https://doi.org/10.33261/jaaru.2022.29.1.006 *Association of Arab Universities Journal of Engineering Sciences (2022)29(1):56-63*

Association of Arab Universities Journal of Engineering Sciences

مجلة اتحاد الجامعات العربية للدراسات والبحوث الهندسية

كلية الهندسة - جامعة بغداد

جمعية كليات الهندسة اعضاء اتحاد الجامعات العربية

دراسة أثر تغيير طبيعة البيانات على نسبة الضغط الكلية للملفات

د. محمد سمير مدبّس[1]، محمد أمين زيّات[2،*]

[1] قسم هندسة الاتصالات، كلية الهندسة الكهربائية والإلكترونية، جامعة حلب، سوريا، Modabbes@alepuniv.edu.sy

[2] قسم هندسة الاتصالات، كلية الهندسة الكهربائية والإلكترونية، جامعة حلب، سوريا، Aminzayyat@gmail.com

*الباحث الممثل، م. محمد أمين زيات، Aminzayyat@gmail.com

نشر في : 31 اذار 2022

الخلاصة – تقدم هذه الورقة البحثية دراسة لطريقة لتغيير طبيعة البيانات من خلال معالجة وتبديل الرموز في مرحلة ما قبل الضغط تسمح بزيادة تكرارات الرموز في البيانات التي سيتم ضغطها باستخدام إحدى خوارزميات الضغط المعروفة، حيث تقوم هذه الطريقة باستبدال الرمز الأكثر تكراراً والرمز التالي الأكثر تكراراً برمز واحد يحمل مجموع تكراراتهما ومن ثم يتم استبدال الرمز التالي الأكثر تكراراً مع الذي يليه برمز آخر يحمل مجموع تكراراتهما وهكذا حتى يتم استبدال كل الرموز، حيث يكون عدد الرموز الجديدة هو نصف عدد الرموز الأصلية، وهنا في هذا البحث تمت دراسة نظام الترميز ASCII (American Standard Code for Information Interchange) الذي يتكون من 256 رمز من 8 bits لكل منها، وبعد أن يتم استبدال كل الرموز نحصل على رموز جديدة عددها يساوي نصف عدد الرموز الأصلية أي 128 رمز من 7 bits لكل منها، ولضمان عدم ضياع البيانات فقد تمت الإشارة لكل رمز بـ bit واحد مميز يتم كتابته في ملف منفصل وهذا الـ bit يجعل استعادة الرموز الأصلية أمراً ممكناً. وفي النهاية، يدمج الملفان معاً ويتم ضغط رموز 7 bits باستخدام خوارزمية الضغط LZW وهذا يجعل الملف النهائي يحتوي على رموز مكررة بشكل أكبر من الملف الأصلي، أما فك الضغط فيحصل بطريقة معاكسة إذ يتم أولاً فك ضغط الملف باستخدام خوارزمية LZW ومن ثم يتم فصل الملفان عن بعضهما وقراءة رمز 7 bits من الملف الأول مع bit من الملف الثاني من أجل استعادة الرمز الأصلي حتى تتم استعادة كل رموز ASCII الأصلية، تمت مقارنة الضغط بإحدى طرق الانتروبي كالترميز الحسابي (Arithmetic Coding) مع الضغط باستخدام طريقة LZW ولماذا تتفوق LZW على الطرق الإحصائية التقليدية بالضغط مثل Huffman أو الترميز الحسابي.

الكلمات الرئيسية –نسبة الضغط، ضغط بدون فاقد، استبدال الرموز، خوارزمية، انتروبي.

1. المقدمة

ضغط البيانات هي عملية تحويل سلسلة بيانات الدخل (input) أو البيانات الأصلية إلى سلسلة بيانات الخرج (output) أو البيانات المضغوطة والتي تكون أصغر حجماً، وتكمن الحاجة إلى عملية ضغط البيانات في كون معظم البيانات التي نولدها لترسل عبر الوسائط المتعددة تكون ذات حجم كبير بالإضافة للحاجة إلى آلية لكشف وتصحيح الأخطاء والمحافظة على السرية.

تقسم تقنيات ضغط البيانات إلى نوعين: الضغط بفاقد والضغط بدون فاقد، حيث يتضمن الضغط بفاقد خسارة معينة من الدقة في مقابل نسبة ضغط عالية، أما الضغط بدون فاقد فيتضمن تقنيات تضمن توليد نسخة مطابقة تماماً من سلسلة بيانات الدخل بعد إتمام دورة الضغط/ فك الضغط، ويستخدم هذا النوع من الضغط للبيانات التي لا تقبل أي نسبة فقد مهما كانت مثل قواعد البيانات، الجداول أو الملفات النصية[11]. ويمكن تقسيم طرق الضغط بدون فاقد إلى نوعين رئيسين:

- الضغط باستخدام النموذج الإحصائي: وفيه يتم أخذ إحصاءات عن الرموز المراد ضغطها ومن ثم الاستفادة من هذه الإحصاءات في ترميز بيانات الخرج. نذكر من هذه الطرق Huffman وArithmetic Coding.

- الضغط باستخدام القاموس: بهذه الطريقة يتم قراءة الملف المراد ضغطه كل رمز على حدة وبناء قاموس مكون من هذه الرموز أثناء قراءتها. نذكر من هذه الطرق LZSS(Lempel-Ziv-Storer-Szymanski) وLZW(Lempel-Ziv-Welch).

تكون معظم البيانات بشكلها الخام كبيرة جداً وغير مناسبة للإرسال عبر شبكة الإنترنت مثلاً ولذلك لابد من ضغطها حتى يصبح حجمها أصغر، يهدف هذا البحث إلى زيادة نسبة الضغط للملفات وتقليل حجمها وبالتالي تخفيض الوقت اللازم لتحميلها عبر الشبكة والتوفير في وسائط التخزين اللازمة لتخزين الملفات في حال كان الضغط للأرشفة مثلاً.

2. خوارزمية الضغط LZW:

تعتبر خوارزمية LZW إحدى طرق الضغط بدون فاقد باستخدام القاموس حيث يتم البدء بقاموس أساسي يحوي على الرموز الأساسية للأبجدية (256 محرف في حالة ترميز ASCII) ويتم بناء القاموس بإضافة المحارف إلى هذا القاموس أثناء قراءة البيانات وهذه المحارف ستشكل كلمات في مرحلة لاحقة، حيث يتم قراءة كل محرف مع الذي يليه فإذا لم يوجدا في القاموس عندها يتم إخراج رمز(Code) هذا المحرف وحجز المحرف مع المحرف الذي يليه في القاموس وإعطائهما رقم أول رمز مضغوط متوفر، أما في حال وجود المحرفين الذين تمت قراءتهما في القاموس عندها سيتم إخراج الرمز المضغوط المعبر عن هذين المحرفين وحجز المحرفين مع المحرف الذي يليهما في القاموس وإعطاء المجموع قيمة الرمز المضغوط الحالي، حيث يتم حجز أول 255 رقم (8 bits) لمحارف ASCII ويتم بناء القاموس بداية من الرقم 256 (9 bits) وحتى 15 bits في خوارزمية LZW15 القياسية.

فإذا كانت بيانات الدخل 9 محارف ASCII بـ 8 bits لكل منها هي:

BABAABAAA

عندها تكون طريقة عمل خوارزمية الضغط LZW موضحة **بالجدول (1)**.

الجدول 1: طريقة عمل خوارزمية ضغط LZW.

Encoder Output		String Table	
Output code	**representing**	**Code word**	**string**
66	B	256	BA

65	A	257	AB
256	BA	258	BAA
257	AB	259	ABA
65	A	260	AA
260	AA		

نلاحظ عند ورود الحرف الأول B تتم قراءته مع الحرف الذي يليه أي BA والبحث عنهما في القاموس وبما أنهما غير موجودان فقد تم إخراج قيمة ASCII للحرف B وهي 66 (بـ 9 bits) وإضافة الحرف الأول والحرف الذي يليه أي BA كـ string جديد في القاموس وحجز أول قيمة له من القاموس وهي 256، وعند ورود الحرف الثاني A تتم قراءته مع الحرف الذي يليه أي AB والبحث عنهما في القاموس وبما أنهما غير موجودين فقد تم إخراج قيمة ASCII للحرف A وهي 65 (بـ 9 bits) وإضافته مع الحرف الذي يليه أي AB كـ string جديد في القاموس وحجز ثاني قيمة له من القاموس وهي 257، وعند ورود الحرف الثالث B تتم قراءته مع الحرف الذي يليه أي BA والبحث عنهما في القاموس وبما أنهما موجودان في القاموس يتم إخراج قيمة الرمز BA وهي 256 وإضافته مع الحرف التالي أي BAA كـ string جديد في القاموس وحجز القيمة التالية له من القاموس وهي 258، وعند ورود الحرف التالي A تتم قراءته مع الحرف الذي يليه أي AB والبحث عنهما في القاموس وبما أنهما موجودان في القاموس تتم إخراج قيمة الرمز AB وهي 257 وإضافته مع الحرف التالي أي ABA كـ string جديد في القاموس وحجز القيمة التالية له من القاموس وهي 259، وعند ورود الحرف التالي A تتم قراءته مع الحرف الذي يليه أي AA والبحث عنهما في القاموس وبما أنهما غير موجودين فقد تم إخراج قيمة ASCII للحرف A وهي 65 وإضافته مع الحرف الذي يليه أي AA كـ string جديد في القاموس وحجز القيمة التالية له من القاموس وهي 260، وعند ورود الحرف التالي A تتم قراءته مع الحرف الذي يليه أي AA والبحث عنهما في القاموس وبما أنهما موجودان في القاموس يتم إخراج قيمة الرمز AA وهي 260، وبهذا نحصل على الخرج النهائي المضغوط الذي يتكون من 6 رموز بـ 9 bits لكل منها هي:

<66><65><256><257><65><260>

تقوم خوارزمية LZW بإخراج رموز 9 bits حتى الوصول بالقاموس إلى الرقم 512 وهو أكبر رقم يمكن تمثيله بـ 9 bits وعندها تقوم الخوارزمية بإخراج رموز 10 bits حتى الوصول إلى الرقم 1024 وهو أكبر رقم يمكن تمثيله بـ 10 bits وعندها يتم إخراج رموز 11 bits وهكذا يتم إخراج رموز متغيرة الأطوال وهو ما يعرف باسم Variable Length Codes أي أن الرموز ليس لها طول محدد وليست مثل رموز ASCII ذات الطول المحدد حيث تكون 8 bits دائماً وأبداً، أما رموز خرج LZW المضغوطة لها أطوال متغيرة من 9 bits حتى 15 bits في خوارزمية LZW15 القياسية.

أما طريقة فك الضغط LZW فتعمل بشكل معاكس حيث يتم إرسال رموز الأبجدية الأساسية (رموز ASCII مثلاً) ومن ثم يتم بناء القاموس الكامل من قبل خوارزمية فك الضغط وتتم قراءة الرموز وإخراج المحرف أو مجموعة المحارف المقابلة لهذا الرمز في هذا القاموس، فإذا كانت البيانات المضغوطة الواردة إلى دخل خوارزمية فك الضغط LZW هي:

<66><65><256><257><65><260>

تكون طريقة عمل خوارزمية فك الضغط LZW موضحة **بالجدول (2)**.

الجدول 2: طريقة عمل خوارزمية فك ضغط LZW.

Encoder Output	String Table

string	Code word	string
B		
A	256	BA
BA	257	AB
AB	258	BAA
A	259	ABA
AA	260	AA

عند ورود الرمز المضغوط الأول <66> فإن المستقبل يعلم أنه الحرف B لأن المقابل للرقم 66 حسب ترميز ASCII هو الحرف B وبالتالي يتم إخراج الحرف B وحجز الحرف B مع الحرف الذي يليه ذو الرقم 65 أي الحرف A كأول قيمة في القاموس عند الرقم 256 وعند ورود الرمز المضغوط الثاني ذو الرقم 65 أي الحرف A يتم إخراج الحرف A وحجز الحرف A مع أول حرف من الرمز المضغوط التالي 256 الذي يتكون من الحرفين BA فيصبح لدينا الحرف A مع أول حرف من الرمز المضغوط الذي يليه أي B كثاني قيمة في القاموس عند الرقم 257 وعند ورود الرمز المضغوط الثالث ذو الرقم 256 وبما أن الرقم قد تجاوز 255 أكبر رقم في ترميز ASCII لذلك فإن المستقبل يعلم أن هذا الرمز موجود في القاموس فيتم إخراج الحرفين BA من القاموس وحجز الحرفين BA مع أول حرف من الرمز المضغوط التالي 257 الذي يتكون من الحرفين AB أي حجز BAA عند القيمة 258 في القاموس وهكذا يتم بناء القاموس من الرموز المضغوطة الواردة وفك ضغط البيانات بالوقت عينه فنحصل بالنهاية على البيانات الأصلية BABAABAAA.

3. الدراسات السابقة

درست الأبحاث السابقة مجموعة طرق مختلفة لتحسين ضغط البيانات نذكر منها استخدام خوارزمية LZW لضغط رموز Unicode بدلاً من البايتات[1] أو إضافة الكلمات المكررة بشكل كبير إلى قاموس LZW بشكل اختياري[7] أو استخدام رموز مضغوطة بطول محدد حسب حجم النص[8]، أما في هذا البحث فقد اعتمدنا طريقة لتغيير طبيعة البيانات عن طريق استبدال رموز البيانات المراد ضغطها لتوفير تكرارات أكبر وبالتالي احتمالات رموز أكبر وبالنتيجة الحصول على نسبة ضغط بيانات أفضل.

3.1 نظام ضغط ملفات Unicode باستخدام طريقة LZW محسنة:

في هذا البحث[1] تم تقديم طريقة لضغط رموز Unicode وهو نظام ترميز يستخدم في الأنظمة التي تعمل مع عدد كبير من اللغات أو للغات ذات المحارف والرموز الكثيرة حيث يكون 256 رمز كما هو الحال في نظام الترميز ASCII غير كافٍ، ولذلك يتم استخدام نظام الترميز Unicode حيث تكون الرموز مجموعة من البايتات (8 bits) كما هو الحال في نظام UTF-8 أو مجموعة من بايتين (16 bits) كما هو الحال في نظام UTF-16 ... الخ مما يسمح باستخدام الآلاف من المحارف والرموز، في هذه الطريقة تم تعديل خوارزمية LZW لكي تضغط بيانات Unicode بإنشاء مدخلات القاموس بصيغة Unicode، وبالتالي ستتم قراءة رمز UTF-8 بدلاً من قراءة بايت في كل مرة، ويكون رمز UTF-8 مؤلف من بايت أو اثنان أو ثلاثة أو أربعة بايتات ويتم إخراج رمز مضغوط بناءً على رموز الدخل UTF-8 وهذا سينعكس بطريقة إيجابية على نسبة الضغط فبدلاً من قراءة بايت في كل مرة كما هو الحال في خوارزمية LZW التقليدية ستتم قراءة مجموعة من البايتات كرمز UTF-8 واحد وإخراج رمز مضغوط لمجموعة من البايتات بدلاً من بايت واحد كما هو الحال في طريقة LZW التقليدية.

3.2 مقاربة جديدة لزيادة نسبة ضغط خوارزمية LZW:

في هذا البحث[7] تم اقتراح إضافة المقاطع المكررة بشكل كبير في اللغة المدروسة إلى القاموس، حيث تتم إضافة الكلمات ذات احتمال الورود الكبير في النص بشكل اختياري، على سبيل المثال، كلمات مثل as, at, an, in, on, ... الخ مما يقلل عدد المرات اللازمة لتشكل الكلمات في القاموس وبالتالي يقلل عدد الـ bits المطلوبة.

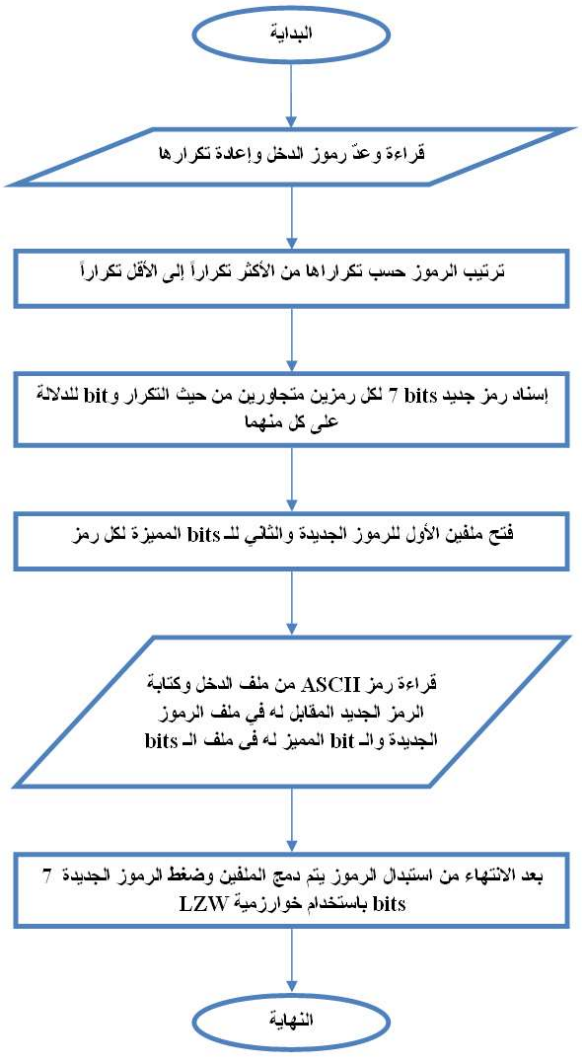
3.3 طريقة فعالة لتخفيض طول رمز الخرج لخوارزمية LZW:

في هذا البحث[8] تم افتراض أن رموز الخرج في خوارزمية LZW بطول ثابت (مع العلم أنه في المرجع [9] تم الضغط باستخدام الرموز متغيرة الأطوال كما تم شرحه في الفقرة 2) ويتم منح المستخدم الخيار لاختيار طول الرمز (code length) بناءً على حجم ملفه، على سبيل المثال: للملفات الصغيرة الحجم يختار المستخدم طول الرمز 9 bits وللملفات المتوسطة الحجم يختار المستخدم طول الرمز 12 bits أما للملفات الكبيرة فيختار المستخدم 14 bits، وبما أنه تم تخفيض طول الرمز فسيتم الحصول على نسبة ضغط أفضل للنص.

4. منهجية البحث

في هذا البحث قمنا بدارسة طريقة لتغيير طبيعة البيانات قبل الضغط تقوم باستبدال الرمز الأكثر تكراراً والرمز التالي الأكثر تكراراً برمز جديد واحد مكرر بقدر تكرار هذين الرمزين الأكثر تكراراً وهكذا مع الرمز الثالث الأكثر تكراراً والرابع الأكثر تكراراً نستبدلهم برمز واحد له مجموع تكراراتهم وبالتالي يمكن تمثيل الرموز الجديدة بعدد أقل من الـ bits، وبما أنه سيتم دمج رمزين أصليين برمز جديد واحد وهذا الرمز الجديد سيعبر عن كلا الرمزين الأصليين ولكيلا نفقد البيانات بعد الاستبدال ونتمكن من استعادة البيانات الأصلية يجب وضع bit يشير فيما إذا كان الرمز الجديد يعبر عن الرمز الأصلي الأول أو الرمز الأصلي الثاني (صفر يدل أن الرمز هو الأول وواحد يدل أن الرمز هو الثاني)، وعند إجراء فك الضغط تتم قراءة الرمز الجديد مع bit من الملف الثاني لاسترجاع الرمز الأصلي، في البداية يتم قراءة رموز الدخل وعدّها ومن ثم يتم تبديلها وفتح ملفين في الملف الأول يتم كتابة الرموز الجديدة (المبدلة) وفي الملف الثاني يتم كتابة bit من أجل كل رمز (لمعرفة فيما إذا كان الرمز الجديد يشير إلى الرمز الأول أو الثاني)، ثم يتم دمج كلا الملفين في ملف واحد يحتوي على الرموز الجديدة ذات 7 bits ومن ثم يتم ضغط هذا الملف باستخدام خوارزمية الضغط LZW، ويتم إرسال عدد الرموز في الترويسة (header) للملف المضغوط النهائي كي نتمكن من استعادة الملفين عند فك الضغط، وكذلك يتم إرسال 256 بايت في ترويسة الملف المضغوط النهائي تعبر عن 256 رمز ASCII مرتبة حسب تكرارها أي في البايت الأول يوجد الرمز الأول الأكثر تكراراً وفي البايت الثاني يوجد ثاني رمز من حيث التكرار وهكذا، وفي طرف الاستقبال تتم قراءة هذه الـ 256 رمز وإسناد الرموز الجديدة لها حسب ترتيبها أي أن الرمز الأكثر تكراراً الموجود بالبايت الأول سيصبح الرمز 0 وبـ 7 bits والـ bit الخاص سيكون 0 والرمز الثاني الأكثر تكراراً الموجود بالبايت الثاني سيصبح أيضاً الرمز 0 بـ 7 bits والـ bit الخاص سيكون 1 أما الرمزين الثالث والرابع من حيث التكرار والموجدان بالبايتين الثالث والرابع سيصبحان الرمز 1 بـ 7 bits والرمز الثالث هو 0 وbit الرمز الرابع هو 1 وهكذا حتى الوصول إلى الرمزين الأخيرين 254 و255 (256 رمز من صفر حتى 255) الموجدان بالبايتين 255 و256 اللذين سيتم استبدالهما بالرمز 127 لكليهما وbit 0 للرمز 254 وbit 1 للرمز 255 وبالتالي نكون عرفنا الرموز الأصلية الـ 256 ذات 8 bits والرموز الجديدة الـ 128 ذات 7 bits ويمكن القيام بعملية معاكسة للحصول على البيانات الأصلية.

يبين **الشكل (1)** المخطط الطوري Flow Chart للطريقة المقترحة.



الشكل 1: المخطط الطوري للطريقة المقترحة.

تم تطبيق هذه الطريقة على ملفات نصية تشكل ثلاث روايات من الأدب العالمي هي هاملت للكاتب وليام شكسبير وقصة مدينتين للكاتب تشارلز ديكينز والبؤساء للكاتب فكتور هوغو، حيث أن طبيعة الرموز في هذه الملفات هي رموز (محارف) ASCII (256 رمز) بـ 8 bits لكل رمز (محارف إنكليزية في هاملت وقصة مدينتين ومحارف فرنسية في البؤساء) وفي **الجدول(3)** نستعرض جدول يضم المحارف حسب نظام الترميز ASCII وبعد تطبيق الطريقة نحصل على رموز جديدة عددها يساوي نصف عدد الرموز الأصلية أي 128 رمز بـ 7 bits لكل منها وتكون الرموز الجديدة مكررة بشكل أكبر من الرموز الأصلية، حيث يكون الرمز الجديد الأكثر تكراراً ذو 7 bits مكرر بعدد مرات يساوي تكرار الرمز الأصلي الأكثر تكراراً ذو 8 bits زائد تكرار الرمز الأصلي التالي الأكثر تكراراً ذو 8 bits، ويكون الرمز الجديد الثاني من حيث التكرار مكرر بقدر تكرار كلا الرمزين الأصليين الثالث والرابع من حيث التكرار، وهكذا حتى الانتهاء من كل الرموز الأصلية لنحصل في النهاية على ملف جديد ذو رموز جديدة بـ 7 bits لكل منها وهو الملف الذي سيتم ضغطه باستخدام خوارزمية الضغط LZW.

الجدول 3: رموز نظام الترميز ASCII.

Standard ASCII The first 32 characters are control codes.				Extended ASCII (DOS)			
0 Null	33 !	81 Q		128 Ç	174 «	220	
1 Start of heading	34 "	82 R		129 ü	175 »	221	
2 Start of text	35 #	83 S		130 é	176	222	
3 End of text	36 $	84 T		131 â	177	223	
4 End of transmit	37 %	85 U		132 ä	178	224	α
5 Enquiry	38 &	86 V		133 à	179	225	ß
6 Acknowledge	39 '	87 W		134 å	180	226	Γ
7 Audible bell	40 (	88 X		135 ç	181	227	π
8 Backspace	41)	89 Y		136 ê	182	228	Σ
9 Horizontal tab	42 *	90 Z		137 ë	183	229	σ
10 Line feed	43 +	91 [		138 è	184	230	µ
11 Vertical tab	44 ,	92 \		139 ï	185	231	τ
12 Form feed	45 -	93]		140 î	186	232	Φ
13 Carriage return	46 .	94 ^		141 ì	187	233	Θ
14 Shift out	47 /	95 _		142 Ä	188	234	Ω
15 Shift in	48 0	96 `		143 Å	189	235	δ
16 Data link escape	49 1	97 a		144 É	190	236	∞
17 Device control 1	50 2	98 b		145 æ	191	237	ø
18 Device control 2	51 3	99 c		146 Æ	192	238	∈
19 Device control 3	52 4	100 d		147 ô	193	239	∩
20 Device control 4	53 5	101 e		148 ö	194	240	≡
21 Neg. acknowledge	54 6	102 f		149 ò	195	241	±
22 Synchronous idle	55 7	103 g		150 û	196	242	≥
23 End trans. block	56 8	104 h		151 ù	197	243	≤
24 Cancel	57 9	105 i		152 ÿ	198	244	⌠
25 End of medium	58 :	106 j		153 Ö	199	245	⌡
26 Substitution	59 ;	107 k		154 Ü	200	246	÷
27 Escape	60 <	108 l		155 ¢	201	247	≈
28 File separator	61 =	109 m		156 £	202	248	°
29 Group separator	62 >	110 n		157 ¥	203	249	∙
30 Record separator	63 ?	111 o		158 ₧	204	250	·
31 Unit separator	64 @	112 p		159 ƒ	205	251	√
32 Blank space	65 A	113 q		160 á	206	252	ⁿ
	66 B	114 r		161 í	207	253	²
	67 C	115 s		162 ó	208	254	■
	68 D	116 t		163 ú	209	255	
	69 E	117 u		164 ñ	210		
	70 F	118 v		165 Ñ	211		
	71 G	119 w		166 ª	212		
	72 H	120 x		167 º	213		
	73 I	121 y		168 ¿	214		
	74 J	122 z		169 ⌐	215		
	75 K	123 {		170 ¬	216		
	76 L	124 |		171 ½	217		
	77 M	125 }		172 ¼	218		
	78 N	126 ~		173 ¡	219		
	79 O	127 ⌂					
	80 P						

5. الدراسة التحليلية

لتحليل عمل الطريقة المقترحة، سنقوم أولاً بدراستها مع طرق الضغط الإحصائية التقليدية ومن ثم نقوم بمقارنة الفرق بين هذه الطرق وطريقة الضغط باستخدام القاموس LZW، من أشهر طرق الضغط الإحصائية هي طريقة هافمان Huffman حيث يتم إسناد رمز مضغوط لكل محرف حسب تكراره حيث تصبح الرموز الأكثر تكراراً رموز مضغوطة بأقل عدد من الـ bits بينما تصبح الرموز الأقل تكراراً رموز مضغوطة بعدد أكبر من الـ bits فمثلاً يمكن ان يصبح الرمز الأكثر تكراراً رمز مضغوط بـ bit واحد أو باثنان من الـ bits وهنا يحصل الضغط حيث أن الرمز الأكثر تكراراً الذي كان 8 bits أصبح bit أو 2 bits ولكن يعيب هذه الطريقة أنها تسند عدد صحيح من الـ bits لكل رمز مضغوط وبالتالي إذا كان الانتروبي للرمز الأكثر تكراراً هو 1,7 bit على سبيل المثال عندها لن نستطيع ترميزه بـ 1,7 bit باستخدام طريقة هافمان وإنما بـ 2 bits لأنه وكما أسلفنا فإن طريقة هافمان تضع عدد صحيح من الـ bits لكل رمز مضغوط مما يعني ضياع 0,3 bit مع كل رمز مضغوط، وهنا تبرز طريقة أفضل للضغط وهي الترميز الحسابي Arithmetic Coding التي تقوم بترميز كل محرف حسب الانتروبي تماماً وبالتالي إذا كان الانتروبي للرمز الأكثر تكراراً هو 1,7 bit فإن هذا الرمز سيتم ترميزه بـ 1,7 bit تماماً دون زيادة أو نقصان عند الضغط باستخدام طريقة الترميز الحسابي حيث تقوم هذه الطريقة بتمثيل الملف المضغوط النهائي برقم كبير جداً وداخل هذا الرقم يكون كل محرف ممثل حسب الانتروبي الخاص به تماماً ولذلك فإن طريقة الترميز الحسابي تعد من أشهر طرق الانتروبي، ومن أجل رواية هاملت كان الرمز الأكثر تكراراً هو المسافة (ASCII 32) بعدد 47876 مرة يليه الحرف e (ASCII 101) بعدد 14722 مرة وهذان الرمزان سيتم دمجهما في رمز واحد جديد يحمل مجموع تكراراتهما أي 47876+14722=62598 مرة.

عدد البايتات الكلي لملف هاملت هو 191734 بايت (رمز).

"تستخدم نظرية المعلومات تعبير الانتروبي (المقياس) (entropy) لقياس كمية المعلومات التي تم ترميزها في الرسالة. تم استعارة كلمة انتروبي من الديناميكا الحرارية (thermodynamics)، ولها معنى مشابه. كلما كان الانتروبي لرسالة ما أعلى، كلما كانت كمية المعلومات التي تحتويها هذه الرسالة أكبر. يعرف الانتروبي لرمز ما باللوغاريتم الثنائي السالب لاحتماله. لتحديد محتوى المعلومات لرسالة بالـ bits، نعرف الانتروبي باستخدام لوغاريتم الأساس 2 على النحو:

Number of bits = - Log base 2 (probability) (1)"[9].

من العلاقة (1) وعند حساب الانتروبي لرمز المسافة ذو الاحتمال 47876/191734= 0,2497 نجد:

$$-\text{Log base 2 } (0{,}2497) = 2{,}0017$$

وهذا يعني أننا سنحتاج 2 bits تقريباً لتمثيل هذا الرمز عند الضغط باستخدام الانتروبي، أما من أجل الرمز e ذو الاحتمال 14722/191734= 0,0768

$$-\text{Log base 2 } (0{,}0768) = 3{,}7027$$

أي أننا سنحتاج 3,7 bits تقريباً لتمثيل الرمز e باستخدام الانتروبي.

وسيكون عدد الرموز الجديدة هو 191734×8÷7=219125 رمز واحتمال الرمز الجديد الأكثر تكراراً هو 62598/219125=0,2857

$$-\text{Log base 2 } (0{,}2857) = 1{,}8074$$

وبالتالي يلزمنا 1,8 bit تقريباً لتمثيل الرمز الجديد باستخدام الانتروبي.

وبما أن الرموز الجديدة لها احتمالات أكبر من الرموز الأصلية فإنها ستحتاج إلى عدد bits أقل لتمثيلها، ففي حين كان يلزمنا في البيانات الأصلية 2 bits لترميز 47876 رمز و3,7 bits لترميز 14722 رمز أصبح لدينا 62598 رمز مُرمزة بـ 1,8 bit في البيانات الجديدة وهو ما يعتبر تحسين لنسبة الضغط إذا قمنا بالضغط باستخدام طريقة الترميز الحسابي.

تتفوق طريقة LZW على طرق الانتروبي بأنها تقوم بترميز مجموعة من المحارف معاً وليس كل محرف منفرداً كما هو الحال في الطرق الإحصائية التقليدية، فالرمز المضغوط الذي كان يسند لمحرف واحد في طريقة هافمان أو طريقة الترميز الحسابي أصبح هذا الرمز المضغوط يسند لكلمة أو جملة بأكملها في طريقة LZW وهو ما يعتبر نقلة نوعية في عالم ضغط النصوص.

الجدول 4 : مراحل تشكل الكلمات في قاموس LZW.

the	_	_t
the	_t	_th
the	_th	_the
the	_the	_the_
there	_the	_ther
there	_ther	_there

there	_there	_there_
_there_is_	_there_	_there_i
_there_is_	_there_i	_there_is
_there_is_	_there_is	_there_is_
_there_are_	_there_	_there_a
_there_are_	_there_a	_there_ar
_there_are_	_there_ar	_there_are
_there_are_	_there_are	_there_are_

وهكذا نجد أنه بحسب مراحل تشكل القاموس في المرة الأولى التي تصادف بها خوارزمية LZW كلمة _the_ (أي مسافة the مسافة) سيتم إخراج المسافة برمز مضغوط وحجز رمز مضغوط للمسافة وt أي t_ وعند ورود كلمة _the_ مرة أخرى سيتم إخراج t_ وحجز th_ وفي المرة الثالثة سيتم إخراج th_ وحجز the_ وفي المرة الرابعة سيتم إخراج the_ وحجز _the_ ويكون اكتمل تشكل كلمة _the_ في القاموس وفي المرات القادمة التي ترد كلمة _the_ سيتم إخراجها برمز مضغوط واحد من 9 إلى 15 bits وحجزها مع المحرف الذي يليها في رمز مضغوط جديد.

الآن وعند ورود كلمة _there_ وبسبب وجود المقطع the_ مسبقاً في القاموس سيتم إخراج the_ وحجز ther_ وفي المرة التالية التي ترد فيها كلمة _there_ سيتم إخراج ther_ وحجز there_ وعند ورود كلمة _there_ مرة أخرى سيتم إخراج there_ وحجز كلمة _there_ برمز مضغوط جديد وبهذا نلاحظ أن تشكل الكلمات الجديدة أصبح يحتاج إلى مرات أقل بسبب وجود مقاطع متطابقة سابقة في القاموس وبنفس الطريقة تتشكل الجملة _there_is_ والجملة _there_are_ ففي كل مرة تجد الخوارزمية مقطع متطابق في القاموس يتم إخراج هذا المقطع وحجزه مع المحرف التالي برمز جديد في القاموس، وبهذا الشكل تستغل الخوارزمية خصائص اللغة وخصائص البيانات لتحقيق أكبر ربح وإخراج أكبر قدر من المحارف المتطابقة برمز مضغوط واحد من 9 إلى 15 bits، فالخوارزمية نهمة ولا تتوقف أبداً وتستمر بإضافة محرف جديد في كل مرة لتشكيل كلمة أو جملة جديدة، فبعد ورود بضعة مئات من كلمات النص المراد ضغطه سنجد أن كلمات النص بدأت بالتشكل داخل القاموس ولذلك تمت تسميته بالقاموس وبعد ورود بضع مئات أخرى من الكلمات سنجد ليس فقط كلمات وإنما الجمل المكررة بالنص داخل القاموس.

ولذلك لن يكون من العدالة المقارنة بين طريقة تضغط كلمات وجمل وطريقة تضغط كل محرف بمفرده كما هو الحال في هافمان أو الترميز الحسابي لأن الغلبة ستكون بكل سهولة للطريقة التي تضغط مجموعة من الرموز (كلمات، جمل)، فالرمز المضغوط الذي كان يتم إسناده لمحرف واحد في الطرق الإحصائية أصبح يسند إلى كلمة أو جملة كاملة في طريقة LZW.

وفي مثالنا السابق نجد أن الجملة _there_are_ المكونة من أحد عشر محرف قد تم ترميزها برمز مضغوط من 9 إلى 15 bits، أي أن 88 bits قد تم ضغطها في 15 bits، الأمر الذي يعتبر مستحيلاً في الطرق الإحصائية وطرق الانتروبي.

ولذلك وبالرغم من أننا سنحصل على تحسين في نسبة الضغط عند استخدام استبدال الرموز مع الطرق الاحصائية والانتروبي إلا أن هذا التحسين سيكون أضعف عند المقارنة بالتحسين الذي سنحصل عليه عند استبدال الرموز والضغط باستخدام طريقة LZW.

الآن وبالعودة إلى الطريقة المقترحة فإن الرمزين التاليين من حيث التكرار سيتم استبدالهما برمز جديد واحد، أي أن الرمز t والرمز o سيتم دمجهما برمز واحد يحمل مجموع تكراراتهما وهكذا حتى يتم استبدال جميع الرموز.

في **الجدول (5)** نوضح آلية استبدال الرموز من أجل الرموز العشرين الأولى في رواية هاملت مرتبة من حيث التكرار من الأكثر إلى الأقل.

وعلى سبيل المثال في مطلع رواية Hamlet توجد الجملة:

THE TRAGEDY OF HAMLET, PRINCE OF DENMARK

وما يقابلها من أرقام رموز ASCII:

<84><72><69><32><84><82><65><71><69><68><89><32><79><70><32><72><65><77><76><69><84><44><32><80><82><73><78><67><69><32><79><70><32><68><69><78><77><65><82><75>

وهنا تجدر الإشارة إلى أنه في نظام الترميز ASCII تكون الحروف الكبيرة باللغة الإنكليزية لها أرقام رموز مختلفة عن الحروف الصغيرة فقد تم إسناد الأرقام من 65 وحتى 90 للحروف الإنكليزية الكبيرة حسب ترتيبها الأبجدي من A حتى Z وتم إسناد الأرقام من 97 وحتى 122 للحروف الإنكليزية الصغيرة حسب ترتيبها الأبجدي من a حتى z كما هو موضح **بالجدول (3)**.

الجدول 5: طريقة استبدال الرموز.

الرمز الأصلي 8 bits	ASCII	الرمز الجديد 7 bits	الـ bit المميز
المسافة	32	0	0
e	101	0	1
t	116	1	0
o	111	1	1
a	97	2	0
s	115	2	1
n	110	3	0
h	104	3	1
i	105	4	0
r	114	4	1
l	108	5	0
d	100	5	1
تغذية السطر	10	6	0
u	117	6	1
m	109	7	0
.	46	7	1
y	121	8	0
,	44	8	1
w	119	9	0
f	102	9	1

ستتم عملية استبدال الرموز بكتابة الرموز الجديدة 7 bits إلى الملف الأول والـ bits المميزة إلى الملف الثاني، وستكون الرموز الجديدة على النحو التالي:

<14><13><21><0><14><24><15><20><21><25><25><0><17><22><0><13><15><19><20><21><14><8><0><21><24><14><23><24><21><0><17><22><0><25><21><23><19><15><24><22>

ومع كتابة كل رمز 7 bits ستتم كتابة الـ bit المميز لكل رمز إلى الملف الثاني، وستكون الـ bits المميزة للرموز السابقة على النحو التالي:

11001100001000101101101100000000000010110

وبعد إجراء عملية استبدال الرموز سيتم دمج الملفين بملف واحد نهائي وضغط رموز 7 bits باستخدام خوارزمية الضغط LZW أما في طرف الاستقبال وبعد فك ضغط الملف النهائي يتم فصل الملفين مجدداً حيث يتم قراءة عدد الرموز الذي تم إرساله بالترويسة وتقوم حلقة for بقراءة العدد المحدد من الرموز والتوقف عند انتهاء العدد وبهذا يكون انتهى الملف الأول ومن ثم يتم قراءة bits بنفس عدد الرموز ونكون حصلنا على الملف الثاني وبعدها تتم قراءة الرموز الأكثر تكراراً من الترويسة واسناد الرموز الجديدة لها حسب تكراراها وبعدها تتم عملية استعادة البيانات الأصلية من خلال قراءة كل رمز 7 bits من الملف الأول مع الـ bit المميز له من الملف الثاني واستعادة الرمز الأصلي ففي مثالنا السابق ستتم قراءة الرمز <14> من الملف الأول والـ bit 1 من الملف الثاني واستعادة الرمز الأصلي <84> أو الحرف T وهكذا حتى تتم استعادة كل الرموز الأصلية.

6. النتائج

هنا نستعرض النتائج لكل من خوارزمية LZW لوحدها وخوارزمية LZW مع استبدال الرموز.

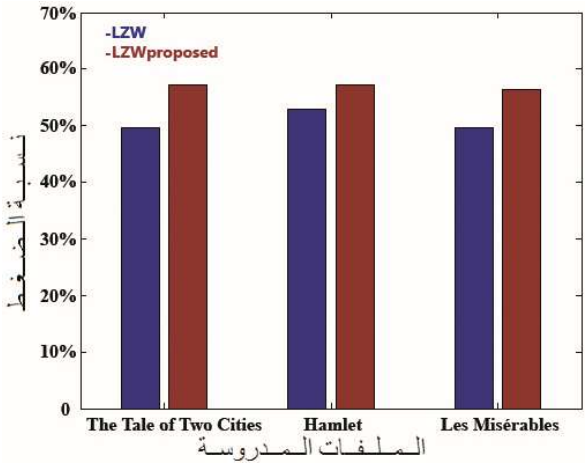
في **الجدول (6)** نستعرض مقارنة بين حجوم الملفات المدروسة الثلاثة قبل الضغط وبعد الضغط باستخدام كل من خوارزمية LZW على البيانات الأصلية وخوارزمية LZW على البيانات المبدلة.

الجدول 6:مقارنة بين أحجام الملفات المدروسة.

	الحجم الأصلي	LZW	$LZW_{proposed}$
The Tale of Two cities	769 KB	387 KB	328 KB
Hamlet	187 KB	88.2 KB	80.1 KB
Les Misérables	1.14 MB	594 KB	512 KB

أما في **الشكل (2)** نستعرض مقارنة بين نسبة الضغط لكل من خوارزمية LZW على البيانات الأصلية وخوارزمية LZW على البيانات المبدلة حيث تم حساب نسبة الضغط من خلال العلاقة (2) المقتبسة من المرجع[9]:

(1 – (compressed size / raw size)) × 100 (2)

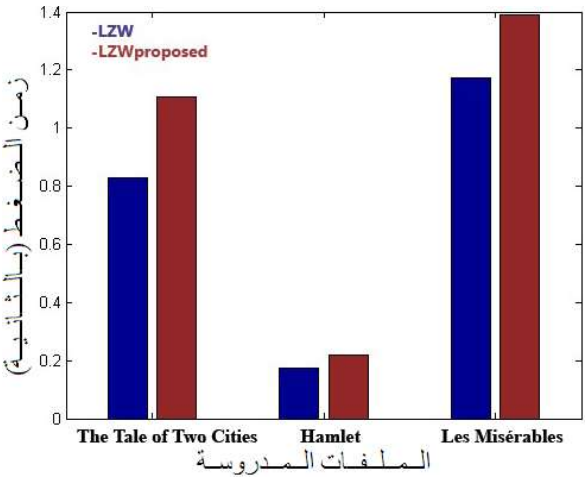


الشكل 2: مقارنة نسب الضغط للخوارزميات المدروسة.

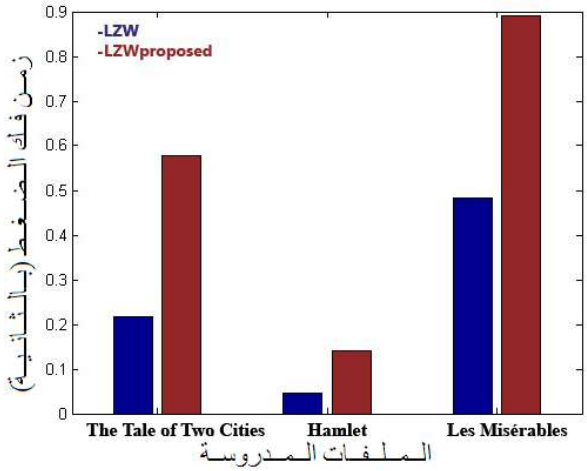
باستخدام خوارزمية LZW كانت نسبة ضغط رواية قصة مدينتين هي 49,546% ونسبة ضغط رواية هاملت هي 52,892% أما نسبة ضغط رواية البؤساء فكانت 49,539%.

بينما كانت نسبة الضغط باستخدام الطريقة المقترحة لرواية قصة مدينتين هي 57,227%، ونسبة ضغط رواية هاملت هي 57,206% أما نسبة ضغط رواية البؤساء فهي 56,45%.

في **الشكل (3)** تمت المقارنة بين زمن الضغط باستخدام خوارزمية LZW لوحدها وخوارزمية LZW مع استبدال الرموز بالثانية وفي **الشكل (4)** تمت مقارنة زمن فك ضغط خوارزمية LZW لوحدها مع زمن فك ضغط خوارزمية LZW مع استبدال الرموز بالثانية.



الشكل 3: مقارنة أزمنة الضغط للخوارزميات المدروسة.



الشكل 4: مقارنة أزمنة فك الضغط للخوارزميات المدروسة.

حيث كان زمن الضغط باستخدام خوارزمية LZW لرواية قصة مدينتين 0,828 ثانية ولرواية هاملت 0,172 ثانية ولرواية البؤساء 1,172 ثانية بينما كان زمن الضغط باستخدام الطريقة المقترحة لرواية قصة مدينتين 1,109 ثانية ولرواية هاملت 0,219 ثانية ولرواية البؤساء 1,391 ثانية.

أما زمن فك الضغط باستخدام خوارزمية LZW لرواية قصة مدينتين فهو 0,218 ثانية ولرواية هاملت 0,047 ثانية ولرواية البؤساء 0,484 ثانية بينما كان زمن فك الضغط باستخدام الطريقة المقترحة لرواية قصة مدينتين 0,578 ثانية ولرواية هاملت 0,141 ثانية ولرواية البؤساء 0,891 ثانية.

7. الاستنتاجات

ومنه نجد أنه بتغيير طبيعة البيانات من خلال استبدال الرموز بالطريقة المقترحة حصلنا على تحسين في نسبة ضغط البيانات بنسبة تتراوح من 5% وحتى 7%، وحصلنا على زيادة بسيطة في كل من زمن الضغط وزمن فك الضغط بسبب إضافة مرحلة استبدال الرموز والزمن الذي تستهلكه.

وبما أن هذه الخوارزميات لا تستخدم عندما يكون الزمن أمراً حاسماً وذلك لأن هذه الخوارزميات بالأصل وبدون أي إضافات تستهلك الكثير من الزمن في معالجة وإخراج الرموز المضغوطة (بناء شجرة كما في خوارزمية هافمان أو الحسابات الرياضية كما في الترميز الحسابي أو بناء القاموس كما في خوارزمية LZW) وهي تستخدم بشكل عام بغرض الضغط قبل الإرسال لتخفيض زمن الإرسال أو بغرض الأرشفة حيث يكون وسيط التخزين هو العنصر الأهم وليس الزمن لذلك فإن استهلاك بعض الزمن الإضافي مقابل التوفير في وسيط التخزين أو تخفيض زمن الإرسال يعتبر أمراً مناسباً.

References

[1] Anto, R., Ramachandran, R., "A Compression System for Unicode Files Using an Enhanced Lzw Method", *Pertanika Journal of Science & Technology,* 28(4), 1427-1444, October 2020.

[2] Dumas, J., Roch, J., Tannier, É., Varrette, S., *Foundations of Coding (Compression, Encryption, Error Correction)*, Wiley, 2015.

[3] Fano, R.M., The Transmission of Information, Technical Report No.65, Research Laboratory of Electronics at MIT, 1949.

[4] Hankerson, D., Harris, G., Johnson, P., Introduction to Information Theory and Data Compression, 2nd Ed, CRC Press LLC, 2003.

[5] Huffman, D., A Method for the Construction of Minimum-Redundancy Codes, Proceedings of the IRE, 40(9), 1098-1101, 1952.

[6] Moffat, A., Turpin, A., Compression and Coding Algorithms, Springer, 2012.

[7] Namburi, S., Rao, K., Muvva, P., Prasad, G., "A New Approach to Increase LZW Algorithm Compression Ratio", *International Journal of* Engineering Applied Sciences and Technology (IJEAST), 10(4), 141-144, February 2020.

[8] Namburi, S., Rao, K., Muvva, P., Kumar, S., Kumar, N., "An Efficient Method to Reduce LZW Algorithm Output Code Length", International Journal of Engineering Applied Sciences and Technology (IJEAST), 11(4), 302-304, March 2020.

[9] Nelson, M., Gailly, J., The Data Compression Book, 2nd Ed, IDG Books Worldwide, Cambridge, 1995.

[10] Salomon, D., A Concise Introduction to Data Compression, Springer, 2008.

[11] Salomon, D., Data Compression the Complete Reference, 4th Ed, Springer, 2007.

[12] Salomon, D., Variable-length Codes for Data Compression. Springer, 2007.

[13] Sayood, KH., Introduction to Data Compression. 3rd Ed, Elsevier, 2005.

[14] Shannon, C.E., A Mathematical Theory of Communication, Bell System Technical Journal, 27, 379-423, 1948.

[15] Venturini, R., Compressed Data Structures for Strings, Atlantis Press, 2013.

[16] Wayner, P., Data Compression for Real Programmers, Elsevier, 1999.

[17] Welch, T., A Technique for High-Performance Data Compression, IEEE Computer, 17(6), 8-19, 1984.

[18] Ziv, J., Lempel, A., Compression of Individual Sequences via Variable-Rate Coding, IEEE Transactions on Information Theory, 24(5), 530-536, 1978.

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

Study on impact of changing the nature of data on the file overall compression ratio

***Mohammad Samir Modabbes*[1], *Muhammad Amin Zayyat*[2,*]**

[1] *Department of Communication Engineering, Faculty of Electrical & Electronic Engineering, University of Aleppo, Syria, Modabbes@alepuniv.edu.sy.*

[2] *Department of Communication Engineering, Faculty of Electrical & Electronic Engineering, University of Aleppo, Syria, Aminzayyat@gmail.com.*

**Corresponding author: Muhammad Amin Zayyat, Aminzayyat@gmail.com.*

Published online: 31 March 2022

Abstract—This paper presents a study of a method to change the nature of the data by processing and replacing symbols in the pre-compression stage allowing to increase the frequency of symbols in the data to be compressed using one of the known compression algorithms, as this method replaces the most frequent symbol and the next most frequent symbol with one symbol carrying a sum of Their occurrences, then the next most frequent symbol with the one that is next to it is replaced with another symbol carrying the sum of their occurrences, and so on until all the symbols are replaced, this makes the number of the new symbols is half the number of the original ones, here in this research the ASCII coding system (American Standard Code for Information Interchange) has been studied. which consists of 256 characters of eight bits each, and to ensure that the data is not lost, each symbol has been referenced by one distinct bit written in a separate file, this bit will make restoring the original symbols possible. in the end, the two files merge together and seven-bits symbols are compressed using the LZW compression algorithm, the result is the final file contains more frequent symbols than the original file, as for decompression, it takes place in the opposite way, as the file is first decompressed using the LZW algorithm, and then the two files are separated from each other and a seven-bit symbol is read from the first file with a bit from the second file in order to restore the original symbol until all the original ASCII symbols are restored, also there is a comparison between one of the algorithms that compresses using entropy like Arithmetic Coding and LZW, and why will LZW give better results than usual statistical methods like Huffman or Arithmetic Coding.

Keywords— Compression Ratio, Lossless Compression, Symbol Replacement, Algorithm, Entropy.