UNIVERSITATEA POLITEHNICA DIN BUCUREŞTI

Şcoala Doctorală de Electronică, Telecomunicaţii şi Tehnologia Informaţiei

Decizie nr. 937 **din** 21-10-2022

REZUMAT TEZĂ DE DOCTORAT

Ing. Sabin BELU

METODE AVANSATE DE COMPRESIE IN AUTOMOTIVE

ADVANCED DATA COMPRESSION METHODS IN AUTOMOTIVE

COMISIA DE DOCTORAT

Prof. Dr. Ing. Gheorghe BREZEANU Univ. Politehnica din Bucureşti	Preşedinte
Prof. Dr. Ing. Daniela COLŢUC Univ. Politehnica din Bucureşti	Conducător de doctorat
Prof.Dr.Ing. Ioan TĂBUŞ Universitatea Tampere, Finlanda	Referent
Prof.Dr.Ing. Daniela TĂRNICERIU Universitatea Tehnica “Gh. Asachi”, Iași	Referent
Conf.Dr.Ing. Daniela FAUR Univ. Politehnica din Bucureşti	Referent

BUCUREŞTI 2022

Cuprins

Capitolul 1

Introducere

Stocarea datelor are o poveste proprie și aparte. De la crearea lor în 1956, capacitățile unităților de stocare au crescut de peste 50 de milioane de ori. Primul hard disk avea o greutate de 1 tonă și putea stoca doar 5 MB de date. După 26 de ani, a fost produs primul hard disk de 1 GigaByte (GB), iar în perioada 2007-2011, capacitățile hard diskului s-au dublat de la 1 TeraByte (TB) la 4 TeraByte. În următorii zece ani, hard disk-urile de 20 TB au devenit la fel de comune ca și camerele digitale.

Viețile noastre sunt acum înconjurate de volume din ce în ce mai mari de date. Acest fenomen numit Big Data, este unul pe care încă încercăm să-l înțelegem în zilele noastre, deoarece nu există o definiție standard a acestuia. Știm doar că Big Data este definit de unele caracteristici.

Big Data este o combinație de date structurate, semi-structurate și nestructurate colectate de anumite organizații, care pot fi exploatate pentru informații și pot fi utilizate în proiecte de învățare automată, modelare predictivă și alte aplicații cum ar fi analiza avansată.

Sisteme ce procesează și stochează Big Data au devenit o componentă comună a arhitecturilor de gestionare a datelor în organizații, combinate cu instrumente care sprijină utilizarea analitică a acestora. Big Data sunt seturi complexe de date, care sunt atât de voluminoase, încât software-ul tradițional de procesare și compresie de date nu le poate gestiona, sau dacă poate, nu le gestionează intr-un mod eficient.

Big Data se caracterizează printr-o varietate mare de volum, viteză, voracitate, veridicitate și valoare. Sunt cinci elemente cheie care ar trebui să fie prezente pentru a caracteriza un ansamblu de date sub denumirea de Big Data. Toate aceste caracteristici sunt cele care fac din Big Data, în cele din urmă, o uriașă afacere pentru organizații.

1.1 Algoritmii Clasici de Compresie de Date şi Big Data

Cand vorbim de Big Data, putem spune de ceva timp că algoritmii clasici au atins de mult timp apogeul. Desigur, aplicarea compresiei clasice in Big Data are beneficiile sale,

dar luând in considerare o rată de compresie extraordinara de 99%, tot rămân 1% din date pentru a fi procesate, stocate sau transportate pe diferite canale de transmisie sau comunicare.

Acest 1% este în sine o provocare uriașă. Întrucât dimensiunea unei arhive de date Big Data comprimate poate fi în intervalul de zeci de petabytes, chiar dacă raportul de compresie este mai mare sau egal cu 99%.

Acest tip de „economii" în compresie nu mai are nici o relevanță în zilele noastre. Este necesară aplicarea unor noi tehnici capabile să ofere rate de compresie mai mari de 99,998%.

Aceste metode se numesc compresie diferențială sau delta.

1.2 Industria Auto si Big Data

„Sindromul" Big Data afectează din ce în ce mai multe industrii, iar industria Auto, numita pe scurt Automotive, este una dintre ele. Sistemele și tehnologiile auto au ajuns la un nivel de complexitate nemaintalnit. Cu cât sistemele sunt mai complexe, cu atât procesează mai multe fișiere, foldere, fluxuri de date, stream-uri etc. Fișiere audio, filme, hărți, informații generale despre autovehicul, alte date de infotainment, cumulate, tind să fie în zona gigaocteților de date. Procesarea unor volume atât de mari de date, când vine vorba de scrierea fișierelor firmware sau instalarea software-ului pe dispozitivele electronice ale automobilelor (ECU) devine o povară din ce în ce mai mare pentru producatorii OEM si Tier-1.

Sistemele auto au multe versiuni iar aceste versiuni noi de software pot apărea chiar și săptămânal. A face față unui numar atât de mare de versiuni de software precum și a unui volum mare de date, necesită algoritmi specializați și mult mai performanți, cu mult peste algoritmii clasici.

Industria Auto are multe motive in a folosi algoritmii de compresie diferenţială sau delta

De ce sistemele auto trebuie sa fie constant actualizate? Cu cât un sistem devine mai complex, cu atât este mai predispus la erori. Codurile de diagnoza ale unei probleme sau defecțiuni a automobilului (Diagnosis Trouble Codes, DTC în termeni Automotive) sunt evenimente ce conduc sistemul software al automobilului intr-o stare necunoscută, intr-o situatie noua pe care inginerii sau arhitectii de sistem nu au prevazut-o.

Dacă aceste defecte apar fără nici un avertisment, se genereaza o eroare numita *Fault*, și în funcție de gravitatea acesteia, ecranul de afișaj al automobilului poate afișa acest defect cu o pictogramă $VerificareMotor$ (Check-Engine). Neverificarea automobilului într-un anumit interval de timp sau distanța (până în 600km) poate avea repercursiuni asupra vehiculului și a proprietarului acestuia cum ar fi afectarea securității și stabilității autovehiculului, pierderea parțială sau totală a garanției autovehiculului, costuri mai mari de întreținere a automobilului, etc.

Capitolul 2

Compresia de Date bazată pe Dicţionar

2.1 Introducere in algoritmii de compresie de date

Acest capitol se concentrează pe algoritmii de compresie de date fără pierdere bazați pe dicționar. Începem cu prezentarea algoritmului Lempel Ziv '77 [115], urmat de variantele sale LZSS [117] și LZP [113]. De asemenea, este prezentată descrierea unui algoritm nou, numit ROLZ (Reduced Offset Lempel Ziv), un algoritm de compresie nedocumentat, nepatentat, a carui descriere a fost facuta dintr-un studiu amanunţit asupra unor texte si referinţe disparate sau din descrierile parțiale ale algoritmilor LZP, LZRW-4 [142] și o aplicatia de compresie numita RKive[112], creat de Malcolm Taylor.

Contribuțiile originale pe tema compresiei bazate pe dicționar sunt:

- Descrierea unui algoritm nedocumentat, algoritmul RoLZ, din notele și descrierile algoritmilor LZP, LZRW-4 și RKive, sau alte discuţii pe forumuri.
- Îmbunătățiri ale algoritmului RoLZ față de versiunile sugerate de Charles Bloom și Malcolm Taylor.

Acest lucru a fost publicat în lucrarea de conferință *RoLZ - The Reduced Offset LZ Data Compression Algorithm* [128].

2.2 Compresia de Date fără pierderi bazată pe dictionar

Am scris despre acest algoritm ROLZ ca un tribut adus epocii de aur a compresiei de date. Dezvăluirea sa a durat ceva timp pentru că am incercat să deduc modul în care acest algoritm funcționează, cât și pentru a încerca să aflăm dacă deducţiile noastre sunt corecte prin diverse teste empirice şi analize amanunţite ale acestui algoritm.

Spre deosebire de implementările clasice ale algoritmului Lempel Ziv, ROLZ folosește un set „redus" de simboluri din care este ales un posibil subset de „potriviri" ulterioare.

ROLZ minimizează informațiile necesare pentru a codifica subşirul gasit. Marele atu al unei astfel de abordări este un raport de compresie mai mare, dezavantajând viteza de decompresie. Cele trei variante de software implementate de noi, LZSS[117], LZP[113] și ROLZ[129], au fost testate pe cinci tipuri de date. In toate testele noastre, raportul de compresie al ROLZ a fost cu mult superior celorlalţi algoritmi: LZSS si LZP.

Apariția programului de compresie RKIVE[112] la începutul anilor '90 ca fiind de facto unul dintre cele mai bune instrumente de compresie a datelor de pe piață, a creat multă agitație și zvonuri în comunitățile BBS online. RKIVE a fost scris de Malcolm Taylor [112], un programator din Noua Zeelandă. A fost adoptat in scurt timp de la lansare de cercetători și ingineri din întreaga lume, deoarece algoritmul atingea rapoarte de compresie nemaiţntalnite până atunci.

De asemenea, în anii '90 și mai ales în epoca de boom Dot-Com, au fost adoptate o mulțime de brevete în compresia de date. Totuşi, ROLZ, acronimul de la *Reduced Offset Lempel Ziv*, care s-a descoperit mai târziu a fi algoritmul de baza în RKIVE, este și astăzi lipsit de brevete si patente, după toate cercetarile noastre.

Merită menționat brevetul americanului Robert Jung 5.140.321 [119], cunoscut și sub numele de *LZ77 Limited Search Patent*. Această idee a jucat un rol important în modelarea ROLZ și va fi discutată mai târziu.

Malcolm Taylor nu a publicat codul sursa pentru RKive, şi, ca o consecinţa directă, algoritmul său ROLZ a rămas nedocumentat ani de zile; unele indicii au apărut in fișierul read.me asociat pachetului de compresie RKIVE. În cadrul documentului, Malcolm [112] îi multumeşte lui Charles Bloom pentru discuţiile avute ţmpreună și pentru ajutor in scrierea programului său.

Revenind cu un an in urmă, în 1995, Charles Bloom inventează un nou algoritm pe care il prezintă la conferinta "Data Compression Conference" din Utah, în anul următor [113]. Acest algoritm a fost numit *Lempel Ziv Prediction* (LZP). În lucrarea sa care descrie LZP, apare prima oară următoarul text: *"Acest algoritm funcționează prin reducerea setului de simboluri disponibile unui codificator LZ77"*. În plus, comparând LZP cu LZ77, se dovedește că sunt folosiți mai puțini biți pentru a codifica un subşir comun, deoarece doar lungimea acestui subşir comun este stocată in arhivă. Charles Bloom a prezentat patru versiuni pentru algoritmii LZP: de la LZP1, care a folosit un context de căutare fix de ordin 3 [1]în tabelul hash și o fereastră LZ de 16K octeţi la LZP4, care a folosit un context de ordin 5 și fără noduri de coliziune in tabela de hash.

Cu toate acestea, nici măcar cea mai bună variantă de LZP nu s-a putut apropia de rezultatele RKive în raport cu rata de compresie. S-a dovedit că RKive este o combinație de succes intre metode de compresie și euristică, de la compresia solidă la analiza optimă a LZ77.

Dar algoritmul complet de compresie din RKive va rămâne un mister la fel de puternic ca si rata acestuia de compresie, pentru mulți ani de acum înainte.

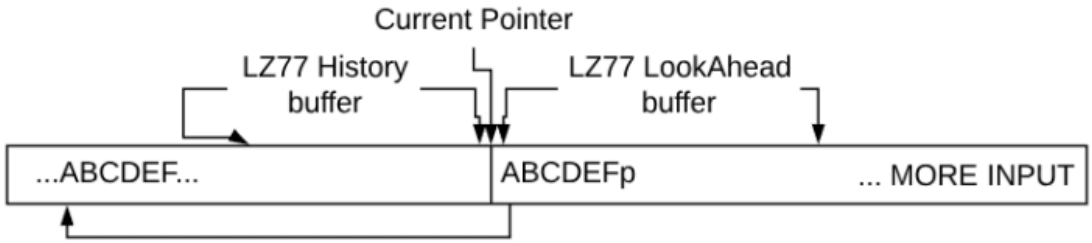

Figura 2.1 Diagram of the *LZ77* algorithm.

Scopul acestui capitol este de a descrie ce se află în spatele acestui program numit RKive și poate, din acest moment, să redea algoritmului ROLZ o utilizare mai largă în cadrul programelor software de compresie de date.

2.3 Algoritmul LZ77

Algoritmul Lempel-Ziv '77 [115] este un algoritm de compresie pe bază de dicționar. Funcționalitatea de bază a algoritmului este înlocuirea sub-șirurilor formate din succesiuni de simboluri frecvent întâlnite din fluxul de intrare în perechi de poziție și lungime. Așa cum se arată în Fig. 2.1, LZ77 împarte fluxul de intrare în bufer de istoric (LZ History) și bufer de anticipare (LZ LookAhead); codificarea de ieșire constă dintr-un triplu ⟨d, l, s⟩ care înseamnă *distanta*, *lungime* si *simbol*, unde *simbol* este primul simbol de nepotrivire (numit si literal) după perechea ⟨d, l⟩.

Să presupunem că există un șir S, care începe la poziția i a fluxului de intrare, numit indicator curent, vezi Fig. 2.1. Simbolurile situate la poziţia indicatorului curent sunt comparate cu simboluri similare din *LZ History* sau simboluri care au aceeași valoare hash atunci când sunt combinate într-un *"șir"*. Şir-ul este o inşiruire de simboluri din alfabetul utilizat. Nu există nici un context[2]implicat în alegerea unui *"șir"* cu care să se potrivească. Doar indicatorul curent face separarea dintre buferele *LZ-History* și *LZ-Lookahead*, adica buferul de istoric și buferul de anticipare din fluxul de intrare.

- Șirul $S_{i\cdots i+l}$ de lungime l are o altă apariție P, care începe d poziții mai devreme în text, $S_{i-d\cdots i-d+l}$.
- Această apariție anterioară a lui S ar trebui să fie întotdeauna mai mică decât l_{max} și ar trebui să înceapă într-o fereastră $S_{i-dmax\cdots i-1}$.
- Valorile d și l trebuie să satisfacă următoarele constrângeri: $d \leq d_{max}$ și $l \leq l_{max}$
- Subşirurile $S_{i\cdots i+l}$ și $S_{id\cdots id+l}$ se suprapun numai dacă $d \leq l$, ceea ce face ca *LZ77 să fie un algoritm autocompresibil*.
- Când se află în modul "greedy", LZ77 încearcă întotdeauna să maximizeze l-ul dintre toate posibilele apariții ale lui P_j în LZ History.

- Tripletul $\langle d, l, s \rangle$ ar putea fi codificat suplimentar folosind $log_2(d_{max}) + log_2(l_{max} - LZ\_MIN\_LEN)$ biți.

LZ_MIN_LEN este lungimea minimă de codare dincolo de care este acceptat un subşir comun.

Cand Storer și Szymanski și-au făcut celebră versiunea lor modificată de LZ77, cu o variantă numită LZSS[117], autorii au introdus conceptul de $codingflag$, folosit pentru a diferenția un literal de o pereche distanță/lungime ce defineşte un subşir comun. Acest $codingflag$ cu o lungime de doar *1-bit* elimină necesitatea de a stoca $tripletul$ LZ77 la fiecare iterație a algoritmului, salvând astfel un literal pentru o nouă căutare și, eventual, o nouă potrivire.

Algoritmul ROLZ este implementat în 7 pași distincţi, dintre care primii 5 sunt derivaţi din LZP:

- 1) În primul pas, ROLZ calculează valoarea de context *hashIndex* pentru poziția curentă i, prin inserarea simbolurilor *cntxN* văzute anterior într-o funcție hash hF.
- 2) În pasul 2, ROLZ folosește această valoare ca index ţintr-un tabel hash. Această valoare este un pointer către o listă de noduri de coliziune; dacă valoarea este *nulă* sau lista de noduri de coliziune este goală, ROLZ trece direct la pasul 4. Dacă lista nu este goală, ROLZ intră în pasul 3.
- 3) În pasul 3, ROLZ efectuează o potrivire încercând să găsească cea mai mare lungime a unei potriviri din toate cele posibile, între șirul aflat la poziția curentă i și șirul aflat la poziția indicată de primul nod al listei de noduri de coliziune;
- 4) În pasul 4, ROLZ încearcă să maximizeze cea mai lungă potrivire, avansând la următorul nod din această listă de noduri de coliziune; deci pasul 3 se repetă pentru fiecare nod până când lista s-a epuizat sau se ajunge la numărul maxim de noduri căutate.
- 5) În pasul 5, ROLZ calculează cea mai mare lungime dintre toate potrivirile, $l_{max} = max(l\_node_N, l\_node_{N-1}, \cdots, \text{l\_node}_1)$; acest l_{max} este de asemenea stocat, împreună cu indexul unde a fost găsit, numit aici n_{max};
- 6) În pasul 6, ROLZ inserează poziția curentă, i, în lista curentă de noduri de coliziune;
- 7) În pasul final, ROLZ scrie l_{max} în locația de ieșire. Dacă l_{max} nu este nul, atunci si n_{max} este trimis in locația de ieșire.

În algoritmul LZP, șirul de la indicatorul curent din poziția *i* a buferului *LZ-Lookahead* este potrivit numai cu șiruri care urmează același context din *LZ-History*. Cea mai lungă

Tabel 2.1 LZSS, LZP AND ROLZ COMPRESSION RATES

	ORIGINAL SIZE (MiB[3])	**LZSS (%)**	**LZP (%)**	**ROLZ (%)**
Executable	701	68.63	65.04	**64.43**
Formatted Text	135	25.15	23.00	**21.87**
Object Files	526	43.68	37.87	**37.33**
Text Files	86	58.95	58.17	**54.68**
Misc. Binaries	555	56.46	54.84	**53.92**

Tabel 2.2 DECOMPRESSION TIMES FOR ROLZ VS LZSS AND LZP

	ROLZ (ms)	**ROLZ vs LZSS (times)**	**ROLZ vs LZP (times)**
Executable	13,312	1.03	2.55
Formatted Text	1,037	3.02	2.46
Object Files	4,413	2.26	2.08
Text Files	954	2.04	2.26
Misc. Binaries	7,933	3.58	2.40

potrivire este codificată numai după lungimea sa, l, nu de o pereche <d,l>. Pointerul curent este întotdeauna salvat în tabelul hash. Dacă nu se găsește nici o potrivire, simbolul curent, S_i, este scris în locația de ieșire și indicatorul curent avansează la următoarea locație. Dacă se găsește o potrivire a lungimii l, l este scris în fluxul de ieșire și pointerul curent avansează l + 1 locații.

În algoritmul ROLZ, șirul de la indicatorul curent din poziția i a buferului *LZ-Lookahead* se potrivește numai cu șirurile care urmează aceluiași context din *LZ-History*. Toate potrivirile l_p sunt stocate și lungimea maximă l_{max} este calculată. Pointerul curent este întotdeauna adăugat la lista de noduri de coliziune. Dacă l_{max} este zero, nu se găsește nici o potrivire, simbolul curent Si este trimis la locația de ieșire și pointerul curent avansează la următoarea locație. Dacă l_{max} nu este zero, este trimis la ieșire și pointerul curent avansează l_{max+1} la locație curentă indexată cu valoarea lui l_{max+1}.

2.4 Rezultate experimentale cu LZSS, LZP si ROLZ

Am implementat variantele LZSS [118], LZP [113] și ROLZ [142]. Ca rezultate ale testelor, oferim ratele de compresie și viteza de decompresie care sunt disponibile din testele efectuate cu aplicațiile pe care le-am implementat pentru acest capitol; a se vedea tabelele 2.1 si 2.2.

2.5 Concluzii

Am găsit ţn ROLZ un algoritm excepțional. In comparație cu LZSS, folosirea unei variabile de context ne permite să păstrăm un anumit grad de selecţie în fluxul de ieșire, ţn funcţie de datele de intrare. ROLZ dovedeşte astfel că este potrivit pentru o modelare de compresie de ordin N, oferind o rată de compresie mult mai bună decăt LZ77, care este de context 0.

În timp ce LZSS este de obicei legat de o fereastră LZ finită, LZP și ROLZ pot funcționa într-un mediu în care poziția și distanța acestor subşiruri sunt infinite ca valori. Această caracteristică îl face mai valoros în industria Auto, unde cantitatea de date procesate crește odată cu orice nouă actualizare de software.

In testele noastre, ROLZ a oferit cea mai bună rată de compresie, dintre toți cei trei algoritmi testaţi.

Capitolul 3

Decodor Rapid Huffman Canonic

Acest capitol se concentrează pe codificatorii de entropie, în special, codarea Huffman și o anumită variantă numită codificarea Huffman canonică. Se începe cu prezentarea binecunoscutului algoritm Huffman, urmată de regulile care crează codurile canonice de prefix, de unde și denumirea variantei, Huffman Canonic.

De asemenea, este prezentată descrierea unui algoritm original, numit Fast Canonical Huffman Decoder (FCHD), care prin design, se concentrează pe crearea unui algoritm cu randament ridicat la decodare.

Contribuțiile originale cu privire la codarea canonică Huffman sunt:

- O nouă metodă canonică de decodare Huffman, cu capacitatea de a procesa 8 până la 12 simboluri într-un singur ciclu de decodare.
- O metodă nouă și îmbunătățită pentru crearea arborelui Huffman clasic.
- O metodă nouă și îmbunătățită pentru stocarea canonică a lungimii cuvintelor de cod Huffman.

Acesta este un algoritm original, nu o îmbunătățire a unei lucrări anterior prezentate in Prior Art. Testat temeinic, algoritmul a reușit să depășească viteza de decompresie de 2 GB/s. Textul acestui articol a fost publicat în următoarea lucrare a conferinței: *Fast Huffman Canonic Decoder* [131].

3.1 Procedeul de codare Huffman

Codurile Huffman există încă din 1951, când au fost inventate de D.A.Huffman [81]. Problema pe care D.A.Huffman încerca să o rezolve era găsirea celei mai eficiente metode de reprezentare a unui simbol intr-o formă binară mult mai compacta. D.A.Huffman a rezolvat această problemă cu o idee simplă, o idee care i-a adus faima în Teoria Informației ca unul dintre cei mai mari realizatori tehnici din toate timpurile.

În teorie, codurile Huffman[81] se apropie de optimalitate, dar când vine vorba de implementare, metoda este sub-optimă, mai ales când sunt codificate puține simboluri. Vom explica acest lucru mai jos.

3.2 Optimalitatea codării Huffman

Codarea clasică Huffman este sub-optimă în practică. Metoda se dovedește a fi mai puțin scalabilă pentru un flux de intrare care are definit un alfabet mai mare (mai multe simboluri), datorită informațiilor suplimentare folosite in stocarea acestuia. În afară de notația folosita la codificarea nodurilor, arborele Huffman clasic și codurile generate nu au proprietăți suplimentare pe care ne putem baza pentru a le stocare şi/sau transmite optim.

Codurile canonice Huffman posedă proprietăți unice care permit transmiterea, stocarea și reconstrucţia simbolurilor de intrare într-un mod optim. Codurile canonice Huffman sunt secvențiale. O altă proprietate a acestor coduri, numită Consecutive Value, permite codurilor canonice Huffman să fie generate automat în ordine secvențială.

3.3 Soluţia noastră de decodare rapidă

Spre deosebire de implementările algoritmului Huffman analizate de noi, algoritmul nostru de decodare rapidă Huffman, numit FCHD, implementează un algoritm proprietar care permite decodarea extrem de rapidă a mai multor simboluri într-un singur ciclu de decodare.

3.3.1 Algoritmul Huffman canonic

Fie $s_1, s_2, \ldots, s_N$ simboluri dintr-un alfabet sursă, sortate în funcție de frecvența lor de apariție în fluxul de intrare. s_1 fiind simbolul cel mai frecvent și s_N fiind cel mai puțin întâlnit. Se dau şi $l_{s_1} \leqslant l_{s_2} \leqslant \ldots \leqslant l_{s_N}$ lungimile de *biți* din arborele clasic Huffman.

Pseudocodul de generare a codurilor canonice Huffman este:

- **Pasul 1**: $codeword_{s1} = 0$
- **Pasul 2**: $codeword_{s2} = (codeword_{s1} + 1) \ll (l_{s2}\check{} - l_{s1})$

Algoritmul începe cu $codeword_{s1}$ = zero, la pasul 1. Repetăm pentru toate simbolurile de intrare și generăm următorul *codeword* utilizând formula de la pasul 2.

3.3.2 Construcţia tabelului de decodare FCHD

Propunem o soluție rapidă de decodare canonică Huffman implementată folosind un singur tabel numit tabel de decodare FCHD. Acesta necesită un timp minim pentru construcție și va fi stocat în fișierul final de ieșire.

Pentru a implementa algoritmul nostru FCHD, împărţim fluxul de ieșire Huffman în segmente care urmează două reguli:

- A. **Regula Greedy**
- B. **Regula de Limitare**

Fie rezultatul codificării mesajului s0 . . . s3 cu codurile canonice Huffman din Tabelul 3.1.

Folosim valoarea unui segment din fluxul de ieşire Huffman, notat C_{bv}, pentru a genera un index în tabelul de decodare FCHD. Acest index este generat dintr-o valoare de segment dar şi dintr-un rest de biţi R_{bv}, dacă există.

Indicele tabelului FCHD are întotdeauna lungimea de N biţi.

Folosind indexul tabelului FCHD, stocăm informaţiile suplimentare necesare pentru decodarea rapidă într-o locaţie a tabelului de decodare indicată de acest index.

3.3.3 Teste Experimentale

Decodorul nostru rapid FCHD a fost testat pe diferite tipuri de date, cum ar fi nucleotide, fișiere bitmap, fișiere text și binare.

3.4 Concluzii

Decodorul nostru FCHD, este proiectat pentru codurile canonice Huffman cu o lungime medie a cuvântului de cod de până la 12 biți. Este o soluție originală care abordează nevoia de algoritmi de codare entropică de mare viteză. Algoritmul a fost conceput pentru a avea o amprentă mică de stocare și memorie și îl considerăm potrivit și pentru medii cu resurse limitate, cum ar fi in diferite circuite electronice (ECU)[1] din diferite

[1] 1

Tabel 3.1 Frecvenţa simbolurilor si codeword-urile asociate

Index	Simbol	Frecventa	Lungime Cod	Cod Canonic	Cod Huffman
0	A	9	1	0	1
1	T	5	2	10	01
2	C	3	3	110	000
3	G	1	3	111	001

domenii, IoT sau Automotive. Am dovedit empiric că metoda noastră FCHD este capabilă să decodeze cu viteze de peste 2 GB/s.

[1]ECU înseamnă Electronic Control Unit, un mic dispozitiv electronic responsabil cu controlul unor funcții specifice. În Automotive, unitățile electronice de control (ECU) sunt dispozitive electronice situate în interiorul vehiculelor care controlează funcții specifice, cum ar fi controlul servodirecției, geamurile electrice, scaunele sau diverşi parametrii de motor.

Capitolul 4

Codarea Aritmetică Cvasi-Statica

Acest capitol se concentrează pe codificatorii de entropie, în special codificarea aritmetică. Începe cu prezentarea algoritmului aritmetic de codificare a datelor, urmată de descrierea unei versiuni cvasi-statice.

Propunem o nouă arhitectură pentru un codificator aritmetic numită Cvasi-Static. Spre deosebire de implementările clasice, codificatorul cvasi-static imparte fluxul de intrare in bufere mai mici și utilizează un model static pentru analiza statisticilor simbolurilor de intrare. Marele avantaj al unei astfel de abordări este viteza de codare mai mare obținută, cu prețul unei rate de compresie putin mai scazută.

Textul acestui capitol a fost publicat în lucrarea noastră de conferință *The Anatomy of a Quasi-Static Arithmetic Encoder* [129].

Deoarece algoritmii de compresie bazați pe dicționar precum Lempel-Ziv [115] conțin un nivel de redundanță în codificarea datelor, este important să reducem această redundanță folosind o metodă diferită, alta decât cea bazată pe codificarea sub-șirurilor comune. Prin urmare, orice metodă completă de compresie a datelor trebuie să includă și un codificator de entropie.

Compresia aritmetică a datelor, ca si codificator de entropie de facto, a trecut printr-un proces de îmbunătățire continuă de la prima sa apariție în celebrul articol „Arithmetic Coding for Data Compression" scris de Ian H. Witten, Radford M. Neal și John C. Cleary [109]. Această versiune a compresiei aritmetice este cunoscută sub numele de CACM Arithmetic.

Una dintre cele mai mari diferențe ale unui codificator aritmetic față de majoritatea codificatorilor de entropie, cum ar fi Huffman, este aceea că întregul mesaj (buffer, fișier sau stream de intrare) este codificat într-un singur număr foarte mare, care este ales cu grijă astfel incat să fie o fracție q în intervalul [0,0 ... 1,0).

Tabel 4.1 Statistical sequence model

Symbol	Frequency	Cumulated Frequency	Range [0, 1)
A	5	0	[0, 0.5)
B	3	5	[0.5, 0.8)
C	2	8	[0.8, 1.0)
SUM		10	

4.1 Codarea Aritmetică

Un mesaj sursă stocastic, este un mesaj finit de codat, compus din simboluri ce aparţin unui alfabet anume. După cum s-a menționat mai sus, codificatorul aritmetic preia mesajul și îl convertește într-un număr fracționar q aparținând intervalului [0,0 ... 1,0).

4.1.1 Fluxul de ieşire

În implementările practice ale compresiei aritmetice, un cod de iesire, denumit *codeword*, este construit progresiv.

Acest proces se bazează pe urmatoarea observatie: în intervalul [0,0 ... 1,0) orice numar care se află sub 0,5 are 0 ca MSB (biţii cei mai semnificativi) si orice numar care se află peste 0,5 are 1 ca MSB.

Când punctele finale LOW si HIGH se afla pe aceeasi parte a lui 0,5, (fie sub, fie peste), *codeword*-ul curent va avea cu siguranţă MSB-ul acelei laturi (fie 0, fie 1). În consecinţa, encoderul aritmetic va trimite in fluxul de ieşire un bit de 0 sau un bit de 1, in funcţie de valoarea MSB-ului.

4.1.2 Procedeul de decodare

La decodor, fluxul de biți este format din *codeword*-uri și pentru fiecare *codeword* se identifică intervalul care l-a creat. Decodorul trebuie să cunoască ţntregul model statistic, altfel o decodare corectă nu este posibilă.

4.2 Considerații practice

Având în vedere intervalul [0,0 ... 1,0) și modul rapid în care se micșorează, pare complet nepractic să se folosească acest interval pentru o implementare a codificării aritmetice. Folosirea [0,0 ... 1,0) are două defecte majore la implementarea codării aritmetice.

Fie *LOW* și *HIGH* capetele intervalului de codare. Frecvențele cumulate ale simbolurilor sunt calculate ca suma frecvențelor simbolurilor anterioare, iar valoarea per unitate definește raportul dintre interval și suma acestora.

Să presupunem că, pentru exemplul nostru, dorim să mapăm intervalul [0 ... 1) la [0 ... 20). Frecvențele simbolurilor și frecvențele cumulate sunt date în tabelul 4.1.

Frecvențele cumulate ne ajută să definim intervalul atribuit fiecărui simbol. Capetele intervalului atribuit simbolului S_i sunt calculate după cum urmează:

$$LOW[S_i] = LOW_0 + V_pU \times CumFreq[S_{i-1}] \tag{4.1}$$

$$HIGH[S_i] = LOW_i + V_pU \times CumFreq[S_i] \tag{4.2}$$

unde V_pU este în cazul nostru 20/10 și LOW_0 este punctul final din stânga intervalului scalat inițial. Aici indexul *i* specifică ordinea simbolurilor în interval. Fără o mapare bună, adică fără utilizarea V_pR, am avea o modificare a probabilităților față de estimarea inițială.

Aplicănd ecuaţia 4.1, capetele intervalului vor fi mapate la numere întregi: pentru simbolul 'A', intervalul [0, 0.5) este mapat la [0, 10], pentru simbolul 'B', intervalul [0.5, 0.8) devine [10, 16], pentru simbolul 'C', intervalul [0.8, 1) este mapat la [16, 20).

Limita superioară a intervalului mapat poate fi maxim 4.294.967.295 pe maşini cu procesoare de 32 de biţi (în bază 16 $0xFFFFFFFF$).

Rezultatul este o mapare pe intervale mult mai mari pentru a lucra cu numere intregi:

- $[0x00000000, 0x80000000)$ pentru simbolul „A"
- $[0x80000000, 0xCCCCCCCC)$ pentru simbolul „B"
- $[0xCCCCCCCC, 0xFFFFFFFF)$ pentru simbolul „C"

4.3 Actualizarea informaţiilor statistice în codificarea cvasi-statică

Codarea aritmetică tinde spre optimalitate atâta timp cât probabilitățile modelului sursă sunt egale cu probabilitățile simbolurilor care urmează să fie codificate.

Orice diferență reduce raportul de compresie. În implementările clasice, codificatorul aritmetic folosește un model dinamic care își adaptează statisticile odată cu procesarea fiecarui simbol de intrare. Toate statisticile sunt calculate pe baza simbolurilor întâlnite anterior.

Deoarece toate simbolurile anterioare sunt deja cunoscute la pasul de codificare *n*, aceste statistici vor fi actualizate în același mod și de către decodor.

4.4 Rezultate Experimentale

Au fost efectuate teste pe trei tipuri de date.

- Prima serie de teste este din Calgary Corpus [1], Dimensiunea primei serii de teste este de 6.285.846 de octeți.
- A doua serie este un *fișier text formatat*. Dimensiunea fișierului este de 11.924.676 de octeți.
- A treia serie de teste este cod binar și constă dintr-o colecție de fișiere executabile. Dimensiunea acestei serii este de 21.742.646 de octeți.

Cu cât buferul de intrare este mai mare, cu atât dimensiunea tabelului de frecvență „stocat" este mai mică în comparație cu dimensiunea finală a arhivei. Astfel, economiile de compresie tind să aiba un impact maxim. Aceasta explică cele mai bune rezultate de compresie pentru dimensiunile bufferului de 64 și 128KB. Dezavantajul utilizării unui tabel de frecvență unic pentru întreg bufer este o posibilă scădere a ratei de compresie din cauza nepotrivirii modelului static cu probabilitățile simbolurilor din fluxul de intrare.

4.5 Concluzii

Credem că această abordare a compresiei de date folosind metoda Cvasi - Statica este o soluție viabilă atunci când se preferă un codificator de entropie. O îmbunătățire drastică a vitezei cu un impact scăzut asupra ratei de compresie a fost observată în timpul testelor comparative cu alte două codificatoare aritmetice dinamice, făcând această abordare o soluție viabilă pentru un codificator de entropie, mai ales atunci când este preferată codarea aritmetică.

[1]The corpus was created in the late 80s by Timothy Bell, John Cleary and Ian Witten, in order to be used in their research paper "MODELING FOR TEXT COMPRESSION" [132] at University of Calgary, Canada. ACM Computing Surveys published the research paper in 1989; in 1990 the corpus was used in their book "Text compression"

Capitolul 5

Actualizarea de Software în Automotive

Pe lângă îmbunătățirile în ceea ce privește funcționalitatea, o actualizare software sau un patch, pot oferi remedieri la probleme majore de funcționalitate legate de diverse unități de comandă, de la cele care controlează funcționalități neesențiale cum ar fi informații personale de profil (PIA), setările individuale pentru pasageri, până la erori de *Infotainment*, *e-Call*, *City Stop* sau *Brake Assist*.

Software-ul auto trebuie să fie întotdeauna 100% fiabil. Actualizările software care instalează versiuni noi de software sau îmbunătățiri sunt singura caracteristică ce poate face acest lucru posibil!

Eliminând factorul uman din această ecuație, ar putea fi posibil ca majoritatea problemelor să fi fost remediate de o singura funcționalitate, si anume de către Actualizarea de Software (Software UPDATE Wired, pe fir sau OTA, adică fără fir).

Defectele din automobile au costuri enorme. Sancțiunile civile, daunele punitive alături de cheltuielile judiciare au depășit 230 de milioane de dolari anul trecut, numai în Statele Unite:

- Într-un an, statistic, 30,000 de oameni au murit pe autostrăzile din Statele Unite;
- În fiecare an, se estimează că numai coliziunile de trafic au costat aproximativ 230 de miliarde de dolari în costuri medicale, de asigurare și in alte pierderi de productivitate numai în SUA. Vezi Fig. 5.1.

La fel ca un smartphone, care are capacitatea de a descărca cea mai nouă versiune de sistem de operare prin rețeaua GSM sau folosind conectivitate wireless fără a fi conectat fizic sau printr-un cablu într-o rețea, același lucru poate fi aplicat si unui model nou de automobil sau dispozitiv IoT.

Aici, un sistem de gestionare de la distanță ca şi centru de comandă pentru actualizarea de software ar fi o caracteristică foarte utilă, dar și extrem de populară datorită multitudinii de avantaje pe care le oferă.

Există o nevoie stringentă de a descărca și instala cu succes cele mai noi update-uri de software în majoritatea circuitelor electronice care alcătuiesc întregul sistem auto. În

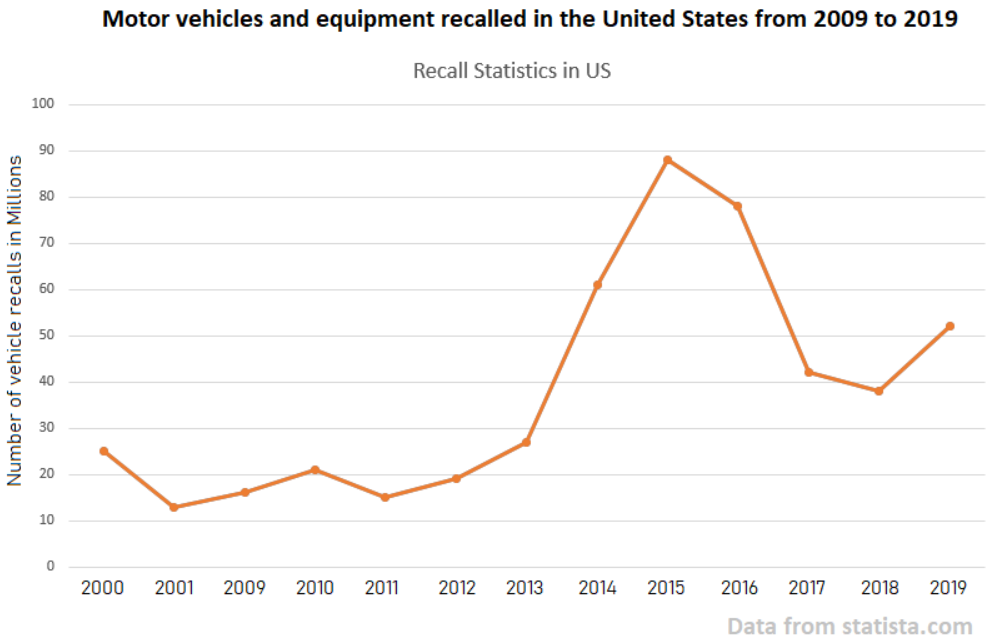

Figura 5.1 Statistici de rechemări (recall) in service doar in Statele Unite.

industria Auto, acest sistem la distanță, care gestionează actualizarea software-ului fără un conector fizic la o rețea sau un dispozitiv de stocare, este intitulat FOTA (Firmware Over-The-Air) sau mai simplu, OTA (Over-the-Air).

Sistemul la distanță care utilizează un conector de legătură cu fir la o rețea sau la un dispozitiv de stocare, se numește OTC sau Wired Update (Actualizare Software prin fir).

Impactul actualizărilor software OTA în industria Auto este extrem de mare.

Din punctul de vedere al proprietarului vehiculului, experiența sa este îmbunătățită constant atunci când este implicată actualizarea de software OTA. Nu sunt necesare opriri la reprezentanță, nu apar alte cheltuieli dacă mașina este în afara garanției si un alt exemplu ar putea fi că un nou sistem de *Infotainment* poate fi disponibil și gata de utilizare în mai puţin de câteva ore.

Absența actualizarilor OTA are un impact mult mai mare deoarece actualizările software critice sau erorile din sistem rămân nerezolvate până la o nouă oprire la reprezentanță, ceea ce poate poate periclita sistemul automobilului și poate afecta siguranţa pasagerilor.

5.1 Actualizarea Manuală a Firmware-ului în Automotive

Când vine vorba de actualizarea manuală a firmware-ului circuitelor electronice dintr-un automobil, ne bazăm pe faptul că în mare parte, toate aceste circuite electronice sunt

interconectate prin diverse protocoale de rețea și magistrale, cum ar fi *CAN*, *Ethernet*, *LIN*, *FlexRay* sau *MOST*. O actualizare manuală a firmware-ului se bazează și pe un modul principal, de obicei numit *Modul de Diagnoză*. Acesta interconectează și comunică cu aproape toate circuitele prezente intr-un automobil. Modulul de Diagnoză exportă diferite funcționalitati, cum ar fi: *Transfer Date*, *Scriere Date*, *Citire Date*, *Verificare Memorie*, *Citire Coduri de Eroare DTC*, *Eliminare Coduri de Eroare DTC* etc. Protocoalele de diagnoză utilizate sunt KWP-2000, OBD sau UDS (Unified Diagnostic Services). Protocolul UDS, descris de ISO-14229 este acum un standard și este utilizat de majoritatea producătorilor si furnizorilor de componente automotive, OEM şi Tier-1.

5.2 Actualizarea Firmware-ului în mod OTA

Actualizarea Firmware-ului în mod OTA (Over-the-Air) poate fi realizată în trei pași. Aceasta combină software-ul de pe un server, care creează pachetul de actualizare software, cu software-ul client, care instalează actualizarea pe maşină sau pe computerul clientului.

Primul pas

Software-ul care rulează pe server este primul pas într-o actualizare FOTA, deoarece atunci se creează pachetul de actualizare software. Există două versiuni de software care sunt generate în acest pas:

- Pachetul de diferențe binare, care conține doar software-ul ce va remedia erorile de sistem.
- Noua versiune de software completă, care deja conține remedierea erorilor și/sau noile funcții software.

Al doilea pas

Al doilea pas în acest proces este distribuirea pachetului de actualizări. In prima faza, pachetul securizat de actualizare software este stocat pe o platformă de distribuţie a update-urilor de software.

Pasul final

În acest pas, un software specializat şi securizat, aflat pe maşina clientului, va executa și realiza instalarea reală a noii versiuni de software, prin aplicarea pachetului de actualizări creat pe server, la primul pas.

5.3 Pachetele de Actualizări FOTA

Un proces FOTA se bazează pe pachete specifice numite *Containere*. Pentru a distribui o actualizare de software utilă circuitelor electronice aflate intr-un automobil, este necesară crearea unui *Container*, numit pe scurt *SWUP* sau *SWFK*.

5.4 Conceptul de Fragmentare a Datelor

Acest concept nou este legat de capacitatea modulelor principale de a efectua operaţie de instalare, în paralel cu operaţia de *download* a datelor noi.

Aceste segmente de date ar putea conţine fie versiunea completă a software-ului, fie datele binare delta sau fișierele de configurare necesare aplicării unei actualizări de software.

Verificarea Instalării

In timpul procesului de actualizare există mai multe verificări ale semnăturilor digitale căt și ale celor de integritate, numite CRC32.

- Verificarea CRC-ului pentru containerul descărcat din reţea (SWDL).
- Verificarea CRC-ului pentru fiecare segment de date binare transferate.
- Verificarea CRC-ului pentru modulele transferate de tip Container.

Actualizarea de Software Selectivă

Fișierele finale de configurare de tip Container sunt generate în timpul procesului de compilare pe baza tipului de sesiune de compilare (de Dezvoltare sau de Producție) și se bazează pe un șablon de configurare.

O actualizare selectivă se bazează pe tipul de model al unității principale hardware și de asemenea, pe nivelurile de complexitate ale hardware-ului, pe zona de distribuţie a hardware-ului princial (ECE, CHN, USA, etc). De exemplu, un M625 Varianta 3 ar putea fi proiectat doar pentru piețele ECE și SUA și vine cu un modul hardware TV-Tuner (varianta 1 și 2 poate să nu conțină acest ECU TV Tuner).

Streaming-ul Containerelor de Date

Unii producători OEM sau furnizori de nivel 1 (Tier-1) au implementat un sistem de actualizare software capabil să transmită datele în fluxuri de blocuri pentru instalarea și procesarea în paralel a acestora.

Dispozitivul principal și anume modulul de diagnoză, va trimite date unui circuit electronic pe bucăti, iar datele sunt primite și instalate în paralel de către aplicația binară securizată de instalare a software-ului.

Capitolul 6

Algoritmii de Diferențiere a Datelor folosind Compresia Referenţiala

Acest capitol se concentrează pe algoritmii de diferențiere a datelor binare și, în special, pe compresia referențială (RC). Se începe cu prezentarea generică a compresiei datelor referențiale, urmată de descrierea datelor adecvate și a cazurilor de utilizare pentru această tehnică, construirea unui dicționar referențial și posibile scenarii de atac din partea hackerilor.

Contribuția noastra la subiect constă într-un algoritm original de diferențiere a datelor bazat pe compresia diferențială care joacă și rolul unei distanțe generalizate similare distanței de compresie normalizate (NCD).

Prin proiectare, acest algoritm poate fi potrivit pentru un mediu cu resurse limitate și ar putea fi utilizat în zone industriale, cum ar fi industria Auto.

Algoritmul propus este inovator sub următoarele aspecte:

- Este un algoritm de diferențiere binar bazat pe NCD, conceput pentru medii cu resurse limitate, potrivit pentru IoT sau industria Auto.
- Implementează o nouă versiune a algoritmului LZ77, modificată astfel încât să lucreze cu diferite dimensiuni de bloc și dimensiuni parametrizate ale fluxului de intrare.
- Prezintă capacitatea de a comprima direct diferenţele dintre două blocuri de date aparținând celor două fluxuri de intrare diferite.

O parte a acestui capitol a fost publicată în lucrarea de conferință *An innovative algorithm for data differencing* [130].

Când software-ul nu funcționează conform specificațiilor de proiect sau când sunt necesare funcții noi și efortul de a instala o noua versiune in totalitate este prea mare, o actualizare de software este soluția optimă. O actualizare software este un manual de instrucțiuni reprezentat ţntr-un mod compact sub forma unui fișier interpretat de aplicația

care instalează actualizarea. Conține comenzi simple, cum ar fi *ADD* / *CUT* / *INSERT* / *COPY*. Aceste comenzi instruiesc decodorul cum să reconstruiască noua versiune a fișierului pornind de la fișierul inițial și efectuând operaţii conform instrucţiunilor din actualizare: inserări, adăugiri, tăieri sau copieri specifice de octeţi.

6.1 Actualizările de Software şi Diferenţierea Datelor

Aplicarea unei actualizări îmbunătățește software-ul nu numai prin actualizarea informațiilor despre versiune, ci și prin schimbarea codului sau a datelor. Prin urmare, actualizările sau patch-urile de software constau în suma tuturor diferențelor dintre versiunea veche și cea îmbunătățită ale aceluiași software, notate ca *sursa* și *destinaţie*, reprezentate ţn cel mai compact mod posibil. Algoritmii care pot produc aceste actualizari sunt algoritmii de diferențiere a datelor.

Compresia referenţială este una dintre tehnicile care pot fi utilizate pentru diferenţierea datelor binare.

6.2 Compresia de Date Referenţiala

Compresia referențială este o subclasă de compresie bazată pe dicționar. O funcționalitate a algoritmului de compresie bazată pe dicționar este de a înlocui subșirurile de simboluri frecvent întâlnite din fluxul de intrare în perechi <distanţa, lungime>. Se împarte fluxul de intrare în bufer de istoric LZ_HISTORY și bufer de anticipare LZ_LOOKAHEAD; orice porțiune de subșir a unei perechi <*distanta, lungime* > indică o copie a acesteia în partea istorică a buferului. Codificarea de ieșire constă dintr-un triplet $\langle d, l, s \rangle$ format din *distanta*, *lungime*, *simbol*, unde *simbol* este primul caracter nepotrivit (numit si litaral) care urmează perechii de potrivire distanţa-lungime $\langle d, l \rangle$.

6.2.1 Moduri de Operare ale Compresiei Referenţiale

Compresia referențială implică două dicționare și funcționează în două moduri atât în procesul de codificare, cât și în cel de decodare. Acestea sunt modurile interne și externe.

Un dicționar intern este utilizat atunci când compresia de referință funcționează în modul intern. Dicționarul intern care începe fără date (gol) se acumulează în zona LZ_HISTORY în timp ce indicatorul LZ curent avansează (CLZP).

În modul extern, Compresia Referențială folosește un dicționar specializat. Acest dicționar specializat, în majoritatea cazurilor, este un dicționar extern (vezi Fig. 5.1) care acționează ca o bază de date specializată pentru anumite subșiruri, dar oferă și un mecanism de criptare care va fi explicat mai jos.

Deoarece decodorul trebuie să emuleze pașii codificatorului, în acest mod de operare decodorul necesită informaţia suplimentară notată cu *token*, pentru a distinge la ce

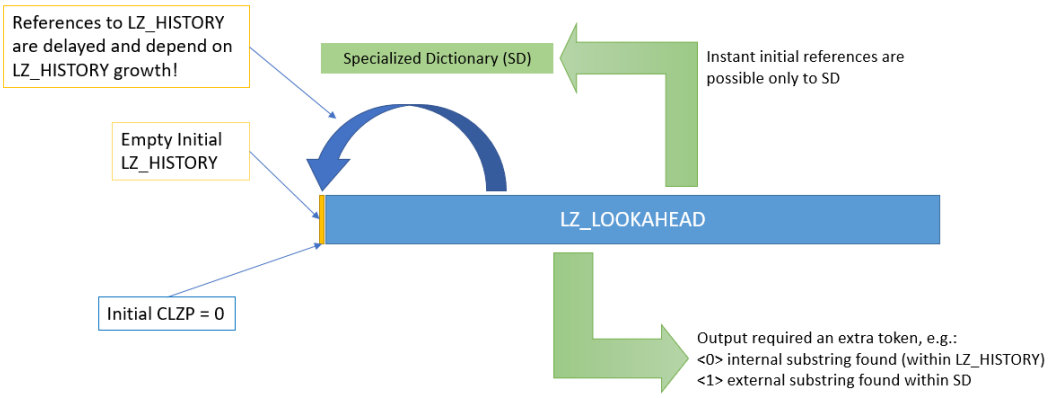

Figura 6.1 Compresia Referenţială cu dicţionar extern specializat.

dicționar se face referire ţntr-o anumită potrivire. În acest mod, codificarea de ieșire constă intr-un cvadruplu $\langle t, d, l, s \rangle$ format din *token*, *distanta*, *lungime* și *simbol*, unde *token* reprezintă indexul dicționarului care a fost folosit in aceasta potrivire reușită.

6.3 Un Nou Algoritm de Compresie Referenţială

Noul nostru algoritm de diferențiere a datelor creează un fișier de date diferit de cele două fișiere de lucru. Cele două fișiere de lucru se numesc *sursa* și *țintă*. $f1$ denotă de obicei fișierul sursă (fișierul vechi), numit si fișier dicționar. $f2$ denotă fișierul țintă (fișierul nou).

Am proiectat algoritmul nostru de diferențiere pentru a utiliza algoritmul LZ77. Așa cum este prezentat mai sus, CURRENT_POINTER-ul împarte fluxul de intrare în două porțiuni, partea de istoric, LZ_HISTORY și partea de anticipare, buferul LZ_Lookahead.

6.3.1 Modul de operare al unui algorithm de compresie diferential

După cum am menționat mai sus, elaborăm un algoritm care funcționează în medii cu resurse limitate. Pentru a ne atinge acest obiectiv, fluxul de intrare este partiţionat în bufere sau bucăți mai mici.

Lucrăm cu două fluxuri de intrare, X și Y, unde X este fluxul de intrare reprezentat de fișierul sursă $f1$, în timp ce Y, este tot un flux de intrare, dar este reprezentat de fișierul țintă $f2$.

Mai mult, trebuie să definim o funcție de similitudine care, în termenii noștri, calculeaza numărul comun de subșiruri între X și Y.

Numim funcția noastră BEST. Definim fluxul de intrare, I, astfel:

$$I = xy : x \in X, y \in Y \tag{6.1}$$

Am ales LZ77 ca algoritm de lucru principal, notat cu Z. Înlocuim $Z(x|y)$ cu funcția BEST, care devine astfel funcția noastră de similitudine.

Prin notatia $BEST(data\_bufer\_k)$ = i, calculam pentru bufer-ul k din fișierului $f2$, cel mai asemănător bufer din fisierul $f1$, si obtinem index-ul i al acestui bufer. Cu cât este mai mare asemănarea celor două bufere, cu atât este mai mare raportul de compresie la comprimarea celor două.

Algoritmul procedează după cum urmează:

- **Pasul 1**: Definim valorile N și K;
- **Pasul 2**: Împărțim fluxul inițial $f1$ în bufere de dimensiunea maximă K;
- **Pasul 3**: Împărțim buferul țintă $f2$ în bucăți de dimensiunea maximă N și citim primul bufer;
- **Pasul 4**: Calculam $BEST(data\_bufer\_k) = i$; rezultatul acestei functii este indexul i; citim bufer-ul i din $f1$;
- **Pasul 5**: Începem procesarea bufer-ului i din $f1$;
- **Pasul 6**: Finalizam codarea buferului k din $f2$ cu buferul i din fișierul $f1$, dat de funcția $BEST(data\_bufer\_k) = i$;
- **Pasul 7**: Repetam pasul 6 până când $f2$ este epuizat;

Am conceput o strategie pentru funcția noastră BEST și, în această strategie, calculăm toate valorile posibile ale asemănărilor dintre bufer-ul i și k.

Acest lucru este posibil din proiectare si a fost gândit pentru a obține cel mai bun rezultat posibil. Pentru orice bufer k din fișierul $f2$, încercăm toate posibilitățile prin aplicarea funcției de similaritate pentru toate bucățile de fișier i din $f1$.

În această strategie analizăm toate buferele din fișierul $f2$ față de alte bufere din același fișier și, de asemenea, față de buferele din fișierul $f1$.

Comparăm algoritmul nostru NCD cu software comercial, DeltaMAX (produs de Indigo Rose Corp.) și x3Delta.

Tabel 6.1 Cum se compară algoritmul nostru de diferențiere cu codificatoarele delta binare existente pe piață.

	BEST(f1,f2)	DeltaMAX	X3 Delta
Log Files	0.41	0.35	0.12
Binare Mingw	30.83	63.96	46.55
Pachet de instalare SW	54.42	63.07	51.21

6.3.2 Concluzii

Algoritmul nostru de compresie diferențiala este capabil să ofere rate de compresie chiar cu 50% mai bune decât software-ul comercial de diferențiere a datelor cu care l-am comparat. Rezultatele din pachetul software Windows arată un fișier delta cu 13,72% mai mic decât software-ul DeltaMAX, produs de Indigo Rose Corp. Când sunt aplicate pe binarele compilatorului *mingw*, rezultatele algoritmului nostru de diferențiere sunt cu 33% mai bune decât DeltaMAX.

Capitolul 7

Diferenţierea Datelor

Acest capitol se concentrează, de asemenea, pe diferenţierea datelor binare și, în special, pe algoritmii de compresie bazaţi pe dicționar, potriviţi pentru software-ul de diferențiere a datelor. Începe cu prezentarea generică a scenariilor de actualizare, urmată de descrierea software-ului care ruleaza pe un server si care crează fişierul binar delta și, de asemenea, de descrierea software-ului client care realizează instalarea si actualizarea software-ului pe o masina client.

De asemenea, este prezentată descrierea unui algoritm nou și original bazat pe LZ77. Sunt prezentate trei metode de creare a diferenţelor de date binare, în mare parte concepute pentru a se adapta diverselor scenarii, pornind de la o procesare mai rapidă la scenarii concepute exclusiv pentru a obține cele mai bune rate posibile de compresie a datelor.

Contribuțiile originale pe tema diferențierii binare bazate pe algoritmi modificați de compresie a datelor sunt:

- Un algoritm original pentru diferențierea datelor binare bazat pe Lempel Ziv '77 modificat [115].
- Algoritmul a fost proiectat pentru medii cu resurse reduse, ceea ce îl face potrivit pentru industrii precum IoT sau Auto.
- Algoritmul de decodare este imbunatațit.
- Dezvoltarea a trei strategii de lucru pentru acest algoritm, pentru a se adapta diverselor cazuri de utilizare ale acestui algoritm.

Acesta este, de asemenea, un algoritm original, nu o îmbunătățire a unei idei prezentate in Prior Art.

Textul acestui articol a fost publicat în următoarea lucrare de conferința: *A Hybrid Data-Differencing and Compression Algorithm for the Automotive Industry* [131].

7.1 De ce diferențierea datelor

Cele mai multe probleme ale vehiculelor sunt rezolvate de către majoritatea producătorilor de autovehicule si echipamente originale (OEM) prin actualizarea periodică a software-ul dintr-o varietate de motive, chiar si altele decât remedierea erorilor software.

Aceste actualizări de software sunt obligatorii pentru ca sistemele din automobile să funcționeze in parametrii normal acceptabili.

7.2 Folosirea Diferentierii de Date

În ultimii ani, diferite industrii au început să utilizeze tehnici de codificare delta pe scară largă, de la stocarea, indexarea și indexarea datelor si informațiilor despre genom, la cod sursa[56].

O actualizare de cod binar bazat pe diferențe binare este, de asemenea, utilizat în industria auto.

După revizuirea cu atenție a celor mai importante articole, soluții și idei, Curtea a constatat că majoritatea soluțiilor disponibile publicului nu adresa de faptul că, indiferent de cât de bun delta dislocate pe plan intern algoritmul este, va exista întotdeauna un fel de redundanță delta algoritmul nu se adresează.

Acest lucru se datorează pur și simplu faptului că, prin design, un algoritm delta nu este un algoritm de compresie a datelor. Este un algoritm de de-duplicare la cele mai bune de abilitățile sale.

7.3 Algoritmul Keops

Algoritmul nostru inovator, numit Keops, derivă din binecunoscutul algoritm de compresie a datelor LZ77 [29] pe care îl folosește ca metodă internă.

LZ77 realizează compresia datelor prin împărțirea fluxului de date de intrare, în două porțiuni, bufer sursa si bufer destinatie; datele sunt împărțite de catre un pointer de procesare curent numit pointer curent sau pointer de compresie *cp*. Cele două secțiuni se numesc LZ77 History și LZ77 LookAhead. Ne vom referi la aceste bufere sub forma bufer sursa si bufer țintă (sau destinație).

7.3.1 Fisierul Delta

În Keops, encoderul LZ77 acționează la fel în buferele „sursă” și „țintă”, dar spre deosebire de *vcdiff*, de exemplu, LZ77 instruiește decoderul sa reconstituie conținutul buferului „ținta” folosind tripletul $< d, l, c >$. Astfel, se realizează si compresie în interiorul fișierul binar delta. Diferențele față de structura originală LZ77 sunt că buferul

sursă constă în întregime din versiunea veche a fișierului și buferul destinație este în întregime alcătuit din versiunea mai nouă.

7.3.2 Strategii de Parsare a Buferelor

LZ77, așa cum este aplicat în Keops, nu este o "distanță" în adevăratul sens al cuvântului. Aceasta nu este simetrică, nu este nulă atunci când buferele sursa si destinaţie sunt identice și nu satisface neapărat formula similitudinii prin inegalitate. În consecință, vorbim de o distanță generalizată.

Prezentăm trei strategii de lucru cu buferele sursa si destinatie. Aceste strategii sunt proiectate pentru optimizarea timpului de compresie sau a ratei.

Strategia 1 la 1 (Optimizarea timpului de codare)

Când un set de modificări este conceput pentru a actualiza sau îmbunătăți un software sau un fișier de date, acesta este numit patch. Patchurile sunt de obicei concepute pentru a îmbunătăți funcționalitatea unui program sau pentru a remedia rapid un defect.

Strategia Brute-Force (Optimizarea Ratei de Delta)

Când diferențele dintre fișierul vechi și cel nou sunt numeroase, Strategia 1 la 1 nu mai poate oferi o solutie bună pentru o rata de compresie ridicata. Trebuie remarcat faptul că in cazul adăugirilor, modificărilor sau substituirilor de cod, maparea 1 la 1 nu se mai păstrează, deoarece în această situație fișierele sunt mult desincronizate.

Strategia Flexy

În cazul in care datele de actualizat conțin numeroase diferențe, s-a observat ca bufere similare se pot afla în vecinătatea celui considerat bufer țintă. Acest lucru se explică prin faptul că desincronizarea este produsâ atât prin eliminarea cât și prin adăugarea blocurilor de cod nou sau de date; astfel, blocurile se deplasează nu se acumulează.

7.4 Rezultate Experimentale

Am testat implementarea software Keops pe cinci tipuri de date. Prima serie de teste contine compilatorul binar pentru platforma Windows *mingw*.

A doua serie de teste constă din fișiere text formatate, reprezentând loguri si date colectate de software-ul de inregistrare a comportamentului automobilului, concatenate într-un singur fişier. Pentru al treilea pachet de testare, am ales două imagini binare dintr-o colecție de fişiere binare de imagine pentru ECU-uri utilizate în mod special în proiecte Autotive. *Replay* a fost al patrulea pachet de testare, care conține un software

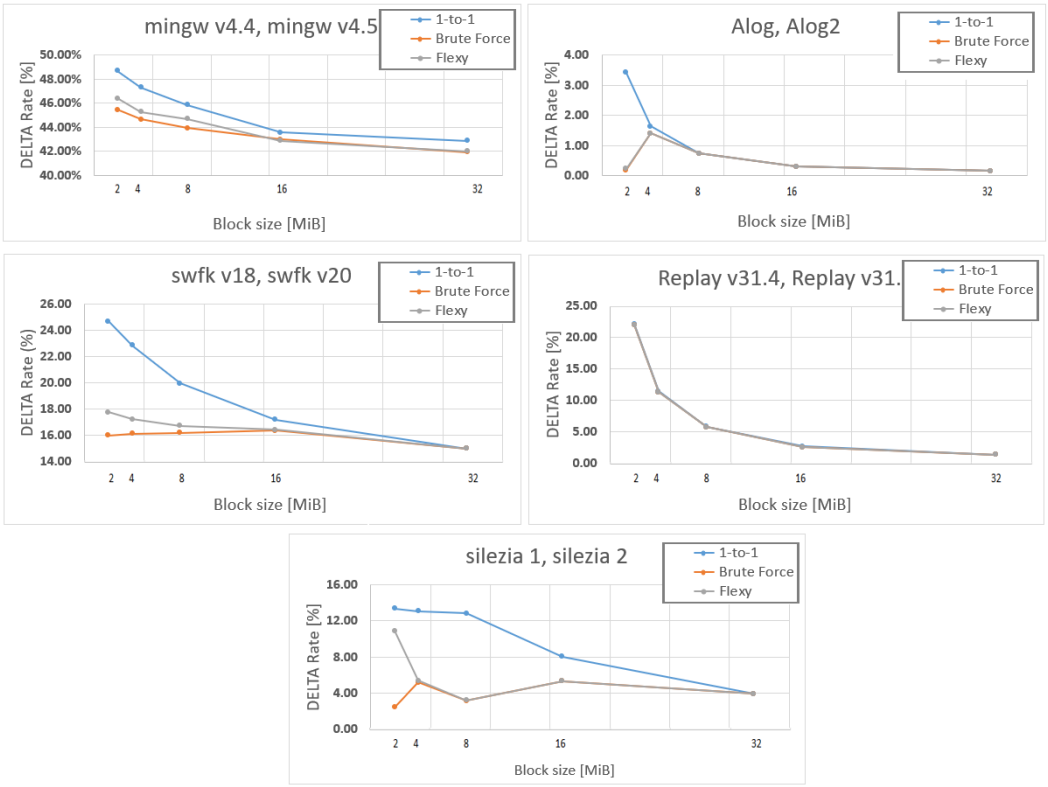

Figura 7.1 Rata de Delta vs. dimensiunea blocului.

specific de media. Am folosit exemple reale ale acestui software, versiunile 31.2, 31.4 și 31.5.

Pentru ultimul pachet de testare, am ales fișierele *Silezia* Corpus, o colecție binecunoscută de fișiere de text în limba engleză și date binare utilizate în testele de compresie a datelor [35].

Trei din cinci pachete de testare—*ALog*, *swfk* și *Replay*—sunt specifice mediului Automotive.

7.4.1 Rata de Compresie

În ceea ce privește strategiile, graficele din Fig. 7.1 arată că strategia Brute Force este în general cea mai bună dacă scopul final este o compresie mai bună. Totuși, acolo sunt excepții, de exemplu, pentru pachetul *Replay*, pentru care toate strategiile au dat aceeași rată delta sau pentru pachetul *Alog*, pentru care s-au comparat strategia Brute Force si Flexy.

După cum arată rezultatele, ratele delta ale fişierelor de test *Alog* și *silezia* sunt foarte mari, depășind, în unele cazuri, 99%, adică, dimensiunea fisierului delta (fișierul de diferențiere binară) este mai mică de 1% în comparație cu dimensiunea fisierului țintă.

7.4.2 Timpul de Codare

Timpul de codare depinde de strategia aleasă. Strategia Brute Force și Flexy, implementează o căutare extinsa pentru blocurile similare dintre $f1$ si $f2$ si prin urmare execuţia acestor strategii durează mai mult timp. În medie, metoda Brute Force a avut durate si de patru ori mai mari, în timp ce metoda Flexy a fost de numai 2.5% mai lenta decat strategia 1 la 1. Evident, cei mai mari timpi de codare corespund codărilor cu blocuri de dimensiuni reduse, indiferent de strategia aleasă.

7.4.3 Timpul de Decodare

În timp ce timpii de codare sar de la 3.75 secunde la 1253 secunde, decodarea este mult mai rapidă. În funcție de dimensiunea blocului, tipul de fișier și de strategia aleasa, acesta poate avea valori între 0,18s și 10,46 secunde. Mai mult, se observă că timpii de decodare scad odată ce dimensiunea blocului crește, cu excepția blocurilor de 32MB. Acest lucru ar putea fi legat de faptul că buferele mai mari tind să permită gasirea mai multor sub

7.5 Concluzii

Keops este o soluţie potrivită pentru un mediu de lucru cu memorie scăzută sau medie care are nevoie de operații de actualizări de software la viteze mari. In funcție de strategia aleasă, Keops ofera cu succes o soluție mai buna in ceea ce priveste rapoartul de compresie şi dimensiunile fisierului delta folosit in actualizarea software-ului, decăt soluţiile comerciale prezentate in acest capitol.

Capitolul 8

Concluzii ale tezei mele de doctorat

Această teză prezintă cercetările noastre privind algoritmii de compresie de date, proiectaţi pentru o compresie fără pierderi în medii cu resurse limitate și, prin urmare, potriviţi pentru industria Auto şi IoT.

Cercetarea a fost efectuată la două nivele: îmbunătățirea algoritmilor de codare și proiectarea unei aplicații complete pentru compresia datelor de tip *BigData* folosind metode de diferențiere a datelor sau compresia delta.

În prima parte a tezei, pornind de la algoritmii de codare a entropiei de ultimă generație, cum ar fi Huffman Canonic sau Compresia Aritmetica Statică, am propus câteva soluții originale pentru decodarea rapidă a codurilor canonice Huffman și o soluție rapidă pentru codificarea aritmetică, numită *Quasi Static Arithmetic Encoder*.

În a doua parte a tezei, am propus o nouă metodă de diferențiere a datelor potrivită pentru un mediu cu resurse limitate. Deși este proiectat pentru viteza ridicata de decompresie ca primă opțiune, credem că acesta ar putea fi util pentru industria Auto. Algoritmul nostru ofera rezultate cu viteze de decompresie mai mari decât primele două produse comerciale de pe piață cu care a fost comparat.

Soluțiile propuse au fost examinate amănunțit prin analiză teoretică, precum și prin utilizarea unui software dedicat si creat pentru a valida rezultatele. Fiecare soluție propusă a fost însoțită de o aplicatie software și de teste exhaustive cu acest software.

8.1 Contribuţii Originale

Contribuțiile originale dezvoltate pe această teză, constau in următoarele:

- Backtrace al unui algoritm nedocumentat RoLZ, din note și descrieri ale algoritmilor LZP și RKive; îmbunătățirea algoritmului RoLZ față de versiunea originală sugerată de Charles Bloom și Mark Taylor (Sectiunea 2.3, Lucrarea de Conferință [128]).

- O nouă metodă de decodare Huffman canonică, cu capacitatea de a procesa 8 până la 12 simboluri într-un singur ciclu de decodare. Acesta cuprinde noi metode de

creare a arborelui Huffman clasic și de stocare a *codeword*-urilor. Este un algoritm original, nu o îmbunătățire a lucrărilor anterior prezentate (Sectiunea 3.2, Lucrarea de Conferință [131]).

- Un algoritm îmbunătățit pentru codificare aritmetică numit codificator aritmetic cvasi-static. Algoritmul a fost proiectat astfel încât să îmbunătățească viteza de codificare prin minimizarea numărului de operatii necesare procesării unui simbol. (Sectiunea 4.3, Lucrarea de Conferință [128]).

- Un algoritm de diferențiere binar bazat pe o distanță generalizată precum NCD, conceput pentru medii cu resurse reduse, potrivit pentru IoT sau industriile auto. Algoritmul include o versiune de LZ77 modificată si se bazeaza pe strategii diferite de procesare a blocurilor și care parametrizeaza diferit fluxul de intrare (Sectiunea 6.3; Document de Conferință [130]).

- Un algoritm original pentru diferențierea datelor binare pentru o decodare mai rapidă în medii cu resurse reduse. Include trei strategii de lucru pentru a se adapta diferitelor cazuri de utilizare în industriile IoT și auto (Sectiunea 7.3, Lucrarea de Conferință [127]).

8.2 Lista de publicatii

8.2.1 Articole de revista

1. S. Belu and D. Coltuc, "A Hybrid Data-Differencing and Compression Algorithm for the Automotive Industry", in *Entropy*, (IF 2.738, Q2), 24(5), 574, 2022, https://doi.org/10.3390/e24050574, WOS:000803286100001.

8.2.2 Lucrari de Conferinta

1. S. Belu and D. Coltuc, "Fast Huffman Canonic Decoder". In *IEEE 14th International Conference on Communications COMM22*, Bucuresti, Romania, 2022.

2. S. Belu and D. Coltuc, "An innovative algorithm for data differencing". In *IEEE International Symposium on Electronics and Telecommunications (ISETC)*, Timisoara, Romania, 2020. DOI: 10.1109/ISETC50328.2020.9301053.WOS: 000612681000078.

3. S. Belu and D. Coltuc, "The Reduced Offset LZ Data Compression Algorithm". In *IEEE International Symposium on Signals, Circuits and Systems (ISSCS)*, Iasi, Romania, 2019. DOI: 10.1109/ISSCS.2019.8801741. WOS:000503459500013.

4. S. Belu and D. Coltuc, "The Anatomy of a Quasi-Static Arithmetic Encoder". In *IEEE 12th International Conference on Communications COMM18*, Bucuresti, Romania, 2018. DOI: 10.1109/ICComm.2018.8484263.WOS: 000449526000029.

8.2.3 Rapoarte de Cercetare pentru Doctorat

1. Raport de Cercetare pentru Doctorat Numarul 1/2018: "The Anatomy of a Quasi-Static Arithmetic Encoder", Conference Paper no. 4.

2. Raport de Cercetare pentru Doctorat Numarul 2/2018: "RoLZ - The Reduced Ofifsef LZ Data Compression Algorithm, Conference Paper no. 3.

3. Raport de Cercetare pentru Doctorat Numarul 3/2019: "ACHD - Advanced Canoniucal Huffman Decoder", Conference Paper no. 1.

4. Raport de Cercetare pentru Doctorat Numarul 4/2019: "An Innovative Data Differencing Algorithm", Conference Paper no. 2.

5. Raport de Cercetare pentru Doctorat Numarul 5/2020: "Analysis and Interpretation of Test Results for KEOPS, an Innovative Data Differencing Algorithm", 28 May 2020

8.3 Direcții viitoare de dezvoltare si cercetare

Am identificat câteva direcții viitoare de cercetare pentru îmbunătățirea tehnicilor și algoritmilor de compresie a datelor folositi si proiectați pentru industria Auto. Vom continua cu îmbunătățirea în continuare a algoritmilor de compresie a datelor de sine stătătoare, atât ca viteză, cât și ca rata de compresie.
Metode sofisticate de actualizare a software-ului trebuiesc abordate cu noi structuri de date, algoritmi și mecanisme de înlănțuire hash de ultimă generație, pentru a permite o procesare mai rapidă a datelor la un nivel rezonabil de utilizare a memoriei.

Un alt interes special va fi acordat metodelor heuristice utilizate în implementările noastre de compresie a datelor, modurilor de streaming specifice și un reglaj mai sofisticat în nucleul algoritmilor. Aceste metode vor fi folosite pentru a îmbunătăți și mai mult securitatea acestui software si în același timp, rata de compresie si viteza de decodare.

Bibliografie

[1] SecureDELTA SDK. Available online: https://agersoftware.com/securedelta_sdk.html (accessed on 13 January 2022).

[2] Guttman, L., A basis for scaling qualitative data, American Sociological Review, 9 (2), 1944, pp. 139–150, doi:10.2307/2086306, JSTOR 2086306.

[3] Jones, D.W, Application of Splay Trees to Data Compression, Communication of ACM, 18, 1988, pp. 996-1007

[4] Sleator, D.D. and Tarjan, R.E., Self-adjusting binary search tree, Communication of ACM, 32, 1985, pp. 652-686

[5] SecureDELTA Application with XtremeDELTA Engine. Available online: https://agersoftware.com/securedelta_app.html (accessed on 13 January 2022).

[6] xdelta org. Available online: http://xdelta.org/ (accessed on 13 January 2022).

[7] Richard C. Pasco, "Source coding algorithms for fast data compression", Stanford, CA 1976

[8] J. S. Vitter, Algorithm 673: Dynamic Huffman Coding, ACM Transactions on Mathematical Software, 15(2), June 1989, pp 158–167.

[9] Donald E. Knuth, "Dynamic Huffman Coding", Journal of Algorithm, 6(2), 1985, pp 163–180.

[10] Robert G. Gallager, "Variations on a Theme by Huffman", IEEE TRANSACTIONS ON INFORMATION THEORY, Vol. IT - 24, No. 6, Nov 1978

[11] RFC 3284—The VCDIFF Generic Differencing and Compression Data Format. Available online: https://tools.ietf.org/html/rfc3284 (accessed on 6 April 2022).

[12] Westerberg, E. *Efficient Delta Based Updates for Read-Only Filesystem Images: An Applied Study in How to Efficiently Update the Software of an ECU*; Degree Project in Computer Science and Engineering, KTH Royal Institute of Technology School of Electrical Engineering and Computer Science, Stockholm, Sweden, 2021.

[13] Falleri, J.R.; Morandat, F.; Blanc, X.; Martinez, M.; Monperrus, M. Fine-grained and accurate source code differencing. In Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering, Vsters, Sweden, 15–19 September 2014.

[14] Gerardo, C.; Luigi, C.; Massimiliano, P. *Identifying Changed Source Code Lines from Version Repositories*; RCOST—Research Centre on Software Technology Department of Engineering—University of Sannio Viale, Viale Traiano - 82100, Benevento, Italy, 2007.

[15] Zimmermann, T.; Weisgerber, P.; Diehl, S.; Zeller, A. Mining version histories to guide software changes. In Proceedings of the 26th International Conference on Software Engineering, Washington, DC, USA, 23–28 May 2004; pp. 563–572.

[16] Ying, A.T.T.; Murphy, G.C.; Ng, R.; Chu-Carroll, M. C. Predicting source code changes by mining revision history. *IEEE Tr. Softw. Eng.* **2004**, *30*, 574–586.

[17] Li, B.; Tong, C.; Gao, Y.; Dong, W. S2: A Small Delta and Small Memory Differencing Algorithm for Reprogramming Resource-constrained IoT Devices. In Proceedings of the IEEE INFOCOM 2021-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), Vancouver, BC, Canada, 10 May 2021.

[18] Christley, S.; Lu, Y.; Li, C.; Xie, X. Human genomes as email attachments. *Bioinformatics* **2009**, *2*, 274–275.

[19] Pavlichin, D.S; Tsachy, W. The Human Genome Contracts Again *Bioinformatics* **2013**, *29*, 2199–2022.

[20] Ochoa, I.; Mikel H.; Tsachy W. iDoComp: A compression scheme for assembled genomes. *Bioinformatics* **2015**, *31*, 626–633.

[21] Kuruppu, S.; Beresford-Smith, B.; Conway, T.; Zobel, J. Iterative dictionary construction for compression of large DNA data sets. *IEEE/AMC Trans. Comput. Biol. Bioinform.* **2010**, *1*, 137–149.

[22] Deorowicz, S.; Grabowski, S. Robust relative compression of genomes with random access, *Bioinformatics* **2011**, *21*, 2979–2986.

[23] Kuruppu, S.; Puglisi, S.J.; Zobel, J. Optimized relative lempel-ziv compression of genomes. In Proceedings of the Thirty-Fourth Australasian Computer Science Conference, Perth, Australia, 17–20 January 2011.

[24] Pinho, A.J.; Diogo P.; Sara, P.G. GReEn: A tool for efficient compression of genome resequencing data. *Nucleic Acids Res.* **2012**, *40*, e27.

[25] Wang, C; Dabing Z. A novel compression tool for efficient storage of genome resequencing data. *Nucleic Acids Res.* **2011**, *39*, e45.

[26] Wandelt, S.; Ulf, L. FRESCO: Referential compression of highly similar sequences. *IEEE/ACM Trans. Comput. Biol. Bioinform (TCBB)* **2013**, *10*, 1275–1288.

[27] Brandon, M.C.; Wallace, D.C.; Baldi, P. Data structures and compression algorithms for genomic sequence data. *Bioinformatics* **2009**, *14*, 1731–1738.

[28] Chern, B.G.; Ochoa, I.; Manolakos, A.; No, A.; Venkat, K.; Weissman, T. Reference based genome compression. In Proceedings of the IEEE Information Theory Workshop (ITW), Visby, Sweden, 25–28 August 2012; pp. 427–431.

[29] Ziv, J.; Lempel, A. A Universal Algorithm for Sequential Data Compression. *IEEE Trans. Inf. Theory* **1977**, *23*, 337–343.

[30] Yufei, T. Patricia Tries Lecture. KAIST. Available online: http://www.cse.cuhk.edu.hk/~taoyf/course/wst540/notes/lec10.pdf (accessed on 1 May 2013).

[31] Daelemans, W.; Bosh, A.V.D.; Antal, Weijters, T. IGTree: Using Trees for Compression and Classification. *Lazy Learn.* **1997**, 407–423

[32] Horspool, R.N. The Effect of Non-Greedy Parsing in Ziv-Lempel Compression Method, In Proceedings of the Data Compression Conference, Snowbird, UT, USA, 28–30 March 1995.

[33] Storer, J.A.; Szymanski, T.G. Data Compression via Textual Substitution. *J. ACM* **1982**, *29*, 928–951.

[34] Korn, D.G; MacDonald J; Mogul, J.C.; Vo, K.-P. The VCDIFF Generic Differencing and Compression Data Format. *RFC* **2002**, *3284*, 1–29.

[35] The Silesia Corpus. Available online: http://sun.aei.polsl.pl/~sdeor/index.php?page=silesia (accessed on 10 January 2022).

[36] Zlib Compression Library. Available online: http://www.zlib.org/rfc1950.pdf (accessed on 13 January 2022).

[37] Chen, X.; Li, M.; Ma, B.; Tromp, J. DNACompress: fast and effective DNA sequence compression. *Bioinformatics* **2002**, *10*, 51–61.

[38] APPNOTE.TXT-ZIP File Format Specification. Available online: https://pkware.cachefly.net/webdocs/casestudies/APPNOTE.TXT(accessed on 13 January 2022).

[39] Constructing a Binary Difference File. Available online: https://agersoftware.com/docs/securedelta_app_v2.56/43Creatingabinarydiffdeltafile.html (accessed on 13 January 2022).

[40] Raghavan, S.; Rohana, R.; Leon, D.; Podgurski, A.; Augustine, V. Dex: A semantic-graph differencing tool for studying changes in large code bases. In Proceedings of the 20th IEEE International Conference on Software Maintenance, Chicago, IL, USA, 11 September 2004; pp. 188–197.

[41] Grumbach,S.; Tahi, F. A new challenge for compression Algorithms: Genetic sequences. *Inf. Process. Manag. Int. J.* **1994**, *6*, 875–886.

[42] Cao, M.D.; Dix, T.I.; Allison, L.; Mears, C. A simple statistical algorithm for biological sequence compression. In Proceedings of the IEEE Data Compression Conference (DCC'07), Snowbird, Utah, 27–29 March 2007.

[43] Deorowicz, S.; Grabowski, S. Data compression for sequencing data. *Algorithms Mol. Biol.*, **2013**, *8*, 1–13.

[44] Deorowicz, S.; Danek, A.; Grabowski, S. Genome compression: A novel approach for large collections. *Bioinformatics* **2013**, *29*, 2572–2578.

[45] Dotzler, G.; Michael, P. Move-optimized source code tree differencing. In Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering (ASE), Singapore, 3–7 September 2016; IEEE: Piscataway, NJ, USA 2016.

[46] Fluri, B.; Wursch, M.; PInzger, M.; Gall, H. Change distilling: Tree differencing for fine-grained source code change extraction. *IEEE Trans. Softw. Eng.* **2007**, *33*, 725–743.

[47] Canfora, G.; Luigi, C.; Massimiliano P. Ldiff: An enhanced line differencing tool. In Proceedings of the IEEE 31st International Conference on Software Engineering, Vancouver, BC, Canada, 16–24 May 2009; IEEE: Piscataway, NJ, USA 2009.

[48] Frick, V.; Grassauer, T.; Beck, F.; Pinzger, M. Generating accurate and compact edit scripts using tree differencing. In Proceedings of the IEEE International Conference on Software Maintenance and Evolution (ICSME), Madrid, Spain, 23–29 September 2018; IEEE: Piscataway, NJ, USA 2018.

[49] Tsantalis, N.; Natalia N; Eleni S. Webdiff: A generic differencing service for software artifacts. In Proceedings of the 27th IEEE International Conference on Software Maintenance (ICSM), Williamsburg, VA, USA, 25–30 September 2011; IEEE: Piscataway, NJ, USA 2011.

[50] Maletic, J.I.; Michael L. C. Supporting source code difference analysis. In Proceedings of the 20th IEEE International Conference on Software Maintenance, Chicago, IL, USA, 11–14 September 2004; IEEE: Piscataway, NJ, USA, **2004**.

[51] Korn, D.G.; Vo, K.P. Engineering a Differencing and Compression Data Format. In Proceedings of the USENIX Annual Technical Conference, General Track, Berkeley, CA, USA, 10–15 June 2002.

[52] Nguyen, H.A.; Nguyen, T.T.; Nguyen, H.V.; Nguyen, T.N. Idiff: Interaction-based program differencing tool. In Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011), Lawrence, KS, USA, 6–10 November 2011; IEEE: Piscataway, NJ, USA, 2011.

[53] Müller, K.; Bernhard R. User-driven adaptation of model differencing results. In *International Workshop on Comparison and Versioning of Software Models (CVSM'14)*; Köllen Druck+Verlag GmbH: Bonn, Germany, 2014.

[54] Onuma, Y.; Nozawa, M.; Terashima, Y.; Kiyohara, R. Improved software updating for automotive ECUs: Code compression. In Proceedings of the IEEE 40th Annual Computer Software and Applications Conference (COMPSAC), Atlanta, Georgia, 10–14 June 2016.

[55] Ni, G.; Yan, Y.; Jiang, J.; Mei, J.; Chen, Z.; Long, J. Research on incremental updating. In Proceedings of the 2016 International Conference on Communications, Information Management and Network Security, Shanghai, China, 25–26 September 2016.

[56] Motta, G.; James G.; Samson C. Differential compression of executable code. In Proceedings of the Data Compression Conference (DCC'07), Snowbird, Utah, 27–29 March 2007.

[57] Belu, S.; Daniela C. An innovative algorithm for data differencing. In Proceedings of the 2020 International Symposium on Electronics and Telecommunications (ISETC), Timisoara, Romania, 5–6 November 2020.

[58] Deorowicz, S.; Agnieszka D.; Marcin N. GDC2: Compression of large collections of genomes. *Sci. Rep.* **2015**, *5*, 1–12.

[59] Kuruppu, S.; Simon J.P.; Justin Z. Relative Lempel-Ziv compression of genomes for large-scale storage and retrieval. In Proceedings of the International Symposium on String Processing and Information Retrieval, Berlin/Heidelberg, Germany, 13–15 October 2010.

[60] M. H. Mohammed, A. Dutta, T. Bose, S. Chadaram, and S. S. Mande, "DELIMINATE-a fast and efficient method for loss-less compression of genomic sequences," Bioinformatics, vol. 28, no. 19, pp. 2527–2529, 2012.

[61] Saha, S.; Sanguthevar, R. ERGC: An efficient referential genome compression algorithm. *Bioinformatics* **2015**, *31*, 3468–3475.

[62] Saha, S.; Sanguthevar, R. NRGC: A novel referential genome compression algorithm. *Bioinformatics* **2016**, *32*, 3405–3412.

[63] Liu, Y.; Peng, H.; Wong, L.; Li, J. High-speed and high-ratio referential genome compression. *Bioinformatics* **2017**, *33*, 3364–3372.

[64] Lempel Ziv Markov Algorithm. Available online: https://www.7-zip.org/sdk.html (accessed on 13 January 2022).

[65] Mary Shanthi Rani, "A New Referential Method for Compressing Genomes",International Journal of Computational Bioinformatics and In Silico Modeling, Vol. 4, No. 1 (2015): 592-596

[66] Alves F, Cogo V, Wandelt S, Leser U, Bessani A. On-Demand Indexing for Referential Compression of DNA Sequences. PLoS One. 2015 Jul 6;10(7):e0132460. doi: 10.1371/journal.pone.0132460. PMID: 26146838; PMCID: PMC4493149.

[67] Yao H, Ji Y, Li K, Liu S, He J, Wang R. HRCM: An Efficient Hybrid Referential Compression Method for Genomic Big Data. Biomed Res Int. 2019 Nov 16;2019:3108950. doi: 10.1155/2019/3108950. PMID: 31915686; PMCID: PMC6930768.

[68] Chern, B.G.; Ochoa, I.; Manolakos, A.; No, A.; Venkat, K.; Weissman, T. Reference based genome compression. In Proceedings of the IEEE Information Theory Workshop (ITW), Visby, Sweden, 25–28 August 2012; pp. 427–431.

[69] Kraft, Leon G. (1949), A device for quantizing, grouping, and coding amplitude modulated pulses, Cambridge, MA: MS Thesis, Electrical Engineering Department, Massachusetts Institute of Technology, hdl:1721.1/12390.

[70] McMillan, Brockway (1956), "Two inequalities implied by unique decipherability", IEEE Trans. Inf. Theory, 2 (4): 115–116, doi:10.1109/TIT.1956.1056818.

[71] Diffie, Whitfield; Hellman, Martin E. (November 1976). "New Directions in Cryptography" (PDF). IEEE Transactions on Information Theory. 22 (6): 644–654. CiteSeerX 10.1.1.37.9720. doi:10.1109/TIT.1976.1055638. Archived (PDF) from the original on 2014-11-29.

[72] Fritz, M.H.Y.; Leinonen, R.; Cochrane, G.; Birney, E. Efficient storage of high throughput DNA sequencing data using reference-based compression. *Genome Res.* **2011**, *21*, 734–740.

[73] Saha, S.; Sanguthevar, R. ERGC: An efficient referential genome compression algorithm. *Bioinformatics* **2015**, *31*, 3468–3475.

[74] Jarek Duda, "Asymmetric numeral systems: entropy coding combining speed of Huffman coding with compression rate of arithmetic coding", Center for Science of Information, Purdue University, W. Lafayette, IN 47907, 2013

[75] Garg, S 32/64 BIT RANGE CODING AND RITHMETIC CODING, https://sachingarg.com/compression/entropy$_c oding/, March$1979

[76] Robert F. Rice , Some Practical Universal Noiseless Coding Techniques, Jet Propulsion Laboratory, JPL Publication 79—22, March 1979

[77] Golomb, Solomon W., Run-length encodings, IEEE Transactions on Information Theory, IT–12(3), 1966, pp. 399–401

[78] J. H. Witten, R. M. Neal and J. G. Cleary, ARITHMETIC CODING FOR DATA COMPRESSION, Communication of the ACM (1987), 30, 1987, 520–540

[79] M. Nelson,Data Compression With Arithmetic Coding,Dr. Dobbs Journal,Nov 4, 2014

[80] P. G. Howard and J. S. Vitter,Practical Implementation of Arithmetic Coding,Kluwer Academic Publishers,1992,85–112

[81] D. A. Huffman, “A method for the construction of minimum-redundancy codes,” Proc. of the IRE, Vol. 40, pp. 1098-1101, Sep. 1952.

[82] D. E. Knuth, "The Art of Computer Programming", Vol. 3, Sorting and Searching, Addison Wesley, Reading, MA, 1973.

[83] R.M. Fano. "The transmission of information", Technical Report 65, Research Laboratory of Electronics, M.I.T., Cambridge, Mass., 1949.

[84] C. E. Shannon. "A mathematical theory of communication", Bell System Technical Journal, 27:379–423, 623–656, July, October 1948.

[85] Alistair Moffat, 2019, "Huffman Coding", ACM Comput. Surv. 52, 4, Article 85, August 2019

[86] Sieminskly, A. “Fast Decoding of the Huffman codes.” Information Processing Letters 26 (1987/88) pp. 237-241 11 January 198

[87] Tanaka, H., “Data structure of Huffman codes and its application to efficient encoding and decoding, “ IEEE Trans. Inf. Theory 33, 1 (Jan 1987), 154-156

[88] Renato Pajarola “Fast Prefix Code Processing”, IEEE ITCC Conference, pages 206–211, 2003.

[89] Alistair Moffat and Andrew Turpin, “On the Implementation of Minimum Redundancy Prefix Codes”, IEEE Transactions on Communications, Vol. 45, No. 10, October 1997

[90] Daniel S. Hirschberg and Debra A. Lelewer, “Efficient Decoding of Prefix Codes”, Communications of the ACM, Vol.33, pp. 449-459, 1990

[91] Chung Wang, Yuan-Rung Yang, Chun-Liang Lee, Hung-Yi Chang “A memory-efficient Huffman decoding algorithm” Published in: 19th International Conference on Advanced Information Networking and Applications (AINA’05) Volume 1.

[92] Habib, A., Rahman, M.S. “Balancing decoding speed and memory usage for Huffman codes using quaternary tree”. Appl Inform 4, 5 (2017).

[93] Swapna R. and Ramesh P., “Design and Implementation of Huffman Decoder for Text data Compression”, International Journal of Current Engineering and Technology, Vol.5, No.3 (June 2015).

[94] Yann Collet, Huff-0,
https://github.com/Cyan4973/FiniteStateEntropy
Accessed on: May 20, 2022.

[95] Yann Collet, zHuff,
https://fastcompression.blogspot.com/p/zhuff.html
Accessed on: May 20, 2022.

[96] Yann Collet, Zstd,
https://facebook.github.io/zstd/, Accessed on: May 20, 2022.

[97] National Library of Medicine, National Center for Biotechnology Information, Bethesda, MD, USA, Available online at: https://www.ncbi.nlm.nih.gov/genomes/VirusVariation/Database/nph-select.cgi, Accessed on May 20, 2022.

[98] Edward R. Fiala and Daniel H. Greene, "Data Compression with Finite Windows", Communications of the ACM, Vol. 32, Issue 4, April 1989, pp. 490-505.

[99] Yann Collet, Range-0,
http://sd-1.archive-host.com/membres/up/182754578/Range0v07.zip
Accessed on: May 20, 2022.

[100] Yann Collet, Huff-X, https://fastcompression.blogspot.com/p/huff0-range0-entropy-coders.html, Accessed on: May 20, 2022.

[101] G. N. N. Martin, "Range encoding: An algorithm for removing redundancy from a digitized message", Video & Data Recording Conference, Southampton, UK, July 24–27, 1979.

[102] Jean-loup Gailly, Mark Adler, “Zlib library”, Available online at: https://www.zlib.net/, Accessed on: May 20, 2022.

[103] Wikipedia. Adler-32. http://en.wikipedia.org/wiki/Adler-32

[104] J. G. Fletcher. An arithmetic checksum for serial transmissions. IEEE Transactions on Communications, COM30(1):247–252, Jan. 1982.

[105] Jarek Duda, "Asymmetric numeral systems: entropy coding combining speed of Huffman coding with compression rate of arithmetic coding", Center for Science of Information, Purdue University, W. Lafayette, IN 47907, 2013

[106] Garg, S 32/64 BIT RANGE CODING AND RITHMETIC CODING https://sachingarg.com/compression/entropy$_c oding/March$1979

[107] Robert F. Rice , Some Practical Universal Noiseless Coding Techniques, Jet Propulsion Laboratory, JPL Publication 79—22, March 1979

[108] Golomb, Solomon W., Run-length encodings, IEEE Transactions on Information Theory, IT–12(3), 1966, pp. 399–401

[109] J. H. Witten, R. M. Neal and J. G. Cleary, ARITHMETIC CODING FOR DATA COMPRESSION, Communication of the ACM (1987), 30, 1987, 520–540

[110] M. Nelson,Data Compression With Arithmetic Coding,Dr. Dobbs Journal,Nov 4, 2014

[111] P. G. Howard and J. S. Vitter,Practical Implementation of Arithmetic Coding,Kluwer Academic Publishers,1992,85–112

[112] Malcolm Taylor, ”RKIVE file archiver”, http://files.mpoli.fi/unpacked/software/dos/utils/diskfile/rkv190b1.zip/

[113] Bloom, Charles. ”LZP: A New Data Compression Algorithm.” Data Compression Conference. 1996.

[114] Kolmogorov, A. (1968). ”Logical basis for information theory and probability theory”. IEEE Transactions of Information Theory, IT-14:662–664, 1968

[115] Ziv, Jacob; Lempel, Abraham (May 1977). ”A Universal Algorithm for Sequential Data Compression”. IEEE Trans. on Info. Theory, 23:337–343, 1977.

[116] Jacob Ziv, Abraham Lempel, Compression of Individual Sequences via Variable-Rate Coding, IEEE Transactions on Information Theory, 24, 5, pp. 530-536, CiteSeerX 10.1.1.14.2892, doi:10.1109/TIT.1978.1055934, 1978

[117] Storer, James A.; Szymanski, Thomas G. (October 1982). "Data Compression via Textual Substitution". doi:10.1145/322344.322346.

[118] James A. Storer, 1992, "Method and apparatus for data compression", US Patent US5379036A, United States

[119] Robert K. Jung, "Data compression/decompression method and apparatus"

[120] W.J.Cody et al., Aug. 1984, IEEE standards 754 and 854 for Floating-Point Arithmetic, IEEE Magazine MICRO, 84 – 100, IEEE

[121] Morrison, Donald R., title=PATRICIA – Practical Algorithm To Retrieve Information Coded in Alphanumeric, journal=Journal of the Association for Computing Machinery, volume=15, number=4, year=October 1968

[122] Okumura, Haruhiko, History of Data Compression in Japan, edu.mie-u.ac.jp, https://oku.edu.mie-u.ac.jp/ okumura/compression/history.html, 1998

[123] R.Fiala, Edward; H. Greene, Daniel, Data Compression with Finite Windows, Communications of the ACM, 32, 4, April 1989, pp. 490-505, https://doi.org/10.1145/63334.63341

[124] Kak S., Generalized unary coding, Circuits Systems and Signal Processing, 35, 4, 1419 – 1426, doi:10.1007/s00034-015-0120-7, S2CID 27902257, 2015

[125] Golomb S. W.; Gordon Basil; Welch L. R., Comma-Free Codes, Canadian Journal of Mathematics, 10, 2, pp. 202 - 209, doi:10.4153/CJM-1958-023-9, 1958

[126] M. Dipperstein, "Arithmetic Code Discussion and Implementation", http://michael.dipperstein.com/arithmetic/index.html, 23rd November, 2014.

[127] Sabin B. and Daniela C., A Hybrid Data-Differencing and Compression Algorithm for the Automotive Industry, Entropy, 24, 5, 2022

[128] Sabin B. and Daniela C., The Anatomy of a Quasi-Static Arithmetic Encoder,2018 International Conference on Communications (COMM),2018

[129] Sabin B. and Daniela C., RoLZ - The Reduced Offset LZ Data Compression Algorithm,019 International Symposium on Signals, Circuits and Systems (ISSCS),2019

[130] Sabin B. and Daniela C., An innovative algorithm for data differencing, 2020 International Symposium on Electronics and Telecommunications (ISETC),2020

[131] Sabin B. and Daniela C.,Fast Canonical Huffman Decoder, IEEE COMMS 2022,2022

[132] I. Witten, T. Bell and John Cleary, MODELING FOR TEXT COMPRESSION, ACM Computing Surveys, 21, 4, 557–591, December 1989, University of Calgary, CA

[133] Timothy Bell, John Cleary and Ian Witten, Text Compression, 0-13-911991-4, 1st edition, February 1, 1990, Prentice-Hall, Englewood, USA

[134] Alistair Moffat and Neil Sharman, AN EMPIRICAL EVALUATION OF CODING METHODS FOR MULTI-SYMBOL ALPHABETS, 0-13-911991-4, 30, 6, 791-804, Information Processing and Management, 1994

[135] Calude, C.S. (1996). "Algorithmic information theory: Open problems"

[136] “Delta Algorithms: An Empirical Analysis”, James J. Hunt University of Karlsruhe, Karlsruhe, Germany and Kiem-Phong Vo AT&T Laboratories, Florham Park, NJ, USA and Walter F. Tichy University of Karlsruhe, Karlsruhe, Germany.

[137] David G. Korn and Kiem-Phong Vo, “Engineering a Differencing and Compression Data Format”, AT&T Laboratories – Research 180 Park Avenue, Florham Park, NJ 07932, U.S.A. dgk,kpv@research.att.com.

[138] James W. Hunt and M.D. McIllroy, “An algorithm for differential file comparison” Technical Report Computing Science Technical Report 41, Bell Laboratories, June 1976.

[139] Webb Miller and Eugene W. Meyers, “A file comparison program. SoftwarelPractice and Experience” 15(11):1025, 1039, November 1985.

[140] C.H. Bennett, P. Gacs, M. Li, P.M.B. Vitanyi, and W. Zurek, Information Distance, IEEE Trans. Inform. Theory, IT-44:4(1998) 1407–1423

[141] M. Li, X. Chen, X. Li, B. Ma, P.M.B. Vitanyi, "The similarity metric", IEEE Trans. Inform. Th., 50:12(2004), 3250–3264". IEEE Transactions on Information Theory. 50 (12): 3250–3264. doi:10.1109/TIT.2004.83810

[142] Williams R.N., Adaptive Data Compression, Kluwer Academic Publishers, 1991

[143] Fraenkel, Aviezri S.; Klein, Shmuel T., Robust universal complete codes for transmission and compression, Discrete Applied Mathematics, Markov Chains, 64, 1996, ISSN=0166-218X, CiteSeerX 10.1.1.37.3064, 10.1016/0166-218X(93)00116-H

[144] Markov, Andreĭ Andreevich, Extension of the limit theorems of probability theory to a sum of variables connected in a chain, Dynamic Probabilistic Systems, reprinted in Appendix B of: R. Howard, Markov Chains, 1, 1971, John Wiley and Sons

[145] Elias, Peter, Universal codeword sets and representations of the integers, IEEE Transactions on Information Theory, Markov Chains, 21, 2, 194-203, 1975, 10.1109/tit.1975.1055349

[146] Fraenkel, Aviezri S. and Klein, Shmuel T., Robust universal complete codes for transmission and compression, Discrete Applied Mathematics, 64, 1, 31 – 55, 1996, 10.1109/tit.1975.1055349, 0166-218X