UNIVERSIDAD TECNICA FEDERICO SANTA MARIA

Repositorio Digital USM **https://repositorio.usm.cl**

Tesis USM TESIS de Pregrado de acceso ABIERTO

2016

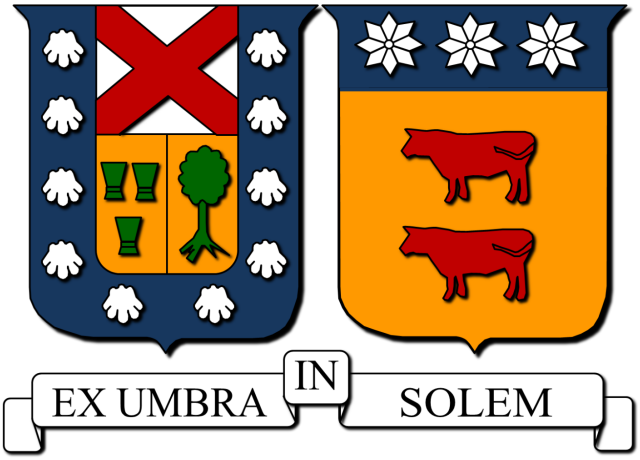
COMPRESIÓN DE DIGRAFOS BASADA EN METODOLOGÍAS DE RENUMERACIÓN DE VÉRTICES

ROJAS ZÚÑIGA, JORGE LUIS

http://hdl.handle.net/11673/19931

Repositorio Digital USM, UNIVERSIDAD TECNICA FEDERICO SANTA MARIA

UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE INFORMÁTICA
SANTIAGO - CHILE



"COMPRESIÓN DE DIGRAFOS BASADA EN METODOLOGÍAS DE RENUMERACIÓN DE VÉRTICES"

JORGE LUIS ROJAS ZÚÑIGA

MEMORIA DE TITULACIÓN PARA OPTAR AL TÍTULO DE INGENIERO CIVIL INFORMÁTICO

PROFESOR GUÍA: **DIEGO ARROYUELO**

PROFESOR CORREFERENTE: **ANDRÉS MOREIRA**

NOVIEMBRE - 2016

Agradecimientos

Durante toda mi vida, en cada vivencia y decisión, mis padres me han acompañado, tanto en lo que me han enseñado, como en la presencia y goce de un momento grato. Han estado conmigo siempre, entregándome lo que ellos han aprendido, y todo el cariño y amor incondicional que pueden dar. Soy quien soy por los principios que ellos sembraron en mí. Gracias.

Mis hermanas, mis cuñados, mis sobrinos, mis tios, y mi (gran) familia. Con todos tengo recuerdos preciados, en donde entendí que cada uno tiene un rol dentro del clan, y que siempre se ha de apoyar. Respetar las individualidades, pero estando siempre unidos, algunos más presentes, otros más implícitos. Siempre juntos, siempre pendientes. Gracias por el cariño y el apoyo, son tantos los recuerdos que se me es difícil resumirlos. Por eso, por el apoyo, las comidas y la alegría, gracias.

Cada amigo que se unió a mi camino y a mi vida, con los que compartí desvelos, juegos, carretes, karaokes, etc. sepan que hicieron todo más alegre, ameno y tranquilizante. Algunos desde los inicios de mi vida, otros incorporándose según crecí, pero siempre unidos, disfrutando una cerveza, riéndonos de todo, levantándonos cuando lo necesitamos, gracias.

A ti amor, por la paciencia, el apoyo constante y el ayudarme a retomar el camino la veces que rompí la constancia. Por ser motivación y paz. Por todo lo que has sido, que para mí era lo que necesitaba. Gracias, te amo.

A los que me han acompañado y se mantuvieron conmigo durante todo el camino. Infinitas gracias.

Resumen

El presente trabajo muestra un estudio sobre cómo el uso de metodologías para la renumeración y reordenamiento de vértices puede afectar y mejorar la tasa de compresión utilizando algoritmos de compresión basados en índices invertidos sobre grafos dirigidos. Estos algoritmos no están diseñados inicialmente para este contexto, por lo que se busca una comparación entre estos y las herramientas actuales creadas específicamente para comprimir estas estructuras de datos. También tienen la capacidad de descomprimir cientos de miles de nodos por segundo, lo cual hace que sean buenos candidatos para comparar. El objetivo será analizar las tasas de compresión utilizando para eso el grafo escrito por listas de adyacencia escritas en orden por grado de salida, entrada, comunidades con grado de salida, comunidades con grado de entrada y aleatorio para contrastar la importancia del orden de los nodos. A su vez se medirá el tiempo que demora el comprimir como descomprimir los archivos binarios obtenidos.

Palabras Clave: Compresión de grafos, Renumeración, Comunidades, Grafos dirigidos.

Abstract

This work shows a study about how the use of renumbering and reordering methodologies over vértices can impact and improve the compression rate using compression algorithms based on inverted index over directed graphs. These algorithms are not initially designed for this context, thus it seeks to compare this with updated tools created specifically to compress this data structure. The inverted index compression algorithms have the ability to decompress hundreds of thousands of nodes per second, and this makes them good candidates to compare. The objective will be to analyze the compression rates using out-degree order, in-degree order, community with out-degree order, community with in-degree order and a no order to contrast them. Also, the time that will take to compress and decompress will be measured.

Keywords: Compression graph, Renumerate, Community, Directed Graph.

Tabla de Contenido

Introducción **1**

1 Antecedentes **3**

1.1 Definición del Problema 3

1.2 Objetivos 4

1.2.1 Objetivo Principal 4

1.2.2 Objetivos Específicos 4

2 Estado del Arte **6**

2.1 Definición de Grafo 6

2.2 Propiedades Relevantes de los Grafos 8

2.3 Comunidades en los Grafos 9

2.4 Utilidad en la Práctica 10

2.5 Métodos de compresión de grafos 11

2.5.1 Compresión de Grafos Usando *Copylist* y *Gaps* [5][11] 11

2.5.2 Compresión por Referencia 13

2.5.3 Compresión Diferenciada 14

2.5.4 Usando Intervalos para Explotar la Consecutividad 15

2.5.5 Representaciones Aproximadas 16

2.5.6 Compresión Manteniendo una Estructura Similar 17

2.6 Definición de Índice Invertido: 18

2.7 Métodos de Compresión para Índices Invertidos 19

2.7.1 VByte [16] 19

2.7.2 S9 y S17 [1] [2] 20

2.7.3 PForDelta [18] 22

2.7.4 Interpolative [12] 22
2.7.5 *Run-length* Encoding [15] 23

3 Diseño del Experimento **25**

3.1 Alcances 25
3.2 Métricas 25
3.3 Datasets 26
3.4 Proceso de Compresión Propuesto 27
3.4.1 Formato 28
3.4.2 Renumeración 30
3.4.3 Compresión 31

4 Presentación de Resultados **33**

4.1 Compresión sin Tabla de Renumeración 33
4.1.1 *Outdegree* 34
4.1.2 *Indegree* 34
4.2 Compresión con Tabla de Renumeración 35
4.2.1 Amazon-2008 36
4.2.2 CNR-2000 37
4.2.3 Eswiki-2013 38
4.2.4 Ljournal-2008 39
4.2.5 web-Stanford 40
4.3 Compresión del 80 % de los Vértices 41
4.3.1 Amazon-2008 42
4.3.2 CNR-2000 43
4.3.3 Eswiki-2013 44

4.3.4 Ljournal-2008 45
4.3.5 web-Stanford 46
4.4 Tiempo de Compresión 47
4.5 Tiempo de Descompresión 49
4.6 Listas de Renumeración Voraz 50

5 Análisis de Resultados y Procesos **51**
5.1 Métodos de Compresión 52
5.1.1 VByte y RL+VByte 53
5.1.2 S9 y RL+S17 53
5.1.3 PForDelta y RL+PForDelta 54
5.1.4 Interpolative 55
5.2 Reordenamiento 56
5.3 Renumeración 56

6 Conclusiones **59**

Referencias Bibliográficas **63**

Anexos **65**
6.1 Tiempo Compresión Amazon-2008 65
6.2 Tiempo Compresión CNR-2000 67
6.3 Tiempo Compresión Eswiki-2013 69
6.4 Tiempo Compresión Ljournal-2008 71
6.5 Tiempo Compresión web-Stanford 73

Índice de Figuras

Figura 1 grafo G compuesto de 11 nodos y 17 arcos. 6

Figura 2 grafo G con su matriz de adyacencia M correspondiente. 7

Figura 3 grafo G con algunas de sus listas de adyacencia l_i respectivas. 7

Figura 4 digrafo G con su conjunto de listas de adyacencia ordenadas L respectivo. 8

Figura 5 representación del grafo G usando el métodoBV+. Imagen extraída de [11]. 16

Figura 6 transformación de un grafo web G a k^2 tree. Imagen extraída de [6]. . . 18

Figura 7 ejemplo de un índice invertido I, con su vocabulario Σ y su colección de documentos D. 19

Figura 8 proceso de compresión en base a S9. 21

Figura 9 ejemplo codificación Run-length. 24

Figura 10 proceso de compresión propuesto para un grafo G. 28

Figura 11 ejemplo de transformación de formato. 29

Figura 12 gráfico resumen compresión de Amazon-2008 en formato Outdegree. . 66

Figura 13 gráfico resumen compresión de Amazon-2008 en orden Indegree. . . . 66

Figura 14 gráfico resumen compresión de CNR-2000 en formato Outdegree. . . 68

Figura 15 gráfico resumen compresión de CNR-2000 en orden Indegree. 68

Figura 16 gráfico resumen compresión de Eswiki-2013 en formato Outdegree. . . 70

Figura 17 gráfico resumen compresión de Eswiki-2013 en orden Indegree. 70

Figura 18 gráfico resumen compresión de Ljournal-2008 en formato Outdegree. . 72

Figura 19 gráfico resumen compresión de Ljournal-2008 en orden Indegree. . . . 72

Figura 20 gráfico resumen compresión de web-Stanford en formato Outdegree. . 74

Figura 21 gráfico resumen compresión de web-Stanford en orden Indegree. . . . 74

Índice de Tablas

Tabla 1 representación de un grafo *G* usando listas de adyacencia [5]. 12

Tabla 2 representación de las listas de adyacencia del grafo *G* (Tabla 1) usando gaps [5]. 13

Tabla 3 representación de un grafo *G* usando compresión por referencia [5]. . . 14

Tabla 4 representación de un grafo *G* usando compresión diferenciada[5]. . . . 14

Tabla 5 representación de un grafo *G* usando compresión diferenciada e intervalos [5]. 15

Tabla 6 codificación de un índice invertido usando Interpolative[12]. 23

Tabla 7 resultados de compresión (bits por arco) usando formato Outdegree sin el uso de tabla de renumeración. 34

Tabla 8 resultados de compresión (bits por arco) usando formato Indegree sin el uso de tabla de renumeración. 34

Tabla 9 resultados de compresión (en bits por arco) con formato Outdegree de Amazon-2008 usando distintas tablas de renumeración. 36

Tabla 10 resultados de compresión (en bits por arco) con formato Indegree de Amazon-2008 usando distintas tablas de renumeración. 36

Tabla 11 resultados de compresión (en bits por arco) con formato Outdegree de CNR-2000 usando distintas tablas de renumeración. 37

Tabla 12 resultados de compresión (en bits por arco) con formato Indegree de CNR-2000 usando distintas tablas de renumeración. 37

Tabla 13 resultados de compresión (en bits por arco) con formato Outdegree de Eswiki-2013 usando distintas tablas de renumeración. 38

Tabla 14 resultados de compresión (en bits por arco) con formato Indegree de Eswiki-2013 usando distintas tablas de renumeración. 38

Tabla 15 resultados de compresión (en bits por arco) con formato Outdegree de Ljournal-2008 usando distintas tablas de renumeración. 39

Tabla 16 resultados de compresión (en bits por arco) con formato Indegree de Ljournal-2008 usando distintas tablas de renumeración. 39

Tabla 17 resultados de compresión (en bits por arco) con formato Outdegree de web-Stanford usando distintas tablas de renumeración. 40

Tabla 18 resultados de compresión (en bits por arco) con formato Indegree de web-Stanford usando distintas tablas de renumeración. 40

Tabla 19 cantidad de listas donde se obtiene el 80 % de los arcos de cada dataset, más el largo de la última lista tomada, para cada tipo de formato. 41

Tabla 20 resultados compresión del 80 % de las aristas de Amazon-2008 para las distintas tablas de renumeración usadas, en formato Outdegree. 42

Tabla 21 resultados compresión del 80 % de las aristas de Amazon-2008 para las distintas tablas de renumeración usadas, en formato Indegree. 42

Tabla 22 resultados compresión del 80 % de las aristas de CNR-2000 para las distintas tablas de renumeración usadas, en formato Outdegree. 43

Tabla 23 resultados compresión del 80 % de las aristas de CNR-2000 para las distintas tablas de renumeración usadas, en formato Indegree. 43

Tabla 24 resultados compresión del 80 % de las aristas de Eswiki-2013 para las distintas tablas de renumeración usadas, en formato Outdegree. 44

Tabla 25 resultados compresión del 80 % de las aristas de Eswiki-2013 para las distintas tablas de renumeración usadas, en formato Indegree. 44

Tabla 26 resultados compresión del 80 % de las aristas de Ljournal-2008 para las distintas tablas de renumeración usadas, en formato Outdegree. 45

Tabla 27 resultados compresión del 80 % de las aristas de Ljournal-2008 para las distintas tablas de renumeración usadas, en formato Indegree. 45

Tabla 28 resultados compresión del 80 % de las aristas de web-Stanford para las distintas tablas de renumeración usadas, en formato Outdegree. 46

Tabla 29 resultados compresión del 80 % de las aristas de web-Stanford para las distintas tablas de renumeración usadas, en formato Indegree. 46

Tabla 30 resumen tiempo promedio de compresión en [μseg] usando formato Outdegree. 47

Tabla 31 resumen tiempo promedio de compresión en [μseg] usando formato *Indegree*. 48

Tabla 32 cantidad de arcos comprimidos por segundo, para cada método. 48

Tabla 33 resumen tiempo promedio de descompresión para cada método de compresión. 49

Tabla 34 resumen proceso de renumeración voraz. 50

Tabla 35 tiempo de compresión en [seg] usando formato Outdegree para Amazon-2008, aplicando todos las formas de renumeración mencionadas. 65

Tabla 36 tiempo de compresión en [seg] usando formato Indegree para Amazon-2008, aplicando todos las formas de renumeración mencionadas. 65

Tabla 37 tiempo de compresión en [seg] usando formato Outdegree para CNR-2000, aplicando todos las formas de renumeración mencionadas. 67

Tabla 38 tiempo de compresión en [seg] usando formato Indegree para CNR-2000, aplicando todos las formas de renumeración mencionadas. 67

Tabla 39 tiempo de compresión en [seg] usando escritura Outdegree para Eswiki-2013, aplicando todos las formas de renumeración mencionadas. 69

Tabla 40 tiempo de compresión en [seg] usando formato Indegree para Eswiki-2013, aplicando todos las formas de renumeración mencionadas. 69

Tabla 41 tiempo de compresión en [seg] usando formato Outdegree para Ljournal-2008, aplicando todos las formas de renumeración mencionadas. 71

Tabla 42 tiempo de compresión en [seg] usando formato Indegree para Ljournal, aplicando todos las formas de renumeración mencionadas. 71

Tabla 43 tiempo de compresión en [seg] usando formato Outdegree para web-Stanford, aplicando todos las formas de renumeración mencionadas. 73

Tabla 44 tiempo de compresión en [seg] usando formato Indegree para web-Stanford, aplicando todos las formas de renumeración mencionadas. 73

Introducción

La compresión de datos es el proceso de reducir el volumen ocupado en el almacenamiento de los mismos. Ésta principalmente se basa en detectar regularidades en los datos, las cuales se usan para comprimir. Existen distintos métodos que aprovechan especificaciones tanto de la estructura a comprimir como el contexto de ésta, por lo que no es extraño encontrar diferentes heurísticas que solucionen dicha tarea.

Por otro lado, una de las estructuras de datos más utilizada para modelar problemas son los grafos, ya que por sus carácterísticas son capaces de adaptarse y representar bien las relaciones entre individuos o entidades distintas. Sin embargo, el mantenimiento y el hacer operaciones sobre ésta resulta muy costoso dado que los grafos pueden crecer a cientos de millones de arcos, y miles de millones de aristas. Un ejemplo de esto es la World Wide Web, la cual posee 4.47 mil millones de páginas web [1].

Dada esta problemática y el conjunto de soluciones existentes, esta memoria busca hacer un análisis comparativo entre dos herramientas utilizadas para comprimir grafos, con otras usadas para comprimir índices invertidos en el área de recuperación de la información que han demostrado un buen rendimiento. La compatibilidad en su uso radica en que ambas estructuras de datos se pueden procesar por metodologías de compresión de índices invertidos debido a que los grafos se pueden escribir como estos.

El primer capítulo de esta memoria trata de la utilidad de los grafos como también de la definición del problema a tratar, junto con los objetivos de la misma. El segundo capítulo consiste en un análisis a la fecha de las metodologías aplicadas actuales que abordan la compresión de grafos, como también de las propiedades de esta estructura. El tercer capítulo expone los datasets y la infraestructura usados para el experimento, como también el modelo del proceso de compresión aplicado para cada metodología estudiada. El cuarto capítulo

[1] Tamaño obtenido al 14 de Octubre del 2016. `http://www.worldwidewebsize.com/`

presenta los resultados para cada una de las iteraciones realizadas. El quinto capítulo expone un análisis de los resultados, además de las configuraciones realizadas para el proceso de compresión. Finalmente el sexto capítulo presenta las conclusiones y apreciaciones de las herramientas estudiadas.

1. Antecedentes

1.1. Definición del Problema

Los grafos son utilizados en una gran cantidad de escenarios, como por ejemplo: redes sociales; internet; redes biológicas; redes de alimentación de energía eléctrica; etc. Debido al desarrollo de la tecnología y la necesidad de modelar cada vez más y de mejor manera problemas del presente, el tamaño de estos ha ido en aumento. Algunos de estos grafos poseen millones de nodos y cientos de millones de arcos. Producto de esto, el almacenar estas representaciones, como también aplicar consultas y minería de datos en ellas, se hace más difícil de lograr debido a las limitaciones de los patrones de minería de grafos. Además, trabajar con estructuras de datos de tales magnitudes está fuera de nuestras capacidades humanas.

Es por este propósito por el cual existen múltiples heurísticas que buscan abordar de mejor manera este problema utilizando para ello diferentes aproximaciones: algunos eligen una lista de adyacencia que sea lo suficientemente representativa dentro de un segmento de estas para usarla de referencia dentro de otros nodos en una misma vecindad; otros usan la representación matricial del grafo para poder sacarle provecho, usando divisiones en distintos segmentos, como también reordenando las filas y las columnas para poder concentrar en la diagonal la mayor cantidad de información relacionada y luego transformarla; mientras que hay otros que hacen una representación aproximada del grafo, agrupando un conjunto de nodos densamente conectado el cual es reemplazado por un nodo virtual, el que tiene su propia estructura y correcciones internas; además de métodos que transforman el grafo en listas de nodos disminuyendo la redundancia. Se ahondará más sobre estos métodos en la siguiente unidad. En su mayoría todos buscan minimizar la cantidad de espacio requerido para almacenarlos haciendo un intercambio con la velocidad de compresión y descompresión, aprovechando a su vez la capacidad de realizar consultas y operaciones sobre la estructura de datos comprimida, ya que es ineficiente tener que elaborar el grafo completo para realizar operaciones sobre este en memoria principal.

Dada la gama de herramientas específicas que existen para la compresión de grafos, y considerando la compatibilidad en la forma de poder escribir estas estructuras en forma de otras, es que surge el problema de estudiar el comportamiento de los compresores de otras estructuras, en particular de índices invertidos, sobre los grafos. Son escasos los estudios que hacen una comparación entre estos procesos para ver el rendimiento y ventajas existentes. Es por eso que este trabajo entrega un análisis comparativo entre herramientas de compresión de índices invertidos y grafos, haciendo experimentos con los primeros sobre diversos grafos dirigidos de distinta magnitud, usando un proceso compuesto de compresión basada en renumeración de vértices.

1.2. Objetivos

A continuación se declaran los objetivos principales de este trabajo:

1.2.1. Objetivo Principal

- Realizar un estudio comparativo de compresión de grafos dirigidos usando diferentes métodos de compresión de índices invertidos, detallando los escenarios en donde mejor y peor se desenvuelve cada método, utilizando además métodos de renumeración de nodos en base a distintos órdenes de escritura, basados en características del grafo.

1.2.2. Objetivos Específicos

Para lograr la realización del objetivo principal, se espera satisfacer lo siguiente:

- Realizar un estudio de velocidad de compresión y descompresión para cada método de compresión de índices invertidos usado, contrastado con su tasa de compresión.

- Comprender las optimizaciones posibles a realizarse en cada método de compresión basándose en la renumeración de nodos.
- Analizar cómo el reordenamiento de nodos afecta directamente en el desempeño de los algoritmos de compresión.
- Conocer bajo qué escenarios y parámetros los métodos de compresión de índices invertidos tienen un mejor desempeño sobre los grafos dirigidos.
- Comparar los algoritmos de compresión de índices invertidos con algunos métodos específicos de compresión de grafos del estado del arte.

2. Estado del Arte

En este capítulo se expone la definición de grafo y digrafo, representaciones comunes y sus aplicaciones en la práctica al igual que algunas propiedades de estos. Además se presenta un estado del arte sobre las diversas técnicas ya existentes para la compresión de grafos, y diversos métodos de compresión de índices invertidos para hacer contraste entre ambas familias.

2.1. Definición de Grafo

Se define un grafo $G = (V, S)$ como el conjunto finito no vacío de nodos (o vértices) $V = \{v_1,\ v_2,\ ...,\ v_n\}$ y el subconjunto S como el conjunto de pares (no ordenado) de vértices. Cada elemento de S recibe el nombre de arco (o arista). Los arcos se representan de la forma $\{v_i,\ v_j\}$ e indican un tipo de relación entre los nodos v_i y v_j a los cuales conecta. Los arcos pueden tener asociado un valor numérico al que se le denomina *peso*.

Se dice que $\forall\ v_i, v_j \in V$, un nodo v_i es vecino de v_j si existe el arco $\{v_i,\ v_j\} \in S$ que los conecta. Si un nodo tiene asociado d arcos, se dice que posee grado d.

Como ejemplo, en la Figura 1, el nodo 1 tiene como vecinos a los nodos 2 y 3, y su grado es 2.

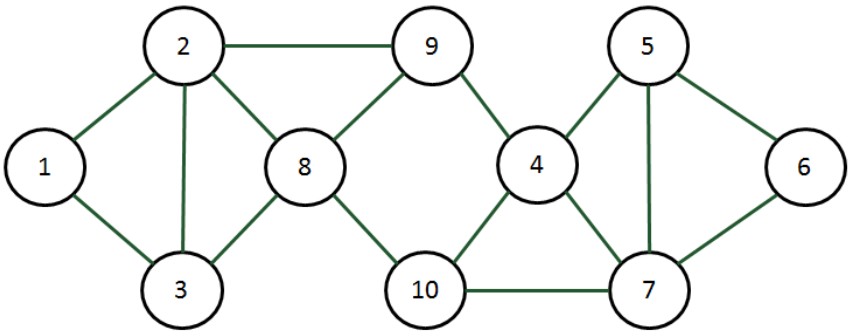


Figura 1: grafo G compuesto de 11 nodos y 17 arcos.

Como se ve en la Figura 2, un grafo se puede representar a través de una matriz de adyacencia $M_a(G)$, la cual en su elemento $m_{i,j}$ muestra si hay una arista entre los nodos v_i y $v_j \in V$, almacenando 0 en caso de no existir la arista, y 1 en caso contrario.

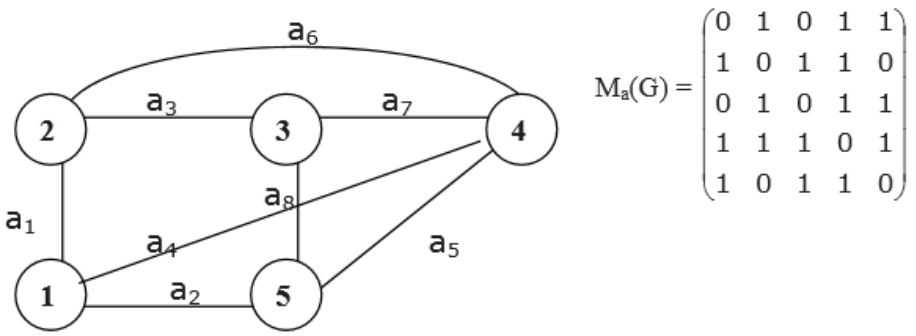


Figura 2: grafo G con su matriz de adyacencia M correspondiente.

Por otra parte, dado que el vértice v_i puede tener d nodos vecinos (o adyacentes), una alternativa para representar un grafo son las *listas de adyacencia*. Para un vértice v_i se define la lista l_i que contiene todos los vértices adyacentes a v_i. El grafo es representado mediante el conjunto de todas las listas l_1, ..., l_n denominado L. Cada lista l_i almacena los vértices ordenados de forma creciente. Ésta será la representación usada en esta memoria. La Figura 3 explica gráficamente este concepto.

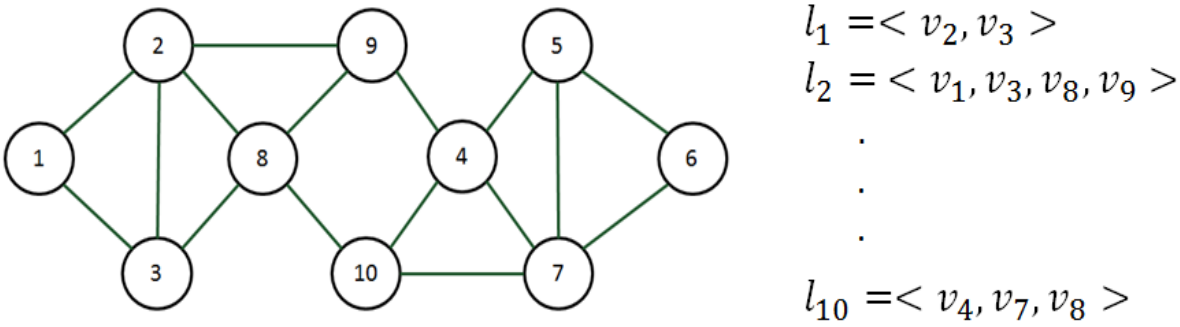
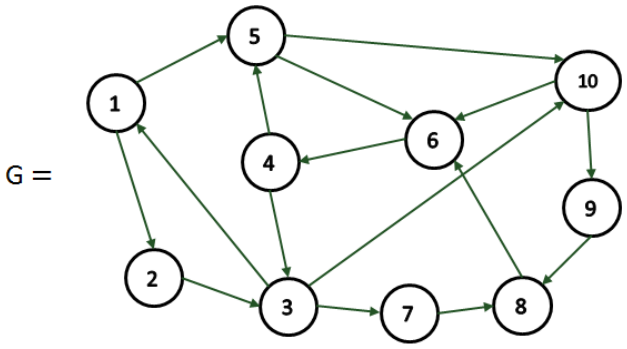


Figura 3: grafo G con algunas de sus listas de adyacencia l_i respectivas.

De forma particular, dentro de los tipos de grafos existentes, se tiene el digrafo o grafo dirigido, que son aquellos en donde los arcos tienen un nodo origen y uno destino, pudiendo

ser ambos el mismo nodo. Estos grafos son utilizados para representar un tipo de relación que poseen el par de nodos involucrados en base al sentido del arco que los une. Ejemplo de esto se puede ver en la Figura 4, en donde se expone además las listas de adyacencia l_i de cada vértice v_i. Notar que al ser un grafo dirigido, las listas de adyacencia contienen sólo los nodos a los que el vértice v_i apunta, por lo que se les nombra listas de adyacencia por nodos de salida. Así mismo, las listas de adyacencia pueden contener solo los nodos que apuntan hacia v_i, y estas son nombradas listas de adyacencia por nodos de entrada.



$$L = \begin{array}{l} l_1 = < v_2, v_5 >, \\ l_2 = < v_3 >, \\ l_3 = < v_1, v_7, v_{10} >, \\ l_4 = < v_3, v_5 >, \\ l_5 = < v_6, v_{10} >, \\ l_6 = < v_4 >, \\ l_7 = < v_8 >, \\ l_8 = < v_6 >, \\ l_9 = < v_8 >, \\ l_{10} = < v_6, v_9 > \end{array}$$

Figura 4: digrafo G con su conjunto de listas de adyacencia ordenadas L respectivo.

2.2. Propiedades Relevantes de los Grafos

Dado el contexto en el cual se utilizan los grafos en la práctica, se han definido algunas propiedades útiles que ayudan a entender la estructura de estos [11]. A continuación se presentan las más importantes:

- *Localidad*: los nodos tienden a tener sucesores que son "cercanos" a ellos en un sentido que depende del contexto y la naturaleza de la conexión de todo el grafo.
- *Similitud*: nodos que suelen estar "cerca" entre ellos tienden a poseer muchos sucesores comunes.

- *Grado de entrada/salida*: se refiere a la cantidad de arcos que entran/salen de un nodo respectivo. Por su traducción en inglés se le llama *Outdegree/Indegree*.

Todas estas propiedades son producto del estudio de la *topología* del grafo. Vale decir, el análisis de la forma en que la red está constituida de forma natural, en donde se genera una distribución de nodos y arcos que hacen que estos tengan distintas características, formas y particularidades.

2.3. Comunidades en los Grafos

Una de las características principales que tienen los grafos es que, debido a su topología, estos forman comunidades que resultan en subgrafos que comparten una alta conexión dentro de su grupo, pero están más pobremente conectados con nodos de otros grupos [13]. Es por esto mismo que en la mayoría de los casos es útil detectarlas para tener una mayor comprensión de la red. Sin embargo el problema de detectar y segmentar la red en comunidades bien definidas es NP-Díficil. Como consecuencia de esto se utilizan comunmente heurísticas voraces que solucionan este problema de forma práctica.

Una métrica llamada *modularidad* (Q), la cual explica la fuerza de la división de una red en subgrafos, indica cuándo los nodos tienen una fuerte conexión dentro de una comunidad, pero escasa fuera de ésta, lo que ayuda en la separación e identificación de comunidades a través de la división. Varios de los métodos que buscan la detección de comunidades, intentan iterativamente incrementar la modularidad entre nodos, haciendo que estos se vayan diferenciando y agrupando entre sus pares.

Existen diversas heurísticas que permiten realizar una partición en comunidades. Entre estos están [9]:

- *Espectrales*: aprovechan la forma matricial de los grafos para calcular los valores y vectores propios del laplaciano.

- *Divisivos*: cortan aristas según un parámetro determinado, y con esto van creando comunidades.
- *Aglomerativos*: usan el grafo sin arcos y luego van agregando aristas según alguna métrica.
- *Basados en modularidad*: buscan incrementar los valores de Q dado que esto indica un buen particionamiento del grafo.
- *Dinámicos*: utilizando un proceso que va recorriendo el grafo para detectar comunidades.
- *Estadísticos*: busca deducir propiedades del grafo a través de observaciones y modelos de hipótesis, haciendo que ésta calce con la topología del grafo.
- *Sobreposición de comunidades*: además de detectar comunidades, detectan los nodos que pertenecen a más de un cluster.

El algoritmo de *Louvain* [3], perteneciente a los métodos divisivos, ha demostrado ser empíricamente eficiente y eficaz, dado que tiene un tiempo de ejecución de $\mathbf{O}(n \log n)$ [2]. El mismo utiliza la modularidad para hacer un particionamiento del grafo en módulos a través de dos etapas: una que optimiza localmente la modularidad haciendo que los nodos vecinos atraigan vértices hacia ellos; y la segunda que agrupa los nodos en un súper nodo virtual que representa a la comunidad.

2.4. Utilidad en la Práctica

Actualmente los grafos se pueden encontrar asociados a muchas aplicaciones. Desde redes sociales, programas para monitorear mallas eléctricas, representaciones de redes biológicas, e incluso la web son representados como un grafo. Estos permiten representar a una

[2]Por convención durante toda esta memoria al referirise a logaritmo será en base dos y en función techo ($\lceil \log_2 n \rceil$), a menos que se especifique lo contrario.

entidad y sus relaciones con otros pares de forma flexible y adaptable, lo que facilita su manejo y distribución. Sin embargo, algunos grafos pueden aumentar de tamaño, llegando a tener millones de nodos, y aún más aristas que los conecten. Por ejemplo, el grafo asociado a Twitter, el cual modela la relación entre usuarios de esta plataforma, al año 2010 [11] poseía aproximadamente 41 millones de nodos y 1,468 millones de aristas. Como consecuencia de esto, surgen problemas de almacenamiento, además de cómo manipular el grafo de forma eficiente para operar sobre él, como por ejemplo recorrerlo. De esto nacen algoritmos que buscan comprimir estas estructuras de datos en representaciones más pequeñas, las cuales ocupan menos espacio de almacenamiento, pero buscando ser funcionales, es decir, la capacidad de obtención de información relacionada a alguna consulta hecha sobre el grafo.

Cabe destacar que el concepto de compresión se entiende como el proceso reversible de transformación de una estructura inicial a otra final, en donde ésta es una representación que utiliza menos espacio de almacenamiento.

2.5. Métodos de compresión de grafos

A continuación se exponen algunas de las técnicas más utilizadas en la literatura para la compresión de grafos.

2.5.1. Compresión de Grafos Usando *Copylist* y *Gaps* [5][11]

Una de las representaciones usadas para la compresión de grafos descrita en [5] y [11] es el uso de listas de adyacencia, gaps y copylist.

En el caso de Webgraph Framework[5] (también llamado BV por sus autores), se hacen referencia a distintos tipos de compresión, los cuales aprovechan las propiedades de similitud

y localidad (ver Sección 2.2) para mejorar las tasas de compresión.

• **Listas de adyacencia:** El uso de una matriz de adyacencia como representación para un grafo de V nodos implica un uso de memoria alto en almacenamiento, el cual no es aprovechado completamente ya que comúnmente los nodos no estan conectados todos con todos. Esto se evidencia con valores 0 que aparecen en la matriz. Para evitar esto, se asigna a cada nodo del grafo una lista de adyacencia (ver Seccion 2.1) en donde se indican sus nodos vecinos. En un grafo de V nodos, la eficiencia para cada nodo v_i con grado d, $d < V$ arcos relacionados corresponde a $(V - d)$. Finalmente se agrega también el grado de cada nodo. La Tabla 1 ejemplifica el método.

Tabla 1: representación de un grafo G usando listas de adyacencia [5].

Nodos	**Outdegree**	**Sucesores**
...	...	...
15	11	13, 15, 16, 17, 18, 19, 23, 24, 203, 315, 1034
16	10	15, 16, 17, 22, 23, 24, 315, 316, 317, 3041
17	0	
18	5	13, 15, 16, 17, 50
...	...	...

• ***Gaps*:** Aprovechando que se puede escribir el digrafo en base a sus listas de adyacencia, y en conjunto con la propiedad de localidad, se crea una nueva representación utilizando gap's, los cuales corresponden a una codificación usando la siguiente regla: si x_i es el grado del nodo v_i, entonces la lista de sucesores $S(x_i) = s_1, s_2, \ldots, s_d$ se le representa como $(s_1 - x_i, s_2 - s_1 - 1, s_3 - s_2 - 1, \ldots, s_d - s_{d-1} - 1)$. El primer término se trabaja de forma distinta dado que puede arrojar un número negativo, por lo que se utiliza una función de mapeo $v(x)$ para evitar esto:

$$v(x) = \begin{cases} 2x & x \geq 0 \\ 2|x| - 1 & x < 0 \end{cases}$$

De esta forma, observando la Tabla 2, se aprecian listas de $0's$ que evidencian la proximidad (localidad) entre los nodos:

Tabla 2: representación de las listas de adyacencia del grafo G (Tabla 1) usando gaps [5].

Nodos	**Outdegree**	**Sucesores**
...	...	...
15	11	3, 1, 0, 0, 0, 0, 3, 0, 178, 111, 718
16	10	1, 0, 0, 4, 0, 0, 290, 0, 0, 2723
17	0	
18	5	9, 1, 0, 0, 32
...	...	...

2.5.2. Compresión por Referencia

Explotando la propiedad de similitud, se puede hacer una representación aún más compacta del grafo. Se aplica el uso de una nueva lista llamada *copylist*, la cual es un arreglo de $0's$ y $1's$, que según el orden de una lista de adyacencia usada como referencia, señala si el respectivo vértice está dentro de los vecinos del nodo en cuestión. Además, se agrega una lista de nodos extra de excepción, los cuales corresponden a los vértices que están fuera de la copylist, pero están asociados al nodo en cuestión. Finalmente, se añade un campo de referencia, que indica la distancia entre la copylist y la lista de adyacencia inicial.

La Tabla 3 presenta el resultado de este método, en donde la similitud se refleja en la baja cantidad de nodos en la lista de nodos extra.

Tabla 3: representación de un grafo G usando compresión por referencia [5].

Nodos	Outdegree	Referencia	Copylist	Nodos extra
...	...	...	...	...
15	11	0		13, 15, 16, 17, 18, 19, 23, 24, 203, 315, 1034
16	10	1	01110011010	22, 316, 317, 3041
17	0			
18	5	3	11110000000	50
...	...	...	...	...

Dado que las copylist son lo más representativas posibles, la cantidad de nodos que se dejan de escribir explícitamente ayuda a que la compresión sea más efectiva.

2.5.3. Compresión Diferenciada

Utilizando tanto las listas de adyacencia como las copylist, se crean *copyblocks*, los cuales almacenan secuencias de $0's$ y $1's$, además de especificar el largo de cada bloque disminuido en 1, excepto para el primero. Las ventajas que posee este tipo de codificación es que permite comprimir un arco en menos de 1 bit (en promedio).

La Tabla 4 muestra cómo el ejemplo anterior es usado y transformado usando compresión diferenciada.

Tabla 4: representación de un grafo G usando compresión diferenciada[5].

Nodos	Outdegree	Referencia	# Bloques	Copyblocks	Nodos extra
...	...	...	...	...	...
15	11	0			13, 15, 16, 17, 18, 19, 23, 24, 203, 315, 1034
16	10	1	7	1, 2, 1, 1, 0, 0, 0	22, 316, 317, 3041
17	0				
18	5	3	1	4	50
...	...	...	...	...	...

2.5.4. Usando Intervalos para Explotar la Consecutividad

En esta técnica se toman los intervalos de nodos consecutivos que estén sobre un límite inferior establecido L_{min}. Con esto, se aprovecha la propiedad de localidad, por lo que listas consecutivas de nodos se resumen en un gap. Los intervalos son tratados utilizando los extremos izquierdos y derechos. Se resta el extremo izquierdo con el próximo derecho más dos. En el caso del primer intervalo, este se resta consigo mismo solamente. Estos valores se utilizan como los nuevos extremos izquierdos.

Además de guardar los extremos, se almacenan los nodos residuales que no poseen un intervalo mayor a L_{min}, la cantidad de intervalos y el largo de cada uno ordenados correspondientemente. La Tabla 5 expone un ejemplo de este método.

Tabla 5: representación de un grafo G usando compresión diferenciada e intervalos [5].

Nodos	Outdegree	Ref.	# Bloques	Copy blocks	# Intervalos	Extremo izq.	Largo	Residuos
...	...	...	...	...	...	...	...	...
15	11	0			2	0, 2	3 , 0	5, 189, 111, 718
16	10	1	7	0, 0, 2, 1, 1, 0, 0	1	600	0	12, 3018
17	0							
18	5	3	1	4	0			50
...	...	...	...	...	...	...	...	...

Debido al tamaño de los grafos con los que trabaja WebGraph framework, éste utiliza algoritmos perezosos los cuales no descomprimen completamente la representación final, sino que iteran a partir de una lista inicial hasta que encuentran la información requerida. De esta forma se ahorra en trabajo de búsqueda, mejorando los tiempos de respuesta a las consultas hechas sobre el grafo.

Por otro lado, trabajos posteriores [11] han obtenido mejores tasas de compresión mezclando las técnicas anteriores en un nuevo proceso. En este caso, se buscan subgrafos dentro

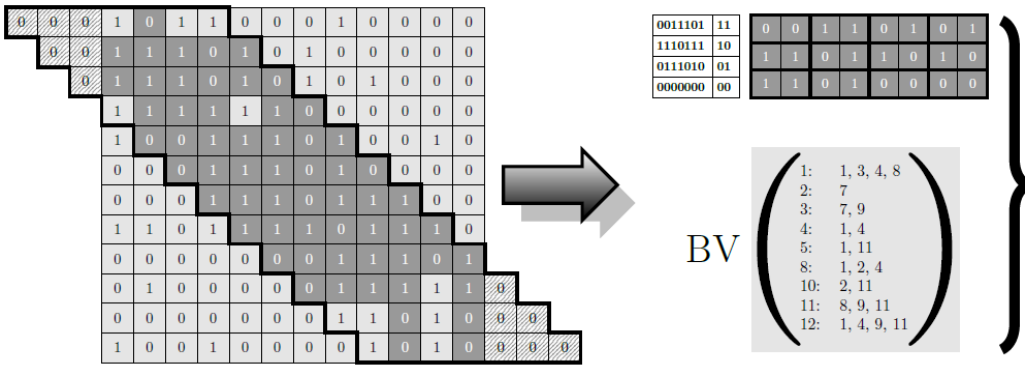


Figura 5: representación del grafo G usando el método BV+. Imagen extraída de [11].

del digrafo inicial que sean lo más densos posibles, para luego comprimirlos utilizando reordenamiento de columnas y filas de la matriz de adyacencia. La teoría detrás de esto aplica para digrafos, en donde se postula que el mayor subgrafo se encuentra alrededor de la diagonal de la matriz del grafo original. Con esto se comprime usando una ventana en base a un factor k, de largo $2k + 1$, llamado *stripe*. Utilizando el stripe se buscan patrones que tengan mayor repetición en la diagonal, como también que contengan mayor cantidad de arcos, con el fin de poder comprimir mejor esos segmentos ya que son más representativos.

El resultado, expuesto en la imagen 5, es un diccionario junto con un set de listas que contienen la diagonal capturada dentro del stripe, con $k = 3$, más una lista de corrección con los nodos que no fueron agregados al diccionario.

2.5.5. Representaciones Aproximadas

Una de las representaciones usadas para comprimir grafos es la detección de subgrafos que sean densos y tengan una gran cantidad de conexiones entre sí, formando comunidades [10][17]. Estos subgrafos reciben el nombre de cliques y bicliques.

Al detectarse estos, se busca hacer el reemplazo por un nodo virtual (a veces llamado Su-

per Nodo) que abarque y condense lo más fielmente posible las interconexiones de los nodos a los que representa. Debido a esto se obtienen resultados con cierta probabilidad de error. En el caso de [7], ellos utilizan una fórmula para expresar el porcentaje de error que posee cada nodo virtual, además de agregar a cada nodo un arco que lo conecte a si mismo debido a que su algoritmo de búsqueda de subgrafos densos obtiene mejores resultados de esta forma.

Por otro lado, en [17] se utiliza una heurística similar para comprimir un grafo, en donde además se agrega una lista de correcciones a los arcos de la representación. Para constrastar su método, el autor define lo que es una representación MDL (Minimum Description length), la que estipula que la mejor representación para un conjunto de datos dado es aquella que lleva a la mayor compresión de estos. Sintetizando, el grafo que posea el menor costo de mapear los nodos dentro de los nodos virtuales, es el mejor.

2.5.6. Compresión Manteniendo una Estructura Similar

Otra de las técnicas usadas para la compresión de grafos es la representación usando estructuras de datos similares a un grafo. En el caso de [6], éste utiliza dentro de sus métodos para tratar grafos web los k^2-tree, que dividen la matriz de adyacencia en k filas y k columnas, creando submatrices de tamaño $\frac{n^2}{k^2}$, para luego asociarlas a un nodo interno de un árbol que lo contiene. Esta transformación se hace recursivamente mientras existan submatrices que contengan 1's en sus casillas, no considerando aquellas que contienen 0's, ya que no hay información que asociar. El proceso se detiene cuando se representa completo el grafo, esto quiere decir, cuando no quedan más $1's$ que asociar. El resultado son t subgrafos representados, más 1 que los une a todos. La Figura 6 explica el método.

Este método, además de que es rápido para navegar, es eficiente para obtener información asociada a los hijos de los nodos internos, del orden de $\mathbf{O}(\log n)$. Esto debido a la estructura de la representación en forma de árbol.

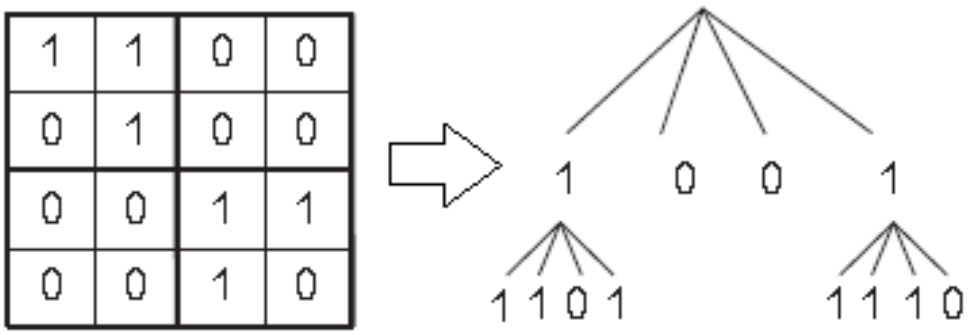


Figura 6: transformación de un grafo web G a k^2 tree. Imagen extraída de [6].

Finalmente, como se está estudiando el comportamiento de métodos de compresión que estén orientados a la compresión de índices invertidos, se presenta la definición de un índice invertido y algunos algoritmos que resuelven este problema. Cabe destacar que esto es posible debido a la compatibilidad entre las representaciones de cada tipo de estructura de datos. Dado que, como se vió en la Sección 2.5.1, es posible construir un grafo dirigido usando listas de números, escribiendo en éstas los nodos de entrada o de salida para cada nodo en particular.

2.6. Definición de Índice Invertido:

Dada una colección de documentos D con un vocabulario de diferentes palabras $\Sigma = w_1, w_2, ..., w_n$, se define el índice invertido para D como el conjunto de listas invertidas $I_{w_1}[1...a1]$, ..., $I_{w_v}[1...av]$ en donde cada lista I_{w_i} contiene el identificador del documento si y solo si la palabra w_i esté dentro de dicho documento.

Observando las listas invertidas en la imagen 7, se destaca la similitud de esta estructura con respecto a las listas de adyacencia de un grafo. Debido a que los métodos de compresión de índices invertidos abordan estas listas para comprimir la estructura, se infiere que estos

ID	Texto
1	Hoy hace calor.
2	Hace tiempo no nevaba.
3	No nevará hoy porque hace calor.

Palabra	Freq.	Lista Invertida
Calor	2	[1],[3].
Hace	3	[1],[2],[3].
Hoy	2	[1],[3].
Nevaba	1	[2].
Nevará	1	[3].
No	2	[2],[3].
Porque	1	[3].
Tiempo	1	[2].

Figura 7: ejemplo de un índice invertido I, con su vocabulario Σ *y su colección de documentos D.*

algoritmos pueden ser usados como candidatos para la compresión de un grafo, siempre y cuando éste representado en base a sus listas de adyacencia.

2.7. Métodos de Compresión para Índices Invertidos

2.7.1. VByte [16]

Un número entero tiene una representación predeterminada de 32 bits (cuatro bytes). En la mayoría de los casos, sin embargo, estos números se pueden expresar usando menor espacio. VByte usa los bytes, desde los menos significativos a los más, para almacenar el número. Utiliza el bit más significativo de cada byte como flag para decir si con ese byte fue suficiente para representar el número. Si es 1 indica que sí, 0 en caso contrario, y continúa agregando bytes hasta que el número logra ser representado con la menor cantidad de bytes posible. Con esto, en el mejor de los casos se puede contener un número en $\frac{1}{4}$ del tamaño requerido inicialmente (1 byte versus 4 bytes).

2.7.2. S9 y S17 [1] [2]

Esta técnica busca almacenar en 32 bits la mayor cantidad de números posibles. Para esto se definen inicialmente nueve posibles casos en los que se puede caer, los cuales quedan indicados en forma de header con los cuatro bits más significativos, teniendo solo 28 bits para guardar información. Esos 28 bits se dividen en una cantidad de trozos, todos del mismo tamaño. Cada trozo es usado para almacenar un número entero. Los nueve casos son indicados a continuación:

- (C_1) Un trozo de 28 bits.
- (C_2) Dos trozos de 14 bits.
- (C_3) Tres trozos de 9 bits, se desperdicia 1 bit.
- (C_4) Cuatro trozos de 7 bits.
- (C_5) Cinco trozos de 5 bits, se desperdicia 3 bits.
- (C_6) Siete trozos de 4 bits.
- (C_7) Nueve trozos de 3 bits, se desperdicia 1 bit.
- (C_8) Catorce trozos de 2 bits.
- (C_9) Veintiocho trozos de 1 bit.

De esta forma, utilizando el header se puede saber a qué caso corresponde y hacer la codificación y decodificación de los números. La Figura 8 presenta el proceso de transformación.

Una variación existente de este método agrega 8 nuevos casos de operación, aprovechando todos los bits del header más los tres bits restantes del caso cinco para implementar compatibilidad con Run-length Encoding (ver Sección 2.7.5). Esta mejora es llamada S17 [2] y sus casos son:

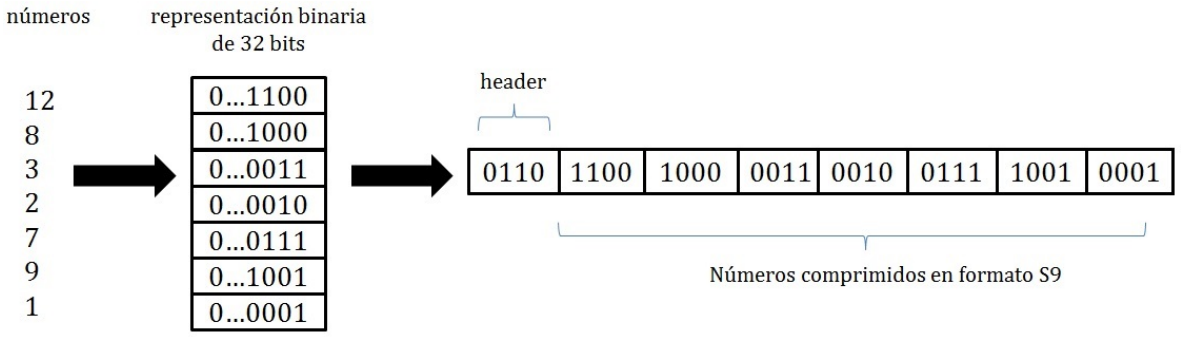


Figura 8: proceso de compresión en base a S9.

- (C_1) Un trozo de 28 bits.
- (C_2) Dos trozos de 14 bits.
- (C_3) Tres trozos de 9 bits, se desperdicia 1 bit.
- (C_4) Cuatro trozos de 7 bits.
- (C_5) Siete trozos de 4 bits.
- (C_6) Nueve trozos de 3 bits, se desperdicia 1 bit.
- (C_7) Catorce trozos de 2 bits.
- (C_8) Veintiocho trozos de 1 bit seguido de C_1.
- (C_9) Veintiocho trozos de 1 bit seguido de C_2.
- (C_{10}) Veintiocho trozos de 1 bit seguido de C_3.
- (C_{11}) Veintiocho trozos de 1 bit seguido de C_4.
- (C_{12}) Veintiocho trozos de 1 bit seguido de C_5.
- (C_{13}) Veintiocho trozos de 1 bit seguido de C_6.
- (C_{14}) Veintiocho trozos de 1 bit seguido de C_7.
- (C_{15}) Veintiocho trozos de 1 bit seguido de cinco trozos de 5 bits.

- (C_{16}) Se usa un bit extra en el header para identificar si corresponde a cinco trozos de 5 bits.

- (C_{17}) Se usa un bit extra en el header para identificar runs de más de 27 bits.

2.7.3. PForDelta [18]

Este método utiliza bloques de, generalmente, 128 números enteros, de los cuales aísla el 10 % de los valores mayores del bloque en un espacio aparte, que es llamado bloque de excepción. Del resto de números, el algoritmo escoge el de mayor valor, llamado x, y codifica al resto de números usando $b = \log_2 x$ bits. Luego se crea un header que contiene a b, más el lugar donde está almacenada la primera excepción.

Para obtener los números que están en el bloque de excepción, se guarda junto a ellos la ubicación de la siguiente excepción, formando una lista enlazada. Mientras que, para decodificar el resto de los números, se utiliza una función especializada para capturar a b, para luego, junto con otro conjunto de funciones específicas para cada trozo, poder decodificar el dato.

2.7.4. Interpolative [12]

Esta heurística está pensada en lista de números enteros en orden creciente, en donde los trata de forma holística para poder realizar la compresión. Inicialmente comprime el largo de la lista junto con el primer y último elemento. Luego toma el término que está en el medio y calcúla el dominio al que éste pertenece. Dado que es una lista creciente y se conocen los límites y el largo de éste, se puede obtener la cantidad mínima necesaria de bits para hacer la codificación.

Una vez comprimido el elemento del medio, recursivamente procede a codificar las dos

Tabla 6: codificación de un índice invertido usando Interpolative[12].

Orden original	**Orden codificado**
Largo = f	Largo = f
i_1	i_1
i_2	i_f
i_3	$i_{f/2}$
i_4	$i_{f/4}$
...	...
i_f	i_{f-1}

mitades resultantes de la lista de forma recursiva, hasta que todos los números sean codificados. Mientras más consecutivos sean los números, mejor será la tasa de compresión de este método, usando en el peor de los casos $k \times (2{,}58 + \log(n/k))$ bits [14], en donde k representa la cantidad de elementos en la lista de números, y n pertenece al límite superior del dominio de f ($f \in [0, n]$).

El resultado es un archivo de tipo binario codificado ordenado en el formato expuesto en la Tabla 6.

2.7.5. *Run-length* Encoding [15]

Run-length es un proceso de codificación en donde, en vez de escribir explícitamente los elementos que existen en una lista, se dice la cantidad y el elemento que se reitera, acotando el uso de memoria a las repeticiones de cada elemento. La Figura 9 ejemplifica la codificación.

Una de las carácterísticas que posee este método de compresión es que puede transformar cualquier ocurrencia de elementos de una lista al elemento mismo y su número de repeticiones, por lo que la compresión es efectiva cuando el elemento se repite más de 2 veces.

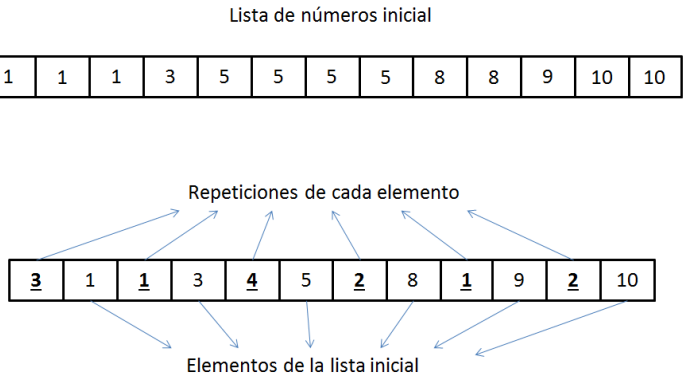


Figura 9: ejemplo codificación Run-length.

Usando esto junto con una representación basada en gaps [5], listas largas de números consecutivos pueden ser reducidos a unos pocos números a tratar, haciendo que cada compresor puede aumentar su tasa de compresión dado el orden y la forma en que se presentan los elementos. Esta técnica es usada en [2] [5].

3. Diseño del Experimento

En esta sección se describen los parámetros, datos y procesos involucrados en esta memoria.

3.1. Alcances

Para la realización de los experimentos se utilizó un único computador que posee 48 GB de RAM junto con un procesador Intel Xeon E5-2630, 15M Cache, 2.30 GHz, 7.20 GT/s Intel QP. Todo el procedimiento sobre un sistema operativo Ubuntu Linux versión 4.4.0-34-generic.

3.2. Métricas

Las métricas particulares que se usaron durante el experimento fueron las siguientes:

- *Bits por arco*: esta medida permite saber cuánto espacio se necesita para poder representar un arco entre dos nodos. En forma particular, si se tiene un nodo origen y un nodo destino definido en una arista, al usarse como base el tipo de dato int32, se espera que el bits por arco sea de 32 bits. Este valor se obtiene dividiendo el peso de las listas comprimidas en bits por la cantidad de arcos del grafo.
- Tiempo de compresión/descompresión: medido en segundos, se usa para tener registro del tiempo que toma el proceso de compresión y descompresión para cada grafo dirigido.

3.3. Datasets

Los digrafos utilizados para realizar los experimentos fueron obtenidos desde la base de datos del laboratorio para algoritmos web [3], propia del framework Webgraph[5] [4], los cuales vienen comprimidos, como también de la página del proyecto de análisis de redes de Stanford [4] [8], los que vienen en formato $nodo_{origen}$ $nodo_{destino}$:

• **Amazon-2008:** es un grafo simétrico que describe la similitud entre libros según lo informado por la tienda de Amazon.

- Arcos: 5,158,388.
- Nodos: 735,323.

• **CNR-2000:** corresponde a una porción pequeña de la red del Consejo de Desarrollo Nacional italiano al año 2000.

- Arcos: 3,216,152.
- Nodos: 325,557.

• **Eswiki-2013:** este digrafo representa un segmento de la parte española de Wikipedia de finales del 2013. Los identificadores son los títulos de las páginas y son representados por los nodos, mientras que los arcos simbolizan una relación o referencia entre dos títulos.

- Arcos: 23,041,488.
- Nodos: 972,933.

[3] `http://law.di.unimi.it/datasets.php`
[4] `http://snap.stanford.edu/data/index.html`

• **Ljournal-2008:** LiveJournal es una comunidad social virtual que comenzó en 1999. Los nodos representan a los usuarios de la red, y existe un arco entre dos nodos v_i y v_j si v_i tiene a v_j en su lista de amigos. Dado que esta relación es unilateral (vale decir, si v_i tiene en su lista de amigos a v_j, no es necesario que v_j tenga en su lista de amigos a v_i), el grafo es dirigido.

- Arcos: 79,023,142.
- Nodos: 5,363,260.

• **web-Stanford:** este grafo representa la red interna de la Universidad de Stanford. Los nodos corresponden a las páginas web, mientas que los arcos los enlaces que los direccionan.

- Arcos: 2,312,497.
- Nodos: 281,903.

Cada uno de los grafos está escrito en formato de lista de arcos. Vale decir, dos columnas separadas de números, donde una contiene los nodos origen y otra los nodos destino.

3.4. Proceso de Compresión Propuesto

El proceso de compresión busca transformar cada uno de los datasets mencionados a una representación más compacta, aplicándoles inicialmente gap encoding (ver Sección 2.5.1), luego los algoritmos de compresión de listas de adyacencia visto en la Sección 2.7, junto con una variación que los adapta para utilizar Run-length (a excepción de Interpolative). Para poder evidenciar el efecto de la renumeración de nodos en la compresión, se aplican distintas tablas de renumeración que modifican los identificadores de los nodos. Cada tabla está creada considerando criterios de comunidad, orden de entrada/salida y un orden aleatorio.

El trabajo consta de 3 etapas, las que se visualizan en la Figura 10 y se presentan como:

- Formato: cambia la forma en la que el grafo viene escrito.
- Renumeración: renumera los nodos pertenecientes al grafo (esta etapa es opcional).
- Compresión: comprime el grafo reescrito en la etapa de formato, pudiendo o no usar una tabla de renumeración.

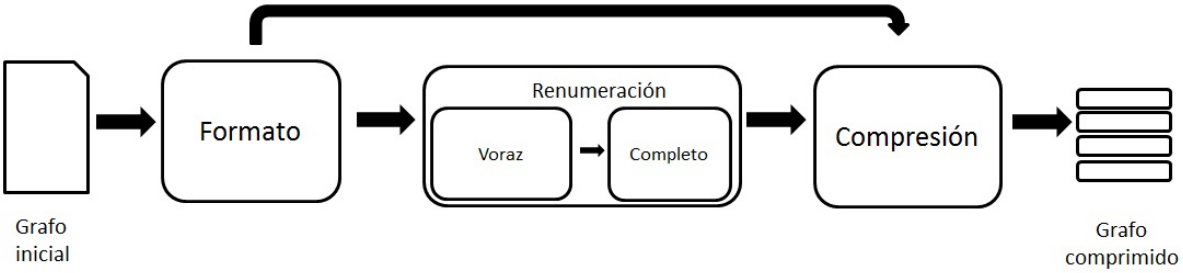


Figura 10: proceso de compresión propuesto para un grafo G.

A continuación se detalla cada etapa del proceso de compresión propuesto.

3.4.1. Formato

Dado que los digrafos vienen en un formato inicial de lista de arcos (ver Sección 2.1), se requirió modificarlos considerando las listas de adyacencia l_i, para cada nodo v_i perteneciente al grafo. Con esto, se reescribe el grafo al protocolo: $v_i,\ |l_i|,\ l_i$. La Figura 11 muestra un ejemplo.

Paralelamente, el programa de modificación de formato puede obtener información extra de los datasets, como es el caso de detección de comunidades, usando el algoritmo de Louvain [3], o las listas de adyacencia de entrada o de salida de cada nodo. Con esto se puede definir un orden previo en el que la red se va a comprimir que es utilizado por el renumerador. Este orden es necesario, ya que el proceso de renumeración es muy sensible a la forma en que se ordenan las lista de adyacencia. Esto determina la cantidad de listas intersectadas y la

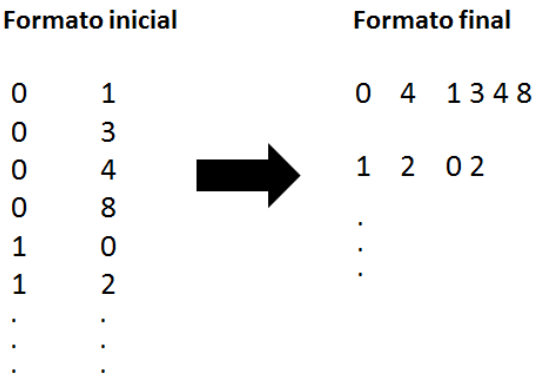


Figura 11: ejemplo de transformación de formato.

cantidad de nodos renumerados por la renumeración voraz (ver Sección 3.4.2).

De esta manera se generan cinco archivos para cada grafo por proceso de compresión, los que están escritos según:

- Las listas de adyacencia de salida, ordenadas decrecientemente según el grado de salida del nodo v_i (llamado en este trabajo formato Outdegree).
- Las listas de adyacencia de entrada, ordenadas decrecientemente según el grado de entrada del nodo v_i (llamado en este trabajo formato Indegree).
- Se divide el grafo en comunidades, las que se ordenan por su tamaño de mayor a menor. Luego se ordenan las listas de adyacencia de salida de forma decreciente, según el grado de salida que posee el nodo v_i dentro de su comunidad, para cada comunidad.
- Se divide el grafo en comunidades, las que se ordenan por su tamaño de mayor a menor. Luego se ordenan las listas de adyacencia de entrada de forma decreciente, según el grado de entrada que posee el nodo v_i dentro de su comunidad, para cada comunidad.
- Escrito de forma aleatoria con lista de adyacencia de salida.

Esta reescritura del grafo en base a la combinación de distintas características busca contrastar dos aproximaciones distintas: la primera; al agruparse los nodos con mayor cantidad de

vecinos, se espera ver que la intersección entre sus listas de adyacencia sea mayor, y permita renumerar más nodos. La segunda; al agruparse todos los nodos pertenecientes a una comunidad, sus listas de adyacencias deberían contener una mayor cantidad de nodos en común debido a las propiedades de localidad y similitud. La escritura del grafo de forma aleatoria se utiliza para tener una base de comparación de la efectividad de las otras.

3.4.2. Renumeración

Una vez obtenido el digrafo con el formato correcto y con las listas de adyacencia en un orden específico, se inicia un proceso de renumeración de nodos que busca darle al compresor una tabla que renumera los vértices para que el grafo sea más compresible. Esta etapa se divide en dos segmentos:

• **Renumeración voraz:** este algoritmo es el mismo usado en [2]. Se consideran inicialmente las intersecciones de nodos de dos listas de adyacencia l_1 y l_2, y estos son renumerados primeros entre $[\,1, |l_1 \cap l_2|]$. En general, se comienza con la intersección $l_1 \cap l_2$, si $|l_1 \cap l_2|$ es mayor a un límite inferior de intersecciones M ($|l_1 \cap l_2| > M$), se siguen agregando las listas l_i hasta que el tamaño de la intersección de las listas sea inferior a M ($|l_1 \cap l_2 \cap ... \cap l_i| < M$). Una vez renumeradas las intersecciones, se procede a renumerar cada lista de forma consecutiva, dejando una mayor cantidad de gaps de $1's$ en cada lista intersectada. Al estar las listas de adyacencia ordenadas decrecientemente por el grado del nodo, aumenta la probabilidad de intersectar más listas y con eso hacer una renumeración más efectiva para el proceso de compresión. Es en base a este mismo hecho que también se decidió realizar un orden por comunidades en el grafo ya que estos pueden tener más vecinos comunes entre sí. Este proceso se ejecuta de forma voraz, dado que, como se dijo en [2], obtener un óptimo global de renumeración para cada grafo y su conjunto de nodos resulta en un problema NP-Difícil, lo que lo hace inviable.

• **Renumeración completa:** esta etapa renumera los nodos que no fueron afectados por la renumeración voraz de forma tal que deja a todos los nodos del digrafo renumerados. Para hacer esto, reserva todas las id's generadas de la renumeración voraz, y luego rellena el dominio con los id's faltantes, completando el proceso. La desventaja de este proceso es que al ser una renumeración en donde el proceso voraz solo actúa en un segmento del grafo, existe la posibilidad de que el resultado final no sea beneficioso, por lo que la tasa de compresión se ve afectada directamente dado que los algoritmos usados buscan que los id's a comprimir sean lo más consecutivos posibles. El resultado es un archivo que muestra la renumeración de los nodos en el formato id_{viejo} id_{nuevo}.

Si bien la renumeración no es una etapa necesaria para la compresión, al efectuarse esta se facilita la aparición de runs (ver Sección 2.7.5) de números consecutivos en las listas de adyacencia más importantes, en donde se concentra la mayor cantidad de información del grafo, haciendo que los algoritmos de compresión, en especial los que tienen la variante Run-length, se enfrenten más frecuentemente a sus mejores escenarios. Esto se ve evidenciado en las mejoras de las tasas de compresión, medidos en bits por arco.

Cada uno de los cinco archivos creados en el paso anterior (ver Sección 3.4.1) es procesado acá, los que son llamados respectivamente: *Outdegree*, *Indegree*, *comunidad Outdegree*, *comunidad Indegree* y *Random* .

3.4.3. Compresión

Por último, en esta etapa se puede elegir qué algoritmo de compresión se usará dentro del buffet de métodos de compresión de índices invertidos mencionados anteriormente, además de una variante que permite utilizar codificación Run-length para cada uno, a excepción de Interpolative (ver Sección 2.7). En el caso del uso de una tabla de renumeración, el compresor crea un mapa que contiene los id's viejos y nuevos, siendo los nuevos utilizados. Cada lista

de adyacencia es transformada a codificación gap, los cuales son transformados a la forma: $v_1, v_2 - v_1, v_3 - v_2, \ldots, v_d - v_{d-1}$.

El compresor es insensible al orden que se le presenten las listas que va a tratar, pero sí es sensible a los números que estan dentro de éstas. Es por este motivo que solo se utilizaron las versiones del grafo escrito en formato Outdegree e Indegree (ver Sección 3.4.1).

4. Presentación de Resultados

En esta sección se exponen los resultados obtenidos por el experimento, mientras que el detalle de estos se encuentra en el anexo. La medida de espacio que aparece en las siguientes tablas es el bits por arco (ver Sección 3.2).

4.1. Compresión sin Tabla de Renumeración

A continuación se presentan los resultados obtenidos en el proceso de compresión de todos los datasets, tanto para su versión de formato en Outdegree como en Indegree. Para este caso no se incorporó una tabla de renumeración, sino que se usó la enumeración original asignada a cada nodo. A su vez, para la medición en bits por arco, se agregó a cada tabla los valores obtenidos por los algoritmos de WebGraph [5] (nombrado como BV) y BV+ [11] a modo de contraste.

4.1.1. *Outdegree*

Tabla 7: resultados de compresión (bits por arco) usando formato Outdegree sin el uso de tabla de renumeración.

Compresor	**Amazon-2008**	**CNR-2000**	**Eswiki-2013**	**Ljournal-2008**	**web-Stanford**
BV	10,77	3,71	10,52	11,84	4,06
BV+	10,07	3,62	–	11,78	3,90
VByte	16,92	12,35	9,90	15,76	23,13
S9	15,91	8,37	11,67	16,03	24,87
PForDelta	36,01	33,50	31,02	30,48	35,71
RL+VByte	16,34	8,20	9,72	14,90	23,13
RL+S17	16,08	8,59	11,88	16,04	24,87
RL+PForDelta	33,01	33,48	31,02	30,47	35,71
Interpolative	12,77	6,55	20,35	12,28	17,64

4.1.2. *Indegree*

Tabla 8: resultados de compresión (bits por arco) usando formato Indegree sin el uso de tabla de renumeración.

Compresor	**Amazon-2008**	**CNR-2000**	**Eswiki-2013**	**Ljournal-2008**	**web-Stanford**
BV	10,77	3,71	10,52	11,84	4,06
BV+	10,07	3,62	–	11,78	3,90
VByte	17,48	13,05	12,74	15,80	19,49
S9	15,77	7,12	12,05	15,99	19,23
PForDelta	36,51	21,07	19,69	30,11	25,98
RL+VByte	16,59	7,93	11,70	15,03	19,49
RL+S17	16,06	6,77	12,00	16,01	19,23
RL+PForDelta	36,51	20,63	19,65	30,10	25,98
Interpolative	12,54	5,13	9,31	12,24	14,14

4.2. Compresión con Tabla de Renumeración

Estos resultados presentan la incorporación de las tablas de renumeración creadas en la Sección 3.4.2, fundadas en los grafos rescritos en la Sección 3.4.1, tanto para la compresión usando formato *Outdegree* como *Indegree*.

Para interpretar la tabla, se escoge un tipo de método de compresión, y luego la tabla de renumeración basada en el tipo de formato de escritura del grafo que se usó (definido en la Sección 3.4.1). Por ejemplo, en la Tabla 9, para el algoritmo RL+S17 utilizando la tabla de renumeración creada a partir del grafo ordenado por comunidades con listas de adyacencia de salida (Comunidad Outdegree), el valor de la compresión es de 15,82 bits por arco.

4.2.1. Amazon-2008

Tabla 9: resultados de compresión (en bits por arco) con formato Outdegree de Amazon-2008 usando distintas tablas de renumeración.

Compresor	Comunidad Indegree	Comunidad Outdegree	Indegree	Outdegree	Random
BV	10,77	10,77	10,77	10,77	10,77
BV+	10,07	10,07	10,07	10,07	10,07
VByte	18.40	17,11	19,97	17,62	26.06
S9	17.23	17.72	25.87	20.81	25.72
PForDelta	36.01	36.01	35.97	36.01	36.01
RL+VByte	16.03	15.83	15.71	15.47	15.95
RL+S17	15.92	15.82	17.30	16.19	18.03
RL+PForDelta	36.01	36.01	35.97	36.01	36.01
Interpolative	13.79	14.14	20.28	16.22	20.15

Tabla 10: resultados de compresión (en bits por arco) con formato Indegree de Amazon-2008 usando distintas tablas de renumeración.

Compresor	Comunidad Indegree	Comunidad Outdegree	Indegree	Outdegree	Random
BV	10,77	10,77	10,77	10,77	10,77
BV+	10,07	10,07	10,07	10,07	10,07
VByte	17,64	19.20	26.92	21.81	25.73
S9	17.06	17.48	25.45	20.56	25.51
PForDelta	36.53	36.53	36.48	36.55	36.56
RL+VByte	16.23	16.34	16.09	16.34	17.14
RL+S17	15.82	16.07	17.37	16.87	19.01
RL+PForDelta	36.53	36.54	36.48	36.56	36.56
Interpolative	13.62	13.76	20.29	15.89	19.75

4.2.2. CNR-2000

Tabla 11: resultados de compresión (en bits por arco) con formato Outdegree de CNR-2000 usando distintas tablas de renumeración.

Compresor	Comunidad Indegree	Comunidad Outdegree	Indegree	Outdegree	Random
BV	3,71	3,71	3,71	3,71	3,71
BV+	3,62	3,62	3,62	3,62	3,62
VByte	13,85	12,89	13,00	15,63	16,30
S9	10,44	8,69	13,89	11,82	13,36
PForDelta	33,59	33,50	33,71	33,54	34,06
RL+VByte	8,54	7,96	8,88	7,88	7,98
RL+S17	9,25	8,05	10,36	8,80	9,77
RL+PForDelta	33,57	33,48	33,67	33,47	34,05
Interpolative	8,40	7,70	11,50	11,11	12,41

Tabla 12: resultados de compresión (en bits por arco) con formato Indegree de CNR-2000 usando distintas tablas de renumeración.

Compresor	Comunidad Indegree	Comunidad Outdegree	Indegree	Outdegree	Random
BV	3,71	3,71	3,71	3,71	3,71
BV+	3,62	3,62	3,62	3,62	3,62
VByte	13,28	14,64	13,18	13,33	20,61
S9	7,49	9,79	8,58	16,40	18,87
PForDelta	21,83	26,36	21,73	31,86	34,21
RL+VByte	7,96	8,80	7,88	10,51	11,43
RL+S17	6,96	8,39	7,13	11,76	13,55
RL+PForDelta	21,42	26,19	21,14	31,83	34,23
Interpolative	5,59	7,77	6,56	14,20	16,94

4.2.3. Eswiki-2013

Tabla 13: resultados de compresión (en bits por arco) con formato Outdegree de Eswiki-2013 usando distintas tablas de renumeración.

Compresor	Comunidad Indegree	Comunidad Outdegree	Indegree	Outdegree	Random
BV	10,52	10,52	10,52	10,52	10,52
BV+	–	–	–	–	–
VByte	30.25	28.59	29.82	28.76	28.17
S9	30.90	30.02	30.15	30.03	29.74
PForDelta	33.37	33.37	33.37	33.37	33.37
RL+VByte	14.16	14.69	13.26	14.39	14.74
RL+S17	18.07	18.94	16.93	18.59	19.10
RL+PForDelta	33.37	33.37	33.37	33.37	33.37
Interpolative	23.66	22.76	23.26	22.87	22.64

Tabla 14: resultados de compresión (en bits por arco) con formato Indegree de Eswiki-2013 usando distintas tablas de renumeración.

Compresor	Comunidad Indegree	Comunidad Outdegree	Indegree	Outdegree	Random
BV	10,52	10,52	10,52	10,52	10,52
BV+	–	–	–	–	–
VByte	26,75	23,89	25,23	24,10	23,95
S9	26,97	24,60	25,68	25,71	24,56
PForDelta	31,78	30,73	31,03	31,04	31,21
RL+VByte	11,89	11,68	11,62	11,75	11,81
RL+S17	15,60	15,06	15,41	15,30	15,41
RL+PForDelta	31,77	30.70	31.00	31.02	31,18
Interpolative	20,68	18,60	19.59	19,52	18,59

4.2.4. Ljournal-2008

Tabla 15: resultados de compresión (en bits por arco) con formato Outdegree de Ljournal-2008 usando distintas tablas de renumeración.

Compresor	Comunidad Indegree	Comunidad Outdegree	Indegree	Outdegree	Random
BV	11,84	11,84	11,84	11,84	11,84
BV+	11,78	11,78	11,78	11,78	11,78
VByte	24,61	24,68	28,30	27,63	25,12
S9	23,60	23,65	27,39	26,59	24,69
PForDelta	32,74	32,75	33,66	33,40	33,52
RL+VByte	14,39	14,27	14,12	13,73	14,74
RL+S17	15,95	15,86	16,43	16,01	16,90
RL+PForDelta	32,73	32,75	33,67	33,40	33,50
Interpolative	18,60	18,71	21,75	21,19	19,64

Tabla 16: resultados de compresión (en bits por arco) con formato Indegree de Ljournal-2008 usando distintas tablas de renumeración.

Compresor	Comunidad Indegree	Comunidad Outdegree	Indegree	Outdegree	Random
BV	11,84	11,84	11,84	11,84	11,84
BV+	11,78	11,78	11,78	11,78	11,78
VByte	24,59	24.41	27,82	27,67	24,77
S9	23.58	23.29	26.81	26.67	24.45
PForDelta	32.76	32.65	33.22	33.55	33.69
RL+VByte	14.44	14.50	13.96	14.07	15.15
RL+S17	15.94	15.90	16.16	16.20	17.18
RL+PForDelta	32.75	32.65	33.22	33.55	33.66
Interpolative	18.60	18.36	21.24	21.19	19.33

4.2.5. web-Stanford

Tabla 17: resultados de compresión (en bits por arco) con formato Outdegree de web-Stanford usando distintas tablas de renumeración.

Compresor	Comunidad Indegree	Comunidad Outdegree	Indegree	Outdegree	Random
BV	4,06	4,06	4,06	4,06	4,06
BV+	3,90	3,90	3,90	3,90	3,90
VByte	23,15	23,59	22,95	23,15	30,34
S9	22,54	22,22	22,53	23,64	29,66
PForDelta	35,76	35,66	35,76	35,66	35,90
RL+VByte	15,91	13,35	16,45	14,34	18,80
RL+S17	16,33	14,59	16,83	16,22	20,43
RL+PForDelta	35,76	35,66	35,76	35,66	35,90
Interpolative	17,30	17,13	17,06	17,57	21,67

Tabla 18: resultados de compresión (en bits por arco) con formato Indegree de web-Stanford usando distintas tablas de renumeración.

Compresor	Comunidad Indegree	Comunidad Outdegree	Indegree	Outdegree	Random
BV	4,06	4,06	4,06	4,06	4,06
BV+	3,90	3,90	3,90	3,90	3,90
VByte	15,04	24,47	15,29	25,60	26,43
S9	13,98	24,99	13,65	26,70	28,49
PForDelta	26,81	35,36	25,83	35,37	35,73
RL+VByte	12,48	12,97	12,61	14,53	17,44
RL+S17	11,73	15,54	11,65	16,27	19,23
RL+PForDelta	26,59	35,36	25,59	35,37	35,73
Interpolative	10,90	19,25	10,50	20,21	20,24

4.3. Compresión del 80 % de los Vértices

En esta sección se presentan los resultados de haber comprimido el 80 % de los arcos para cada dataset, con el propósito de estudiar el efecto que tiene comprimir listas de adyacencia de pequeño tamaño, tomándolas de mayor a menor grado. Por otro lado, la Tabla 19 contrasta la cantidad de listas necesarias para lograr el 80 % entre los distintos tipos de escritura usados para cada grafo.

La Tabla 19 se interpreta escogiendo primero un grafo para luego leer la información asociada. Por ejemplo, para el grafo web-Stanford, se entiende que posee 281,903 nodos (y por lo tanto 281,903 listas de adyacencia). Escrito en formato Outdegree (ver Sección 3.4.1), el 80 % de los arcos se obtiene hasta la lista 122,947. Escrito en formato Indegree (ver Sección 3.4.1), el 80 % de los arcos se obtiene hasta la lista 53,324, y el largo de la última lista tomada en formato Outdegree es 5, mientras que en formato Indegree es 5.

Por otro lado, las tablas de las Secciones 4.3.1, 4.3.2, 4.3.3, 4.3.4 y 4.3.5 se leen de la misma forma que en la Sección 4.2.

Tabla 19: cantidad de listas donde se obtiene el 80 % de los arcos de cada dataset, más el largo de la última lista tomada, para cada tipo de formato.

Dataset	# Nodos	Cantidad Listas Formato Outdegree	Cantidad Listas Formato Indegree	Largo Última Lista Outdegre/Indegree
Amazon-2008	735,323	412,672	330,520	10/5
CNR-2000	325,557	88,643	35,498	11/9
Eswiki-2013	972,933	475,968	82,631	16/31
Ljournal-2008	5,363,260	1,233,012	1,203,340	17/15
web-Stanford	281,903	122,947	53,324	5/5

4.3.1. Amazon-2008

Tabla 20: resultados compresión del 80 *% de las aristas de Amazon-2008 para las distintas tablas de renumeración usadas, en formato Outdegree.*

Compresor	**Comunidad Indegree**	**Comunidad Outdegree**	**Indegree**	**Outdegree**	**Random**	**Sin Renumeración**
BV	10,77	10,77	10,77	10,77	10,77	10,77
BV+	10,07	10,07	10,07	10,07	10,07	10,07
VByte	17.26	15,77	25,61	20,52	25,05	15,54
S9	16,20	16,92	25,23	19,81	25,29	14,62
PForDelta	35,20	35,20	35,20	35,20	35,20	35,20
RL+VByte	14,47	14,40	14,46	14,10	14,73	14,86
RL+S17	14,59	14,75	16,65	15,21	17,48	14,77
RL+PForDelta	35,20	35,20	35,20	35,20	35,20	35,20
Interpolative	13,16	13,60	19,96	15,53	19,96	11,88

Tabla 21: resultados compresión del 80 *% de las aristas de Amazon-2008 para las distintas tablas de renumeración usadas, en formato Indegree.*

Compresor	**Comunidad Indegree**	**Comunidad Outdegree**	**Indegree**	**Outdegree**	**Random**	**Sin Renumeración**
BV	10,77	10,77	10,77	10,77	10,77	10,77
BV+	10,07	10,07	10,07	10,07	10,07	10,07
VByte	14,67	16,80	16,84	19,32	23,70	14,51
S9	15,28	15,94	24,46	19,04	24,82	13,45
PForDelta	34,52	34,53	34,52	34,55	34,56	34,50
RL+VByte	13,01	13,21	13,03	13,26	14,24	13,40
RL+S17	13,71	14,11	15,90	15,03	17,69	13,82
RL+PForDelta	34,52	34,53	34,52	34,55	34,56	34,50
Interpolative	12,43	12,67	19,94	14,78	19,48	10,78

4.3.2. CNR-2000

Tabla 22: resultados compresión del 80 *% de las aristas de CNR-2000 para las distintas tablas de renumeración usadas, en formato Outdegree.*

Compresor	**Comunidad Indegree**	**Comunidad Outdegree**	**Indegree**	**Outdegree**	**Random**	**Sin Renumeración**
BV	3,71	3,71	3,71	3,71	3,71	3,71
BV+	3,62	3,62	3,62	3,62	3,62	3,71
VByte	11,71	10,25	10,92	10,63	13,84	10,27
S9	8,43	6,56	11,57	9,47	10,64	6,35
PForDelta	32,01	31,90	32,16	31,95	32,60	31,90
RL+VByte	5,73	5,13	6,16	5,01	5,26	5,41
RL+S17	7,21	5,88	8,44	6,63	7,45	6,48
RL+PForDelta	31,98	31,87	32,11	31,86	32,60	31,87
Interpolative	6,79	5,94	9,92	9,52	10,63	4,86

Tabla 23: resultados compresión del 80 *% de las aristas de CNR-2000 para las distintas tablas de renumeración usadas, en formato Indegree.*

Compresor	**Comunidad Indegree**	**Comunidad Outdegree**	**Indegree**	**Outdegree**	**Random**	**Sin Renumeración**
BV	3,71	3,71	3,71	3,71	3,71	3,71
BV+	3,62	3,62	3,62	3,62	3,62	3,62
VByte	8,78	8,79	9,68	15,00	16,67	8,77
S9	3,49	6,30	4,27	13,59	16,30	3,09
PForDelta	15,68	21,34	15,55	28,22	31,14	14,73
RL+VByte	2,48	3,55	2,39	5,66	6,76	2,43
RL+S17	2,88	4,76	3,02	8,81	10,86	2,63
RL+PForDelta	15,16	21,13	14,82	28,18	31,14	14,18
Interpolative	2,31	5,00	3,22	12,51	15,72	1,76

4.3.3. Eswiki-2013

Tabla 24: resultados compresión del 80 *% de las aristas de Eswiki-2013 para las distintas tablas de renumeración usadas, en formato Outdegree.*

Compresor	**Comunidad Indegree**	**Comunidad Outdegree**	**Indegree**	**Outdegree**	**Random**	**Sin Renumeración**
BV	10,52	10,52	10,52	10,52	10,52	10,52
BV+	–	–	–	–	–	–
VByte	26.75	24,56	26,95	24,87	28,29	13,91
S9	28,91	27,97	28,48	29,00	28,97	15,56
PForDelta	32,83	32,83	32,84	32,84	32,84	30,45
RL+VByte	12,31	10,98	12,37	11,04	10,93	13,58
RL+S17	16,93	15,17	16,88	15,29	15,19	15,57
RL+PForDelta	32,83	32,83	32,84	32,84	30,45	30,45
Interpolative	21,42	21,90	21,79	22,59	22,65	12,04

Tabla 25: resultados compresión del 80 *% de las aristas de Eswiki-2013 para las distintas tablas de renumeración usadas, en formato Indegree.*

Compresor	**Comunidad Indegree**	**Comunidad Outdegree**	**Indegree**	**Outdegree**	**Random**	**Sin Renumeración**
BV	10,52	10,52	10,52	10,52	10,52	10,52
BV+	–	–	–	–	–	–
VByte	21,70	21,42	23,67	21,71	21,61	10,70
S9	22,67	21,68	24,67	24,75	23,33	9,81
PForDelta	29,32	28,73	29,51	29,51	29,71	15,33
RL+VByte	9,75	9,80	9,85	10,03	10,05	9,44
RL+S17	13,89	13,44	14,38	14,25	14,24	9,75
RL+PForDelta	29,29	28,70	29,47	29,49	29,69	15,29
Interpolative	17,12	16,36	18,93	18,86	17,71	7,66

4.3.4. Ljournal-2008

Tabla 26: resultados compresión del 80 *% de las aristas de Ljournal-2008 para las distintas tablas de renumeración usadas, en formato Outdegree.*

Compresor	Comunidad Indegree	Comunidad Outdegree	Indegree	Outdegree	Random	Sin Renumeración
BV	11,84	11,84	11,84	11,84	11,84	11,84
BV+	11,78	11,78	11,78	11,78	11,78	11,78
VByte	23,11	22,03	25,63	26,96	24,13	16,17
S9	24,36	24,31	26,54	25,73	23,58	16,33
PForDelta	33,03	33,05	33,85	33,55	33,69	31,86
RL+VByte	14,49	14,33	14,49	14,17	14,95	15,21
RL+S17	15,97	15,83	16,40	16,03	16,83	16,34
RL+PForDelta	33,06	33,07	33,87	33,58	33,67	21,88
Interpolative	19,31	19,34	21,25	20,66	18,81	12,50

Tabla 27: resultados compresión del 80 *% de las aristas de Ljournal-2008 para las distintas tablas de renumeración usadas, en formato Indegree.*

Compresor	Comunidad Indegree	Comunidad Outdegree	Indegree	Outdegree	Random	Sin Renumeración
BV	11,84	11,84	11,84	11,84	11,84	11,84
BV+	11,78	11,78	11,78	11,78	11,78	11,78
VByte	24,89	24,92	26,50	25,21	23,47	13,39
S9	22,59	22,28	26,49	26,43	24,04	14,24
PForDelta	31,03	30,90	31,61	32,02	32,17	27,72
RL+VByte	12,12	12,18	11,63	11,76	12,89	12,47
RL+S17	14,70	14,63	15,12	15,17	16,07	14,24
RL+PForDelta	31,02	30,90	31,61	32,03	32,16	27,70
Interpolative	17,91	17,65	21,08	21,08	19,08	10,84

4.3.5. web-Stanford

Tabla 28: resultados compresión del 80 *% de las aristas de web-Stanford para las distintas tablas de renumeración usadas, en formato Outdegree.*

Compresor	Comunidad Indegree	Comunidad Outdegree	Indegree	Outdegree	Random	Sin Renumeración
BV	4,06	4,06	4,06	4,06	4,06	4,06
BV+	3,90	3,90	3,90	3,90	3,90	3,90
VByte	26,51	24,84	30,32	31,17	28,46	23,30
S9	27,87	26,32	30,45	30,89	29,25	23,30
PForDelta	34,13	34,13	34,13	34,13	34,13	33,89
RL+VByte	17,52	18,38	15,63	15,10	16,52	20,62
RL+S17	20,68	21,15	18,70	18,03	19,70	23,30
RL+PForDelta	34,13	34,13	34,13	34,13	34,13	33,89
Interpolative	20,32	19,38	22,46	23,01	21,61	16,88

Tabla 29: resultados compresión del 80 *% de las aristas de web-Stanford para las distintas tablas de renumeración usadas, en formato Indegree.*

Compresor	Comunidad Indegree	Comunidad Outdegree	Indegree	Outdegree	Random	Sin Renumeración
BV	4,06	4,06	4,06	4,06	4,06	4,06
BV+	3,90	3,90	3,90	3,90	3,90	3,90
VByte	22,96	25,17	27,90	25,16	26,11	15,17
S9	24,74	22,48	29,22	29,98	27,58	16,14
PForDelta	33,05	33,01	33,05	33,05	33,05	20,88
RL+VByte	14,01	14,27	13,27	13,15	13,49	15,17
RL+S17	17,58	17,07	17,51	17,40	17,49	16,14
RL+PForDelta	33,05	33,01	33,05	33,05	33,05	20,88
Interpolative	17,67	16,36	21,10	21,76	19,84	12,84

4.4. Tiempo de Compresión

A continuación se presentan tres tablas: las dos primeras indican la velocidad que demora cada método en comprimir 1 arco, medido en microsegundos, de forma promedio entre cada uno de los grafos usados, para cada tabla de renumeración usada; mientras que la última expone la cantidad de arcos comprimidos por segundo por cada algoritmo.

La Tabla 30 y 31 se leen escogiendo primero un algoritmo de compresión y una tabla de renumeración, para luego en la entender la intersección como el tiempo en [μseg] que tarda en comprimir 1 arco. Por ejemplo, escogiendo el método S9 y la tabla de renumeración Outdegree, el tiempo que tarda en comprimir 1 arco es de 0,898 [μseg].

Mientras que la Tabla 32 se lee escogiendo un método de compresión, luego la modificación aplicada (si se usó una tabla de renumeración o no, como también si se utilizó la variante RL), y finalmente el formato de escritura del grafo tratado (ver Sección 3.4.1). Por ejemplo, para el algoritmo VByte, utilizando la variante RL y sin tabla de renumeración, con formato Indegree, se obtiene que se comprimen 2,468,928 [arcos/segundo].

Tabla 30: resumen tiempo promedio de compresión en [μseg] usando formato Outdegree.

Compresor	**Comunidad Indegree**	**Comunidad Outdegree**	**Indegree**	**Outdegree**	**Random**	**Sin Renumeración**
VByte	0,915	0,905	0,907	0,903	0,902	0,383
S9	0,884	0,848	0.856	0.898	0.864	0.327
PForDelta	0.832	0.830	0.831	0.839	0.825	0.318
RL+VByte	0.917	0.915	0.915	0.912	0.913	0.387
RL+S17	0.857	0.857	0.851	0.855	0.862	0.337
RL+PForDelta	0.839	0.840	0.836	0.833	0.838	0.329
Interpolative	0.843	0.845	0.834	0.839	0.838	0.322

Tabla 31: resumen tiempo promedio de compresión en [µseg] usando formato Indegree.

Compresor	Comunidad Indegree	Comunidad Outdegree	Indegree	Outdegree	Random	Sin Renumeración
VByte	0.907	0.906	0.915	0.912	0.912	0.392
S9	0.901	0.872	0.857	0.872	0.876	0.345
PForDelta	0.876	0.861	0.850	0.852	0.854	0.331
RL+VByte	0.955	0.924	0.918	0.926	0.918	0.405
RL+S17	0.891	0.862	0.864	0.865	0.870	0.348
RL+PForDelta	0.856	0.861	0.852	0.864	0.867	0.341
Interpolative	0.649	0.653	0.651	0.645	0.651	0.272

Tabla 32: cantidad de arcos comprimidos por segundo, para cada método.

Método de compresión	Modificación	Orden Outdegree	Orden Indegree
VByte	sin RL, sin renumeración	2,613,108	2,551,274
	con RL, sin renumeración	2,587,289	2,468,928
	sin RL, con renumeración	1,103,524	1,098,482
	con RL, con renumeración	1,093,546	1,077,486
S9 y S17	sin RL, sin renumeración	3,055,275	2,902,078
	con RL, sin renumeración	2,967,500	2,872,681
	sin RL, con renumeración	1,149,561	1,142,060
	con RL, con renumeración	1,167,599	1,149,214
PForDelta	sin RL, sin renumeración	3,144,654	3,023,817
	con RL, sin renumeración	3,038,983	2,931,301
	sin RL, con renumeración	1,202,543	1,164,710
	con RL, con renumeración	1,194,415	1,162,735
Interpolative	sin renumeración	3,108,609	3.673.144
	con renumeración	1,190,591	1.538.491

4.5. Tiempo de Descompresión

A continuación se presenta en la Tabla 33 la velocidad promedio de descompresión calculada para cada método de compresión. Ésta interpreta escogiendo un algoritmo de compresión, y luego leyendo los valores asociados. Por ejemplo, para Interpolative, se entiende que es capaz de comprimir 1 arco en 0,3408[μseg] y 2,934,626 [arcos/segundo].

Tabla 33: resumen tiempo promedio de descompresión para cada método de compresión.

Compresor	**Tiempo de Descompresión de 1 Arco en** [μseg]	**Arcos/Segundo**
VByte	0,2325	4,299,622
S9	0,3339	2,994,202
PForDelta	0,2774	3,400,031
RL+VByte	0,2952	3,387,480
RL+S17	0,3304	3,026,409
RL+PForDelta	0,3128	3,196,467
Interpolative	0,3408	2,934,626

4.6. Listas de Renumeración Voraz

La Tabla 34 expone los resultados por el algoritmo voraz (ver Seccion 3.4.2) en el siguiente orden: indica la cantidad de listas intersectadas y luego los primeros nodos renumerados, luego presenta la cantidad de nodos renumerados involucrados en la intersección.

Para leerla, primero se escoge el grafo a analizar. Luego, para cada tabla de renumeración, se toman las listas de adyacencia consideradas por la renumeración voraz denotadas por "Listas intersectadas". En "Documentos renumerados", el primer número indica la cantidad de nodos existente en la intersección de las listas, mientras que el segundo el total de nodos renumerados que se vieron afectados por la renumeración de la intersección. Por ejemplo, para Amazon-2008 y considerando la tabla de renumeración por Comunidad Indegree, se desprende que se intersectaron 5 listas de adyacencia, en donde la intersección contiene 32 nodos que fueron inicialmente renumerados, y luego 71,107 nodos que a causa de la renumeración de la intersección se les devió reasignar un id.

Tabla 34: resumen proceso de renumeración voraz.

Dataset	Subproceso	Comunidad Indegree	Comunidad Outdegree	Indegree	Outdegree	Random
Amazon-2008	Listas intersectadas	5	2	4	2	6
	Documentos renumerados	32 / 71,107	0 / 339,365	32 /452,067	0 / 564,036	1 / 244,207
CNR-2000	Listas intersectadas	13	613	11	3	2
	Documentos renumerados	18,236 / 44,273	619 / 54,062	18,236 /218,515	699 /198,963	0 / 117,131
Eswiki-2013	Listas intersectadas	12	4	7	4	2
	Documentos renumerados	1,232 /559,810	10 /291,550	10,483 /952,701	10 / 341,680	0 / 259,780
Ljournal-2008	Listas intersectadas	10	3	5	4	2
	Documentos renumerados	396 / 1,111,281	12 /1,025,539	3,482 / 1,551,558	587 /1,405,649	0 / 527.913
web-Stanford	Listas intersectadas	11	14	4	2	2
	Documentos renumerados	19,321 / 281,677	6 /261,603	19,321 / 281,677	0 / 261,603	0 / 261,603

5. Análisis de Resultados y Procesos

De los resultados obtenidos se desprende que es posible encontrar una renumeración y reordenamiento adecuados que permiten obtener mejores desempeños que los algoritmos específicos de compresión de grafos usados en esta memoria. En una segunda aproximación, tanto RL+VByte como Interpolative, y en algunos casos RL+S17, obtuvieron los mejores resultados en todo el experimento. Incluso, en la iteración en donde se utilizó el grafo en fortamo Indegree, sin el uso de tabla de renumeración, Interpolative fue mejor que los resultados de referencia mostrados en [5] y [11] (ver Tabla 8, Interpolative usado sobre Eswiki-2013).

Analizando la configuración usada para obtener dichos resultados, la escritura de las listas de adyacencia por formato Indegree logra, generalmente, mejor rendimiento con respecto al formato Outdegree (ver Tablas 8 y 7), debido al largo de listas de vecinos asociadas. Esto se reafirma con los resultados obtenidos por [2], en donde los índices invertidos usados contenían una menor cantidad de listas, pero de mayor largo con respecto a las listas de adyacencia de los grafos. Contínuamente, observando las tablas de la Sección 4.3, se refleja una mejora notoria en cuanto a bits por arco para todos los métodos, explicado por la reducción de listas de pequeño tamaño en el proceso de compresión, ayudando a que los gaps sean más homogéneos.

Por otro lado, como la distribución de grados cambia entre el formato Indegree y Outdegree, al haber listas más largas, implica que otras listas de adyacencia tienen que ser más pequeñas (ver Tabla 19). El formato Indegree posee una mayor cantidad de listas vacías, lo que implica que se deberán comprimir menos listas de adyacencia, mejorando la tasa de compresión.

De todo esto se infiere que el mejor contexto en el que se desempeñan estos algoritmos de compresión de índices invertidos es trabajando con pocas listas de gran tamaño, que posean gaps lo más homogéneos posible y evitando las listas de largo pequeño y con gaps muy

grandes entre sí.

Con respecto a la velocidad de compresión de cada método, observando las Tablas 32, 30 y 31, se infiere que el uso de una tabla de renumeración, cualquiera que sea, amplifica el tiempo del proceso de compresión en un promedio de $2,52$ veces. Interpolative fué el único que obtuvo diferencias en el tiempo de compresión, cambiando de formato Outdegree a Indegree, en aproximadamente un $24\,\%$, por lo que se deduce éste es sensible al largo de las listas que comprime.

Por otro lado, al ver la Tabla 33, la velocidad de descompresión es pareja para cada método sin importar si se aplicó una tabla de renumeración en la compresión. Evidenciando que VByte, RL+VByte y PForDelta son los métodos más rápidos en esta operación.

Dada la variedad de resultados obtenidos, tanto aplicando tabla de renumeración como el formato en que el grafo estaba escrito por el formateador, se determinó que uno de los factores implícitos que más afectó en la compresión fué la topología de los grafos. Esto apunta a que, dada la forma que cada grafo posee y sus comunidades, como también en la distribución de grados de entrada/salida por nodo, distintas propiedades de cada grafo se pueden aprovechar para obtener una compresión admisible. La compresión del grafo web-Stanford (ver Sección 4.2.5) es un ejemplo de esto, teniendo resultados muy diferentes en compresión usando renumeración basada en Outdegree e Indegree, tanto global como usando comunidades.

5.1. Métodos de Compresión

Con respecto a los métodos de compresión de índices invertidos se desprende lo siguiente:

5.1.1. VByte y RL+VByte

RL+VByte fue uno de los algoritmos que obtuvo mejor tasa de compresión, siendo incluso el mejor dentro de todo el buffet en ciertas iteraciones usando tabla de renumeración (ver Tablas 28, 27, 26, 25, 24, 22, 17, 15, 16, 13, 14, 11). Esto debido a que, por un lado la forma en que VByte puede comprimir números, sumado a la forma en que los codifica Run-length, hace que tenga un buen desempeño trabajando sobre listas de adyacencia que son homogéneas y de gran tamaño, haciendo que los runs queden señalados con una marca de 1 bit al principio del byte que usa para comprimir, dejando un espacio de $2^7 - 1$ para representar el largo del run. Por otro lado, para tratar las listas más pequeñas este se adapta haciendo que los runs se representen en menor cantidad de bits.

En contraste a su tasa de compresión, VByte y RL+VByte son los que más demoran en descomprimir dentro del set de métodos de compresión (ver Sección 4.4). Esto es independiente de la aplicación de Run-length, por lo que se entiende que es propio del proceso de codificación y lectura de los bits usados como indicadores, como también en la construcción de los id's completos los que implican un mayor tiempo de operación.

Con respecto a VByte, éste se ve fuertemente penalizado por las listas de menor tamaño con id's muy separados entre sí, ya que su forma de comprimir siempre utiliza 1 byte en el mejor de los casos para comprimir un número. Esto se enfatiza más con listas de adyacencia con solo un vecino con un gran id, en donde se requiere más de un byte para poder representarlo.

5.1.2. S9 y RL+S17

Al tener variados casos de compresión de números, estos métodos se ven beneficiados fuertemente si se aplica una renumeración que respete la estructura de comunidades del grafo. Esto debido a que al renumerarse correctamente favorece la aparición de runs de 1's, lo

que permite representar una mayor cantidad de números de forma eficiente dentro de los bloques de memoria de S17 y S9. También la diversidad de casos que este presenta para comprimir, identificando y aprovechando 16 distintos escenarios en caso de S17, y los 9 de S9 hace que sea más versatil que los otros algoritmos.

Por otro lado, ambos métodos reservan 4 bytes para representar números, por lo que, al tratar con listas de adyacencia de pocos vecinos provoca que todo el espacio no sea necesario. Más aún, si los nodos en la lista de vecinos tienen id's demasiados discretos, esto puede generar que los gaps generados sean mayor a 2^{14}. Esto logra que, para cada gap de este tipo, se requieran 4 bytes para comprimir para ambos métodos, lo que reafirma la necesidad de una renumeración que deje a las listas de vecinos lo más contínua posible.

S17 se ve perjudicado debido a que solo es capaz de detectar runs que tengan largo 10 o más debido a sus casos de compresión, por lo que hay listas que si bien tienen runs que se pueden comprimir, al no detectarse el algoritmo los ignora. Sin embargo, al existir distintos casos de compresión el algoritmo obtiene resultados que si bien no caen en el óptimo, son buenos de igual manera en forma general.

5.1.3. PForDelta y RL+PForDelta

Este método fue el que peor desempeño obtuvo en cuanto a tasa de compresión en todos los casos aplicados, incluso no siendo alterado por la incorporación de Run length. Una de las causas fue la forma con que este método comprime los nodos. Al clasificar el 10 % de los números más grandes y al existir muchas listas que pueden tener gaps grandes, esto conlleva a que queden muchos id's codificados de magnitud considerable en la lista. Por otro lado, el hecho de que exista en todos los grafos una distribución de grados de los nodos que indica que la mayoría tiene pocos vecinos hace que el método se enfrente a su peor caso constantemente. Además al hacer una reserva de solo 128 números enteros como espacio predeterminado para

su proceso interno de compresión, hace que quede mucha memoria sin utilizar eficientemente.

A pesar de esto, PForDelta tiene una de las mejores velocidades de compresión y descompresión en promedio, siendo capaz de comprimir 3,000,000 [arcos/segundo] sin el uso de una tabla de renumeración y 1,180,000 [arcos/segundo] usándola (ver Tabla 32).Esto es debido a la forma en que la lista queda codificada; la descompresión es practicamente continua en el 90 % de los casos, mientras que el 10 % restante se enfoca a la obtención de los id's de mayor magnitud numérica haciendo referencia al bloque de excepciones y volviendo a la lista inicial. Dado que las listas eran de pequeño tamaño, eran escasas las búsquedas de números en el bloque de excepciones.

5.1.4. Interpolative

Este algoritmo fue el único que pudo sobrepasar a las herramientas específicas de compresión de grafos, con un resultado de 5,13 bits por arco logrado en el dataset CNR-2000 usando el grafo ordenado por formato Indegree (ver Tabla 8). En contraste, este algoritmo no obtuvo mejoras al aplicársele una renumeración de vértices. Esto debido principalmente a que Interpolative usa memoria según el id a comprimir y el largo del dominio al cual pertenece (ver Sección 2.7.4). Mientras más grande el dominio, mayor es la memoria reservada, y mezclado con una renumeración que puede dejar los id's muy distintos entre sí, cae en su peor caso de operación.

Por otro lado, Interpolative muestra diferencias de rendimiento al medir su velocidad de descompresión (ver Tabla 33), siendo más rápido al usar un formato Indegree que Outdegree. Esto se explica principalmente por la distribución de grados de los nodos, como también por la diferencia de tamaños de la lista de adyacencia de estos. En formato Indegree existen, para cada grafo, listas de adyacencia más largas, pero además mayor cantidad de listas de menor tamaño.

5.2. Reordenamiento

Uno de los hechos que más se destaca es, que en la mayoría de los experimentos hechos, la renumeración basada en comunidades tuvo una tasa de compresión significativamente mejor con respecto a las que no, llegan incluso a 9,28 bits de diferencia (ver Tabla 10). Esto es consecuencia del reordenamiento hecho en base a algún parámetro o técnica que agrupe nodos con mayor probabilidad de tener vecinos comunes, ya que obtiene mejores resultados sobre otros con órdenes universales (como el caso de solo ordenarlos por grado de entrada o salida), o aleatorios. Más aún, el aplicar alguna métrica que reordene los nodos y destaque los vértices más importantes tendrá generalmente mejores resultados que hacerlo de forma aleatoria. Esto reafirma la importancia de generar métodos de compresión de grafos considerando las propiedades de localidad y similitud.

Continuamente, el reordenamiento de nodos en base a comunidades ayudó a que aumentara la cantidad de intersecciones en el renumerador (ver Tabla 34), y por consiguiente se incrementó la cantidad de vértices renumerados, haciendo crecer la cantidad de runs al considerar más listas de adyacencia. Esto, nuevamente, se explica por la similitud y localidad de los nodos con respecto a sus vecinos definida en las Secciones 2.2 y 2.3.

Por otra parte, examinando las Tablas 7 y 8, se ven diferencias en los mejores resultados de compresión de cada algoritmo con respecto al orden de escritura del grafo, para cada grafo. Por lo que se demuestra que los métodos de compresión tienen casos favorables diferentes, pero dependientes de la topología del grafo, ya que el largo de las listas y la distribución de éstas varía entre escritura por formato Outdegree e Indegree (ver Tabla 19).

5.3. Renumeración

Examinando los resultados expuestos en la Tabla 34, la cantidad de listas de adyacencia alcanzadas durante la intersección depende directamente de la forma en que se ordenen. En la

mayoría de los casos, al renumerar la red que fue reordenada por comunidades, se obtuvo una mayor intersección de listas. Esto se debió al principio de localidad y similitud (ver Sección 2.2) ya que los nodos se agruparon en clusters con vecinos comunes. Al ocurrir esto, se logró que las listas de adyacencia de los vértices más importantes de la comunidad quedaran unidos al ser renumerados, generando mejores runs para los métodos con Run-length, mientras que a los otros algoritmos les ayuda a comprimir gaps más pequeños.

Esto se contrasta con la renumeración considerando orden por grado de salida y de entrada sin comunidades, ya que, salvo Interpolative, los algoritmos sin la adaptación Run-length obtuvieron peores resultados. Esto se debe a que la intersección entre las listas de adyacencia era menor ya que se renumeraban listas que no tenían muchos nodos en común, producto de la topología del grafo, ya que al ordenarse los nodos bajo un único parámetro, es altamente probable que los nodos más importantes de cada comunidad queden escritos contínuamente, pero debido a lo señalado en el apartado de comunidades (ver Sección 2.3), estos no tienen muchos vecinos comunes entre sí.

Por otra parte, resaltando los resultados entre tasas de compresión hechas con y sin renumeración de vértices, hay variaciones sobre cúal metodología posee el óptimo de compresión. Se infiere que el proceso de renumeración es un problema de búsqueda de solución óptima en donde el universo de ésta son todas las posibles renumeraciones de una lista de adyacencia arbitraria que cumpla con ser lo más consecutiva posible sin romper la continuidad de las demás listas de vecinos, para cada nodo del grafo. Por lo que una de las contraproducencias que tiene el renumerar de forma voraz es el que al obtenerse un óptimo local en la renumeración, esta nueva configuración puede perjudicar la tasa de compresión del grafo completo, dejando números demasiado grandes de comprimir y obligando a los algoritmos de compresión enfrentar sus peores escenarios de desempeño. Sumado a esto, la distribución del largo de las listas de adyacencia, considerando tanto listas de nodos de salida como de entrada, indica que son escasos los nodos que poseen muchos vecinos, frente a los que tienen pocos, lo que

aumenta la probabilidad de la aparición de gaps de mayor magnitud.

Al incluirle una tabla de renumeración al compresor se crea un mapa el cual renumera los nodos en el proceso, éste incrementa el tiempo de compresión en promedio 2,52 veces. Sin embargo, si la renumeración es adecuada, puede mejorar tanto la tasa de compresión,como la velocidad propia del método. Esto se debe a que, al lograrse listas de adyacencia más consecutivas y con números más pequeños a comprimir, producto de la aplicación de gaps, la codificación se acelera ya que se cae en escenarios óptimos para los algoritmos.

Finalmente analizando los nodos que no eran seleccionados por la renumeración voraz, además de los excluídos por la ventana de búsqueda, existe también un subconjunto de vértices que no poseen lista de adyacencia contraria al orden que se le entregó al renumerador. Vale decir, si la lista de nodos está ordenada por formato Outdegree, los vértices excluidos serán preferentemente aquellos que tengan lista de vecinos de entrada vacíos. Esto se produce por el orden en las listas que entran al renumerador, ya que estos vértices quedan retraidos al final del archivo, no siendo afectados y filtrándose de la renumeración voraz.

6. Conclusiones

El desarrollo del estudio sobre el desempeño de variados métodos de compresión aplicados a diversas estructuras de datos conllevó un gran desafío dado los factores a tener en cuenta para poder manejar una tasa de compresión que fuese aceptable. Los algoritmos de compresión de índices invertidos son compatibles concrétamente con la tarea de comprimir grafos dirigidos y no dirigidos, gracias a su representación por listas de adyacencia. Pueden llegar a obtener resultados competentes con respecto a la compresión de digrafos en relación a los métodos específicos si y solo si se les da una configuración apropiada para que estos exploten sus mejores características. Dado esto, conseguir una mejora de un bit en la tasa de compresión promedio implica haber reducido el espacio que el grafo ocupa en $|S|$ bits, siendo $|S|$ la cantidad de arcos del grafo, lo que es un gran logro.

La variedad de parámetros considerados en este trabajo, como también la elección de distintos datasets pertenecientes a distintos tópicos (grafos sociales y web), hicieron que durante el estudio resaltaran las cualidades de cada método, logrando que cada una pudiese destacar sobre las otrás en distintas categorías. En términos bastante generales se pueden clasificar los algortimos en diferentes segmentos importantes:

- Tasa de compresión: RL+VByte e Interpolative.
- Velocidad de compresión: Interpolative y PForDelta.
- Velocidad de descompresión: VByte, RL+VByte y PForDelta.
- Adaptabilidad: RL+S17.

De igual manera, se concluye que los escenarios ideales para cada uno de los métodos probados son:

- VByte y RL+VByte: hacer que los gaps obtenidos sean de magnitud máxima 127 ($2^{7\ bits} - 1$) para poder usar el bloque correctamente.
- S9 y RL+S17: los id's de los nodos deben ser lo más consecutivos posible, así utilizando la codificación Run-length múltiples nodos pueden ser comprimidos en un bloque de 28 bits, o bien los gaps deben ser los más pequeños posibles para caer en los casos de mayor tasa de compresión.
- PForDelta y RL+PForDelta: listas de vecinos que sean lo suficientemente largas para llenar la reserva de 128 enteros que el algoritmo reserva inicialmente.
- Interpolative: el largo del dominio de los id's de los nodos inicialmente debe ser lo más similar al largo de la lista de vecinos, además de agregar la excepción de considerar listas de adyacencia con largo menor a 2, ya que entonces solo se comprimen los extremos.

Sobre el proceso de renumeración de los nodos usado en este caso, es deseable que las listas a renumerar contengan alguna propiedad de comunidad o clustering apropiada, como también que el contenido de estas sean nodos que faciliten la intersección y la continuidad de las listas. Esto apuntaría a mayor número de intersecciones y facilitaría la realización de una renumeración de nodos más homogénea. Por otro lado, otra recomendación posible a este algoritmo de renumeración es hacer que ésta se aplique a cada comunidad, reservando id's en base al tamaño del cluster. Luego se le da prioridad a que estos sean usados dentro de las listas de vecinos, dejando al final de la renumeración los arcos que se comunican con otras comunidades.

La inclusión de una tabla de renumeración altera en un factor de $2,52$ la velocidad de compresión. Sin embargo, esto es parte de un trade-off con la tasa de compresión, ya que se obtienen mejores resultados aplicando la renumeración correcta.

Por otro lado, se lograron los objetivos específicos, en lo que destacan:

- La realización de un estudio comparativo entre algoritmos de compresión de índices invertidos y grafos bajo distintos escenarios.
- Un análisis sobre los efectos que la renumeración puede tener entre los métodos de compresión probados y cómo esto puede ser usado para obtener mejores tasas de compresión.
- Entender de qué forma se puede mejorar el rendimiento individual de los métodos probados aprovechando la topología propia de los grafos aplicada al proceso de renumeración de nodos.
- Obtener una comparación entre velocidades de compresión y descompresión de cada método, entendiendo que estos poseen hardcoding para agilizar sus procesos.
- Hacer una comparación entre herramientas específicas de compresión de grafos e índices invertidos, destacando las tasas de compresión logradas sobre los mismos digrafos.

En base a esto, se puede hacer una elección sobre qué métodos usar preferentemente para comprimir distintos tipos de grafos, como también los casos en donde estos deben ser evitados.

Gracias a este trabajo se obtuvo una visión sobre las competencias que debe tener un algoritmo de compresión, como también en los factores que este se debe de aprovechar para obtener un resultado que pueda reducir la cantidad de bits usados por la nueva representación lo más posible, dado que cada bit por arco cuenta y hace la diferencia, como también la capacidad de adaptarse a los diversos casos de compresión aplicables para el método, manteniendo una velocidad de compresión/descompresión lo más eficiente posible. La correcta combinación de estos factores es lo que hace a un compresor viable.

En todo este proceso la formación de la carrera fue una ayuda primordial, dado que el poner a prueba diversos algoritmos de compresión, tener una conexión remota con el servidor usado, el diseccionamiento hecho a cada algoritmo para ver sus fortalezas y debilidades, el uso de herramientas de debugging y conocimiento de lenguajes de programación y estructuras de datos apropiadas. También la adaptabilidad a los diversos problemas que fueron surgiendo, solucionando y aprendiendo formas de enfrentar situaciones adversas, tanto en la forma técnica como humano, es fruto de una formación profesional exitosa. El autoaprendizaje y la resilencia son las características principales adquiridas en la universidad, en donde en el área de la informática actual se espera se posea debido a la velocidad con que ésta evoluciona.

Referencias Bibliográficas

[1] Anh V. N., Moffat A. *Inverted index compression using word-aligned binary codes. Information Retrieval, pages 151–166.* 2005.

[2] Arroyuelo Diego, González Sénen, Oyarzún Mauricio, Sepúlveda Victor. *Document Identifier Reassignment and Run-Length-Compressed Inverted Indexes for Improved Search Performance. SIGIR, pages 173-182.* 2013.

[3] Blondel Vincent D, Guillaume Jean-Loup, Lambiotte Renaud, Lefebvre Etienne. *Fast unfolding of communities in large networks. CoRR.* 2008.

[4] Boldi Paolo, Rosa Marco, Santini Massimo, Vigna Sebastiano. *Layered Label Propagation: A MultiResolution Coordinate-Free Ordering for Compressing Social Networks. World Wide Web (WWW), pages 587-596* 2011.

[5] Boldi Paolo, Vigna Sebastiano. *The WebGraph Framework I: Compression Techniques. In World Wide Web (WWW), pages 595-602. ACM.* 2004.

[6] Brisaboa Nieves R., Ladra Susana, Navarro Gonzalo. *K^2trees for Compact Web Graph Representation. SPIRE, pages 18-30.* 2009.

[7] Claude Francisco, Ladra Susana. *Practical Representations for Web and Social Graphs. CIKM 2011: 1185-1190.* 2011.

[8] Dasgupta A, Lang K, Leskovec J, Mahoney M. *Community Structure in Large Networks: Natural Cluster Sizes and the Absence of Large Well-Defined Clusters. Internet Mathematics 6(1) 29-123.* 2009.

[9] Fortunato Santo. *Community detection in graphs. Computing Research Repository.* 2009.

[10] Hernández Cecilia, Navarro Gonzalo. *Compressed representations for web and social graphs. Knowledge and Information Systems, Volume 40(2): 279-313.* 2014.

[11] Liakos Panagiotis, Papakonstantinopoulou Katia, Sioutis Michael. *Pushing the Envelope in Graph Compression. CIKM 2014: 1549-1558.* 2014.

[12] Moffat Alistair, Stuiver Lang. *Binary Interpolative Coding for Effective Index Compression. Information Retrieval, Volume 3(1): 25-47.* 2000.

[13] Newman, M. E. J. *Detecting community structure in networks. Eur. Phys. J. B 38.* 2004.

[14] Puglisi Simon J. *Data Compression Techniques, Lecture 4: Integer Codes 2.* [Online][Cited: Octubre 1,2016] `http://www.cs.helsinki.fi/u/puglisi/dct2015/slides4.pdf` 2015.

[15] Solomon W. Golomb. *Run-length Encodings. Information Theory 12(3), pages 399-401. IEEE Trans.* 1966.

[16] Williams H. and Zobel J. *Compressing integers for fast file access. The Computer Journal, 42(3):193–201.* 1999.

[17] Zhou Fang. *Graph Compression. Department of Computer Science and Helsinki Institute for Information Technology HIIT. pages 1-12.* 2014.

[18] Zukowski M, Héman S, Nes N, Boncz P. *Super-scalar RAM-CPU cache compression. In Proc. of 22nd Int. Conf. on Data Engineering (ICDE), page 59. IEEE Computer Society.* 2006.

6.1. Tiempo Compresión Amazon-2008

• Tiempo de compresión en segundos con grafo en formato Outdegree:

Tabla 35: tiempo de compresión en [seg] usando formato Outdegree para Amazon-2008, aplicando todos las formas de renumeración mencionadas.

Compresor	Comunidad Indegree	Comunidad Outdegree	Indegree	Outdegree	Random	Sin Renumeración
VByte	4.797	4.778	4.767	4.804	4.858	2.259
S9	4.636	4.658	4.628	4.743	4.671	2.053
PForDelta	4.424	4.406	4.534	4.575	4.513	2.126
RL+VByte	4.882	4.942	4.929	4.850	4.943	2.387
RL+S17	4.544	4.566	4.554	4.573	4.615	2.074
RL+PForDelta	4.661	4.685	4.490	4.633	4.590	2.204
Interpolative	4.484	4.501	4.415	4.425	4.522	2.111

• Tiempo de compresión en segundos con grafo en orden Indegree:

Tabla 36: tiempo de compresión en [seg] usando formato Indegree para Amazon-2008, aplicando todos las formas de renumeración mencionadas.

Compresor	Comunidad Indegree	Comunidad Outdegree	Indegree	Outdegree	Random	Sin Renumeración
VByte	5.080	5.124	5.165	5.110	5.097	2.369
S9	5.065	5.052	4.980	5.004	5.069	2.254
PForDelta	4.800	5.017	4.859	4.809	4.872	2.072
RL+VByte	5.133	5.215	5.225	5.301	5.280	2.453
RL+S17	5.016	4.913	5.102	4.960	5.054	2.238
RL+PForDelta	4.880	4.944	4.887	4.910	4.928	2.129
Interpolative	4.852	4.885	4.887	4.867	4.961	2.083

- **Gráfico resumen tasa de compresión formato Outdegree:**

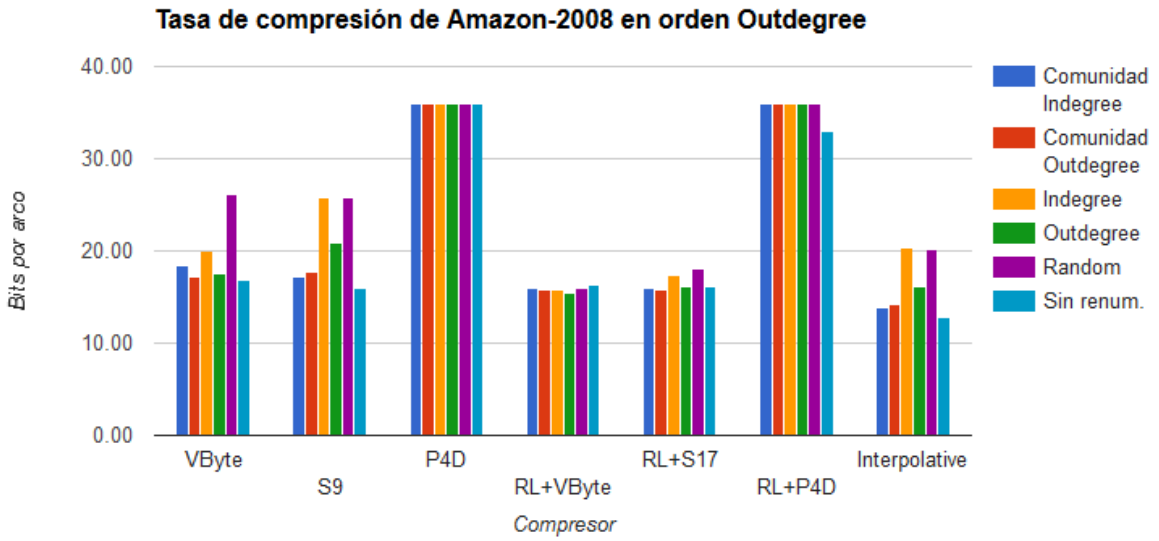


Figura 12: gráfico resumen compresión de Amazon-2008 en formato Outdegree.

- **Gráfico resumen tasa de compresión orden Indegree:**

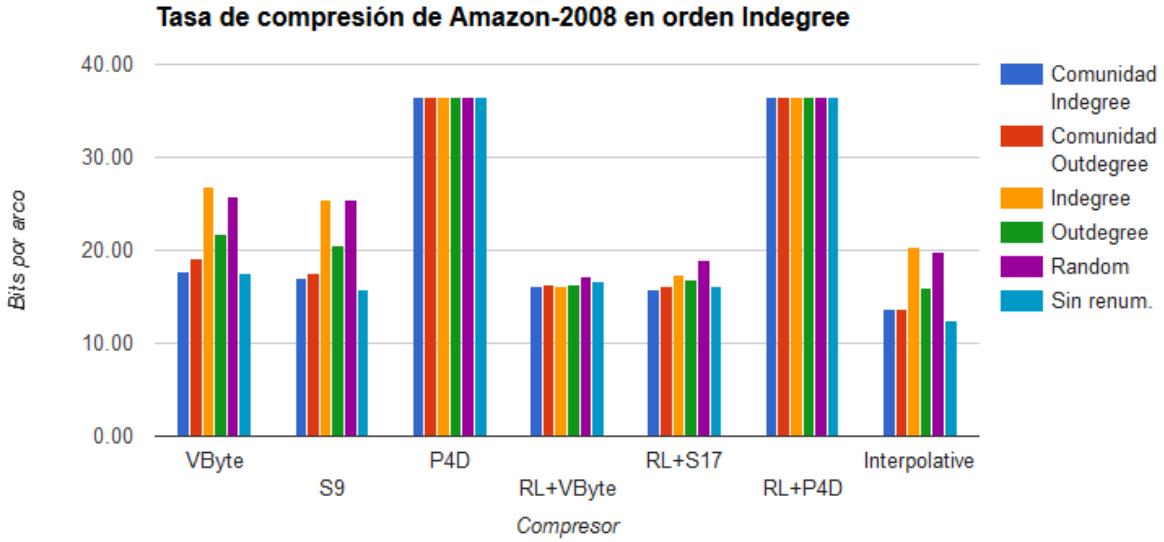


Figura 13: gráfico resumen compresión de Amazon-2008 en orden Indegree.

6.2. Tiempo Compresión CNR-2000

• Tiempo de compresión en segundos con grafo en formato Outdegree:

Tabla 37: tiempo de compresión en [seg] usando formato Outdegree para CNR-2000, aplicando todos las formas de renumeración mencionadas.

Compresor	Comunidad Indegree	Comunidad Outdegree	Indegree	Outdegree	Random	Sin Renumeración
VByte	2.183	2.183	2.233	2.103	2.115	1.156
S9	2.087	2.039	2.107	2.055	2.081	1.001
PForDelta	1.977	2.035	2.005	1.998	2.003	1.010
RL+VByte	2.105	2.099	2.159	2.146	2.129	1.144
RL+S17	2.064	2.021	2.063	2.005	2.030	1.025
RL+PForDelta	2.030	2.029	2.008	1.986	1.992	1.053
Interpolative	2.001	1.993	1.981	2.022	2.000	0.971

• Tiempo de compresión en segundos con grafo en orden Indegree:

Tabla 38: tiempo de compresión en [seg] usando formato Indegree para CNR-2000, aplicando todos las formas de renumeración mencionadas.

Compresor	Comunidad Indegree	Comunidad Outdegree	Indegree	Outdegree	Random	Sin Renumeración
VByte	2.245	2.223	2.310	2.313	2.289	1.202
S9	2.189	2.170	2.071	2.179	2.211	1.056
PForDelta	2.122	2.183	2.112	2.227	2.226	1.053
RL+VByte	2.293	2.267	2.268	2.306	2.316	1.193
RL+S17	2.179	2.133	2.136	2.142	2.199	1.101
RL+PForDelta	2.161	2.177	2.192	2.253	2.248	1.121
Interpolative	2.143	2.140	2.112	2.101	2.134	1.101

- **Gráfico resumen tasa de compresión formato Outdegree:**

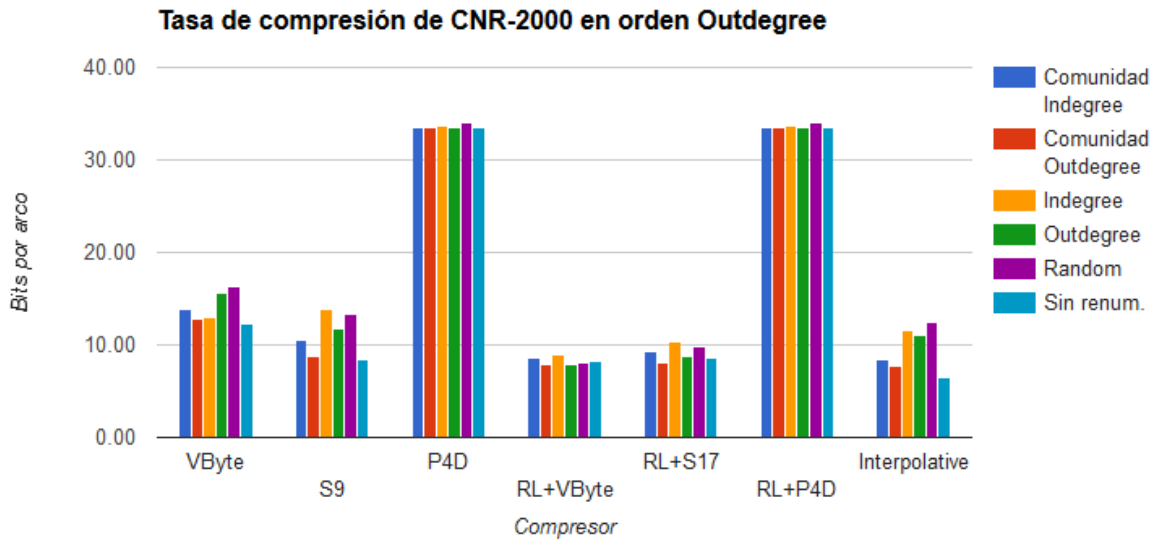


Figura 14: gráfico resumen compresión de CNR-2000 en formato Outdegree.

- **Gráfico resumen tasa de compresión orden Indegree:**

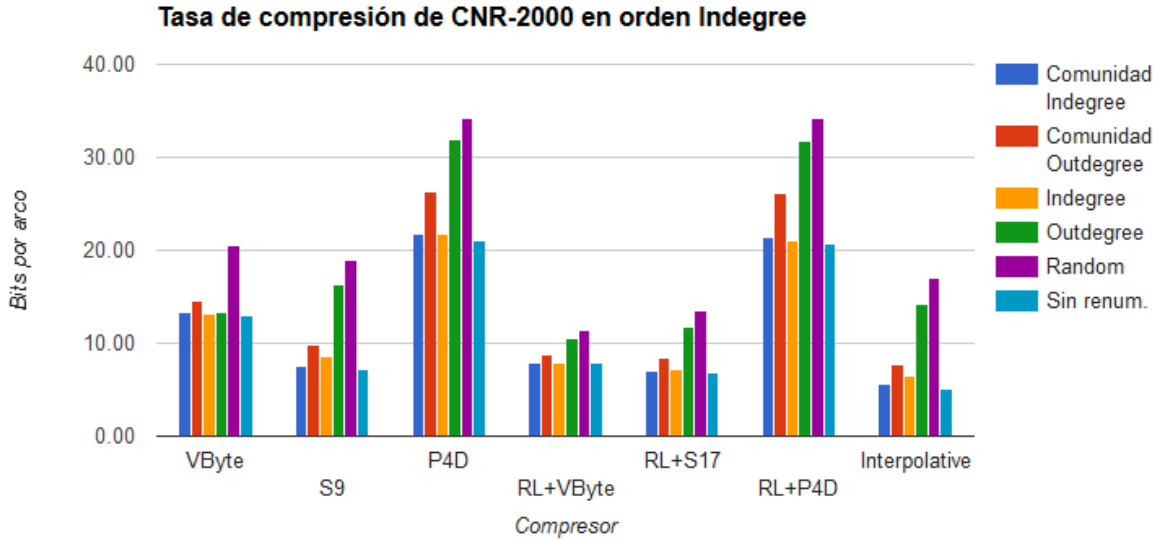


Figura 15: gráfico resumen compresión de CNR-2000 en orden Indegree.

6.3. Tiempo Compresión Eswiki-2013

• Tiempo de compresión en segundos con grafo en formato Outdegree:

Tabla 39: tiempo de compresión en [seg] usando escritura Outdegree para Eswiki-2013, aplicando todos las formas de renumeración mencionadas.

Compresor	Comunidad Indegree	Comunidad Outdegree	Indegree	Outdegree	Random	Sin Renumeración
VByte	19.945	19.881	19.677	19.795	19.916	6.042
S9	18.449	18.526	18.621	18.533	18.621	5.137
PForDelta	17.710	17.628	17.693	17.568	17.593	4.791
RL+VByte	20.135	20.060	19.967	19.949	20.102	6.145
RL+S17	18.460	18.422	18.415	18.521	18.624	5.387
RL+PForDelta	17.604	17.562	17.650	17.546	17.755	4.705
Interpolative	18.067	18.267	18.126	18.182	18.132	5.125

• Tiempo de compresión en segundos con grafo en orden Indegree:

Tabla 40: tiempo de compresión en [seg] usando formato Indegree para Eswiki-2013, aplicando todos las formas de renumeración mencionadas.

Compresor	Comunidad Indegree	Comunidad Outdegree	Indegree	Outdegree	Random	Sin Renumeración
VByte	18.213	18.037	18.051	18.165	18.122	7.434
S9	16.826	17.016	16.831	17.071	16.988	5.841
PForDelta	16.623	16.704	16.653	16.900	16.664	5.743
RL+VByte	18.414	18.479	18.230	18.457	18.344	7.500
RL+S17	16.929	16.980	16.947	17.039	16.911	5.948
RL+PForDelta	16.725	16.982	16.713	16.916	16.963	5.927
Interpolative	16.565	16.570	16.786	16.590	16.660	5.635

- **Gráfico resumen tasa de compresión formato Outdegree:**

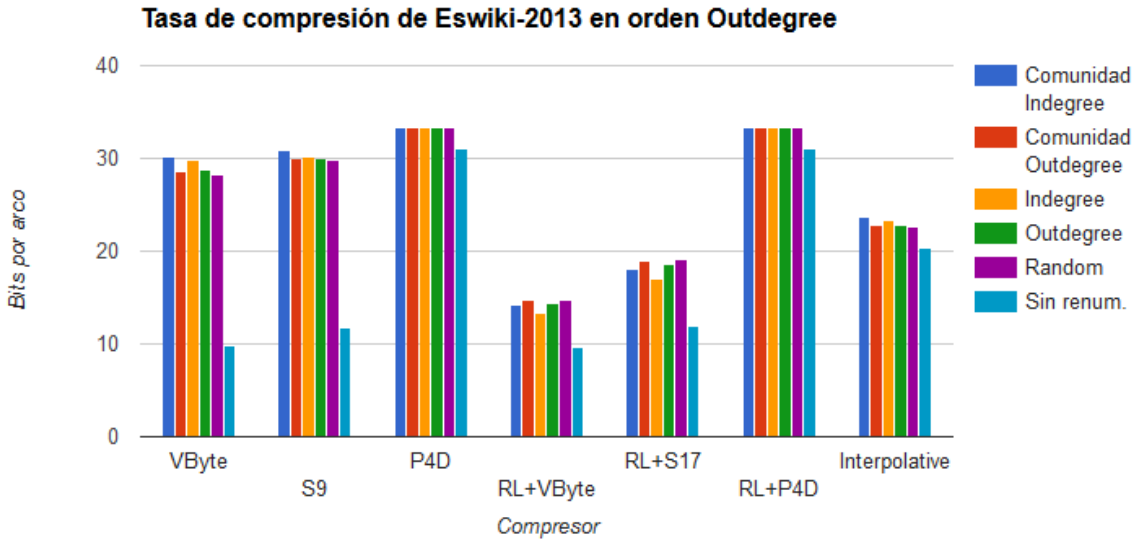


Figura 16: gráfico resumen compresión de Eswiki-2013 en formato Outdegree.

- **Gráfico resumen tasa de compresión orden Indegree:**

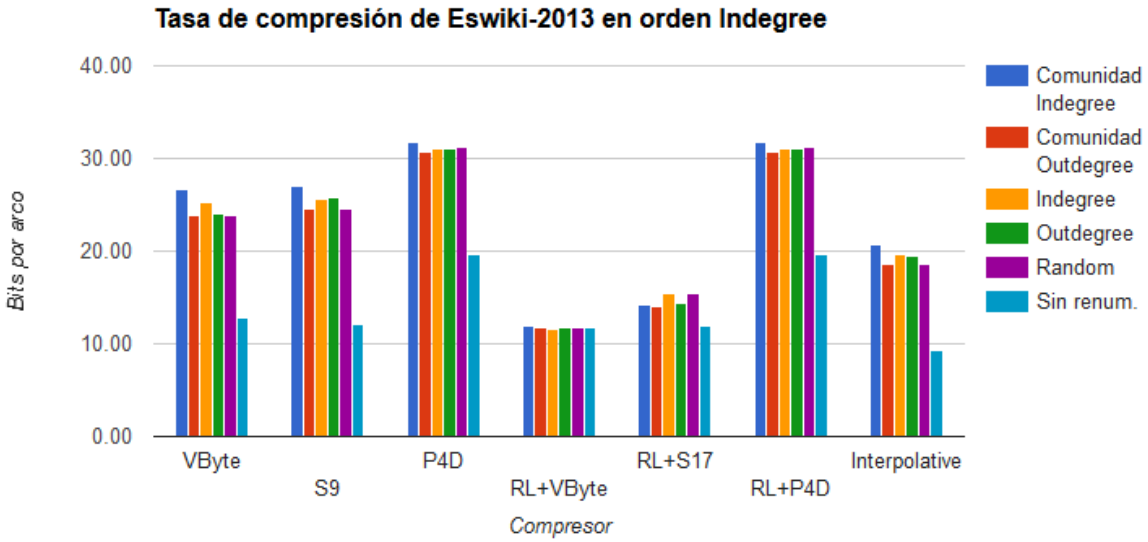


Figura 17: gráfico resumen compresión de Eswiki-2013 en orden Indegree.

6.4. Tiempo Compresión Ljournal-2008

• Tiempo de compresión en segundos con grafo en formato Outdegree:

Tabla 41: tiempo de compresión en [seg] usando formato Outdegree para Ljournal-2008, aplicando todos las formas de renumeración mencionadas.

Compresor	Comunidad Indegree	Comunidad Outdegree	Indegree	Outdegree	Random	Sin Renumeración
VByte	83.494	82.205	83.006	83.498	82.781	28.719
S9	78.851	78.652	78.138	92.601	78.884	24.216
PForDelta	75.436	75.873	76.450	79.894	76.039	22.317
RL+VByte	84.423	83.033	83.278	83.728	83.575	28.946
RL+S17	78.214	78.850	77.935	78.793	78.949	25.235
RL+PForDelta	76.625	76.002	76.398	76.593	76.248	22.828
Interpolative	78.179	77.574	77.377	78.505	77.167	23.285

• Tiempo de compresión en segundos con grafo en orden Indegree:

Tabla 42: tiempo de compresión en [seg] usando formato Indegree para Ljournal, aplicando todos las formas de renumeración mencionadas.

Compresor	Comunidad Indegree	Comunidad Outdegree	Indegree	Outdegree	Random	Sin Renumeración
VByte	83.681	83.264	85.426	83.307	84.213	28.488
S9	91.858	79.537	79.156	81.887	79.887	24.306
PForDelta	88.849	77.139	77.028	77.347	76.181	22.712
RL+VByte	98.213	85.080	83.500	83.866	83.571	29.487
RL+S17	89.021	79.137	78.135	79.467	79.563	24.625
RL+PForDelta	76.339	77.292	77.215	77.327	78.446	23.132
Interpolative	77.511	77.699	78.028	77.769	78.124	23.392

- **Gráfico resumen tasa de compresión formato Outdegree:**

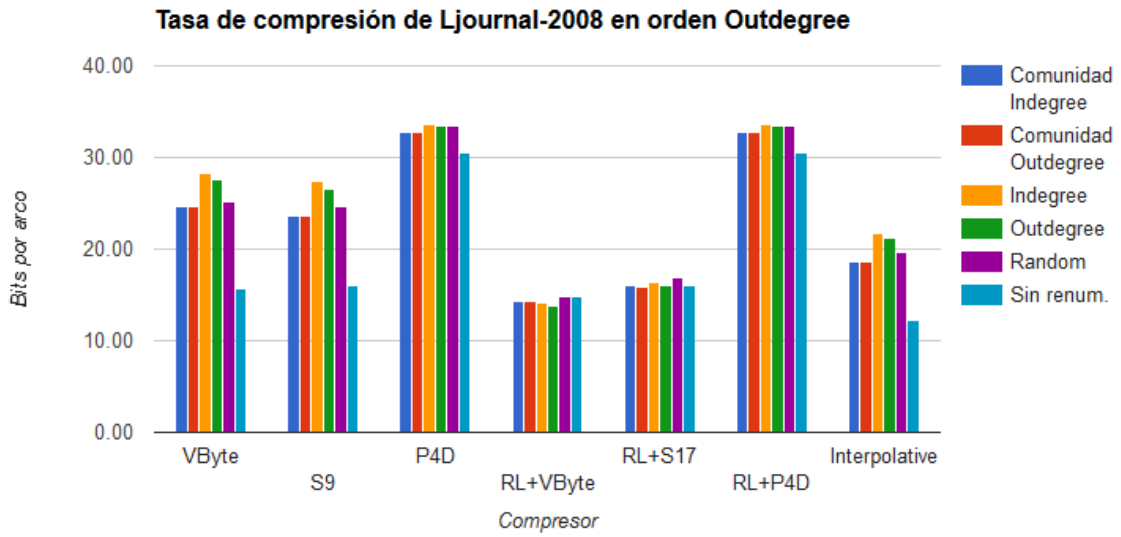


Figura 18: gráfico resumen compresión de Ljournal-2008 en formato Outdegree.

- **Gráfico resumen tasa de compresión orden Indegree:**

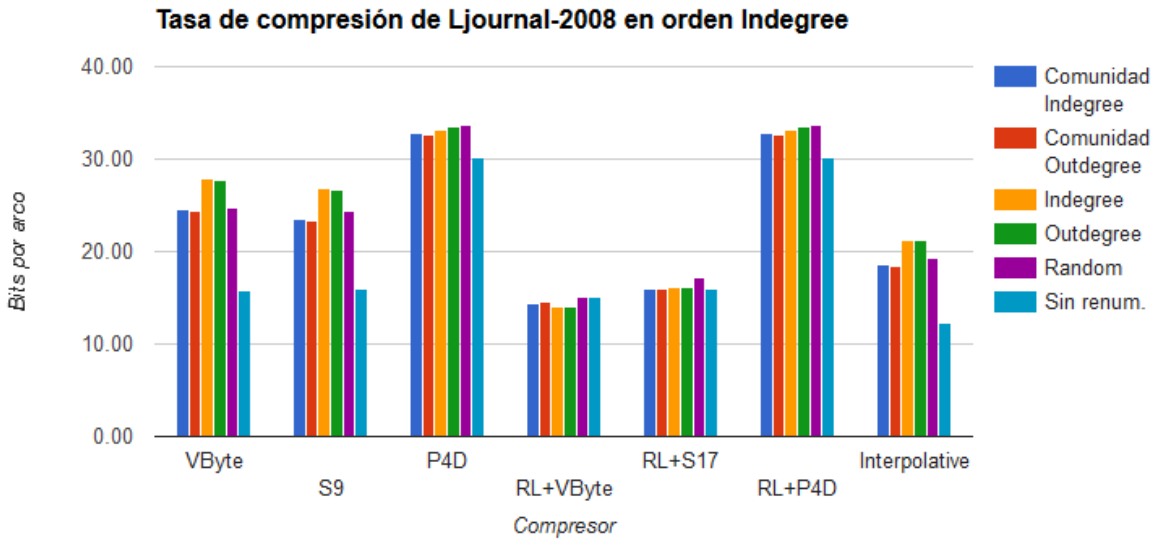


Figura 19: gráfico resumen compresión de Ljournal-2008 en orden Indegree.

6.5. Tiempo Compresión web-Stanford

• Tiempo de compresión en segundos con grafo en formato Outdegree:

Tabla 43: tiempo de compresión en [seg] usando formato Outdegree para web-Stanford, aplicando todos las formas de renumeración mencionadas.

Compresor	Comunidad Indegree	Comunidad Outdegree	Indegree	Outdegree	Random	Sin Renumeración
VByte	2.409	2.381	2.345	2.340	2.309	1.134
S9	2.485	2.085	2.151	2.211	2.218	0.920
PForDelta	2.235	2.166	2.120	2.117	2.088	0.859
RL+VByte	2.414	2.408	2.378	2.376	2.346	1.113
RL+S17	2.252	2.252	2.186	2.230	2.258	0.950
RL+PForDelta	2.147	2.166	2.198	2.129	2.184	0.919
Interpolative	2.192	2.222	2.158	2.146	2.146	0.879

• Tiempo de compresión en segundos con grafo en orden Indegree:

Tabla 44: tiempo de compresión en [seg] usando formato Indegree para web-Stanford, aplicando todos las formas de renumeración mencionadas.

Compresor	Comunidad Indegree	Comunidad Outdegree	Indegree	Outdegree	Random	Sin Renumeración
VByte	2.321	2.336	2.287	2.327	2.331	1.026
S9	2.195	2.221	2.182	2.166	2.225	0.917
PForDelta	2.186	2.201	2.204	2.139	2.184	0.897
RL+VByte	2.375	2.371	2.363	2.360	2.295	1.110
RL+S17	2.183	2.208	2.175	2.197	2.186	0.913
RL+PForDelta	2.244	2.205	2.151	2.208	2.203	0.912
Interpolative	2.131	2.156	2.133	2.104	2.101	0.857

- **Gráfico resumen tasa de compresión formato Outdegree:**

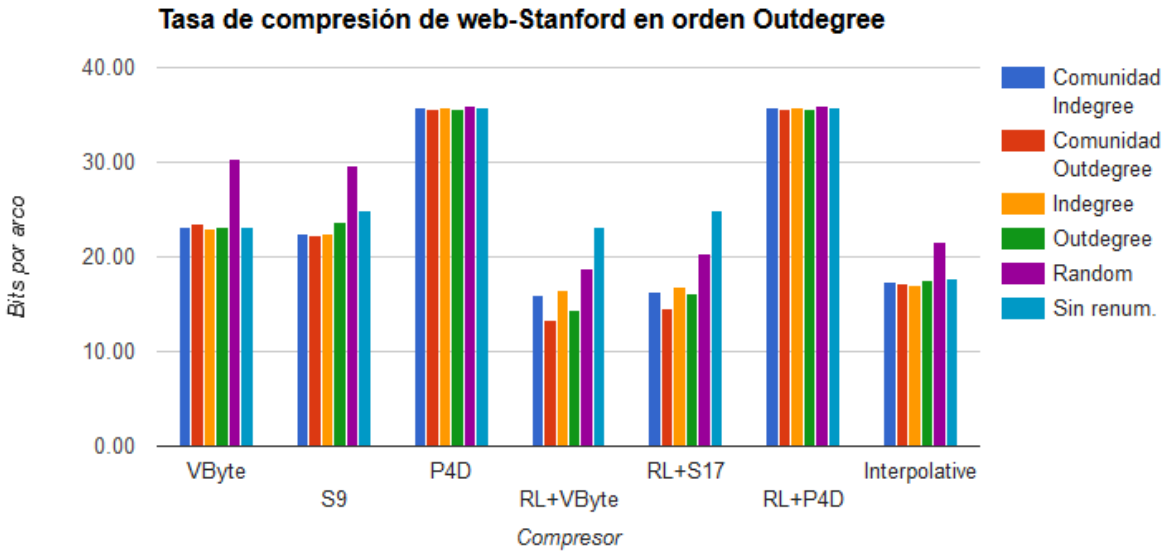


Figura 20: gráfico resumen compresión de web-Stanford en formato Outdegree.

- **Gráfico resumen tasa de compresión orden Indegree:**

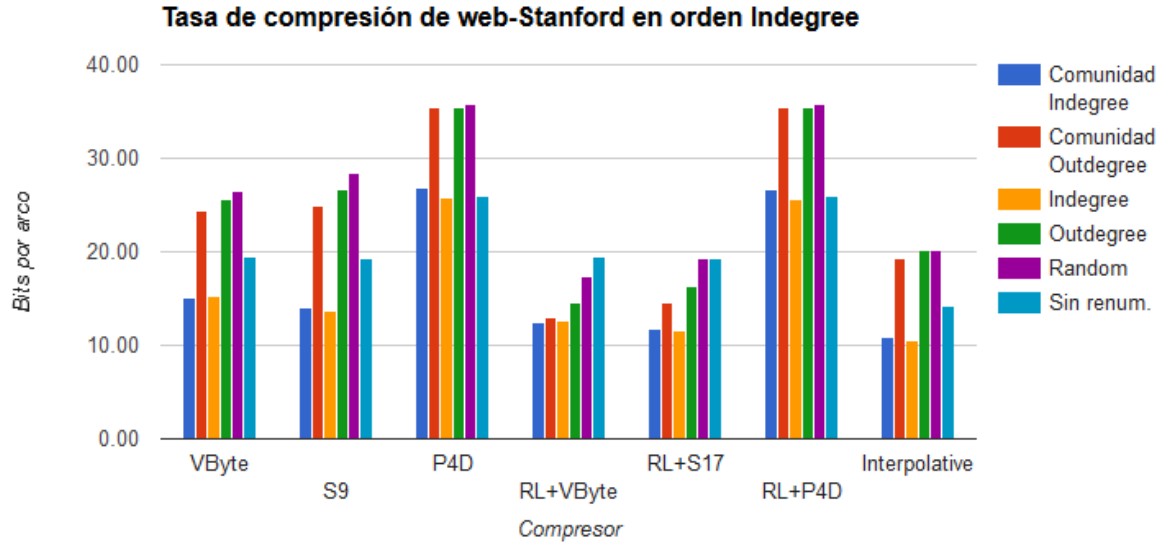


Figura 21: gráfico resumen compresión de web-Stanford en orden Indegree.