KOMPRIMERING

Hvordan komprimerer man en kort besked

P1-projekt

af Datalogi grp. B226

Aalborg Universitet
Institut for Datalogi
P1-projekt - 1. semester
Hovedvejleder: Simon Laursen
18. december 2012

Institut for Datalogi
Aalborg Universitet

1. Semester

TITEL:
KOMPRIMERING - Hvordan komprimerer man en kort tekst

PROJEKTPERIODE:
P1, 8. okt - 19. dec, 2012

PROJEKT GRUPPE:
B226

GRUPPEMEDLEMMER:
Anders Hermansen
Isabella Kaufmann
Joachim Klokkervoll
Lynge Poulsgaard
Mike Pedersen
Samuel Nygaard Pedersen
Søren Moss Nielsen

VEJLEDERE:
Simon Laursen
Amanda Hill

ANTAL KOPIER: 11

SAMLET SIDEANTAL: 83

ANTAL CD'ER VEDLAGT: 1

SYNOPSIS:

Dette projekt omhandler komprimering af data, mere præcist tekst komprimering. Den kontekstuelle del af rapporten er baseret på SMS-teknologiens udfordringer både i det allerede etablerede marked og i u-landene. Rapporten indeholder informationer vedr. emnet tekstkomprimering på et basalt niveau. Der beskrives bla. eksisterende tekst komprimeringsalgoritmer og redegøres for SMS-teknologiens udfordringer og dens interessenter. Som en del af dette projekt er der udviklet et program, baseret på en komprimerings algoritme, der, som prototype, kan forbedre SMS-teknologien. Programmet testes med flere forskellige former for tekst for, at se effektiviteten af programmet.

Aalborg, 18. december 2012

Anders Hermansen
studienr.: 20124269
mail: aherm12@student.aau.dk

Isabella Kaufmann
studienr.: 20125900
mail: ikaufm12@student.aau.dk

Joachim Klokkervoll
studienr.: 20124277
mail: jklokk12@student.aau.dk

Lynge Poulsgaard
studienr.: 20124272
mail: lkpo12@student.aau.dk

Mike Pedersen
studienr.: 20124283
mail: mipede12@student.aau.dk

Samuel Nygaard Pedersen
studienr.: 20124278
mail: snpe12@student.aau.dk

Søren Moss Nielsen
studienr.: 20124275
mail: smni12@student.aau.dk

Forord

Denne rapport er skrevet som et P1 projekt i Datalogi på Aalborg Universitet, og målgruppen er derved også datalogi-studerende på første semester. I denne rapport er afsnittene inddelt på en sådan måde, at de i den valgte rækkefølge er nemmere at forstå og bygger op omkring hinanden, end hvis de læses individuelt. Indledningen kan desuden læses for sig selv, for at få et hurtigt overblik over projektet og de overvejelser, vi er kommet frem til i udarbejdelsen. I rapporten er der desuden brugt fodnoter og kildehenvisninger. Kildehenvisninger er markeret ved brug af tal, som f.eks. "[1]" eller "Shannon [3]", der henviser til kilderne i listen bagerst i rapporten. Fodnoter er markeret med et tal i superscript, som f.eks. "udtryk[1]", som henviser til fodnoten nederst på den side, hvor den er benyttet. Fodnoter er kun brugt i forbindelse med ordforklaringer. Denne rapport er udarbejdet af Anders Hermansen, Isabella Kaufmann, Joachim Klokkervoll, Lynge Kærlund Poulsgaard, Mike Pedersen, Samuel Nygaard og Søren Moss Nielsen, under vejledning af Simon Laursen og Amanda Hill ved det Tekniske- Naturvidenskabelige Fakultet, Aalborg Universitet.

Indhold

Kapitel 1

Indledning

Kommunikation er væsentligt for menneskets eksistens, og foregår ikke kun verbalt, men også nonverbalt, f.eks. via SMS eller gennem tekst over internettet. Kommunikation over internettet bliver mere og mere effektiv, og har potentiale til at udkonkurrere eksisterende non-verbale kommunikationsmetoder, som f.eks. SMS.

SMS har visse restriktioner; bl.a. maksimal tekstlængde, prisen pr. besked og krav om mobildækning [1]. SMS lever dog, i bedste velgående, i den fattige del af Afrika (de såkaldte U-lande) pga. manglende internet-opkobling [2].

Dette projekt har til hensigt at undersøge mulighederne for, at komprimere SMS-beskeder, og belyse nogen af de teknologiske udfordringer, der findes ved elektronisk kommunikation. Hertil undersøges der bl.a. hvilke eksisterende komprimeringsmetoder og muligheder for komprimering af SMS'er, der findes.

I dette projekt bliver det beskrevet, hvordan den datalogiske forståelse af komprimering kan sammenkobles med SMS, som kommunikationsmiddel. Ydermere uddybes en generel forståelse af komprimering, herunder også hvilke metoder og algoritmer, der anvendes til komprimering. Som afslutning på projektet er der udviklet en programprototype baseret på en komprimeringsalgoritme, der skal kunne vise princippet bag at øge loftet for tegnbegrænsningen i SMS'er.

Formålet med dette projekt er overordnet, at forbedre SMS-teknologiens forudsætninger vha. komprimering og dermed muliggøre en forlængelse af

SMS-teknologiens levetid. For at nå dette mål, udarbejdes en problemanalyse hvor informationsteori, SMS-teknologien og interessenter bliver beskrevet. Der er også præsenteret en problemformulering med fokus på et program, der kan komprimere kort tekst.

Dertil er der i afsnittet med implementering beskrevet hvilke typer af eksisterende algoritmer, der forefindes og herudfra, hvilken algoritme-type løsningen benytter sig af. Ydermere præsenteres og testes løsningen, og projektet afrundes i konklusionen.

Del I

Problemanalyse

Kapitel 2

Informationsteori

Informationsteori omhandler grundlæggende opfattelse og formidling af data. Den datalogiske opfattelse forstås, som mængden af bits og bytes, der kan opbevares, transformeres, krypteres, komprimeres, dekomprimeres og transmitteres. Den datalogiske opfattelse er ikke nødvendigvis afhængig af den menneskelige opfattelse og formuleringsevne, fordi den hovedsageligt er baseret på teknologisk forståelse og udvikling [3].

I den datalogiske forståelse af Informationsteori arbejdes der ud fra en matematisk teori, som omhandler overførslen af information i tekniske systemer. Denne er blandt andet blevet udarbejdet af Shannon, der fokuseres på begrænsningerne, ved overførsel og komprimering af data. Informationsteori har forskellige anvendelser og bliver blandt andet anvendt i kildekodning.

Kodning kan deles op i to; kildekodning, der beskæftiger sig med komprimering af information, og kanalkodning, som forsøger, at beskytte information under overførsel og forhindre fejl og forstyrrelser i dataene. Her vil der dog kun fokuseres på kildekodning, da beskyttelse af information under overførslen ikke har direkte sammenhæng til dette projekt.

Formålet med kildekodning er, helt fundamentalt, at reducere antallet af bits, der repræsenterer en given information. Der findes overordnet to forskellige typer af komprimering, tabsfri komprimering og komprimering med tab. I tabsfri komprimering fjernes de unødvendige gentagelser, og den redundans, meddelelsen indeholder, mindskes. I komprimering med tab fjernes der derimod de dele af meddelelsen, der ikke er vurderet som vigtigt for den essentielle

forståelse. Komprimering gør overførslen af informationen mere effektiv, da det mindsker antallet af bits, som skal sendes, og dermed også den mængde lager der benyttes til, at opbevare informationen. Inden for kildekodning benyttes Shannons kildekodesætning, som beskriver hvor meget en meddelelse kan komprimeres, før oplysninger går tabt.

2.1 Entropi

Entropi er i informationsteori et mål for uforudsigeligheden af data. Hvis der haves en ukendt variabel, X, fra et alfabet, χ, samt sandsynligheden for den enkelte variabel i alfabetet, $p(x)$, kan der udregnes en værdi for den uforudsigelighed, der haves for systemet, nemlig den såkaldte entropi. Entropien er helt basalt den mængde bits, der minimum skal til, for at beskrive en værdi i det undersøgte alfabet.

Entropien, $H(X)$, kan udregnes efter følgende formel [4]:

$$H(X) = -\sum_{i=1}^{n}(p_i * \log_2(p_i)) \tag{2.1}$$

Et eksempel med entropi kunne være, at skulle beskrive resultatet af, at slå plat eller krone. Her findes der en entropi på 1 bit, da der er to udfald med lige stor chance for forekomst. Hvis mønten har plat på begge sider, falder entropien til 0 da alle kast er komplet forudsigelige. Ligeledes vil en "unfair" mønt, som lander oftere på den ene side, have en entropi under 1 bit, men dog over 0, da den ikke er komplet forudsigelig. Entropien behøver dermed ikke at være et heltal af bit.

I forhold til tekstkomprimering er det interessant, at se på entropien af tekst, da data med lav entropi kan komprimeres mere, end data med højere entropi kan. Entropien af en tekst afhænger ikke alene af alfabetet, der bruges til at skrive det, men også tekstens sprog, da forekomsten af de forskellige tegn vil variere. For engelsk er det vist, at entropien ligger imellem 0.6 til 1.5 bit pr. tegn, ifølge forsøg med mennesker udført af Shannon [5]. Dette betyder at engelsk gennemsnitligt kan komprimeres til denne entropi, såfremt at komprimeringen er optimal.

Entropi er også vigtig i forhold til kryptering. I kryptering ønsker man ofte at skabe en høj entropi, selvom den underliggende kilde kode måske kun har lav entropi. Dette betyder blandt andet også, at krypteret data ofte er svært at komprimere, da entropien ofte er blevet langt højere.

2.2 Shannons kildekode sætning

Følgende afsnit er skrevet med udgangspunkt i bogen "Elements of Information Theory" af Thomas M. Cover og Joy A. Thomas [6].

Shannon's kildekode sætning forklarer, hvor komprimerbart et givent sæt af tegn (f.eks. en tekst) er i et givent alfabet med en given mængde af mulige kombinationer (f.eks. ord).

Læresætningen siger følgende:

1: Det er ikke muligt, at lave en indkodet besked, der beskriver en mængde information på mindre bits end beskrevet i entropien uden, at der samtidigt mistes data.

2: Det er dog muligt, at komme meget tæt på det antal bit, som entropien beskriver, hvilket gør, at entropien kan give et godt billede af, hvor komprimerbar en kode kan blive.

Hvis χ_1 og χ_2 hver definerer et givent alfabet, kan den følgende formel bruges til at udregne det antal bits, der skal bruges til, at beskrive en besked indkodet fra χ_1 til χ_2 uden tab af data.
Shannons kildekode sætning 1.

$$H_D(X) \leq L^* < H_D(X) + 1 \tag{2.2}$$

Hvor entropien for den givne sammensætning af ord er $H_D(X)$, som udregnes ved hjælp af ligningen beskrevet i afsnit 2.1. Derudover ses også udtrykket L^*, som beskriver summen af produkterne mellem sandsynligheden og ordenes længde i kodnings-alfabetet ($L^* = \sum(p_i * l_i)$), altså den estimerede størrelse af koden.

For en given mængde sammensætninger af tegn må entropien altså ligge imellem de to værdier givet ved henholdsvis $H_D(X)$ og $H_D(X) + 1$.

Ved at kende antallet af værdier, i, og sandsynligheden for, at hver værdi fremkommer, p_i, kan den mængde bits, som skal bruges for at indkode en tekst uden at miste information udregnes.

2.3 Komprimering

Datakomprimering, også kaldet datakompression, er en computerteknologisk proces, der formindsker en mængde af bits, der bruges til at lagre data. En måde at komprimere på tager udgangspunkt i begrebet redundans, ved at finde gentagende bogstaver, ord eller f.eks. farveværdier på pixels. Selve komprimeringen foregår således ved, at en algoritme finder og udskifter de gentagende dataelementer med en midlertidig mindre værdi, som senere genetableres ved dekomprimeringen. Komprimering kan opdeles i to kategorier, tabsfri komprimering og komprimering med tab.

Tabsfri komprimering

Tabsfri komprimering (engelsk: lossless compression) bevarer alle dataene, så de kan genskabes 100 % som før komprimeringen. Dette bruges ved komprimering af f.eks. sikkerhedskopier og tekst, hvor alle dataene er nødvendige, for at informationen kan bruges.

Komprimering med tab

Komprimering med tab (engelsk; lossy compression) bevarer ikke al data, da der sker et gradvist kvalitetstab undervejs, specielt set ved komprimering af billeder (.JPEG format) og musik (.MP3 format) [7]. Ved denne type komprimering, kan teskt-filer typisk ikke læses, fordi de, som regel, ikke kan dekomprimeres til det oprindelige data pga. datatabet ved komprimeringen [8].

2.3.1 Tekstkomprimering

Ved tekstkomprimering benyttes tabsfri komprimering, fordi teksten skal kunne genskabes 100 % når den dekomprimeres. Når tekst skal komprimeres, benyttes ofte en komprimering-algoritme, som f.eks. ZL77 eller Huffman (se kapitel 7).

2.4 Præfiks-fri Kodning

Følgende afsnit er skrevet med tæt tilknytning til "Codes, trees and the prefix property" af Macphee [9] og "The Data Compression Guide" af Tamayo [10].

Enhver form for datarepræsentation benytter sig af et kodetræ til, at skelne mellem de forskellige tegn, som datasættet består af. Da datamaskiner ikke forstår andet end 0- og 1-taller, er kodetræerne på det laveste niveau bestående af to-kantede træer med 1'ere og 0'ere.

ASCII tegnsættet har kun 128 tegn, som kan repræsenteres. Hver tegn skrives binært med 7 bits, dvs. syv 0- eller 1-taller i rækkefølge. Efter hvert 7. bit ved datamaskinen, som afkoder tegnene, at koden for tegnet er slut, og det næste tegn begynder. En tabel over den udvidede ASCII-tegntabel, hvor hvert tegn er beskrevet med 8 bits, hvilket giver plads til 256 tegn, i binær kode kan findes i kilde [11].

Det kan diskuteres, om det virkelig kræver 7 eller 8 bits, at beskrive hvert tegn. F.eks. skrives tegnet "," (komma), som "00101100". Er det nødvendigt, at have de første nul-taller med? Idéen med, at alle tegn er lige lange, beskrevet binært, var, at man præcist vidste hvor tegnene skulle adskilles fra hinanden. Når man ved hvornår tegn skal adskilles, uden at de nødvendigvis har samme bit-længde, er kilde-koden præfiks-fri, eller har med andre ord præfiks-frie egenskaber.

2.4.1 Præfiks-frie egenskaber

En kode har præfiks-frie egenskaber, når en repræsentation af et tegn, ikke er et præfiks af en repræsentation af et andet tegn. Som eksempel kan følgende binære koder betragtes:

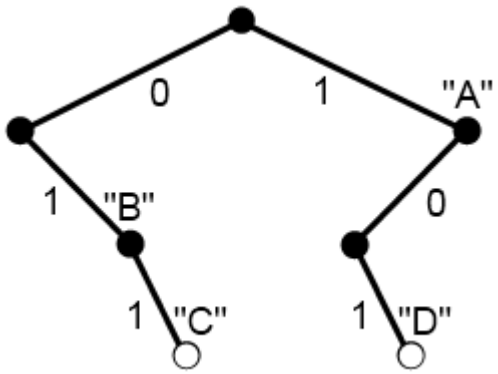
	"A"	"B"	"C"	"D"
Kode 1:	0	11	100	101
Kode 2:	1	01	011	101

Tabel 2.1: Kodesystemer 1 og 2

Ved brug af kode 1, ville beskeden “ABDACB” kunne skrives, som “011101010011”. Da koden for et af tegnene ikke er et præfiks af koden til nogle af de andre tegn, er koden præfiks-fri. Det gør at koden for “ABDACB” er unik. “011101010011” kan altså kun deles op til “0 11 101 0 100 11”, som netop er sekvensen “ABDACB”. Kodes samme besked med kode 2, bliver “ABDACB” til “101101101101”. Betragtes koden nærmere ses det, at koden for “B” er et præfix af koden for “C”. Koden for “A” er også præfix for koden til “D”. Koden er altså ikke unik. “101101101101” kan deles op på mange måder:

“101 101 101 101” : “DDDD”
“1 01 1 01 1 01 1 01”: “ABABABAB”
“1 011 01 1 01 101”: “ACBABD” osv.

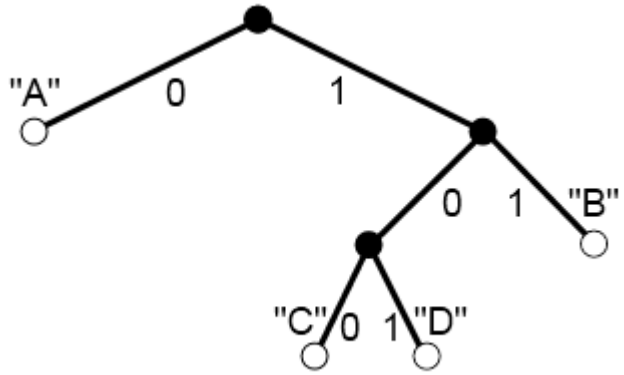
Kode 2 er altså ikke en særlig velegnet metode til, at lagre data på. Hvis et binært træsystem opstilles, kan det visualiseres, hvorfor det går galt. Betragt kode 2’s binære system (figur 2.1).



Figur 2.1: Binært kodetræ for kode 2

De sorte knuder kaldes knuder (interior nodes) og de hvide knuder kaldes blade (leaf nodes). For at en kode kan beskrive data unikt, skal den kun

beskrive tegn med bladene. Kode 2 har tegnene “A” og “B” i to knuder og er derfor ikke præfiks-fri. Funktionen, der koder med kode 2, er ikke injektiv (one-to-one) da data, som kodes, ikke har en entydig afkodning. Kigges der derimod på kodetræet for kode 1 (figur 2.2), ses det, at alle tegnene, som



Figur 2.2: Binært kodetræ for kode 1

koden kan beskrive, ligger i bladene. Der er altså ikke nogle tegn, hvis kode starter med et andet tegns kode. En algoritme, der gør stor brug af denne metode, er Huffman algoritmen, som beskrives nærmere i afsnit 7.3.

Kapitel 3

Short Message Service

SMS er en telekommunikationsmodel opfundet af Friedhelm Hillebrand i 1985. Efter grundige undersøgelser af korte beskeder, konkluderede han, at det ville være tilstrækkeligt for SMS-beskeder at kunne rumme 160 tegn for at formidle de fleste korte beskeder. En SMS-besked kan sende op til 140 byte, som svarer til 160 tegn ved brug af tegnsættet GSM-7, som kan beskrive hvert tegn med 7 bits [1]. Med GSM-7 er det imidlertid ikke muligt at bruge mange specialtegn, som bruges i bl.a. Kyrriliske sprog, Kinesisk o.lign. Det er dog muligt at benytte UCS-2, som understøtter mange flere tegn, men dette begrænser antallet af tegn i en SMS til 70 [1].

Inden for de sidste 10 år har udbredelsen af SMS taget fart, og har udviklet sig til, at være et verdensomspændende marked. I 2011 blev servicen benyttet af 5,9 milliarder brugere [12]. SMS har i dets første år hovedsageligt været anvendt til kommunikation i brugernes sociale netværk, men er i de seneste år også blevet en vigtig del af mange virksomheders marketing [13].

3.1 GSM

GSM er et mobildata transfer system, som har været brugt i Europa siden 1991 [14]. Systemet er fuldt digitaliseret og giver mulighed for både tele- og datakommunikation, baseret på SIM-kortet, der styrer enhedens udveksling med netværket. Den datakommunikation GSM systemet tilbyder har gjort

SMS'er mulige ved at indføre SMSC[1] som lagrer den korte besked og videresender den til modtagerens enhed. Størrelsen af denne besked kan være op til 1120 bit (140 bytes) [1].

3.2 Tegnsæt

I forbindelse med komprimering af korte beskeder er det relevant, at se på hvilke tegnsæt, der eksisterer, og hvilke former for tegnkodning de forskellige sæt benytter. Tegnsættene beskriver hvilke tegn, der kan repræsenteres med dette, mens tegnkodning beskriver, hvordan de enkelte tegn repræsenteres. Det sætter en grænse for hvilke sprog, der er kompatible med koden, idet nogle sprog indeholder specialtegn, og er bestemmende i forhold til, hvordan de forskellige tegn skal håndteres. I dette afsnit beskrives tre relevante tegnsæt, GSM-7, ASCII og Unicode, samt den tegnkodning de benytter.

3.2.1 GSM 7-bit alfabet

SMS beskeder bygger på GSM 7-bit alfabetet [1]. 7-bit alfabetet medfører, at hvert tegn lagres med 7 bit. Når hver besked kan indeholde op til 1120 bit, giver det mulighed for, at skrive 160 tegn i hver besked, skrevet med 7-bit alfabetet. Alfabetet består af 128 tegn fra det oprindelige ASCII. Som en del af komprimeringen indeholder GSM 7-bit en fejlbehæftet indkodning. Nogle græske bogstaver og latinske bogstaver med accenter og tryk, som "$\acute{a}$", "$\ddot{e}$" og "$\hat{u}$", findes ikke i karaktersættet og bliver i stedet konverteret til deres simple latinske modstykke, og kan altså ikke beskrives som en af de 128 tegn, alfabetet indeholder [15].

3.2.2 ASCII

ASCII (American Standard Code for Information Interchange) blev offentliggjort i 1963, som en standard for tegnkodning, baseret på det engelske alfabet. Det blev vurderet, at 7 bits var den mest effektive størrelse til, at repræsentere de forskellige tegn i det engelske alfabet. Dette gav således plads

[1]SMSC: Short Message System Center

til 128 forskellige tegn, og af disse var 33 kontrol tegn, og 95 grafiske tegn [16].

De originale 128 tegn var dog ikke nok til, at akkommodere forskellige sprogbestemte specialtegn, og der begyndte efterhånden, at blive udviklet flere udgaver af ASCII, som tillod disse ekstra tegn. Standarden blev dog stadig kun repræsenteret med 7 bits, og det opstod derfor inkompatibilitet mellem de forskellige udgaver af ASCII. For at undgå denne problemstilling, blev der udviklet et udvidet ASCII alfabet med 256 tegn, der blev repræsenteret i 8 bits. Dette førte stadig til flere forskellige inkompatible udgaver, men problemet var stærkt reduceret [17]. En af de mest populære udgaver af udvidet ASCII er ISO Latin-1. Denne udgave indeholder de forskellige sprogbestemte tegn fra de latinske sprog, og kan derved benyttes til langt de fleste vestlige sprog [18].

3.2.3 Unicode

Karaktersættet Unicode er et karaktersæt udviklet med ambitionen om, at understøtte alle levende sprog i verden og på den måde sikre kompatibilitet verden over [19]. Dette krævede dog mere end de 8 bits, der ellers var standarden inden for tegnkodning, og karaktersættet blev udvidet til, at kunne fylde op til 16 eller 32 bits pr. tegn. I Unicode tildeles hvert tegn et kodepunkt, der er unikt for netop dette tegn. Unicode kan i øjeblikket understøtte 1.114.112 forskellige kodepunkter, men benytter lige nu kun omkring 110.000 af disse [19].

Dette rækker dog også til at repræsentere alle levende, samt de fleste uddøde sprog. Kodepunkterne er desuden kun en repræsentation af, hvordan tegnene bliver fortolket, ikke hvordan de vises eksplicit på skærmen. Ethvert kodepunkt i Unicode består af "U+" efterfulgt af 4 tal og/eller bogstaver, f.eks. repræsenteres tegnet "P" som "U+0050". Kodepunkterne er fordelt over 17 planer, der hver især indeholder 65.536 kodepunkter. Plan 0 kaldes BMP (Basic Multilingual Plane). Her er langt de fleste af de mest udbredte tegn, samt mange special tegn. Det næste plan indeholder blandt andet matematiske symboler, og tegn fra oldtiden, som runer og hieroglyffer.

Unicode har 3 forskellige tegnkodnings-formater, der hovedsagligt benyttes til, at implementere karaktersættet. Det er formaterne UTF-8, UTF-16 og

UTF-32. Alle tegn kan skrives i disse formater, og de er derfor også komplet kompatible med hinanden. UTF-8 er, som beskrevet i underafsnit 3.2.4, kompatibel med ASCII, og er derved også nyttig til meget eksisterende software. I UTF-16 tildeles der 16 bits i en unit til, at beskrive et kodepunkt. Hvis et tegn har behov for en længere beskrivelse, kan dette klares ved, at bruge flere af disse 16 bit enheder. Ligeledes gives der, i UTF-32, 32 bits til at beskrive et tegn. Dette er nyttigt, når plads ikke er en bekymring, men hvor det derimod er adgang til tegnene, som en enkelt enhed, der er eftertragtet [19].

3.2.4 UTF-8

UTF-8 er den mest brugte metode til at indkode tegn fra UCS(Universal Character Set) karaktersættet. UTF-8 har den fordel, at den beholder de gamle US-ASCII karakterværdier. Dette betyder, at den er bagud kompatibelt med ældre software, der afhænger af ASCII-tabellen, så længe der kun benyttes tegn, som eksisterer i ASCII-tabellen [19].

UTF-8 indkodning beskriver tegn med en eller flere bytes (8-bits, heraf navnet UTF-8). Alle tegnene fra ASCII-tabellen kan beskrives, som en enkelt byte med de samme kodepunkter (positioner i tabellen), som i ASCII. De andre tegn fra Unicode beskrives af et varierende antal bytes, hvor antallet og værdien af bytene bestemmes af tegnets kodepunkt i Unicode-tabellen [20].

I et tegn, der kan beskrives, som en enkelt byte, er den ledende indgang sat til 0, mens de sidste 7 bits bruges til, at indkode tegnet. I tegn, der beskrives af n bytes, er de første n indgange af den første byte sat til 1, efterfulgt af et 0. De sidste bits bruges til indkodningen. De efterfølgende bytes i sekvensen har alle den ledende bit sat til 1, efterfulgt af et 0. Resten bruges til indkodningen.

F.eks: Et tegn, der beskrives med én byte ser således ud: 0xxxxxxx, og et tegn, der beskrives med to bytes ser således ud: 110xxxxx 10xxxxxx, hvor de bit-pladser markeret med x beskriver tegnet [20].

De første 128 tegn (ASCII tegnsættet) beskrives af en enkelt byte, og de næste 1920 tegn beskrives af to bytes. Dette dækker blandt andet alle latin-afledte sprog samt en del andre sprog. Tre bytes bruges til, at beskrive resten af det såkaldte "Basic Multilingual Plane", dvs. de fleste japanske, kinesiske og

andre tegn. De sidste planer med tegn, der ikke bruges så ofte, beskrives med fire bytes. Der bruges ikke flere end fire bytes til UTF-8 indkodning [19].

3.2.5 8-bit data kodning

I GSM 03.38-standarden findes desuden endnu en form for tegn-indkodning. Det er muligt at sende den "rå" binære kode direkte via det mobile netværk. Hertil kommer, at det er nødvendigt, at bruge et brugerdefineret tegnsæt til at indkode og afkode beskeden.

Beskeder indkodet som 8-bit data kan, i en enkelt SMS, indeholde op til 140 bytes, eller svarende til 1120 bits [1]. Der kan altså sendes den samme mængde data, som med de andre tegnsæt, her er tegnene dog ikke defineret efter noget fast bibliotek,og datatypen skal i stedet være af en kendt art på både modtagerens og afsenderens side for, at kunne oversættes korrekt.

Kapitel 4

Interessenter

I mange forskellige situationer kan data, med fordel, komprimeres. Dette er primært for enten, at spare plads, når dataene skal opbevares, eller for at spare båndbredde, når dataene skal overføres.

Dette betyder, at komprimering er relevant i næsten alle former for telekommunikation. Der findes dog situationer, hvor fordelene ved komprimering er så små, at de ikke kan retfærdiggøre omkostningerne i forøget kompleksitet. Et eksempel på dette, er situationer, hvor rigeligt båndbredde er tilgængeligt, men kun små mængder data skal sendes.

Et muligt område, hvor komprimering af korte beskeder kunne være relevant, er i mobilnetværk. I mobilnetværk håndterer hver radiomast flere tusinde mobilenheder. De fleste af disse mobilenheder sender kun mindre beskeder. Her ville komprimering betyde, at hver mast vil kunne understøtte flere brugere. Da det ofte er svært, at retfærdiggøre brugen af komprimering af korte beskeder, i kraft af at der ikke spares særligt meget opbevaringsplads pr. besked, kan enheder, der skal håndtere store antal beskeder, med fordel betragtes, fremfor enheder der kun skal håndtere én besked af gangen.

Et andet område relevant til komprimering er, at omgå tekniske begrænsninger som f.eks tekstlængde-begrænsningen i SMS.

4.1 Brugeren

Det er interessant at se på hvilke fordele brugeren af SMS-servicen vil kunne drage af komprimeringsløsningen til korte beskeder, da det hovedsagligt er her forskellen vil kunne mærkes. Her er det vigtigt at medtage hvilke problemer brugeren kan have med det nuværende system og hvordan denne løsning vil kunne afhjælpe problemet.

Da nogle teleselskaber tager et fast beløb pr. SMS, ville det være attraktivt for brugeren af SMS-servicen, at kunne skrive længere beskeder ved, at komprimere den sendte tekststreng. Bliver en SMS for lang (over 1120 bits), bliver den delt op i flere SMS'er, som brugeren, som regel, betaler for, som individuelle SMS'er. Derudover bliver der også reserveret plads til en SMS-header, som skal holde styr på rækkefølgen af de sendte SMS'er, hvilket gør den reelle plads til selve teksten i SMS'erne mindre.

Ved brug af komprimering, kan tekststrenge reduceres, så de fylder mindre. Dog ville dette kun kunne lade sig gøre med SMS, ved at implementere en fælles metode, da en afkodnings-tabel sendt sammen med den sendte SMS ville opveje den fordel, som selve komprimeringen af teksten ville give. Komprimeringen ville da sørge for, at brugeren ville kunne få flere tegn med i sin SMS-besked, og det ville i nogle tilfælde kunne spare brugeren for, at skulle betale for to SMS'er for at sende beskeden.

4.2 Udbyderen

Udbyderen af SMS-servicen har en del at sige hvis implementeringen af en løsning skal finde sted. Derfor er det nødvendigt at se på hvilke fordele udbyderen kan drage af løsningen og hvilke problemer løsningen kan afhjælpe.

Det er blevet estimeret, at der på nuværende tidspunkt lever omkring 7 milliarder mennesker i verden, samtidig viser statestik fra ICT[12] at der er over 5,9 milliarder mobil abonnementer. Herudover er forbruget af SMS oppe på omkring 6,1 billioner ($6,1 * 10^{12}$) SMS'er årligt. Hvis der tages udgangspunkt i en gennemsnitlig global pris på SMS 0,4 DKK (0,07 USD), genererer SMS omkring 4686000 DKK/minut (812000 USD/minut) [12].

Dette betyder, at SMS er en kæmpe forretning med en massiv omsætning.

Det må derfor også antages, at de mobilselskaber, der får en del af deres profit herfra, vil værne om denne form for indtjening. En måde at forlænge SMS'ens levetid og relevans i samfundet, kunne bl.a. være ved at løfte begrænsningen for, hvor mange tegn en SMS kan indeholde, ved hjælp af komprimering. En komprimering af dataene, der skal sendes over netværket, vil også medføre en mindsket belastning på udbyderens udstyr, hvilket vil betyde, at udbyderen vil kunne håndtere flere kunder uden udvidelse af de allerede eksisterende systemer.

4.3 Virksomheders brug af korte beskeder

Et alternativt marked for SMS kunne findes i globale virksomheder. Korte beskeder kan her bruges til små meddelelser, der skal kommunikeres ud. Virksomheder kan have god brug af dette til, at videregive information på en hurtig måde. Eksempelvis kan et link til en hjemmeside eller en kort tekst, som skal bruges digitalt, sendes mere effektivt via interne beskedklienter og PDA'er end ved, at skrive på en lap papir. Dette kan bruges af virksomheder til at lette arbejdsopgaver, effektivisere både intern og ekstern kommunikation og ellers forsimple skriftlig kommunikation i dagligdagen [21].

Virksomheder med afdelinger og filialer på tværs af kloden, kan også have stor nytte af korte beskeder, til daglig kommunikation. En SMS besked kan sendes til den anden side af jorden, hvorefter informationen hurtigt kan behandles, selvom modtageren ikke nødvendigvis sidder på kontoret [22].

Effekten af at komprimere disse beskeder varierer (se afsnit 2.1), men belastningen på netværket bliver mindre, når beskederne fylder mindre. Virksomheder, med et stort forbrug af disse beskeder, kan derfor muligvis finde besparelser. Hvis en besked skal sendes til flere modtagere, kan effekten af en komprimeret besked blive endnu større. En medarbejder i en virksomhed kan, eksempelvis, have behov for, at sende en besked til alle i sin arbejdsgruppe. Beskeden bliver komprimeret til en lidt mindre størrelse, hvilket muligvis kan spare brugen af en ekstra SMS pr. modtager. Besparelsen ved, at sende sådanne beskeder, stiger lineært med antallet af modtagere. En virksomhed kunne have interesse i dette, hvis information skulle sendes til adskillige modtagere, i kort format, samt være tilgængelig på en brugervenlig enhed 12.2.

Udover meddelelser kan korte beskeder også bruges til, at føre hele samtaler. På SMS kan en samtale frem og tilbage, bestående af få sætninger, føre til planlægning af møder og aftaler. Beskederne i sig selv er ikke store, men samtalernes længde kan variere meget. Det gør korte beskeder til et redskab for en virksomhed, som kan gøre nytte af beskedernes digitale format, der kan sendes nemt og effektivt på tværs af kloden.

4.4 Udviklingslandene

Et sted hvor markedet for SMS ikke er i nedgang, men derimod kunne opleve en stor stigning, er i udviklingslandene. Her er SMS ved at opnå en stor udbredelse, og allerede 79 % af befolkningen ejer et mobil abonnement [12]. Mobilen, og specielt SMS, bliver her, benyttet i næsten alle fortagender. I forhold til handel kan bønderne tjekke aktuelle markedspriser, tjekke op på salget, samt betale og overføre penge via SMS. Herudover benyttes mobilen i forhold til uddannelse, jobsamtaler og til, at effektivisere den medicinale sektor [23].

Et eksempel på denne form for brug af SMS kommer til udtryk i et initiativ fra UNICEF kaldet RapidSMS. RapidSMS er et open source framework baseret på SMS, som skal hjælpe med datadeling, kommunikation samt logistik. Et af de tidlige projekter med RapidSMS blev udført på et overvågningsprojekt, sat i gang, som et samarbejde mellem EU, UNICEF, ACF og Malawis regering, efter hungersnøden havde hærget i Malawi i 2002 [24]. Da dette program blev overladt til landet selv, faldt antallet af data, der blev tastet ind i databaserne kraftigt, og store forsinkelser på test samt manglende information gjorde programmet enormt ineffektivt og dyrt. Ved hjælp af RapidSMS fik man dog lavet en række væsentlige forbedringer. Blandt andet blev papirhelvedet erstattet med SMS, mængden af data, der gik tabt og blev fejlrapporteret faldt drastisk, og omkostningerne blev skåret ned til et minimum, der kun dækkede SMSernes pris.

Mobil og SMS er også specielt relevant i udviklingslandene på et andet punkt, da de ikke har andre vidt udbredte muligheder for hurtig kommunikation. De platforme, som i dag konkurrer med SMS, f.eks. Facebook, Skype og anden online kommunikation, er langt mindre væsentlig i udviklingslandene end i resten af verden. Grunden til dette, er en høj pris for internet i forhold til den

vestlige verden, hvilket for mange almindelig familier i u-landene er en for stor udgift. Ifølge statistik fra International Telecommunication Union, har kun 22,5 % af befolkningen i u-landene computer i hjemmet, mens det yderligere kun er 21 %, som har internet adgang [25]. Når dette ses i modsætning til de 79 % med et mobil abonnement, giver det mening, at SMS kommer til, at være en væsentlig faktor inden for teknologisk udvikling i disse lande. Hertil medhører, at hastigheden på bredbånd, gennemsnitligt er omkring 2 MB [25], og dette er langt fra optimalt til store dataoverførsler.

Kapitel 5

Problemformulering

5.1 Afgrænsning

Komprimering af kort tekst kan primært bruges i telekommunikation, hvor korte beskeder ofte bliver sendt, behandlet og modtaget.

Efter at have set på de forskellige markeder inden for SMS og korte beskeder generelt, ses der flere steder, som kunne gavne af komprimering. I virksomheder tales der om en effektivisering af den interne og eksterne kommunikation, samt et væsentligt mindre papir- og tidsspilde, der ellers er forbundet med skriftlig kommunikation. Dette er dog klart et lukket marked, som højst sandsynligt ikke ser den store besparelse ved brug af komprimering. Især ikke når denne sættes i forhold til den tid og energi det ville kræve at gennemføre en sådan forandring i fremgangsmåden virksomheden ellers benytter.

De mere realistiske bud findes ved de større markeder, der omfatter det veludviklede og købevenlige vestlige samfund, samt i de organisatoriske projekter, der i øjeblikket arbejder med, at forbedre levevilkårene i udviklingslandene.

I de udviklede lande er komprimering primært relevant for, at gennemtvinge et lille prisfald i taksten på SMS, hvorefter SMS'ens levetid måske kunne forlænges lidt endnu. Dette kunne ske, som en konsekvens af den nedsatte pris, der i så fald kunne medføre en rebound effekt, der kunne give nyt liv til markedet. Komprimering ville også hjælpe SMS'ens konkurrenceevne i forhold til andre sociale medier, der stadig bliver mere og mere relevante i

takt med udbredelsen af smartphones.

For udviklingslandene kan komprimering betyde, at SMS bliver et mere effektivt og omkostningsvenligt værktøj i udfordringen med, at forhøje levestandarden. Dette kunne benyttes til store organisatoriske projekter, samt være relevant for den enkelte borger, som ville opleve mulig økonomiske besparelser.

Løsningens afgrænsning er at kunne komprimere tekst til forskellige formål. Programmet i dette projekt er prototype for en løsning til det vestlige marked, som med mindre modifikationer kan tilpasses den 3. verden.

5.2 Problemformulering

Arbejdet med, at komprimere korte tekstbeskeder, er endt ud i en problemformulering, lavet på baggrund af analysen.

Hvordan kan komprimering bruges til, at forlænge SMS-standardens levetid, og udvide brugsområde, f.eks. i 3. verdens lande?

Ved at udvælge en eksisterende komprimeringsmetode, forventes det, at kunne skrive et C-program, der kan illustrere komprimeringen af korte beskeder. Yderligere bliver mulighederne for en implementering i GSM systemet udersøgt.

Del II

Implementering

Kapitel 6

Introduktion til Løsning

I denne del af rapporten, arbejdes med en løsning for projektet, der omhandler komprimering af korte beskeder. Det vil komme til udtryk ved en udvælgelse af en allerede eksisterende algoritme, som uddybes, og danner baggrund for et C-program, der kan komprimere et stykke tekst. Herudover udvikles et tilsvarende program, der dekomprimerer den komprimerede streng tilbage til den oprindelige tekst. Løsningen henvender sig både til brug i den vestlige verden, og til brug i u-landende, hvor en øget mængde information over SMS kan bidrage til den teknologiske udvikling i landene, især når nogle af de SMS-baserede nødhjælpsprojekter tages i betragtning.

Den samlede løsning er efterfølgende blevet omtalt som "DSC", som er produktnavnet for løsningen. DSC står for "DSC SMS Compressor".

DSC er, i første omgang, kun en prototype af en egentlig løsning. Løsningen bygger her på en kode hvor, tekst fra det engelske sprog komprimeres og dekomprimeres. En endelig løsning kan blive brugbar med f.eks. Swahili og andre relevante sprog, da DSCs løsningsimplementering er meget fleksibel, idet komprimeringen bygger på små tabeller og træstrukturer, der indeholder nogle oplysninger om det sprog, der bruges. Disse tabeller og træstrukturer kan bygges for ethvert sprog, og det er derfor muligt at tilpasse løsningen til ethvert ønsket sprog. Prototypen, DSC, arbejder med engelsk, hovedsageligt fordi det er nemmere, at finde test-materiale på engelsk, men også fordi engelsk er det mest benyttet internationale sprog.

Kapitel 7

Algoritmer

Der findes mange forskellige algoritmer til komprimering. De har ofte forskellige styrker og svagheder, hvilket gør dem mere eller mindre egnede i bestemte situationer. Derfor er det vigtigt, at overveje hvilken algoritme, der skal implementeres i den kontekst vi arbejder med. Vi analyserer i dette afsnit de forskellige algoritmer og metoder, som er relevante i forhold til komprimering af korte beskeder.

7.1 Run-length kodning

Den såkaldte run-length kodning, eller oversat til dansk; kørsels-længde kodning, er en metode til at komprimere sekvenser af data, hvor den samme data-værdi fremkommer flere gange i træk [26].
Ved sådanne forekomster erstattes den kontinuere sekvens af data-værdien med et nummer, der indikerer hvor mange gange data-værdien gentages efterfulgt af data-værdien selv [26].

Et simpelt eksempel kunne se således ud:

Input: LLLLLLLLOOOOOOOLLLLLOLLLLOOLLLLL

Output: 8L7O5L1O4L2O5L

Run-lenght metoden virker dog kun optimalt, når data-værdierne i gennemsnit optræder mere end 2 gange i træk, da strengen ellers ikke komprimeres.

Run-lenght metoden bruges bl.a. til, at komprimere billeder, og bruges bl.a. i mange bitmap formater som TIFF, BMP og PCX [26].

7.2 Aritmetisk kodning

Aritmetisk kodning fungerer ved, at kende antallet af mulige forskellige dataværdier, samt sandsynligheden for de forskellige værdiers forekomst. Herved kan en mængde data kodes til, at være repræsenteret af et enkelt tal [27]. Ud fra dette tal, længden af den oprindelige mængde data og sandsynligheden for de forskellige værdier er det muligt, at afkode tallet til den oprindelige datamængde.

Dette foregår ved, at inddele intervallet ([0;1[er startinterval) i delintervaller svarende til antallet af mulige dataværdier, i størrelser svarende til dataværdiernes sandsynlighed for forekomst. Efter den første værdi er fundet, indskrænkes fokusintervallet til intervallet for den pågældende værdi. Herefter inddeles det nye interval igen i delintervaller svarende til sandsynligheden af de enkelte dataværdier.
Et eksempel kunne være, som følger:

Disse mulige dataværdier med følgende sandsynligheder for forekomst er kendt:

A : 60 % som beskriver intervallet [0.0;0.6[

B : 20 % som beskriver intervallet [0.6;0.8[

C : 10 % som beskriver intervallet [0.8;0.9[

D : 10 % som beskriver intervallet [0.9;1.0[

Hertil skal sekvensen "ACD" kodes til en tilsvarende talværdi. Dette gøres ved at bestemme en talværdi som gør sig gældende i delintervallerne til de pågældende værdier. For eksemplet "ACD" ville dette være som følger: A findes i delintervallet [0.0;0.6[, herefter inddeles dette i delintervaller.

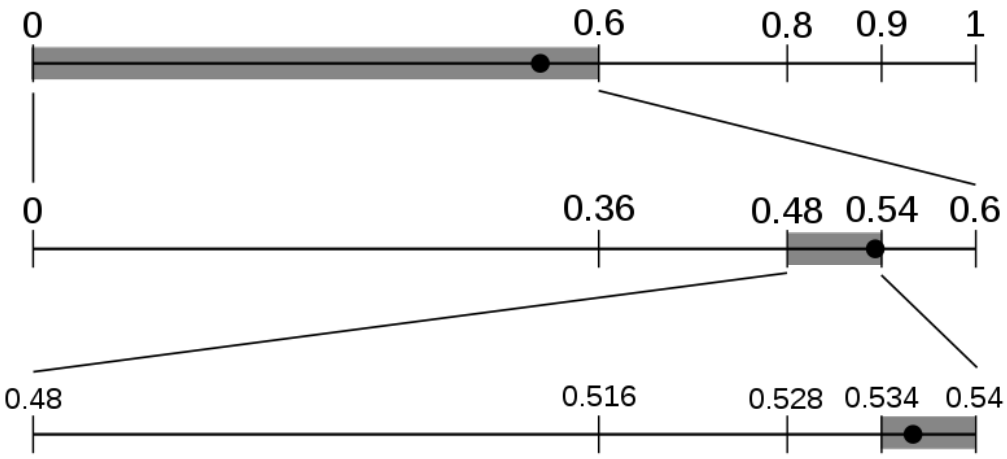
A : [0.0;0.36] eller 60 % af [0.0;0.6[

B : [0.36;0.48] eller 20 % af [0.0;0.6[

C : [0.48;0.54] eller 10 % af [0.0;0.6[

D : [0.54;0.6] eller 10 % af [0.0;0.6[

Her findes delintervallet for C, som i dette tilfælde bliver [0.48;0.54[. Herefter skal dette interval inddeles i delintervaller svarende til sandsynlighederne for forekomst, som tidligere, hvorefter delintervallet for D vælges. Dette er [0.534;0.54[. En hvilken som helst værdi i intervallet [0.534;0.54[beskriver datasekvensen "ACD" når længden af data og sandsynligheden for de forskellige dataværdiers forekomst er kendt. En illustration på dette kan ses i figur 7.1.



Figur 7.1: Illustration over hvordan delintervallerne udvælges og opdeles som beskrevet ovenfor [28].

7.3 Huffman kodning

En algoritme, som benytter sig af Huffman metoden, analyserer først det data, som skal lagres tegn for tegn. Den sorterer derefter tegnene efter sandsynlighed for forekomst. De tegn, som har den største sandsynlighed for, at forekomme, placeres i de højere lag af et binært kodetræ, og de tegn, som ikke forekommer så ofte, placeres i bunden af træet. Kodetræet har de samme præfiks-frie egenskaber, som beskrives i afsnit 2.4. De høje blade i træet kodes med en mindre tegnsekvens end de lave blade i træet, og på den måde bliver der

brugt så lidt plads, som muligt, på de tegn, der er flest af. Da koden samtidig er præfiks-fri, komprimerer den data tabsfrit [7].

Den egentlige fremgangsmåde, som Huffman metoden gør brug af, er en "prioritets-kø". Alle tegnene bliver sorteret efter hyppighed og sat i "prioritets-køen". Der oprettes to blade for de to tegn med lavest hyppighed, og bladene bliver knyttet til en knude. Den samlede hyppighed for bladene bliver ført over på knuden, som bliver sendt tilbage til "køen", der igen bliver sorteret efter hyppighed. Sådan fortsætter metoden, indtil der kun er én knude eller tegn tilbage. Det til sidst stående element bliver det øverste blad på træet, tættest på roden. Derefter tildeles alle bladene, den tilsvarende binære repræsentation, som passer i forhold til træets opbygning, hvilket betyder at alle træets kanter bliver tildelt enten "1" eller "0", og sekvenserne gemmes, eventuelt sammen med det data, som er blevet komprimeret [29].

Huffman metoden er en udbredt måde, at forbedre komprimeringsalgoritmer, som også bygger på sandsynlighed eller, som bygger på mønstre, da den kun komprimerer tegn, repræsenteret som tal(f.eks. ASCII), som andre mindre tal, og til sidst bitsekvenser. Efter en komprimering ved brug af enten sandsynlighedsregning eller mønstre, kan Huffman metoden benyttes på det allerede kompakte data, som en "back-end", til at komprimere det endnu mere. Huffman metoden bruges ofte som "back-end", en efterfølgende behandlingsprocess, til andre komprimeringsalgoritmer, så som DEFLATE(PKZIP) og flere codecs, heriblandt det populære lydformat ".MP3" og billeformatet ".JPEG" [7].

7.4 Jacob Ziv og Abraham Lempel

I 1977 og 1978 udgav Jacob Ziv og Abraham Lempel to algoritmer til komprimering , henholdsvis LZ77 og LZ78 [30]. Disse algoritmer ligger til grund for meget af den komprimerings software, som bliver benyttet i dag. Deres algoritmer bliver benyttet i forskellige former og modifikationer i kompressionssoftware. Firmaerne bag programmer, som winzip og 7-zip, har Ziv og Lempels grundtanker indbygget i deres produkter [31].

Zip-filer kan bruges til, at samle flere filer, og få dem til, at fylde mindre, eller blot komprimere enkelte filer. Dette kan være en fordel, når mere data skal

flyttes af gangen, eller filerne skal lagres på begrænset plads. Algoritmerne ligger også til grund for nogen af de mere kendte billedformater. ".GIF" og senere ".PNG" bygger på disse algoritmer [30].

7.4.1 LZ77

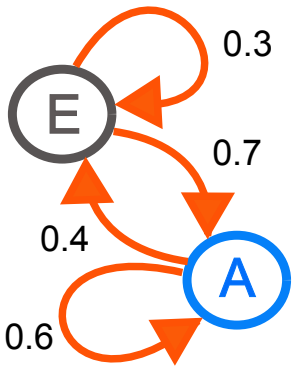
Algoritmen er baseret på et dynamisk arkiv, som bliver opbygget, mens krypteringen finder sted. Ved at kigge på forudgående tegnkombinationer og finde de samme kombinationer bagudrettet, kan kombinationerne navngives [32]. Dette betyder at gentagende mønstre, i teorien kan, skrives i kort form, når det fremkommer efter det er blevet samlet op i arkivet. Når dette skal effektiveres i praksis, giver dette dynamiske arkiv mulighed for, at "glemme" tidligere mønstre. Det sker ved, at fastholde en ramme for, hvor langt algoritmen kigger tilbage for, at finde en kombination som matcher et mønster i en tilsvarende fast ramme fremadrettet. Ziv og Lempel kalder selv disse vinduer for bufferere. Det er disse buffervinduer, som gør det muligt, at afkode den krypterede version. Bufferne betyder at arkivet bliver adaptivt til præcist det, som er inden for rammerne af de to buffere. Mønstrene, som går igen i hver buffer kan på den måde indgå i det midlertidige arkiv. Når afkodningen igen skal ske, er hele arkivet blevet glemt, og skal genopbygges på baggrund af den krypterede kode. Ved hjælp af kodens udformning, kan afkodnings algoritmen udregne arkivets indhold.

7.5 PPM Komprimering

Prediction by partial matching (PPM) er en komprimeringsmetode først udviklet af Cleary and Witten [33]. PPM fungerer ved, at bruge de tidligere tegn til at forudsige de kommende tegn. Denne forudsigelse kan så bruges til komprimering, da data kun skal bruges i de tilfælde, hvor forudsigelsen er forkert.

PPM er bygget op omkring Markov kæder. Markov kæder er et statistisk værktøj, der beskriver sandsynligheden for, at en tilstand fører til en anden.

I figur 7.2 ses tilstande for to forskellige tegn. Her overvejer Markov kæden kun et tegn for, at forudsige den næste. Dog er en Markov kæde ikke begrænset



Figur 7.2: Et eksempel på en Markov kæde med to forskellige tilstande [34]

til et enkelt tegn, men kan også gøre brug af to eller flere tegn. En Markov kæde af n tegn kaldes for, at være af n-te orden.

PPM implementeres ofte med n antal Markov kæder af $1, 2, \ldots, n$ orden. En sådan PPM implementering beskrives som $PPM(n)$.

PPM komprimering er en af de bedste metoder til, at komprimere tekst af naturlige sprog. Cleary and Witten [33] opnåede en komprimering på 2.2 bit pr. tegn af engelsk tekst. Dette ligger tæt på den teoretiske entropi for engelsk, der er estimeret til, at være mellem 0.6 bit til 1.3 bit [5].

Dog har PPM komprimering nogle problemer. Først og fremmest bruger alle Markov kæderne store mængder hukommelse, hvilket kan være et problem på indlejrede systemer hvor systemressourcerne ofte er begrænsede [35]. Et andet problem ved PPM komprimering er, at den ikke er god til at håndtere komprimering af sekvenser af tegn den ikke har set før, da systemet naturligvis ikke kender sandsynligheden for et tegn, det ikke har set før. Dette kaldes for nul-frekvens problemet, som beskrevet i afsnit 7.6.

Hvordan kodningen af symbolerne sker, varierer meget mellem implementeringer. Som regel sker dette med aritmetisk kodning (se afsnit 7.2), men det er også muligt at bruge Huffman kodning (se afsnit 7.3).

7.6 Nulfrekvensproblemet

PPM, artimetisk kodning og Huffman kodning bruger frekvensen af en række symboler til, at kode de symboler, der forekommer oftere, med færre bit. Frekvensen af de forskellige symboler kan komme fra beskeden, eller den kan være kendt på forhånd. Der er dog et problem, hvis frekvensen er kendt på forhånd, men ikke stemmer overens med beskeden. Som regel resulterer dette i en ikke-optimal komprimering, men det kan også medføre, at et symbol eller flere symboler overhovedet ikke kan kodes.

Eksempelvis kan frekvenserne af de forskellige bogstaver i alfabetet findes ved, at analysere store mængder af engelsk tekst. Dette er dog et problem, hvis frekvenserne bruges til, at komprimere dansk tekst hvor ikke-engelske tegn, som Æ, Ø og Å, kan forekomme. I en sådan situation er det ganske enkelt umuligt, at kode disse symboler, da der ikke er nogen kendt frekvens for dem.

En mulig løsning på dette problem er, at tildele alle symboler en minimumfrekvens som er så lav som muligt. Dette betyder at det er muligt at komprimere alle symboler, men det betyder også en mindre effektiv komprimering for andre symboler. Dog er det i et begrænset omfang da minimumsfrekvensen er meget lav, og mange af disse symboler derfor skal lægges sammen før at de har en frekvens stor nok til have indflydelse på placereingen af de andre symboler i Huffman træet.

Et alternativ til at forøge frekvensen af alle tegn, er at have et forestående såkaldt "escape tegn", der instruerer afkodningsalgoritmen til, at læse det næste symbol uden Huffman kodning.

En helt tredje metode er, at algoritmen simpelthen ikke håndterer symboler, som har ukendt frekvens. Dette betyder, at tegn som Æ, Ø og Å ikke er mulige at sende, hvis Huffman-træet er baseret på engelske symboler. Der er også nogle få tegn i engelsk, som bruges meget få gange, som for eksempel ï eller ê, der bruges i visse fremmedord. Hvis engelsk tekst analyseres, mødes disse symboler måske ikke, og et huffman træ baseret på den analyse kan derfor heller ikke håndtere disse tegn.

7.7 Udvælgelse

De forskellige komprimeringsmetoder har forskellige styrker og svagheder. PPM er et af de mest effektive metoder til, at komprimere naturlige sprog, men Rein et al. [35] har vist, at hukommelsesforbruget af algoritmen gør den ineffektiv i situationer med begrænsede ressourcer. Dog har Rein et al. [35] også arbejdet med en modificeret version af PPM, der angiveligt skulle have et lavere hukommelsesforbrug, som egner sig bedre til indlejrede systemer.

Huffman kodning er en af de simpleste former for tabsfrit kodning og er derfor også et af de komprimeringstyper, der bruges oftest i forskellige formater og komprimeringssystemer [7]. Huffman kodning er derfor en meget oplagt metode at vælge.

Run-length metoden er ikke velegnet til komprimering af tekst, da længere sekvenser af ens data sjældent fremkommer.

Aritmetisk kodning er relativt ny i den forstand, at patenterne for, at lave denne kodning, først udløb i 2011 [36]. Derfor er aritmetisk kodning ikke blevet benyttet meget, selvom det altid medfører en bedre eller identisk komprimering i forhold til Huffman kodning [37]. Forbedringen i forhold til Huffman kodning varier meget, i forhold til det data, der komprimeres, dog er komprimeringstiden ofte højere end 50 % i forhold til tilsvarende Huffman kodning ifølge forsøg udført af Shahbahrami et al. [37].

LZ77 har stor fleksibilitet. Det kan komprimere på både billeder og lange tekster effektivt. Svagheden ligger i det dynamiske arkiv. Når metoden bruges til at komprimere tekst er bufferen bred, så gentagede ord kan findes i et spektre af flere sætninger. I korte stykker af tekst er gentagelsen af det samme ord mindre hyppig. Det betyder at arkivet bliver mindre effektivt.

Ud fra disse forskellige styrker og svagheder er det blevet besluttet, at arbejde videre med Huffman kodning. Dette er blandt andet baseret på enkeltheden ved Huffman kodning, der gør det betydeligt lettere, at implementere på tværs af systemer. Selvom aritmetisk kodning har en højere effektivitet end Huffman kodning, så bliver det ophævet af den forøgede kompleksitet og det større brug af processorkraft.

Huffman kodning er påvirket af det førnævnte nulfrekvensproblem (se afsnit 7.6), da frekvensen er forudbestemt og muligvis ikke er optimal for enhver

besked, der skal komprimeres. I dette projekt løses dette problem ved, at tilgive alle alfabetets tegn en frekvens. Specialtegn får dertil en meget lille frekvens. Dette betyder også, at den samlede komprimeringsratio ikke vil blive påvirket synderligt, da disse tegn vil blive placeret nederst i Huffmantræet og altså kun have meget lille indflydelse på resten af træets opbygning.

Kapitel 8

Huffmans kode algoritme

Tidligere er Huffman koden blevet beskrevet(se afsnit 7.3). Dette kapitel omhandler en dybere beskrivelse af algoritmen og forklare hvordan, algoritmen er blevet benyttet i DSC.

8.1 Huffmantræ

Som beskrevet i afsnit 7.3, sorterer algoritmen først alle tegnene i en prioritetskø. De bliver sorteret så mindste tegn bliver stillet forrest i køen. Efter sorteringen sker opbygningen af træet, baseret på Huffmans kode algoritme, se algoritme 1 afsnit 8.1.

Programmet udvælger de to tegn forrest i køen og sætter dem yderst i træet. De to tegn bliver nu samlet til én knude, som bliver sat tilbage i køen, for senere at blive sat sammen med et andet tegn eller en anden knude. Eksempelvis med et alfabet bestående af a, b, c og d(se figur 8.1).

Her er sekvensen "acdc" binært skrevet "10010001001", altså en streng, der i alt optager 11 bit hvis det i figur 8.1 nævnte Huffmantræ benyttes. Skulle denne streng skrives med ukomprimerede ASCII karakterer, ville den binære streng til at beskrive karaktererne, se således ud; "01100001 01100011 01100100 01100011" og altså fylde 4x8 bits. Beskeden kunne altså, i dette tilfælde, skrives på godt en tredjedel af de bits, som skulle bruges i rent ASCII.

Algorithm 1 Algoritme for dannelse af Huffman træ

Input: En prioritetskø p med n_p symboler og tilsvarende frekvenser
Output: Et korresponderende Huffman træ
procedure HUFFMAN(p)
  while $n_p > 1$ **do** ▷ Så længe at der er mere end 1 indgang i køen
    $a \leftarrow dequeue\,(p)$ ▷ To knuder med mindst frekvens fjernes fra køen
    $b \leftarrow dequeue\,(p)$
    $c \leftarrow \{a, b\}$ ▷ En knude laves med a og b som kanter
    $f_c \leftarrow f_a + f_b$ ▷ Frekvensen af c er summen af kanterne
    $enqueue\,(p, c)$ ▷ c indsættes i køen
  end while
  return $dequeue\,(p)$ ▷ Den sidste knude er roden af Huffman træet
end procedure

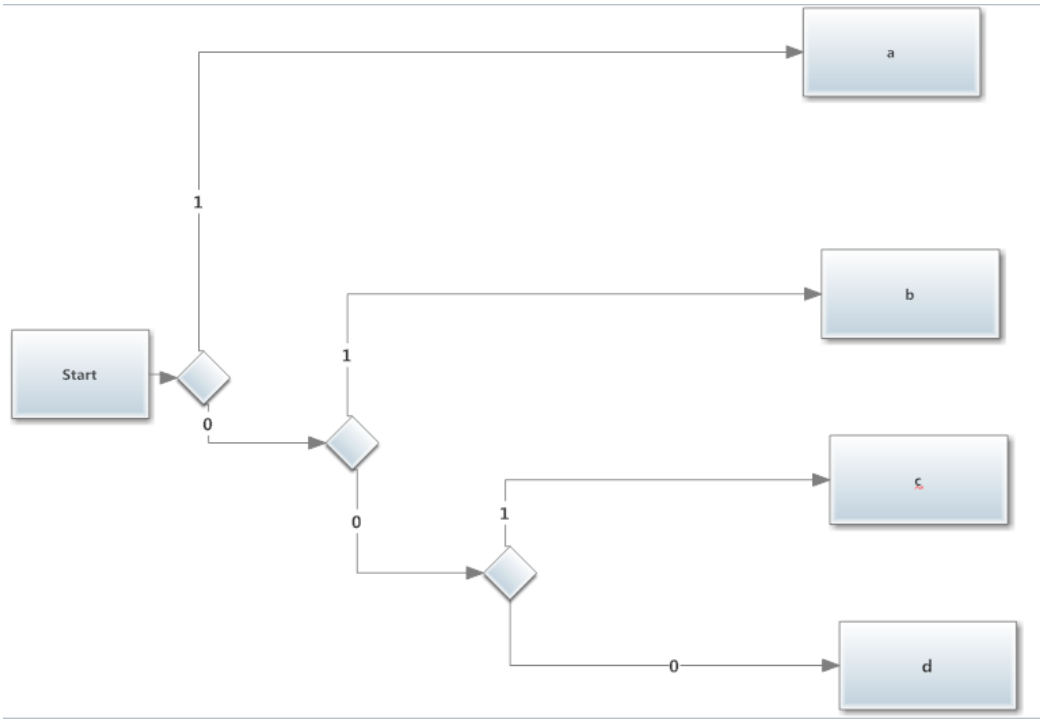
Kilde: [38], side 99, Algoritme 5.4. Huffmans kode algoritme

Her er det dog også nødvendigt, at huske på at der i ASCII tabellen kan beskrives 128 tegn, hvorimod dette Huffman træ er begrænset til de 4, der er i træet. I praksis skulle Huffmantræet dog blot udvides til, at indeholde alle tegn fra ASCII. Dette ville i nogle tilfælde betyde, at et tegn (det mindst brugte tegn)kræver flere bits for, at blive beskrevet end med ASCII. Men fordi træet er opbygget efter sandsynlighed for brugen af tegnet, vil den binære streng højst sandsynligt ikke blive længere end den oprindelige ASCII streng.

8.2 Tekstens form

Den binære streng skal imidlertid sendes via en mobiltelefon, hvis den skal implementeres ordenligt. Hvis den binære streng blot blev skrevet ind med normale karakterer i det almindeligt brugte GSM-7 alfabet, bliver beskeden en del længere end oprindeligt, da hvert tal i så fald repræsenteres af 7 bit. Den binære data kan imidlertid sendes ganske uafhængigt af diverse tegnsæts- alfabeter. Dette kan gøres ved hjælp af GSM's 8-bit encoding (se underafsnit 3.2.5), hvor den binære data sendes direkte i stykker af 8-bit.

Hvis eksemplet fra før tages op igen, kan strengen "acdc" via Huffmantræet skrives som "10010001001". Nu deles strengen så op i dele af 8-bit, som kan



Figur 8.1: Huffmantræ

sendes via det mobile netværk. Dataene kan nu modtages af modtagerenheden og dekomprimeres via et identisk Huffmantræ.

Stadie	Nuværende streng
Oprindelig besked	acdc
Konverteret via Huffmantræ	10010001001
Stykker af 8 bit	10010001 00100000

I ovenstående eksempel bruges der kun 3 ud af de 8 bit i den sidst sendte byte. Her kan der opstå et problem på modtagersiden, da dette muligvis kan forstås som et tegn, eller blot få DSC til at gå i hårknude. I næste afsnit kigges der på en løsning på dette problem.

Kapitel 9

Padding

Komprimeringsalgoritmer, som for eksempel Huffman-træer, laver oftest et output, der er i individuelle bits. Dette skaber dog et problem, da størstedelen af data-overførsel er baseret på blokke af flere bits. Dette er også tilfældet med GSMs 8-bit kodning, hvor antallet af bits, der sendes, skal være deleligt med 8. Det er derfor nødvendigt, at tilføje et antal bits til sidst i den komprimerede besked for, at gøre antallet af bit deleligt med 8. Dette kaldes for "data padding".

Den padding, der tilføjes på den komprimerede besked, skal dog også være adskillelig fra den egentlige data, som den komprimerede besked indeholder. Hvis for eksempel et antal 0 bit tilføjes som padding, så er det ikke være muligt, at skelne mellem data og padding, hvis den egentlige datastreng ender med et 0-bit. Hvis sådanne data dekomprimeres, tolkes dekomprimeringsalgoritmen paddingen som komprimerede tegn og dermed lave et forkert output. Dette problem ses i tabel 9.1

Data								Padding							
1	1	1	0	1	1	1	0	0	0	0	0	0	0	0	0
Byte 1								Byte 2							

Tabel 9.1: Problemet ved at bruge 0-padding: Der er ingen mulig måde, at skelne mellem data og padding.

For at håndtere dette problem, er det nødvendigt, at have et 1-bit som skelner mellem data og padding. Dette 1-bit kan så bruges under dekomprimerings-

processen til, at skelne mellem data og padding. Et eksempel på dette ses i tabel 9.2. For at finde ud af, hvor mange antal bits, der er i beskeden, læses beskeden bagfra indtil, 1-bit'et mødes, og de alle bit før dette læses som bits, der beskriver det sendte data.

Dette giver dog et problem i de tilfælde hvor, datastrengen allerede er delelig med 8. I dette tilfælde er det nødvendigt, at tilføje en ekstra byte, og fylde den byte med padding (se tabel 9.3), da dekomprimeringsprocessen ellers håndterer sendt data som padding. Hvis det antages, at antallet af bit i den sidste byte har ligefordeling[1], så sker dette kun i $\frac{1}{8}$ af gangene, og det er derfor ikke noget betydeligt problem.

Data								Padding							
1	1	1	0	1	1	1	0	0	1	0	0	0	0	0	0
Byte 1								Byte 2							

Tabel 9.2: Samme data som i tabel 9.1, men der kan her skelnes mellem data og padding.

Data								Padding							
1	1	1	0	1	1	1	0	1	0	0	0	0	0	0	0
Byte 1								Byte 2							

Tabel 9.3: Antallet af data-bit er deleligt med 8, og det er derfor nødvendigt, at tilføje en ekstra byte, som udelukkende består af padding

Denne metode er demonstreret her med blokke af 8-bit, men den samme metode kan bruges til alle tilfælde, hvor en række bit skal kodes som n-bit blokke.

[1] At sandsynligheden for, at der er $0, 1, 2, \ldots, 8$ bits i den sidste byte, er lige store

Kapitel 10

Programstruktur

Som en implementering af Huffman komprimering i programmeringssproget C, gør DSC brug af følgende elementer.

10.1 Datastruktur

Når det kommer til struktureringen af data, gør DSC hovedsagligt brug af tre structs. Den første struct beskriver et knudepunkt og indeholder to heltalsværdier værdier, der beskriver henholdsvis nul-kanten og et-kanten til knudepunktet. Den andet struct er huffman-træet. Denne struct indeholder to arrays, der sammen udgør en tabel over tegnene og deres sti gennem træet. Structen indeholder derudover et array over alle knudepunkterne samt et heltal, der beskriver roden. Den sidste struct indeholder to heltal, der sammen beskriver frekvensen for et knudepunkt.

10.2 Prioritetskø

For at kunne bygge et Huffman træ, er det nødvendig først, at lave en prioritetskø (se afsnit 7.3). En prioritetskø kan bygges på flere forskellige måder, hvoraf den hurtigste er en *heap* - en træ-baseret datastruktur. En heap gør det muligt, at indsætte og fjerne ting fra køen i $\log n$ operationer [39].

Dog har vi valgt, at implementere denne prioritetskø, som en sorteret liste i stedet for en heap. I en sorteret liste fungerer indsættelse ved, at der søges lineært igennem listen, elementet indsættes og de resterende elementer rykkes tilbage. Ved fjernelse af elementer fra listen, fjerner vi blot det første element i listen og rykker de resterende elementer frem. Begge disse handlinger sørger for at listen altid er sorteret og bruger n operationer, hvilket er langsommere end det tilsvarende for en heap. Den sorterede liste er dog meget simplere at implementere og vi har derfor valgt at bruge den, da fokus for DSC ikke er effektivitet, men komprimeringsforhold. En visualisering på en prioritetskø, som en liste, kan ses i figur 10.1.

Prioritetskøen er implementeret med funktionerne `p_enqueue` og `p_dequeue`, der henholdsvis tilføjer og fjerner et element fra prioritetskøen.

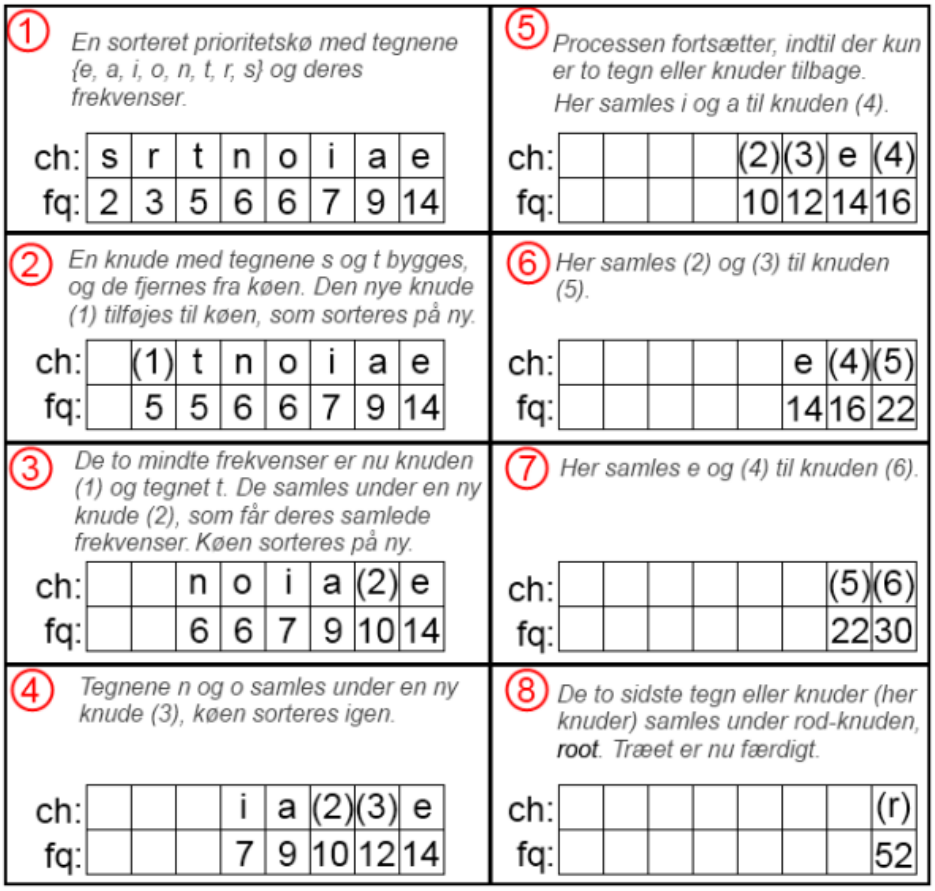
10.3 Initialisering

I initialiseringsdelen af DSC dannes selve Huffman træet, der bruges til komprimeringen. Denne del er baseret ud fra Huffmans algoritme, der er beskrevet i algoritme 1 i kapitel 8.1. Denne algoritme gør brug af en prioritetskø til, at danne et Huffman træ.

Denne algoritme håndteres af funktionen `huffman_build`, der bygger træet ud fra et histogram af frekvensen af de forskellige symboler. Dog bliver den sidste del af algoritmen håndteret af funktionen `huffman_build_characters`, der rekursivt bevæger sig ned gennem træet og danner en tabel over hvert symbol og tilhørende kode. Denne tabel er essentiel for komprimeringen, hvorimod træet er essentielt for dekomprimeringen. Herefter omtales denne tabel som *symboltabellen*.

10.4 Komprimering

Komprimering er meget simpelt, når Huffman træet først er bygget. Her itereres over hvert tegn i en besked, og den korresponderende kode, som findes via symboltabellen bruges. Funktionen `huffman_compress` udfører dette.



Figur 10.1: Visualisering af prioritetskø. Huffman-træet ses ikke visuelt.

10.5 Dekomprimering

Ved dekomprimering bruges træet i stedet for symboltabellen, som ved komprimering. Der startes ved roden af træet og fortsættes langs enten 0- eller 1-kanten af hvert knudepunkt, alt efter det korresponderende bit der modtages som input. Til sidst nås et blad i træet, som repræsenterer et symbol, som så er det dekomprimerede symbol. Dette gentages indtil, der ikke er flere bit at læse.

10.6 Padding

Padding og fjernelse af padding (beskrevet i kapitel 9) er håndteret af funktionerne `pad` og `unpad`, der henholdsvis tilføjer og fjerner paddingen af et antal bit.

10.7 Kørsel af program

Programmet og den tilhørende kildekode kan findes på den medfølgende CD. På CD'en findes derudover en tilhørende readme-fil som forklarer brugen af programmet og de forskellige syntakser programmet kan og skal bruge. Programmet er udgivet under MIT-licens.

Kapitel 11

Tekstanalyse

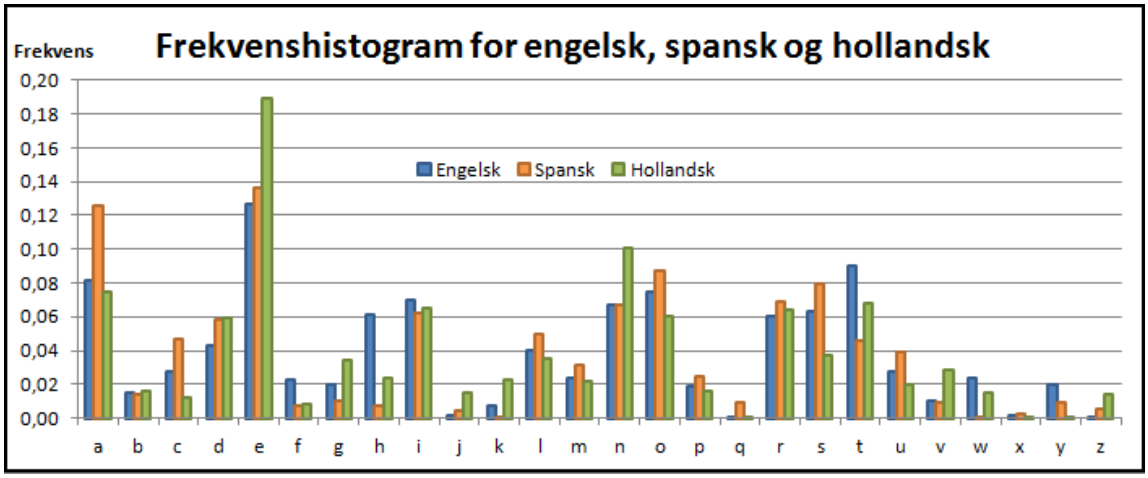
Dette kapitel omhandler analyse af tekst skrevet på engelsk. Det har til formål, at vise, hvilke tegnfrekvenser, beskeder og andre små tekster har, hvilket resulterer i et binært træ. Træet bestemmer bit-sekvenserne af de forskellige tegn i DSCs Huffman kode. Sproget engelsk er valgt på baggrund af anvendeligheden i hele verdenen. Samtidig er tegnsættet identisk med latin-tegnsættet, hvilket er brugt i mange sprog også sprog anvendt i f.eks. Afrika.

For at håndtere tegnfrekvenserne, benytter DSC sig af frekvenshistogrammer. Disse histogrammer er datasamlinger med frekvens på hvert eneste tegn. De fulde histogrammer, benyttet i projektet, kan findes i vedlagte filer (se CD bilag).

11.1 Sprogforskellighed

Tages det engelske sprog som udgangspunkt, kan der udvikles en løsning på tekstkomprimering vha. Huffman-kodning, som kan implementeres mange steder, uden at skulle ændres. Skal DSC tilpasses andre sprog med en anden tegn-frekvens, er en ændring frekvenshistogrammet, nødvendig for, at opretholde den mest effektive komprimering. Det ses på figur 11.1, at tegnfrekvenserne for tre udvalgte sprog, ikke er så ens, at samme kode ville kunne bruges til, at behandle dem alle optimalt. Figur 11.1 er en behandling af

engelsk, spansk og hollandsk sprog, for at vise, at det ikke ville være optimalt, at benytte samme tegn-frekvenser til komprimering af netop de tre udvalgte sprog.

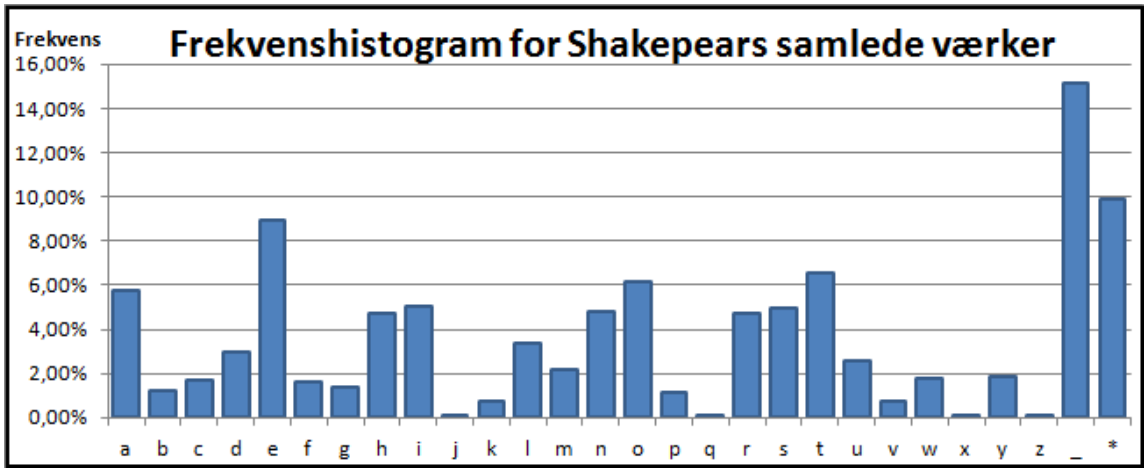


Figur 11.1: Kilder: [40] [41] [42]

Hvis flere forskellige standarder blev implementeret i forskellige dele af verden, ville det betyde at to mobiltelefoner, som brugte hver sin komprimeringsmetode, ikke ville kunne læse eller åbne tekstbeskeder fra hinanden, da der bruges én metode til, at komprimere beskeden, og en anden til, at dekomprimere den.

For at komme denne situation til livs, kunne den sendte SMS-besked have en header, som fortalte, hvilken metode, der var blevet brugt til, at komprimere beskeden. Dette ville i så fald kræve, at alle mobiler indeholdt alle de forskellige metoder til komprimering og dekomprimering, men situationer, hvor SMS-beskeder ikke ville kunne dekomprimeres korrekt, kan ikke opstå.

At understøtte flere forskellige sprog, er ikke noget stort problem. Det eneste, som skal bruges, er en fil, der indeholder de karakteristiske tegn-frekvenser, som det pågældende sprog måtte have, eller alternativt en fil, som beskriver det tilhørende huffman-træ for indkodning og afkodning af et bestemt sprog. I begge tilfælde, ville hukommelseskravet til den håndholdte enhed ikke være nogen yderligere belastning for enhedens ydelse eller hastighed, da den mængde data, som enheden skulle behandle af gangen ikke ændres.



Figur 11.2: The Complete Works of William Shakespeare in plain text UTF-8
Kilde: *http://www.gutenberg.org/ebooks/100.txt.utf-8*

11.2 Engelsk

Da det, som tidligere beskrevet i afsnit 11.1, er let, at implementere og understøtte flere forskellige sprog, tages der her udgangspunkt i, at prototypen af DSC udvikles til det engelske sprog. For at sikre den mest optimale komprimering, laves der 3 forskellige histogrammer, der hver har en forskellig tilgang til det engelske sprog.

11.2.1 Shakespeare

I dette underafsnit følger en analyse af Shakespears samlede værker på engelsk (se figur 11.2).

De to mest højrestående søjler med værdierne "_" og "*" repræsenterer hhv. mellemrums-tegnet og alle øvrige ikke i forvejen repræsenterede tegn, så som andre tegnsætningstegn (".", ",", "?" osv.), parenteser og ikke-printbare tegn, som "line-feed" og "vertical tab"(se figur 11.2).

Dette histogram giver en god repræsentation af korrekt, klassisk engelsk, da shakespeares samlede værker indeholder over 5 millioner tegn samlet i en bred vifte af engelske ord. Engelsk SMS-sprog varierer muligvis en smule fra

dette frekvens-histogram, men de primære tegn, som udgør de overordnede træk, forventes, at have samme rækkefølge, hvis de sorteres faldende efter frekvens. Dog ville såkaldte "emoticons", smiley'er og andre tegn, som skal forestille ansigter eller følelser, ikke blive særligt godt komprimeret, da disse tegn (punkttegn og parenteser) sjældnere forekommer i Shakespears værker.

11.2.2 Twitter

Et radikalt anderledes udgangspunkt for det engelske sprog findes i de sociale medier. Blandt disse er Twitter et af de nemmeste steder, at få tilgang til et stort antal af beskeder, som er velegnede til tekstanalyse. De såkaldte "tweets" giver et indblik i en anden måde, at benytte det engelske sprog, som måske ofte forekommer i almindelige SMS-beskeder. Forekomsten af de specialtegn, forkortelser og talesprog er markant hyppigere i "tweets" sammenlignet med Shakespeares samlede værker.

11.2.3 Facebook

Som et alternativ til de forgående histogrammer, er der blevet fortaget tekst-analyse på en samling normale Facebook-beskeder, indsamlet med tilladelse fra personerne, der har skrevet dem. Disse er skrevet i en blanding af engelsk og dansk, med et højt antal af de førnævnte "emoticons". Beskederne giver en god repræsentation af slang, og den måde engelsk bliver benyttet af unge mennesker i de sociale medier, også når det kommer til de dele af verden, der ikke har engelsk som førstesprog.

Kapitel 12

Test

Efter DSC's færdiggørelse, er der blevet lavet flere test på dets funktionalitet. Testene er blevet foretaget ved, at opbygge forskellige huffman-træer, og lade dem komprimere den samme tekst. Teksten er en fil med egentlige beskeder på engelsk af forskellig længde, for at teste komprimeringens effektivitet. Teksten og alle resultaterne kan findes i rapportens bilag (se del III). De komplette histogrammer ligger på den vedlagte CD.

Formålet med testen er, at se hvor stor en andel af beskeden, der kan komprimeres og mere konkret, at finde ud af, hvorvidt en besked, der før havde en længde svarende til 2 eller 3 beskeder, kan komprimeres ned til, at fylde 1 eller 2 beskeder.

Herudover laves der diverse test på programmet, for at se, hvordan bl.a. ukendte tegn og tal håndteres.

12.1 Tal og ukendte tegn

Den første test blev foretaget for, at se hvordan programmet håndterer specialtegn og tal. Tal blev komprimeret og dekomprimeret succesfuldt. Shakespeare-histogrammet, som blev benyttet, indeholdte alle de 10 cifre, hvilket gjorde dette muligt. Effekten af at komprimere tal sammenlignet med tekst var mindre effektiv, da fremkomsten af tal, ikke er særlig høj når det sammenlignes med mange af de bogstaver som fremkommer hyppigt i Shakespeares samlede

værker. Specialtegn skabte et andet problem. De tegn, som histogrammet indeholdte, skabte ingen problemer, men tegn, som ikke fandtes i histogrammet, kunne i første omgang ikke afkodes efter, at have være indkodet med DSC. Dette blev, i den endelige prototype, løst, som beskrevet i afsnittet om nulfrekvens (se afsnit 7.6) ved manuelt, at tildele alle kendte tegn en lille frekvens.

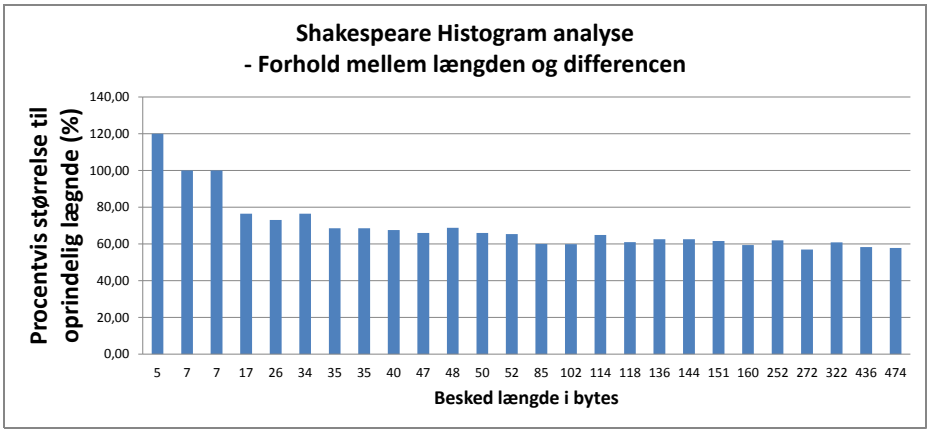
12.2 Komprimeringseffekt

Histogrammerne blev i testen indlæst hver for sig, og testen blev kørt med dette som eneste ændring. Alle histogrammerne kunne indkode og afkode testens tekstfil. DSC kan altså håndtere de tegn, som ofte bliver brugt i SMS-samtaler. De brugte SMS beskeder kan ses i bilaget (se afsnit III). Teksten, som blev komprimeret, blev i testen komprimeret linjevis. Resultaterne for, hvor meget teksten blev komprimeret, er på den måde repræsentativt for en enkelt SMS pr. linje. Først undersøgtes det, om den dekomprimerede besked var identisk med den oprindelige. Hvis dette var tilfældet, afleverede testprogrammet selv størrelsen på beskeden før og efter komprimeringen. Der blev dernæst udregnet forholdet mellem disse, for bedre at kunne sammenligne effekten. Effekten er for hvert histogram blevet udregnet på 26 beskeder af længde mellem 5 og 474 tegn. 474 tegn er lige under grænsen til, at fylde 4 SMS-beskeder. Disse 26 beskeder er skrevet i forbindelse med denne test, og de er valgt for at teste løsningen under forskellige forhold. Det er normale tekstbeskeder, der simulerer tekstbeskeder sendt gennem SMS, og derfor giver det mening at bruge disse beskeder til testen.

12.2.1 Shakespeare histogram

Shakespeare histogrammet (se figur 12.1) giver en god repræsentation af det engelske sprog, men giver dog ikke umiddelbart den mest optimale komprimering under testen af de 26 beskeder. Sammenhængen mellem beskedens antal af tegn før komprimering, og den andel af de oprindelige tegn, der er tilbage efter selve komprimeringen ses på figur 12.1.

Beskederne havde en gennemsnitsstørrelse på 69,4 % af den oprindelige. Der opstod dog problemer med komprimeringen når det kom til de helt korte



Figur 12.1: Komprimering med Shakespeare Histogram

beskeder. Som det fremgår af grafen blev beskeden på fem tegn større end den oprindelige. Med dette histogram kunne der ikke læses tegn som ikke fremgik af Shakespeares samlede værker. Danske tegn kunne altså ikke læses igen efter en komprimering.

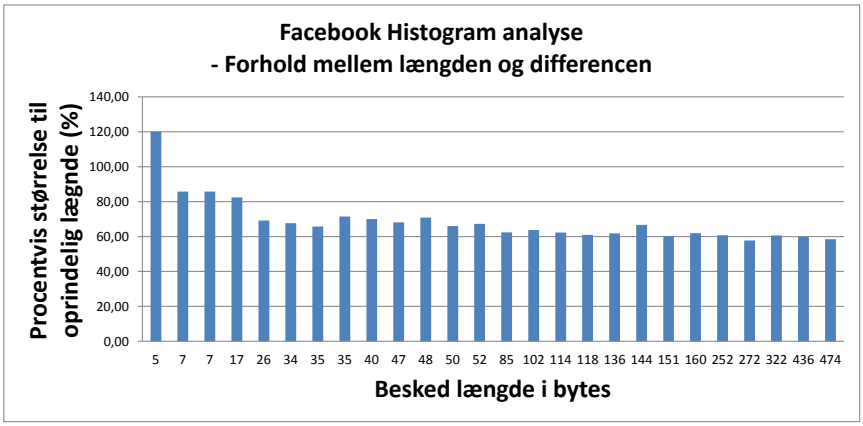
12.2.2 Facebook histogram

Histogrammet bygger på forskellige statusopdateringer fra Facebook. Det nye Huffman-træ, som blev bygget af det nye histogram, blev så testet på de samme 26 beskeder for, at sammenligne effekten. Resultatet er afbilledet på figur 12.2.

Beskedernes gennemsnitsstørrelse var denne gang 68,7 % af den oprindelige. Som det fremgår af grafen, havde dette histogram også problemer med den helt lille besked, som igen blev større end den oprindelige. Til forskel fra Shakespeare histogrammet, blev de to beskeder på 7 tegn denne gang komprimeret.

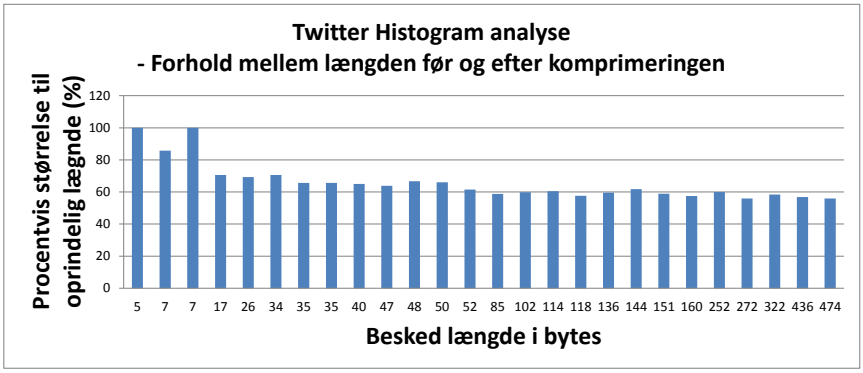
12.2.3 Twitter histogram

For at efterkomme mængden af specialtegn i en SMS besked, blev der også testet et histogram, som bygger på en lang række Tweets fra Twitter. Smiley'er,



Figur 12.2: Komprimering med Facebook Histogram

forkortelser og tal fremkommer oftere på Twitter, ligesom de gør i beskeder. Da dette blev testet på de samme 26 beskeder, blev resultatet bedre, som set på figur 12.3.



Figur 12.3: Komprimering med Twitter Histogram

Som det eneste histogram, blev ingen af disse beskeder større af komprimeringen. Gennemsnitligt fyldte de komprimerede beskeder 65,9 % af den oprindelige beskedstørrelse. Dertil, er det i nogle tilfælde være muligt, at komprimere nogle beskeder så meget, at de i stedet for at skulle fylde 4 SMS-beskeder, kun behøvede at fylde 2. Den nye grænse for antal tegn pr. SMS-besked med denne komprimering, ligger på 241 tegn, hvis man tager højde for 8 bit til padding.

Kapitel 13

Diskussion

Testen af DSC illustrerer hvor effektiv en komprimering, der kan opnås i forhold til almindelige SMS-beskeder. I alle de benyttede histogrammer lå komprimeringsraten på omkring 30-35 %. Denne komprimering blev opnået med en forholdsvis begrænset mængde tekst til tekstanalyse, og kunne sandsynligvis også forbedres, med en mere repræsentativ baggrund, f.eks. en million SMS-beskeder på de relevante sprog. Dog er sådanne mængder af data svær at få fat på, med mindre den stammer fra et telefonselskab, hvor adgangen til beskederne ville ses som brud på straffelovens § 264 d, der forbyder videregivelse af personlige meddelelser uden tilladelse Justitsministeriet [43].

Hertil kommer den forventede rebound-effekt, en ændring af et produkt kan medføre. Da belastningen på udbyderens netværk nedsættes, og nettet dermed kan håndtere flere kunder uden en merpris for udbyderen, kan dette resultere i en mindsket SMS-takst. Forbedring af SMS-teknologien ved, at komprimere beskederne, vil derved betyde, at brugeren ville have mulighed for, at skrive mere pr. SMS eller få servicen billigere. Det kan antages, at der muligvis kan ses en stigning i antallet af sendte beskeder grundet den mindskede SMS-takst. Dette kunne medføre, at flere får adgang til servicen, da den kan blive billigere.

Komprimering har derved potentiale til, at kunne påvirke det vestlige marked, enten ved en direkte eller indirekte prisnedsættelse af SMS-taksten. En direkte nedsættelse kunne ske ved, at give udbyderen en chance for, at sænke prisen pr. SMS, men beholde det oprindelige loft for det maksimale antal tegn i

en besked. På baggrund af de test der er foretaget burde udbyderen kunne forvente en reel aflastning af netværket samt en pladsbesparelse på omkring 30 %. Dette ville give udbyderen mulighed for at have flere kunder uden øgede omkostninger. En anden mulighed ville være den indirekte nedsættelse, som ville gøre sig gældende i alle de tilfælde hvor en SMS er på over 160 tegn. Nogle af disse kunne komprimeres ned til blot at fylde en enkelt besked, og hvis der tages udgangspunkt i DSC løsningen ville det give omkring 80-90 ekstra tegn, i gennemsnit, pr. besked.

Under testen af de forskellige histogrammer, fremkommer det tydeligt, at et mere specialiseret histogram giver et bedre resultat. Da tweets benyttes, som baggrund for et histogram, blev det tilsvarende huffman-træ mere præcist i forhold til den type besked, testen indeholdte. Dette resulterede i, at der i mange tilfælde, blev en højere komprimeringsratio.

I forhold til de organisatoriske projekter, der foregår i u-landende, betyder dette, at et mere specialiseret histogram ville være nødvendigt, hvis det skulle give en effektiv komprimering af syge-SMS'er eller andet. De beskeder, der bliver benyttet i UNICEF projektet i Malawi, bestod hovedsagligt af tal og enkelte bogstaver. En simpel tekstanalyse på en del af deres arkiv ville kunne give et meget præcist histogram, som desuden kun ville indeholde de relevante tegn. Dette ville medføre en utrolig effektiv komprimering, og give mulighed for længere beskeder, med mere information i hver.

En medicin-status, der før ville have fyldt 2 beskeder, ville kunne komprimeres til 1 og derved halvere udgiften i forhold til SMS-taksten.

13.1 Videreudvikling af program

Løsningen, som er udviklet i denne rapport, er kun en prototype af et færdigt program. Hvis dette skal videreudvikles til en egentlig løsning, er der flere ting, som skal implementeres. De to vigtigste opgaver er implementering på mobiltelefoner og automatisk komprimering og dekomprimering ved afsendelse og modtagelse af beskeder.

Der kan også ses på en løsning, hvor der komprimeres hele ord, fra en database bestående af ord, frem for enkelte tegn, og sammenligne resultaterne mellem de to. Det betyder, at meget brugte ord, bliver meget mindre, da de bliver

beskrevet med meget få bits. Derimod ord, som sjældent bliver brugt, fylder meget mere end hvis ordet ikke er komprimeret. Det kan diskuteres om det overhovedet kunne være nyttigt, at komprimere på tekst bestående af meget brugte ord, da sådanne beskeder sjældent kommer til at fylde mere end én besked, hvorved komprimering ikke er særlig relevant, da beskedantallet ikke kan formindskes.

Ud over det kan der også ses på muligheden for, at implementere dele af de allerede eksisterende komprimeringsalgoritmer, som findes i GSM standarden, men som aldrig er blevet taget i brug [44]. En version af Huffman-algoritmen er tilgængelig, samt andre komprimeringsmetoder, så som tegn-gruppering, komprimering af specielle ord vha. et "escape tegn" og en enkelt metode, som fjerner tegnsætning, og ikke yderligere komprimerer teksten. Den sidste metode er altså en "komprimering" med tab af data.

Til sidst kan der også ses på en optimering af løsningen blandt andet vha. heaps, som beskrevet i afsnit 10.2.

Kapitel 14

Konklusion

Igennem dette projekt er der blevet gennemgået de forskellige markeder for SMS og vurderet, hvorvidt disse ville have gavn af en eventuel komprimering af beskederne. Dette er blevet vurderet på baggrund af hvilke ressourcer, der er til rådighed både i forhold til udbyderen af SMS servicen og i forhold til brugeren. Det er elementer som lagerplads af beskederne, belastning af netværket, SMS-takster samt hvilke alternativer de forskellige målgrupper har til SMS, som kommunikationsmiddel, der undersøges og vurderes på.

Denne analyse endte ud i en problemformulering, som projektet efterfølgende benyttede til, at vurdere den endelige løsningskvalitet og relevans ud fra.

Hvordan kan komprimering bruges til, at forlænge SMS standardens levetid, og udvide brugsområde, f.eks. i 3. verdens lande?

For at illustrere, hvordan problemet kan løses, er der udviklet et program med en huffman kode, som kan komprimere tekst. Denne løsning, kaldet DSC, fungerer ved hjælp af en sorteret liste af tegnfrekvenser lavet på baggrund af twitter-beskeder. Programmet benytter denne liste til, at konstruere et huffman træ, der bestemmer de forskellige tegns nye bitsekvenser. Dette bruges til, at komprimere korte tekstbeskeder. Effekten af DSC blev vurderet ud fra en række test hvortil, der blev udviklet 3 forskellige versioner med hver deres frekvenshistogram. Hver af disse versioner blev sat til at komprimere 26 engelske beskeder af forskellig længde, hvorefter resultaterne blev sammenlignet og vurderet. Den mest effektive version af programmet, var baseret på et histogram bygget på twitter beskeder. Denne fik en komprimeringsrate på

omkring 30-35 %.

Denne besparelse betyder, at beskeder, der før overskred den faste øvre grænse for antal tegn, og før måtte sendes, som 2 beskeder, kunne komprimeres ned til blot at fylde én. Hvis denne ændring af SMS blev implementeret, som en fast del af den SMS-service, der findes i dag, kunne det f.eks. give grundlag for en såkaldt rebound effekt. Forbedring af SMS-teknologien ville muligvis fremme en enten direkte eller indirekte nedsættelse af SMS-taksten. Hvis udbyderen tog fordel af komprimeringen kunne en direkte prisnedsættelse finde sted, da udbyderen kunne understøtte flere kunder på sit netværk, og derved finansiere en lavere SMS-takst. En mere indirekte nedsættelse kunne komme, som en direkte konsekvens af, at flere beskeder kun ville kræve en enkelt besked at sende, og derved spare dobbeltbeskeder, som en stor del af brugerne sender, væk.

Uanset hvilken form denne prisnedsættelse ville antage, kan det forventes, at dette ville give brugerene i den udviklede verden incitament til, at benytte SMS oftere. Herudover kunne en billigere SMS ydelse være med til, at sænke udgifterne til de organisatoriske projekter, der forgår i u-landene, og hjælpe til, at etablere SMS som en brugt teknologi i disse lande.

Litteratur

[1] European Telecommunications Standards Institute. Digital cellular telecommunications system (phase 2+); alphabets and language-specific information (gsm 03.38), Juli 1996. URL `http://www.etsi.org/deliver/etsi_gts/03/0338/05.03.00_60/gsmts_0338v050300p.pdf`.

[2] Th. Mobiltelefoner mod aids. *Berlingske*, 1:18, Februar 2007.

[3] Den engelske Wikipedia. Informationsteori, . URL `http://en.wikipedia.org/wiki/Information_theory`.

[4] C. E. Shannon. A mathematical theory of communication. 27, 1948. URL `http://plan9.bell-labs.com/cm/ms/what/shannonday/shannon1948.pdf`.

[5] C. E. Shannon. Prediction and entropy of printed english. *The Bell System Technical Journal*, pages 50–64, 1951.

[6] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. Wiley, 2nd edition, 2006. ISBN 978-0471241959.

[7] Ken Huffman. Huffman algorithm description. Website www.huffmancoding.com. URL `http://www.huffmancoding.com/my-uncle/huffman-algorithm`.

[8] Den engelske Wikipedia. Komprimering, . URL `http://en.wikipedia.org/wiki/Data_compression`.

[9] Kona Macphee. Codes, trees and the prefix property. +Plus Magazine Website, January 2000. URL `http://plus.maths.org/content/os/issue10/features/infotheory/index`.

[10] Gerald R. Tamayo. Prefix-free codes. via sites.google.com, December 2009. URL https://sites.google.com/site/datacompressionguide/prefix-free.

[11] Ascii binary list, . URL http://www.theskull.com/javascript/ascii-binary-list.html.

[12] International telecommunication union, 2010 - mobile, . URL http://www.itu.int/ITU-D/ict/material/FactsFigures2010.pdf.

[13] rcgroup. Meget andet end tekstbeskeder. URL http://www.rcogm.dk/SMS-610.aspx.

[14] Den engelske Wikipedia. Gsm, . URL http://en.wikipedia.org/wiki/GSM.

[15] Symbian Software Ltd. Types of encoding. URL http://library.developer.nokia.com/index.jsp?topic=/S60_5th_Edition_Cpp_Developers_Library/GUID-35228542-8C95-4849-A73F-2B4F082F0C44/sdk/doc_source/guide/System-Libraries-subsystem-guide/CharacterConversion/SMSEncodingConverters/SMSEncodingTypes.html.

[16] ANSI INCITS. Coded character sets - 7-bit american national standard code for information interchange (7-bit ascii). URL http://sliderule.mraiow.com/w/images/7/73/ASCII.pdf.

[17] Den engelske Wikipedia. Ascii, . URL http://en.wikipedia.org/wiki/ASCII.

[18] Ascii table and description, . URL http://www.asciitable.com/.

[19] Unicode. Unicode. URL http://www.unicode.org/standard/principles.html.

[20] Rfc 3629 - utf-8, a transformation format of iso 10646. URL http://www.faqs.org/rfcs/rfc3629.html.

[21] IT og telestyrelsen. Sms fra arbejdspladsen. Website http://borger.itst.dk. URL http://borger.itst.dk/aktuelt/sms-fra-arbejdspladsen.

[22] Apropos kommunikation. Hver tredje får sms- beskeder fra jobbet uden for arbejdstiden. Website http://www.aprokom.dk. URL http://www.aprokom.dk/cm648/.

[23] Fremtiden for mobil og sms i afrika. URL http://techcrunch.com/2012/05/27/mobile-developing-world/.

[24] Rapidsms - malawi – nutritional surveillance. URL http://www.rapidsms.org/case-studies/malawi-nutritional-surviellence/.

[25] International telecommunication union, 2011 - internet, . URL http://www.itu.int/ITU-D/ict/facts/2011/material/ICTFactsFigures2011.pdf.

[26] James D. Murray and William Van Ryper. *Encyclopedia of Graphics File Formats, 2nd Edition*. URL http://www.fileformat.info/mirror/egff/ch09_03.htm.

[27] Dan Hirschberg. Data compression - static defined word schemes. URL http://www.ics.uci.edu/~dan/pubs/DC-Sec3.html. Section 3.4.

[28] Dcoetzee. Arithmetic encoding. URL http://en.wikipedia.org/w/index.php?title=File:Arithmetic_encoding.svg&page=1.

[29] David A. Huffman. Encoding the "neatness"of ones and zeroes. *Scientific American*, September:54–58, September 1991. URL http://www.huffmancoding.com/my-uncle/scientific-american.

[30] Christina Zeeh. The lempel ziv algorithm. 2003. URL https://ece.uwaterloo.ca/~ece611/LempelZiv.pdf.

[31] Den engelske Wikipedia. Zip (file format) - structure. . URL http://en.wikipedia.org/wiki/Zip_(file_format)#Structure.

[32] J. Ziv and A Lempel. *A universal algorithm for sequential datacompression*. 1977. URL http://www.cs.duke.edu/courses/spring03/cps296.5/papers/ziv_lempel_1977_universal_algorithm.pdf.

[33] J. Cleary and I. Witten. Data compression using adaptive coding and partial string matching. *Communications, IEEE Transactions on*, 32(4): 396 – 402, apr 1984. ISSN 0090-6778. doi: 10.1109/TCOM.1984.1096090.

[34] Den engelske Wikipedia. Markov chain, . URL http://en.wikipedia.org/w/index.php?title=File:Markovkate_01.svg.

[35] Stephan Rein, Clemens Gühmann, and Frank Fitzek. Compression of short text on embedded systems. *Journal of Computers*, 1(6):1–10, September 2006.

[36] Glenn George, Langdon Jr, and Jorma Johannen Rissanen. Method and means for arithmetic string coding, October 1977. URL http://www.google.com/patents/US4122440.

[37] Asadollah Shahbahrami, Ramin Bahrampour, Mobin Sabbaghi Rostami, and Mostafa Ayoubi Mobarhan. Evaluation of huffman and arithmetic algorithms for multimedia compression standards. URL http://arxiv.org/ftp/arxiv/papers/1109/1109.0216.pdf.

[38] David J. C. MacKay. *Information Theory, Inference & Learning Algorithms*. Cambridge University Press, New York, NY, USA, 2002. ISBN 0521642981.

[39] Robert Sedgewick and Kevin Wayne. Priority queues, December 2012. URL http://algs4.cs.princeton.edu/24pq/.

[40] Statistical distributions of english text. URL http://www.data-compression.com/english.html.

[41] Jan Paul van Soest. *Stichting voor Publieksvoorlichting over Wetenschap en Techniek*. Aramith, 1988.

[42] Fletcher Pratt. *Secret and Urgent: the Story of Codes and Ciphers Blue Ribbon Books*. Blue Ribbon Books, 1942.

[43] Justitsministeriet. Straffeloven §264, Juli 2001. URL https://www.retsinformation.dk/forms/R0710.aspx?id=138671#Kap27.

[44] Digital cellular telecommunications system (phase 2+); alphabets and language-specific information. Technical Specification GSM 03.38 version 7.2.0, European Telecommunications Standards Institute, 07 1998.

Del III

Bilag

Testresultater

Data for Twitter histogram

Status for komprimering	Oprindelig størrelse i bytes	Post komprimeret størrelse i bytes	Størrelse i forhold til oprindelig størrelse i %	Difference i bytes
OK	5	5	100,00	0
OK	7	6	85,71	1
OK	7	7	100,00	0
OK	17	12	70,59	5
OK	26	18	69,23	8
OK	34	24	70,59	10
OK	35	23	65,71	12
OK	35	23	65,71	12
OK	40	26	65,00	14
OK	47	30	63,83	17
OK	48	32	66,67	16
OK	50	33	66,00	17
OK	52	32	61,54	20
OK	85	50	58,82	35
OK	102	61	59,80	41
OK	114	69	60,53	45
OK	118	68	57,63	50
OK	136	81	59,56	55
OK	144	89	61,81	55
OK	151	89	58,94	62
OK	160	92	57,50	68
OK	252	151	59,92	101
OK	272	152	55,88	120
OK	322	188	58,39	134
OK	436	248	56,88	188
OK	474	265	55,91	209
Gennemsnitlig størrelse efter komprimering :			65,85	

Tabel 14.1: Twitter histogram

Data for Facebook histogram

Status for komprimering	Oprindelig størrelse i bytes	Post komprimeret størrelse i bytes	Størrelse i forhold til oprindelig størrelse i %	Difference i bytes
OK	5	6	120,00	-1
OK	7	6	85,71	1
OK	7	6	85,71	1
OK	17	14	82,35	3
OK	26	18	69,23	8
OK	34	23	67,65	11
OK	35	23	65,71	12
OK	35	25	71,43	10
OK	40	28	70,00	12
OK	47	32	68,09	15
OK	48	34	70,83	14
OK	50	33	66,00	17
OK	52	35	67,31	17
OK	85	53	62,35	32
OK	102	65	63,73	37
OK	114	71	62,28	43
OK	118	72	61,02	46
OK	136	84	61,76	52
OK	144	96	66,67	48
OK	151	91	60,26	60
OK	160	99	61,88	61
OK	252	153	60,71	99
OK	272	157	57,72	115
OK	322	195	60,56	127
OK	436	261	59,86	175
OK	474	277	58,44	197
Gennemsnitlig størrelse efter komprimering :			68,74	

Tabel 14.2: Facebook histogram

Data for Shakespeare histogram

Status for komprimering	Oprindelig størrelse i bytes	Post komprimeret størrelse i bytes	Størrelse i forhold til oprindelig størrelse i %	Difference i bytes
OK	5	6	120,00	-1
OK	7	7	100,00	0
OK	7	7	100,00	0
OK	17	13	76,47	4
OK	26	19	73,08	7
OK	34	26	76,47	8
OK	35	24	68,57	11
OK	35	24	68,57	11
OK	40	27	67,50	13
OK	47	31	65,96	16
OK	48	33	68,75	15
OK	50	33	66,00	17
OK	52	34	65,38	18
OK	85	51	60,00	34
OK	102	61	59,80	41
OK	114	74	64,91	40
OK	118	72	61,02	46
OK	136	85	62,50	51
OK	144	90	62,50	54
OK	151	93	61,59	58
OK	160	95	59,38	65
OK	252	156	61,90	96
OK	272	155	56,99	117
OK	322	196	60,87	126
OK	436	254	58,26	182
OK	474	274	57,81	200
Gennemsnitlig størrelse efter komprimering :			69,40	

Tabel 14.3: Shakespeare histogram

Testbeskeder

1. In case I don't return to Ireland it's because marie has killed me for trying to pick her up at the wrong terminal =D
2. Monsieur, please attend to the drinking that WILL go down at the court yard shortly.
3. Ok, don't worry bout us no more, it's aight now, but can u check if she lost her phone at your place?
4. Hi. Marian here. RU off work? Wanna the ciggies
5. Whenever you close let us know. Bring alkohol!
6. Hi everyone .. i have my birthday Tomorrow .. I'm going to bring the evening with my danish girls whos here to visit me in ireland it could be nuce and cool if you would joint me to some beers for celebrate my 20 years birthday . We going to be in fritzersimon in templebar 8 a clock. Many new years hug maria Bjerregaard
7. Cheese
8. Burger
9. At o'connels now
10. Ok will leave in the next 15 mins
11. We'll grab a beer then :P
12. Cool
13. Hey, im going to Dublin where r u?
14. You not with chis??? Im still in lauras
15. I don't have any money :-(not hitting town tonight
16. Ah shit buzz dude, tlk 2 ya monday hve a good one
17. Im in laura hving a quick powernap
18. Dont forget to ask mariusz and glenn for reference today :) you can send the info to jakobalbertbjork@hotmail.com

19. Just came home,they left this morning and we still have the car so went around co.wicklow.How are ya,where are ya and how is the quite smoking for a day going?

20. They left yesterday,so Im FREE!We went down to Ring of Kerry and then Galway and Cliffs of mohair.When are they leaving..when are ya back home?

21. sounds like youre having a good time. i guess as long as theres enough booze its all good .is your studies going fine so far as well? im all good as well, since I returned to germany it was on the whole way better than id expected. i spent the last 2 months relaxing, going to a festival, meeting up with friends/going out and finally did some apartment hunting for university. tomorrow im gonna move to Nuremburg and in around a week university will start for me as well.

22. youre right, christian, tina, rebecca and me went to the Munich Oktoberfest . it was great there, nice atmosphere, bavarian beer, traditional music for dancing on the benches and tables etc. thanks to some generous french people next to us we didnt even have to pay for our beer and food so happy days . im already looking forward to the Oktoberfest next year. did you actually manage to have the long-planned 'Kir-Festival' this year?

23. Keepin just fine :P Denmark is better than I remember, and I get to drink alot more than usual, so its all goodˆ Making new friends at the university and stuff so happy days! How 'bout you? Could see that you met some of the old ppl at oktoberfestˆˆ

24. fair point. would appear everyone i know have left by now... But life is good in denmark :) Studies are good indeed :D Drinking is a uild in par of it

25. not reight now, no i gotta write some cl for universities, bu.. i dont want to im coming back to dublin on the 3rd, 4th and 5th of july

26. hey :)concerning going to gym today ill try to be there outside the gym on time and wait for you there. ill have to get my badge reactivated before but i hope it wont take too long. in case you dont see me outside gym yet, could you possibly wait for me there? cu later :)