Diseño e implementación de método de compresión de grafos basados en clustering de cliques maximales

Defensa de Tesis

Felipe A. Glaría Grego

Lilian Salinas, Cecilia Hernández
Departamento de Ingeniería Informática y Ciencias de la Computación
Facultad de Ingeniería
Universidad de Concepción

2 de octubre de 2019

Motivación

Gran crecimiento en grafos de redes sociales y de la web.

- Estimación sitios indexados: 5,68 mil millones[1].
- Usuarios activos diarios en Facebook: 1,56 mil millones. Crecimiento de 8 % anual[2].

Estos grafos son muy usados por algoritmos de ranking, detección de SPAM, detección de comunidades y actores relevantes, entre otros.
Alto costo en recursos que demanda su procesamiento.

- Principalmente espacio en memoria.
- Jerarquía de memoria penaliza tiempo de acceso a datos alejados de unidades de procesamiento.

[1] `http://www.worldwidewebsize.com/`, consultado el 07 de agosto del 2019.

[2] `https://investor.fb.com`, informe de resultados del primer trimestre del 2019.

Motivación (2)

Proponer estructuras compactas que permitan navegación basado en consultas básicas.
Modelo propuesto enumera cliques maximales de un grafo, y luego los representa en una estructura compacta.

Marco teórico

- Un **grafo** $G = (V, E)$ como el conjunto finito de *vértices* V (nodos) y el conjunto de *aristas* $E \subseteq V \times V$ (arcos).
- Dos vértices v_1 y $v_2 \in V(G)$ son **adyacentes** o **vecinos** si $(v_1, v_2) \in E(G)$ y $v_1 \neq v_2$.
- Un grafo es **no dirigido** cuando la arista conlleva ambos sentidos, es decir $(v_1, v_2) = (v_2, v_1)$.
- El **grado de un vértice** $d(v)$ es la cantidad de vértices en $V(G)$ que son adyacentes con v.
- La **matriz de adyacencia** de un grafo G corresponde a una matriz binaria cuadrada $|V(G)| \times |V(G)|$ donde cada celda (i, j) almacena un 1 si el par de vértices v_i y $v_j \in V(G)$ son vecinos. De lo contrario almacena un 0.

Marco teórico (2)

- Un **clique** es un subgrafo donde todos los vértices son adyacentes entre sí. Un **clique maximal** no es subconjunto de otro clique más grande.

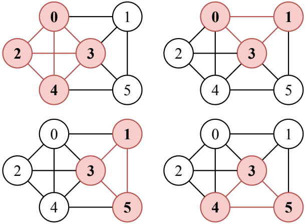


Figura 1: Ejemplo de grafo y sus cliques maximales.

Marco teórico (3)

- Un **triángulo** es un subgrafo de tres vértices y tres aristas. Se define $\lambda(v)$ como la cantidad de triángulos donde participa un nodo v.
- Se define $\lambda(G)$ como la cantidad de triángulos de un grafo. Se calcula sumando $\lambda(v)$ para cada vértice v, y dividiendo el total en tres.

$$\lambda(G) = \frac{1}{3} \sum_{v \in V} \lambda(v) \tag{1}$$

- Un **triplete** es un subgrafo de tres vértices y dos aristas, donde las aristas comparten un vértice común. Se define $\tau(v)$ como la cantidad de tripletes donde v es el vértice común.
- Se define $\tau(G)$ como la cantidad de tripletes de un grafo.

$$\tau(G) = \sum_{v \in V} \tau(v) \tag{2}$$

Marco teórico (4)

- El **coeficiente de clusterización** de un vértice indica cuánto está conectado con sus vecinos, y se define como $c(v) = \lambda(v)/\tau(v)$.
- El coeficiente de clusterización de un grafo $(C(G))$ es el promedio del coeficiente de todos los nodos del grafo.

$$C(G) = \frac{1}{|V'|} \sum_{v \in V'} c(v) \tag{3}$$

con $V' = \{v \in V | d(v) \geq 2\}$.

- La **transitividad** de un grafo $(T(G))$ es la probabilidad que un par de nodos adyacentes estén interconectados.

$$T(G) = \frac{3\lambda(G)}{\tau(G)} \tag{4}$$

Marco teórico (5)

- La **codificación Huffman** es un algoritmo greedy basado en definir códigos mas cortos para aquellos elementos mas frecuentes.
- Es un técnica de compresión de datos óptima que define códigos de largo variable libre de prefijos.

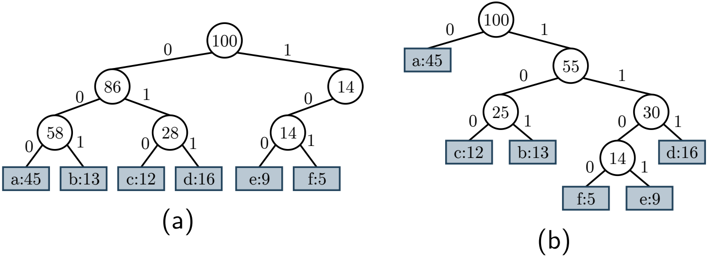


Figura 2: Ejemplos de Árboles: (a) Árbol para código de largo fijo. (b) Árbol para código Huffman.

Trabajos Relacionados

Compresión de grafos

- **The WebGraph Framework**, *Boldi y Vigna* (2003).
- **Graph Compression by BFS**, *Apostolico y Drovandi* (2009).
- **Using Re-Pair**, *Claude y Navarro* (2010).
- **Virtual Node Mining**, *Buehrer y Chellapilla* (2008).
- **k2-tree**, *Brisaboa, Ladra y Navarro* (2009).
- **List Merging**, *Grabowski y Bieniecki* (2014).

Trabajos Relacionados (2)

Estructuras compactas: Tres operaciones básicas:

- $Rank_S(a, i)$: Cuenta las ocurrencias del símbolo a hasta la posición i en la secuencia S.
- $Select_S(a, i)$: Encuentra la posición de la ocurrencia i del símbolo a en la secuencia S.
- $Access_S(i)$: Retorna el símbolo en la posición i de la secuencia S.

Se utilizarán:

- Secuencias binarias, *Raman, Raman y Rao*.
- Wavelet tree, *Grossi, Gupta y Vitter*.
- Wavelet matrix, *Claude, Navarro y Ordóñez*.

Disponibles en el repositorio **SDSL**[3] (*Succinct Data Structure Library*).

[3] `https://github.com/simongog/sdsl-lite`

Trabajos Relacionados (3)

Enumeración de cliques maximales

- Es un problema NP-Hard.
- *Eppstein, Löffler y Strash* proponen un método apropiado para grafos poco densos.
- Para un grafo con n nodos y *degeneracy* d, en un tiempo $O(dn3^{d/3})$

Disponible en el repositorio **Quick Cliques**[4].

[4] `https://github.com/darrenstrash/quick-cliques`

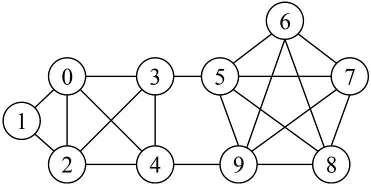
Método de Compresión Propuesto

Consta de tres etapas:

1. Listar todos los cliques maximales del grafo.
2. Particionar listado de cliques, utilizando heurística eficiente que explote su superposición.
3. Comprimir particiones en estructura compacta.

Etapa 1: Cliques Maximales

Se obtiene el listado de cliques, usando **Quick Cliques**[5].



(a)

$C_0 : 0, 1, 2$

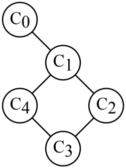
$C_1 : 0, 2, 3, 4$

$C_2 : 3, 5$

$C_3 : 5, 6, 7, 8, 9$

$C_4 : 4, 9$

(b)



(c)

Figura 3: (a) Grafo no dirigido. (b) Lista de cliques maximales. (c) Grafo de cliques.

[5] https://github.com/darrenstrash/quick-cliques

Etapa 2: Particionar listado de cliques

Problema

Encontrar particiones de cliques para el grafo de cliques $CG_{\mathcal{C}}$. Dado un grafo de cliques $CG_{\mathcal{C}} = (V_{\mathcal{C}}, E_{\mathcal{C}})$, encontrar un set de particiones de cliques $\mathcal{CP} = \{cp_1, cp_2, ..., cp_M\}$ de $CG_{\mathcal{C}}(V_{\mathcal{C}}, E_{\mathcal{C}})$ con $M \geq 1$, tal que

1. $\bigcup_{i=1}^{M} cp_i = CG_{\mathcal{C}}$
2. $cp_i \cap cp_j = \varnothing$ *para* $i \neq j$
3. *cualquier* $cp_i \in \mathcal{CP}$ *es un subgrafo de* $CG_{\mathcal{C}}(V_{\mathcal{C}}, E_{\mathcal{C}})$ *inducido por el subset de vértices en* cp_i

Etapa 2: Algoritmo de particionamiento

Definir una heurística que permita agrupar los cliques en particiones eficientes, que exploten su redundancia de vértices.

Definición

Función de ranking

Dado un grafo $G = (V, E)$ *y* $\mathcal{C} = \{c_1, c_2, ..., c_N\}$ *el conjunto de tamaño* N *de cliques maximales que cubren* G*, una función de ranking es una función* $r : V \to \mathbb{R}^+$ *que retorna un valor de puntuación para cada vértice* $v \in V$*.*

Etapa 2: Algoritmo de particionamiento (2)

Funciones de ranking propuestas, para $C(u) = \{c \in \mathcal{C} | u \in c\}$:

$$r_f(u) = |C(u)| \tag{5}$$

$$r_c(u) = \sum_{c \in C(u)} |c| \tag{6}$$

$$r_r(u) = \frac{r_c(u)}{r_f(u)} \tag{7}$$

Etapa 2: Algoritmo de particionamiento (3)

$$C_0 : 0, 1, 2$$
$$C_1 : 0, 2, 3, 4$$
$$C_2 : 3, 5$$
$$C_3 : 5, 6, 7, 8, 9$$
$$C_4 : 4, 9$$

$u \in G$	0	1	2	3	4	5	6	7	8	9
R_{rf}	2,0	1,0	2,0	2,0	2,0	1,0	1,0	1,0	1,0	2,0
R_{rc}	7,0	3,0	7,0	6,0	6,0	7,0	5,0	5,0	5,0	7,0
R_{rr}	3,5	3,0	3,5	3,0	3,0	3,5	5,0	5,0	5,0	3,5

Figura 4: Listado de cliques y puntajes de ranking asociados.

Etapa 2: Algoritmo de particionamiento (4)

Algorithm 1 Algoritmo de particionamiento del grafo de cliques.

Require: $\mathcal{C}$ maximal clique collection ($N = |\mathcal{C}|$), ranking function $r(u)$
Ensure: Returns clique-graph partition collection $\mathcal{CP}$

$(D, R) \leftarrow computeRanking(r, \mathcal{C})$ (array D y R, $\forall u \in V$)
Initialize bit array Z of size N and set each bit to 0
for $u \in R$ **do**
    $cpid \leftarrow \emptyset$
    for $id \in D[u]$ and $Z[id] = 0$ **do**
        $Z[id] \leftarrow 1$
        $cpid \leftarrow cpid \cup \{id\}$
    end for
    if $cpid \neq \emptyset$ **then**
        $\mathcal{CP} \leftarrow \mathcal{CP} : cpid$
    end if
end for
return $\mathcal{CP}$

Complejidad: $O(N + V)$

Etapa 2: Algoritmo de particionamiento (5)

$$C_0 : 0, 1, 2$$
$$C_1 : 0, 2, 3, 4$$
$$C_2 : 3, 5$$
$$C_3 : 5, 6, 7, 8, 9$$
$$C_4 : 4, 9$$

$u \in G$	0	1	2	3	4	5	6	7	8	9
R_{rf}	2,0	1,0	2,0	2,0	2,0	1,0	1,0	1,0	1,0	2,0
R_{rc}	7,0	3,0	7,0	6,0	6,0	7,0	5,0	5,0	5,0	7,0
R_{rr}	3,5	3,0	3,5	3,0	3,0	3,5	5,0	5,0	5,0	3,5

$\mathcal{CP}_{rf}$	C_0 C_1	C_2	C_4	C_3
$\mathcal{CP}_{rc}$	C_0 C_1	C_2 C_3	C_4	
$\mathcal{CP}_{rr}$	C_3	C_0 C_1	C_2	C_4

Figura 5: Particiones de cliques para cada función de ranking.

Etapa 3: Estructuras Compactas - Secuencias

- **X**: Listas concatenadas de vértices en $\mathcal{CP}$.
- **B**: Bitmap indicando inicio de particiones.
- **BB**: Secuencia de bytes codificando presencia de vértices en cliques por partición.
- **Y**: Secuencia de enteros indicando primer byte en **BB** para cada partición.

Etapa 3: Estructuras Compactas - Secuencias (2)

$$C_0 : 0, 1, 2$$
$$C_1 : 0, 2, 3, 4$$
$$C_2 : 3, 5$$
$$C_3 : 5, 6, 7, 8, 9$$
$$C_4 : 4, 9$$

$\mathcal{CP}_{rr}$				
C_0 C_1	C_3	C_2	C_4	

X:	0	1	2	3	4	5	6	7	8	9	3	5	4	9	
B:	1	0	0	0	0	1	0	0	0	0	1	0	1	0	1
BB:	3	1	3	2	2										
Y:	0	5													

Figura 6: Secuencias finales.

Algoritmos de consulta - Reconstrucción secuencial

Recorre las particiones en orden secuencial.

- Si contiene solo un clique, sus vértices son vecinos.
- Si no, se comparan los bytes de cada vértice entre si.
  - Si la comparación es distinto a cero, son vecinos.
  - No es necesario comparar siguientes bytes.

Complejidad: $\begin{cases} O(P_0 \cdot N^2), & bpu_p = 0 \\ O(P_1 \cdot N^2 \cdot bpu_p), & bpu_p \neq 0 \end{cases}$

bpu_p: Bytes por vértice de particiones.
P_0: Particiones que no tienen bytes por vértice.
P_1: Particiones que sí tienen bytes por vértice.

N: Largo de particiones.

Algoritmos de consulta - Listado de vecinos

Para un vértice u aleatorio.

- Cuenta ocurrencias de u en X.
- Por cada una, se identifica su partición.
    - Si contiene solo un clique, sus vértices son vecinos de u.
    - Si no, se comparan sus bytes con los de cada posible vecino.
        - Si la comparación es distinto a cero, son vecinos.
        - No es necesario comparar siguientes bytes.

Complejidad: $\begin{cases} O(M_0 \cdot N), & bpu_p = 0 \\ O(M_1 \cdot N \cdot bpu_p), & bpu_p \neq 0 \end{cases}$

bpu_p: Bytes por vértice de particiones.
M_0: Particiones que contienen a u y no tienen bytes por vértice.
M_1: Particiones que contienen a u y sí tienen bytes por vértice.

N: Largo de particiones.

Algoritmos de consulta - Vecindad de dos vértices

Para dos vértices u_1, u_2 aleatorios.

- Cuenta ocurrencias de ambos vértices en X.
- Revisa si coinciden en una partición
    - Si coinciden, se comparan sus bytes.
        - Si la comparación es distinto a cero, son vecinos.
        - No es necesario comparar siguientes bytes.

Complejidad: $\begin{cases} O(M_1 + M_2), & bpu_p = 0 \\ O((M_1 + M_2) \cdot bpu_p), & bpu_p \neq 0 \end{cases}$

bpu_p: Bytes por vértice de particiones.
M_1: Particiones que contienen a u_1.
M_2: Particiones que contienen a u_2.

Algoritmos de consulta - Cliques maximales

Recorre las particiones en orden secuencial.

- Si contiene un solo clique, lista sus vértices.
- Si no, se revisan los bytes de cada vértice ordenadamente.
  - Si comparten un 1 en mismo bit, son un clique.

Complejidad: $\begin{cases} O(P_0 \cdot N), & bpu_p = 0 \\ O(P_1 \cdot N \cdot 8 \cdot bpu_p), & bpu_p \neq 0 \end{cases}$

bpu_p: Bytes por vértice de particiones.
P_0: Particiones que no tienen bytes por vértice.
P_1: Particiones que sí tienen bytes por vértice.

N: Largo de particiones.

Resultados - Grafos

Tabla 1: Cantidad de vértices, aristas y cliques de los grafos a comprimir.

Grafo	$\|V\|$	$\|E\|$	$\|\mathcal{C}\|$
marknewman-astro	16.706	242.502	15.794
marknewman-condmat	40.421	351.386	34.274
dblp-2010	326.186	1.615.400	196.434
dblp-2011	986.324	6.707.236	806.320
snap-dblp	317.080	2.099.732	257.551
snap-amazon	403.394	4.886.816	1.023.572
coPapersDBLP	540.486	30.491.458	139.340
coPapersCiteseer	434.102	32.073.440	86.303

Resultados - Grafos (2)

Tabla 2: Grado medio y máximo de los vértices, degeneracy, coeficiente de clusterización y transitividad de los grafos a comprimir.

Grafo	$\overline{d}$	d_{max}	$D(G)$	$C(G)$	$T(G)$
marknewman-astro	14,51	360	56	0,66	0,42
marknewman-condmat	8,69	278	29	0,64	0,24
dblp-2010	4,95	238	74	0,61	0,39
dblp-2011	6,80	979	118	0,63	0,20
snap-dblp	6,62	2.752	113	0,63	0,30
snap-amazon	12,11	343	10	0,41	0,16
coPapersDBLP	56,41	3.299	336	0,80	0,65
coPapersCiteseer	73,88	1.188	844	0,83	0,77

Resultados - Distribución del grado de grafos

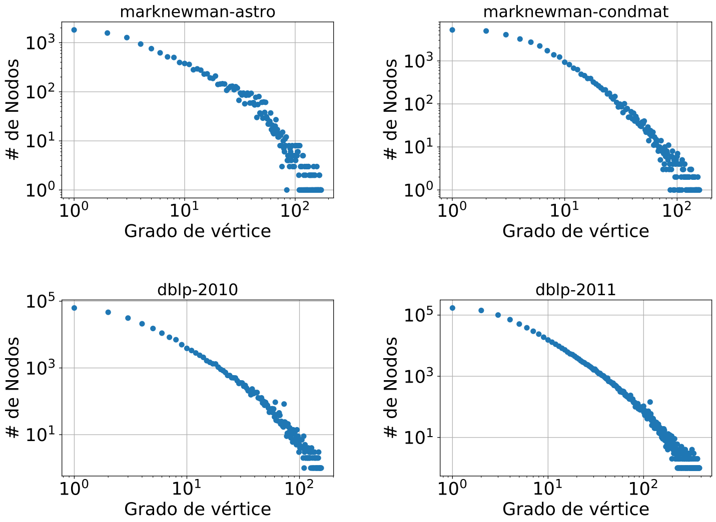


Figura 7: Distribución del grado de los vértices para cada grafo (1).

Resultados - Distribución del grado de grafos (2)

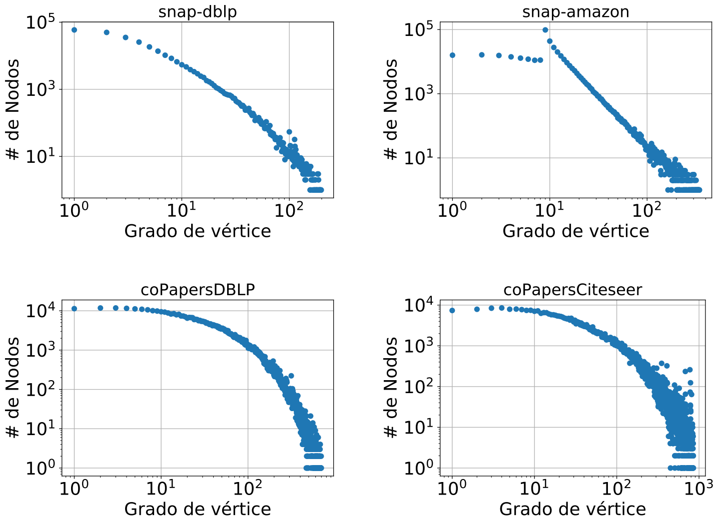


Figura 8: Distribución del grado de los vértices para cada grafo (2).

Resultados - Estructura compacta

Se utiliza `SDSL`. Se seleccionan las siguientes estructuras:

- Para las secuencias de símbolos X e Y: estructuras basadas en wavelet matrix (*wm*).
- Para la secuencia de bits B: estructuras basadas en bitmaps comprimidos de Raman, Raman y Rao (*rrr*).

La secuencia de bytes BB se comprime usando código Huffman.

- Se actualizan índices en Y para acceso por bits.

Resultados - Funciones de ranking

Tabla 3: Comparativa de BPE de las estructuras compactas para las funciones de ranking.

Grafo	r_r	r_c	r_f
marknewman-astro	3,96	3,86	**3,82**
marknewman-condmat	5,74	5,47	**5,44**
dblp-2010	5,85	5,97	**5,73**
dblp-2011	6,58	6,63	**6,48**
snap-dblp	6,89	6,50	**6,41**
snap-amazon	**10,44**	10,53	10,46
coPapersDBLP	0,78	0,79	**0,76**
coPapersCiteseer	0,52	0,50	**0,48**

Resultados - Funciones de ranking (2)

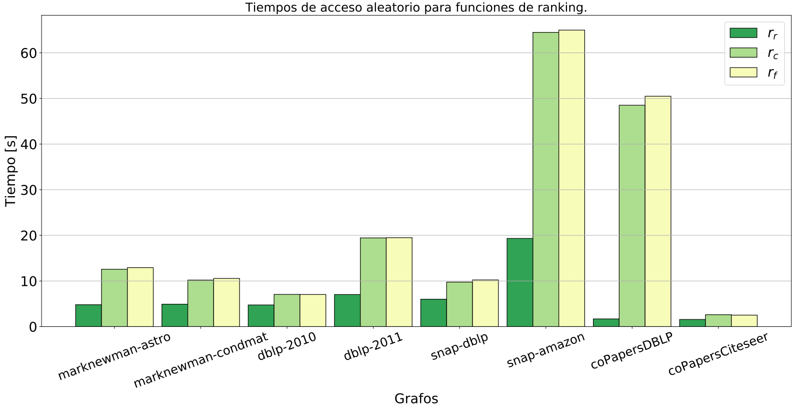


Figura 9: Tiempos de acceso aleatorio para las funciones de ranking.

Resultados - Tiempos de generación

Tabla 4: Tiempos de obtención de listado de cliques maximales y construcción de la estructura compacta, en segundos.

Grafo	$t_{\mathcal{C}}$	t_{CS}	t_T	$t'_{\mathcal{C}}$
marknewman-astro	0,18	0,28	0,46	0,05
marknewman-condmat	0,28	0,40	0,68	0,11
dblp-2010	1,12	1,46	2,58	0,57
dblp-2011	5,58	7,30	12,88	2,77
snap-dblp	1,68	2,30	3,98	0,86
snap-amazon	5,93	8,44	14,37	3,01
coPapersDBLP	17,96	3,44	21,40	1,39
coPapersCiteseer	26,70	4,70	31,40	0,96

$t_{\mathcal{C}}$: generación del listado de cliques maximales $\mathcal{C}$.

t_{CS}: generar la estructura compacta desde listado de cliques $\mathcal{C}$.

$t_T = t_{\mathcal{C}} + t_{CS}$

$t'_{\mathcal{C}}$: recuperar el listado de cliques $\mathcal{C}$.

Resultados - Comparando con estado del arte

Para la estructura compacta propuesta:

- C_{rf}: Usando la estructura con función de ranking $r_f(u)$.
- C_{rr}: Usando la estructura con función de ranking $r_r(u)$.

En el caso de Webgraph:

- WG_s: Para el caso de acceso secuencial.
- WG_a: Para el caso de acceso aleatorio.

Para el caso de k2tree:

- $k2T$: Cuando el algoritmo usa el orden del grafo original.
- $k2T_{BFS}$: Cuando el algoritmo usa el orden por BFS.

AD: El algoritmo BFS de Apostolico y Drovandi.

Resultados - Comparando con estado del arte (2)

Tabla 5: BPE de algoritmos de compresión.

Grafo	C_{rf}	C_{rr}	$k2T$	$k2T_{BFS}$	AD	WG_a	WG_s
marknewman-astro	**3,82**	3,96	4,89	4,34	5,67	8,10	7,30
marknewman-condmat	**5,44**	5,74	6,28	5,60	7,86	11,78	10,45
dblp-2010	6,41	5,84	**4,23**	4,30	6,71	8,67	6,91
dblp-2011	10,46	6,58	**5,48**	5,89	9,67	10,13	8,71
snap-dblp	5,73	6,89	5,88	**5,23**	8,14	11,80	10,17
snap-amazon	6,48	10,44	8,02	**6,38**	10,96	14,50	13,35
coPapersDBLP	**0,76**	0,78	1,67	0,94	1,81	2,71	2,48
coPapersCiteseer	0,48	0,52	1,21	**0,45**	0,85	1,79	1,63

Resultados - Comparando con estado del arte (3)

Tabla 6: Tiempos de acceso aleatorio, en microsegundos por arco.

Grafo	C_{rf}	C_{rr}	$k2T$	$k2T_{BFS}$	AD	WG_a
marknewman-astro	2,97	2,67	2,58	1,33	1,79	**0,052**
marknewman-condmat	3,32	3,16	5,53	2,81	2,32	**0,063**
dblp-2010	3,80	3,70	5,55	4,84	2,15	**0,097**
dblp-2011	5,21	4,66	11,43	10,69	2,36	**0,114**
snap-dblp	4,06	4,07	10,35	6,93	2,30	**0,125**
snap-amazon	12,17	6,99	13,97	7,13	2,47	**0,087**
coPapersDBLP	1,69	1,51	1,89	1,16	0,73	**0,045**
coPapersCiteseer	1,25	1,30	0,95	0,50	0,45	**0.037**

Resultados - Comparando con estado del arte (4)

Tabla 7: Tiempos de reconstrucción secuencial del grafo, en segundos.

Grafo	C_{rf}	C_{rr}	$k2T$	$k2T_{BFS}$	WG_s
marknewman-astro	0,09	0,09	0,03	**0,02**	0,28
marknewman-condmat	0,16	0,16	0,07	**0,04**	0,52
dblp-2010	0,79	0,82	0,18	**0,16**	1,09
dblp-2011	4,61	4,45	**1,10**	1,31	2,41
snap-dblp	1,16	1,26	0,58	**0,35**	1,20
snap-amazon	7,09	4,53	1,36	**1,13**	1,30
coPapersDBLP	5,68	5,81	1,45	**1,01**	1,59
coPapersCiteseer	4,62	5,46	1,33	**0,65**	1,56

Conclusiones

Se desarrolla método de compresión:

- Para grafos no dirigidos y poco densos.
- Basado en clustering de cliques maximales.
- Usando estructuras compactas.

Estructura comprimida permite responder consultas:

- Reconstrucción del grafo original.
- Listado de vecinos de un nodo.
- Consultar vecindad de dos nodos.
- Generar listado de cliques maximales.

Conclusiones (2)

Nivel de compresión competitivo al estado del arte

- Solo superado por k2tree.

Buen tiempo de acceso aleatorio

- Competitivo con k2tree.
- Otros algoritmos logran mejor tiempo.

Menor desempeño en reconstrucción secuencial.

- Algunos casos compiten con Webgraph.
- Otros algoritmos logran mejor tiempo.

Conclusiones (3)

Posible aplicación:

- Dispositivos con poca memoria.
- Responder consultas sin descomprimir.

Trabajo futuro:

- Mejorar tiempos de acceso.
  - Nuevas heurísticas en funciones de ranking.
- Potencial de paralelismo.
  - Acceso paralelo por partición.
  - Comparar bytes usando instrucciones paralelas (SIMD[6]).

[6] SIMD: Single Instruction, Multiple Data. Una instrucción, múltiples datos.

Muchas gracias

Resultados - Distribución de tamaño de cliques

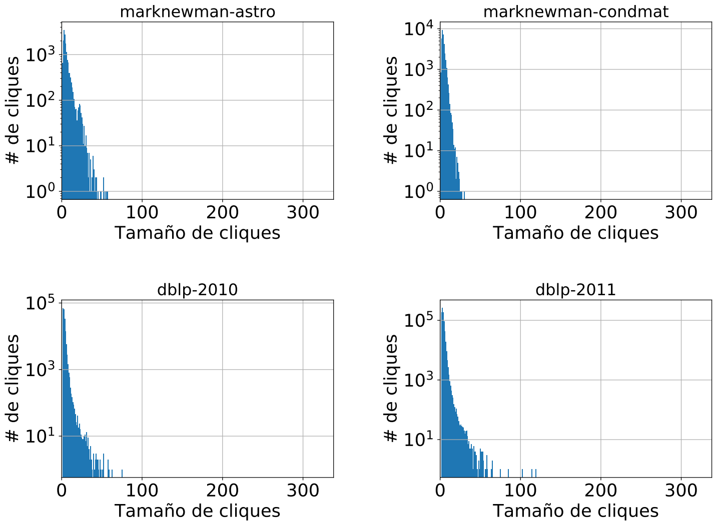


Figura 10: Distribución del grado de los vértices para cada grafo (1).

Resultados - Distribución de tamaño de cliques

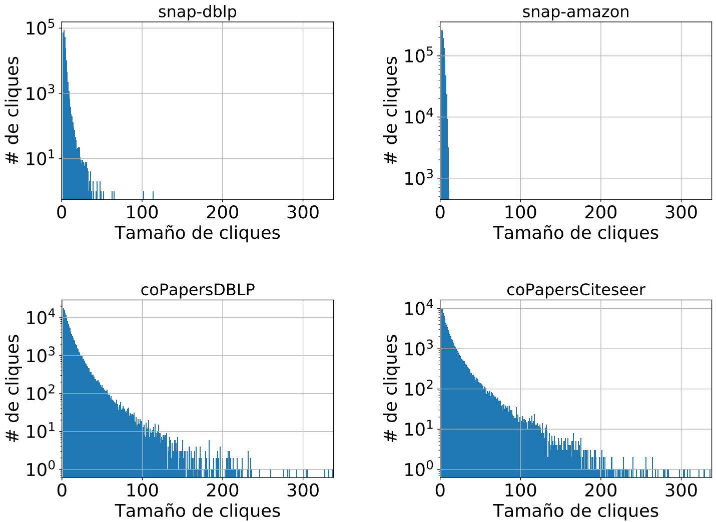


Figura 11: Distribución del grado de los vértices para cada grafo (2).

Resultados - Funciones de ranking

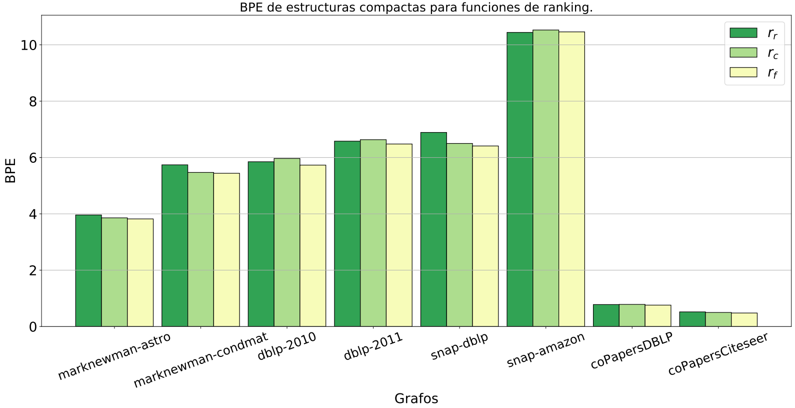


Figura 12: BPE de las estructuras compactas para las funciones de ranking.

Resultados - Funciones de ranking

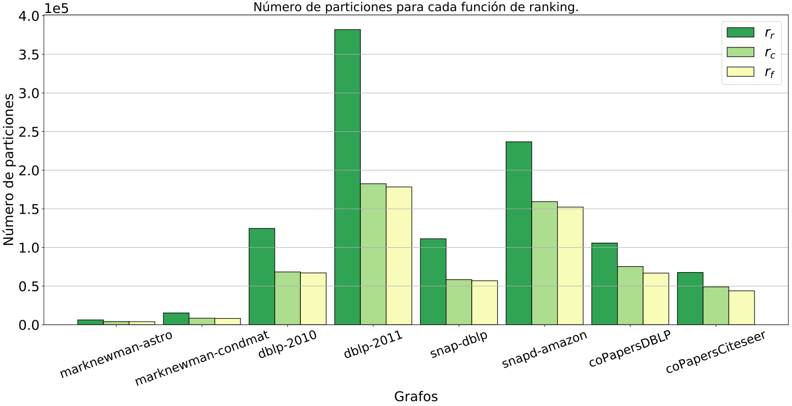


Figura 13: Número de particiones en las estructuras compactas para las funciones de ranking.

Resultados - Funciones de ranking

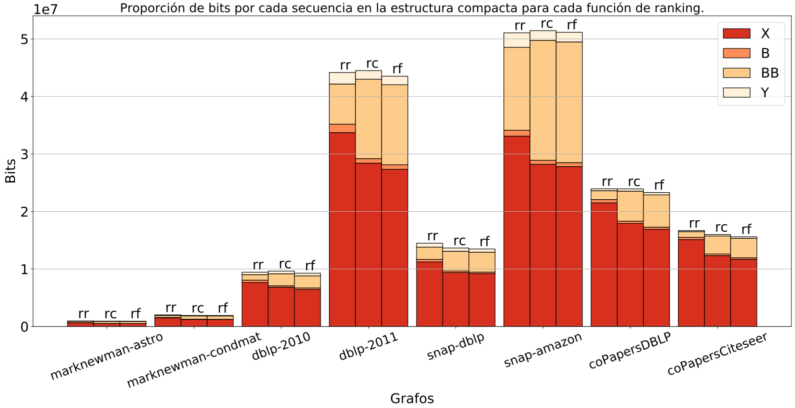


Figura 14: Proporción de bits por cada secuencia en la estructura compacta, para cada función de ranking

Resultados - Funciones de ranking

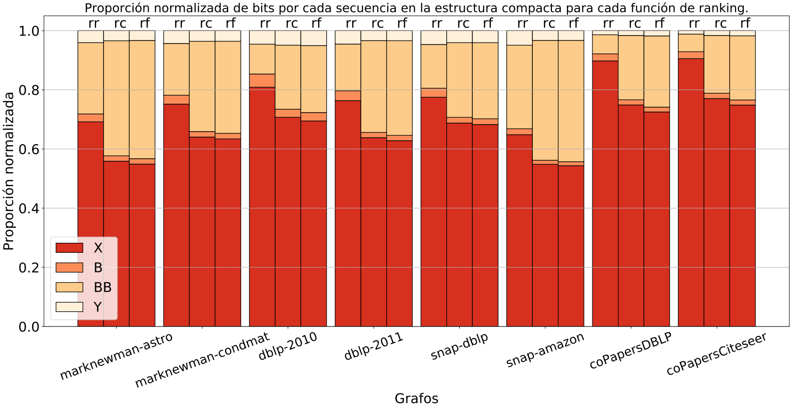


Figura 15: Proporción normalizada de bits por cada secuencia en la estructura compacta, para cada función de ranking

Resultados - Funciones de ranking

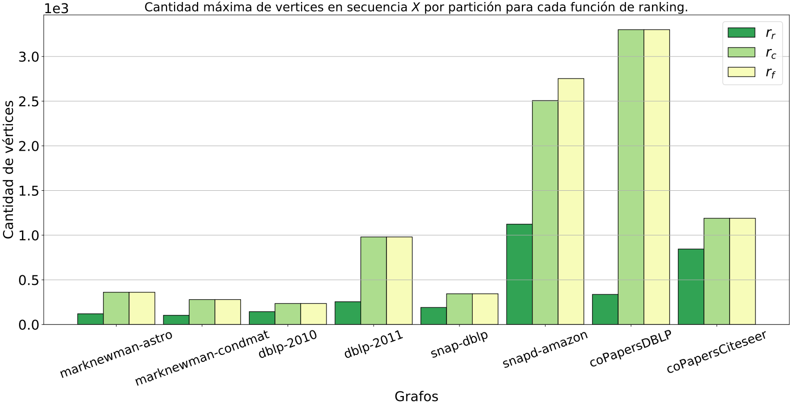


Figura 16: Número máximo de vértices en la secuencia X para las funciones de ranking.

Resultados - Funciones de ranking

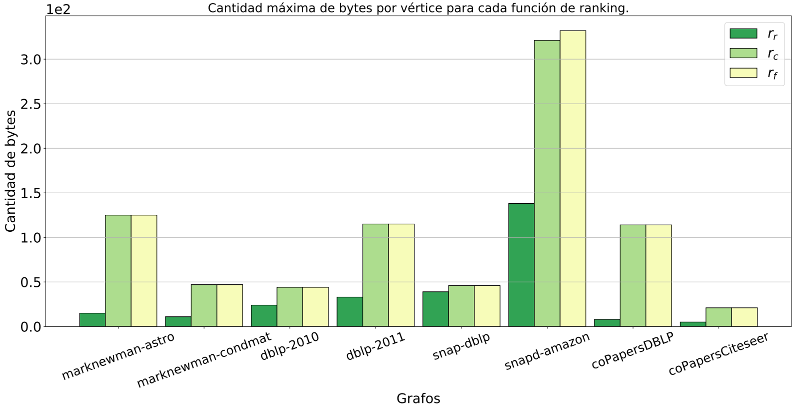


Figura 17: Número máximo de bytes por nodo para las funciones de ranking.

Resultados - Funciones de ranking

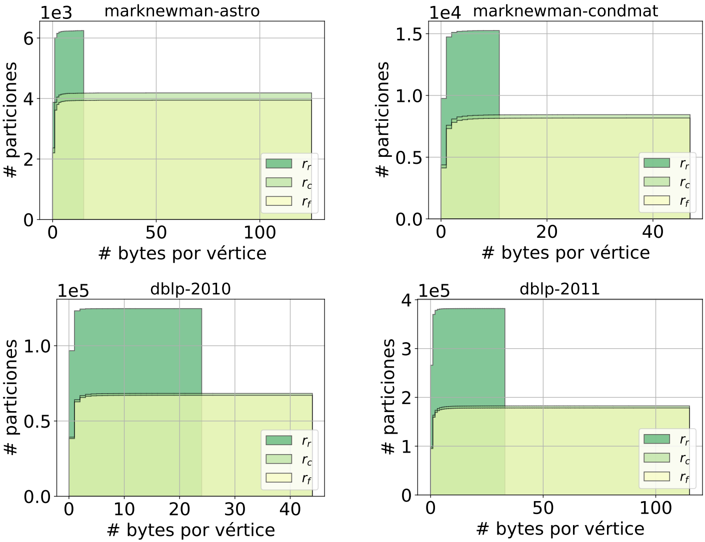


Figura 18: CDF para bytes por vértice en estructuras compactas para cada función de ranking (1).

Resultados - Funciones de ranking

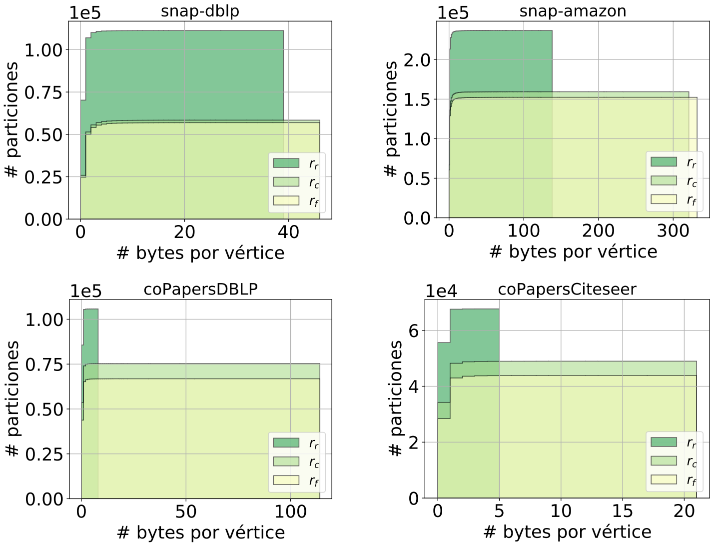


Figura 19: CDF para bytes por vértice en estructuras compactas para cada función de ranking (2).

Resultados - Estructura compacta

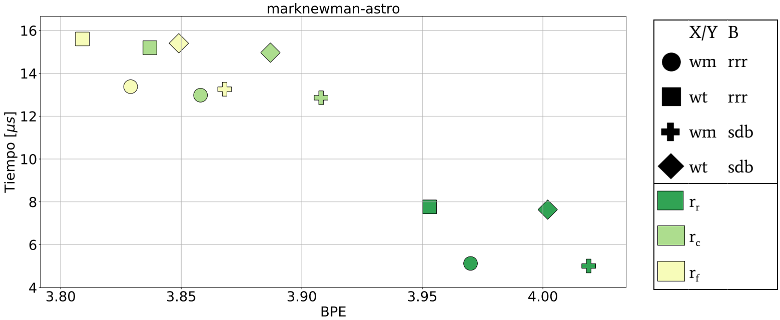


Figura 20: BPE y Tiempo de acceso aleatorio medio para posibles estructuras compactas, por cada función de ranking, para marknewman-astro.

Resultados - Estructura compacta

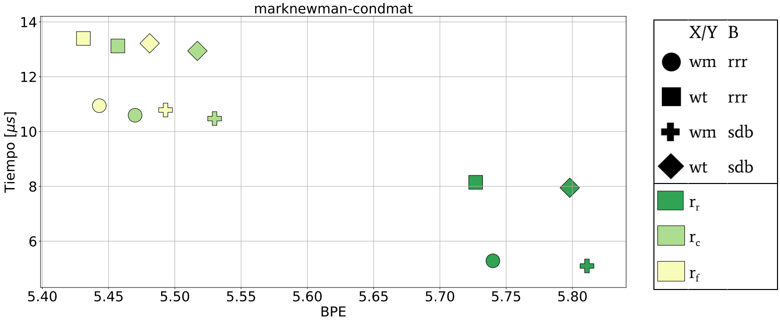


Figura 21: BPE y Tiempo de acceso aleatorio medio para posibles estructuras compactas, por cada función de ranking, para marknewman-condmat.

Resultados - Estructura compacta

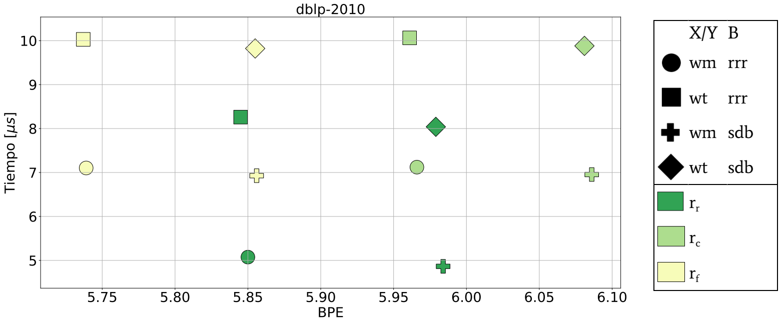


Figura 22: BPE y Tiempo de acceso aleatorio medio para posibles estructuras compactas, por cada función de ranking, para dblp-2010.

Resultados - Estructura compacta

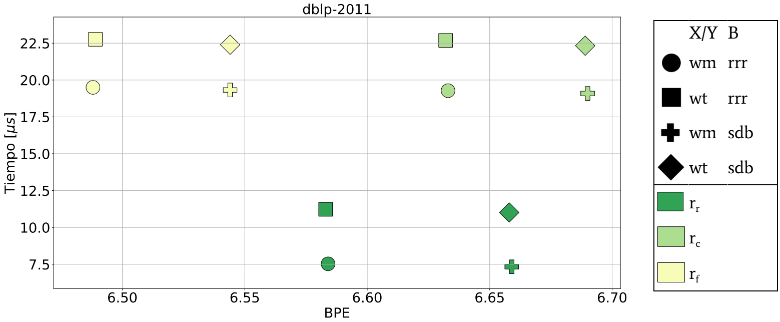


Figura 23: BPE y Tiempo de acceso aleatorio medio para posibles estructuras compactas, por cada función de ranking, para dblp-2011.

Resultados - Estructura compacta

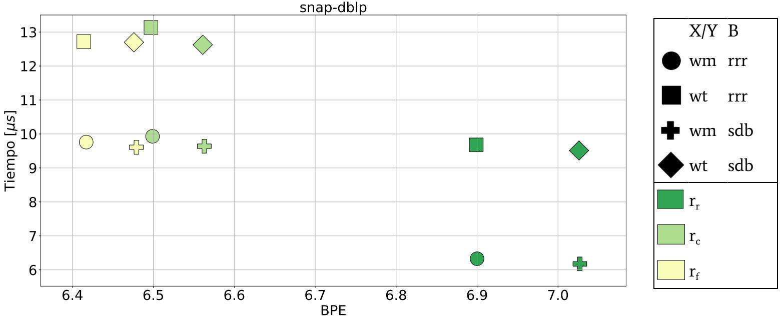


Figura 24: BPE y Tiempo de acceso aleatorio medio para posibles estructuras compactas, por cada función de ranking, para snap-dblp.

Resultados - Estructura compacta

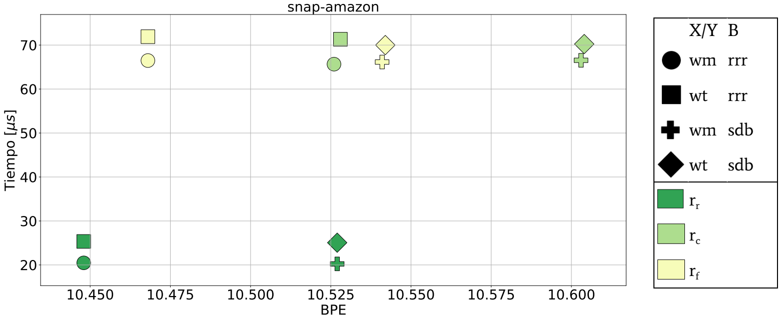


Figura 25: BPE y Tiempo de acceso aleatorio medio para posibles estructuras compactas, por cada función de ranking, para snap-amazon.

Resultados - Estructura compacta

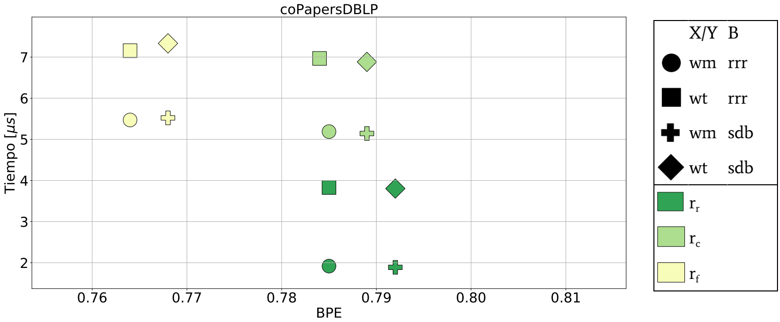


Figura 26: BPE y Tiempo de acceso aleatorio medio para posibles estructuras compactas, por cada función de ranking, para coPapersDBLP.

Resultados - Estructura compacta

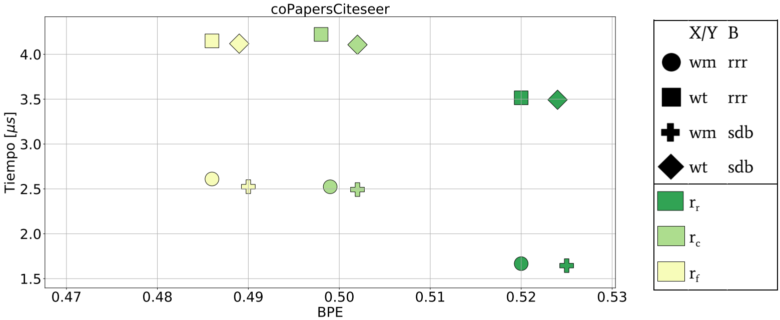


Figura 27: BPE y Tiempo de acceso aleatorio medio para posibles estructuras compactas, por cada función de ranking, para coPapersCiteseer.

Resultados - Estructura compacta

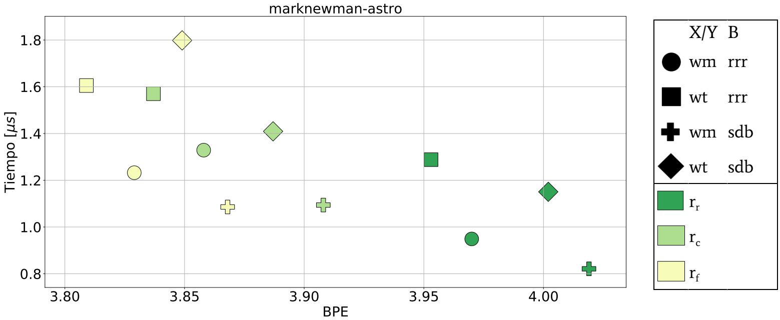


Figura 28: BPE y Tiempo de acceso secuencial medio para posibles estructuras compactas, por cada función de ranking, para marknewman-astro.

Resultados - Estructura compacta

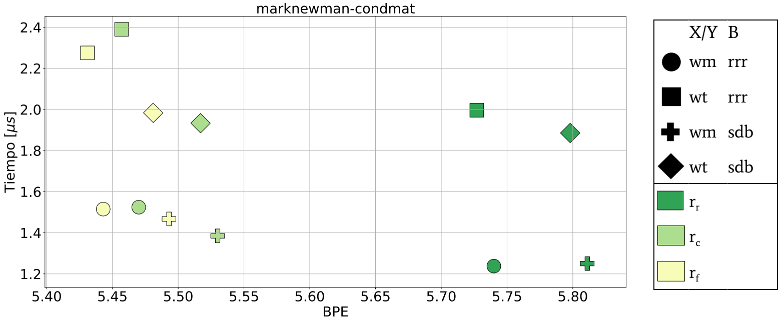


Figura 29: BPE y Tiempo de acceso secuencial medio para posibles estructuras compactas, por cada función de ranking, para marknewman-condmat.

Resultados - Estructura compacta

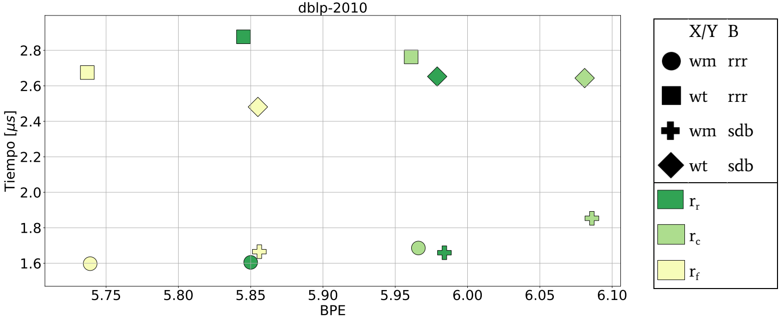


Figura 30: BPE y Tiempo de acceso secuencial medio para posibles estructuras compactas, por cada función de ranking, para dblp-2010.

Resultados - Estructura compacta

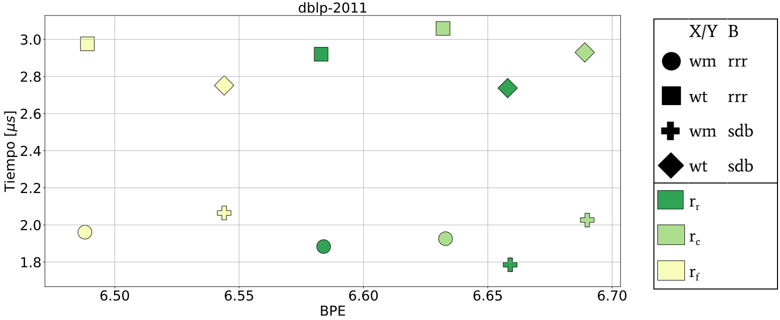


Figura 31: BPE y Tiempo de acceso secuencial medio para posibles estructuras compactas, por cada función de ranking, para dblp-2011.

Resultados - Estructura compacta

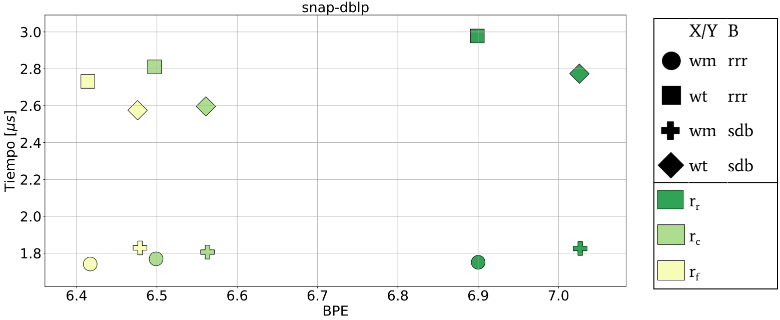


Figura 32: BPE y Tiempo de acceso secuencial medio para posibles estructuras compactas, por cada función de ranking, para snap-dblp.

Resultados - Estructura compacta

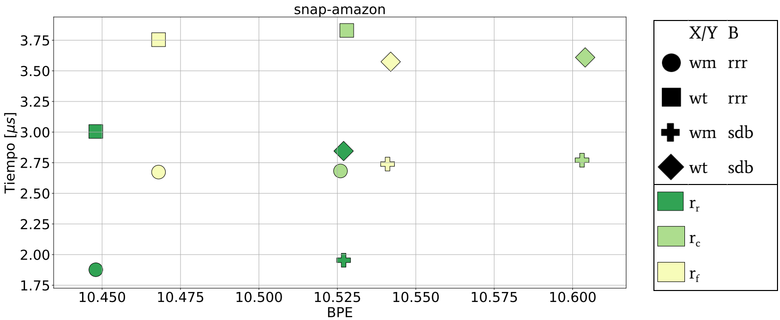


Figura 33: BPE y Tiempo de acceso secuencial medio para posibles estructuras compactas, por cada función de ranking, para snap-amazon.

Resultados - Estructura compacta

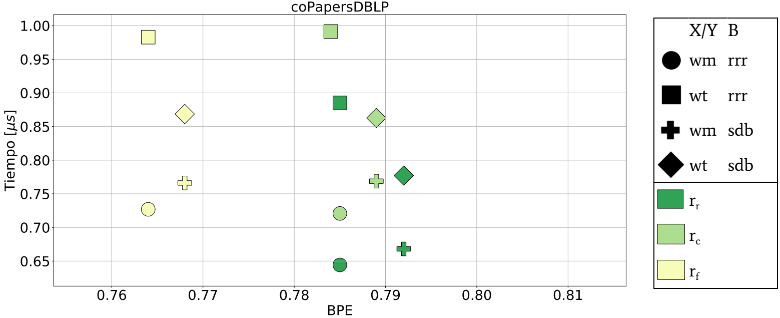


Figura 34: BPE y Tiempo de acceso secuencial medio para posibles estructuras compactas, por cada función de ranking, para coPapersDBLP.

Resultados - Estructura compacta

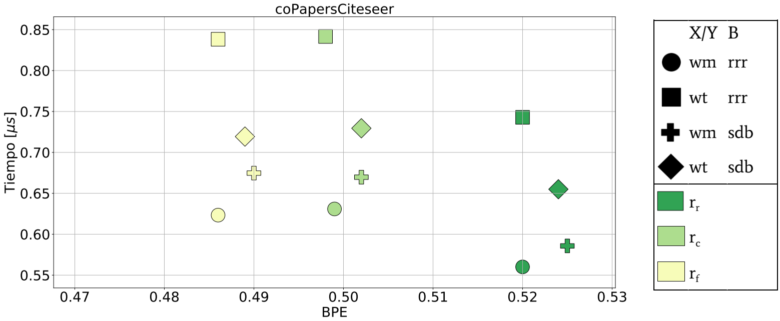


Figura 35: BPE y Tiempo de acceso secuencial medio para posibles estructuras compactas, por cada función de ranking, para coPapersCiteseer.

Resultados - Comparando con estado del arte

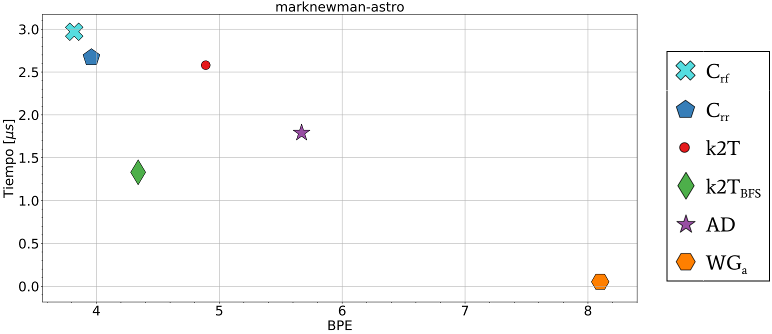


Figura 36: BPE y tiempo de acceso aleatorio en microsegundos de cada algoritmo, para marknewman-astro.

Resultados - Comparando con estado del arte

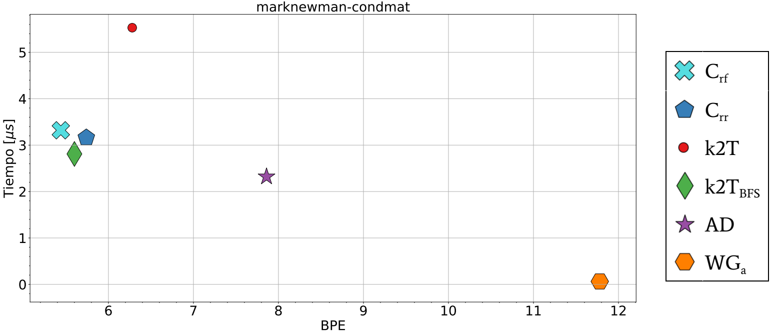


Figura 37: BPE y tiempo de acceso aleatorio en microsegundoss de cada algoritmo, para marknewman-condmat.

Resultados - Comparando con estado del arte

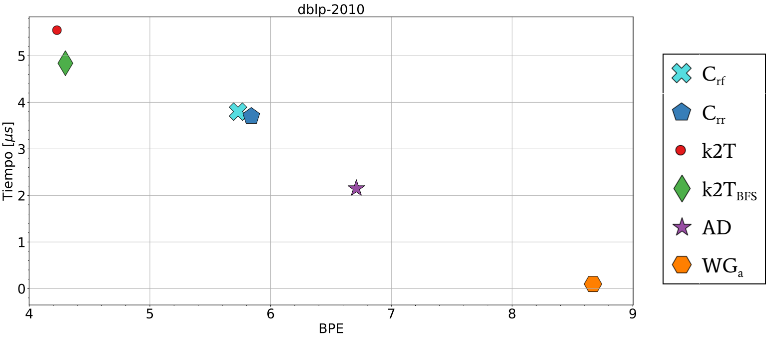


Figura 38: BPE y tiempo de acceso aleatorio en microsegundos de cada algoritmo, para dblp-2010.

Resultados - Comparando con estado del arte

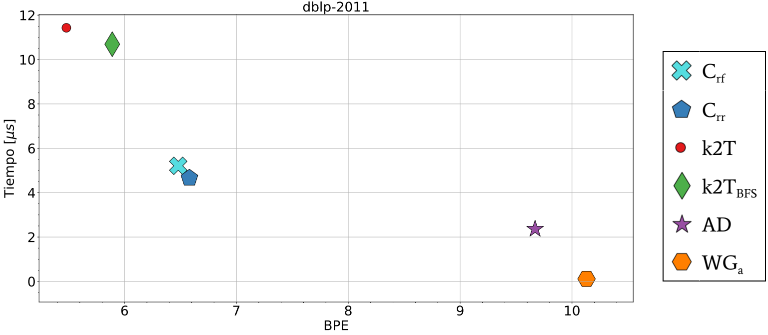


Figura 39: BPE y tiempo de acceso aleatorio en microsegundos de cada algoritmo, para dblp-2011.

Resultados - Comparando con estado del arte

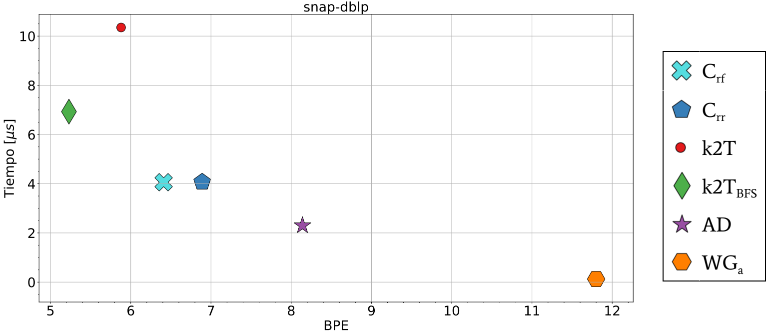


Figura 40: BPE y tiempo de acceso aleatorio en microsegundos de cada algoritmo, para snap-dblp.

Resultados - Comparando con estado del arte

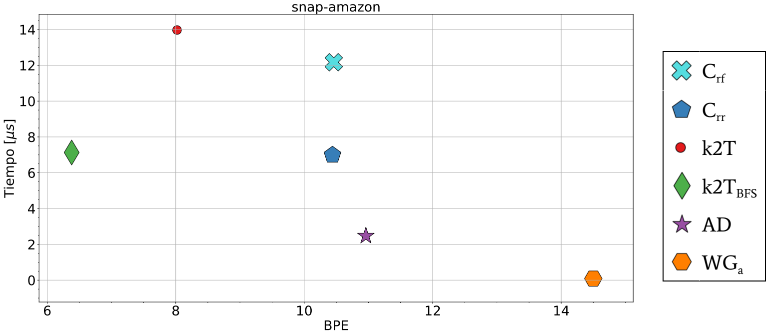


Figura 41: BPE y tiempo de acceso aleatorio en microsegundos de cada algoritmo, para snap-amazon.

Resultados - Comparando con estado del arte

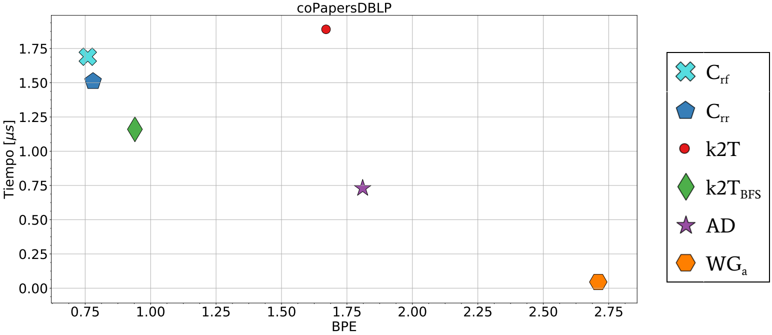


Figura 42: BPE y tiempo de acceso aleatorio en microsegundos de cada algoritmo, para coPapersDBLP.

Resultados - Comparando con estado del arte

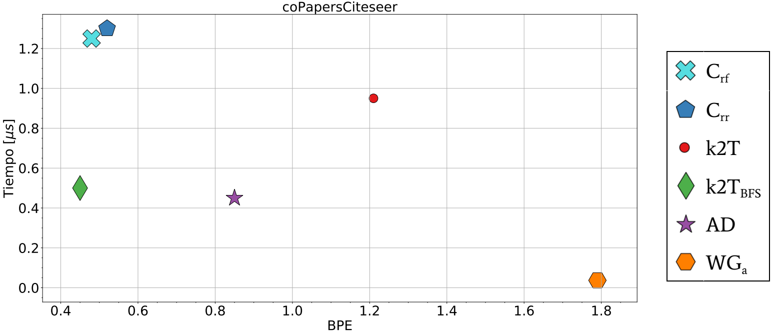


Figura 43: BPE y tiempo de acceso aleatorio en microsegundos de cada algoritmo, para coPapersCiteseer.

Resultados - Comparando con estado del arte

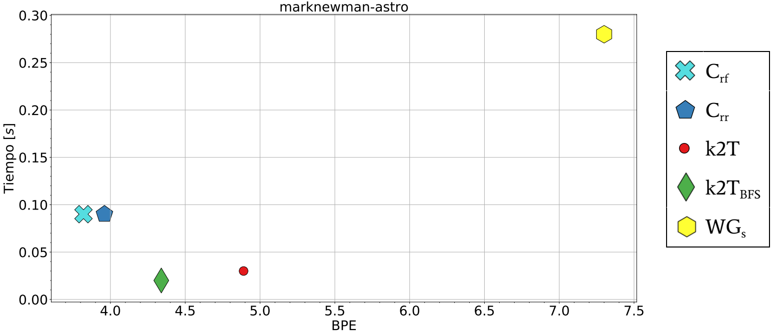


Figura 44: BPE y tiempo de reconstrucción secuencial en segundos de cada algoritmo, para marknewman-astro.

Resultados - Comparando con estado del arte

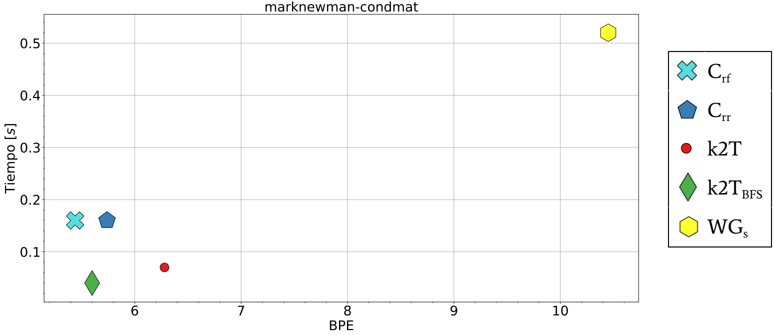


Figura 45: BPE y tiempo de reconstrucción secuencial en segundos de cada algoritmo, para marknewman-condmat.

Resultados - Comparando con estado del arte

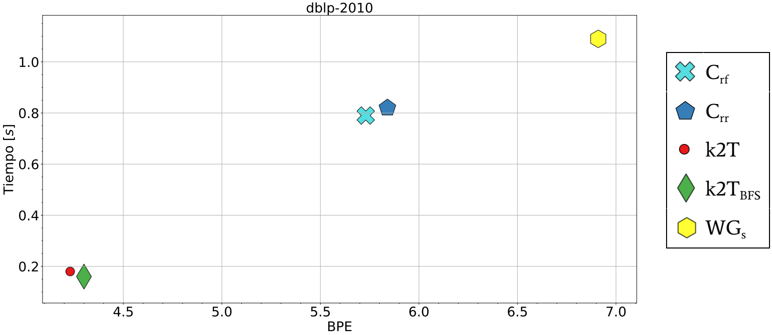


Figura 46: BPE y tiempo de reconstrucción secuencial en segundos de cada algoritmo, para dblp-2010.

Resultados - Comparando con estado del arte

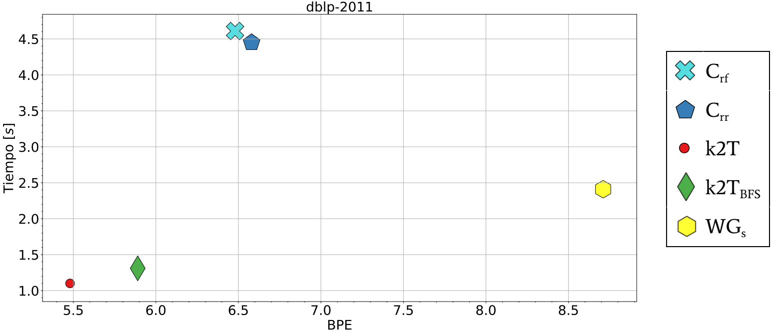


Figura 47: BPE y tiempo de reconstrucción secuencial en segundos de cada algoritmo, para dblp-2011.

Resultados - Comparando con estado del arte

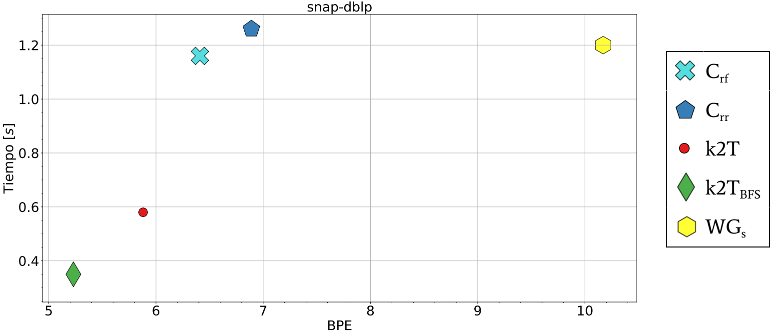


Figura 48: BPE y tiempo de reconstrucción secuencial en segundos de cada algoritmo, para snap-dblp.

Resultados - Comparando con estado del arte

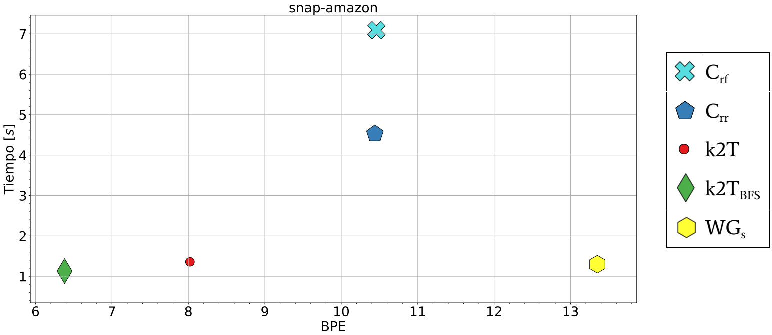


Figura 49: BPE y tiempo de reconstrucción secuencial en segundos de cada algoritmo, para snap-amazon.

Resultados - Comparando con estado del arte

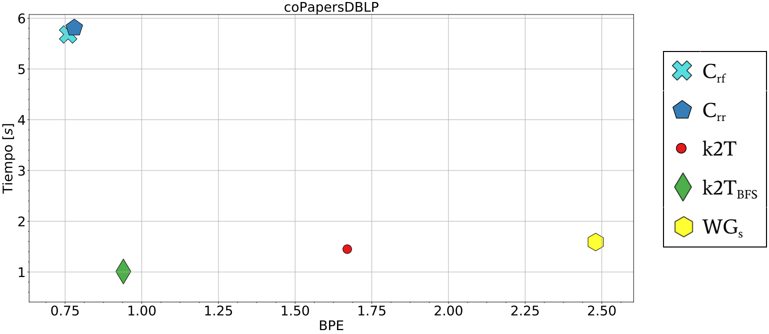


Figura 50: BPE y tiempo de reconstrucción secuencial en segundos de cada algoritmo, para coPapersDBLP.

Resultados - Comparando con estado del arte

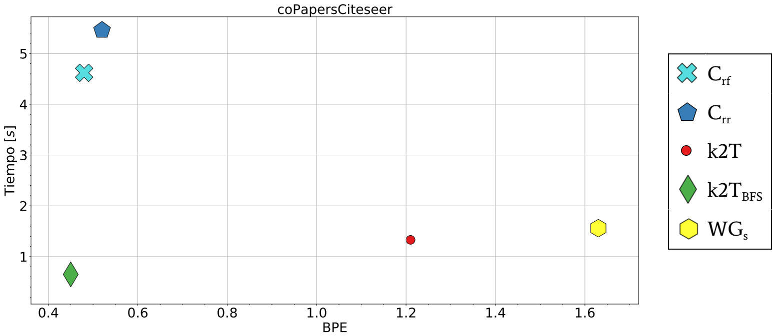


Figura 51: BPE y tiempo de reconstrucción secuencial en segundos de cada algoritmo, para coPapersCiteseer.

The WebGraph Framework

Tabla 8: Representación usando listado de sucesores directo y brechas.

Nodo	Outd.	Sucesores	Usando brechas
...	...	...	...
15	11	13, 15, 16, 17, 18, 19, 23, 24, 203, 315, 1034	3, 1, 0, 0, 0, 0, 3, 0, 178, 111, 718
16	10	15, 16, 17, 22, 23, 24, 315, 316, 317, 3041	1, 0, 0, 4, 0, 0, 290, 0, 0, 2723
17	0		
18	5	13, 15, 16, 17, 50	9, 1, 0, 0, 32
...	...	...	...

Tabla 9: Representación usando copy list.

Nodo	Outd.	Ref.	Copy list	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
...	...	...	...	...

The WebGraph Framework (2)

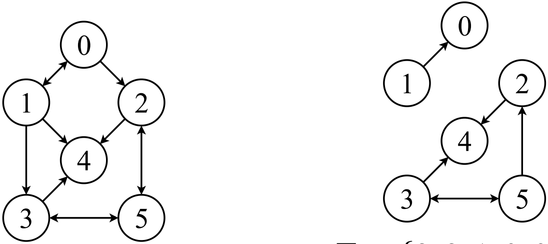
Tabla 10: Representación usando copy blocks.

Nodo	Outd.	Ref.	# b	Copy blocks	Nodos extra
...	...	...	...	...	...
15	11	0			13, 15, 16, 17, 18, 19, 23, 24, 203, 315, 1034
16	10	1	7	0, 0, 2, 1, 1, 0, 0	22, 316, 317, 3041
17	0				
18	5	3	1	4	50
...	...	...	...	...	...

Tabla 11: Representación usando intervalos, con umbral $L_{min} = 2$.

Nodo	Outd.	Ref.	# b	Copy blocks	# inter	Ext. izq.	Largo	Residuales
...	...	...	...	...	...	...	...	...
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
...	...	...	...	...	...	...	...	...

Graph Compression by BFS



(a)

$$T = \{2, 2, 1, 0, 0, 0\}$$

(b)

Figura 52: Ejemplo de Fase 1 de BFS. (a) Índices asignados a los nodos. (b) Aristas restantes después de BFS, junto listado de recorrido T.

Graph Compression by BFS (2)

Tabla 12: Lista de adyacencia para BFS, v_i el primer nodo de un trozo.

Nodo	Grado	Adyacentes
...	...	...
i	8	13, 15, 16, 17, 20, 21, 23, 24
i + 1	9	13, 15, 16, 17, 19, 20, 25, 31, 32
i + 2	0	
i + 3	2	15, 16
...	...	...

Tabla 13: Codificación BFS del listado de adyacencia.

Nodo	Grado	Adyacentes
...	...	...
i	8	$\phi 13$, $\phi 1$, $\phi 0$, $\phi 0$, $\phi 2$, $\phi 0$, $\phi 1$, $\phi 0$
i + 1	9	$\beta 0$, $\beta 0$, $\beta 0$, $\beta 0$, $\chi 0$, $\alpha 0$, $\beta 2$, $\phi 5$, $\phi 0$
i + 2	0	
i + 3	2	$\beta 2$, $\alpha 0$
...	...	...

Graph Compression by BFS (3)

Tabla 14: Ejemplo de redundancias a explotar en listado de adyacencia de BFS.

Grado	Adyacentes									
...	...									
0										
9	$\beta 1,$	$\phi 1,$	$\phi 1,$	$\phi 1,$	$\phi 0,$	$\phi 1,$	$\phi 1,$	$\phi 1,$	$\phi 1,$	
9	$\beta 0,$	$\beta 1,$	$\beta 0,$	$\beta 0,$	$\beta 0,$	$\beta 0,$	$\beta 0,$	$\beta 0,$	$\beta 2,$	
10	$\beta 0,$	$\beta 1,$	$\beta 0,$	$\beta 0,$	$\beta 0,$	$\beta 0,$	$\beta 0,$	$\beta 0,$	$\beta 1,$	$\phi 903$
10	$\beta 0,$	$\beta 1,$	$\beta 0,$	$\beta 0,$	$\beta 0,$	$\beta 0,$	$\beta 0,$	$\beta 0,$	$\beta 223,$	$\phi 900$
10	$\beta 0,$	$\beta 1,$	$\beta 0,$	$\beta 0,$	$\beta 0,$	$\beta 0,$	$\beta 0,$	$\beta 0,$	$\beta 1,$	$\alpha 0$
10	$\beta 0,$	$\beta 1,$	$\beta 0,$	$\beta 0,$	$\beta 0,$	$\beta 0,$	$\beta 0,$	$\beta 0,$	$\beta 1,$	$\beta 0$
10	$\beta 0,$	$\beta 1,$	$\beta 0,$	$\beta 0,$	$\beta 0,$	$\beta 0,$	$\beta 0,$	$\beta 0,$	$\beta 1,$	$\beta 0$
10	$\beta 0,$	$\beta 1,$	$\beta 0,$	$\beta 0,$	$\beta 0,$	$\beta 0,$	$\beta 0,$	$\beta 0,$	$\beta 1,$	$\beta 0$
10	$\beta 0,$	$\beta 1,$	$\beta 0,$	$\beta 0,$	$\beta 0,$	$\beta 0,$	$\beta 0,$	$\beta 0,$	$\beta 1,$	$\beta 0$
10	$\beta 0,$	$\beta 1,$	$\beta 0,$	$\beta 0,$	$\beta 0,$	$\beta 0,$	$\beta 0,$	$\beta 0,$	$\alpha 76,$	$\alpha 232$
9	$\beta 0,$	$\beta 1,$	$\beta 0,$	$\beta 0,$	$\beta 0,$	$\beta 0,$	$\beta 0,$	$\beta 0,$	$\beta 0$	
...	...									

Graph Compression by BFS (4)

Tabla 15: Ejemplo de redundancias codificadas de BFS.

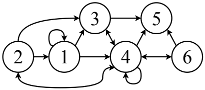
Líneas	Grado	Enlaces			
...	...	...			
0	0				
0	9	$\beta 7$,	$\phi\,\Sigma\,2\;1\;1$,	$\phi 0$,	$\phi\,\Sigma\,2\;1\;2$
0	0	$\beta\,\Sigma\,3\;0\;7\;5$,	$\beta 1$,	$\beta\,\Sigma\,2\;0\;4$,	$\beta 2$
0	1	$\beta 1$,	$\phi 903$		
0	0	$\beta 223$,	$\phi 900$		
0	0	$\beta 1$,	$\alpha 0$		
3	0	$\beta 1$,	$\beta 0$		
0	0	$\alpha 76$,	$\alpha 232$		
0	-1	$\beta 0$			
...	...	...			

Using Re-Pair

Tabla 16: Ejemplo de Re-Pair. Las reglas en la tabla conforman el diccionario asociado a la compresión.

Reglas	String
	$singing.do.wah.diddy.diddy.dum.diddy.do$
$A \rightarrow .d$	$singingAo.wahAiddyAiddyAumAiddyAo$
$B \rightarrow dd$	$singingAo.wahAiByAiByAumAiByAo$
$C \rightarrow Ai$	$singingAo.wahCByCByAumCByAo$
$D \rightarrow By$	$singingAo.wahCDCDAumCDAo$
$E \rightarrow CD$	$singingAo.wahEEAumEAo$
$F \rightarrow in$	$sFgFgAo.wahEEAumEAo$
$G \rightarrow Ao$	$sFgFgG.wahEEAumEG$
$H \rightarrow Fg$	$sHHG.wahEEAumEG$

Using Re-Pair (2)



(a)

B1	111101
B2	1111001

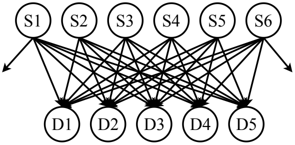
(c)

$T(G)$	-1	1	3	4	-2	1	3	4	-3	4	5	-4	3	4	5	6	-5	-6	4	5
$7(4,5)$	-1	1	3	4	-2	1	3	4	-3	7	-4	3	7	6	-5	-6	7			
$8(1,3)$	-1	8	4	-2	8	4	-3	7	-4	3	7	6	-5	-6	7					
$9(8,4)$	-1	9	-2	9	-3	7	-4	3	7	6	-5	-6	7							
No < 0	9	9	7	3	7	6	7													

(b)

Figura 53: Ejemplo para Re-Pair aplicado a grafos por Claude y Navarro. (a) Grafo de ejemplo. (b) Listado concatenado $T(G)$ y resultado final luego de tres reemplazos y eliminar nodos de referencia. (c) Bitmaps indicadores de nodos de referencia removidos.

Virtual Node Mining



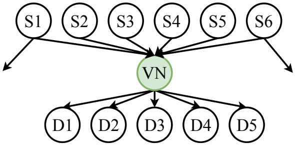
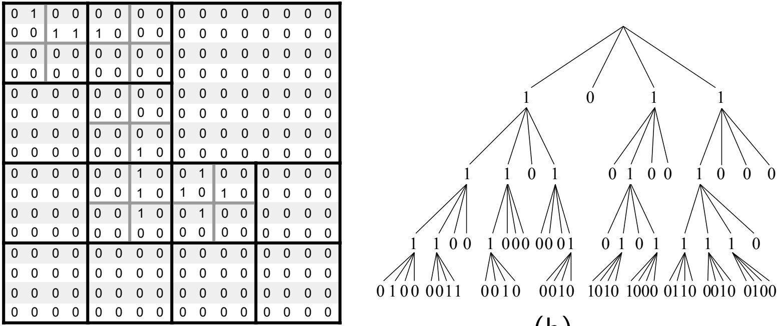


Figura 54: Ejemplo de reemplazo por nodo virtual. (a) Biclique. (b) Biclique con reemplazo de aristas por nodo virtual V.

k2-tree



(b)

(a)

Figura 55: Ejemplo de k2-tree. (a) Matriz de adyacencia. (b) Diagrama de estructura.

Wavelet tree

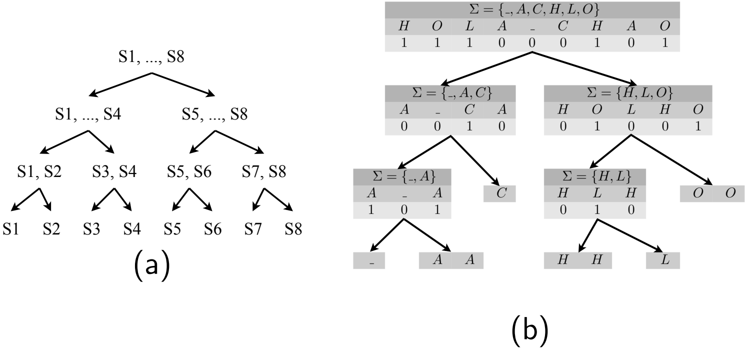


Figura 56: Ejemplos de wavelet-tree. (a) Ejemplo básico de subdivisión de secuencia ordenada. (b) Ejemplo práctico con alfabetos y bitmaps por nodo.

Wavelet matrix

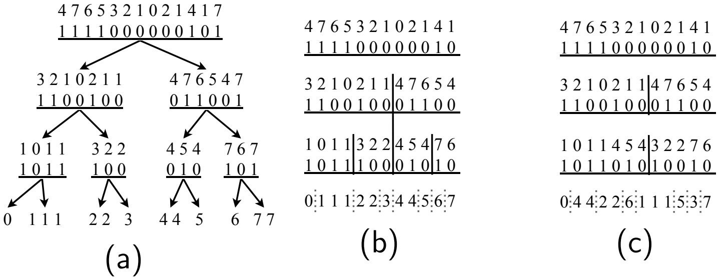


Figura 57: Ejemplos para wavelet-matrix. (a) Un wavelet tree. (b) El mismo wavelet tree sin punteros. (c) La wavelet matrix correpondiente.