation und wesentlich schneller zu berechnen als die Hintransformation.

Die Funktionsweise der Hin- und Rücktransformation wird nun anhand der Zeichenkette *ABRAKADABRA* mit 11 Zeichen verdeutlicht. Wie in der Informatik üblich, beginnen Indizierungen mit dem Index 0.

Bei der Hintransformation wird die Zeichenkette so oft untereinander angeordnet wie sie Zeichen hat. Danach wird jede Zeile - angefangen mit der zweiten - gegenüber der vorigen um ein Zeichen nach rechts zyklisch rotiert. Die Zeilen werden hierbei als Ringpuffer angesehen, so daß Zeichen, die rechts hinauslaufen, links wieder eingeschoben werden. Das Ergebnis der Rotationen ist in Abbildung 4 ersichtlich.

Als nächstes werden die einzelnen Zeilen lexikographisch von oben nach unten aufsteigend sortiert, wie in Abbildung 5 dargestellt. Hierbei sind zwei Spalten von besonderer Bedeutung. Es handelt sich um die erste und die letzte Spalte. Die erste Spalte - *F*-Spalte (First) genannt - ist aufgrund der lexikographischen Zeilensortierung komplett alphabetisch sortiert, wodurch Zeichen mit ähnlichem Folgekontext direkt untereinander angeordnet sind. Die letzte Spalte - *L*-Spalte (Last) genannt - ist zwar nicht komplett sortiert, hat aber dennoch einige bemerkenswerte Eigenschaften. Aufgrund des Ringpuffers stellen die Zeichen der *L*-Spalte nämlich die Vorgängerzeichen der *F*-Spalte dar. Dadurch sind auch in der *L*-Spalte Zeichen mit ähnlichem Kontext nahe beieinander angeordnet, wodurch sich in der *L*-Spalte eine Vielzahl von Folgen gleicher Zeichen bilden. Gegenüber der ursprünglichen Zeichenkette läßt sich die *L*-Spalte somit wesentlich effizienter komprimieren.

Die *L*-Spalte sowie der Index *i*, der auf die Zeile mit der ursprünglichen Zeichenkette verweist, bilden die Ausgabe der BWT. Im vorliegenden Beispiel wurde die ursprüngliche Zeichenkette auf die Zeile mit dem Index $i = 2$ verschoben. Für das Beispiel *ABRAKADABRA* besteht die Ausgabe der BWT damit in der Zeichenkette *RDAKRAAAABB* sowie der Zahl 2.

Die Ausgabe der BWT für die Zeichenkette *ABRAKADABRAABRAKADABRAABRAKADABRA* ist in Abbildung 3 (b) aufgeführt.

Index	**F-Spalte**										**L-Spalte**
0	*A*	*B*	*R*	*A*	*K*	*A*	*D*	*A*	*B*	*R*	*A*
1	*A*	*A*	*B*	*R*	*A*	*K*	*A*	*D*	*A*	*B*	*R*
2	*R*	*A*	*A*	*B*	*R*	*A*	*K*	*A*	*D*	*A*	*B*
3	*B*	*R*	*A*	*A*	*B*	*R*	*A*	*K*	*A*	*D*	*A*
4	*A*	*B*	*R*	*A*	*A*	*B*	*R*	*A*	*K*	*A*	*D*
5	*D*	*A*	*B*	*R*	*A*	*A*	*B*	*R*	*A*	*K*	*A*
6	*A*	*D*	*A*	*B*	*R*	*A*	*A*	*B*	*R*	*A*	*K*
7	*K*	*A*	*D*	*A*	*B*	*R*	*A*	*A*	*B*	*R*	*A*
8	*A*	*K*	*A*	*D*	*A*	*B*	*R*	*A*	*A*	*B*	*R*
9	*R*	*A*	*K*	*A*	*D*	*A*	*B*	*R*	*A*	*A*	*B*
10	*B*	*R*	*A*	*K*	*A*	*D*	*A*	*B*	*R*	*A*	*A*

Abbildung 4: Rotation der Zeilen

Index	**F-Spalte**										**L-Spalte**
0	*A*	*A*	*B*	*R*	*A*	*K*	*A*	*D*	*A*	*B*	*R*
1	*A*	*B*	*R*	*A*	*A*	*B*	*R*	*A*	*K*	*A*	*D*
2	*A*	*B*	*R*	*A*	*K*	*A*	*D*	*A*	*B*	*R*	*A*
3	*A*	*D*	*A*	*B*	*R*	*A*	*A*	*B*	*R*	*A*	*K*
4	*A*	*K*	*A*	*D*	*A*	*B*	*R*	*A*	*A*	*B*	*R*
5	*B*	*R*	*A*	*A*	*B*	*R*	*A*	*K*	*A*	*D*	*A*
6	*B*	*R*	*A*	*K*	*A*	*D*	*A*	*B*	*R*	*A*	*A*
7	*D*	*A*	*B*	*R*	*A*	*A*	*B*	*R*	*A*	*K*	*A*
8	*K*	*A*	*D*	*A*	*B*	*R*	*A*	*A*	*B*	*R*	*A*
9	*R*	*A*	*A*	*B*	*R*	*A*	*K*	*A*	*D*	*A*	*B*
10	*R*	*A*	*K*	*A*	*D*	*A*	*B*	*R*	*A*	*A*	*B*

Abbildung 5: Sortierung der Zeilen

2.4 Die Rücktransformation der Burrows-Wheeler-Transformation

Da zu einer normalen lexikographischen Sortierung keine Rücktransformation existiert, stellt sich für einen Außenstehenden die Frage, wie man die ursprünglichen Daten rekonstruieren kann. Eine der erstaunlichsten Eigenschaften der BWT sind die Tatsachen, daß man aus der *L*-Spalte und dem Index *i* nicht nur die ursprünglichen Daten - wie Phoenix aus der Asche - vollständig rekonstruieren kann, sondern daß der Aufwand für die Rekonstruktion der Ursprungsdaten sogar noch wesentlich geringer ist als der Aufwand für die Sortierung der Hintransformation.

Da es sich bei der BWT lediglich um eine Umsortierung handelt, befinden sich in der *L*-Spalte die gleichen Zeichen wie in der *F*-Spalte und der ursprünglichen Zeile. Die *F*-Spalte kann somit durch eine einfache Sortierung der Zeichen der *L*-Spalte wiederhergestellt werden. Die *L*-Spalte und die *F*-Spalte werden danach nebeneinander angeordnet und bilden eine einfache Tabelle, mit deren Hilfe die ursprünglichen Daten wiederhergestellt werden können. Hierzu durchschreitet man die Tabelle ähnlich einer verketteten Liste, beginnend mit dem Index *i*. Jedesmal, wenn man eine neue Zeile der Tabelle erreicht, wird dabei das Zeichen ausgegeben, welches sich in der *F*-Spalte befindet. Danach geht man zu demjenigen Buchstaben innerhalb der L-Spalte, welches innerhalb desselben Buchstaben des letzten ausgegebenen Zeichens an der gleichen Stelle in der *L*-Spalte steht.

Das Verfahren soll nun anhand des vorigen Beispieles mit *RDAKRAAAABB* als *L*-Spalte sowie Index 2 verdeutlicht werden. Zunächst sortiert man die Zeichen der *L*-Spalte alphabetisch und erhält damit die *F*-Spalte wie in Abbildung 6 angezeigt. Der Index *i* gibt die Position der ursprünglichen Zeile an, mit der man beginnt. Das erste Zeichen, welches ausgegeben wird, ist das Zeichen in der F-Spalte der aktuellen Zeile, hier also das Zeichen *A*.

	Zeile	L-Spalte	F-Spalte
	0	*R*	*A*
	1	*D*	*A*
Index *i* →	**2**	*A*	***A***
	3	*K*	*A*
	4	*R*	*A*
	5	*A*	*B*
	6	*A*	*B*
	7	*A*	*D*
	8	*A*	*K*
	9	*B*	*R*
	10	*B*	*R*

Abbildung 6: Sortierung der *F*-Spalte und Wiederherstellen des ersten Buchstabens

Dieses *A* ist das dritte *A* in der *F*-Spalte. Nun geht man zum dritten *A* der *L*-Spalte, wie in Abbildung 7 angedeutet. Dies befindet sich in Zeile 6 und man gibt das dazugehörige Zeichen *B* der *F*-Spalte aus.

Zeile	L-Spalte	F-Spalte
0	*R*	*A*
1	*D*	*A*
2	*A*	*A*
3	*K*	*A*
4	*R*	*A*
5	*A*	*B*
6	*A*	***B***
7	*A*	*D*
8	*A*	*K*
9	*B*	*R*
10	*B*	*R*

Abbildung 7: Ermittlung des zweiten Buchstaben

Das ausgegebene *B* ist das zweite *B* in der *F*-Spalte. Als nächstes geht man zum zweiten *B* der *L*-Spalte und gibt das dazugehörige *R* aus der *F*-Spalte aus. Die Ermittlung der restlichen Zeichen erfolgt nach dem gleichen Schema und ist in Abbildung 8 wiedergegeben. Die Ausgabe endet, wenn man auf die Zeile stößt, mit welcher man begonnen hat. Alternativ könnte man auch die ausgegebenen Zeichen zählen und das Verfahren dann beenden, wenn so viele Zeichen ausgegeben worden sind, wie die Länge der *L*-Spalte angibt.

Zeile	L-Spalte	F-Spalte
0	*R*	*A*
1	*D*	*A*
2	*A*	*A*
3	*K*	*A*
4	*R*	*A*
5	*A*	*B*
6	*A*	*B*
7	*A*	*D*
8	*A*	*K*
9	*B*	*R*
10	*B*	*R*

Abbildung 8: Ermittlung der restlichen Buchstaben

Das gesamte Verfahren der Rücktransformation ist in Abbildung 9 nocheinmal in Kurzform aufgeführt.

1. Wiederherstellen der *F*-Spalte durch Sortierung der *L*-Spalte
2. Start mit der Zeile, auf welche der Index *i* verweist
3. Ausgabe des Zeichens in der *F*-Spalte der aktuellen Zeile
4. Fortfahren mit dem Zeichen der *L*-Spalte, welches innerhalb desselben Buchstaben des letzten ausgegebenen Zeichens an der gleichen Stelle in der *L*-Spalte steht
5. Wiederhohlen der Punkte 3 und 4 solange, bis die aktuelle Zeile gleich dem Index *i* ist

Abbildung 9: Schema der Rücktransformation

2.5 Die Lauflängenkodierung (RLE)

Lauflängenkodierungen gehören zu den ältesten und einfachsten Kompressionstechniken. Die Grundidee besteht in einer sequentiellen Durchsuchung der Eingabedaten nach Folgen gleicher Zeichen und dem Ersetzen dieser Folgen durch eine kürzere Folge und eine Lauflängeninformation.

Da die BWT aufgrund Ihrer Sortierung eine große Anzahl von Folgen gleicher Zeichen erzeugt, kann eine RLE im Anschluß an die BWT sinnvoll eingesetzt werden. In dem hier vorgestellten Kompressionsschema dient die RLE als Unterstützung der folgenden WFC, da sie zwei Vorteile bietet. Zum einen führt die Verkürzung der Folgen zu einer besser zu komprimierenden Indexfolge innerhalb der WFC. Zum anderen ist eine RLE wesentlich schneller als eine WFC, und trägt damit insgesamt zu einer höheren Verarbeitungsgeschwindigkeit bei [1].

Für die RLE existieren viele unterschiedliche Techniken, welche sich in der Länge der zu ersetzenden Folgen sowie in der Art und Weise unterscheiden, wie die Lauflängen kodiert werden. Im vorliegenden Fall wurde eine Technik gewählt, welche eine Folge gleicher Zeichen, die eine Länge von mehr als einem Zeichen besitzt, durch eine Folge ersetzt, deren Länge dem binären Logarithmus der ursprünglichen Länge plus Eins entspricht. Die restlichen Lauflängeninformationen werden nicht - wie bei den meisten RLE-Techniken - als Zahl hinter die verkürzte Folge geschrieben. Statt dessen werden sie als Bitfolge in einen separaten Datenstrom umgeleitet, der direkt zur EC geführt wird und in Abbildung 2 ersichtlich ist. Dadurch behindern die Lauflängeninformationen nicht die Indexbildung der WFC.

In früheren Burrows-Wheeler-Kompressionsalgorithmen wurde die RLE nicht direkt nach der BWT sondern erst vor der EC eingesetzt. Hier wurde eine Technik benutzt, die speziell auf Folgen von Nullen (RLE0) abgestimmt war. Dieses Verfahren geht auf Wheeler zurück und wurde von Fenwick in seinem BWT-Forschungsbericht beschrieben [10]. Der Einsatz direkt hinter der BWT geht auf eine Idee von Gringeler zurück und wurde von Abel im Jahre 2003 erstmals erwähnt [1]. Es stellt aus den oben genannten Gründen die bessere Alternative dar.

Für die BWT-Eingangsdaten *ABRAKADABRAABRAKADABRAABRAKADABRA* ist in Abbildung 3 (c) die Ausgabe der RLE aufgeführt.

2.6 Die Weighted-Frequency-Count-Stufe (WFC)

Die WFC stellt eine verbesserte Version einer Move-To-Front-Stufe (MTF) dar und wurde von Deorowicz im Jahre 2002 beschrieben [9] und von Abel im Jahre 2003 noch einmal verbessert [1].

Die ursprüngliche MTF verwaltet eine Liste von 256 Zahlen im Bereich von 0 bis 255. Die Eingabedaten werden hierbei sequentiell abgearbeitet. Für jedes Eingabezeichen wird der Index des Eingabezeichens inner-

halb der Liste ausgegeben, danach das Eingabezeichen in der Liste auf den ersten Eintrag der Liste verschoben und dazwischen liegende Zeichen um eins aufgeschoben. Dadurch erhalten häufig vorkommende Zeichen kleinere Indices, und Zeichen, welche seltener vorkommen, erhalten größere Indices zugewiesen. Da sich durch die BWT viele ähnliche Zeichen nahe beieinander befinden, entsteht als Ausgabe der MTF eine Indexfolge mit relativ kleinen Indices.

Die WFC unterhält ebenfalls eine Liste der Zeichen von 0 bis 255 und gibt für jedes Eingabezeichen einen Index innerhalb der Liste aus. Sie unterscheidet sich von der MTF dadurch, daß sie das aktuelle Zeichen nicht direkt auf den Index 0 der Liste verschiebt. Vielmehr führt die WFC eine Statistik darüber, welche Zeichen in der jüngsten Vergangenheit wie oft aufgetreten sind. Anhand dieser Statistik wird die Liste nach jedem Eingabezeichen komplett neu berechnet, wobei häufig vorkommende Zeichen weiter vorne eingeordnet werden [9]. Dadurch ist die WFC zwar wesentlich langsamer als die MTF, aber als Ergebnis erhält man eine Indexfolge mit kleineren Indices, was insgesamt zu einer besseren Kompression führt.

Abbildung 3 (d) enthält die Ausgabedaten der WFC für die BWT-Eingangsdaten *ABRAKADABRA-ABRAKADABRAABRAKADABRA.*

2.7 Die Entropiekodierung (EC)

Die Ausgabedaten der WFC sind - abgesehen von der Lauflängenkodierung der RLE - noch nicht komprimiert. Die Aufgabe der eigentlichen Komprimierung erfüllt die EC. Die EC überführt die Indexfolgen der WFC in eine möglichst kurze Bitfolge. Je nach Implementierung können hierzu sowohl Huffmankodierungen [7][20] als auch arithmetische Kodierungen [3][8][10] eingesetzt werden. Huffmankodierungen bieten den Vorteil einer höheren Verarbeitungsgeschwindigkeit, haben jedoch eine deutlich schlechtere Kompressionsrate. In dem vorliegenden Kompressionsschema wird daher der arithmetischen Kodierung mit einem adaptiven Modell der Vorzug gegeben. Die arithmetische Kodierung stellt die Eingabedaten als eine einzige rationale Zahl dar, dargestellt aus einer Folge von binären Nachkommastellen. Zur Erzeugung der Nachkommastellen dienen die Wahrscheinlichkeiten des adaptiven Modelles. Da das Thema arithmetische Kodierung sehr umfangreich ist, sei der Leser an dieser Stelle auf das Standardwerk „Arithmetic Coding for Data Compression" von Witten, Neal und Cleary [23] sowie auf den Artikel „Arithmetische Kodierung" von Bodden, Clasen und Kneis [6] verwiesen.

Abbildung 3 (e) enthält die endgültigen Ausgabedaten der EC für die BWT-Eingangsdaten *ABRAKADABRA-ABRAKADABRAABRAKADABRA.* In diesen Daten sind die Ausgabedaten der WFC und die Lauflängeninformationen der RLE enthalten.

3. ERGEBNISSE

3.1 Die Kompressionsrate

Der Vergleich der Kompressionsraten zwischen dem vorgestellten Algorithmus und anderen Datenkompressionsalgorithmen wird mit Hilfe der 14 Dateien des Standard-Calgary-Corpus [5] durchgeführt. Die Kompressionsrate ist hierbei definiert als das Verhältnis der Größe der Ausgangsdatei zu der Größe der Eingangsdatei, gemessen in Bits der Ausgabedatei zu Bytes der Eingabedatei (bps). Kleinere Kompressionsraten bedeuten eine bessere Kompression. Wie im Rahmen der Informationstheorie üblich, zieht man für den Vergleich den ungewichteten Durchschnitt der Kompressionsraten heran, der in der untersten Zeile aufgeführt ist. Der ungewichtete Durchschnitt ist definiert als die Summe der einzelnen Kompressionsraten für jede Datei divi-

diert durch die Anzahl der Dateien. Dadurch werden größere Dateien nicht stärker gewichtet als kleinere Dateien.

Für den Vergleich wurden die folgende Implementierungen herangezogen und sind mit ihren jeweiligen Kompressionsraten in Abbildung 10 aufgeführt:

- GZIP93-V1.2.4 mit Option -9, von Jean-loup Gailly und Mark Adler 1993, basierend auf LZ77, [11]
- PPMC90, von Moffat 1990, basierend auf PPM, [17]
- cPPMII64-02, von Dmitry Shkarin 2002, basierend auf PPM, [21]
- CTW95, von Frans Willems, Yuri Shtarkov und Tjalling Tjalkens 1995, basierend auf CTW, [22]
- BW94, von Michael Burrows und David Wheeler 1994, die ursprüngliche BWT Implementierung, [7]
- BZIP2-01, von Julian Seward 2001, basierend auf BWT, [20]
- FUE03, von Jürgen Abel 2003, die hier vorgestellte Implementierung, basierend auf BWT

Mit einem Wert von 2,25 bps erreicht FUE03 eine Kompressionsrate, welche mehr als 16% besser als diejenige von GZIP93 ist. Damit liegt FUE03 im oberen Bereich der Datenkompressionsprogramme und braucht den Vergleich mit PPM-Implementierungen, die in der Regel wesentlich mehr Speicherbedarf und Rechenzeit benötigen, nicht zu scheuen.

Datei	**Größe**	**GZIP93**	**PPMC 90**	**cPPMII 64-02**	**CTW95**	**BW94**	**BZIP2-01**	**FUE03**
bib	111.261	2,52	2,12	1,68	1,79	2,07	1,98	1,90
book1	768.771	3,26	2,54	2,14	2,19	2,49	2,42	2,28
book2	610.856	2,70	2,25	1,78	1,87	2,13	2,06	1,96
geo	102.400	5,36	4,91	4,16	4,46	4,45	4,45	4,18
news	377.109	3,07	2,68	2,14	2,29	2,59	2,52	2,40
obj1	21.504	3,84	3,72	3,50	3,68	3,98	4,01	3,74
obj2	246.814	2,63	2,52	2,11	2,31	2,64	2,48	2,38
paper1	53.161	2,79	2,48	2,14	2,25	2,55	2,49	2,41
paper2	82.199	2,88	2,45	2,12	2,21	2,51	2,44	2,35
pic	513.216	0,82	0,99	0,70	0,79	0,83	0,78	0,71
progc	39.611	2,68	2,48	2,16	2,29	2,58	2,53	2,44
progl	71.646	1,81	1,84	1,39	1,56	1,80	1,74	1,67
progp	49.379	1,81	1,80	1,39	1,60	1,79	1,73	1,68
Trans	93.695	1,61	1,72	1,17	1,34	1,57	1,53	1,45
Durchschnitt		2,70	2,46	2,04	2,19	2,43	2,37	2,25

Abbildung 10: Kompressionsraten unterschiedlicher Datenkompressionsalgorithmen in bps

3.2 Die Geschwindigkeit der Kompression und der Dekompression

Der Vergleich der Geschwindigkeiten für die Kompression und die Dekompression erfolgt wieder am Beispiel der 14 Dateien des Standard-Calgary-Corpus. Da die tatsächliche Geschwindigkeit von der Beschaffenheit

des Rechners abhängt, wird als Maßstab das Standardkompressionsprogramm GZIP93 mit Option –9 herangezogen, welches zu den schnellsten Kompressionsprogrammen zählt und weit verbreitet ist [11]. Dadurch ist ein indirekter Rückschluß auch auf andere Rechnersysteme möglich. Die Messungen wurden auf einem WINDOWS 2000 PC mit einem 700 MHz Pentium III Prozessor durchgeführt. Die Zeiten für die Kompression und die Dekompression umfassen die Summe der Zeiten für alle 14 Dateien einschließlich des Ladens der Eingabedaten und des Schreibens der Ausgabedaten und haben die Einheit Sekunden.

Wie man Abbildung 11 entnehmen kann, ist das Programm GZIP93 gegenüber FUE93 etwa dreimal so schnell bei der Kompression und etwa neunmal so schnell bei der Dekompression. Der Unterschied kommt insbesondere durch die WFC und EC zustande, welche von Natur aus relativ langsam sind. Weiterhin benötigen WFC und EC bei der Rücktransformation der Daten etwa genausoviel Zeit wie bei der Hintransformation, so daß sich die Zeiten von Kompression und Dekompression bei FUE03 nicht wesentlich unterscheiden. GZIP93 hat jedoch eine wesentlich höhere Dekompressions- als Kompressionsgeschwindigkeit, was zur Folge hat, daß der Unterschied zwischen den beiden Programmen bei der Dekompression mit dem Faktor 9 deutlicher zu Buche schlägt als bei der Kompression mit dem Faktor 3.

Vorgang	GZIP93	FUE03
Kompression	2,60	8,08
Dekompression	0,81	7,27

Abbildung 11: Gesamtausführungszeiten in Sekunden für die 14 Dateien des Calgary-Corpus

4. ZUSAMMENFASSUNG

Die universelle verlustlose Datenkompression hat durch die Burrows-Wheeler-Transformation einen neuen hochinteressanten Ansatz gefunden. Kompressionsalgorithmen auf Grundlage der Burrows-Wheeler-Transformation erzielen hervorragende Kompressionsraten und besitzen hohe Verarbeitungsgeschwindigkeiten. Gegenüber der ursprünglichen Implementation von Burrows und Wheeler aus dem Jahre 1994 wurden inzwischen eine ganze Reihe von Verbesserungsmöglichkeiten vorgestellt.

Eine dieser verbesserten Versionen hat der vorliegende Artikel präsentiert. Zunächst erfolgt eine Umsortierung der Daten mit Hilfe der BWT. Dadurch werden Zeichen mit ähnlichem Kontext nahe beieinander angeordnet. Danach durchlaufen die Daten eine Lauflängenkodierung, welche lange Folgen gleicher Zeichen reduziert. Im Anschluß wird der lokale Kontext der Zeichen durch die WFC in eine Indexfolge mit globalem Kontext umgeformt. Zum Schluß werden die Zeichen durch eine arithmetische Kodierung endgültig in eine kurze Bitfolge komprimiert. Mit einem Wert von 2,25 bps stellt das vorgestellte Kompressionsschema eines der besten Burrows-Wheeler-Kompressionsalgorithmen dar und liegt im Bereich der stärksten PPM- und CTW-Implementierungen.

LITERATURVERZEICHNIS

[1] Abel, J. "Improvements to the Burrows-Wheeler Compression Algorithm: After BWT Stages"; Vorabdruck, eingereicht an das ACM im März 2003, http://www.data-compression.info/JuergenAbel/Preprints/Preprint_After_BWT_Stages.pdf; 2003.

[2] Arnavut, Z., Magliveras, S.S.; "Block sorting and compression"; Proceedings of the IEEE Data Compression Conference; 1997.

[3] Balkenhol, B., Kurtz, S.; "Universal Data Compression Based on the Burrows-Wheeler Transformation: Theory and Practice"; IEEE Transactions on Computers; 49(10); 2000.

[4] Balkenhol, B., Shtarkov, Y.M.; "One attempt of a compression algorithm using the BWT"; Discrete Structures in Mathematics, Mathematische Fakultät, Universität Bielefeld; 1999.

[5] Bell, T.C., Witten, I.H., Cleary, J.G.; Calgary Corpus: "Modeling for text compression"; Computing Surveys 21(4): 557-591; 1989.

[6] Bodden, E., Clasen, M., Kneis J.; "Arithmetische Kodierung"; Lehrstuhl für Informatik IV der RWTH Aachen, 2001.

[7] Burrows, M., Wheeler, D.J.; "A block-sorting lossless data compression algorithm"; Technical report, Digital Equipment Corporation, Palo Alto, California; 1994.

[8] Deorowicz, S.; "Improvements to Burrows-Wheeler compression algorithm"; Software Practice and Experience, 30(13), 1465-1483; 2000.

[9] Deorowicz, S.; "Second step algorithms in the Burrows-Wheeler compression algorithm", Software Practice and Experience, 32(4), 99-111, 2002.

[10] Fenwick, P.M.; "Block-Sorting Text Compression - Final Report"; The University of Auckland, New Zealand, Technical Report 130; 1996.

[11] Gailly, J.L., Adler, M.; "The GZIP program", Version 1.2.4; 1993.

[12] Itoh, H., Tanaka, H.; "An Efficient Method for Construction of Suffix Arrays"; IPSJ Transactions on Databases, Abstract Vol.41, No. SIG01 - 004; 1999.

[13] Kao, T.-H; "Improving Suffix-Array Construction Algorithms with Applications". Master's thesis, Department of Computer Science, Gunma University, Japan; 2001.

[14] Kurtz, S.; "Reducing the Space Requirement of Suffix Trees"; Report 98-03, Technische Fakultät, Universität Bielefeld; 1998.

[15] Larsson, N.J. "Structures of String Matching and Data Compression". PhD thesis, Department of Computer Science, Lund University, Schweden; 1999.

[16] Lempel, A., Ziv, J.; "A Universal Algorithm for Sequential Data Compression"; IEEE Transactions on Information Theory, 23(3):337-343, 1977.

[17] Moffat, A.; "Implementing the PPM data compression scheme"; IEEE Transactions on Communications, 38,1917-1921; 1990.

[18] Nelson, M.; "Data compression with the Burrows-Wheeler transform"; Dr. Dobbs' Journal; September 1996.

[19] Sadakane, K.; "Improvements of Speed and Performance of Data Compression Based on Dictionary and Context Similarity"; Master's thesis, Department of Information Science, Faculty of Science, University of Tokyo, Japan; 1997.

[20] Seward, J.; "The BZIP2 program 1.0.2"; http://sources.redhat.com/bzip2; 2001.

[21] Shkarin, D; "PPM: One step to practicality"; Proceedings of the IEEE Data Compression Conference 2002, Snowbird, Utah, Storer, J.A. and Cohn, M. Eds. 202-211; 2002.

[22] Willems, F.M.J., Shtarkov, Y.M., Tjalkens, T.J.; "The Context Tree Weighting Method: Basic Properties"; IEEE Transactions on Information Theory, Vol. IT-41 (3), 653-664; 1995.

[23] Witten, I., Neal, R., Cleary, J.; "Arithmetic Coding for Data Compression", Communication of the ACM, 30(6) 520-540; 1987.