On Optimally Partitioning a Text to Improve Its Compression

Davide Cologni[1], PhD Student

Venice, Italy – 19 January 2026

[1] RAVEN Group – Ca' Foscari University of Venice,
Github: **github.com/colobrodo**,
Email: **davide.cologni@unive.it**

Problem: Text Partitioning

We have a compressor $\mathscr{C}$ and a Text T of size n, is it possible to divide T into $k \leq n$ parts, $T[1..i_1 - 1]T[i_1..i_2 - 1]...T[i_{k-1}..n]$ and compress each of them individually with $\mathscr{C}$ to improve the overall compression?

Intuitively we can group the most similar parts of the string together so each partition is better compressed by $\mathscr{C}$.

Note: We do **not** *permute* the string.
We are only interested in *partitioning* it.

Text Partitioning Example

Suppose we have the text $T = \mathrm{a}^n\mathrm{b}^n$.

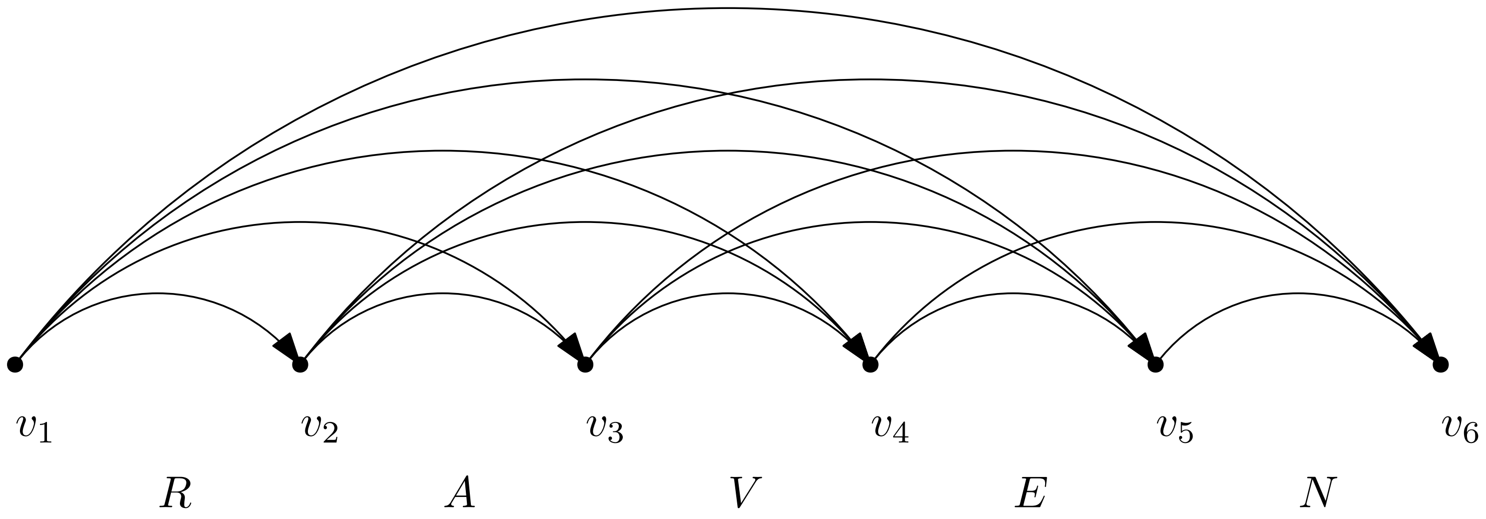
If we compress the entire text at once we should use one bit per symbol, or $\boldsymbol{O(n)}$ **bits**.

If instead we partition the text to compress a^n and b^n separately we can compress the whole string using only $\boldsymbol{O(\log_2(n))}$ **bits** indicating just the length of each substring.

Reduction to SSSP

We can model the partition problem as a directed graph with $n+1$ vertices, where an edge exists between v_i and v_j only if

$$1 \leq i < j \leq n+1$$



Reduction to SSSP - Bijection between paths and partitions

We can then show that there exists a bijection from each path $\pi = \left(v_1, v_{i_1}\right)...\left(v_{i_k}, v_{n+1}\right)$ in the graph, and the partitioning of the text T in the form $T[1..i_1 - 1]T[i_1..i_2 - 1]...T[i_{k-1}..n]$

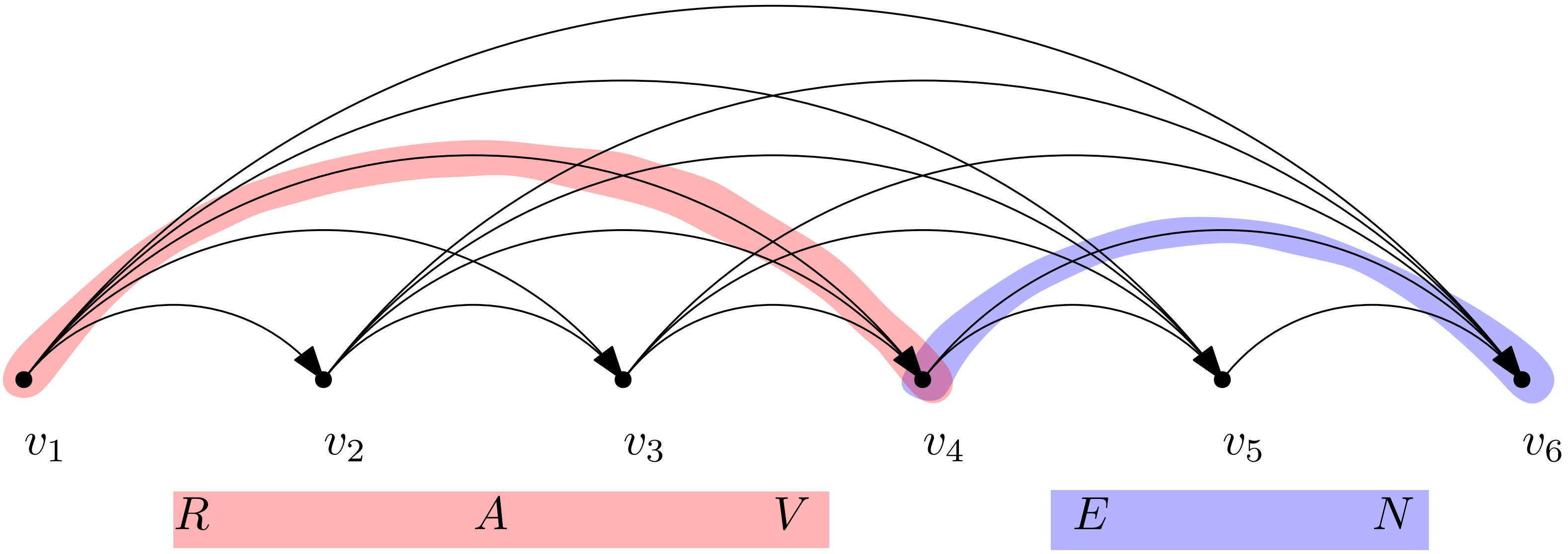


Figure 2: We can map the path $\pi = (v_1, v_4)\ (v_4, v_6)$ to the partitioning of the string $T[1,3]\ T[4,5]$

Reduction to SSSP - Bijection between paths and partitions

If we weight each edge (i, j) of the graph by the cost of compressing the corresponding text segment $w(i, j) = |\mathscr{C}(T[i, j - 1])|$, we can solve the partitioning problem *optimally* computing the **Single Source Shortest Path (SSSP)**

It can be computed efficiently in $O(|E|)$ time using a classic dynamic programming algorithm.

Problems:

1. Our graph has $O(n^2)$ nodes by construction
2. To initialize the weight $w(i, j)$ we should execute $\mathscr{C}$ on every substring of the text

Assumption on $\mathscr{C}$

- Our compressor is *monotonic*: the compressed output on a suffix or a prefix of the string is always smaller than the compression on the whole string:

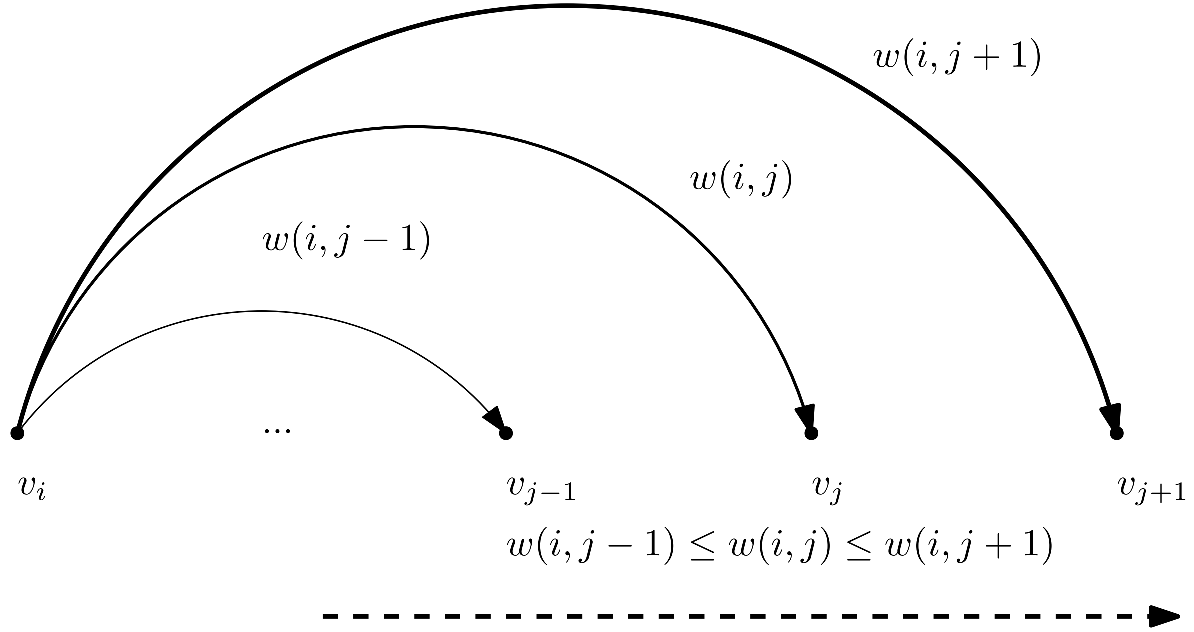
$$|\mathscr{C}(T[i,j])| \geq |\mathscr{C}(T[i,j-1])|$$

$$|\mathscr{C}(T[i,j])| \geq |\mathscr{C}(T[i+1,j])|$$

- We can compute the size of the compressed output incrementally: computing $|\mathscr{C}(T[i,j])|$ from the state of $\mathscr{C}(T[i-1,j])$ or $\mathscr{C}(T[i,j-1])$ takes constant time

Monotonicity of w

Due to the monotonicity of the compressor for every node $1 \leq i < k < j \leq n+1$ we have that $w(i,k) \leq w(i,j)$



Sparsification of the DAG

Thanks to this property we can obtain an approximated algorithm by **sparsifying** the graph thus selecting only some edges.

We are able to obtain a $(1+\varepsilon)$-approximation, for every $\varepsilon \geq 0$, with a time complexity of $O(n \log_{1+\varepsilon} L)$

where $L = w(1, n)$, so the cost of compressing the entire text.

This algorithm can be applied to every dynamic programming algorithm in the form $E[j] = \min_{1 \leq i < j}(E[i] + w(i, j))$ when w is *monotone*!

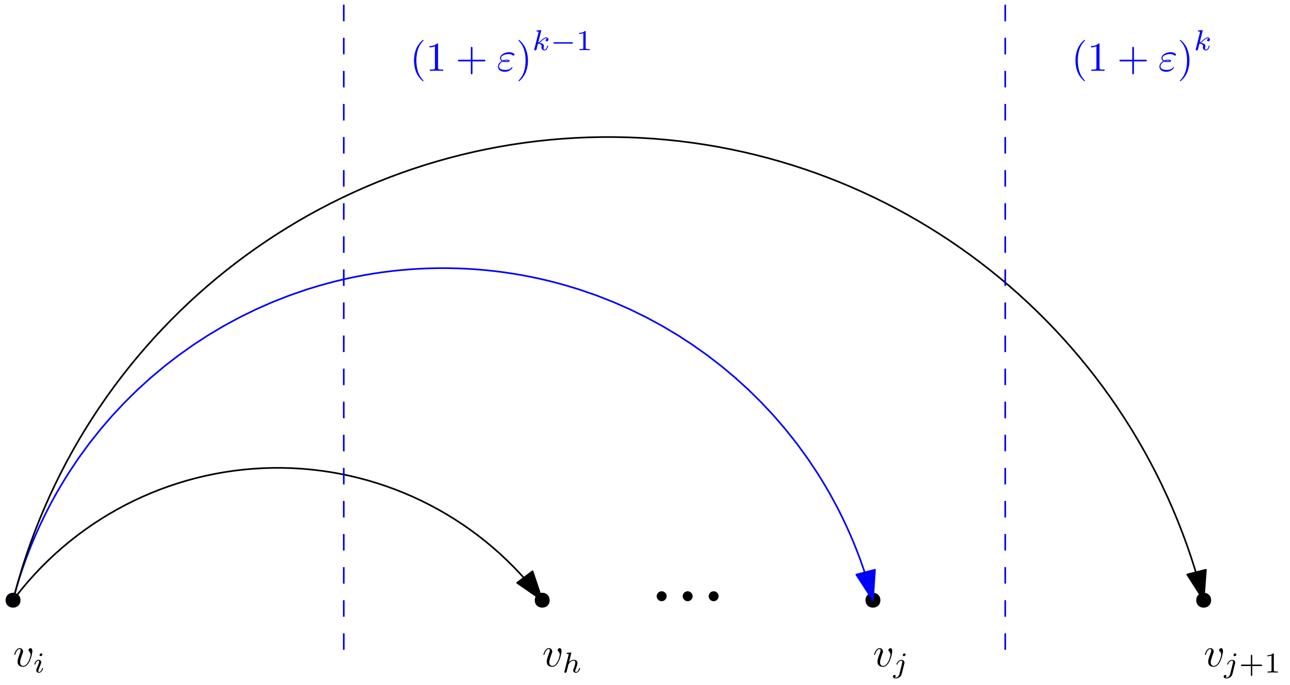
Key Idea: ε-maximal edges

How we can select some edges to obtain the $(1+\varepsilon)$ approximation factor?

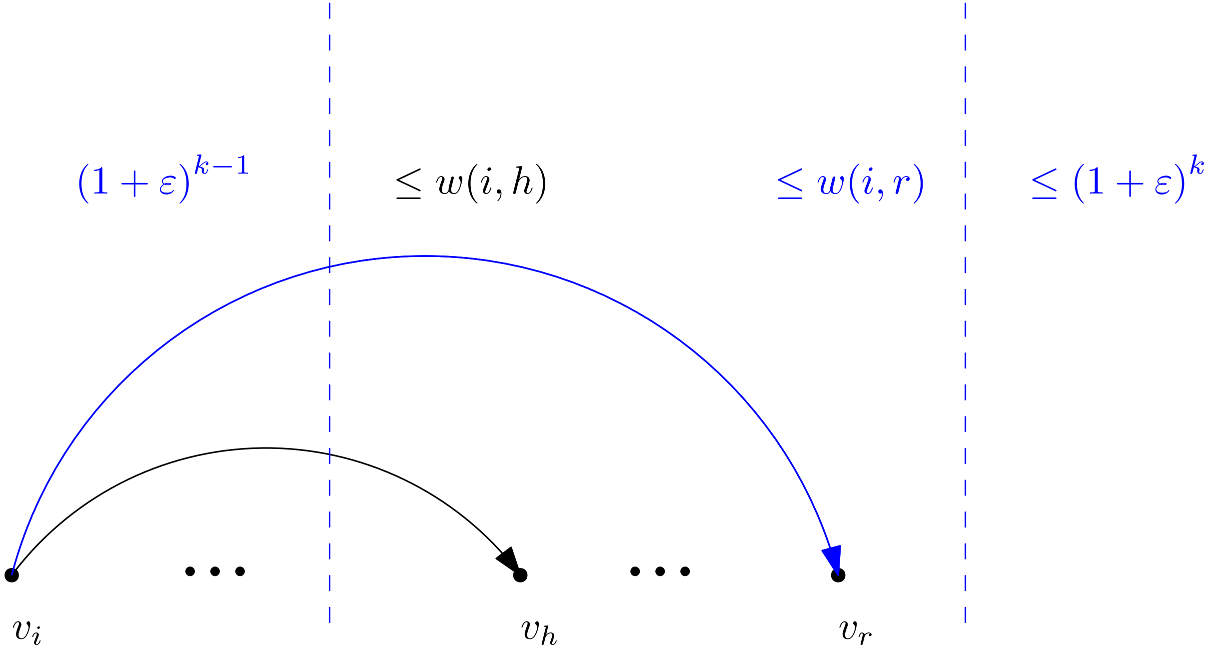
For each node i select the ε-maximal edges, so the outgoing edge from i that satisfy one of these conditions:

- The edges (i, j) such that $w(i, j) \leq (1+\varepsilon)^k < w(i, j+1)$ for any integer $k \geq 1$
- The last outgoing edge: $(i, n+1)$

So we select the best approximations of the powers of $(1+\varepsilon)$ from below: We then have at most $\log_{1+\varepsilon} L$ outgoing edges for each node.



Each edge is then *“covered”* by an ε-maximal edge: The weight of the edge is then approximated by $(1+\varepsilon)$ times the weight of the maximal edge that covers it.



$$\frac{w(i,r)}{w(i,h)} \leq \frac{(1+\varepsilon)^k}{(1+\varepsilon)^{k-1}}$$

$$\frac{w(i,r)}{w(i,h)} \leq (1+\varepsilon)$$

$$w(i,r) \leq (1+\varepsilon)w(i,h)$$

Lemma 1

Let $d_{\mathscr{G}}(i)$ be the cost of the shortest path π_i in our graph $\mathscr{G}$ from v_i to v_{n+1} then

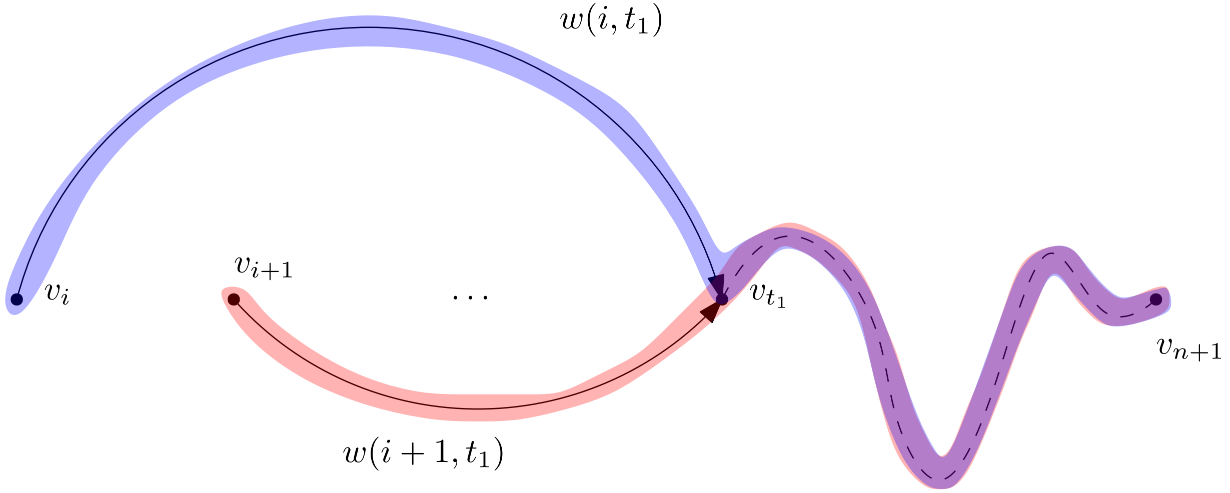
For all the vertices $i, j : 1 \leq i < j \leq n+1$, $d_{\mathscr{G}}(i) \geq d_{\mathscr{G}}(j)$

Proof by induction:

- Base, trivial case for $n+1$
- Then we need to show that $d_{\mathscr{G}}(i) \geq d_{\mathscr{G}}(i+1)$ by constructing a path π'_{i+1} that starts from $i+1$ and it is always shorter than $d_{\mathscr{G}}(i)$

Let $d_{\mathscr{G}}(i)$ be $\left(v_i, v_{t_1}\right)\left(v_{t_1}, v_{t_2}\right)\ldots\left(v_{t_k}, v_{n+1}\right)$

1. Trivial if $t_1 = i + 1$: $\pi'_{i+1} = \left(v_{t_1}, v_{t_2}\right)\ldots\left(v_{t_k}, v_{n+1}\right)$
2. If $t_1 > i + 1$ then we can construct a shorter path $\pi'_{i+1} = \left(v_{i+1}, v_{t_1}\right)\left(v_{t_1}, v_{t_2}\right)\ldots\left(v_{t_k}, v_{n+1}\right)$ because thanks to the definition of *monotonicity* we know that $w(i, t_1) \geq w(i + 1, t_1)$



Theorem

Let $\mathscr{G}_\varepsilon$ be the graph containing only ε-maximal edges, then $d_{\mathscr{G}_\varepsilon}(i) \leq (1+\varepsilon) d_{\mathscr{G}}(i)$ for every $1 \leq i \leq n+1$.

Proof by induction:

- **Base**, trivial case for $n+1$
- Then let $\pi(i) = \left(v_i, v_{t_1}\right)..\left(v_{t_h}, v_n\right)$ the shortest path starting from node v_i and let $d_{\mathscr{G}} = w(i, t_1) + d_{\mathscr{G}}(t_1)$ be its cost. We choose the ε-maximal node r that covers t_1: So $r > t_1$ and we already know that

$$w(i, r) \leq (1+\varepsilon) w(i, t_1)$$

By *Lemma 1*:

$$d_{\mathscr{G}}(r) \leq d_{\mathscr{G}}(t_1)$$

By inductive hypothesis:

$$d_{\mathscr{G}_\varepsilon}(r) \leq (1+\varepsilon)d_{\mathscr{G}}(r) \leq (1+\varepsilon)d_{\mathscr{G}}(t_1)$$

In the end

$$d_{\mathscr{G}_\varepsilon}(i) = w(i,r) + d_{\mathscr{G}_\varepsilon}(r) \leq (1+\varepsilon)(w(i,t_1) + d_{\mathscr{G}}(t_1)))$$

Problem: DAG Construction

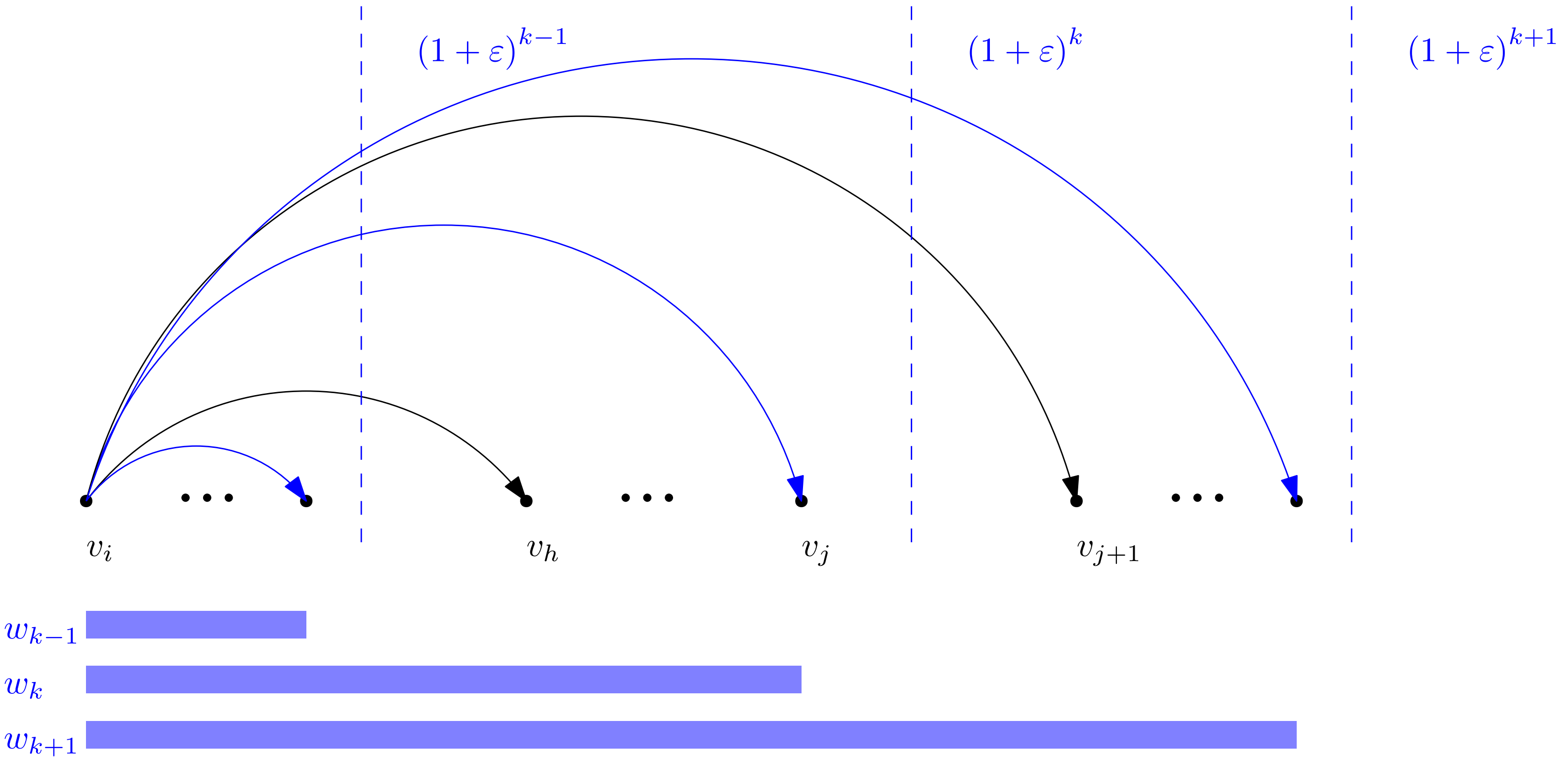
We still have two problems:

1. if we construct naively this graph we should remove edges from a $O(n^2)$ graph
2. We should compute the weight of the graph

We can solve both these problems efficiently at once!

Sliding windows

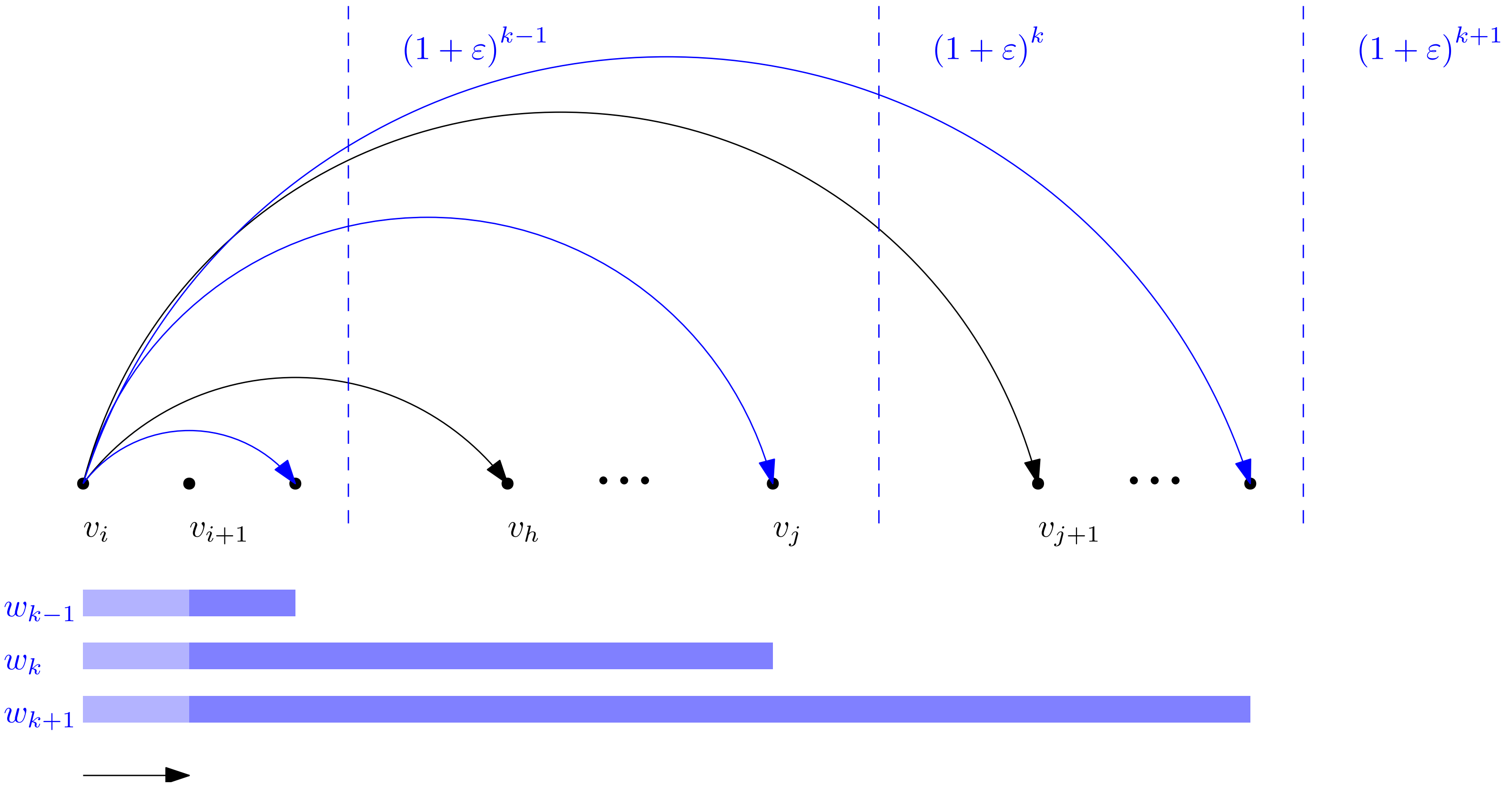
We keep $\log_{1+\varepsilon} L$ sliding windows all starting at v_i, but ending in a different position. The k-th window find the k-th ε-maximal edge.



$(1+\varepsilon)^{k-1}$
$(1+\varepsilon)^{k}$
$(1+\varepsilon)^{k+1}$
v_i
v_h
v_j
v_{j+1}
w_{k-1}
w_k
w_{k+1}

Sliding windows

For each compressor we should implement 2 operations on the windows `advance_left`, `advance_right`: The first operation advances the start of **all** the windows.

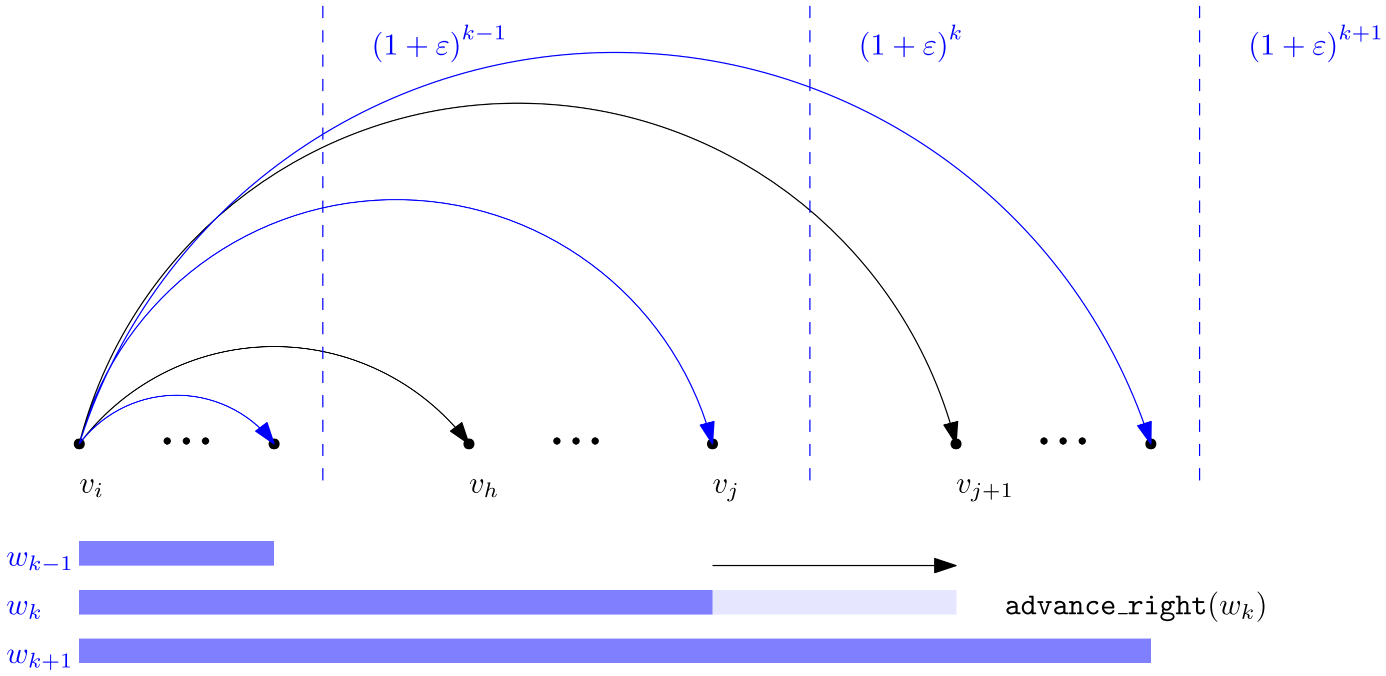


$(1+\varepsilon)^{k-1}$
$(1+\varepsilon)^{k}$
$(1+\varepsilon)^{k+1}$
v_i
v_{i+1}
v_h
v_j
v_{j+1}
w_{k-1}
w_k
w_{k+1}

Sliding windows

`advance_right` advance the end of the k-th window of one position.

We call this function until we reach the last edge smaller than $(1+\varepsilon)^k$, so until we find the k-th maximal edge starting from node i.



if the operations `advance_left` and `advance_right` have respectively a complexity of $O(L)$ and $O(R)$ our algorithm execute asymptotically $O(Ln + Rn \log_{1+\varepsilon} n)$ steps

The authors provide several implementations of the sliding windows framework to estimate the size of different compressors, among the others statistical compressors (using 0-th order and k-order entropy)

Computing Zero Order Entropy

Zero-th order entropy is a well-known lower bound for the performance of statistical compressors.

For each windows w_i, we maintain an histogram, $A_i[c]$, indexed by the symbol $c \in \sum$

$$E_i = \sum_{c \in \sum} A_i[c] \log_2 A_i[c]$$

Using E_i, we can calculate a lower bound on the output of the statistical compressor, $|\mathscr{C}(T[..i])|$ based on the zero-th order entropy as

$$|T[..i]|\ H_0(T[..i]) = |T[..i]| \log_2 |T[..i]| - E_i$$

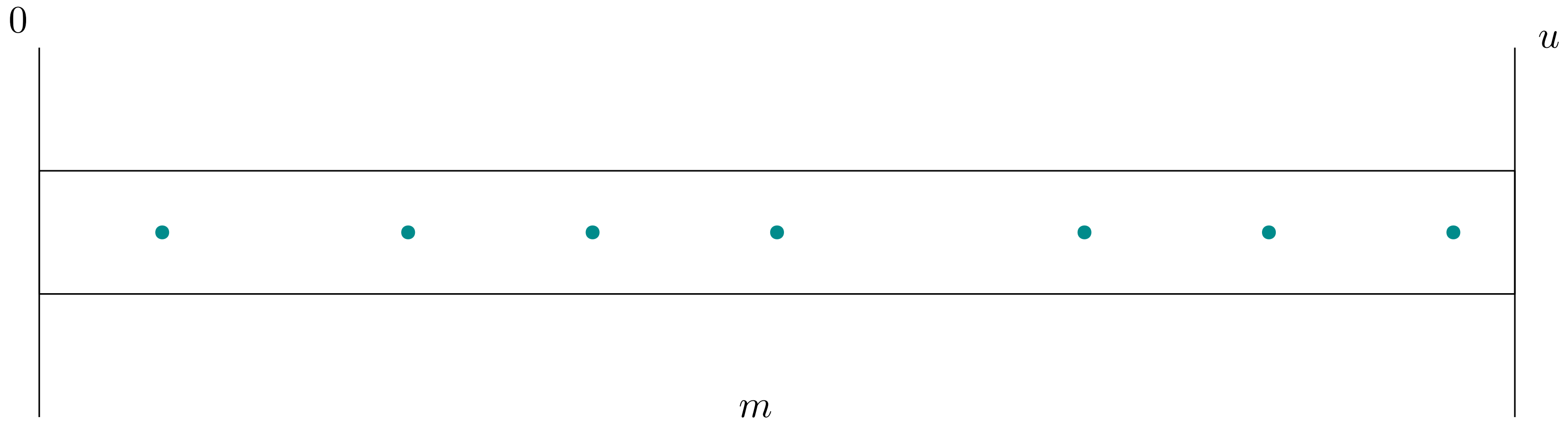
From this we can calculate incrementally the value of E_{i+1} removing the old term from the summation and adding the new one:

Let $c = T[i+1]$ then

$$E_{i+1} = E_i - A_i[c] \log_2 A_i[c] + (A_i[c] + 1)(\log_2 A_i[c] + 1)$$

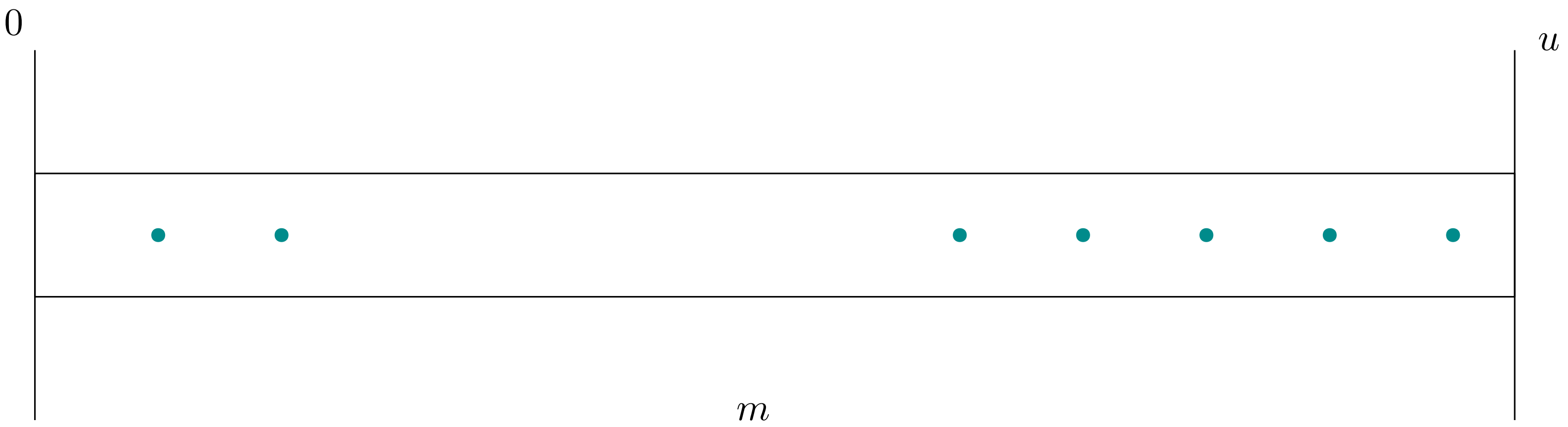
Application: Partitioned Elias-Fano

A compact data structure to store a set of m monotonically increasing integers upper-bounded by u, that uses $\approx \left\lceil \log_2 \frac{u}{m} \right\rceil + 2$ bits per element.



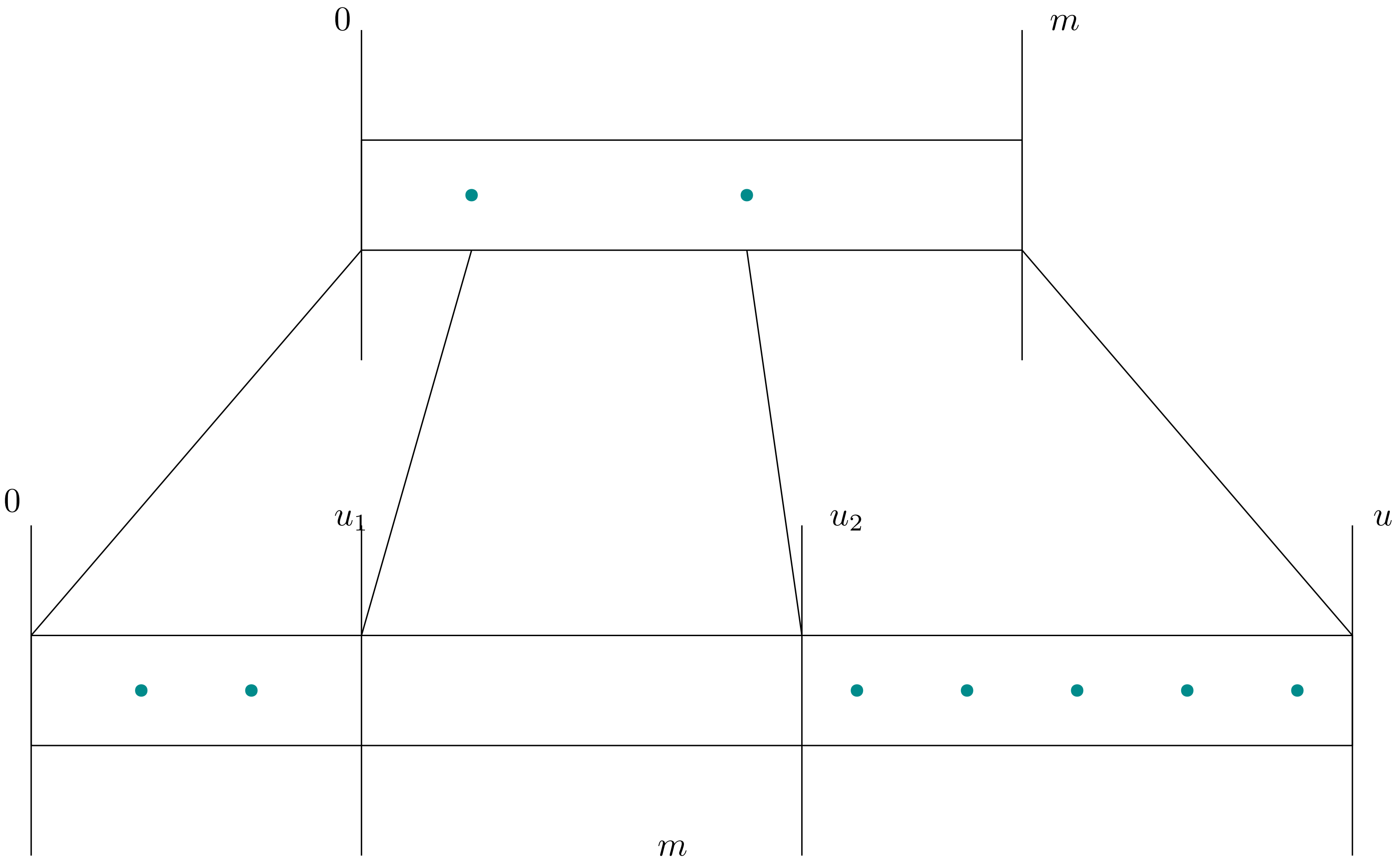
Note that $\frac{u}{m}$ is the average distance between consecutive elements. It doesn't exploit the distribution of the data, but denser lists require fewer bits.

Some sequence are more compressible than others



Partitioned Elias-Fano

We can improve compression by exploiting clusters of data with a two-level structure. The first level determines the bounds of the b clusters, and the second level contains smaller Elias-Fano lists.


0
m
0
u_1
u_2
u
m

How can we find the best partitioning to minimize the space occupancy of both levels?

We can use our partitioning algorithm, assigning a weight to each edge based on the number of bits required to represent the partition in the first level and the Elias-Fano structure in the second level.

The authors also improved the bound by showing that substituting an edge in the path with two sub-edges is always bounded by a constant factor. Thus, we can limit the search to a constant number of outgoing edges for each node.

Thank You!