Byte-Kodierung und verlustfreie Kompression von Fahrzeugbewegungsdaten

Bachelorarbeit

von

Andreas Disterhöft

aus

Duschanbe, Tadschikistan

vorgelegt am

Lehrstuhl für Rechnernetze und Kommunikationssysteme

Prof. Dr. Martin Mauve

Heinrich-Heine-Universität Düsseldorf

Januar 2012

Betreuer:

Markus Koegel M. Sc.

Zusammenfassung

In dieser Arbeit wird der Einsatz von herkömmlichen verlustfreien Kompressionsverfahren auf Fahrzeugbewegungsdaten, sogenannte GPS-Traces, untersucht. Hierzu wird zunächst ein verlustfreier Bytekodierer auf Basis von Differenzvektoren implementiert. Die Laufzeit des Bytekodierers ist $O(n)$, wobei n die Anzahl an GPS-Punkten im GPS-Trace ist.

Der Bytekodierer bietet zwei Varianten der Speicherung von GPS-Punkten an. Die apart-Variante speichert den Vor- und Nachkommateil der Dezimalgrade getrennt und die united-Variante verschiebt lediglich das Komma der Dezimalgrade, so dass eine ganze Zahl gespeichert werden kann. Bei jeder Speicherungsvariante spiegeln drei Profile unterschiedliche Strategien zur Generierung von Differenzvektoren erster bzw. zweiter Ordnung wider. Hierbei sind Differenzvektoren erster bzw. zweiter Ordnung im wesentlichen Geschwindigkeits- bzw. Beschleunigungsvektoren.

Abschließend wird eine Auswahl an verlustfreien Kompressionsprogrammen auf den erzeugten Bytestrom angewandt und deren Kompressionsrate ausgewertet. Hierbei erreicht die von Bob Carpenter entwickelte arithmetische Kodierung mit dem PPM(16)-Modell [Cap02], angewandt auf das analytische Profil mitsamt der *united*-Speichervariante, die beste Kompressionsrate. Vergleicht man die erzielte Kompressionsrate mit einem vorhandenen verlustbehafteten Kompressionsverfahren, wie in [KM11] vorgestellt, so werden in etwa die gleichen Kompressionsraten bei einen maximalen Fehler von fünf cm erreicht.

Danksagung

Zunächst möchte ich mich bei Markus Koegel für seine freundliche Betreuung bedanken. Während meiner Arbeit gab er mir wertvolle Tipps für die Implementierung des Kodierers und Ausarbeitung der Arbeit.

Weiterhin möchte ich Dominik Weinhold und Marc Ewert für deren Hilfe bei der Korrektur danken. Sie nahmen sich viel Zeit und versorgten mich mit Verbesserungsvorschlägen und Korrekturhilfen. Viel Zeit nahm sich auch meine Freundin Sandra Golor, die mir in den letzten Tagen wertvolle Hinweise gegeben und mich während der Arbeit unterstützt hat.

Abschließend bedanke ich mich ganz herzlich bei meinen Eltern für die Unterstützung während meiner gesamten Studienzeit.

Inhaltsverzeichnis

Abbildungsverzeichnis **ix**

Tabellenverzeichnis **xi**

1 Einleitung **1**

1.1 Problemstellung 1
1.2 Aufgabenstellung 2
1.3 Aufbau der Arbeit 2

2 Verwandte Arbeiten **3**

3 Grundlagen **5**

3.1 verlustfreie Kompressionsverfahren 5
3.1.1 Huffman-Kodierung 5
3.1.2 LZ-basierte Kompressionsalgorithmen und -programme 7
3.1.3 BZip2 12
3.1.4 Arithmetische Kodierung 13

4 Bytekodierer **15**

4.1 Realisierungsarten 16
4.2 Kodierung des Bytestroms
(*apart*-Variante) 19
4.2.1 Problematik 20
4.2.2 Profil 1 22
4.2.3 Profil 2 23
4.2.4 Profil 3 24

4.3 Kodierung des Bytestroms
(*united*-Variante) 26
4.3.1 Profil 1 27
4.3.2 Profil 2 28
4.3.3 Profil 3 28

5 Ergebnisanalyse **31**
5.1 Datenbasis und Methodik 31
5.2 Auswertung 32
5.2.1 p-Tupel des dritten Profils 33
5.2.2 Paradigmenvergleich 36
5.2.3 Profilvergleich 38
5.2.4 Nachkommastellenvergleich 40

6 Fazit **43**
6.1 Ergebnis 43
6.2 Ausblick 44

Anhang **44**

A **45**
A.1 p-Wert Vergleich 45
A.2 Profilvergleich 45

Literaturverzeichnis **49**

Abbildungsverzeichnis

3.1 LZ-Algorithmenfamilie und Packprogramme, die diese nutzen. 6

4.1 Schema des Headers für die Bytekodierung. 16
4.2 Genereller Aufbau des erzeugten Bytestroms. 16
4.3 Ablaufdiagramm des Algorithmus. 17
4.4 Kumulative Verteilungungsfunktionen. 24

5.1 apart-Paradigma: p-wert für Differenzvektoren erster Ordnung: 1.0 . . . 32
5.2 apart-Paradigma: p-wert für Differenzvektoren erster Ordnung: 1.0 . . . 33
5.3 apart-Paradigma, Profil 3 (1.0;0.8): Verteilung der Kodierungsstufen. . . 34
5.4 united-Paradigma: p-wert für Differenzvektoren erster Ordnung: 1.0 . . 35
5.5 Paradigmenvergleich: Profil 1 36
5.6 Paradigmenvergleich: Profil 2 37
5.7 Verteilung der Kodierungsstufen des zweiten Profils. 38
5.8 Paradigmenvergleich: Profil 3, p-Tupel (1.0 ; 0.8) 39
5.9 Profilvergleich: united-Paradigma 40
5.10 Nachkommastellenvergleich: united-Paradigma Profil 3, p = (1.0 ; 0.8). . 41

A.1 p-Wert Vergleich: apart-Paradigma 46
A.2 p-Wert Vergleich: united-Paradigma 47
A.3 Profilvergleich: apart-Paradigma 48

Tabellenverzeichnis

3.1 LZ77 Beispiel: 8 Byte Suchpuffer, 4 Byte Vorschaupuffer. Zu kodierender String: *Mississippi*. 7
3.2 LZ78: Beispiel: Zu kodierender String: *Mississippi*. 8
3.3 LZSS Beispiel:Zu kodierender String: *Mississippi*. 10
3.4 LZW Beispiel: Zu kodierender String: *Mississippi*. 11

4.1 Speicherplatzbreiten der apart-Profile. 22
4.2 Speicherplatzbreiten der united-Profile. 27

5.1 Verteilung der Anzahl an Nachkommastellen. 40

Kapitel 1

Einleitung

1.1 Problemstellung

Die großen Ziele der Fahrzeug-zu-Fahrzeug-Kommunikation sind es, die Sicherheit und Effizienz des Straßenverkehrs zu steigern. Aus diesem Grund speichern Fahrzeuge gefahrene Routen in sogenannten Bewegungstraces, welche im Wesentlichen aus GPS-Punktfolgen bestehen, und verbreiten diese. Auf Basis der gesammelten Daten werden beispielsweise Kartenverbesserungssysteme für Baustellenbereiche [KP08] implementiert. Da Fahrzeuge ihre gesammelten Daten schließlich drahtlos übertragen und die verwendeten Kanalkapazitäten beschränkt sind, ist eine geschickte Kodierung und Kompression der Daten unabdingbar. Somit werden Sicherheitsanwendungen, wie einem elektronischen Bremslicht, so viele Ressourcen wie möglich überlassen. Bisherige Kompressionen der Fahrzeugbewegungsdaten beruhen auf verlustbehafteten Verfahren. Obwohl herkömmliche verlustfreie Kompressionsverfahren ausgereift sind, wurde diese Art der Kompression bisher nicht betrachtet.

1.2 Aufgabenstellung

Das Ziel dieser Arbeit ist es, einen Binärkodierer zu entwickeln, dessen Ausgabe sich gut mit herkömmlichen Kompressionsverfahren verlustfrei komprimieren lässt. Hierzu gilt es zunächst zu untersuchen, wie GPS-Traces kodiert werden sollen, um eine gut komprimierbare Ausgabe zu erhalten. Des Weiteren muss eine Auswahl an herkömmlichen Kompressionsverfahren getroffen und diskutiert werden, die für eine Kompression von Fahrzeugbewegungsdaten geeignet sind. Unter Umständen muss die Ausgabe des Binärkodierers optimiert werden, um eine bessere Kompressionsrate durch die Auswahl an Kompressionsverfahren zu erreichen. Schließlich sollen die ausgewählten Kompressionsverfahren auf die binär kodierten GPS-Traces angewandt und die Ergebnisse ausgewertet werden.

1.3 Aufbau der Arbeit

Diese Arbeit beginnt mit verwandten Arbeiten in Kapitel 2. Hier werden ähnliche Arbeiten vorgestellt und wesentliche Unterschiede zu dieser Arbeit geschildert. In Kapitel 3 befasst sich die Arbeit mit den Grundlagen herkömmlicher Kompressionsverfahren und deren Implementierungen. Aufbauend auf Kapitel 3 wird in Kapitel 4 der Bytekodierer vorgestellt, diskutiert und dessen Ausgabe für die vorgestellten Kompressionsverfahren optimiert. Die Analyse des Bytekodierers und der Kompression der Ausgabe findet in Kapitel 5 statt. Hier wird zunächst die Datenbasis vorgestellt, die Methodik erklärt und schließlich die Ergebnisse nach ausgewählten Kriterien ausgewertet. Das Fazit der Arbeit und ein Ausblick auf zukünftige Entwicklungen befinden sich in Kapitel 6.

Kapitel 2

Verwandte Arbeiten

Bisherige Arbeiten, die sich mit der Kompression von Fahrzeugbewegungsdaten befassen bzw. eine solche modellieren, beruhen auf verlustbehafteten Methoden. Einige der Verfahren möchte ich hier benennen.

Zum einen wurden Verfahren vorgestellt, die Fahrzeugbewegungen durch lineare Approximation beschreiben und somit keinerlei Kartenmaterial benötigen. Zu diesen Arbeiten zählen [LR01], [CJP05], [TCS+06] und [LFDR09]. Eine andere Art, Wegverläufe zu beschreiben bieten die sogenannten nicht-linearen minimax-Polynome [NR07]. Diese verlustbehaftete Methode minimiert den maximalen Näherungsfehler der gegebenen Parameter durch Annäherung an eine gegebene Funktion durch das minimax-Polynom.

Weitere Kompressionstechniken, die sich mit nicht-linearen Annäherungen befassen, verwenden kubische Splines [KKKM10] und Klothoiden [BLP10] und werden in [KBMS11] diskutiert. Generell versuchen diese Verfahren eine Glättung der Fahrzeugbewegung zu modellieren. Ein weiteres verlustbehaftetes Verfahren in der Roboternavigation ist das sogenannte *Probabilistic Positioning*. Hierbei werden statt präzisen Positionen Wahrscheinlichkeiten der Position auf einer diskreten Karte aufgezeigt [Fox98], [RBFT99].

Die beste Kompressionsrate der verlustbehafteten Verfahren erreicht zur Zeit das arithmetische Kodierverfahren vorgestellt in [KM11]. Hier wird der Informationsgehalt von Fahrzeugbewegungen anhand von Wahrscheinlichkeitsmodellen erfasst und damit ein

approximatives arithmetisches Kodierungsschema abgeleitet. Dabei kommt die erreichte Kompressionsrate der oberen Schranke, gegeben durch die Entropie der Daten, sehr nahe.

Einige der oben genannten Verfahren versuchen lediglich den Fehler der verlustbehafteten Methode zu minimieren. Bislang existiert jedoch kein Verfahren, welches eine verlustfreie Kompression der Fahrzeugbewegungsdaten anbietet.

Kapitel 3

Grundlagen

3.1 verlustfreie Kompressionsverfahren

3.1.1 Huffman-Kodierung

David A. Huffman entwickelte im Jahr 1952 eine verlustfreie Kompression durch *Entropiekodierung* (Huffman-Kodierung [Huf52]), welche jedem vorkommenden Zeichen in einer Zeichenfolge je nach deren Häufigkeit eine entsprechend lange Folge von Bits zuordnet. Nach dem Prinzip der informationstheoretischen Entropie, bedingt eine höhere Wahrscheinlichkeit eines Symbols einen kleinen Informationsgehalt. Je kleiner der Informationsgehalt, umso kürzer die Binärkette, mit der das Symbol kodiert werden kann. Im Allgemeinen werden dem Entropiekodierer andere Komprimierungsverfahren vorgeschaltet, um die Entropie der Daten zu verringern. Der Algorithmus der Huffman-Kodierung baut während der Laufzeit einen binären Codebaum auf, und stellt dabei sicher, dass sich die Häufigkeit der vorkommenden Zeichen in der Codelänge widerspiegelt.

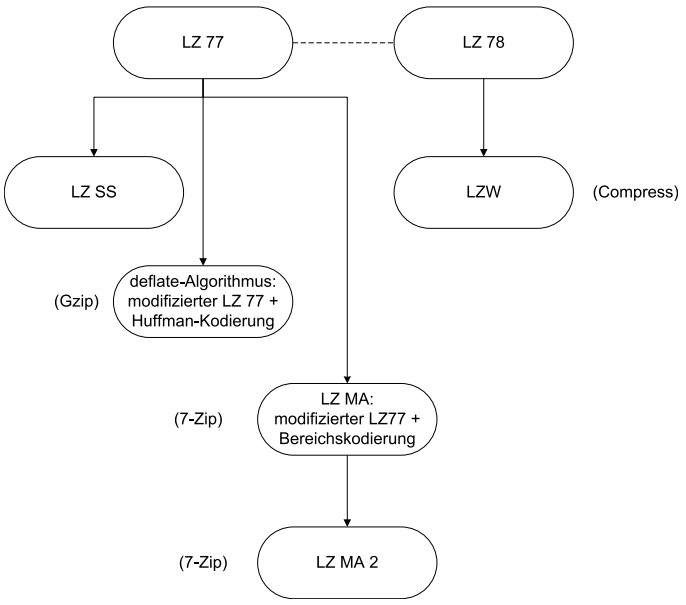


Abbildung 3.1: LZ-Algorithmenfamilie und Packprogramme, die diese nutzen.

Das Erstellen des binären Codebaums erfolgt in den folgenden Schritten:

1. Jedem vorkommenden Zeichen in der zu kodierenden Zeichenfolge wird ihre absolute Häufigkeit zugeordnet und diese in einen Knoten geschrieben.

2. Ordne zwei Knoten mit der geringsten Häufigkeit einem neuen Knoten zu. Dabei soll der entstandene Knoten die Summe der zwei Knoten speichern.

3. Wiederhole Schritt 2, bis sich alle Knoten in einem Baum befinden, wobei bereits zusammengefasste Knoten nicht weiter betrachtet werden.

Jedes Blatt des entstandenen Codebaums enthält nun je ein Zeichen der Zeichenfolge. Der Code für jedes Zeichen lässt sich daher anhand des Pfades, von der Wurzel zum Blatt, ableiten. Wird die linke bzw. rechte Kante von der Wurzel zum Blatt gewählt, so wird dem Code eine 0 bzw. 1 hinzugefügt.

Tabelle 3.1: LZ77 Beispiel: 8 Byte Suchpuffer, 4 Byte Vorschaupuffer. Zu kodierender String: *Mississippi*.

Suchpuffer								*Vorschaupuffer*				*Kodierung*
0	1	2	3	4	5	6	7					
								M	i	s	s	→ (0,0,'M')
							M	i	s	s	i	→ (0,0,'i')
						M	i	s	s	i	s	→ (0,0,'s')
					M	i	s	s	i	s	s	→ (7,1,'i')
			M	i	s	s	i	s	s	i	p	→ (5,3,'p')
i	s	s	i	s	s	i	p	p	i	*EOF*		→ (7,1,'i')
s	i	s	s	i	p	p	i	*EOF*				→ (0,0,*EOF*)

3.1.2 LZ-basierte Kompressionsalgorithmen und -programme

Nachfolgend werden zunächst LZ-basierte Kompressionsalgorihmen und anschließend deren Implementierungen erläutert. Abbildung 3.1 stellt den Zusammenhang der Algorithmenfamilien und deren Implementierungen grafisch dar.

Algorithmen

LZ 77 Der Lempel-Ziv-Algorithmus, der 1977 von Abraham Lempel und Jacob Ziv entwickelt wurde, dient der verlustfreien Datenkompression [ZL77]. Der Algorithmus nutzt die Eigenschaft häufig wiederkehrender Zeichenfolgen und ersetzt diese durch entsprechende Ersatzsymbole. Ein Zeichen bedeutet nachfolgend ein nach dem ASCII-Zeichensatz interpretiertes ein-Byte Datum. Alle Ersatzsymbole werden in einem sogenannten Wörterbuch gespeichert. Diese Komprimierungsmethode wird als *Stringersatzverfahren* bezeichnet.

Der Algorithmus: Zunächst wird ein Fenster konstanter Größe, beispielsweise 12 Byte, in einen Suchpuffer und einen Vorschaupuffer eingeteilt. Der Suchpuffer enthält bereits komprimierte Zeichenketten und stellt somit unser Wörterbuch dar. Der Vorschaupuffer, auch *sliding window* genannt, wandert durch die zu komprimierenden Daten und ver-

Tabelle 3.2: LZ78: Beispiel: Zu kodierender String: *Mississippi*.

kodiert	*zu kodieren*	*hinzugef. Wörterbucheintr.*	*Kodierung*
	Mississippi	(1,'M')	→ (0,'M')
M	ississippi	(2,'i')	→ (0,'i')
Mi	ssissippi	(3,'s')	→ (0,'s')
Mis	sissippi	(4,'si')	→ (3,'i')
Missi	ssippi	(5,'ss')	→ (3,'s')
Mississ	ippi	(6,'ip')	→ (2,'p')
Mississip	pi	(7,'p')	→ (0,'p')
Mississipp	i	(8,'pi')	→ (2,*EOF*)

sucht diese anhand des Wörterbuchs zu ersetzen. Somit werden anstelle der Originaldaten nur Rücksprünge auf bisher kodierte Sequenzen gespeichert.

Kodiert wird durch ein drei-Tupel:

- Position des Eintrags im Wörterbuch.
- Länge der Sequenz.
- Erstes abweichendes Zeichen.

Ist das zu kodierende Datum noch nicht im Wörterbuch enthalten, so ist die Position des Eintrags, sowie dessen Länge gleich null. Der Algorithmus terminiert, sobald der Vorschaupuffer keine weiteren Zeichen mehr enthält. Eine Beispielkodierung wird in Tabelle 3.1 gezeigt.

Es folgen mehrere Verbesserungen des Lempel-Ziv-Algorithmus, darunter die wichtigsten Vertreter LZ78, LZSS, LZW und LZMA.

LZ 78 Das LZ78-Verfahren ist eine Verbesserung des LZ77-Verfahrens, welches Abraham Lempel und Jacob Ziv ein Jahr nach dem Basisverfahren veröffentlichten [ZL78]. Das LZ78-Verfahren nutzt im Gegensatz zum LZ77-Verfahren keine Kombination aus

Adresse und Länge der Zeichenkette, um auf Wörterbucheinträge zurückzugreifen. Wörterbucheinträge werden hier in einer indizierten Tabelle abgelegt und bei wiederkehrenden Zeichenketten per Index abgerufen.

Kodiert wird durch ein zwei-Tupel:

- Index auf einen Eintrag im Wörterbuch.
- Erstes abweichendes Zeichen.

Ist das zu kodierende Datum noch nicht im Wörterbuch enthalten, so ist der Index 0. Das Wörterbuch selbst wird dynamisch zur Laufzeit gebildet. In jedem Kodierungsschritt wird ein Eintrag in das Wörterbuch getätigt: das erste abweichende Zeichen wird an den Eintrag an Stelle des Index angehängt und in einem neuen Index abgelegt. Bei einem stetig wachsendem Wörterbuch werden zunehmend mehr Bits benötigt, um einen Eintrag im Wörterbuch zu adressieren. Hier gilt: Anzahl an benötigten Bits $=\lceil \log_2 Index \rceil$. Eine Beispielkodierung wird in Tabelle 3.2 gezeigt.

LZSS Das LZSS-Verfahren ist nach den beiden Autoren James A. Storer und Thomas G. Szymanski, sowie dem Basisverfahren LZ77, benannt [SS82]. Im Gegensatz zum Basisverfahren LZ77 wird im LZSS-Verfahren per Flag angegeben, ob ein einzelnes Zeichen oder aber ein ganzer String kodiert wird; bei einem einzelnen Zeichen werden 1 Bit Flag und 8 Bit Zeichen gespeichert, bei einem kodierten String werden 1 Bit Flag, Position und Länge gespeichert. Das Positionsfeld im kodierten String zeigt auf eine Adresse im Wörterbuch und die Länge gibt an, wie viele Zeichen kodiert werden. Somit hängt der Speicherbedarf für das Positionsfeld von der maximalen Anzahl an Einträgen im Wörterbuch ab. Das Längenfeld dagegen hängt von der Größe des Vorschaupuffers ab. Das Wörterbuch wird ähnlich zum LZ77-Verfahren durch bereits kodierte Daten gebildet. Abhängig von der Feldbreite des Positions- und Längenfeldes wird entschieden, ab welcher Länge ein String kodiert wird. Eine Beispielkodierung zeigt Tabelle 3.3.

LZW Das LZW-Verfahren ist ein von Terry Welch verbessertes Stringersetzungsverfahren auf Basis des LZ78-Verfahrens [Wel84] und verwendet ebenfalls ein separates

Tabelle 3.3: LZSS Beispiel:Zu kodierender String: *Mississippi*.
3 Bit Positionsfeld, 2 Bit Längenfeld $\widehat{=}$ maximal 2^3 = 8 Positionen im Wörterbuch und maximal 2^2 Byte = 4 Byte Vorschaupuffer. Ein kodierter String würde somit 1 Bit-Flag + 3 Bit + 2 Bit = 6 Bit Speicher belegen. Hieraus folgt, dass es sich schon ab einer Länge von einem Zeichen lohnt statt einem einzelnen Zeichen, einen kodierten String zu erzeugen. Ein einzelnes Zeichen würde nämlich 1 Bit-Flag + 8 Bit = 9 Bit Speicher belegen.

Suchpuffer								*Vorschaupuffer*				*Kodierung*
0	1	2	3	4	5	6	7					
								M	i	s	s	→ (0,‘M’)
							M	i	s	s	i	→ (0,‘i’)
						M	i	s	s	i	s	→ (0,‘s’)
					M	i	s	s	i	s	s	→ (1,7,1)
				M	i	s	s	i	s	s	i	→ (1,5,3)
	M	i	s	s	i	s	s	i	p	p	i	→ (1,2,1)
M	i	s	s	i	s	s	i	p	p	i	*EOF*	→ (0,‘p’)
i	s	s	i	s	s	i	p	p	i	*EOF*		→ (1,7,1)
s	s	i	s	s	i	p	p	i	*EOF*			→ (1,2,1)
s	i	s	s	i	p	p	i	*EOF*				→ (0,*EOF*)

Wörterbuch. Der wesentliche Unterschied zwischen dem LZW- und LZ78-Verfahren ist der, dass das LZW-Verfahren mit einem teilweise gefüllten Wörterbuch startet, während das Wörterbuch des LZ78-Verfahrens zu Beginn leer ist. Die ersten 256 Einträge (0 bis 255) des LZW Wörterbuchs sind mit den ASCII-Zeichen gefüllt. Neu hinzugefügte Zeichen starten somit mit dem Index 256. Eine Beispielkodierung zeigt Tabelle 3.4.

Compress

Das Kompressionsprogramm *Compress* nutzt das LZW-Verfahren als Kompressionsalgorithmus. Als Parameter kann die maximale Anzahl an Bits für den Index des Wörterbuchs angegeben werden. Der Standardwert hierfür ist 16, welcher zugleich das Maximum darstellt. Compress wird bei der Auswertung in Kapitel 5 auf den kodierten Bytestrom angewandt und mit anderen Kompressionsprogrammen verglichen.

Tabelle 3.4: LZW Beispiel: Zu kodierender String: *Mississippi*.

kodiert	*zu kodieren*	*hinzugef. Wörterbucheintr.*	*Kodierung*
	Mississippi	(256,‘Mi’)	→ (77,‘i’)
M	ississippi	(257,‘is’)	→ (105,‘s’)
Mi	ssissippi	(258,‘ss’)	→ (115,‘s’)
Mis	sissippi	(259,‘si’)	→ (115,‘i’)
Miss	issippi	(260,‘iss‘)	→ (257,‘s’)
Missis	sippi	(261,‘sip’)	→ (259,‘p’)
Missis	ppi	(262,‘pp’)	→ (112,‘p’)
Mississip	pi	(263,‘pi’)	→ (112,‘i’)
Mississipp	i	-	→ (105,EOF)
Mississippi	*EOF*	-	→ (4,-)

GZip

Das Kompressionsprogramm *GZip* [Deu96b] implementiert den sogenannten *deflate*-Kompressionsalgorithmus [Deu96a]. Der deflate-Algorithmus verwendet eine modifizierte Version des LZ77-Algorithmus mit anschließender Huffman-Kodierung [Huf52]. An die komprimierte Datei wird daraufhin ein Header und Trailer angefügt, welche u.a. Informationen über die komprimierte Datei, sowie eine Checksumme der unkomprimierten Daten enthalten. Beim Start des Kompressionsprogramms kann ein Performance-Parameter übergeben werden. Der Parameter beginnt bei 1, schnelle Kompression, und geht bis 9, beste Kompression. Im Wesentlichen beeinflusst der Parameter das Suchverhalten des im LZ77-Algorithmus angelegten Wörterbuchs. So wird bei einer schnellen Kompression der erste Treffer im Wörterbuch gewählt; bei der besten Kompression wird das Wörterbuch unter Umständen sogar *nach* einem Treffer ein zweites Mal durchsucht.

GZip ist ein sehr weit verbreitetes Kompressionsprogramm, welches gute Kompressionsraten bei einer vergleichsweise kurzen Ausführungsdauer bietet. Aus diesem Grund wird GZip, mit dem Performance-Parameter 9, bei der Auswertung mit anderen Kompressionsprogrammen verglichen.

7-Zip

Das Kompressionsprogramm *7-Zip* nutzt unter anderem den Kompressionsalgorithmus *LZMA* bzw. *LZMA2*:

LZMA1: Das LZMA-Verfahren beinhaltet unter anderem eine Modifizierung des LZ77-Verfahrens, welches seit 1998 von Igor Wiktorowitsch Pavlov entwickelt wird. Der Algorithmus wurde von ihm selbst in 7-Zip [Pav07] implementiert. Benannt wurde der Algorithmus nach den Urvätern Abraham Lempel und Jacob Ziv sowie Andrei Andrejewitsch Markov's *Markov-Ketten*. Der Algorithmus wendet zunächst den modifizierten LZ77-Algorithmus an, um anschließend eine Bereichskodierung durchzuführen. Die Bereichskodierung ist eine Art arithmetischer Kodierung [Mar79]. Die Markovketten kommen bei Wahrscheinlichkeitsberechnungen in der Bereichskodierung sowie Teilen des modifizierten LZ77-Algorithmus zur Geltung.

LZMA2: Das LZMA2-Verfahren ergänzt das LZMA-Verfahren lediglich um ein Performance-Tuning und eine Unterstützung von *multi-threading*. Um das LZMA2-Verfahren mit den Vorgängern der LZ-Familie zu vergleichen, wird das Kompressionsprogramm 7-Zip samt LZMA2-Algorithmus in die Auswertung einbezogen.

3.1.3 BZip2

Das Kompressionsprogramm BZip2 besteht aus einem dreistufigen Verfahren. Zunächst werden die Eingangsdaten blockweise mit der *Burrows-Wheeler-Transformation* sortiert und deren Ausgabe einer *move-to-front-Transformation* unterzogen. Beide Verfahren werden in [BW94] ausführlich anhand eines Beispiels erklärt. Diese ersten beiden Verfahren verringern lediglich die Entropie der Daten, so dass Entropiekodierer bessere Kompressionsraten erzielen können. Abschließend durchlaufen die vorbehandelten Daten eine Entropiekodierung nach Huffman [Huf52]. Je nach Wahrscheinlichkeitsmodell komprimieren Entropiekodierer, wie beispielsweise die Huffman-Kodierung, wiederkehrende Zeichen sehr effizient. Aus diesem Grund wird das Kompressionsprogramm BZip2

in der Auswertung auf die kodierten GPS-Traces angewandt und mit anderen herkömmlichen Kompressionsprogrammen verglichen.

3.1.4 Arithmetische Kodierung

Die arithmetische Kodierung [Mac02] ist ein weiteres verlustfreies Kompressionsverfahren, das eine Entropiekodierung nutzt. Zur Erinnerung: Je wahrscheinlicher das Auftreten eines Symbols im Symbolstrom, umso kleiner ist auch der Informationsgehalt und somit kann das Symbol mit einer kurzen Binärkette kodiert werden.

Gegeben sei ein Symbolstrom, welcher Symbol für Symbol verarbeitet wird. Der Algorithmus teilt das reelle Intervall $[0,1)$ rekursiv bezüglich eines Modells in Teilintervalle auf und stellt dieses einem binär geschachtelten Intervall gegenüber, durch das am Schluss der Code bestimmt wird. Mögliche Modelle wären beispielsweise eine Häufigkeitsverteilung der vorkommenden Symbole im Symbolstrom oder das sogenannte *prediction by partial matching* (PPM)-Verfahren. Die Häufigkeitsverteilung ist kein optimales Modell, da die Verteilung der Symbole bekannt sein muss. Das PPM(n)-Verfahren hingegen ist ein lernfähiges statistisches Verfahren, welches n vorangehende Symbole des Symbolstroms nutzt, um das nächste Symbol vorherzusagen.

Ist das Intervall nun bezüglich des gewählten Modells in Teilintervalle aufgeteilt, so wird jedem Teilintervall ein Symbol zugeteilt. Das Teilintervall, in welchem das nächste zu kodierende Symbol des Symbolstroms liegt, wird zum aktuellen Intervall. Gilt es weitere Symbole zu kodieren, so wird der oben beschriebenen Prozess für das aktuelle Intervall wiederholt. Sind alle Symbole des Symbolstroms verarbeitet, so besteht das Komprimat aus einem reellen Subintervall und einem zugehörigen Binärsubintervall. Das Binärintervall wird dabei so gewählt, dass es im reellen Subintervall maximal ist.

Die Implementierung des Verfahrens in seiner Grundform weist einen großen Nachteil auf: Der Algorithmus ist auf den unendlichen reellen Zahlen definiert, somit benötigt man unendlich lange Fließkommazahlen. Es stehen jedoch nur reelle Zahlen mit einer gewissen Präzision zur Verfügung. Viele Implementierungen einer arithmetischen Kodierung, wie beispielsweise die von *Bob Carpenter* [Cap02], nutzen einen sogenannten

Bereichskodierer (Range-Coder), welcher auf Basis ganzer Zahlen arbeitet. Bob Carpenters Implementierung nutzt zudem eine breite Auswahl an Modellen bezüglich der Intervallaufteilung. Das PPM(n)-Modell verspricht bei einem hinreichend großen n eine hohe Kompressionsrate unter Einbüßung von Kompressionsgeschwindigkeit. Weitere Informationen hinsichtlich Bob Carpenters Implementierung sind [Cap02] zu entnehmen. In meiner Auswertung wird der arithmetische Kodierer von Bob Carpenter samt dem PPM(16)-Modell auf den kodierten Bytestrom angewandt und mit anderen Kompressionsprogrammen verglichen.

Kapitel 4

Bytekodierer

Im Hauptteil dieser Arbeit gilt es einen Bytekodierer zu entwerfen, welcher eine Folge von GPS-Punkten geschickt, sowie verlustfrei kodiert. Die GPS-Punktfolgen liegen jeweils in einer CSV-Datei vor und die GPS-Punkte selbst sind in Dezimalgrad gegeben. In meiner Implementierung werden gerichtete Differenzvektoren erster und zweiter Ordnung verwendet; Differenzvektoren erster Ordnung werden anhand von zwei GPS-Punkten errechnet, wohingegen Differenzvektoren zweiter Ordnung anhand von zwei Differenzvektoren erster Ordnung errechnet werden. Ein Differenzvektor wird im Allgemeinen aus der koordinatenweise Differenz zweier Objekte gebildet. Nachfolgend werden Differenzvektoren erster/zweiter Ordnung durch Δ_1-/Δ_2-Vektor abgekürzt.

Die Idee Differenzvektoren zu speichern beruht auf der Tatsache, dass sich GPS-Punkte bei einem ausreichend kleinen Abtastintervall nicht drastisch verändern. Das Abtastintervall liegt in unserer Datenbasis bei einer Sekunde. Diese Beobachtung trägt dazu bei, dass die gebildeten Δ_1-Vektoren relativ klein ausfallen und somit wenig Speicherbedarf anfällt. Nehmen wir an, dass sich unser Fahrzeug mit gleichmäßiger Geschwindigkeit in eine Richtung bewegt (vergleiche eine Autobahnfahrt). So ändern sich die gebildeten Δ_1-Vektoren kaum, so dass die errechneten Δ_2-Vektoren sehr klein bzw. null sind. Man kann somit den Δ_1-Vektor auch als *Geschwindigkeitsvektor* und den Δ_2-Vektor als *Beschleunigungsvektor* bezeichnen. Hieraus lässt sich folgern, dass unser generierter Bytestrom unter einigen Voraussetzungen oft wiederkehrende Symbole enthält. Die Eigenschaft von

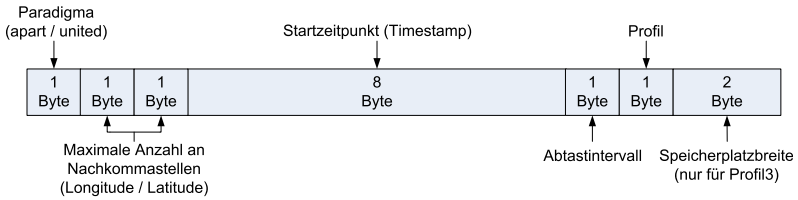


Abbildung 4.1: Schema des Headers für die Bytekodierung.

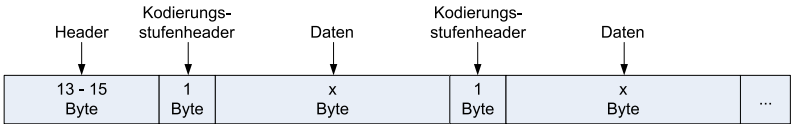


Abbildung 4.2: Genereller Aufbau des erzeugten Bytestroms.

immer wiederkehrenden Symbolen im Bytestrom wirkt sich auf die Kompression durch verlustfreie Kompressionsverfahren positiv aus.

4.1 Realisierungsarten

In diesem Abschnitt möchte ich den Aufbau des implementierten Bytestroms diskutieren. Anzumerken sei, dass ich einen ausgerichteten Bytestrom erzeugen werde. Das heißt, dass der Bytestrom byteweise, und nicht etwa bitweise, erzeugt wird. Da der erzeugte Bytestrom anschließend mit Kompressionsprogrammen aus Kapitel 3 komprimiert wird und jedes der vorgestellten Kompressionsprogramme byteweise arbeitet, sollte unser Bytestrom ebenfalls ausgerichtet sein. Zunächst einmal betrachten wir die Art und Weise wie wir GPS-Punkte speichern. Die oben genannten CSV-Dateien enthalten GPS-Punkte in Dezimalgraden mit einer variablen Anzahl an Nachkommastellen. Die Speicherung dieser Dezimalzahlen im Gleitkommaformat (also beispielsweise mittels IEEE 754) kommt aus einem bestimmten Grund nicht in Frage: Die Gleitkommaarithmetik verursacht unter Umständen Rundungsfehler, die in der verlustfreien Kompression nicht tragbar sind. Es gibt genau zwei Möglichkeiten Dezimalzahlen im Gleitkommaformat zu ganzen Zahlen zu konvertieren. Aus diesem Grund habe ich beide Speicherungsvarianten implementiert.

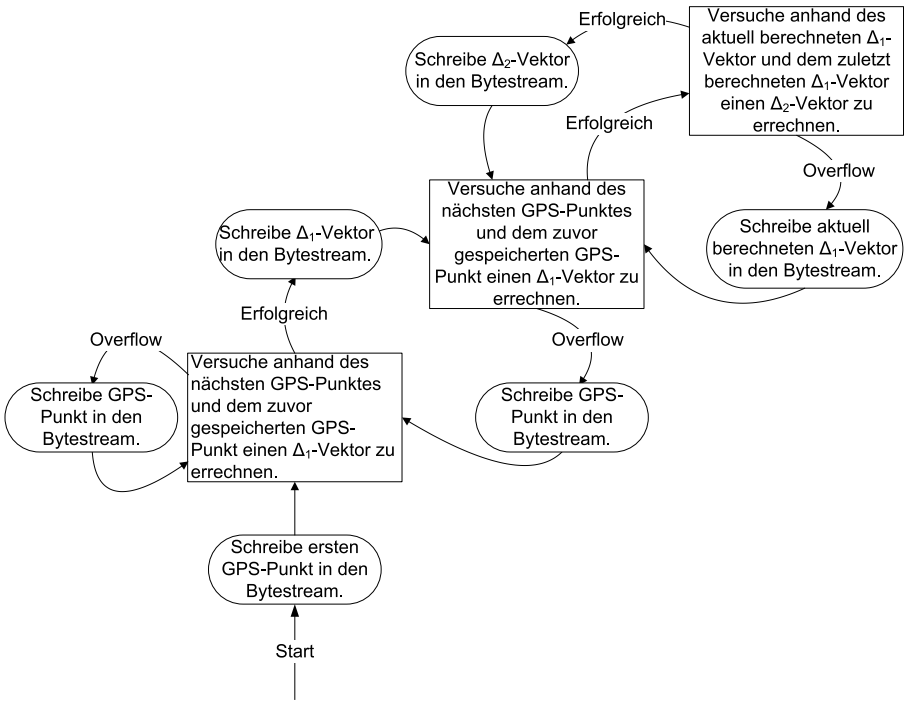


Abbildung 4.3: Ablaufdiagramm des Algorithmus.

- Die *apart*-Variante: GPS-Punkte werden koordinatenweise in einen Vorkommateil, dem *prefix*, und einen Nachkommateil, dem *suffix*, getrennt und gespeichert. Mehr dazu in Kapitel 4.2.

- Die *united*-Variante: Koordinaten eines GPS-Punktes werden durch eine Verschiebung des Kommas zu einer ganzen Zahl und werden in dieser Form gespeichert. Mehr dazu in Kapitel 4.3.

Für beide Paradigmen ist es notwendig, die maximale Anzahl an Nachkommastellen des zugrunde liegenden GPS-Traces zu kennen. Diese Information ist für den Dekodierer essenziell, um aus kodierten ganzen Zahlen wiederum Dezimalzahlen im Gleitkommaformat zu rekonstruieren. Diese, sowie einige weitere Informationen müssen in einem Header gespeichert werden. Vergleiche hierzu Abbildung 4.1. Nach dem einleitenden Header folgen die eigentlichen zu kodierenden Daten.

Wie weiter oben beschrieben gibt es hier drei Kodierungsstufen.

- GPS-Punkt
- Δ_1-Vektor
- Δ_2-Vektor

Um dem Dekodierer mitzuteilen in welcher Kodierungsstufe kodiert wurde, wird in einem führenden Byte, dem *Kodierungsstufenheader*, eben diese Information gespeichert. Nach dem Kodierungsstufenheader folgt letztendlich das Datum. Vergleiche hierzu Abbildung 4.2. Den Zusammenhang zwischen den Kodierungsstufen stellt Abbildung 4.3 dar. Hier wird deutlich, dass der Algorithmus in einem Stufensystem arbeitet, d.h. die Wahl der nächsten Kodierungsstufe ist von der zuletzt verwendeten Kodierungsstufe abhängig. Hierbei wird lokal optimiert und immer die nächsthöhere Kodierungsstufe angestrebt, wodurch man den Algorithmus als *greedy* bezeichnen kann. Gibt es ein Überlauf (Overflow) bei der Bildung eines Differenzvektors, so wählt man die nächstkleinere Kodierungsstufe. Wie es zu einem Überlauf kommen kann, wird in Kapitel 4.2 bzw. 4.3 erläutert.

Weiterhin habe ich je Speicherungsvariante drei Profile implementiert. Die Profile unterscheiden sich untereinander im wesentlichen durch verschiedene Speicherplatzbreiten für die einzelnen Kodierungsstufen. Folglich sind die Speicherplatzbreiten für das erste Profil, auch *Bruteforce*-Kodierung genannt, maximal. Das zweite Profil hingegen implementiert eine Halbierung der Speicherplatzbreiten bei wachsendem Grad des Differenzvektors. Die Speicherplatzbreite für die GPS-Punkte ist hierbei so gewählt, dass alle möglichen Absolutwerte hineinpassen. Letztlich ist das dritte Profil ein analytisches Profil. Anfangs wird eine Statistik über den übergebene GPS-Trace erstellt, anhand dessen die Speicherplatzbreiten der Kodierungsstufen festgelegt werden. Die erstellte Statistik enthält eine Liste mit allen GPS-Punkten und möglichen Differenzvektoren erster, sowie zweiter Ordnung. Zunächst ermittelt das Programm die minimale Speicherplatzbreite für die GPS-Punkte. Hierbei wird die Speicherplatzbreite so gewählt, dass alle vorkommenden GPS-Punkte des Traces hineinpassen. Anders als bei den Differenzvektoren erster bzw. zweiter Ordnung. Hier wird lediglich für einen Prozentsatz der möglichen Differenzvektoren die minimale Speicherplatzbreite ermittelt und gewählt.

Die gewünschten Prozentsätze für Δ_1- und Δ_2-Vektoren werden dem Programm zu Beginn der Ausführung übergeben. Nachfolgend wird der übergebene Prozentsatz für Δ_1-Vektoren als p_1 und für Δ_2-Vektoren als p_2 bezeichnet. Die Berechnung der Speicherplatzbreite für einen gegebenen Prozentsatz erfolgt mithilfe einer kumulativen Verteilungsfunktion über die möglichen Differenzvektoren. In Kapitel 4.2.4 bzw. 4.3.3 wird die Berechnung der Speicherplatzbreite anhand eines konkreten Beispiels präzisiert.

Das analytische Profil würde in dieser Form nicht in der Praxis auftreten sondern dient lediglich theoretischen Zwecken. In der Praxis würde zunächst ein optimaler p-Wert errechnet werden, der die besten Ergebnisse für den individuellen Einsatzzweck erzielt und anschließend würden dementsprechende Speicherplatzbreiten fest definiert werden. Dies ist notwendig, da große Lastspitzen, beispielsweise erzeugt durch die Berechnung des p-Werts, bei der Kodierung von GPS-Traces vermieden werden sollten. Um die Last optimal über die Zeit zu verteilen würde die Kodierung außerdem on-the-fly, d.h. während der Fahrt, erfolgen.

4.2 Kodierung des Bytestroms (*apart*-Variante)

Ich betrachte in diesem Teil der Arbeit die Kodierung des Bytestroms bei einer Trennung des Vor und Nachkommateils der GPS-Koordinaten.

Dabei kann die Vorkommazahl einer Koordinate vorerst bedenkenlos als ganze Zahl übernommen werden. Zur Abtrennung der Nachkommazahl wird anfangs für den GPS-Trace die maximale Anzahl an Nachkommastellen für die Längen- (longitude) und die Breitengradkoordinaten (latitude) ermittelt. Anschließend wird jede Nachkommazahl zunächst bis zur Maximallänge mit Nullen von rechts aufgefüllt und abschließend werden führende Nullen eliminiert. Das Ergebnis ist eine ganze Zahl, mit der weitergerechnet werden kann. Der Dekodierer kann mit der Information der maximalen Anzahl an Nachkommastellen, Vor- und Nachkommateil verlustfrei zusammenfügen.

Nachdem Vor- und Nachkommateil erfolgreich voneinander getrennt wurden, steht die Berechnung von Differenzvektoren an. Ein Differenzvektor ist die koordinatenweise Differenz des Vor- und Nachkommateils. Seien $(a;b)$ und $(c;d)$ aufeinanderfolgende GPS-Punkte in einem GPS-Trace mit den Längengraden a bzw. c und den Breitengraden b bzw. d. Des Weiteren seien a_{Pre} und a_{Suf} der Vorkomma- bzw. Nachkommateil der Koordinate a (b, c und d analog). Der Differenzvektor setzt sich wie folgt zusammen:

$$\Delta_1 = (c_{Pre} - a_{Pre}.c_{Suf} - a_{Suf}; d_{Pre} - b_{Pre}.d_{Suf} - b_{Suf}) \tag{4.1}$$

Die Berechnung eines Differenzvektors zweiter Ordnung erfolgt analog zur Berechnung eines Differenzvektors erster Ordnung. Der Unterschied ist, dass Differenzvektoren zweiter Ordnung die koordinatenweise Differenz von zuvor errechneten Differenzvektoren erster Ordnung sind. Somit bedarf es bei der Bildung von Differenzvektoren zweiter Ordnung keine weitere Behandlung der Differenzvektoren erster Ordnung. Ein Überlauf wie in Abbildung 4.3 erfolgt, falls eine der Differenzen außerhalb der definierten Speicherplatzbreite des Differenzvektors liegt. Dies tritt auf, da die Speicherplatzbreite der Kodierungsstufen, je nach verwendetem Profil, variiert. Bevor die drei Profile des apart-Paradigmas betrachtet werden, wird auf eine Problematik bei der Trennung des Vor- und Nachkommateils hingewiesen.

4.2.1 Problematik

Durch die Trennung des semantisch zusammenhängenden Vor- und Nachkommateils entsteht eine Ineffizienz bei der Bildung von Differenzvektoren. Betrachte einen Auszug aus einem GPS-Trace:

```
# Timestamp;longitude;latitude
[...]
1304072340;4.9998;52.4256
1304072341;5.0001;52.7323
[...]
```

Betrachtet wird lediglich die Längenkoordinate (longitude) mit einer maximalen Anzahl an Nachkommastellen von vier. Der Differenzvektor erster Ordnung wird anhand der Formel in 4.1 wie folgt berechnet: Der Vorkommateil des Differenzvektors beträgt $5-4=1$ und der Nachkommateil $1-9998=-9997$. Die Berechnung des Nachkommateils ist ineffizient, da der Sprung von 9998 auf 0001 bei einer maximalen Anzahl an Nachkommastellen von 4, lediglich 3 ist. Wir wollen also gewährleisten, dass der Differenzvektor im Nachkommateil immer den minimalen Abstand der Nachkommazahlen berechnet, wobei die maximale Anzahl an Nachkommastellen beachtet wird. Umgesetzt wird dies durch eine modulo-Operation, gefolgt von einer Fallunterscheidung. Folgende Rechenschritte sind notwenig: Sei $\text{diff}_{old-result}$ das Ergebnis der Differenz und $\max_{Suf}$ die maximale Anzahl an Nachkommastellen des Traces.

$$\text{diff}_{interim-result} = \text{diff}_{old-result} \; mod \; 10^{\max_{Suf}}$$

$$Falls: \text{diff}_{interim-result} > \frac{10^{\max_{Suf}}}{2} \Rightarrow \text{diff}_{new-result} = \text{diff}_{interim-result} - 10^{\max_{Suf}}$$

$$Falls: \text{diff}_{interim-result} < -\frac{10^{\max_{Suf}}}{2} \Rightarrow \text{diff}_{new-result} = \text{diff}_{interim-result} + 10^{\max_{Suf}}$$

$$Andernfalls: \text{diff}_{new-result} = \text{diff}_{interim-result}$$

Bei der Dekodierung wird nach der Berechnung des Nachkommateils des zugrunde liegenden GPS-Punktes lediglich das Ergebnis $mod 10^{\max_{Suf}}$ gerechnet.
Eine weitere Problematik wird anhand eines GPS-Trace Auszugs erklärt:

```
# Timestamp;longitude;latitude
[...]
1304072340;0.0036;52.4256
1304072341;-0.0001;52.7323
[...]
```

Betrachtet wird hierbei nur der Vorkommateil bei der Berechnung eines Differenzvektors der Längenkoordinate. Dieser wäre $-0-0=0$. Da die Null kein Vorzeichen trägt, ist es nicht möglich -0 von 0 zu unterscheiden. Aus diesem Grund wird in der Implementierung bei solchen Sonderfällen das Vorzeichen des kodierten GPS-Punktes im Kodierungsstufenheader gespeichert.

Tabelle 4.1: Speicherplatzbreiten der apart-Profile.

Art der Information	*Längengrad*		*Breitengrad*	
	Vorkommateil	Nachkommateil	Vorkommateil	Nachkommateil
Profil 1	2	8	2	8
Profil 2				
GPS-Punkt	2	4-8	1	4-8
Δ_1-Vektor	2	2-4	1	2-4
Δ_2-Vektor	2	1-2	1	1-2
Profil 3				
GPS-Punkt	2	1-8*	1	1-8*
Δ_1-Vektor	1	1-8**	1	1-8**
Δ_2-Vektor	-	1-8**	-	1-8**

[*]: minimal nach Analyse.
[**]: Ermittlung per kumulativer Verteilungsfunktion.

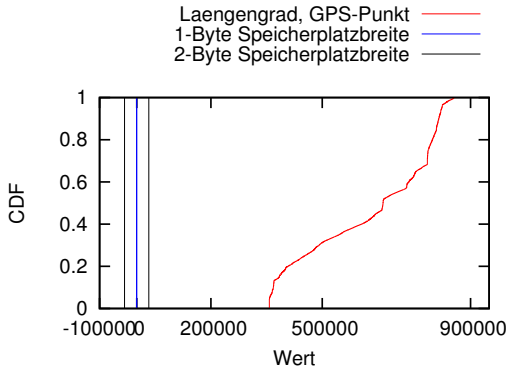
4.2.2 Profil 1

Wie in Kapitel 4.1 bereits angesprochen, handelt es sich bei Profil 1 um ein Bruteforce-Profil. Tabelle 4.1 zeigt, dass für den Vorkommateil zwei Byte und für den Nachkommateil acht Byte pro Koordinate reserviert werden. Zwei Byte fassen den Zahlenbereich von -32768 bis 32767; acht Byte umfassen den Zahlenbereich von $-9.22 \cdot 10^{18}$ bis $9.22 \cdot 10^{18}$. Dies gilt für einen gespeicherten GPS-Punkt, Δ_1-Vektor, sowie Δ_2-Vektor. Hieraus folgt, dass alle Dezimalgrade mit bis zu 18 Nachkommastellen gespeichert werden können. Die großzügige Wahl der Speicherplatzbreiten bedeutet, dass es bei der Bildung der Differenzvektoren nie zu einem Überlauf kommen kann. Somit wird der mit Profil 1 erzeugte Bytestrom für ein gegebenen GPS-Trace immer genau einen GPS-Punkt, einen Δ_1-Vektor und sonst nur Δ_2-Vektoren beinhalten. Anhand des ersten Profils möchte ich in der Auswertung testen, wie gut solche Byteströme von verlustfreien Kompressionsverfahren komprimiert werden.

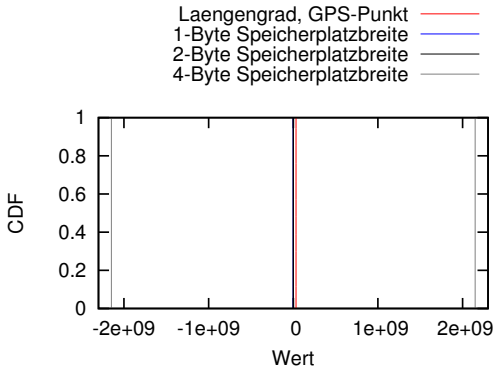
4.2.3 Profil 2

Das zweite, intuitive Profil samt der Speicherplatzbreiten der Kodierungsstufen wird in Tabelle 4.1 gezeigt. Die Speicherplatzbreite für GPS-Punkte ist dabei so bemessen, dass alle möglichen Absolutwerte bis 18 Nachkommastellen hineinpassen. Der mögliche Wertebereich für den Vorkommateil der Längengrade ist $(-180;180]$. Hierfür reicht ein Byte nicht aus, weshalb zwei Byte reserviert werden. Die Breitengrade beinhalten im Vorkommateil die Werte $(-90;90]$, somit ist hier ein Byte Speicherplatz ausreichend. Je nach Anzahl der Nachkommastellen werden jeweils für die Längen- und Breitenkoordinate vier bzw. acht Byte für den Nachkommateil reserviert. Bis einschließlich neun Nachkommastellen werden vier Byte und ab zehn Nachkommastellen werden acht Byte reserviert.

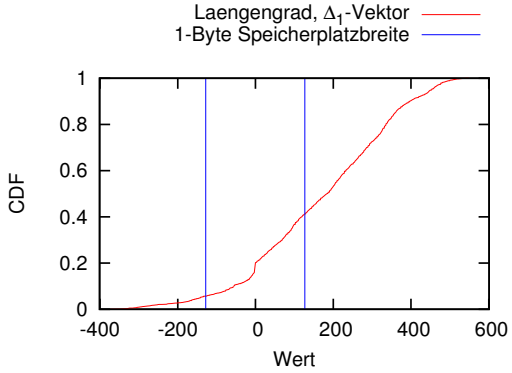
Bei einem hinreichend kleinen Abtastintervall der GPS-Traces, sollten die Abweichungen ersten Grades intuitiv weniger Speicherplatz benötigen. Das Gleiche gilt für die Abweichungen zweiten Grades. Wie weiter oben beschrieben ist dies lediglich ein intuitives Profil, welches, im Gegensatz zu Profil 1, für die unterschiedlichen Kodierungsstufen verschiedene Speicherplatzbreiten anbietet. Hierdurch sollte der kodierte Bytestrom wesentlich kompakter als im ersten Profil ausfallen. Aus diesem Grund werden in diesem Profil Nachkommaspeicherplatzbreiten mit zunehmendem Grad der Differenzvektoren halbiert. Somit werden für den Nachkommateil der Differenzvektoren erster Ordnung zwei bzw. vier Byte reserviert. Für den Nachkommateil der Differenzvektoren zweiter Ordnung werden folglich ein bzw. zwei Byte Speicher reserviert. Die Speicherplatzbreite des Vorkommateils der Differenzvektoren wird im Gegensatz zu dem Nachkommateil nicht halbiert. Der Speicherbedarf für den Vorkommateil bleibt somit, über die Differenzvektorgrade hinweg, unverändert. Durch den anschließenden Einsatz von verlustfreien Kompressionsverfahren soll geprüft werden, welchen Einfluss ein bereits kompakt kodierter Bytestrom auf die Kompressionsrate der Kompressionsverfahren hat.



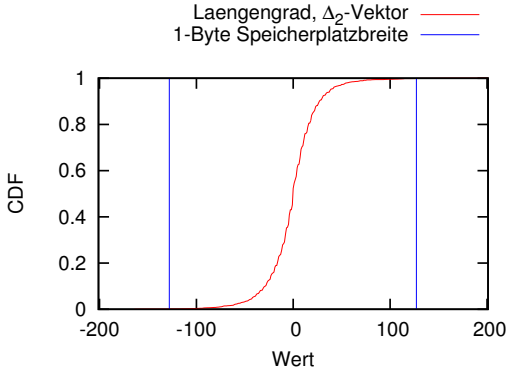
(a) apart: CDF des Nachkommateils aller Längengrade eines GPS-Traces.



(b) united: CDF aller Längengrade eines GPS-Traces.



(c) CDF des Nachkommateils aller Δ_1-Vektoren für die Längengrade eines GPS-Traces.



(d) CDF des Nachkommateils aller Δ_2-Vektoren für die Längengrade eines GPS-Traces.

Abbildung 4.4: Kumulative Verteilungungsfunktionen.

4.2.4 Profil 3

Tabelle 4.1 zeigt die Speicherplatzbreiten für die einzelnen Kodierungsstufen des dritten Profils. Anders als bei den zuvor betrachteten Profilen, werden die Speicherplatzbreiten für den Nachkommateil der Kodierungsstufen dynamisch zur Laufzeit gebildet. Wie in Kapitel 4.1 bereits angedeutet, werden für den Nachkommateil die Speicherplatzbreiten der Kodierungsstufen anhand einer kumulativen Verteilung berechnet. Dies wird nun anhand eines Beispiels konkretisiert.

Abbildung 4.4(a) zeigt die kumulative Datenverteilung des Nachkommateils aller Längengrad-Koordinaten eines konkreten repräsentativen GPS-Traces. Da sich alle Werte des Nachkommateils im Bereich von etwa 360000 bis 850000 bewegen, hat der Dezimalteil eine Länge von sechs. Die Y-Achse stellt dabei die kumulative Verteilungsfunktion (*cumulative distribution function*, cdf) dar. Außerdem werden die Speicherplatzbreiten markiert. Der Übersichtlichkeit wegen wird die 4-Byte Speicherplatzbreite nicht in die Grafiken eingezeichnet. Da es sich hierbei um die cdf der Längengrade (GPS-Punkte) handelt, müssen alle Werte in die Speicherplatzbreite hinein passen. Aus diesem Grund wird für dieses Beispiel eine Speicherplatzbreite von vier Byte reserviert.

Abbildung 4.4(c) zeigt eine kumulative Datenverteilung desselben GPS-Traces für die zugehörigen möglichen Differenzvektoren ersten Grades. Die Differenzvektoren bewegen sich hier in einem Bereich von etwa -370 bis 550. Übersichtlichkeitshalber ist hier nur die Speicherplatzbreite von einem Byte aufgezeigt. Zu sehen ist, dass etwa 35 % aller Δ_1-Vektoren in der 1-Byte Speicherplatzbreite liegen. Somit würde bei einem gegebenen p_1-Wert von < 0.35 ein Byte für die Speicherplatzbreite des Nachkommateils der Δ_1-Vektor-Längenkoordinate reserviert werden. Ist der gegebene p_1-Wert ≥ 0.35, so würden zwei Byte reserviert werden.

Schließlich zeigt Abbildung 4.4(d) die Verteilungsfunktion desselben GPS-Traces für die zugehörigen möglichen Differenzvektoren zweiten Grades. Auffällig ist, dass $> 99\,\%$ aller Δ_2-Vektoren in der 1-Byte Speicherplatzbreite liegen. Nur $< 1\,\%$ der möglichen Δ_2-Vektoren liegen außerhalb des 1-Byte Bereichs. Somit würde für die Δ_2-Vektor-Längengradkoordinate bei einem p_2-Wert von ≤ 0.99 ein Byte und bei einem p_2-Wert von 1.00, zwei Byte reserviert werden.

Analoge Berechnungen erfolgen für den Nachkommateil der Breitengrad-Koordinate. Somit wird je eine Speicherplatzbreite ermittelt, in der alle vorkommenden Breitengrad-Koordinaten, p_1 aller möglichen Δ_1-Vektoren und p_2 aller möglichen Δ_2-Vektoren hineinpassen. Da der Dekodierer die verwendeten Speicherplatzbreite benötigt, um den zuvor kodierten Bytestrom zu dekodieren, werden alle sechs errechneten Speicherplatzbreiten in den Header kodiert. Vergleiche hierzu Abbildung 4.1.

Eine weitere Besonderheit des dritten Profils ist die Verkleinerung der Speicherplatzbreite für den Vorkommateil bei Δ_1-Vektoren bzw. der Wegfall des Vorkommateils bei Δ_2-Vektoren. Um den kodierten Bytestrom so kompakt wie möglich zu gestalten, speichern Δ_1-Vektoren nur noch ein Byte Vorkomma Längengrade ab. Mit dem gleichen Ziel erfolgte der gänzliche Wegfall des Vorkommateils bei Δ_2-Vektoren. Anzumerken sei, dass Differenzvektoren zweiten Grades auch bei einer Veränderung des Vorkommateils gespeichert werden können. Nämlich genau dann, wenn anhand des Überlaufverhaltens des Nachkommateils der Vorkommateil errechnet werden kann. Im Prinzip wird hierbei, ähnlich zum Verfahren in Kapitel 4.2.1, zunächst der Differenzvektor zweiten Grades gebildet und anschließend geprüft, ob der zugrunde liegende GPS-Punkt anhand des Überlaufverhaltens des Nachkommateils rekonstruiert werden kann. Ist dies der Fall, so wird der berechnete Δ_2-Vektor in den Bytestrom geschrieben. Lässt sich der Vorkommateil nicht anhand des Überlaufverhaltens rekonstruieren, so wird stattdessen der zuvor errechnete Δ_1-Vektor in den Bytestrom geschrieben. Treten im GPS-Trace keine großen Sprünge auf, so ist die Rekonstruktion immer möglich. Durch diese Einsparung wird eine kompakte Bytekodierung erhofft.
Die Wahl eines optimalen p_1 bzw. p_2 und deren Einfluss auf die Kompressionsrate wird in der Auswertung ausführlich diskutiert.

4.3 Kodierung des Bytestroms (*united*-Variante)

Im Gegensatz zum apart-Paradigma werden im *united*-Paradigma Dezimalgrade als Festkommazahlen gespeichert. Dies ist im wesentlichen eine Ganzzahl mit einer Kommaverschiebung um die ermittelte maximale Anzhahl an Nachkommastellen des zugrunde liegenden GPS-Traces. Der Dekodierer kann anhand der mitgelieferten maximalen Anzahl an Nachkommastellen die GPS-Punkte durch eine Repositionierung des Kommas wiederherstellen. Die Berechnung eines Differenzvektors erfolgt folgendermaßen: Seien $(a;b)$ und $(c;d)$ aufeinanderfolgende GPS-Punkte in einem GPS-Trace mit den Längengraden a bzw. c und den Breitengraden b bzw. d. Sei weiterhin a_{united} der Zusammenschluss des Vor- und Nachkommateils der Koordinate a (b, c und d analog).

Tabelle 4.2: Speicherplatzbreiten der united-Profile.

Art der Information	*Längengrad*	*Breitengrad*
Profil 1	8	8
Profil 2		
GPS-Punkt	4-8	4-8
Δ_1-Vektor	2-4	2-4
Δ_2-Vektor	1-2	1-2
Profil 3		
GPS-Punkt	1-8*	1-8*
Δ_1-Vektor	1-8**	1-8**
Δ_2-Vektor	1-8**	1-8**

[*]: minimal nach Analyse.
[**]: Ermittlung per kumulativer Verteilungsfunktion.

Der Differenzvektor erster Ordnung ist eine koordinatenweise Differenz von zwei GPS-Punkten. Die Berechnungen lauten wie folgt:

$$x_{united} = x \cdot 10^{Nachkommastellen}$$
$$\Delta_1 = (c_{united} - a_{united}; d_{united} - b_{united})$$

Die Berechnung des Differenzvektors zweiter Ordnung erfolgt durch die koordinatenweise Differenz von zwei Differenzvektoren erster Ordnung. Anders als im apart-Paradigma, gibt es hier weder Probleme bei der Erstellung von Differenzvektoren noch müssen Sonderfälle bei der Speicherung in Betracht gezogen werden. Im apart-Paradigma war dies der Preis, den man für die Trennung der Vorkomma- von der Nachkommazahl, zahlen musste. Nachfolgend werden die Profile der united-Variante vorgestellt. Da sich die Profile der beiden Paradigmen ähneln, werden nur die Unterschiede erläutert. Diese liegen größtenteils in der Aufteilung der Speicherplatzbreite.

4.3.1 Profil 1

Wie auch im apart-Paradigma, ist das Profil 1 ein Bruteforce-Profil. Da man im Gegensatz zum apart-Paradigma nur noch ein Objekt pro Koordinate speichert und weiterhin

acht Byte zur Verfügung hat, können in der Längengradkoordinate, je nach Vorkommateil, Dezimalgrade mit 16 bis 19 Nachkommastellen gespeichert werden. Die Breitenkoordinate fasst bei acht Byte, je nach Vorkommateil, Dezimalgrade mit 17 bis 19 Nachkommastellen. Die Speicherplatzbreiten werden in Tabelle 4.2 nochmals aufgezeigt. Eine Einschränkung bedeutet das nicht. Diese Speicherform, angewandt auf die Datenbasis, bot keinerlei Einschränkungen bzgl. der Speicherung von GPS-Punkten. Der Vorteil zum ersten Profil des apart-Paradigmas ist eine Speicherplatzeinsparung von vier Byte pro gespeichertem GPS-Punkt. Das heißt, dass der kodierte Bytestrom des ersten Profils etwa 20 % kleiner ist, als der Bytestrom des apart-Paradigmas. In der Auswertung werden die Profile genauer untersucht.

4.3.2 Profil 2

Reservierte Speicherplatzbreiten für das intuitive zweite Profil werden der Tabelle 4.2 entnommen. Liegt ein GPS-Trace mit einer maximalen Anzahl von mehr als sechs Nachkommastellen vor, so werden acht Byte, sonst vier Byte Speicherplatz für GPS-Punkte reserviert. Im Gegensatz zum apart-Paradigma muss hierbei schon früher ein Wechsel der Speicherplatzbreite stattfinden, da Vor- und Nachkommateil verschmolzen wurden. Die Halbierung der Speicherplatzbreite mit wachsendem Differenzvektorgrad geschieht analog zum apart-Paradigma. Die Auswirkung auf den kodierten Bytestrom wird im nächsten Kapitel aufgezeigt.

4.3.3 Profil 3

Das analytische Profil berechnet, ähnlich zum analytischen Profil des apart-Paradigmas, zunächst dynamisch zur Laufzeit die Speicherplatzbreiten für die einzelnen Kodierungsstufen. Um die Unterschiede der Paradigmen darzustellen, wird nun anhand desselben Beispiels die Berechnung der Speicherplatzbreiten erläutert. Abbildung 4.4(b) zeigt eine kumulative Datenverteilung aller Längengrad-Koordinaten des GPS-Traces. Da die Werte im united-Paradigma ebenfalls den Vorkommateil der GPS-Koordinaten enthalten, fallen diese Werte sehr viel größer aus. Zur Erinnerung: im apart-Paradigma wurde lediglich

die Verteilungsfunktion über die Nachkommastellen gebildet. Für dieses Beispiel würde man vier Byte Speicherplatz wählen, da durch diesen alle Längengrad-Koordinaten abbildbar sind.

Die kumulative Verteilungen desselben GPS-Traces für die zugehörigen möglichen Differenzvektoren ersten, sowie zweiten Grades stimmen mit der des apart-Paradigmas überein, weil sich bei einer hohen Abtastrate die GPS-Punkte nicht erheblich verändern. Insbesondere werden keine großen Sprünge im Vorkommateil der Dezimalgrade vernommen. Die Berechnung des Nachkommateils eines Δ_1-/Δ_2-Vektors erfolgt, wie im Kapitel 4.2.1 beschrieben, oftmals über Vorkommagrenzen hinweg.

Kapitel 5

Ergebnisanalyse

In diesem Kapitel wird der nach Kapitel 4 erzeugte und mit den verlustfreien Kompressionsprogrammen compress, BZip2, GZip, 7-Zip und der arithmetischen Kodierung nach Bob Carpenter komprimierte Bytestrom analysiert. Hierzu werden die erreichten Kompressionsraten der Kompressionsprogramme anhand mehrerer Kriterien verglichen.

5.1 Datenbasis und Methodik

Die Datenbasis umfasst eine Sammlung von 7209 GPS-Traces, davon beinhalten 4946 Autobahnfahrten und die restlichen 2263 Stadtfahrten. Das Abtastintervall lag in allen Fällen bei einer Sekunde. Nachdem jeder GPS-Trace mit verschiedenen Profileinstellungen kodiert wurde, wurden diese Ergebnisse wiederum mit herkömmlichen Kompressionsverfahren komprimiert.

Die wichtigste Metrik, die uns an dieser Stelle interessiert, ist die Kompressionsrate, also das Größenverhältnis des Komprimats gegenüber einem Referenzwert. Der Referenzwert ist die Anzahl an Bytes, die benötigt wird, um den GPS-Trace zu kodieren. Berechnet wird dieser Wert anhand des möglichen Wertebereichs des spezifizierten GPS-Traces und deren Anzahl an GPS-Punkten. Sei hierzu $max_{Suf_{lon}}$ die maximale Anzahl an Nachkommastellen der Längenkoordinate und $max_{Suf_{lat}}$ die maximale Anzahl an Nachkomma-

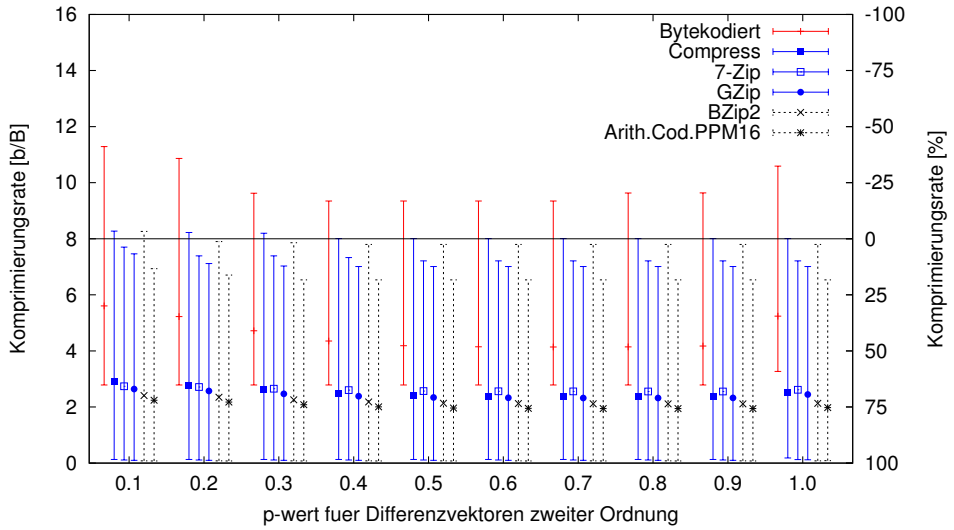


Abbildung 5.1: apart-Paradigma: p-wert für Differenzvektoren erster Ordnung: 1.0

stellen der Breitenkoordinate des Traces. Dann werden die Anzahlen der benötigten Bits pro Längenkoordinate (lon_{bit}) bzw. Breitenkoordinate (lat_{bit}) wie folgt berechnet:

$$lon_{bit} = \lceil \log_2 360 \cdot 10^{maxSuf_{lon}} \rceil$$
$$lat_{bit} = \lceil \log_2 180 \cdot 10^{maxSuf_{lat}} \rceil$$

Die Anzahl benötigter Bytes für den Startzeitpunkt liegt bei acht Byte. Für das Abtastintervall wird ein weiteres Byte benötigt. Somit liegt die Anzahl benötigter Bytes für den Timestamp bei einmalig $time = 9Byte$. Sei außerdem $total_{points}$ die Anzahl an GPS-Punkten des Traces. Schließlich setzt sich der Referenzwert wie folgt zusammen:

$$ref = total_{points} \cdot \frac{lon_{bit} + lat_{bit}}{8} + time$$

5.2 Auswertung

Die Auswertung wird anhand mehrerer Gesichtspunkte durchgeführt. Zunächst wird ein optimales p-Tupel für das dritte Profil bestimmt. Anschließend wird die Kompressions-

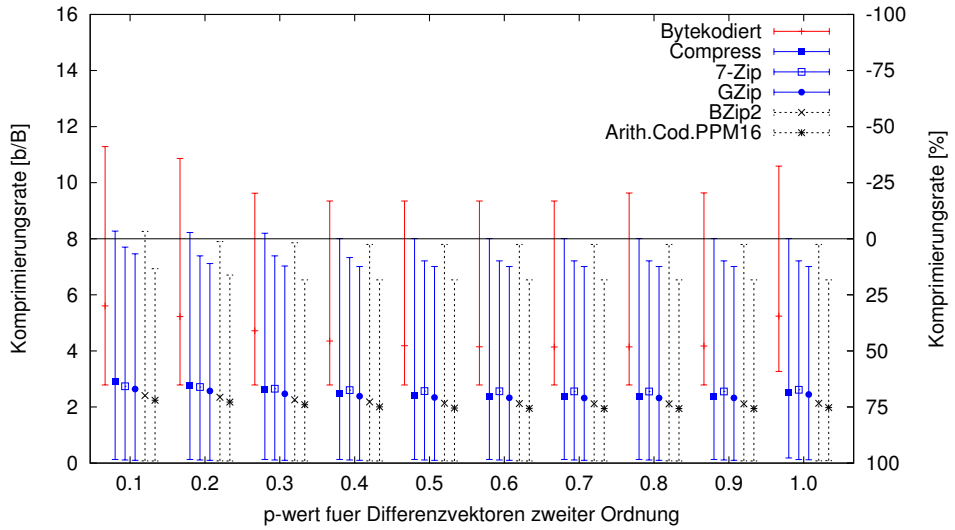


Abbildung 5.2: apart-Paradigma: p-wert für Differenzvektoren erster Ordnung: 1.0

rate der Paradigmen, sowie deren Profile untereinander verglichen. Schließlich wird der Einfluss der Anzahl an Nachkommastellen auf die Kompressionsrate analysiert.

5.2.1 p-Tupel des dritten Profils

Um das optimale p-Tupel der beiden Paradigmen zu bestimmen, wurden insgesamt 100 Kombinationen pro Paradigma getestet. So erfolgte die Auswertung in 0.1-Schritten für p_1, sowie p_2. Nachfolgend werden nur die optimalen p_1-Abbildungen gezeigt. Eine Übersicht über einen breiteren p-Tupel Vergleich bietet Anhang A.1.

apart-Paradigma

Abbildung 5.2 zeigt die Kompressionsraten für $p_1 = 1.0$ und die p_2-Werte, welche auf der x-Achse in 0.1 Schritten abgetragen werden. Zu sehen ist für jeden p_2-Wert eine Gruppe von Kompressionsergebnissen, bei denen das Minimum, Maximum und der Durchschnitt angegeben sind. Die erreichte Kompressionsrate wird der Übersichtlichkeit wegen in Bit pro Byte (y_1-Achse) und in Prozent (y_2-Achse) angegeben. Für kleinere p_1 sind die Kom-

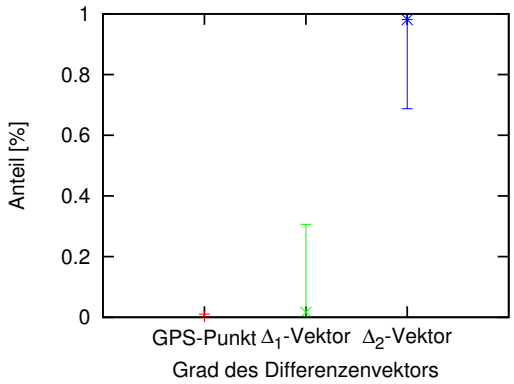


Abbildung 5.3: apart-Paradigma, Profil 3 (1.0;0.8): Verteilung der Kodierungsstufen.

pressionsraten durchweg schlechter, weil unter anderen bloße GPS-Punkte schlechter zu komprimieren sind. Die Darstellung mit Δ_1-Vektoren verringert den Informationsgehalt des zu komprimierenden GPS-Traces, was sich positiv auf die Kompressionsrate auswirkt. Auffällig bei der Bestimmung des optimalen p_2-Werts ist der parabelförmige Verlauf der Durchschnittswerte der Komprimate. Der Tiefpunkt der Parabel liegt bei dem p_2-Wert 0.8. Die beste durchschnittliche Kompressionsrate liegt hier bei etwa 75.7 % für die arithmetische Kodierung mit dem PPM(16)-Modell. Dicht dahinter liegt die BZip2-Komprimierung mit 73.5 %. Die Durchschnittswerte von GZip und compress liegen bei 70.9 % bzw. 70.3 %. 7-Zip bildet das Schlusslicht der Kompressionsverfahren mit 68 %. Der unkomprimierte Bytestrom erreicht im Mittel eine Kompressionsrate von 48.2 %.

Der große Unterschied zwischen dem Minimum und dem Maximum der Kompressionsraten lässt sich damit erklären, dass sich das Bewegungsverhalten des betrachteten Objekts sehr stark auf die Kompressionsraten auswirkt. So werden gleichmäßige Bewegungen relativ effektiv komprimiert. Bei ungleichmäßig beschleunigten Bewegungen hingegen variieren die Δ_2-Vektoren sehr stark, wodurch eine hohe Kompressionsrate mit Hilfe von herkömmlichen Kompressionsverfahren nicht möglich ist. Außerdem spielen weitere Faktoren wie Rauschen bei der Positionsbestimmung oder fehlerbehaftete GPS-Traces eine Rolle. Abbildung 5.3 zeigt den prozentualen Anteil der gebildeten GPS-Punkte, Δ_1- und Δ_2-Vektoren. Wie man sieht werden im Durchschnitt fast ausschließlich Δ_2-Vektoren gespeichert. Da $p_2 = 0.8$ gewählt wurde, gab es in einigen Fällen einen Überlauf bei der Berechnung eines Δ_2-Vektors, so dass stattdessen ein Δ_1-Vektor gespei-

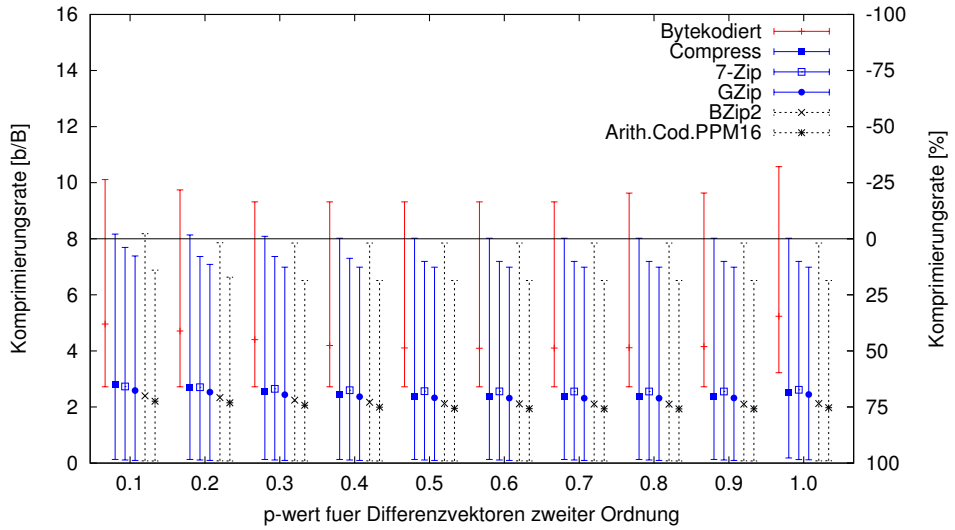


Abbildung 5.4: united-Paradigma: p-wert für Differenzvektoren erster Ordnung: 1.0

chert wurde. Hierdurch lassen sich die Abweichungen bei der Bildung der Δ_1- und Δ_2-Vektoren erklären.

united-Paradigma

Analog zum apart-Paradigma werden in Abbildung 5.4 die Kompressionsraten für p_1= 1.0 gezeigt. Wie auch im apart-Paradigma wirken sich kleinere p_1 schlecht auf die Kompressionsrate aus. Ähnlich zum apart-Paradigma, zeichnet sich eine Parabel über die Durchschnittswerte der Komprimate aus. Der optimale p_2-Wert liegt ebenfalls bei 0.8. Die durchschnittlichen Kompressionsraten verbessern sich im Vergleich zum apart-Paradigma um wenige Zehntelprozentpunkte. So erreicht die arithmetische Kodierung eine durchschnittliche Kompressionsrate von 75.8 % (+0.1), BZip2 73.7 % (+0.2), GZip 71.1 % (+0.2), compress 70.4 % (+0.1), 7-Zip 68.1 % (+0.1) im Vergleich zum unkomprimierten Bytestrom mit 48.6 % (+0.4). Diese Verbesserung ist auf die Einsparung des Speicherplatzes, der im apart-Paradigma für den Vorkommateil der Δ_1-Vektoren anfällt, zurückzuführen. Außerdem kann es sein, dass im apart-Paradigma einige Δ_2-Vektoren nicht gebildet werden konnten, da der Vorkommateil nicht durch das Überlaufverhalten des Nachkommateils abgeleitet werden konnte. Diese Vermutung belegt die Abweichung von +0.0002 bei dem durchschnittlichen Anteil der gebildeten Δ_2-Vektoren im united-

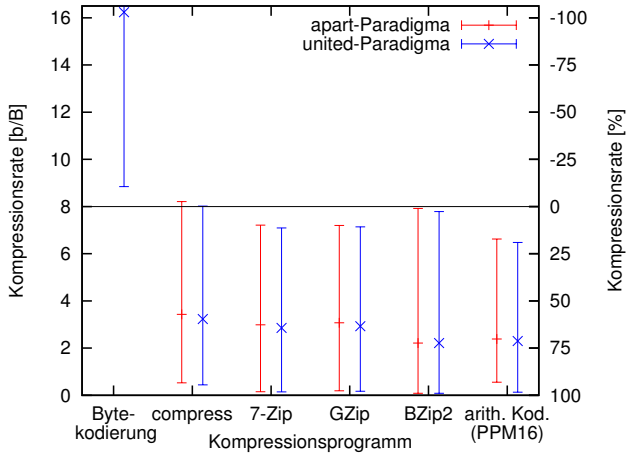


Abbildung 5.5: Paradigmenvergleich: Profil 1

Paradigma. Die Anzahl der gebildeten Differenzvektoren erster und zweiter Ordnung ist, bis auf die oben genannte Abweichung, identisch. Die Verbesserung gegenüber dem apart-Paradigma ist letztlich nicht signifikant. Das united-Paradigma ist jedoch einfacher zu implementieren, da Problematiken, die im apart-Paradigma eine Rolle spielen, nicht betrachtet werden müssen. Aus diesem Grund ist im dritten Profil das united-Paradigma dem apart-Paradigma vorzuziehen.

5.2.2 Paradigmenvergleich

Nachdem ein optimales p-Tupel (1.0;0.8) gefunden wurde, wird profilweise ein Paradigmenvergleich durchgeführt. Gestartet wird mit dem ersten Profil.

Profil 1

Abbildung 5.5 zeigt den Paradigmenvergleich im ersten Profil. Wie bereits in Kapitel 4 erwähnt, erzeugt das erste Profil einen überladenen Bytestrom. Im Falle des apart-Paradigmas wird der Bytestrom sogar durchschnittlich um das 2.5-fache vergrößert und findet deshalb aus Übersichtlichkeitsgründen keinen Platz in der Abbildung. Der Bytestrom des united-Paradigmas wird durchschnittlich um etwa das 2-fache vergrößert. Im

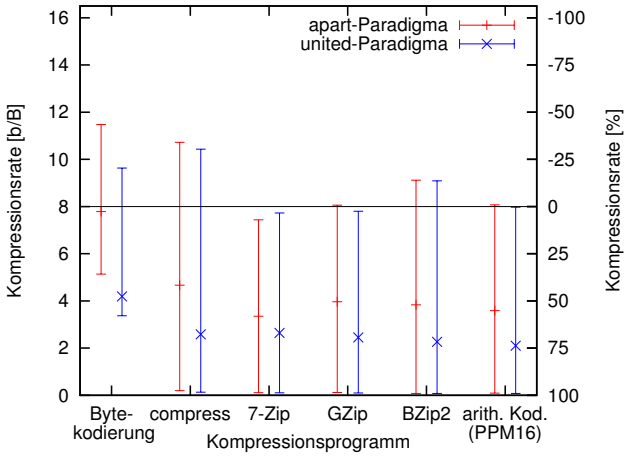
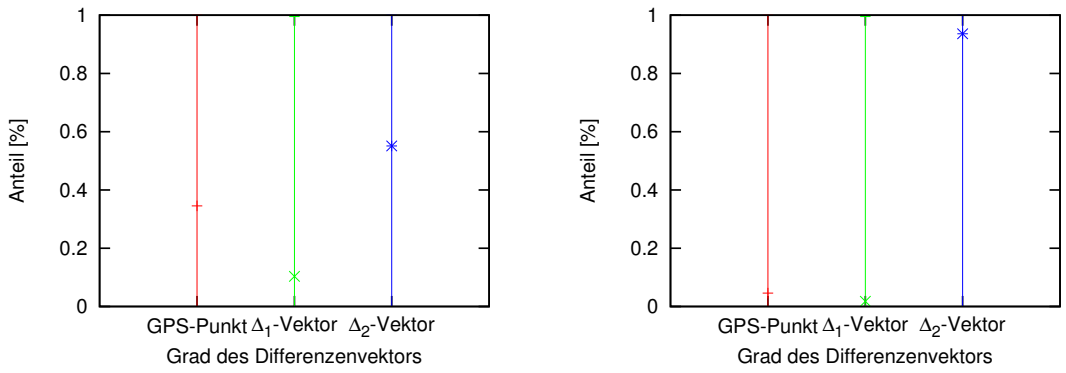


Abbildung 5.6: Paradigmenvergleich: Profil 2

direkten Paradigmenvergleich des ersten Profils ist das united-Paradigma bei fast allen Kompressionsprogrammen besser. Bei den Kompressionsprogrammen comress, GZip, 7-Zip und arithmetische Kodierung (PPM16) liegt das united-Paradigma um 2.5, 1.8, 1.7 bzw. 1.1 Prozentpunkte vor dem apart-Paradigma. Lediglich das Kompressionsprogramm BZip2 komprimierte beide Bytestöme auf Zehntelprozent gleich. Da im Profil 1 in beiden Paradigmen immer nur ein GPS-Punkt, ein Δ_1-Vektor und sonst nur Δ_2-Vektoren gespeichert werden, ist zunächst klar, dass der Bytestrom des apart-Paradigmas größer ist als der des united-Paradigmas (siehe Kapitel 4.3.1). Dies ist auch der Grund für die besseren Kompressionsraten im united-Paradigma. Das Kompressionsprogramm BZip2 komprimiert die im apart-Paradigma oft vorkommenden Null-Bytes des Vorkommateils der Kodierungsstufen sehr effektiv. Diese Beobachtung spiegelt sich in der Kompressionsrate des apart-Paradigmas wider.

Profil 2

Abbildung 5.6 zeigt den Paradigmenvergleich des zweiten Profils. Auffällig sind die großen Unterschiede der durchschnittlichen Kompressionsrate der einzelnen Kompressionsprogramme. Um diese Unterschiede zu verstehen, widmen wir uns zunächst Abbildung 5.7. Dieser Abbildung ist zu entnehmen, dass im united-Paradigma im Durchschnitt sehr viel mehr Δ_2-Vektoren und jeweils weniger GPS-Punkte und Δ_1-Vektoren als



(a) apart-Paradigma (b) united-Paradigma

Abbildung 5.7: Verteilung der Kodierungsstufen des zweiten Profils.

im apart-Paradigma gebildet wurden. Dies hat eine große Auswirkung auf die Kompressionsrate des unkomprimierten Bytestroms. Hier ist im Durchschnitt ein Unterschied von 44.9 Prozentpunkten in Abbildung 5.6 erkennbar. Durch die Kompressionsprogramme verringert sich der Unterschied zwischen den Paradigmen, jedoch ist weiterhin ein Unterschied von durchschnittlich 8.9 (7-Zip) bis zu 26 (compress) Prozentpunkten vorhanden.

Profil 3

Der Paradigmenvergleich des dritten Profils mit dem p-Tupel (1.0;0.8) ist der Abbildung 5.8 zu entnehmen. Die Unterschiede der Paradigmen in diesem Profil sind, wie bereits in Kapitel 5.2.1 beschrieben, marginal und lassen sich auf die Problematik des apart-Paradigmas zurückführen.

5.2.3 Profilvergleich

Nachdem in den Kapiteln 5.2.1 und 5.2.2 das optimale p-Tupel von (1.0;0.8) bzw. das optimale Paradigma, united, gefunden wurde, wird in diesem Kapitel das optimale Profil ermittelt. Der Profilvergleich des apart-Paradigmas befindet sich im Anhang A.2.

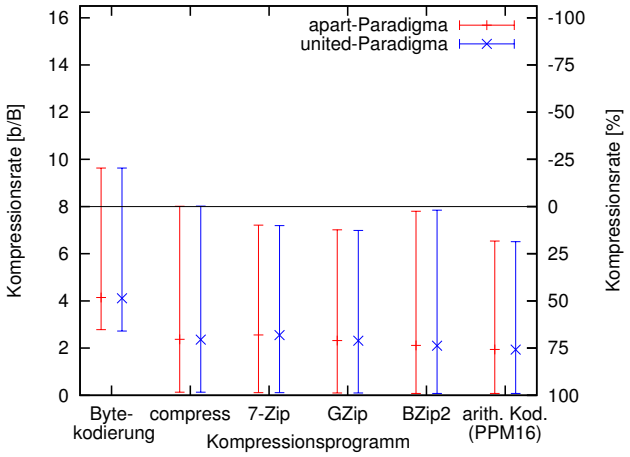


Abbildung 5.8: Paradigmenvergleich: Profil 3, p-Tupel (1.0 ; 0.8)

Abbildung 5.9 zeigt den Profilvergleich des united-Paradigmas. Der unkomprimierte Bytestrom ist, wie bereits erwähnt, im ersten Profil überladen. Das zweite und dritte Profil erreichen eine annähernd gleiche durchschnittliche Kompressionsrate. Der Unterschied liegt hier bei lediglich einem Prozentpunkt. Bei den Kompressionsprogrammen compress, GZip, 7-Zip sowie der arithmetischen Kodierung wird die Kompressionsrate zunehmend von Profil 1 über Profil 2 bis Profil 3 besser. Das Kompressionsprogramm BZip2 hingegen komprimiert den Bytestrom, erzeugt durch das erste Profil, effizienter als den Bytestrom des zweiten Profils. Dies liegt vorallem daran, dass die Bildung von Differenzvektoren den Informationsgehalt (die Entropie) der Daten sinken lässt. Wie in Kapitel 3.1.3 beschrieben, steigt die Kompressionsrate mit sinkender Entropie der Daten. Weiterhin fällt auf, dass das zweite Profil, im Vergleich zu den beiden anderen Profilen, eine schlechte Mindestkompressionsrate erreicht. Auch dies ist anhand der Anteile der Kodierungsstufen zu verstehen, welche im zweiten Profil, im Vergleich zu den anderen beiden Profilen, weniger Differenzvektoren erster und zweiter Ordnung aufweist. Abschließend lässt sich sagen, dass sich der erzeugte Bytestrom des dritten Profils mit den vorgestellten Kompressionsprogrammen am besten komprimieren lässt.

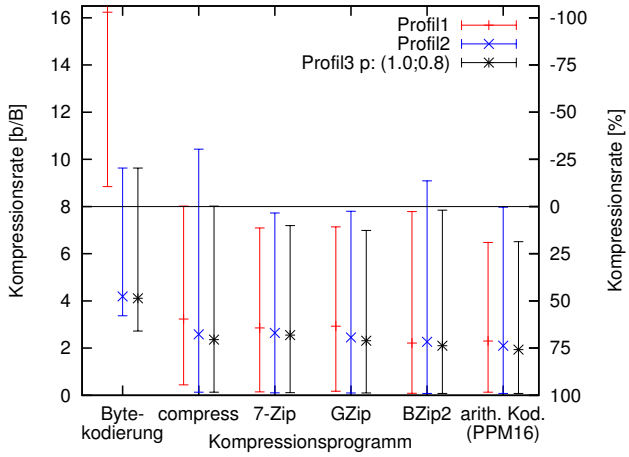


Abbildung 5.9: Profilvergleich: united-Paradigma

Tabelle 5.1: Verteilung der Anzahl an Nachkommastellen.

Länge Dezimalteil	5	6	7	8	9	10	13	15	16	17
Anzahl an Traces	10	3084	394	357	2984	12	13	278	65	12

5.2.4 Nachkommastellenvergleich

In diesem Teil wollen wir verstehen, welche Auswirkung die Anzahl an Nachkommastellen der GPS-Punkte eines GPS-Traces auf die Kompressionsrate haben. Hierfür werden die aus den letzten Abschnitten als optimal befundenen Parameter verwendet. Da es durchaus vorkommen kann, dass in einem GPS-Trace die maximale Anzahl an Nachkommastellen pro Koordinate variiert, verwenden wir immer das Maximum der beiden Koordinaten. Tabelle 5.1 stellt die Verteilung der maximalen Anzahl an Nachkommastellen heraus. Zu beachten ist, dass bei 5, 10, 13, 16 und 17 Nachkommastellen die Auswertung nicht wirklich repräsentativ ist, da es jeweils weniger als 100 Traces gibt; Wir betrachten für diese Auswertung daher ausschließlich Traces mit 6, 7, 8, 9 und 15 Nachkommastellen.

Abbildung 5.10 zeigt die Kompressionsrate, sortiert nach der Anzahl an Nachkommastellen, der Kompressionsprogramme. Die arithmetische Kodierung mit dem PPM(16)-Modell erzielte durchweg die besten Kompressionsraten. Ausschlaggebend sind die durchschnittlich erzielten Kompressionsraten bei 6 und 9 Nachkommastellen. Hier erreicht die

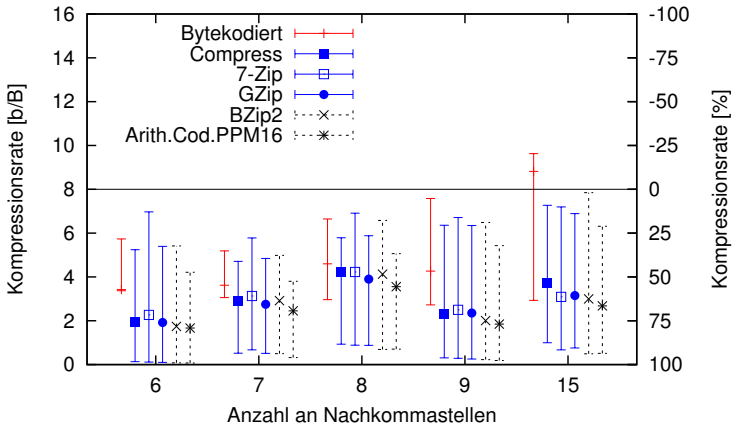


Abbildung 5.10: Nachkommastellenvergleich: united-Paradigma Profil 3, p = (1.0 ; 0.8).

arithmetische Kodierung 79.2 % bzw. 76.9 %. Dicht gefolgt von dem BZip2 Kompressionsprogramm, welches bei 6 bzw. 9 Nachkommastellen eine Kompressionsrate von 78.2 % bzw. 74.9 % erreicht. Compress und GZip erzielen bei 6 bzw. 9 Nachkommastellen in etwa die gleiche Kompressionsrate, wohingegen 7Zip immer die schlechteste Kompressionsrate erzielt.

Wie wir sehen, sorgt die Bildung von Differenzvektoren für eine zusätzliche Verringerung der Entropie der Daten, wodurch Kompressionsverfahren, die eine Entropiekodierung nutzen, insgesamt bessere Kompressionsraten erzielen. Der kodierte Bytestrom scheint bei 8 Nachkommastellen einen Kompressionsverlust zu erleiden. Dies könnte daran liegen, dass im united-Paradigma ein Wechsel auf eine größere Speicherplatzbreite erfolgt, woran die Kompressionsrate leidet. Ein weiterer Wechsel auf eine größere Speicherplatzbreite scheint bei 15 Nachkommastellen zu erfolgen. Der große Sprung könnte eine Vergrößerung der Speicherplatzbreite des ersten Differenzvektors bedeuten. Generell erhöht sich mit steigender Anzahl an Nachkommastellen die Entropie der Daten. Somit rechnet man bei einer steigenden Anzahl an Nachkommastellen mit einer abfallenden Kompressionsrate der Entropiekodierer. Damit lässt sich die schlechtere Kompressionsrate bei fünfzehn Nachkommastellen erklären.

Warum jedoch die Kompressionsrate der Kompressionsprogramme bei 9 bzw. 15 Nachkommastellen besser als bei 7 bzw. 8 Nachkommastellen ist, bleibt ungeklärt. Ein Ansatz wäre, dass der Bytestrom bei 9 bzw. 15 Nachkommastellen vergleichsweise mehr Δ_2-

Vektoren enthält, da die Speicherplatzbreiten für die Differenzvektoren großzügig gewählt werden mussten, um den Anforderungen des p-Tupels gerecht zu werden. Hierdurch verschlechtert sich die Kompressionsrate des Bytekodierers, jedoch verbessert sich die Kompressionsrate der Kompressionsprogramme durch die geringere Entropie der Daten.

Alles in allem wird die beste Kompressionsrate mit der arithmetische Kodierung, angewandt auf ein Profil 3 (1.0;0.8) Bytestrom des united-Paradigmas erzielt.

Kapitel 6

Fazit

6.1 Ergebnis

In dieser Arbeit wurde der Einsatz von herkömmlichen verlustfreien Kompressionsverfahren auf Fahrzeugbewegungsdaten untersucht. Hierzu wurde zunächst ein verlustfreier Bytekodierer auf Basis von Differenzvektoren implementiert, um anschließend eine Auswahl an verlustfreien Kompressionsprogrammen auf den erzeugten Bytestrom anzuwenden. Die Laufzeit des Bytekodierers ist $O(n)$, wobei n die Anzahl an GPS-Punkten im GPS-Trace ist. Es wurden zwei Varianten zur Speicherung von GPS-Punkten implementiert. Die apart-Variante speichert den Vorkomma und Nachkommateil der Dezimalgrade getrennt und die united-Variante verschiebt lediglich das Komma der Dezimalgrade, so dass eine ganze Zahl gespeichert werden kann. Bie jeder Speicherungsvariante spiegelten drei Profile unterschiedliche Strategien zur Generierung von Differenzvektoren erster bzw. zweiter Ordnung wider. Ein Bruteforce-Profil, welches maximale Speicherplatzbreiten reserviert und bis auf den initialen GPS-Punkt und einen Differenzvektor erster Ordnung nur Differenzvektoren zweiter Ordnung speichert. Ein intuitives Profil, welches mit zunehmenden Differenzvektorgrad die Speicherplatzbreite halbiert und zuletzt ein analytisches Profil, welches zunächst den GPS-Trace scannt und anhand von gegebenen Prozentsätzen Speicherplatzbreiten für Differenzvektoren erster bzw. zweiter Ordnung reserviert.
Die Auswertung des optimalen p-Tupels für das dritte Profil ergab (1.0;0.8). Dies spiegelt

durchaus die Erwartung wider: Durch die konsequente Speicherung von Differenzvektoren ersten Grades wird der Informationsgehalt der Daten verringert. Außerdem ist es sinnvoll lediglich 80 % der möglichen Differenzvektoren zweiten Grades zu bilden, denn diese können unter Umständen vergleichsweise groß werden.
Das zweite Auswertungskriterium ergab, dass die Kompressionsraten des united-Paradigmas besser als die des apart-Paradigmas ausfielen. Dies lässt sich durch die Trennung des semantisch zusammenhängenden Vor- und Nachkommateil und den resultierenden Problematiken erklären. Da das Abtastintervall in der Datenbasis hinreichend klein war, ist die Speicherung nach dem united-Paradigma sinnvoll.
Beim Profilvergleich stellte sich wie erwartet heraus, dass das dritte Profil die besten Kompressionsraten erreicht. Zuletzt wurde der Einfluss der Anzahl an Nachkommastellen im GPS-Trace auf die Kompressionsrate analysiert. Es überzeugen die Entropiekodierer BZip2 und die arithmetische Kodierung. Einige Leistungseinbrüche wurden bei einer Vergrößerung der Speicherplatzbreite für eine der Kodierungsstufen verzeichnet. Bei den aussagekräftigen GPS-Traces mit 6 bzw. 9 Nachkommastellen erzielte der arithmetische Kodierer nach Bob Carpenter mit einem PPM(16)-Modell eine Kompressionsrate von 79.2 % bzw. 76.9 %. Vergleicht man die erzielte Kompressionsrate mit einem vorhandenen verlustbehafteten Kompressionsverfahren, wie in [KM11] vorgestellt, so werden in etwa die gleichen Kompressionsraten bei einen maximalem Fehler von fünf cm erreicht.

6.2 Ausblick

Zunächst könnte man die bessere Kompression bei 9 bzw. 15 Nachkommastellen genauer untersuchen. Dieses Artefakt konnte nicht abschließend geklärt werden. Weiter könnte man den Einfluss eines weiteren Differenzvektorgrades auf die Kompressionsrate untersuchen. Möglich wäre eine generelle Verbesserung der Kompressionsrate. Außerdem wäre eine eigene Implementierung eines arithmetischen Kodierers denkbar. Durch das erlangte Wissen über den Informationsgehalt eines GPS-Traces könnte man ein Wahrscheinlichkeitsmodell aufstellen, welches die Abweichung gegenüber dem erwarteten Datum misst. Dieses Modell könnte gegenüber dem PPM-Modell eine bessere Komprimierung erreichen.

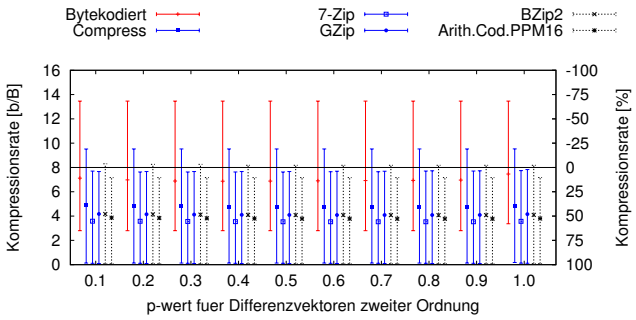
Anhang A

A.1 p-Wert Vergleich

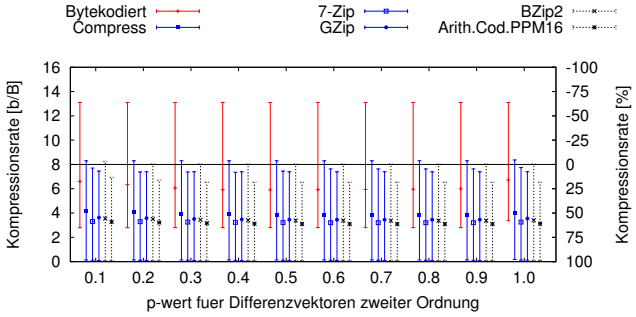
Die Abbildungen A.1 und A.2 zeigen den p-Wert Vergleich des apart- bzw. united-Paradigmas für die p_1-Werte 0.2, 0.4, 0.6 und 0.8. Wie bereits in Kapitel 5.2.1 beschrieben, zeichnet sich hier ein klarer Trend aus. Mit wachsendem p_1-Wert verbessert sich die Kompressionsrate der Kompressionsprogramme.

A.2 Profilvergleich

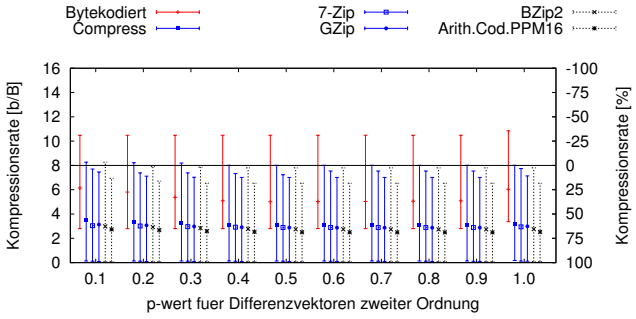
Abbildung A.3 zeigt den Profilvergleich des apart-Paradigmas. Anders als beim Profilvergleich des united-Paradigmas ist hier das erste Profil bis auf die Bytekodierung besser als das zweite Profil. Dies kann anhand der Abbildung 5.7(a) erklärt werden. Das zweite Profil des apart-Paradigmas speichert gegenüber dem ersten Profil des apart-Paradigmas sehr viel weniger Δ_2-Vektoren. Es gibt sogar Fälle, wo nur GPS-Punkte gespeichert werden, was sich negativ auf die Kompressionsrate auswirkt.



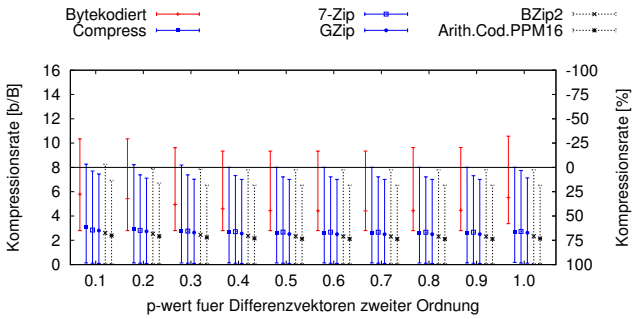
(a) p-wert für Δ_1-Vektoren: 0.2



(b) p-wert für Δ_1-Vektoren: 0.4

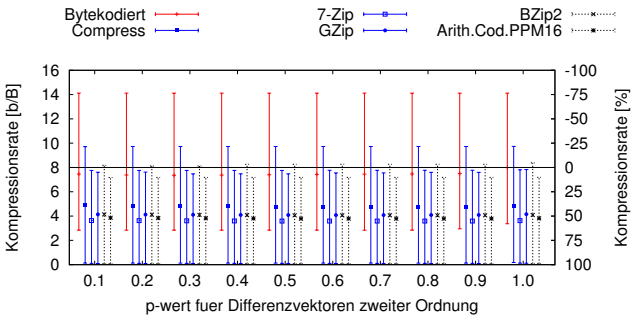


(c) p-wert für Δ_1-Vektoren: 0.6

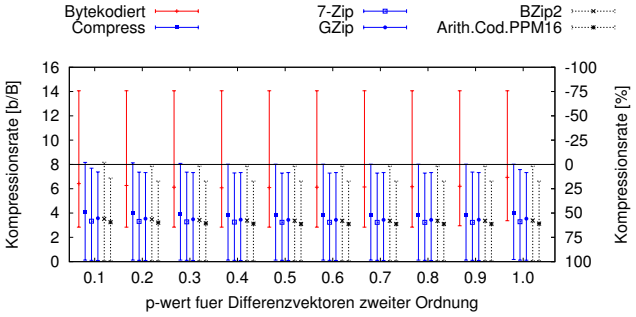


(d) p-wert für Δ_1-Vektoren: 0.8

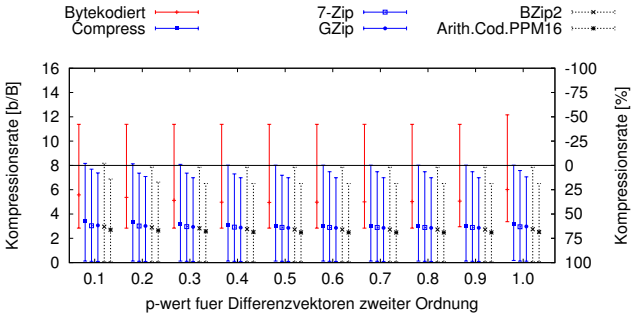
Abbildung A.1: p-Wert Vergleich: apart-Paradigma



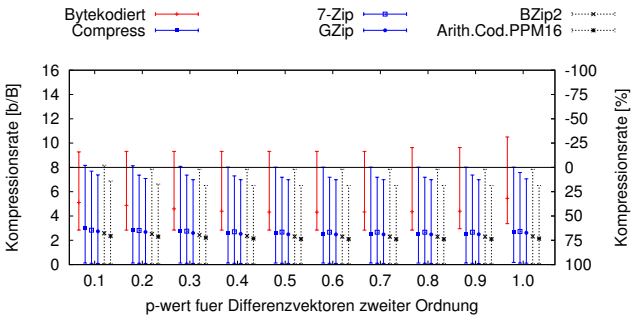
(a) p-wert für Δ_1-Vektoren: 0.2



(b) p-wert für Δ_1-Vektoren: 0.4



(c) p-wert für Δ_1-Vektoren: 0.6



(d) p-wert für Δ_1-Vektoren: 0.8

Abbildung A.2: p-Wert Vergleich: united-Paradigma

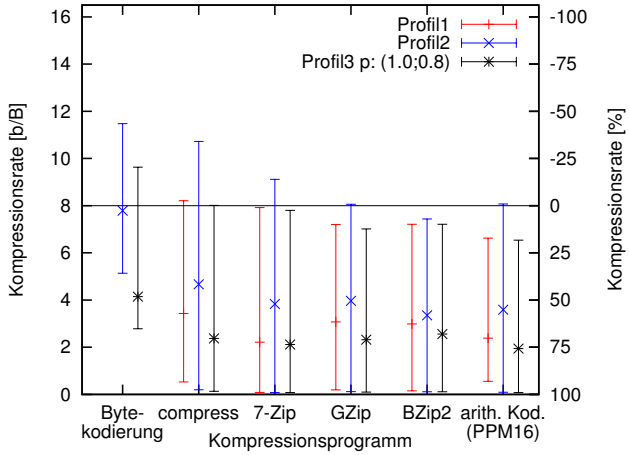


Abbildung A.3: Profilvergleich: apart-Paradigma

Literaturverzeichnis

[BLP10] BARAN, Ilya; LEHTINEN, Jaakko; POPOVIC, Jovan: Sketching Clothoid Splines Using Shortest Paths. In: *Computer Graphics Forum* (2010), S. 655–664.

[BW94] BURROWS, M.; WHEELER, D. J.: A block-sorting lossless data compression algorithm. 1994. Forschungsbericht.

[Cap02] CAPENTER, Bob: *ArithCode Project: Compression via Arithmetic Coding in Java. Version 1.1*. online resource, 2002. http://www.colloquial.com/ArithmeticCoding/

[CJP05] CIVILIS, Alminas; JENSEN, Christian S.; PAKALNIS, Stardas: Techniques for Efficient Road-Network-Based Tracking of Moving Objects. In: *IEEE Transactions on Knowledge and Data Engineering* 17 (2005), Mai, Nr. 5, S. 698–712.

[Deu96a] DEUTSCH, P.: *DEFLATE Compressed Data Format Specification version 1.3*. RFC 1951 (Informational). `http://www.ietf.org/rfc/rfc1951.txt`. Version: Mai 1996 (Request for Comments).

[Deu96b] DEUTSCH, P.: *GZIP file format specification version 4.3*. RFC 1952 (Informational). `http://www.ietf.org/rfc/rfc1952.txt`. Version: Mai 1996 (Request for Comments).

[Fox98] FOX, Dieter: *Markov Localization: A Probabilistic Framework for Mobile Robot Localization and Navigation*. Germany, University of Bonn, Diss.,

1998

[Huf52] HUFFMAN, David: A Method for the Construction of Minimum-Redundancy Codes. In: *Proceedings of the IRE* 40 (1952), September, Nr. 9, 1098–1101. `http://dx.doi.org/10.1109/JRPROC.1952.273898`. DOI 10.1109/JRPROC.1952.273898. ISSN 0096–8390.

[KBMS11] KOEGEL, Markus; BASELT, Daniel; MAUVE, Martin; SCHEUERMANN, Björn: A Comparison of Vehicular Trajectory Encoding Techniques. In: *MedHocNet '11: Proceedings of the 10th Annual Mediterranean Ad Hoc Networking Workshop*, Sicily, Italy, 2011.

[KKKM10] KOEGEL, Markus; KIESS, Wolfgang; KERPER, Markus; MAUVE, Martin: Compact Vehicular Trajectory Encoding (extended version) / Computer Science Department, Heinrich Heine University, Düsseldorf, Germany. 2010 (TR-2010-002). Forschungsbericht.

[KM11] KOEGEL, Markus; MAUVE, Martin: On the Spatio-Temporal Information Content and Arithmetic Coding of Discrete Trajectories. In: *MobiQuitous '11: Proceedings of the 8th Annual International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*, Copenhagen, Denmark, 2011.

[KP08] KRANKE, Florian; POPPE, Holger: Traffic Guard - Merging Sensor Data and C2I/C2C Information for proactive, Congestion avoiding Driver Assistance Systems. In: *FISITA '08: World Automotive Congress of the Int'l Federation of Automotive Engineering Societies*, Munich, Germany, 2008.

[LFDR09] LANGE, Ralph; FARRELL, Tobias; DÜRR, Frank; ROTHERMEL, Kurt: Remote Real-Time Trajectory Simplification. In: *PerCom '09: Proceedings of the 7th IEEE International Conference on Pervasive Computing and Communications*, Galveston, TX, USA, 2009, S. 184–193.

[LR01] LEONHARDI, Alexander; ROTHERMEL, Kurt: A Comparison of Protocols for Updating Location Information. In: *Cluster Computing: The Journal*

of Networks, Software Tools and Applications 4 (2001), Oktober, Nr. 4, S. 355–367.

[Mac02] MACKAY, David J. C.: *Information Theory, Inference & Learning Algorithms*. New York, NY, USA : Cambridge University Press, 2002

[Mar79] MARTIN, G. N. N.: Range encoding: An algorithm for removing redundancy from a digitised message. (1979).

[NR07] NI, Jinfeng; RAVISHANKAR, Chinya V.: Indexing Spatio-Temporal Trajectories with Efficient Polynomial Approximations. In: *IEEE Transactions on Knowledge and Data Engineering* 19 (2007), Mai, S. 663–678.

[Pav07] PAVLOV, Igor: LZMA SDK (Software Development Kit), http://www.7-zip.org/sdk.html, Juli 2007.

[RBFT99] ROY, Nicholas; BURGARD, Wolfram; FOX, Dieter; THRUN, Sebastian: Coastal Navigation – Mobile Robot Navigation with Uncertainty in Dynamic Environments. In: *ICRA '99: Proceedings of the IEEE Int'l Conference on Robotics and Automation*, Detroit, MI, USA, 1999, S. 35–40.

[SS82] STORER, James A.; SZYMANSKI, Thomas G.: Data compression via textual substitution. In: *J. ACM* 29 (1982), October, 928–951. `http://dx.doi.org/http://doi.acm.org/10.1145/322344.322346`. DOI http://doi.acm.org/10.1145/322344.322346. ISSN 0004–5411.

[TCS+06] TRAJCEVSKI, Goce; CAO, Hu; SCHEUERMANN, Peter; WOLFSON, Ouri; VACCARO, Dennis: Online Data Reduction and the Quality of History in Moving Objects Databases. In: *MobiDE '06: Proceedings of the 5th ACM International Workshop on Data Engineering for Wireless and Mobile Access*, Chicago, IL, USA, 2006.

[Wel84] WELCH, Terry A.: A Technique for High-Performance Data Compression. In: *IEEE Computer* 17 (1984), Nr. 6, S. 8–19.

[ZL77] ZIV, Jacob; LEMPEL, Abraham: A universal algorithm for sequential data compression. In: *IEEE TRANSACTIONS ON INFORMATION THEORY* 23 (1977), Nr. 3, S. 337–343.

[ZL78] ZIV, Jacob; LEMPEL, Abraham: *Compression of Individual Sequences via Variable-Rate Coding*. 1978.

Ehrenwörtliche Erklärung

Hiermit versichere ich, die vorliegende Bachelorarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt zu haben. Alle Stellen, die aus den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Düsseldorf, 30.Januar 2012 Andreas Disterhöft