Graph encoding using an automorphism

Jaka Velkaverh, Primož Potočnik

August 4, 2025

Abstract

This document describes the process and usage of an encoding algorithm, which encodes a graph together with one of its automorphisms more efficiently than is possible without using the graph's symmetries. The process encodes the graph as a sequence of printable ASCII characters starting with two colons `::`.

1 Theoretical basis of the encoding

Let $\Gamma = (V, E)$ be a simple undirected graph and σ one of its automorphisms. Let k be the number of cycles in σ's cyclic decomposition (i.e. the number of orbits σ has in its action on V) and $m_1, \ldots, m_k$ be the lengths of those cycles.

We construct a quotient graph Γ/σ as an undirected graph with loops and parallel edges with labelled vertices and edges. Its set of vertices is $V' = \{u_1, \ldots, u_k\}$ with labels being $\{m_1, \ldots, m_k\}$, and edges being constructed as follows:

Label the cycles of σ as $C^i = \left(c_0^i, c_1^i, \ldots, c_{m_i-1}^i\right)$. We interpret the subscripts as being elements of $\mathbb{Z}_{m_i}$, whenever we write c_{a+b}^i we mean $c_{a+b \bmod m_i}^i$. Then for $i, j \in \{1, \ldots, k\}$ let

$$D_{i,j} \coloneqq \left\{\delta \in \mathbb{Z} \mid c_k^i \sim_\Gamma c_{k+\delta}^j \text{ for some } k\right\}.$$

Because σ is an automorphism of Γ if $\delta \in D_{i,j}$ then $c_{k+1}^i \sim_\Gamma c_{k+1+\delta}^j$. Since c_k^i is in an orbit of length m_i it is equal to $c_{k+m_i}^i$. Therefore $c_k^i \sim_\Gamma c_{k+m_i+\delta}^j$. We can therefore conclude $\delta \in D_{i,j} \Rightarrow \delta + m_i \in D_{i,j}$. Because $c_k^j = c_{k+m_j}^j$ we can also say $\delta \in D_{i,j} \Rightarrow \delta + m_j \in D_{i,j}$. To reconstruct the entire set $D_{i,j}$ all we need to encode is $D_{i,j}/\left\langle m_i, m_j\right\rangle = D_{i,j} \bmod \gcd\left(m_i, m_j\right)$. We also notice that if $c_k^i \sim_\Gamma c_{k+\delta}^j$ we can apply σ^{-k} and get $c_0^i \sim_\Gamma c_\delta^j$, so we could say

$$D'_{i,j} \coloneqq D_{i,j} \bmod \gcd\left(m_i, m_j\right) = \left\{\delta \bmod \gcd\left(m_i, m_j\right) \mid c_0^i \sim_\Gamma c_\delta^j\right\}.$$

For each element $\delta \in D'_{i,j}$ we add an edge between u_i and u_j and label it with δ. Using the vertex labels of Γ/σ we can reconstruct σ's orbits and using the labels δ we can reconstruct $D_{i,j}$ and from it all the edges. The encoding is based on saving the graph Γ/σ.

2 Description of the encoding process

(a) Arbitrary data as printable ASCII characters

A sequence of bits of known length divisible by 6 can be encoded in printable ASCII characters by dividing the sequence into chunks of 6 bits, interpreting them as numbers in binary, adding 63, and selecting the character with that index from the ASCII table.

We can also encode a number n between 0 and 2^{36-1} with no prior information about its size like in the graph6 and sparse6 encodings, described in https://users.cecs.anu.edu.au/ bdm/data/formats.txt, which we denote as $N(n)$.

(b) The process

The string representing the given graph $\Gamma = (V, E)$ with an automorphism σ is split into 3 parts, the first encoding the number of vertices of Γ, the second encoding the cyclic structure of σ, and the third encoding the edges and edge labels of Γ/σ.

The number of vertices is encoded as $N(n)$ $(n := |V|)$.

For the cyclic structure of σ et b_n be the number of bits required to represent n in binary (so $\lfloor \log_2 n \rfloor + 1$). Then represent σ as a sequence of numbers $(f_1, x_1, \ldots, f_s, x_s, 0, y_1, \ldots, y_{s'}, 0)$, where each pair f_i, x_i means σ has f_i cycles of length x_i and each number y_i means σ has one cycle of length y_i, while the zeros serve as a delimiter. Encode each of the numbers in this sequence using exactly b_n bits and concatenate them, pad them with zero bits on the right to make its length divisible by 6 and then encode it as ASCII as described earlier.

The edges of Γ/σ are encoded as a sequence of bits using ASCII characters. Define b_k as the number of bits needed to represent k (= the number of cycles of σ). For each $i, j \in \{1, \ldots, k\}$ define $b_{i,j}$ as the number of bits needed to represent $\gcd(m_i, m_j)$. The sequence of bits is obtained using the following algorithm:

Set $v = 1$.
For each i which $1 \leq i \leq k$:
  For each j which $1 \leq j \leq i$:
    Compute $D'_{i,j}$.
    If $D'_{i,j} \neq \emptyset$ and $v < i$:
      Append the bit 0,
      append i in binary as b_k bits,
      set $v = i$.
    If $D'_{i,j} \neq \emptyset$:
      Append the bit 0,
      append j in binary as b_k bits.
    For each $\delta \in D'_{i,j}$:
      Append the bit 1,
      append δ in binary as $b_{i,j}$ bits.

This can be decoded by keeping track of the variable v. Any time the bit 0 is followed by a number smaller or equal to v we are adding an edge, any time the bit 0 is followed by number greater than v, we update v.

3 Using the C++ code

The code is composed of the classes `Permutation` and `Graph`.

An instance of `Permutation` is constructed from an `vector<int>` of length n, where n is the size of the permutation. The vector must contain the tabular form of the permutation using integers in the range $[1, n]$.

An instance of `Graph` is constructed from an `vector<vector<int>>` representing its neighbours list. Note that the indexing is 1 based (the outer vector is padded) so `neighbors[1]` are the neighbours of the first vertex. The encoding of a graph can then be obtained by calling the member function `encode(Permutation automorphism, bool sparse)`, with an automorphism of the graph in the first parameter. Currently only the sparse encoding is implemented, so the second parameter must be `true`. The encoding can be decoded by using the function `decode(string encoded)`, which returns a graph object. There is also a member function `to_sparsegraph`, which convers to the `sparsegraph` type used by Nauty.