CPA. Compressing Graphs

Maximilien Danisch

LIP6 – CNRS and Sorbonne University

first_name.last_name@lip6.fr

Rendu des TMEs

- Mail à envoyer à "maximilien.danisch@lip6.fr".
- Deadline : Mercredi 10 Mars 23h59.
- Le mail contiendra un seul fichier PDF (concaténation des 4 premiers TMEs).
- Un lien vers vos codes sera donné dans le PDF.
- Le PDF fera moins de 2Mo.
- Le non du fichier PDF sera : "nom1_nom2.pdf" pour un travail fait en binôme et "nom.pdf" pour un travail fait seul.
- Le sujet du mail sera : CPA rapport TME.
- Seul le premier envoi sera pris en compte.

NB. Tout écart à ces indications sera sanctionné.

This course is mainly based on

The Webgraph Framework I: Compression Techniques.
Boldi and Vigna, WWW2004.

The Webgraph Framework II: Codes for the World-Wide Web.
Boldi and Vigna, DCC2004.

Why compressing graphs?

Some graphs are LARGE: Twitter, Facebook, the Web graph.

Sometimes too large to fit in the memory of a single commodity machine without compression.

Compress a graph to:

1. Store it in disk and use less disk space
2. Store it in the main memory and be able to carry computations with the compressed structure

Note that using compression can also fasten computations.

Alternatively (not covered in this class): use several machines and distributed algorithms.

Note that most real-world graphs fit in the main-memory of a supercomputer such as the SunwayTaihuLight with 1.31PB of RAM. However such a machine costs around 200M$.

Outline

Graph algorithmics without a graph

A “more efficient” graph representation

Instantaneous variable-length codes

Some improvements

Performance in practice

Outline

Graph algorithmics without a graph

Assume that:

- you have enough memory to store few “values” for each node in the graph
- you do not have enough memory to store all edges in the graph (without compression)

Question: Which operations are needed to implement the following algorithms?

- **BFS:**
- **PageRank:**
- **LabelProp:**
- **CountingTriangles:**

Graph algorithmics without a graph

Assume that:

- you have enough memory to store few “values” for each node in the graph
- you do not have enough memory to store all edges in the graph (without compression)

Question: Which operations are needed to implement the following algorithms?

- **BFS:** iterate over all neighbors of a node
- **PageRank:**
- **LabelProp:**
- **CountingTriangles:**

Graph algorithmics without a graph

Assume that:

- you have enough memory to store few “values” for each node in the graph
- you do not have enough memory to store all edges in the graph (without compression)

Question: Which operations are needed to implement the following algorithms?

- **BFS:** iterate over all neighbors of a node
- **PageRank:** iterate over all edges
- **LabelProp:**
- **CountingTriangles:**

Graph algorithmics without a graph

Assume that:

- you have enough memory to store few “values” for each node in the graph
- you do not have enough memory to store all edges in the graph (without compression)

Question: Which operations are needed to implement the following algorithms?

- **BFS:** iterate over all neighbors of a node
- **PageRank:** iterate over all edges
- **LabelProp:** iterate over all neighbors of a node
- **CountingTriangles:**

Graph algorithmics without a graph

Assume that:

- you have enough memory to store few “values” for each node in the graph
- you do not have enough memory to store all edges in the graph (without compression)

Question: Which operations are needed to implement the following algorithms?

- **BFS:** iterate over all neighbors of a node
- **PageRank:** iterate over all edges
- **LabelProp:** iterate over all neighbors of a node
- **CountingTriangles:** iterate over all neighbors of a node

Graph algorithmics without a graph

Need a compressed graph structure, that allows to

- Iterate over all edges and/or
- Iterate over all neighbors of a node and/or
- Check whether two nodes are adjacent.

Outline

“Naive” representation

Node	d^{out}	Successors
...	...	...
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
...	...	...

Table: “Naive” representation using out-degree and adjacency lists

Question: What can you notice?

Locality and similarity

I can notice that there is some **locality** and **similarity**!

Locality: The gaps between a node and its successors is small and the gaps between the successors of a node is small.

Similarity: The set of neighbors of two proximal nodes tend to be similar.

Question: Think about the Web graph: “urls pointing to urls”. How can you number the urls to get some locality/similarity?

Locality and similarity

Locality: Many links are intra-domain, and therefore likely to point to pages nearby in the lexicographic order.

Similarity: Pages that are proximal in the lexicographic ordering tend to have similar sets of neighbors.

Sort the pages according to the lexicographic order and number them (from 0 to $n-1$) according to the obtained order.

“Naive” representation

Node	d^{out}	Successors
...	...	...
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
...	...	...

Table: “Naive” representation using out-degree and adjacency lists

Question: What do we do now?

Think about the node 1,000,000,000 connected to the nodes: 1,000,000,002, 1,000,000,003, 1,000,000,004, 1,000,000,006, 100,000,007 and 1,000,000,009

Representation using gaps

Node	d^{out}	Successors
...	...	...
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
...	...	...

Table: “Naive” representation using out-degree and adjacency lists

Node	d^{out}	Successors
...	...	...
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
...	...	...

Table: Representation using gaps

Representation using copy lists

Node	d^{out}	Successors
...	...	...
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
...	...	...

Table: "Naive" representation using out-degree and adjacency lists

Node	d^{out}	Ref.	Copy list	Extra nodes
...	...	...	...	...
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
...	...	...	...	...

Table: Representation using copy lists

Representation using copy blocks

Node	d^{out}	Ref.	Copy list	Extra nodes
...	...	...	...	...
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
...	...	...	...	...

Table: Representation using copy lists

Node	d^{out}	Ref.	#blocks	Sizes	Extra nodes
...	...	...	...	...	...
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
...	...	...	...	...	...

Table: Representation using copy blocks

Final representation using intervals

Node	d^{out}	Ref.	#blocks	Sizes	Extra nodes
...	...	...	...	...	...
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
...	...	...	...	...	...

Table: Representation using copy blocks

Node	d^{out}	Ref.	#blocks	Sizes	#inter.	left	len.	Residuals
...	...	...	...	...	...	...	...	...
15	11	0			2	0,2	3,0	3,189,111,718
16	10	1	7	0,0,2,1,1,0,0	1	600	0	12,3018
17	0							
18	5	3	1	4	0			64
...	...	...	...	...	...	...	...	...

Table: Representation using intervals (interval threshold is 2)

Summary of the final format

$$d \left[\overbrace{r \left[b\ B_1 \cdots B_b \right]_{r>0}}^{W>0} \left[\overbrace{i\ E_1 L_1 \cdots E_i L_i}^{L_{\min}<\infty}\ R_1\ \cdots\ R_k \right]_{\beta<d} \right]_{d>0}$$

Datum	Meaning	Notes	Represented as
d	out-degree	$d \geq 0$	
r	reference	$0 \leq r \leq W$	
b	# blocks	$b \geq 0$	
$B_1, \ldots, B_b$	blocks	$B_1 \geq 0, B_2, \ldots, B_b > 0$	$B_1, B_2 - 1, \ldots, B_b - 1$
i	# interval	$i \geq 0$	
$E_1, \ldots, E_i$	left extremes	$E_{k+1} \geq E_k + L_k + 1$	$\nu(E_1 - x), E_2 - E_1 - L_1 - 1, \ldots$
$L_1, \ldots, L_i$	interval length	$L_1, \ldots, L_i \geq L_{min}$	$L_1 - L_{min}, \ldots, L_i - L_{min}$
$R_1, \ldots, R_k$	residuals	$0 \leq R_1 < R_2 < \ldots < R_k$	$\nu(R_1 - x), R_2 - R_1 - 1, \ldots$

- W is a parameter: the maximum gap to a reference node.
- L_{min} is a parameter: the minimal length of an interval.
- β is the number of successors that have been copied from the reference list
- $\nu(x) = 2 \cdot x$ if $x \geq 0$, $\nu(x) = 2 \cdot |x| - 1$ if $x < 0$

Outline

Graph algorithmics without a graph

A “more efficient” graph representation

Instantaneous variable-length codes

Some improvements

Performance in practice

Instantaneous variable-length codes

x = positive integer, b = its binary representation, l = length(b)

- **Unary.** write $x - 1$ zeros and a one.
- γ**-coding.** $l - 1$ in unary followed by the last $l - 1$ digits of b
- δ**-coding.** Write l in γ coding followed by the last $l - 1$ digits of b
- **nibble coding.** Add zeros on the left of b so that l is a multiple of 3. Break b in blocks of 3 bits and prefix each block with a bit: 0 for all blocks except for the last one.
- ζ_k**-coding.** h in unary such that $x \in [\![2^{hk}, 2^{(h+1)k} - 1]\!]$ followed by a minimal binary coding of $x - 2^{hk}$ in the interval $[\![0, 2^{(h+1)k} - 2^{hk} - 1]\!]$.

$\neq$ `https://en.wikipedia.org/wiki/Huffman_coding`

Question: Write 10 using each instantaneous code.

Instantaneous variable-length codes

Integer	$\gamma = \zeta_1$	ζ_2	ζ_3	ζ_4	δ	nibble
1	1	10	100	1000	1	1000
2	010	110	1010	10010	0100	1001
3	011	111	1011	10011	0101	1010
4	00100	01000	1100	10100	01100	1011
5	00101	01001	1101	10101	01101	1100
6	00110	01010	1110	10110	01110	1101
7	00111	01011	1111	10111	01111	1110
8	0001000	011000	0100000	11000	00100000	1111
9	0001001	011001	0100001	11001	00100001	00011000
10	0001010	011010	0100010	11010	00100010	00011001
11	0001011	011011	0100011	11011	00100011	00011010
12	0001100	011100	0100100	11100	00100100	00011011
13	0001101	011101	0100101	11101	00100101	00011100
14	0001110	011110	0100110	11110	00100110	00011101
15	0001111	011111	0100111	11111	00100111	00011110
16	000010000	00100000	01010000	010000111	001011001	10000111

Coding in practice

- /** The coding for outdegrees. By default, we use gamma coding. */
- /** The coding for copy-block lists. By default, we use gamma coding. */
- /** The coding for residuals. By default, we use zeta coding. */
- /** The coding for references. By default, we use unary coding. */
- /** The coding for block counts. By default, we use gamma coding. */

Outline

Some improvements

Question: What if there is no “natural” ordering?

Some improvements

Question: What if there is no “natural” ordering?

Question: Suggest some ordering of the nodes.

Some improvements

Question: What if there is no “natural” ordering?

Question: Suggest some ordering of the nodes.

Question: Suggest some optimization to find a good ordering.

Some improvements

Question: What if there is no “natural” ordering?

Question: Suggest some ordering of the nodes.

Question: Suggest some optimization to find a good ordering.

For more:

- On Compressing Social Networks.
  Chierichetti et al., KDD2009.
- Layered Label Propagation: A MultiResolution Coordinate-Free Ordering for Compressing Social Networks.
  Boldi et al. WWW2011.
- Compressing Graphs and Indexes with Recursive Graph Bisection.
  Dhulipala et al., KDD2016.

Outline

Graph algorithmics without a graph

A “more efficient” graph representation

Instantaneous variable-length codes

Some improvements

Performance in practice

Performance in practice

Question: Given a graph with n nodes and m directed edges, what is:

$$\log_2 \left(\binom{n^2}{m} \right) ?$$

Performance in practice

Question: Given a graph with n nodes and m directed edges, what is:

$$\log_2 \left(\binom{n^2}{m} \right) ?$$

- $\binom{n^2}{m}$ is the number of graphs with n nodes and m directed edges.

Performance in practice

Question: Given a graph with n nodes and m directed edges, what is:

$$\log_2\left(\binom{n^2}{m}\right)?$$

- $\binom{n^2}{m}$ is the number of graphs with n nodes and m directed edges.
- $\lceil \log_2(\binom{n^2}{m}) \rceil$ is the number of bits you need to encode any graph with n nodes and m directed edges (using a bitword of fixed length).

Performance in practice

Question: Given a graph with n nodes and m directed edges, what is:

$$\log_2\left(\binom{n^2}{m}\right)?$$

- $\binom{n^2}{m}$ is the number of graphs with n nodes and m directed edges.
- $\lceil \log_2(\binom{n^2}{m}) \rceil$ is the number of bits you need to encode any graph with n nodes and m directed edges (using a bitword of fixed length).

In practice (for large sparse real-world graphs), we have:

$$\text{“memory naive adjacency list”} > \log_2\left(\binom{n^2}{m}\right) > \text{“memory BV”}$$

Performance in practice

Question: How many bits per link are needed to encode a graph with n nodes using the “naive” adjacency list format?

Performance in practice

Question: How many bits per link are needed to encode a graph with n nodes using the “naive” adjacency list format?

- Around $\log_2(n)$
- If $n = 1G$ then 30 bits per link are needed ($\rightarrow 32$).
- If $n = 100G$ then 37 bits per link are needed ($\rightarrow 40 \rightarrow 64$).

- BV can encode a Web graph using around 2 bits per link.
- BV can encode a social network using around 10 bits per link.
- The performance of BV depends on (i) the ordering and on (ii) the structure of the graph.

Performance in practice

- `http://law.di.unimi.it/datasets.php`
- Live Demo

Practical (optional)

Use the BV framework `http://webgraph.di.unimi.it` to implement an algorithm of your choice among the ones seen in class.
Make your program scale to graphs with several billions of edges on your laptop.