Università degli Studi di Milano
Computer Science department

Large-scale graph compression via asymmetric numeral systems

Francesco Tomaselli

Supervisor:

Sebastiano Vigna

Co-Supervisor:

Paolo Boldi

Academic year 2021/2022

Abstract

Graphs are one of the most versatile data structures in computer science and mathematics, finding an extreme variety of applications from path-finding, to ranking in search engines, community detection, and so on. One of the key requirements of graph analysis is the possibility to efficiently visit them, which is an easy task when they are small, but not so much when billions of nodes and hundreds of billions of arcs are involved. A web graph is a structure where nodes are made out of web pages, and there exist a link between nodes x and y if x contains a hyperlink to page y. On the other hand, social graphs are structures where nodes are connected via a social relation, for instance Wikipedia pages, friends and followers on a social network, and so on. These types of graphs certainly store a large amount of information, thus we need a way to visit them efficiently.

Why the classical representation do not work The classical ways of storing graphs are through adjacency matrixes or lists: in the first case a $n \times n$ square matrix is considered and a boolean flag on cell i, j indicates whether there exists an arc between node i and j, adjacency lists represent a graph through a list of successors for each node. The classic representations are perfectly fine for small graphs, but occupy a huge amount of memory for large-scale ones. In fact, one of the main problems associated with web and social graphs is their dimensions, we are talking about structures with possibly billions of nodes and hundreds of billions of arcs, thus a classical representation can not be loaded in memory. A possible workaround to visit a large-scale graph is to load smaller parts of the structure from disk when needed, but this is slow and unpractical.

Data compression One solution to visit large-scale graphs is data compression. The goal is to store structures in fewer bits than a classical representation, and this often translates to finding ways to compress integers. Data compression makes it possible to load larger structures in memory but also benefits the scenario where we are forced to partially load them from disk, as each partial load considers a greater percentage of the total structure.

There exist many techniques for data compression, and in the end, they all involve integer compression. Some classical approaches are found in the application of instantaneous codes to encode a given integer source. Those have been widely used to store all sorts of data and are straightforward to apply. The idea is to encode an integer n in a codeword, where the length of it determines its probability, hence one needs to assign shorter codewords to frequent integers.

Outside of the realm of instantaneous codes, we find techniques to efficiently code blocks of integers, such as patched frame of reference, and structures that use a number of bits close to the succinct bound such as Elias-Fano lists.

Other techniques do not rely on an intrinsic distribution of codewords and are tailored to specific sources, such as asymmetric numeral systems, that are a family of entropy encoders. They use optimal space for a given source, paying the price of the need to store additional information. This type of encoders are the main focus of this thesis, indeed we want to find out if they can be fruitfully applied to graph compression.

Graph compression has been already studied in the past, a good example is the Webgraph framework, which exploits the structural properties of web and social graphs to store them in a few bits per link. The approach relies on the empirical similarity of successors of close nodes, using reference compression to represent a node's outgoing links: the idea is to fix a reference node and represent its successors with a bitmask. The residuals, which are nodes not included in the reference, are written exploiting consecutivity, which creates several consecutive intervals in a node successors list.

Another starting point for this study is the compression of inverted indexes. An inverted index is a data structure that builds the backbone of search engines, the

idea is to assign to each word the document ids, plus other additional information, where it appears. As the reader can imagine this structure is similar to an adjacency list, as we are storing a list of integers for each term. Many techniques can be applied to store this type of index, but one work, in particular, exploits asymmetric numeral systems for the task. The authors suggest some key disadvantages to keep into consideration when applying ANS to large-scale datasets.

The proposed methodology The proposed methodology is a compression scheme for graphs, where asymmetric numeral systems are applied to the gap encoding of the nodes' successors. We start with a naïve application to then refine it. One of the characteristics of entropy encoders is that they can be tailored to a statistical source, hence they offer optimal compression ratios, the tradeoff is that we need to store additional information to rebuild these ad hoc compressors. Indeed, ANS in particular uses symbols maps to store observed probabilities, and they can become quite large if a source exhibits several different symbols.

The naïve approach fails exactly for this reason, as we are building a specialized model for each node in a graph. The technique generates optimal compression while wasting an enormous amount of space writing all the additional information required by each encoder on disk. To solve this problem, various approaches are employed, but the idea is to first limit the size of the symbols maps of the encoders. One way to do so is through symbols escaping, which consists in removing some symbols from the maps and encoding them with another compression method. The other point left to solve is the excessive amount of specialized models, as we are dealing with a different encoder for each node. The idea that comes to mind is clustering, which consists in finding some similarity relations between models, reducing a group of similar ones to a single encoder. For this purpose, a Gray code ordering is employed on the models' symbols, to then partition them heuristically. Finally, the idea of escaping is applied to the clustered models, where we find a suitable set of symbols to remove from the frequency maps by minimizing a space estimation function.

The final result is a methodology that combines asymmetric numeral systems,

patched frame of reference, and instantaneous codes to store large-scale graphs. The results are promising and the compression schema is capable of storing web graphs in as low as 3.5 bits per link, saving 76 percent of the space required by quasi-succinct data structures. The access speed to the graph is also comparable to the existing methods, keeping in mind that entropy encoders are on the slower side of compression methods.

Experimental results and conclusions All the implementation is written in Java and publicly available. The project implements Webgraph interfaces making it easier to store and load graphs from the public datasets. Compression results are expressed in bits per link, and are always in between of the Webgraph methodology and the quasi-succinct representation. Speed is tested by randomly accessing nodes successors, and the obtained speed per link is comparable to the Webgraph compression scheme.