Compressing Inverted Indexes with Recursive Graph Bisection: A Reproducibility Study

Joel Mackenzie[1] Antonio Mallia[2] Matthias Petri[3]
J. Shane Culpepper[1] Torsten Suel[2]

[1]RMIT University, Melbourne, Australia

[2]New York University, New York, USA

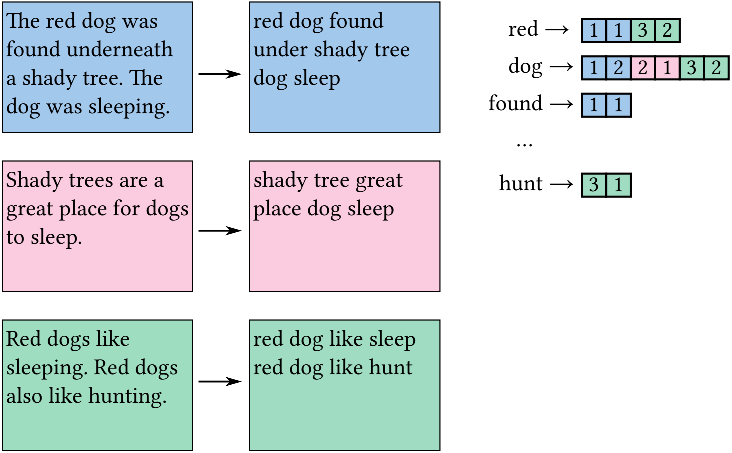
[3]The University of Melbourne, Melbourne, Australia

April, 2019

Overview

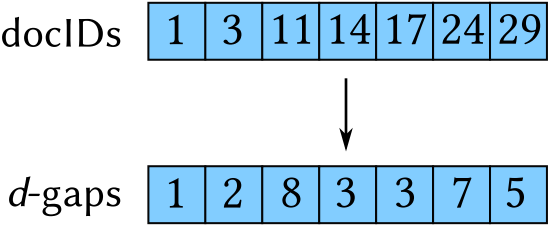
Overview: Text Indexing

- Documents can be efficiently represented in an *inverted index* as a list of *postings*.



Overview: Postings Lists

- A postings list L_t for a term t contains a monotonically increasing list of document identifiers, represented as *delta* gaps, with a corresponding list of term frequencies (stored seperately).



Motivation

Motivation

- The space consumption of a postings list can be reduced if the size of the *deltas* (d-gaps) can be reduced.
  - Compressors are more effective at compressing smaller integers.

- Reducing these d-gaps can be achieved by *reordering* the space of document identifiers.

- Given a collection of documents D with $n = |D|$, an arrangement of document identifiers can be defined as a bijection:

$$\pi : D \to \{1, 2, \ldots, n\},$$

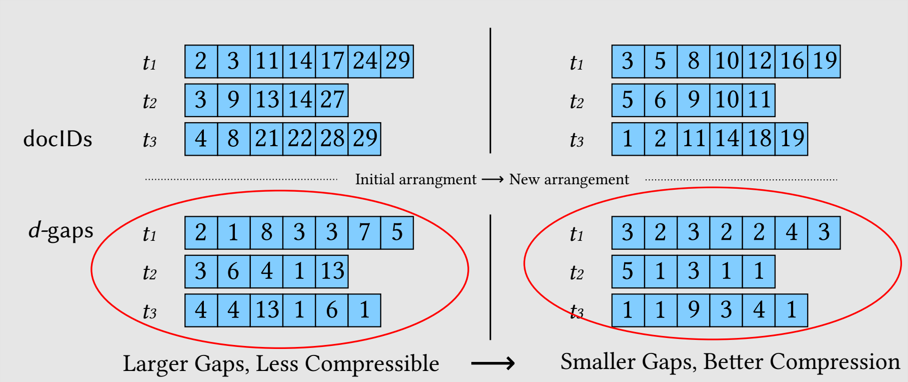
where document d_i is mapped to identifier $\pi(d_i)$.

A Basic Example

		Initial arrangment	New arrangement
docIDs	t_1	2, 3, 11, 14, 17, 24, 29	3, 5, 8, 10, 12, 16, 19
	t_2	3, 9, 13, 14, 27	5, 6, 9, 10, 11
	t_3	4, 8, 21, 22, 28, 29	1, 2, 11, 14, 18, 19
d-gaps	t_1	2, 1, 8, 3, 3, 7, 5	3, 2, 3, 2, 2, 4, 3
	t_2	3, 6, 4, 1, 13	5, 1, 3, 1, 1
	t_3	4, 4, 13, 1, 6, 1	1, 1, 9, 3, 4, 1

Initial arrangment ⟶ New arrangement

A Basic Example



Agenda: Reproducibility

- The current state-of-the-art in graph/index reordering is proposed in a KDD paper from 2016.[1]
- Given that most authors are from Facebook, the primary focus of this work was compressing graphs.
- No implementation was made available. Can we reproduce, from scratch, the results found in their original work?

[1] L. Dhulipala et al. Compressing Graphs and Indexes with Recursive Graph Bisection. In KDD, 2016.

Baselines

Random Ordering

- Randomly assign a unique identifier in $\{1, 2, \ldots, n\}$ to each document.
- Arrangements are poor due to lack of clustering - larger *d*-gaps.
- Used as a yardstick for comparison, not used in practice.

Natural Orderings

- Assign identifiers in the order that is natural to the collection.
- Crawl ordering is generally the *default* ordering of a text collection, as the crawler will assign identifiers as new documents are indexed.
- Crawl order effectiveness can depend on the method of crawling.
- URL ordering is usually very effective for document collections.
  - Implicit localized clustering of similar documents.

URL Ordering

docID		URL
1	→	abc.com/a
2	→	xyz.com/
3	→	xyz.com/index
4	→	zzz.com/wake_up
5	→	hello.edu/
6	→	abc.com/b_and_c
7	→	xyz.com/products
8	→	hello.edu/programs/cs_101

$\xrightarrow{\pi}$

docID		URL
1	→	abc.com/a
2	→	abc.com/b_and_c
3	→	hello.edu/
4	→	hello.edu/programs/cs_101
5	→	xyz.com/
6	→	xyz.com/index
7	→	xyz.com/products
8	→	zzz.com/wake_up

Minhash Ordering

- Minhash is an algorithm that approximates the *Jaccard* similarity of documents.
- This means similar documents are clustered together, resulting in smaller d-gaps and improved compression.
  - This works under the same assumption as URL ordering.
- Minhash requires k different hash functions, $h_1(x), h_2(x), \ldots, h_k(x)$.

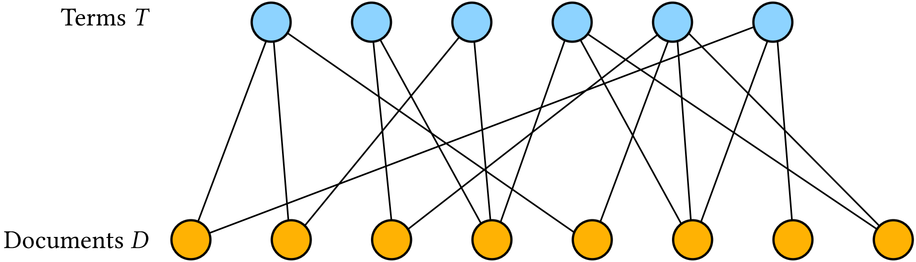
Preliminaries

Preliminaries

- Previous approaches look at *implicitly* clustering similar documents together through some heuristic.
  - Use the URL of a document as a proxy for its content.
  - Approximate Jaccard distances of document content.
- Instead, why not directly optimize this goal?

Preliminaries: Graph theory framework

- Consider our document index as a graph $G = (V, E)$ with $m = |E|$.
  - V is a disjoint set of *terms*, T, and *documents*, D.
  - Each edge $e \in E$ corresponds to an arc (t, d) - term t is contained in document d.
  - Therefore, m is the number of postings in the collection.



Preliminaries: BiMLogA

- Bipartite Minimum Logarithmic Arrangement (BiMLogA)[1]
- NP-Hard.[2]
- Requires a bipartite graph, but can capture non-bipartite graphs via transformation.

Find an arrangement $\pi : D \to \{1, 2, \ldots, n\}$ according to:

$$\underset{\pi}{\operatorname{argmin}} \sum_{t \in T} \sum_{i=0}^{d_t} \log_2(\pi(u_{i+1}) - \pi(u_i)),$$

where d_t is the degree of vertex t, t has neighbours $\{u_1, u_2, \ldots, u_{d_q}\}$ with $\pi(u_1) < \pi(u_2) < \cdots < \pi(u_{d_q})$, and $u_0 = 0$.

[1] F. Chierichetti et al. On compressing social networks. In KDD, 2009.
[2] L. Dhulipala et al. Compressing Graphs and Indexes with Recursive Graph Bisection. In KDD, 2016.

BIMLOGA visualized

t_1	3	5	8	10	12	16	19	24	34	67	90
t_2	5	6	9	10	11	19	33	35	77	81	
$\ldots$											
t_z	8	9	41	50	62	70					

BiMLogA visualized

cost = $\log_2(5 - 3)$

t_1	3	5	8	10	12	16	19	24	34	67	90
t_2	5	6	9	10	11	19	33	35	77	81	

. . .

t_z	8	9	41	50	62	70

BɪMLᴏɢA visualized

$$\text{cost} = \log_2(5 - 3) + \log_2(8 - 5)$$

t_1	3	5	8	10	12	16	19	24	34	67	90
t_2	5	6	9	10	11	19	33	35	77	81	

...

t_z	8	9	41	50	62	70

BIMLOGA visualized

$$\text{cost} = \log_2(5 - 3) + \log_2(8 - 5) + \ldots + \log_2(70 - 62)$$

t_1	3	5	8	10	12	16	19	24	34	67	90
t_2	5	6	9	10	11	19	33	35	77	81	
. . .											
t_z	8	9	41	50	62	70					

Solutions to BiMLogA

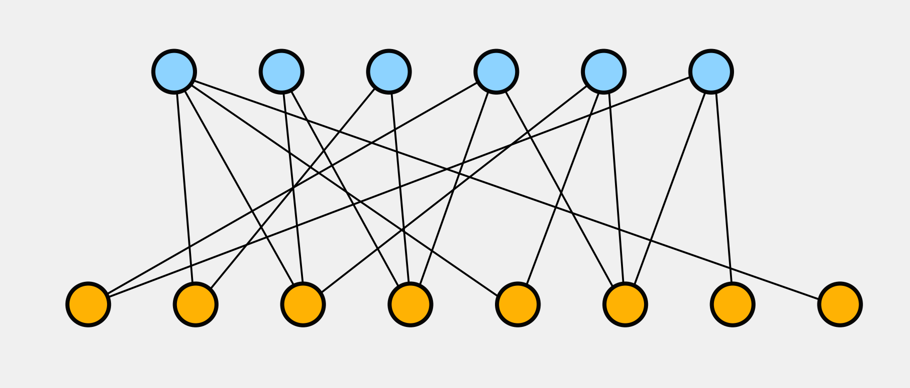
- BiMLogA is *directly* optimizing the space required to store *d*-gaps.
- We call the cost of a solution to BiMLogA the *LogGap* cost.
- NP-Hard, so we must approximate: how to do so practically?

Recursive Graph Bisection

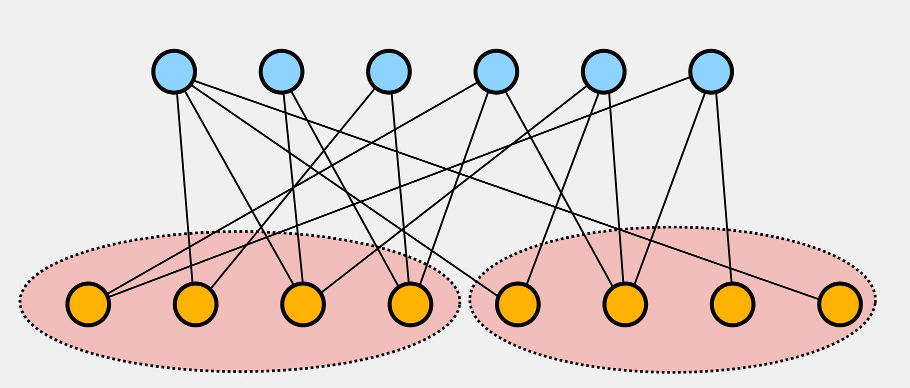
Recursive Graph Bisection (BP)

- We split our input graph into two subgraphs, D_1 and D_2.
- For each *document* $d \in D$, we compute the change in our LogGap cost if we moved d from D_1 to D_2 (or vice versa).
- We sort these gains from high to low, and then while we continue to yield positive gains, we *swap* pairs of documents.
- This process happens a constant number of times, or can be terminated early if no swaps occur.
- Until we reach our maximum depth, we recursively run the same procedure on D_1 and D_2.

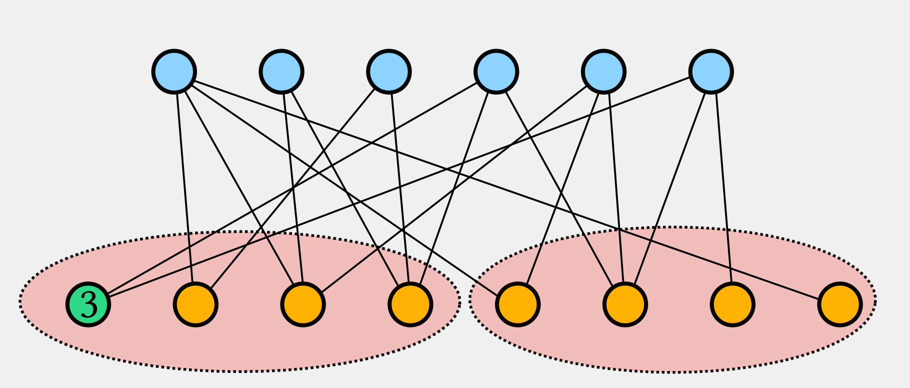
Recursive Graph Bisection: Local Optimization



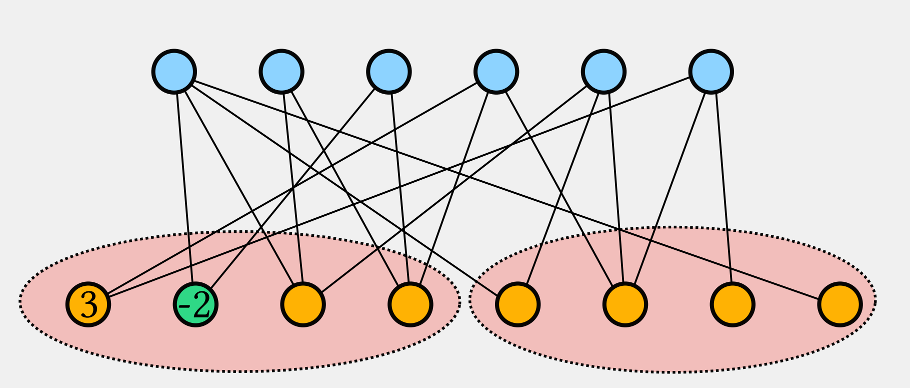
Recursive Graph Bisection: Local Optimization



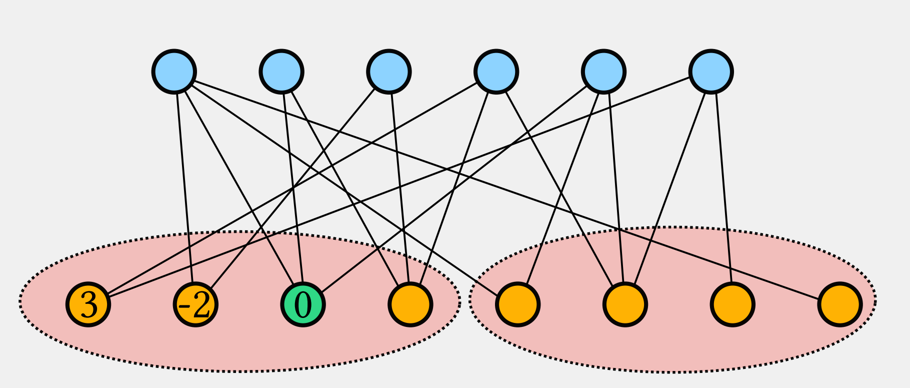
Recursive Graph Bisection: Local Optimization



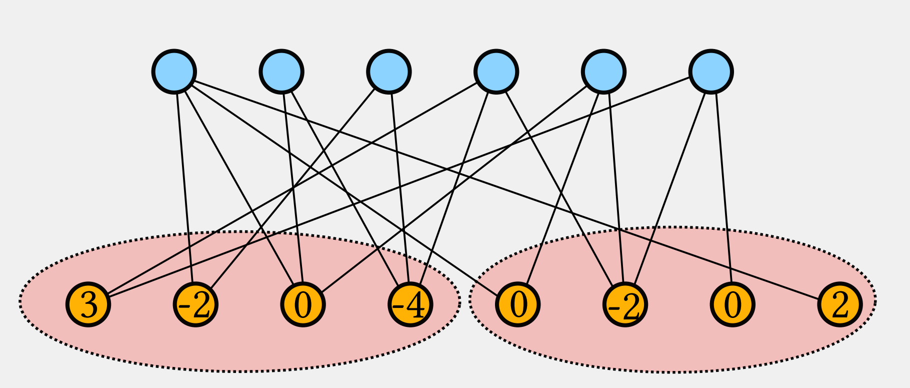
Recursive Graph Bisection: Local Optimization



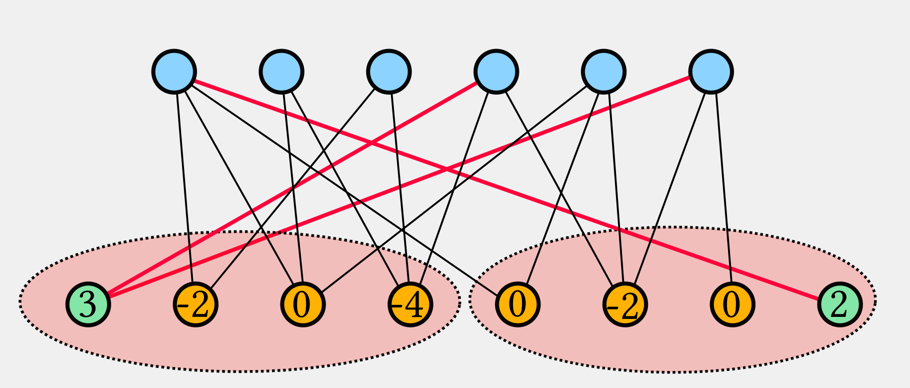
Recursive Graph Bisection: Local Optimization



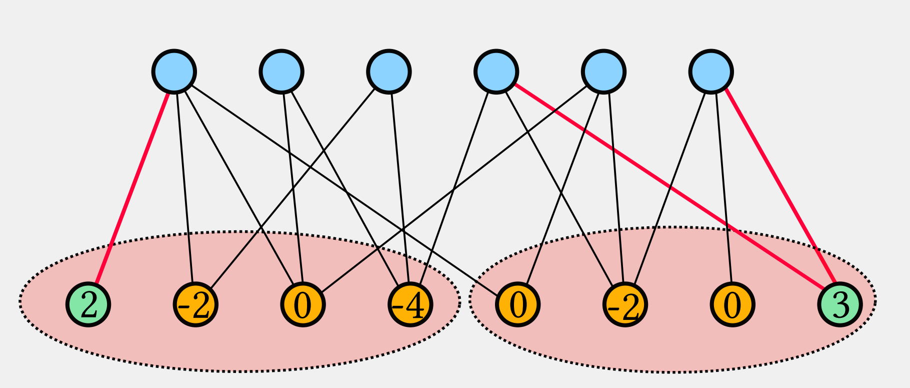
Recursive Graph Bisection: Local Optimization



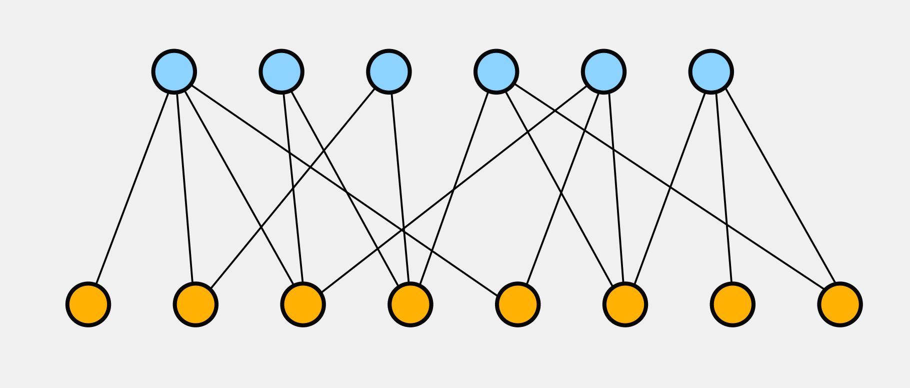
Recursive Graph Bisection: Local Optimization



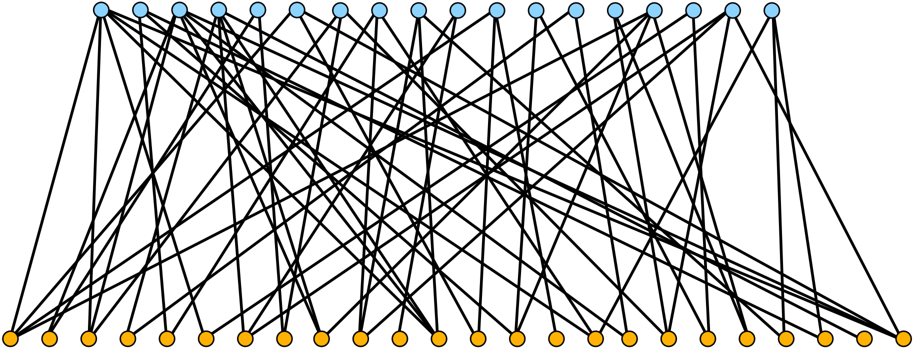
Recursive Graph Bisection: Local Optimization



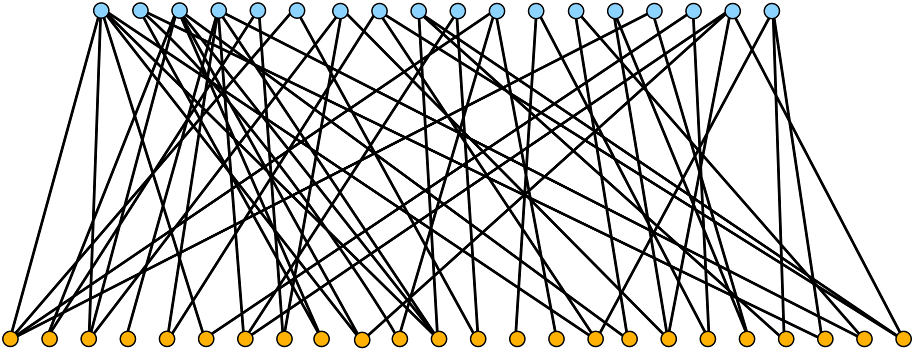
Recursive Graph Bisection: Local Optimization



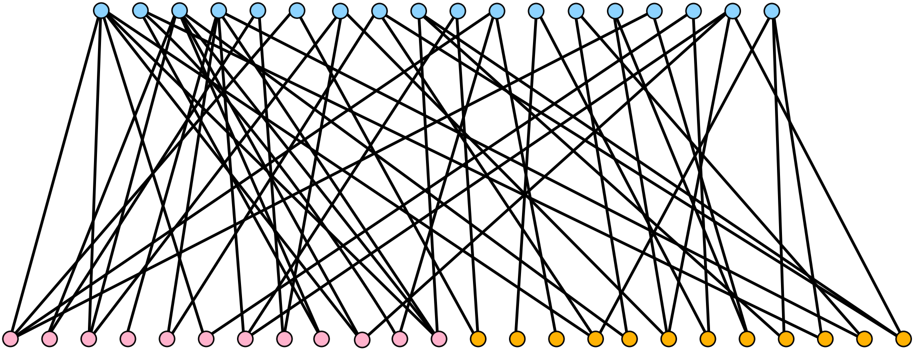
Recursive Graph Bisection: Sketch



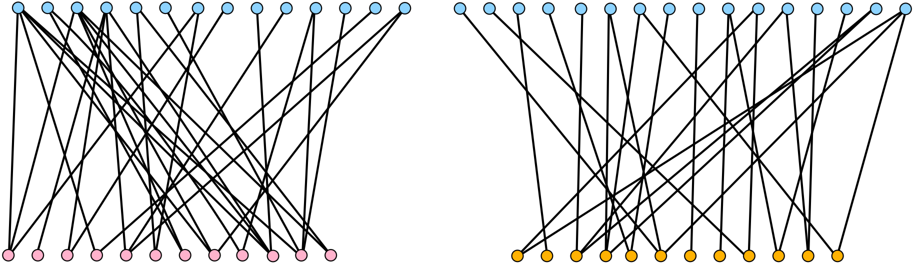
Recursive Graph Bisection: Sketch



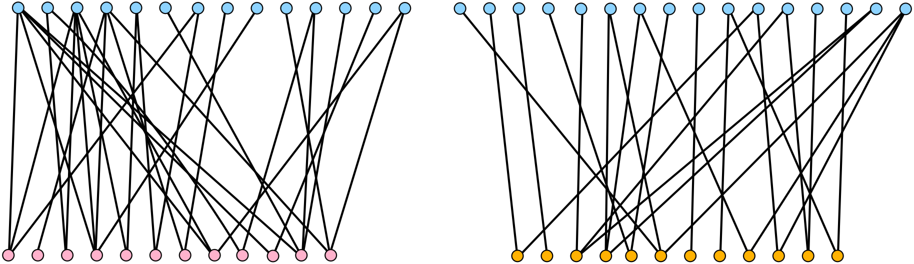
Recursive Graph Bisection: Sketch



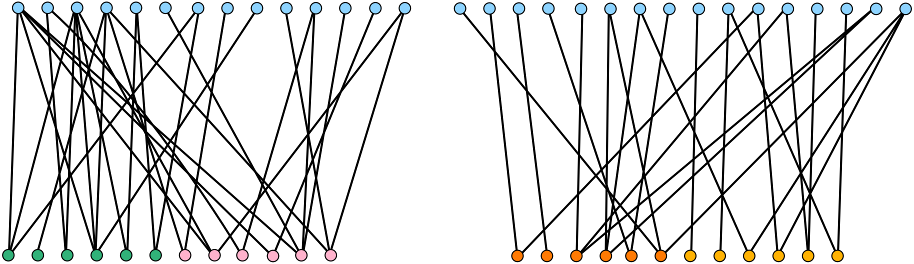
Recursive Graph Bisection: Sketch



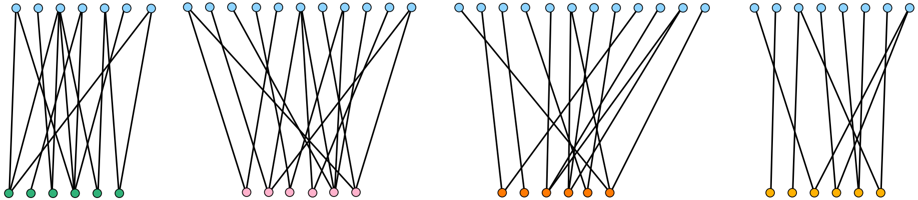
Recursive Graph Bisection: Sketch



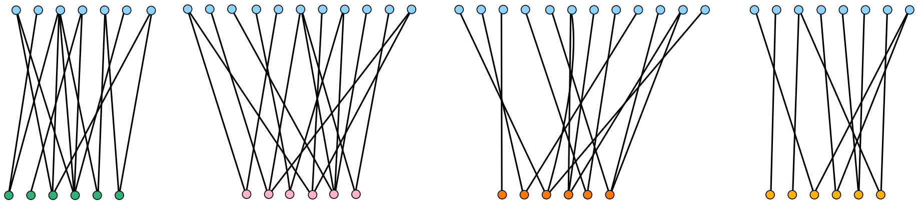
Recursive Graph Bisection: Sketch



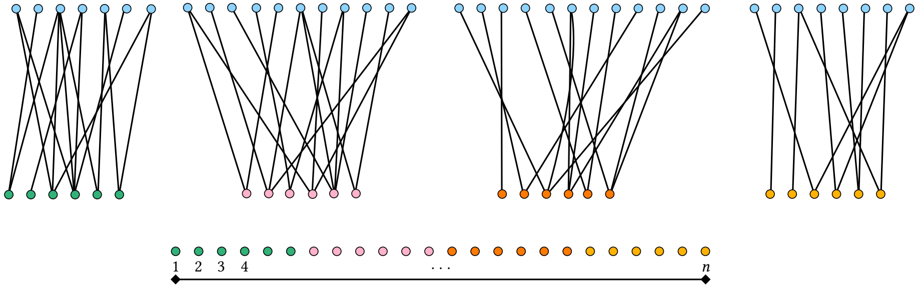
Recursive Graph Bisection: Sketch



Recursive Graph Bisection: Sketch



Recursive Graph Bisection: Sketch



Efficient Implementation

- *Swaps* are just pointer swaps and *references* are used where possible to avoid any move or copy operations.
- A global thread-pool is allocated once, and jobs are allocated according to the Intel Thread Building Blocks scheduling policy.
  - Each recursive call is independent.
  - Computing term degrees and sort operations.
- SIMD intrinsics used when computing term costs, allowing four values to be computed per CPU cycle.
- Branch prediction information to avoid pipeline stalls due to branch misprediction.
- Precompute and store values of $\log_2(x)$ for all $x \leq 4{,}096$.

Experiments

Experimental Differences

- Collections: Parsing, Stemming, Stopping...
  - Collections are somewhat but not completely comparable.
- Hardware: Facebook vs ours.
  - CPU: Intel Xeon E5-2660 vs Intel Xeon Gold 6144.
    - Speed: 2.20GHz vs 3.50GHz.
    - Cores: 32 vs 32.
    - Cache (L2): 2MiB vs 8MiB.
    - Cache (L3): 20MiB vs 24.75MiB.
  - RAM: 128GiB vs 512GiB.
- Emphasis on *Reproducibility*, not *Repeatability.*
  - Different group, different codebase, different collections.

Collections: Full

Graph	$\|D\|$	$\|T\|$	$\|E\|$
NYT	1,855,658	2,970,013	501,568,918
Wikipedia	5,652,893	5,604,981	837,439,129
Gov2	25,205,179	39,180,840	5,880,709,591
ClueWeb09-B	50,220,423	90,471,982	16,253,057,031
ClueWeb12-B	52,343,021	165,309,501	15,319,871,265
CC-News	43,530,315	43,844,574	20,150,335,440

Collections: terms in $\geq$ 4,096 documents

Graph	$\|D\|$	$\|T\|$	$\|E\|$
NYT	1,855,658	10,191	457,883,999
Wikipedia	5,652,893	14,038	749,069,767
Gov2	25,205,179	42,842	5,406,607,172
ClueWeb09-B	50,220,423	101,676	15,237,650,447
ClueWeb12-B	52,343,021	88,741	14,130,264,013
CC-News	43,530,315	76,488	19,691,656,440

Compression Effectiveness

Index	Algorithm	LogGap	PEF	BIC
NYT	Random	3.79	6.36 / 2.22	6.48 / 2.16
	Natural	3.50	6.31 / 2.20	6.23 / 2.13
	Minhash	3.18	5.91 / 2.19	5.79 / 2.11
	BP	2.61	5.24 / 2.13	5.06 / 2.04
Wikipedia	Random	5.12	8.03 / 2.20	8.01 / 1.98
	Natural	4.76	7.83 / 2.17	7.65 / 1.93
	Minhash	3.94	7.08 / 2.11	6.71 / 1.85
	BP	3.13	6.17 / 2.03	5.74 / 1.77
Gov2	Random	5.05	7.96 / 2.97	7.93 / 2.53
	Natural	1.91	4.37 / 2.31	4.01 / 2.07
	Minhash	1.99	4.57 / 2.34	4.17 / 2.10
	BP	1.54	3.67 / 2.20	3.41 / 2.01

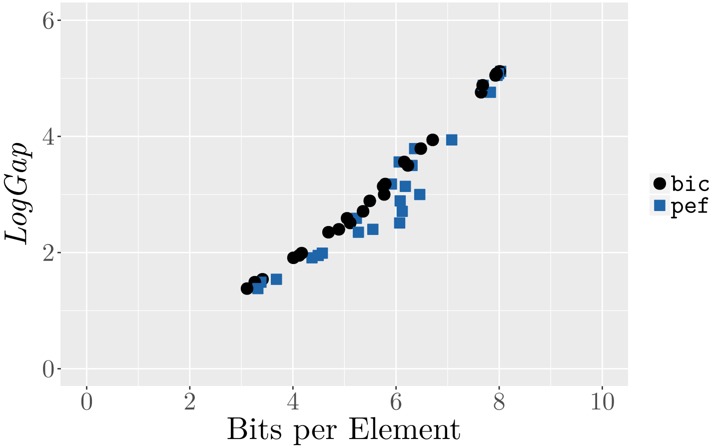
BIC → A. Moffat and L. Stuiver: Binary Interpolative Coding for Effective Compression. Infr. Retr. 3(1), 2000.
PEF → G. Ottaviano and R. Venturini: Partitioned Elias-Fano Indexes. In SIGIR, 2014.

Compression Effectiveness

Index	Algorithm	LogGap	PEF	BIC
ClueWeb09-B	`Random`	4.88	7.69 / 2.39	7.68 / 2.08
	`Natural`	2.71	6.12 / 2.20	5.36 / 1.84
	`Minhash`	3.00	6.46 / 2.23	5.77 / 1.87
	`BP`	2.38	5.49 / 2.12	4.84 / 1.79
ClueWeb12-B	`Random`	5.08	7.99 / 2.39	7.95 / 2.09
	`Natural`	2.51	6.07 / 2.20	5.11 / 1.81
	`Minhash`	2.89	6.08 / 2.17	5.49 / 1.86
	`BP`	2.32	5.20 / 2.07	4.64 / 1.77
CC-News	`Random`	3.56	6.06 / 2.19	6.16 / 2.06
	`Natural`	1.49	3.38 / 1.91	3.26 / 1.73
	`Minhash`	1.95	4.49 / 2.02	4.12 / 1.82
	`BP`	1.39	3.31 / 1.90	3.11 / 1.72

BIC $\longrightarrow$ A. Moffat and L. Stuiver: Binary Interpolative Coding for Effective Compression. Infr. Retr. 3(1), 2000.
PEF $\longrightarrow$ G. Ottaviano and R. Venturini: Partitioned Elias-Fano Indexes. In SIGIR, 2014.

LogGap vs True Cost

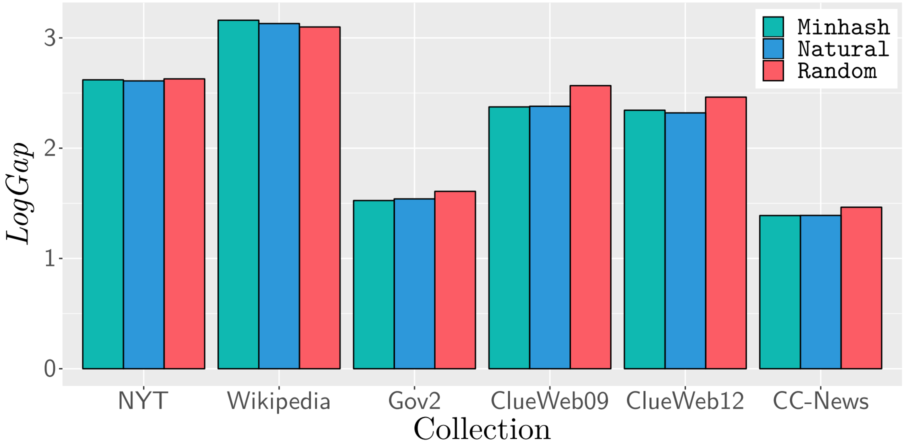


Time to generate a BP arrangement

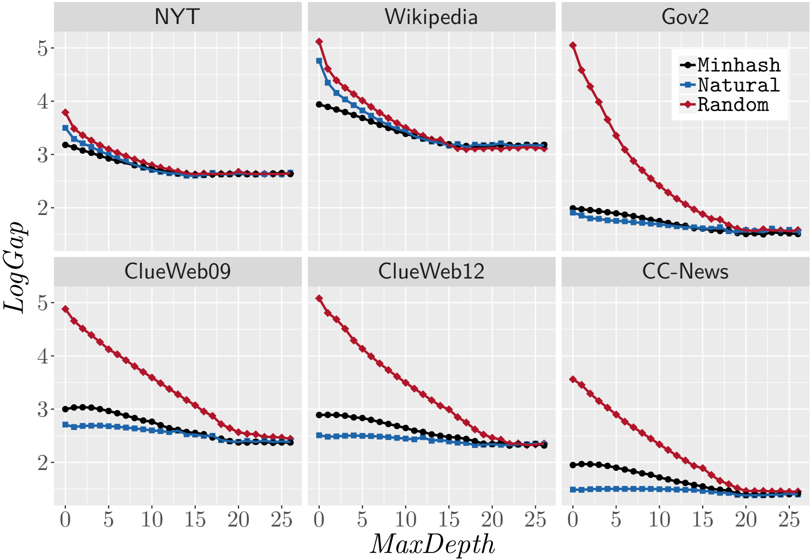
NYT	Wikipedia	Gov2	ClueWeb09-B	ClueWeb12-B	CC-News
2	5	28	90	86	97

- Time taken to process each dataset with recursive graph bisection, in minutes.
- Assumes the input is a VarintGB compressed forward index.
- Uses up to 32 threads, processing entirely in-memory.
- Comparison: Facebook processes Gov2 in 29 minutes, and ClueWeb09-B in 129 minutes.

Sensitivity to input order



Sensitivity to recursion depth



Implications

- Well ordered indexes improve space occupancy.
  - Independent of the compression scheme (well, the most commonly used ones).
  - Lossless and free (apart from computing the ordering).

- Side effect: well ordered indexes improve query time efficiency.[1,2,3,4]
  - Higher throughput.
  - Reduced running costs.

[1] S. Ding and T. Suel: Faster Top-*k* Document Retrieval using Block-Max Indexes: In SIGIR, 2011.
[2] D. Hawking and T. Jones: Reordering an Index to Speed Query Processing without Loss of Effectiveness: In ADCS, 2012.
[3] A. Kane and F. Wm. Tompa: Split-Lists and Initial Thresholds for WAND-Based Search. In SIGIR, 2018.
[4] A. Mallia, M. Siedlaczek, and T. Suel: An Experimental Study of Index Compression and DAAT Query Processing Methods. In ECIR, 2019.

Challenges and Summary

- No codebase - design from ground up.
  - Took many attempts and rounds of analysis to make things efficient.
- Pseudocode in original paper does not shed much light on implementation details.
- Successful in reproducing the original work and extending this analysis to new text collections.

Questions and Acknowledgements

- We thank the authors of the original paper for helpful discussions regarding the nuances of their algorithm.

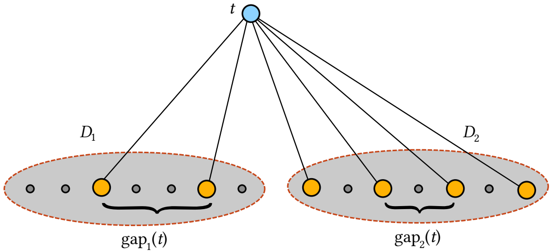
- Codebase:
  - `https://github.com/pisa-engine/pisa`
  - `https://github.com/pisa-engine/ecir19-bisection`

- Funding:
  - National Science Foundation (IIS-1718680)
  - Australian Research Council (DP170102231)
  - Australian Government (RTP Scholarship)

RMIT UNIVERSITY · THE UNIVERSITY OF MELBOURNE · NYU · PISA

Recursive Graph Bisection: Computing Gains

- Assume that identifiers are uniformly distributed in the arrangement.
- The cost is then related to the average gap between consecutive entries in t's adjacency list, which can be easily computed.
- For each document, compute and store the total cost of moving the document from D_1 to D_2 or vice versa.
- While we continue to yield positive gains, swap pairs of candidate documents between the two partitions.



Parameters

- Recursion depth = $\log(n) - 5$.
- Maximum iterations per recursion = 20.
- Default params are based on the paper we are reproducing.
- We investigate these parameters further in following experiments.

Complexity Analysis

- We recurse $\lceil \log n \rceil$ times,
- Each recursion involves computing move gains in $\mathcal{O}(m)$ time.
- Each recursion also involves sorting n elements in $\mathcal{O}(n \log n)$ time.
- Summing the subproblems together, we can see that the algorithm produces a vertex order in $\mathcal{O}(m \log n + n \log^2 n)$ time.
- Recall that $n = |D|$ and $m = |E|$.

Sensitivity to iterations and input order