# ANS graph compression report

Francesco Tomaselli

July 19, 2022

## Contents

# 1 Basic model

This is the basic approach to graph compression using asymmetric numeral systems. The idea is to have a model for each node that encodes its successors gaps. This is of course infeasible for larger graphs but gives an idea about the technique performance.

## 1.1 Computing symbols statistics

The first step to encode the outgoing links of a node is to compute statistics about the gaps. Let *succ* be the array containing the sorted successors of the $i$-th node, we compute the array *gaps*, with the following rule:

$$gaps[j] = \begin{cases} succ[j] & \text{if } j = 0 \\ succ[j] - succ[j-1], & \text{otherwise} \end{cases}$$

For each distinct value appearing in *gaps*, we compute its frequency. We then scale these counts to a given range $2^d$. The result is a quantity $f_s$ for each $s \in S$, thus the probability of $s$ can be written as $p_s = \frac{f_s}{2^d}$.

**Implementation details** In practice, we store an array *freq* containing the frequencies in descending order, and a map $m$ that maps a symbol to the corresponding index in the *freq* array, so $f_s = freq[m(s)]$. For decoding purposes, an inverse mapping is also computed, as the inverse of $m$. This two mappings are now implemented as two hashmaps, but they could be minimal order preserving hash functions to reduce runtime memory usage.

## 1.2 Ans based compression

After computing the frequencies of the gaps of a given node, we encode them with ANS compression.

We define an Ans Model, as the collection of the required data structures to encode a symbols source. This model contains the state, an integer storing the encoded symbols, the frequencies defined in the last Subsection, so an array storing normalized counts and the two mappings, as well as the cumulatives. The cumulative associated with a symbol is defined as $c_s = \sum_{t<s} f_t$, where $f_t > f_s \to t < s$.

The goal of the cumulative is, given $n \in [1, 2^d]$, to find $sym(n) = \max_{c_s \leq n} S$, this is implemented in practice with an Elias-Fano indexed monotone list.

**Encoding and decoding** The state, initially set to zero, is updated at each encoding and decoding, the primitives are defined as:

$$encode(x, s) = \lfloor state/f_s \rfloor * M + x_s + state \bmod f_s$$
$$decode(x) = \langle \lfloor (state - r)/M \rfloor * f_s - c_s + r, s \rangle$$

where $M = 2^d$, $r = 1 + (state - 1)\ mod\ M$, and the decoded $s = sym(r)$.

For the above definition, the encoded sequence is decoded in reverse order. Also, in practice, once the state overflows, it is normalized by storing the current state and resetting it to zero.

**Encoding successors** Given the gaps of the successors of a node, they are encoded in reverse order to avoid inverting the decoded sequence. Once the process is completed, the state sequence and the required structures to rebuild the model are written to disk.

**Model coding on disk** To rebuild the ans model for decoding purposes, some information is written on disk. We start with the size of the list of symbols, where each one written ordered by inverse mapping index. After that the sorted descending frequencies are written backwards by gap. All the integers are written in $\gamma$. $\delta$ code is also an option, but $\gamma$ works better on this use-case. To rebuild the symbols mapping, we just need to read the symbols, associating an incrementing index to each one of them. Frequencies must be filled in inverse order, and cumulative is obtained at run time.

For example, given the symbols: $1, 2, 3, 15$, and their relative frequencies $100, 30, 45, 10$ we write the size, the symbols ordered by descending frequency, and the sorted frequencies by gap:

$$3|1, 3, 2, 15|10, 20, 15, 55$$

**Performance** Data about a given node is written on two separate files, one is the .graph, that stores the model states, and the other is the .models file, that contains all the encoder structures. The performance in terms of occupied space is optimal for the first one, but tremendous for the second, as the required structures contains a lot of different symbols. The idea of the next Section is to tackle this problem.

The following is a table of the bits-per-link on different graphs:

| Graph | BV | EF | Basic ANS |
|---|---|---|---|
| eu-2005 | 3.726 | 17.504 | 13.505 |
| uk-2014 | 10.117 | 18.482 | 29.394 |
| eu-2015-tpd | 4.492 | 14.867 | 12.223 |
| eu-2015-tpd-t | 5.158 | 15.678 | 14.682 |
| en-wiki-2013 | 13.114 | 19.500 | 28.838 |
| en-wiki-2022 | 13.518 | 19.923 | 29.162 |
| en-wiki-2022-t | 13.518 | 19.923 | 29.162 |

# 2 Clustered model

The approach described in the previous Section becomes quickly unfeasible on larger graphs, for the simple reason that the models structures are too big to even get loaded in memory. The goal is now to reduce both the number of symbols and models.

## 2.1 Escaping gaps

One of the scenarios in which ANS compression works well, is when the overall probability of the alphabet is concentrated in a few symbols. On web graphs this is somehow true, as, for topological reasons, the gaps tends to be small with a few outliers.

We can build on top of this idea and escape some symbols to encode them in a different way. Given a list of successors gaps as follows:

$$10, 1, 1, 1, 2, 1, 2, 123, 256, 312, 3, 3, 2, 1$$

the frequency map looks like this:

| Symbol | Frequency |
|---|---|
| 1 | 5 |
| 2 | 3 |
| 3 | 2 |
| 10 | 1 |
| 123 | 1 |
| 256 | 1 |
| 312 | 1 |

We can see some symbols appearing only one time, perhaps we could avoid writing these ones to the .model file, and encode them with an instantaneous code or a different compression method. By escaping only the hapax and summing their frequencies, the map becomes:

| Symbol | Frequency |
|---|---|
| 1 | 5 |
| 2 | 3 |
| 3 | 2 |
| * | 4 |

If one would want to encode a list with this map, the technique would be to encode a $*$ sym to the state, and write the escaped symbol explicitly somewhere else. This worsen the encoder performance, but removes a lot of symbols from the ANS models, overall, the performance is better.

**Choosing what to escape** In the example above, the symbols appearing once were discarded, in practice, a percentage $p$ is selected. The symbols are sorted in descending ordered by frequency, in case of a tie, the smaller one come first. Then, the last $\frac{n}{100} * p$ symbols are escaped. The reason behind this sorting is that of course we want to escape rarer symbols, thus the descending frequency, but, if we had to choose, we prefer smaller symbols, as they take less space in the ans model maps. Another reason is that they are potentially more useful in the later clustering.

**PFOR delta escape encoding** The escaped symbols must be encoded in some explicit way on disk. The chosen approach is similar to PFOR delta but slightly modified. Given $v$

as the list of integers to encode, assume it for clearer notation to be sorted, we find $b_{max}$, the binary magnitude of the maximum, and we consider $b_i$, the required bits to store the i-th symbol.

Fixed a $b_i$, if a number can be written in that number of bits, it is written as is, else, the lower $b_i$ bits are written. All the required $b_{max} - b_i$ upper bits are written after the whole list. Each number has a flag bit before its binary code, to determine if upper bits are needed. For instance, given $b_i = 5$:

$$27 \rightarrow \mathit{flag} = 0,\ \mathit{bits} = 11011$$
$$327 \rightarrow \mathit{flag} = 1,\ \mathit{lower\ bits} = 00111,\ \mathit{upper\ bits} = 1010$$

Fixed a $b_i$, the space for the whole list can be easily computed, the minimum is selected.

**Performance** As hinted previously, this approach cuts the number of symbols to be written by a lot, reducing the overall space taken by the graph and model files. The bits required to store only successors increase, as some values might be escaped and written separately.

## 2.2 Clustering algorithm

After reducing the number of symbols per model, the goal is to avoid having a different model for each node. The idea is to use K-means to cluster the models, that can be seen as probability distributions. For now, we assume that no escaping is done for each model. Later the techniques are merged.

**Initialization** The algorithm starts with random centroids, indeed, given $k$ as the number of clusters, the i-th point is assigned to the $i \ mod \ k$ cluster. Each centroid aggregates the symbols from its data points and builds a frequency map. For instance, given two data points $D_1, D_2$, with the following frequency maps, the cluster $C$ will have as many points as the union of the two maps, with the sum of frequencies for each symbol:

| Symbol | $D_1$ | $D_2$ | $C$ |
|---|---|---|---|
| 1 | 5 | 10 | 15 |
| 2 | 3 | 2 | 15 |
| 3 | 2 | | 2 |
| 5 | | 1 | 1 |
| 13 | | 1 | 1 |
| 21 | | 1 | 1 |

**Clustering iteration** During an iteration, each point picks the centroid that minimizes a distance metric. This metric is the KL divergence between the point frequency map and the centroid one. At the end of the process, each centroid updates its frequency map like in the initialization.

**Escaping the clustered models** Once the clustering converges or the iterations are finished, each centroid builds the required structures to initialize the $k$ ans models. Is it safe to assume that, by merging a lot of frequency maps, each cluster have a lot of symbols, thus, when a single node encodes his successors, it might use a model that is way to big in terms of size of frequency map. This worsen the encoding performance, as the probability distribution used for encoding is not optimal.

To reduce the impact of encoding with a sub-optimal distribution, we apply escaping on the model before initializing the ans structures, by minimizing a function. All the frequencies are now considered at the cluster level.

The formula is made of four components, the first one is the entropy of the ANS encoding:

$$E_{ans} = -\sum_{s \in S} c_s \log p_s$$

where $S$ is the alphabet, $c_s$ is the count of a symbol, $p_s$ is defined as the frequency of the symbol over the total count of frequencies.

The second is the entropy of the escapes, let $S'$ be the set of escaped symbols, $p_{esc} = \sum_{s \in S'} p_s$ is the sum of the probabilities, $c_{esc} = \sum_{s \in S'} c_s$ the sum of counts:

$$E_{esc} = -c_{esc} \log p_{esc}$$

The third element is the cost of the PFOR delta encoding for writing escapes on disk, that can be approximated with:

$$PFOR = \alpha(c_{esc} * b_{max})$$

where $b_{max}$ is the binary magnitude of the maximum escape, $\alpha$ is the percentage of space gained by using PFOR instead of $b_{max}$ bits for each element, it can be measured empirically, but a good value can be 0.8.

Lastly, the final term of the formula is the space required for the ans models, that is all the symbols written in Gamma code, plus the sorted frequencies by gap, this term is referred to as *model bits*.

The formula to minimize becomes:

$$S = E_{ans} + E_{esc} + PFOR + \textit{model bits}$$

An important note is that by minimizing this formula, we are considering the symbols statistics at the cluster level. To make a more accurate estimation, one would need to fix a subset $S'$ of escaped symbols, iterate over all the points in the cluster and compute the space required by each node. This would cost too much in terms of efficiency.

**Choosing the escape symbols set** The escape symbols set $S'$ is the core of the space formula to minimize. The approach does not considers all possible subsets of $S$, indeed, given the symbols sorted by descending frequency, we fix a cut point and take all the elements after that point as the escape set. For instance, considering the symbols ordered by frequency:

$$syms = 1, 2, 5, 3, 13, 110, 45$$

The cut point starts at the end of the symbols, so $S' = \emptyset$, then with each iteration, the last symbol of the list is added, so the set progressively becomes $\{45\}$, $\{45, 110\}$, $\{45, 110, 13\}$ and so on. This permits to have a runtime of $O(|S|)$, as all the components can be updated in constant time from one iteration to the other.

**Escaping before clustering** The idea of escaping a single node model introduced in the previous subsection can be used before clustering. Indeed, the escape wildcard is treated like any symbol in the alphabet during the algorithm. When choosing the $S'$ set, a prior escape translates to starting with a non-empty set. This reduces the execution time of the algorithm by quite a lot. About the sorting criteria when choosing what to escape in the single nodes, preferring smaller symbols is more useful for the clustering procedure, as they are more likely to appear in a wider range of nodes. Lastly, escaping before clustering does not lead to the same results as clustering alone, but this technique is empirically better.

**Performance** Those are the bits-per-link on the same graphs presented in the first Section, the clustering is still in development as of now, so the results might be better in the future. Also, the results were obtained without any iteration after the random initialization.

| Graph | BV | EF | Basic ANS | Clustered ANS |
|---|---|---|---|---|
| eu-2005 | 3.726 | 17.504 | 13.505 | 8.275 |
| uk-2014 | 10.117 | 18.482 | 29.394 | 17.855 |
| eu-2015-tpd | 4.492 | 14.867 | 12.223 | 7.723 |
| eu-2015-tpd-t | 5.158 | 15.678 | 14.682 | 8.877 |
| en-wiki-2013 | 13.114 | 19.500 | 28.838 | 16.771 |
| en-wiki-2022 | 13.518 | 19.923 | 29.162 | 16.779 |
| en-wiki-2022-t | 13.518 | 19.923 | 29.162 | 14.891 |

On average, clustering saves 40 percent of space compared to basic ANS.